ATARI ST+STE

BIEN DEBUTER STOS

TOUTES LES INFORMATIONS POUR CONSTRUIRE VOTRE JEU D'ARCADE DE A À Z

EDITIONS MICRO APPLICATION

LIVRE MA

Michel MARTIN

Bien Débuter STOS

EDITIONS MICRO APPLICATION

Auteur Michel Martin

ISBN : 2-86899-321-4

Collection dirigée par Philippe Olivier
Edition réalisée par Frédérique Beaudonnet

Introduction

Le STOS est un environnement de développement BASIC essentiellement axé sur la réalisation de jeux de toutes sortes (d'arcades, de réflexion, pédagogiques, etc.).

Le livre que vous avez entre les mains est plus particulièrement destiné aux débutants qui désirent apprendre puis exploiter les principaux domaines d'action du STOS : sprites, musique, banques mémoire. Après sa lecture, vous serez en mesure de créer vous-même vos jeux à l'aide du BASIC STOS.

Le livre se décompose en trois parties.

La première partie est une brève introduction au STOS. Vous y verrez comment :

- ✓ dupliquer et décompacter les disquettes du constructeur,
- ✓ lancer et utiliser l'éditeur du STOS,
- ✓ maîtriser les principaux utilitaires rattachés au STOS : éditeur de sprites, éditeur de caractères, éditeur d'icônes, éditeur de musique, compacteur d'écran.

La seconde partie est la plus importante du livre. Elle vous permettra de manipuler les mots-clés du langage STOS. Pour rendre l'apprentissage plus productif, ces derniers sont étudiés par thèmes.

Dans un premier temps, vous apprendrez à utiliser les instructions de base :

- ✓ instructions d'ordre général,
- ✓ instructions dédiées aux périphériques de sortie (écran, imprimante, lecteur de disque),
- ✓ instructions dédiées aux périphériques d'entrée (clavier, souris, joystick).

Vocabulaire et conventions d'écriture :

Tout au long du livre, nous allons utiliser quelques mots qui ne font pas forcément partie de votre vocabulaire. En voici la signification exacte.

- ✓ CLIQUER signifie appuyer sur le bouton gauche de la souris,
- ✓ Dans une fenêtre, un BOUTON est une des options proposées. Appuyer sur un bouton signifie positionner le curseur de la souris sur ce bouton et cliquer.
- ✓ BOOTER l'ordinateur signifie appuyer sur le bouton RESET situé sur la partie arrière du clavier (pour les 520 et 1040 ST), et sur la face arrière du boîtier (pour les Méga ST 2 et 4).
- ✓ Un ICONE est une représentation graphique qui peut être sélectionnée avec la souris pour effectuer une action bien précise.

Maintenant que vous êtes initié au vocabulaire du livre, nous allons passer aux conventions d'écriture concernant les mots-clés du BASIC STOS.

Les paramètres obligatoires d'un mot-clé sont représentés par des noms délimités par les signes < et >. Par exemple, l'instruction

```
key speed <v>,<d>
```

demande deux paramètres obligatoires : <v> et <d>.

Attention, les paramètres obligatoires d'un mot-clé ne doivent pas être confondus avec les touches du clavier qui sont systématiquement encadrées des mêmes délimiteurs. Par exemple, <CR> représente la touche RETURN et <Insert> la touche INSERT du clavier.

Les paramètres facultatifs d'un mot-clé sont représentés par des noms délimités par les signes [et]. Par exemple, l'instruction

```
search [<chaîne>]
```

admet un paramètre facultatif de nom <chaîne>.

Dans un second temps, vous étudierez les instructions orientées plus particulièrement vers la réalisation de jeux :

- ✓ création et mise en oeuvre de sprites,
- ✓ réalisation de musiques et d'effets sonores spéciaux destinés à accompagner les jeux,
- ✓ utilisation des banques mémoire.

Arrivé à la troisième partie, vous connaissez bien le STOS, son langage et ses accessoires. Vous mettrez en pratique vos connaissances pour créer un jeu d'arcades complet incluant plusieurs sprites en mouvement, un fond sonore et une manipulation au joystick.

Le livre se termine par quatre annexes qui vous seront utiles une fois que vous maîtriserez bien le STOS, c'est-à-dire après la lecture et l'assimilation de la seconde partie du livre.

La première annexe présente les diverses fontes système. Elle vous permet de choisir en connaissance de cause le type de caractères le plus approprié à la réalisation d'un programme donné. Elle peut également être le point de départ de la création d'un nouveau jeu de caractères.

La seconde annexe vous permet de retrouver rapidement un des mots-clés du langage, en fonction de son domaine d'application.

La troisième annexe liste et donne la syntaxe et la fonction de tous les mots-clés du STOS. Elle est particulièrement utile lors du développement d'un programme.

Enfin, la quatrième annexe liste les diverses erreurs système que vous pourriez rencontrer, et en donne la cause.

Lorsqu'un mot-clé demande un paramètre obligatoire qui doit appartenir à un ensemble, celui-ci est délimité par les signes { et }, et les valeurs sont séparées entre elles par le signe |. Par exemple, l'instruction

```
mode {0|1|2}
```

demande un paramètre obligatoire de valeur 0, 1 ou 2.

Sommaire

Partie I : Indroduction au STOS 11

1. Pour s'échauffer 13

1.1. Avant de commencer : copie et décompactage des disquettes 13
1.2. Mise en route 15
1.3. L'éditeur du STOS 15
1.4. Les divers aspects du STOS 17

2. Les accessoires 19

2.1. L'éditeur de sprites 19
2.2. L'éditeur de caractères 47
2.3. L'éditeur d'icônes 49
2.4. L'éditeur de musique 49
2.5. Le compacteur d'écran 70

Partie II : Le savoir de base en basic STOS 73

3. Instructions de base du STOS 75

3.1. Instructions d'ordre général 75
3.1.1. Les instructions généralement utilisées en mode direct ... 75
3.1.2. Instructions utilisées en mode programmé 88
3.2. Instructions à caractère mathématique 106
3.3. Instructions dédiées aux chaînes 115

4. Organes de sortie 123

4.1. L'écran 123
4.1.1. Instructions texte 124
4.1.2. Instructions graphiques 130
4.1.3. Manipulation de l'écran 138

4.1.4. Fenêtres et jeux de caractères 145
4.1.5. Menus 148
4.2. Imprimante 151
4.3. Lecteurs de disques 151

5. Organes d'entrée 161
5.1. Le clavier 161
5.2. La souris 163
5.3. Le joystick 164

6. Les sprites 165
6.1. Affichage d'un sprite 166
6.2. Déplacement d'un sprite 168
6.3. Animation d'un sprite 171
6.4. Tests de collision 173

7. Musique et effets sonores 179
7.1. Intégration de commandes sonores dans un programme 180
7.2. Utilisation de l'éditeur de musique 183

8. Les instructions particulières du STOS 189
8.1. Les accessoires 189
8.2. Les banques mémoire 190

Partie III : Création d'un programme de jeux de A à Z 193

9. Première approche. Généralités 195
9.1. Première étape : Réflexion subliminale. 196
9.2. Seconde étape : Création du décor et des personnages .. 196
9.3. Troisième étape :
Création de la musique de fond et des effets sonores 198
9.4. Quatrième étape : La programmation 198

10. Décor et personnages 201
10.1. Utilisation de l'éditeur de sprites 201
10.2. Intégration des sprites dans le programme 202
10.3. Création du décor 203

11. Musique 205
11.1. Utilisation de l'éditeur de musique 205
11.2. Intégration du morceau dans le programme de jeu 213

12. Le programme de jeu 215

A. Les polices de caractères 227

B. Les mots-clés du basic STOS classés par genres 231

C. Vue d'ensemble sur le BASIC STOS 235

D. Les erreurs système 263

Index 271

Partie I

Introduction au STOS

Chapitre 1

Pour s'échauffer

1.1. Avant de commencer : copie et décompactage des disquettes

Je sais que vous mourrez d'envie d'essayer votre nouvelle acquisition, le STOS. Mais, je vous conseille d'effectuer au préalable une duplication des trois disquettes du constructeur. Ces disquettes sont de type simple face. Elles peuvent être utilisées sur des lecteurs simple ou double face.

Avant d'effectuer la duplication des disquettes, il convient de formater 3 disquettes vierges si vos lecteurs sont double face (2 en simple face et une en double face), ou 4 disquettes vierges s'ils sont simple face.

Pour formater une disquette, procédez comme suit :

❶ mettez votre ordinateur sous tension, comme vous le faites habituellement,

❷ introduisez la disquette à formater dans le lecteur A:,

❸ cliquez sur l'icône qui représente le disque A,

❹ déplacez la souris sur le menu "Fichier" (en haut de l'écran), et cliquez sur l'option "Formatage..." dans ce menu,

❺ un message vous informant que les données de la disquette A: vont être effacées est affiché dans une fenêtre sur l'écran. Appuyez sur le bouton CONFIRMER pour valider le formatage.

❻ une nouvelle fenêtre est affichée. Choisissez le type de formatage désiré : simple face ou double face. Appuyez enfin sur le bouton FORMATER.

Quelques secondes après cette dernière opération, l'ordinateur affiche la capacité de la disquette formatée : 357376 octets pour une disquette simple face, 726016 octets pour une disquette double face. Appuyez sur le bouton OK. Recommencez ces opérations pour formater les 3 ou 4 disquettes nécessaires à la duplication. Appuyez sur le bouton SORTIR lorsque la dernière disquette est formatée.

Les disquettes sont maintenant formatées. Nous pouvons procéder à la duplication. Si vous possédez deux lecteurs de disquettes, introduisez la disquette "Langage" dans le lecteur A: et une des disquettes formatées en simple face dans le lecteur B:. Positionnez le curseur de la souris sur l'icône du lecteur A:. Appuyez sur le bouton gauche de la souris. Maintenez-le appuyé et déplacez la souris pour amener l'icône du lecteur A: sur l'icône du lecteur B:. Lorsque cela est fait, lâchez le bouton gauche de la souris. Un message apparaît sur l'écran. Il vous indique que les données de la disquette B: vont être effacées par la copie. Appuyez sur les boutons CONFIRMER, puis COPIER pour effectuer la duplication.

Recommencez ce qui vient d'être dit pour dupliquer la disquette "Jeux" sur une autre disquette vierge simple face.

Si vous ne possédez qu'un lecteur de disquettes, les manipulations sont les mêmes. L'ordinateur vous demandera de changer les disquette source et cible autant de fois que nécessaire.

La disquette "Accessoires" ne doit pas être dupliquée par la méthode qui vient d'être décrite, car les données qu'elle contient sont compactées. Pour dupliquer cette disquette, il vous suffit de l'insérer dans le lecteur A: et de booter l'ordinateur. La démarche à suivre est clairement indiquée sur l'écran.

1.2. Mise en route

Pour lancer l'éditeur du STOS, insérez la disquette "Langage" dans le lecteur A: et bootez l'ordinateur. Quelques secondes après, vous vous trouvez dans l'éditeur du BASIC STOS.

1.3. L'éditeur du STOS

Le STOS possède un éditeur de lignes : les programmes écrits en BASIC STOS seront constitués de lignes numérotées, comme par exemple :

```
10 print "Bonjour"
```

La partie réservée à la saisie des programmes ou des instructions exécutées en mode direct sera appelée "fenêtre de texte" dans la suite du livre.

Les caractères tapés au clavier sont affichés sur l'écran à l'emplacement occupé par le curseur.

Certaines touches du clavier ont des fonctions spécifiques sous STOS. En voici la liste :

- ✓ Les touches fléchées du clavier modifient l'emplacement du curseur.
- ✓ Par défaut, les caractères tapés se superposent aux éventuels caractères qui se trouvent sous le curseur (mode remplacement). Si vous le désirez, il est également possible d'insérer des caractères entre la position courante du curseur et le caractère

qui se trouve à gauche du curseur (mode INSERT). Pour cela, il suffit d'appuyer sur la touche <Insert> du clavier. Un autre appui sur cette touche vous fait repasser en mode remplacement.

- ✓ La touche <Backspace> efface le caractère situé à gauche du curseur.
- ✓ La touche <Delete> efface le caractère situé sous le curseur.
- ✓ La touche <Home> positionne le curseur en haut et à gauche de la fenêtre texte.
- ✓ La touche <Clr> efface l'écran et positionne le curseur en haut et à gauche de la fenêtre de texte.
- ✓ L'appui deux fois de suite sur la touche <Undo> réinitialise l'écran et l'éditeur du STOS.

Dans la partie supérieure de l'écran, vous pouvez voir une fenêtre dans laquelle sont listées les chaînes affectées aux touches de fonction <F1> à <F10>, par exemple :

```
f6: run'
```

Cette fenêtre peut être effacée en tapant :

```
key off
```

et réaffichée en tapant :

```
key on
```

L'appui sur une des touches de fonction équivaut à la frappe au clavier de la chaîne qui lui est affectée.

Appuyez sur une des touches <Shift> du clavier. Les touches <F1> à <F10> se transforment en <F11> à <F20>.

Les chaînes affectées aux touches <F1> à <F20> peuvent être affichées sur l'écran en tapant :

```
key list<CR>
```

ou en appuyant sur la touche <F20> (<Shift> <F10>).

1.4. Les divers aspects du STOS

Si vous pensez que le STOS est un langage BASIC comme les autres, je vais essayer de vous convaincre du contraire.

Certes, le BASIC STOS peut être utilisé comme un BASIC interprété classique. Mais certaines de ses instructions permettent à un programmeur débutant de gérer des sprites et d'accéder au générateur sonore sous interruptions. En d'autres termes, gérer des sprites sous interruptions signifie qu'un ou plusieurs objets peuvent être animés sur l'écran pendant qu'un programme est en train d'exécuter une autre séquence d'instructions. De même, accéder au générateur sonore sous interruptions permet d'écouter une mélodie alors qu'un programme est en train d'exécuter une séquence d'instructions qui n'a rien à voir avec le son produit.

Si vous n'êtes pas encore convaincu des nombreuses possibilités du STOS, sachez que plusieurs programmes utilitaires (appelés "accessoires") sont livrés avec le STOS. Ils sont accessibles pendant l'édition d'un programme. Ces accessoires vous permettent entre autres de créer des sprites, des jeux de caractères ou des mélodies interruptibles. Ils seront étudiés en détail dans la suite du livre.

Chapitre 2

Les accessoires

2.1. L'éditeur de sprites

L'éditeur de sprites est un outil de dessin et d'animation qui permet de créer des sprites de toutes tailles, et de les enchaîner selon une séquence et une vitesse paramétrables. Ces sprites sont stockés sur disque dans des banques directement exploitables par des programmes écrits en STOS.

Pour accéder à l'éditeur de sprites, il vous faut le charger en mémoire. Comme l'éditeur est un accessoire écrit en STOS, vous pouvez procéder de deux façons.

Dans tous les cas, bootez avec la disquette STOS puis insérez la disquette Accessoires dans le lecteur A:.

Première méthode

Chargement en tant que programme :

Tapez en mode direct

```
load "sprite.acb"<CR>
```

Quelques secondes après, le programme "sprite.acb" est en mémoire. Si vous le souhaitez, vous pouvez le lister en tapant :

```
list<CR>
```

Le but de ce chapitre n'est pas l'étude du programme, aussi nous contenterons-nous de l'exécuter. Tapez :

```
run<CR>
```

Seconde méthode

Chargement en tant qu'accessoire :

Si votre ordinateur est un 520 ST, nous vous conseillons de vider la mémoire des éventuels accessoires qu'elle contient en tapant en mode direct :

```
accnew<CR>
```

Chargez l'accessoire en tapant :

```
accload"sprite.acb"<CR>
```

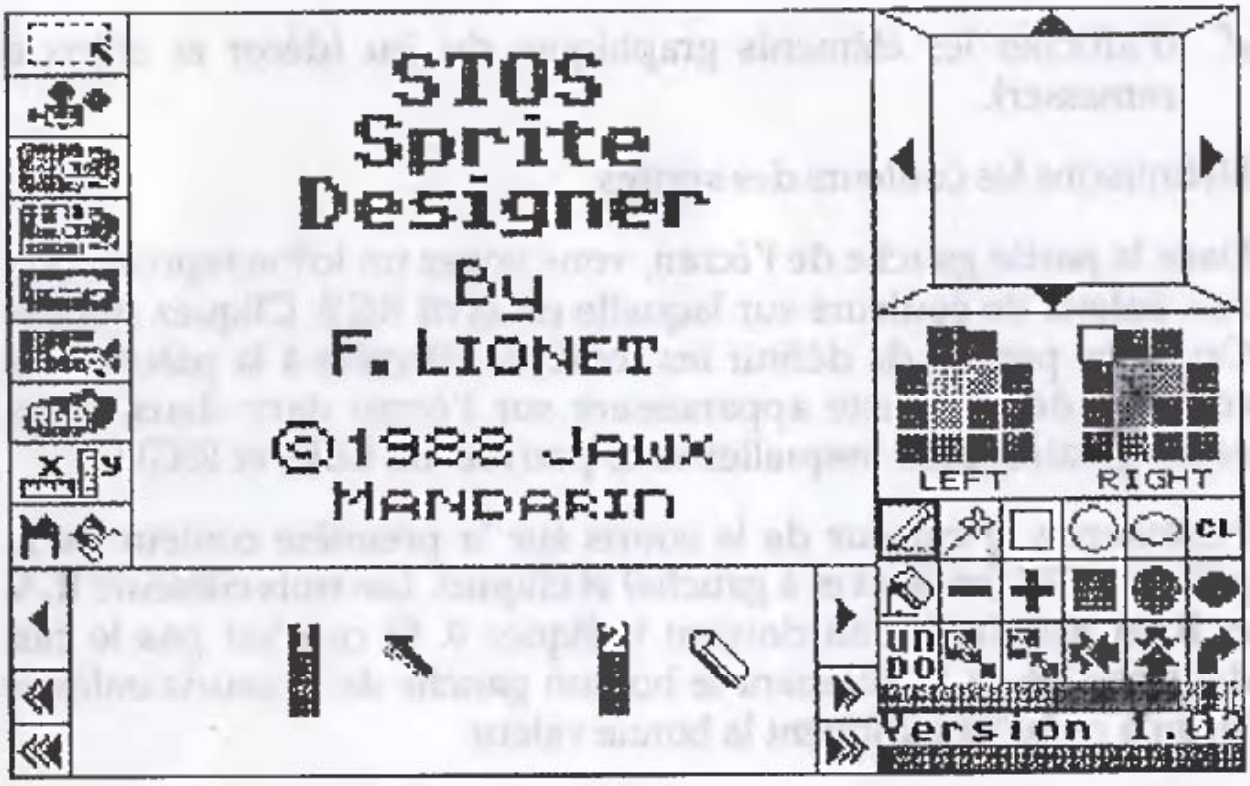
Activez l'éditeur de sprites en appuyant sur la touche <Help> du clavier. Une fenêtre du type suivant est affiché sur l'écran :

```
                Programme édité: 1
-------------------------------------------------
|P| Taille |Fn #1|Fn #2|Fn #3|Fn #4|
-------------------------------------------------
|1|97946   |end  |     |     |     |
|2|0       |     |end  |     |     |
|3|0       |     |     |end  |     |
|4|0       |     |     |     |end  |
-------------------------------------------------

        Accessoires basic chargés:
f1-SPRITE    f5-          f9-
f2-          f6-          f10-
f3-          f7-          f11-
f4-          f8-          f12-

  Mémoire restante: 1510570bytes.
```

Appuyez sur la touche <F1> pour activer l'accessoire SPRITE. Quelques secondes après, l'écran doit se présenter comme suit :



Cliquez sur un des boutons de la souris. Le texte affiché dans la fenêtre centrale s'efface et le curseur de la souris apparaît sur l'écran. Déplacez-le et vous verrez qu'il change de forme en fonction de sa position. Il se transforme en un crayon dans la partie centrale de l'écran et en une flèche partout ailleurs. La zone où le curseur a la forme d'un crayon est la zone de dessin.

Plutôt que de décrire la fonction de chacun des icônes affichés sur l'écran, nous allons directement vous apprendre à créer des sprites. Lorsque vous aurez acquis toutes les connaissances relatives à un élément particulier de l'éditeur de sprites, nous marquerons une pause récapitulative, puis nous repartirons sur un nouvel élément, jusqu'à avoir fait le tour de l'éditeur.

Vous êtes prêt ? Alors, commençons.

Nous allons créer les sprites utilisés par le jeu présenté dans la troisième partie du livre. Ces sprites permettront entre autres :

- ✓ d'animer les deux personnages principaux du jeu,
- ✓ d'afficher les éléments graphiques du jeu (décor et objets à ramasser).

Définissons les couleurs des sprites.

Dans la partie gauche de l'écran, vous voyez un icône représentant une palette de couleurs sur laquelle est écrit RGB. Cliquez dessus. Cet icône permet de définir les couleurs affectées à la palette. Les couleurs de la palette apparaissent sur l'écran dans deux zones rectangulaires sous lesquelles vous pouvez lire LEFT et RIGHT.

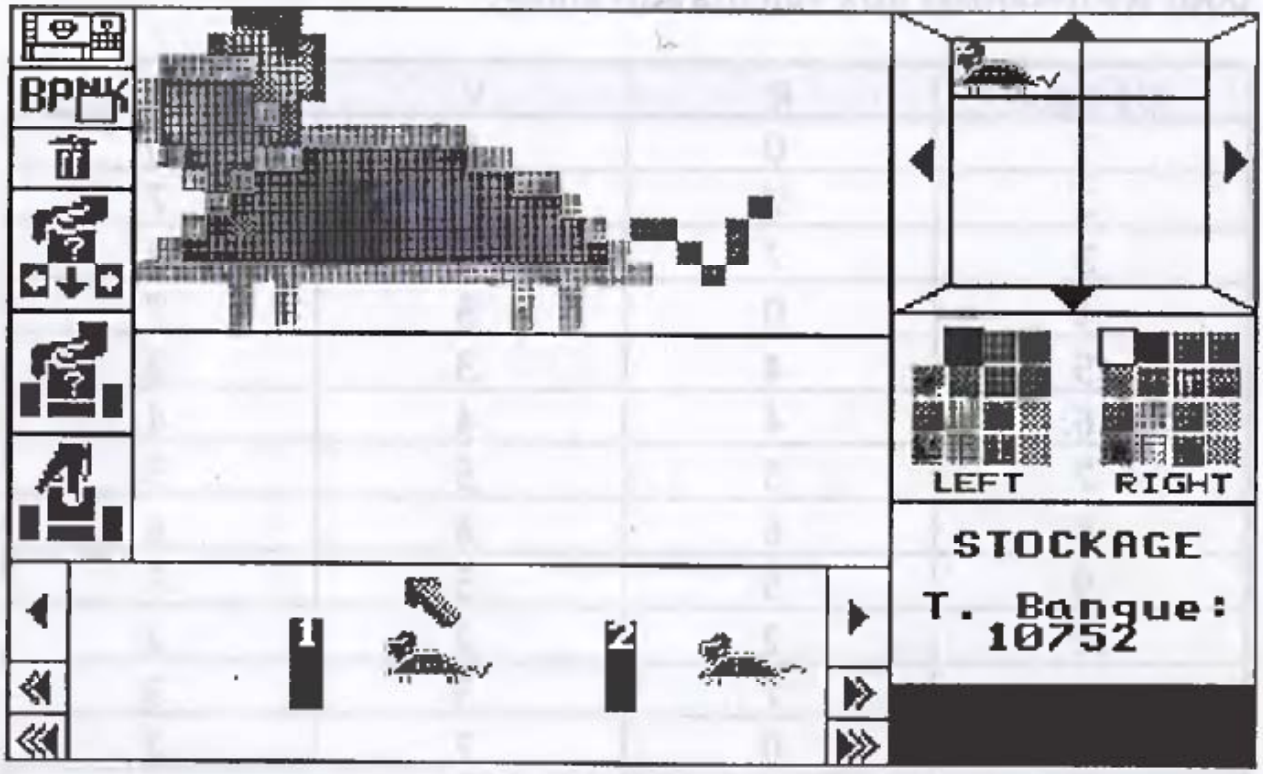
Positionnez le curseur de la souris sur la première couleur de la palette LEFT (en haut et à gauche) et cliquez. Les trois curseurs R, V et B en bas de l'écran doivent indiquer 0. Si ce n'est pas le cas, déplacez-les en maintenant le bouton gauche de la souris enfoncé jusqu'à ce qu'ils indiquent la bonne valeur.

Recommencez ces opérations sur les autres couleurs de la palette pour les initialiser aux valeurs suivantes :

Couleur	R	V	B
1	0	0	0
2	7	7	7
3	7	6	2
4	0	5	7
5	4	3	2
6	4	4	4
7	5	5	5
8	6	6	6
9	5	5	5
10	2	2	2
11	7	3	3
12	0	7	2
13	4	2	1
14	3	1	1
15	7	5	3
16	5	3	1

Cliquez sur le bouton droit de la souris pour revenir au menu principal.

Définissez le sprite suivant :



Il utilise les couleurs 6, 7, 8, 10 et 11.

Pour connaître les couleurs utilisées pour chaque partie du sprite, repérez-vous à partir des trames présentes sur le sprite et la palette LEFT ou RIGHT.

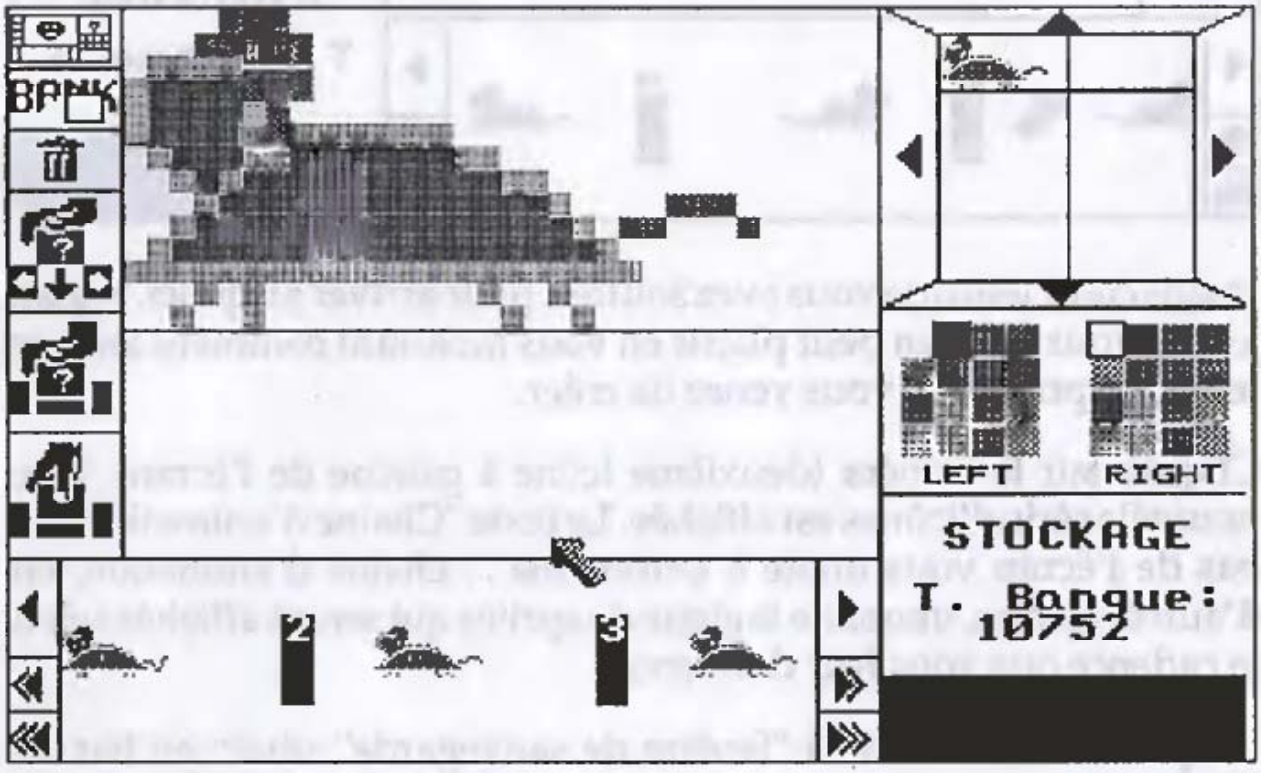
Pour affecter une couleur à un des boutons de la souris, il suffit de positionner le curseur de la souris sur la couleur désirée dans la palette LEFT ou RIGHT et de cliquer dessus. Tout appui sur ce bouton dans la fenêtre de dessin utilisera la couleur ainsi définie.

Modifiez la taille du sprite pour l'amener à 32x14 pixels. Pour cela, cliquez sur le huitième icône du menu principal (icône XY). Reportez-vous dans la petite fenêtre située en haut et à droite de l'écran. Pointez le curseur de la souris à l'intersection des lignes verticale et horizontale. Maintenez le bouton gauche de la souris enfoncé et déplacez la souris jusqu'à avoir les dimensions requises. Repassez au menu principal en cliquant sur le bouton droit de la souris.

Sauvegardez le sprite dans la fenêtre située en bas de l'écran. Pour ce faire, cliquez sur l'icône STOCKER, c'est-à-dire sur le neuvième icône à gauche de l'écran (le dernier).

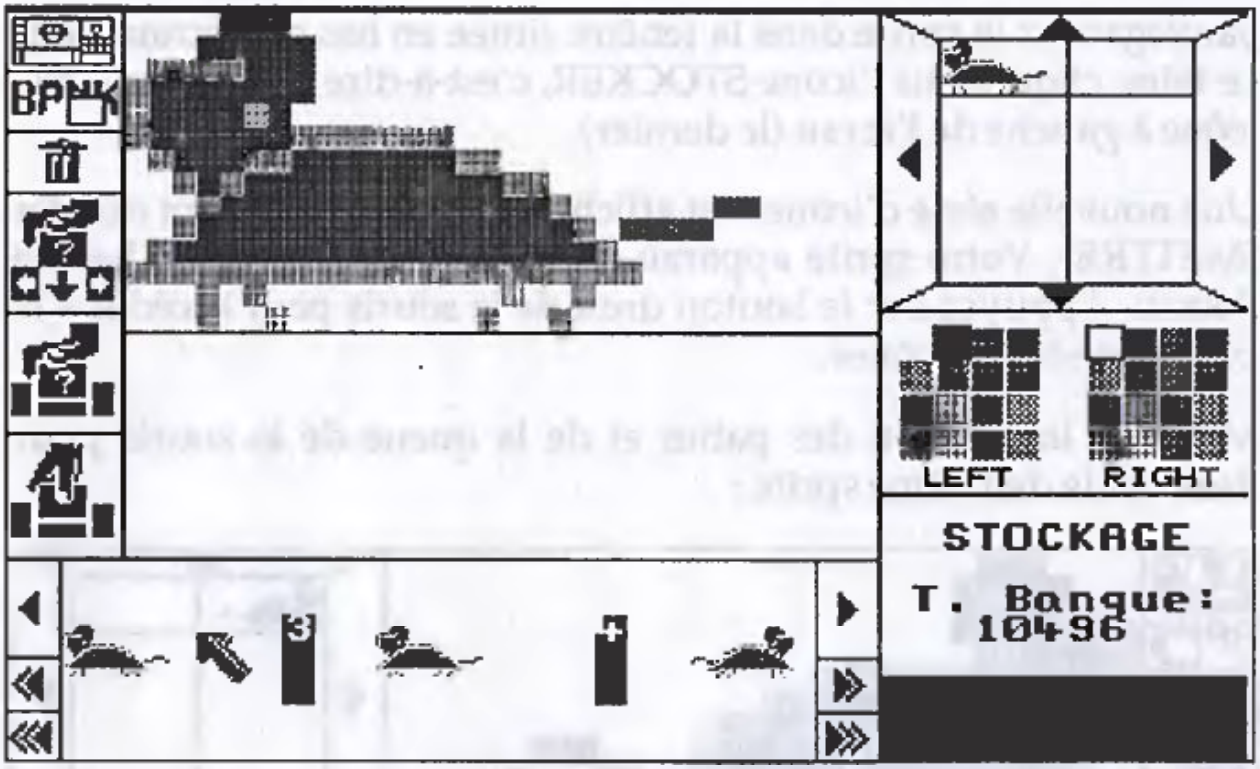
Une nouvelle série d'icônes est affichée. Cliquez sur l'avant dernier (METTRE). Votre sprite apparaît dans la fenêtre située en bas de l'écran. Appuyez sur le bouton droit de la souris pour accéder à la première série d'icônes.

Modifiez la position des pattes et de la queue de la souris pour dessiner le deuxième sprite :



Sauvegardez ce sprite dans la fenêtre située en bas de l'écran comme expliqué précédemment.

Modifiez une nouvelle fois la position des pattes et de la queue de la souris, et déplacez son oeil pour dessiner le troisième sprite :



Je suis conscient que vous avez souffert pour arriver jusqu'ici. Aussi, je vais vous faire un petit plaisir en vous montrant comment animer les trois sprites que vous venez de créer.

Cliquez sur la caméra (deuxième icône à gauche de l'écran). Une nouvelle série d'icônes est affichée. Le texte "Chaine d'animation" en bas de l'écran vous invite à entrer une ... chaîne d'animation, en d'autres termes, une suite logique de sprites qui seront affichés selon la cadence que vous leur donnerez.

De part et d'autre de la "fenêtre de sauvegarde" située en bas de l'écran se trouvent plusieurs icônes représentant des flèches. Les flèches simples vous permettent de faire défiler vers la gauche ou vers la droite les sprites sauvegardés. Les flèches doubles ont la même fonction, mais le déplacement est plus rapide. Les flèches triples initialisent la fenêtre sur le premier ou sur le dernier sprite.

Cliquez sur la triple flèche gauche. La fenêtre affiche les premier et second sprites. Cliquez maintenant sur le premier sprite. La chaîne (1,5) apparaît sous le texte "Chaîne d'animation". Cliquez sur la

simple flèche droite, sur le deuxième, puis sur le troisième sprite. La chaîne d'animation doit maintenant être (1,5)(2,5)(3,5). La souris se déplace en remuant les pattes et la queue.

Vous avez certainement repris quelques forces à la vue de ce spectacle grandiose...

Mais peut-être trouvez-vous que la souris gambade un peu trop vite. Essayez donc de cliquer sur les icônes flèches situés en bas et à droite de l'écran. Comme vous pouvez vous en rendre compte, les icônes marqués A modifient la chaîne d'animation.

Encore un petit mot avant de nous remettre au travail. L'icône représentant une main tournée vers la droite anime le sprite selon la chaîne d'animation. L'icône main vers la gauche anime le sprite dans l'ordre inverse de la chaîne d'animation. L'escalier anime le sprite en mode pas à pas. Les icônes BACK GRND, LOAD NEO et LOAD DEGAS permettent de charger un écran au format Néochrome ou Degas pour constituer un décor.

Récapitulatif

Jusqu'ici, nous avons vu que :

- ✓ l'icône RGB permet de redéfinir les couleurs de la palette,
- ✓ une couleur est affectée à un des boutons de la souris par un simple clic dans la palette LEFT ou RIGHT,
- ✓ l'icône STOCKER donne accès à un nouveau jeu d'icônes qui permettent entre autres de sauvegarder les icônes courants dans une fenêtre de stockage,
- ✓ l'icône CAMERA permet de définir et de paramétrer une ou plusieurs chaînes d'animation.

Poursuivons notre définition de sprites. Appuyez sur le bouton droit de la souris pour revenir au menu principal. Nous allons maintenant définir les trois sprites qui permettront d'animer la souris vers la

droite. Je vous entends gémir. Rassurez-vous, vous n'allez pas devoir dessiner ces trois sprites. Il vous suffira d'utiliser une des symétries de l'éditeur.

Chargez le premier sprite dans la fenêtre de dessin. Pour cela :

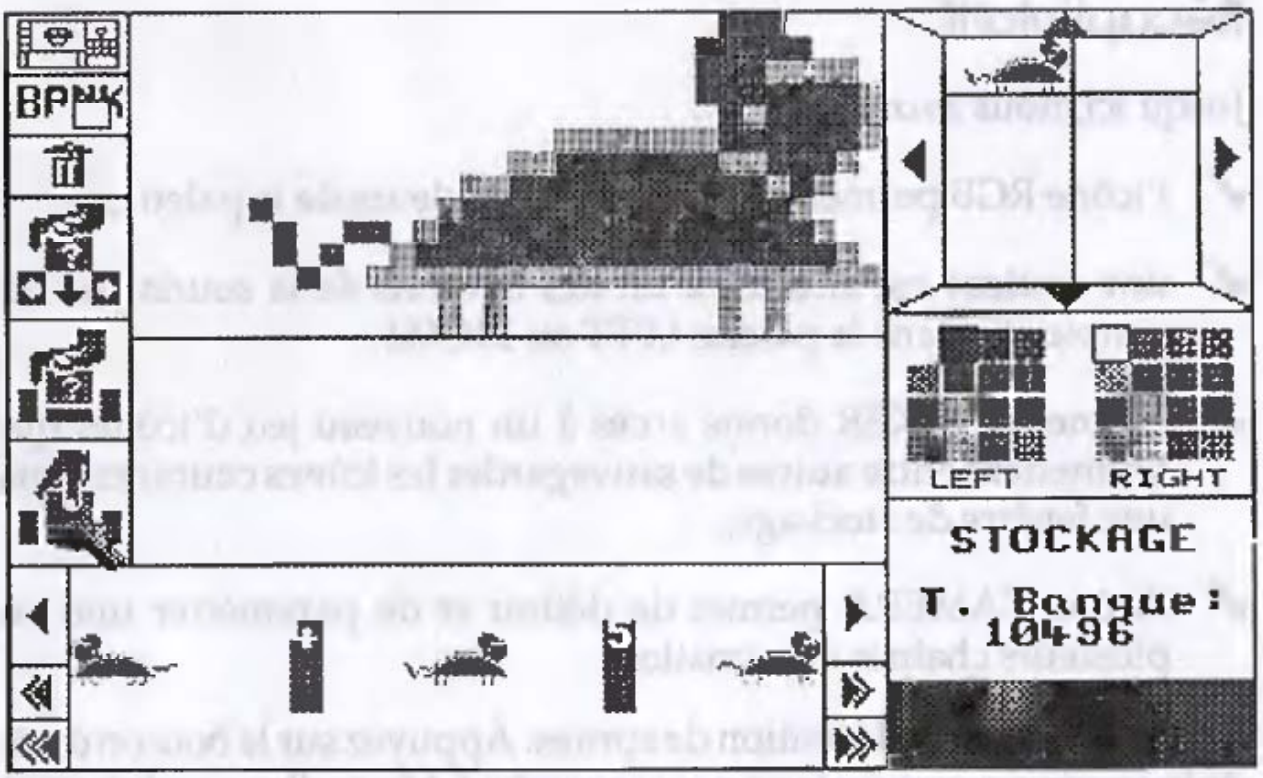
✓ cliquez sur l'icône STOCKER,

✓ appuyez sur la triple flèche gauche dans la fenêtre du bas de l'écran,

✓ cliquez sur l'icône PRENDRE (le dernier à gauche de l'écran).

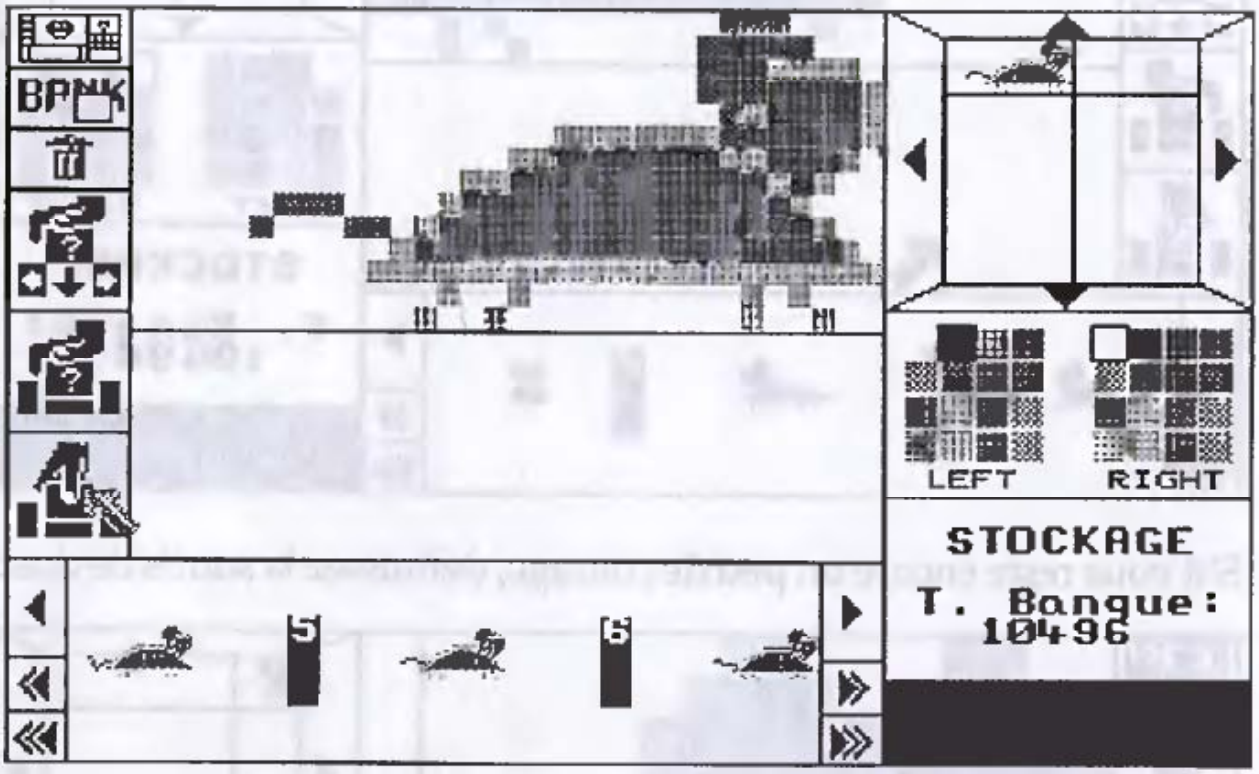
Repassez au menu principal en appuyant sur le bouton droit de la souris.

Déplacez le curseur de la souris sur la série d'icônes située en bas et à droite de l'écran, et cliquez sur l'icône représentant deux flèches droite/gauche supperposées. La souris est redessinée, la tête vers la droite.



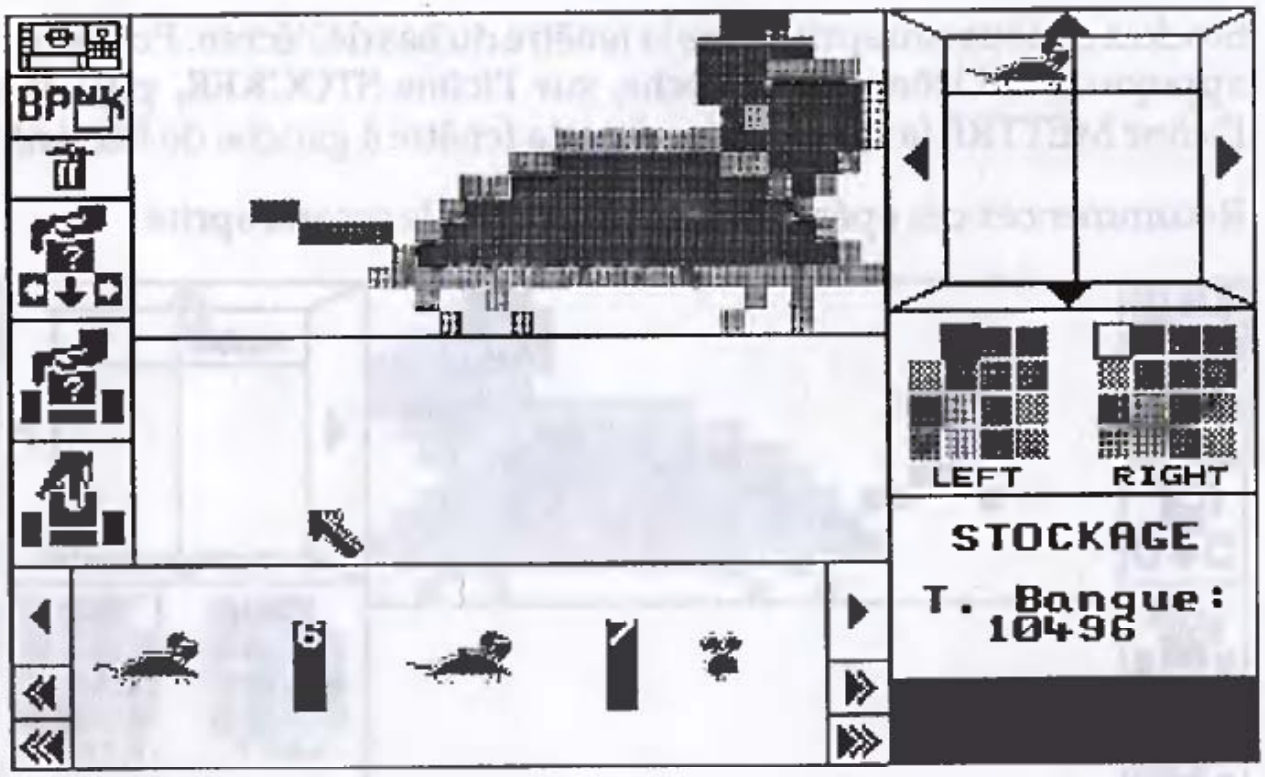
Stockez ce nouveau sprite dans la fenêtre du bas de l'écran. Pour cela, appuyez sur l'icône triple flèche, sur l'icône STOCKER, puis sur l'icône METTRE (avant dernier dans la fenêtre à gauche de l'écran).

Recommencez ces opérations pour inverser le second sprite :

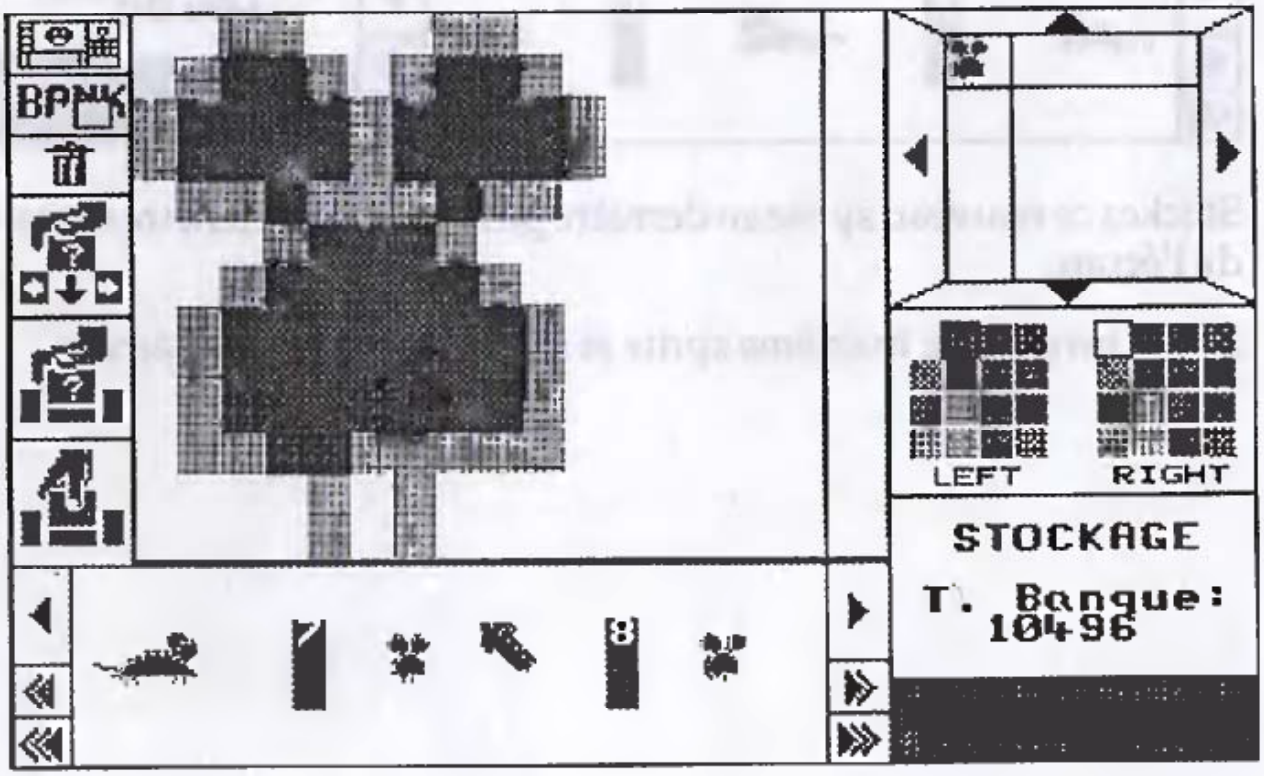


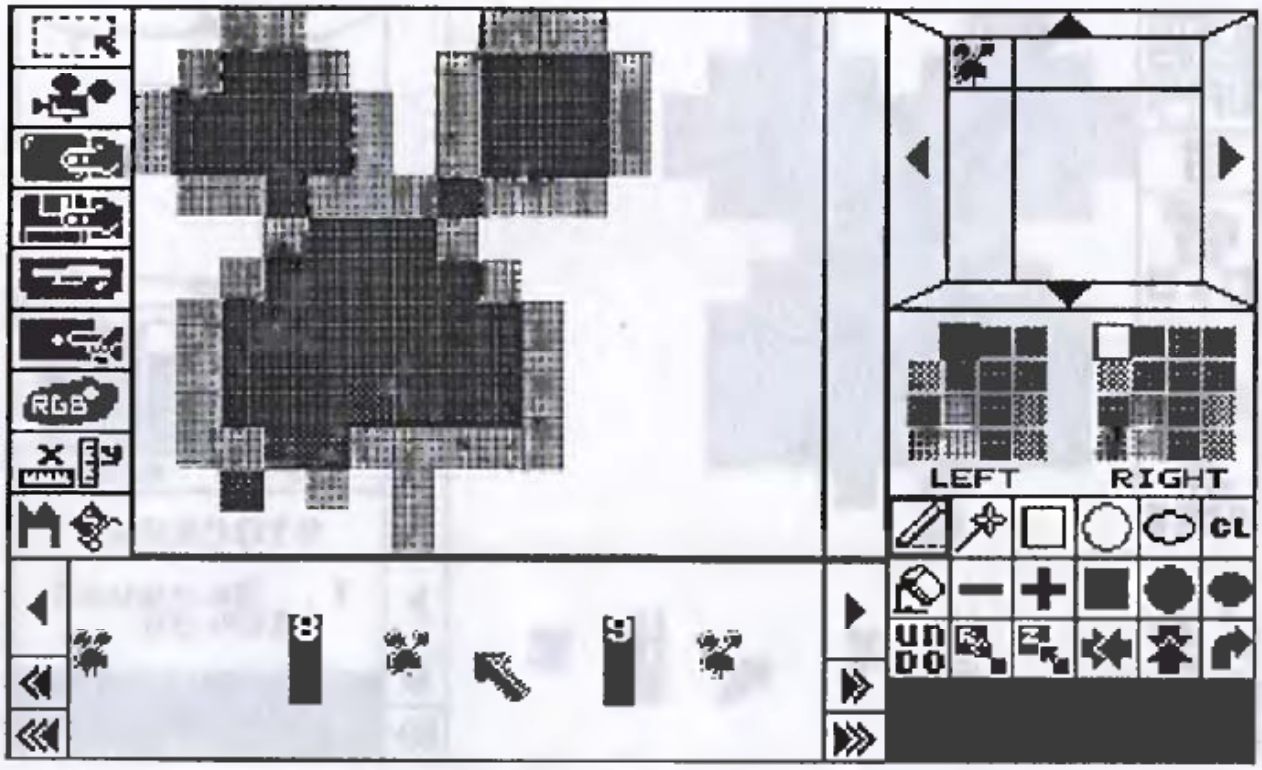
Stockez ce nouveau sprite en dernière position dans la fenêtre du bas de l'écran.

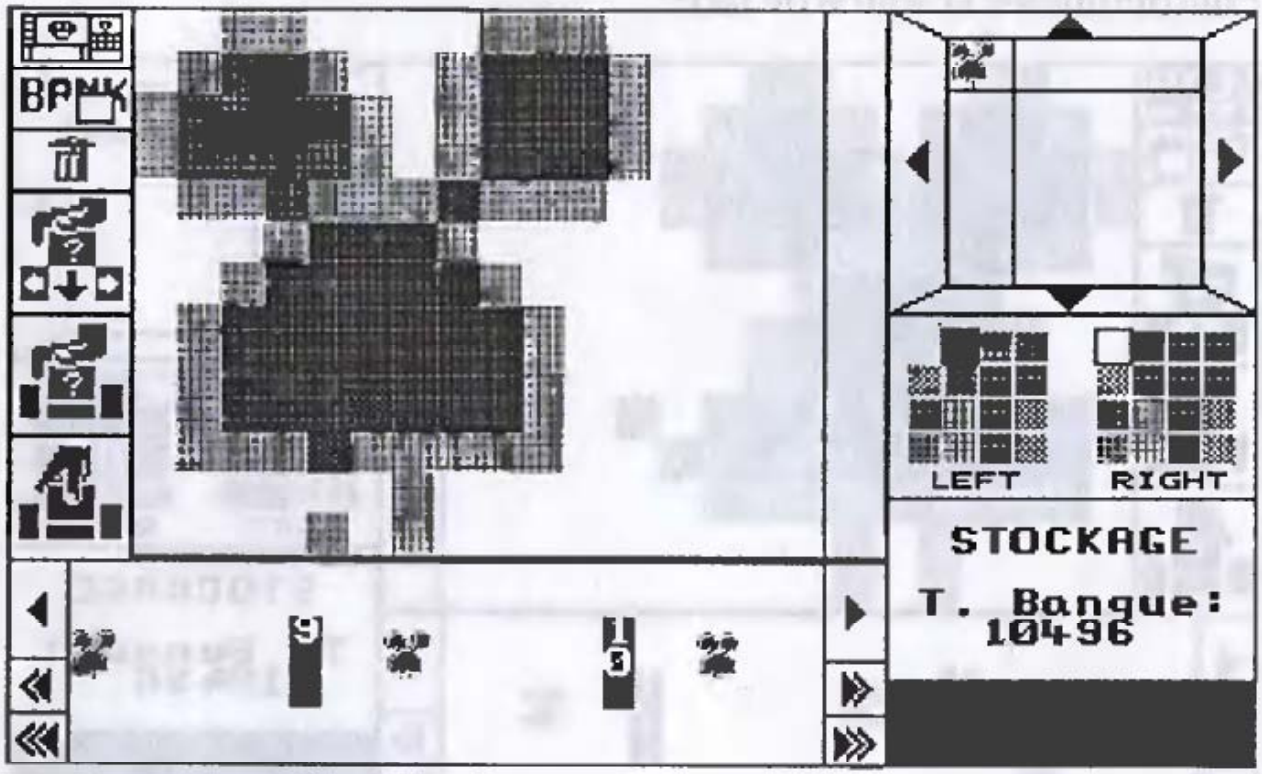
Enfin, inversez le troisième sprite et stockez-le en bas de l'écran:

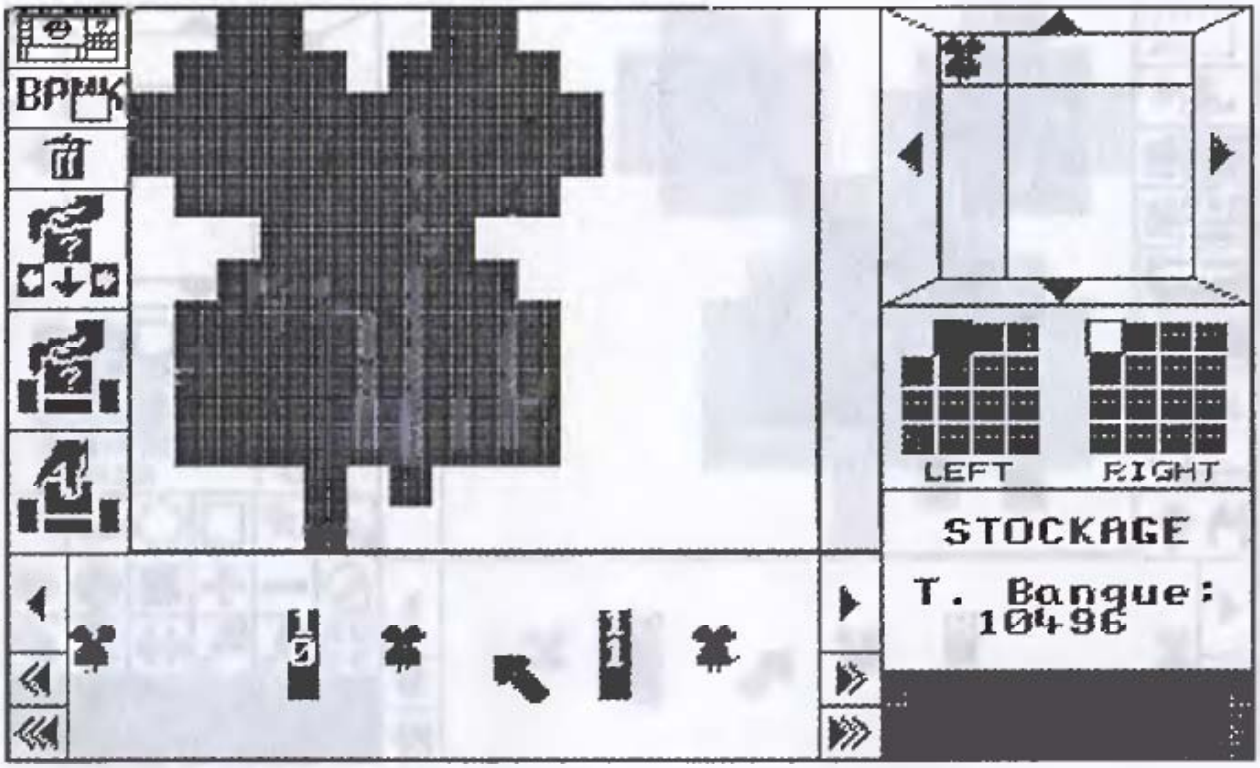


S'il vous reste encore un peu de courage, définissez la souris de dos :

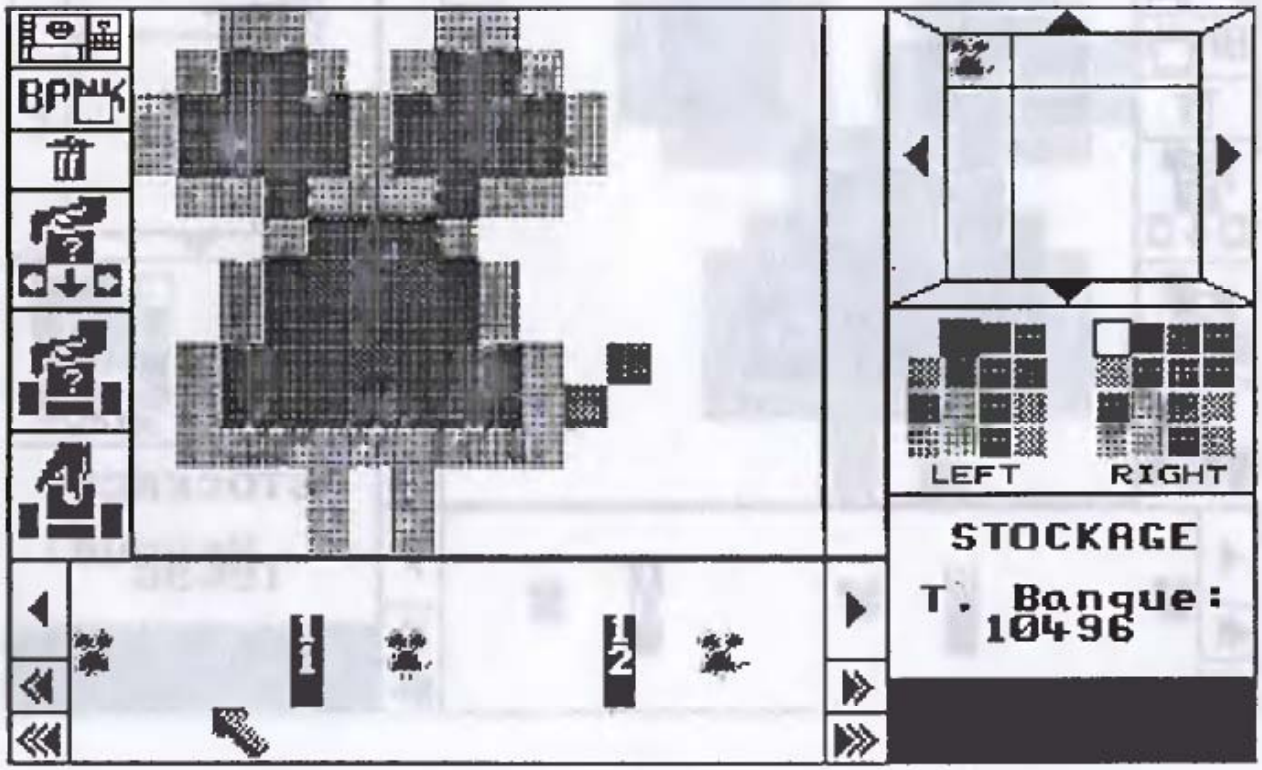


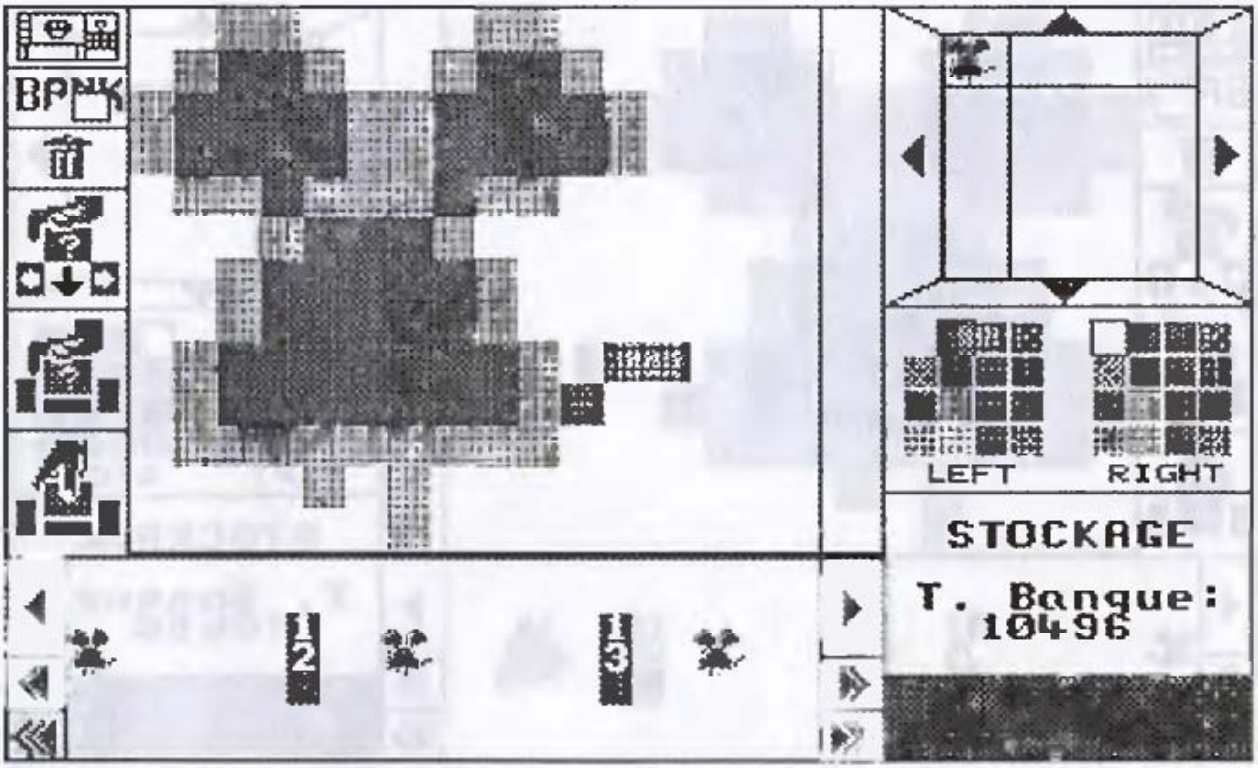

RGB
LEFT
RIGHT
CL
UN
DO

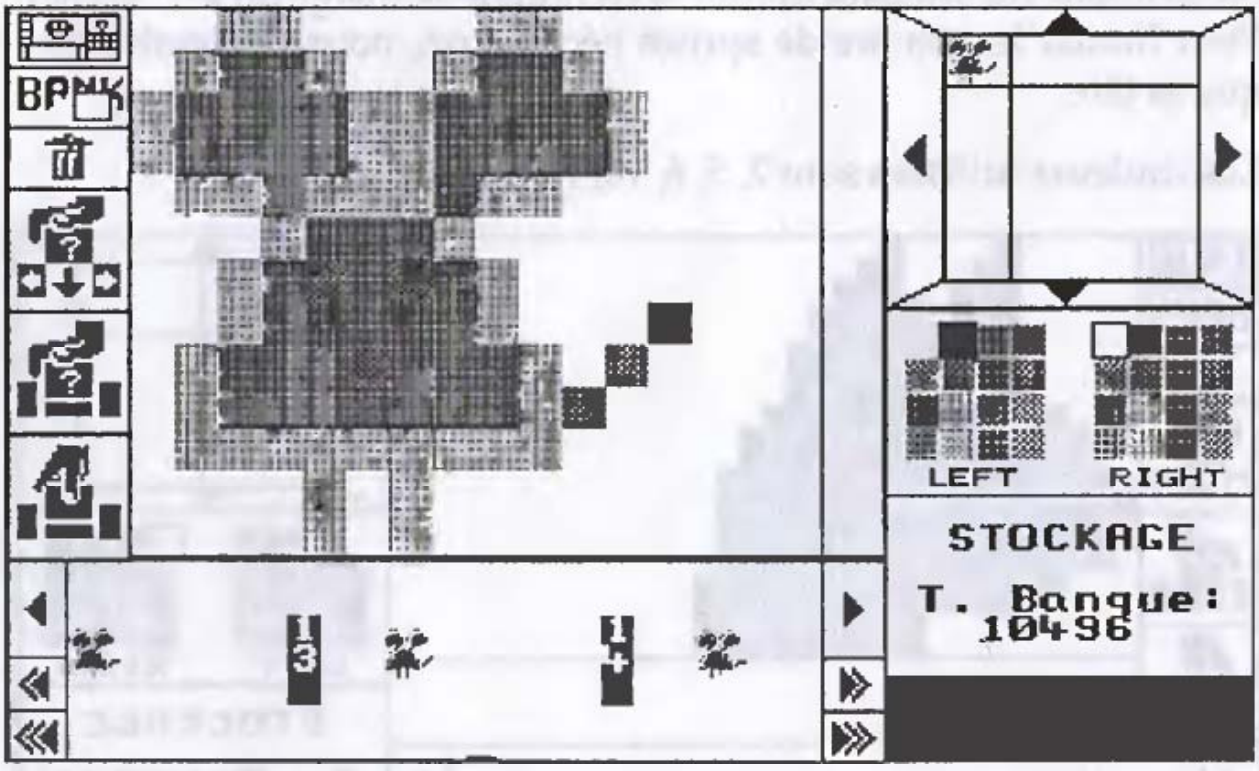

BP
LEFT
RIGHT
STOCKAGE
T. Banque:
10496

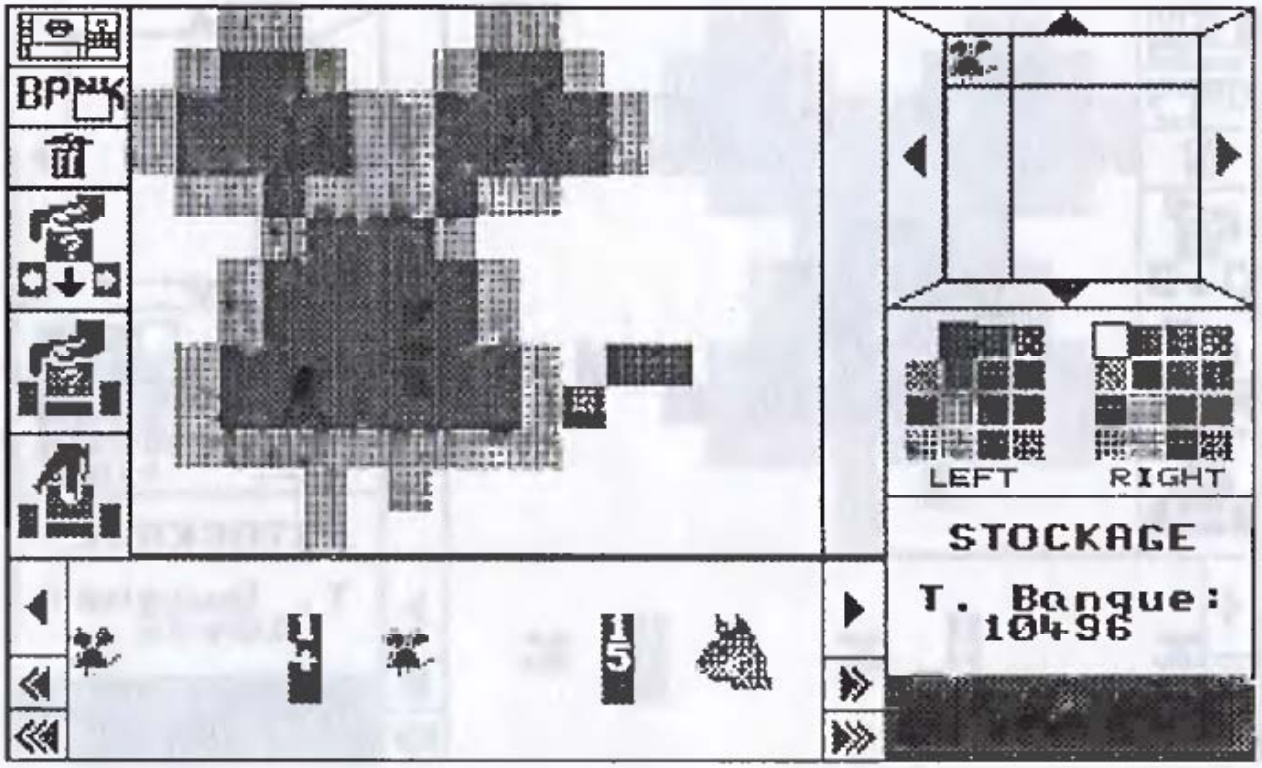


Puis définissez la souris de face :



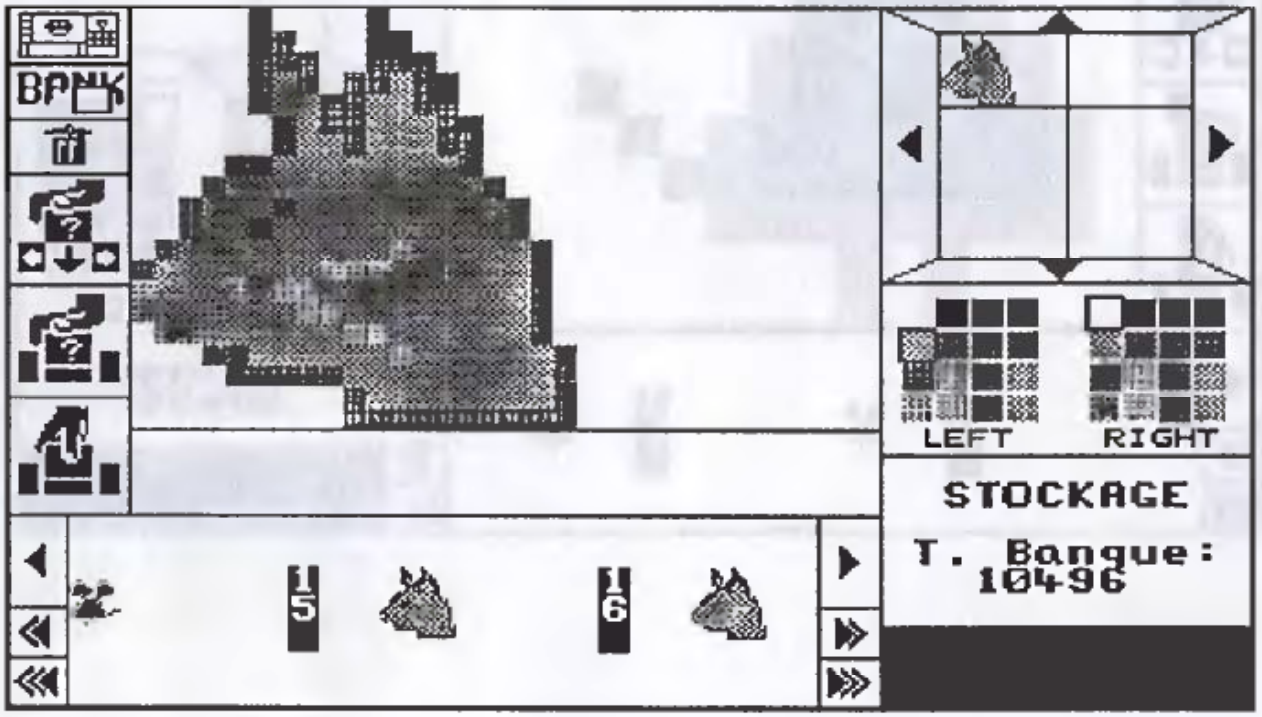

BANK
LEFT
RIGHT
STOCKAGE
T. Banque:
10496


BANK
LEFT
RIGHT
STOCKAGE
T. Banque:
10496

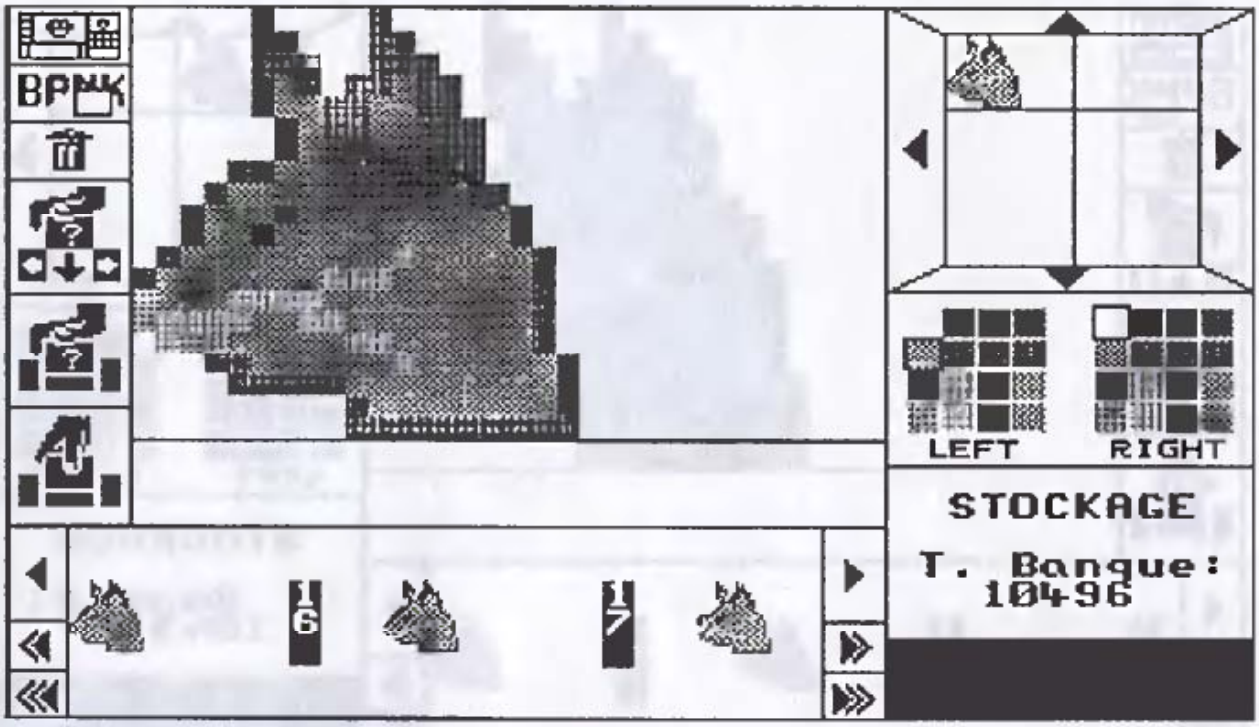


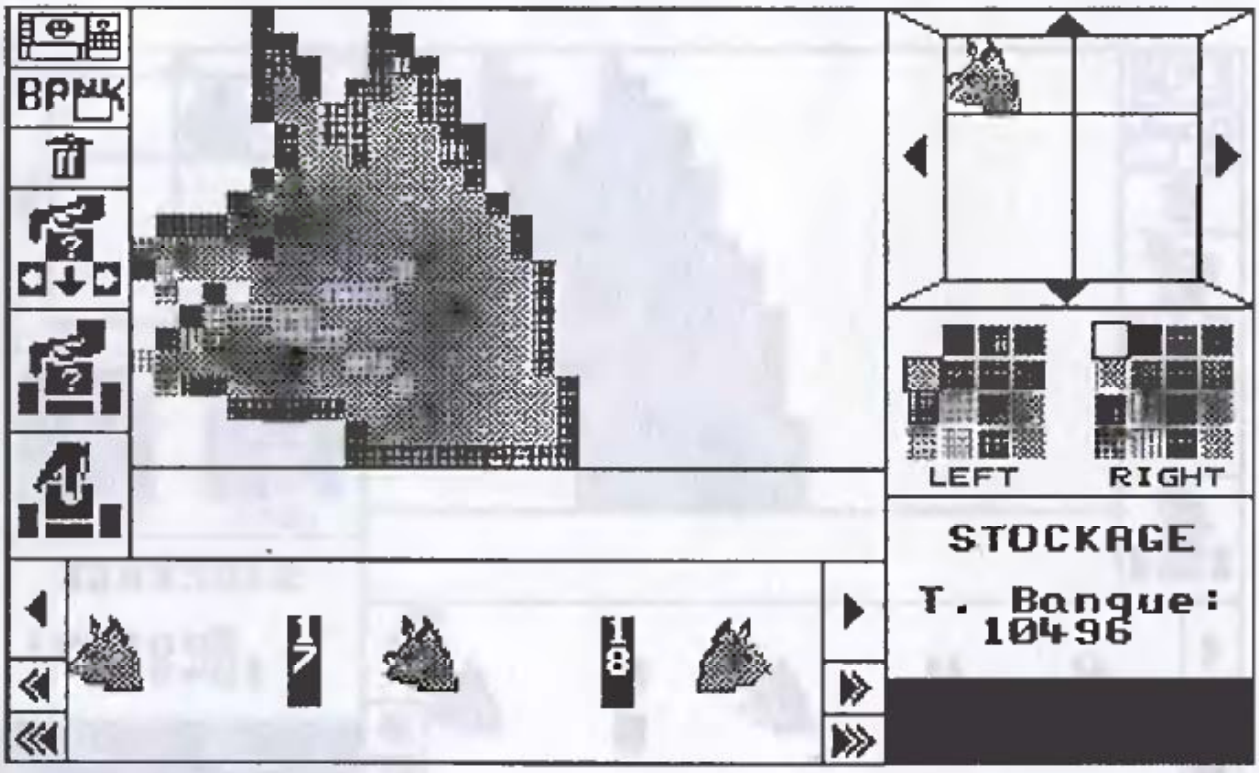
Nous allons maintenant définir le second personnage du jeu : le chat. Pour limiter le nombre de sprites nécessaires, nous ne dessinerons que sa tête.

Les couleurs utilisées sont 2, 5, 6, 10, 13, 15 et 16 :

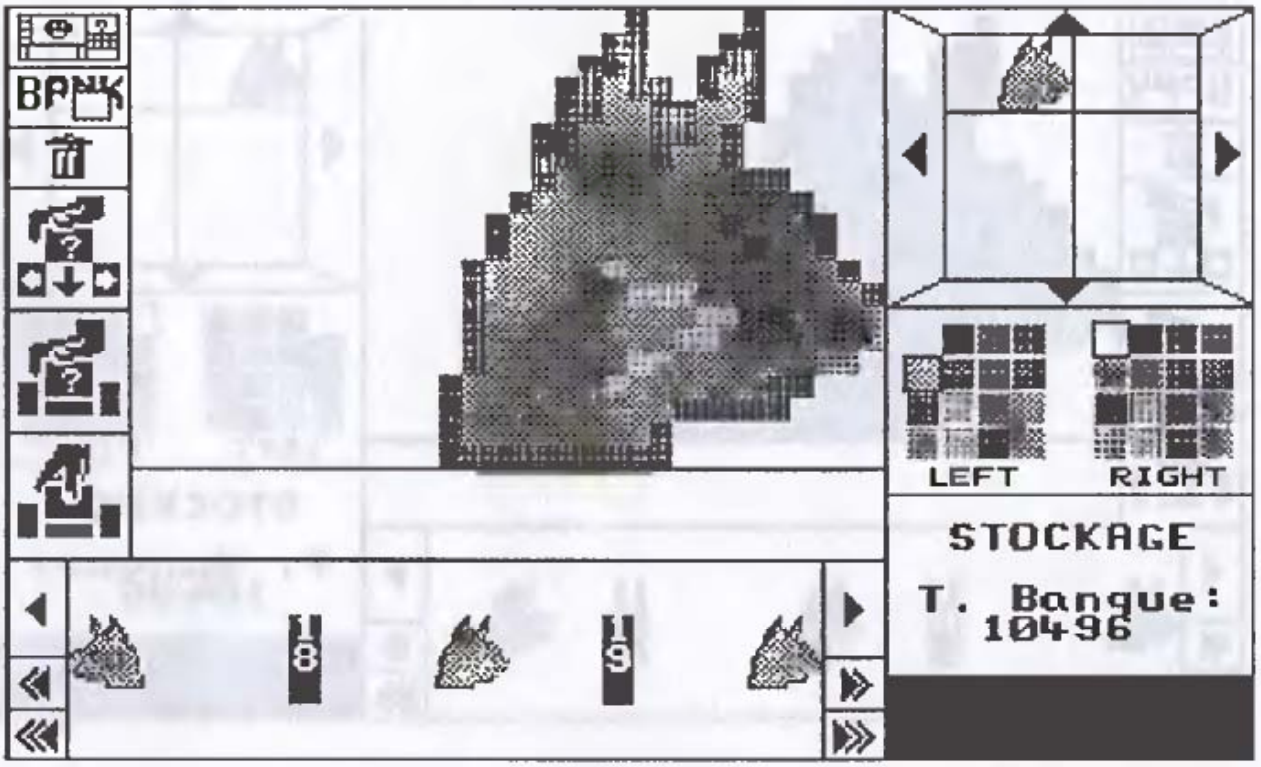


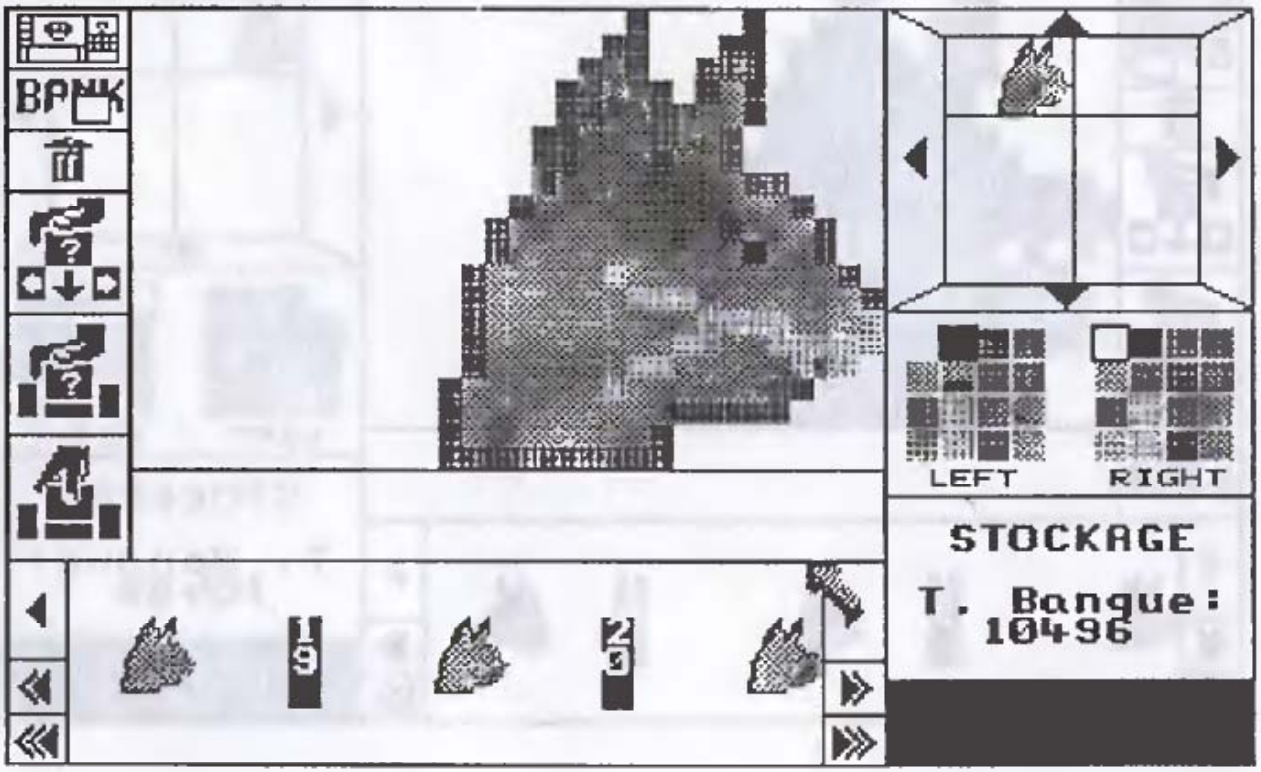
Stockez ce sprite dans la fenêtre en bas de l'écran selon le procédé habituel et modifiez-le pour créer ses deux variantes :

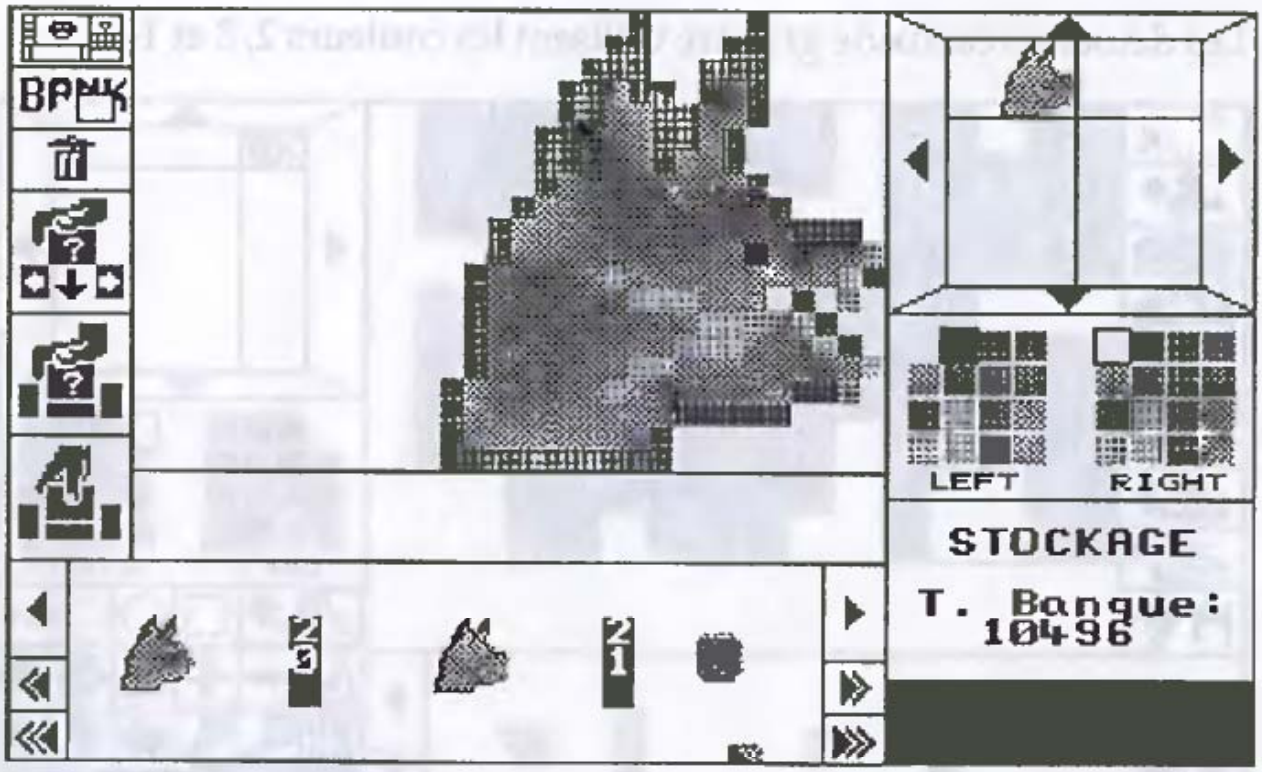




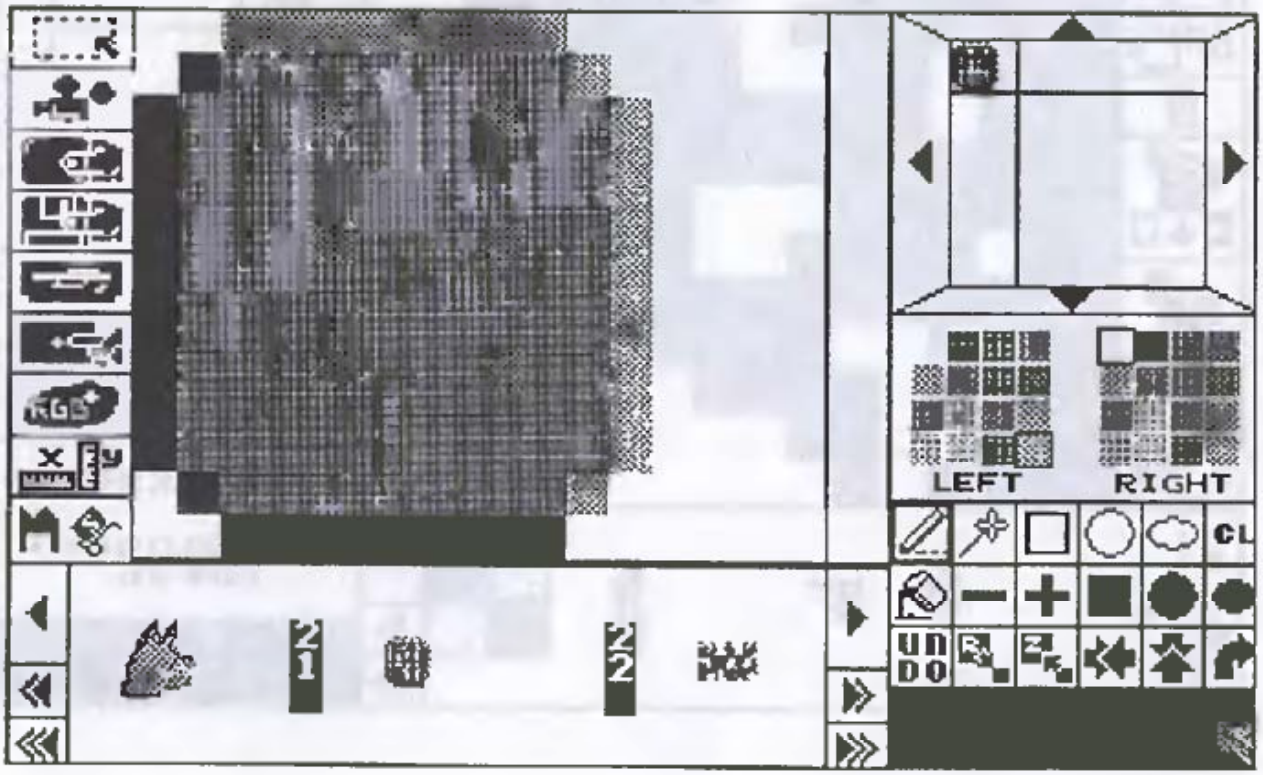
Utilisez l'icône de symétrie selon l'axe vertical pour créer les trois sprites symétriques des précédents :



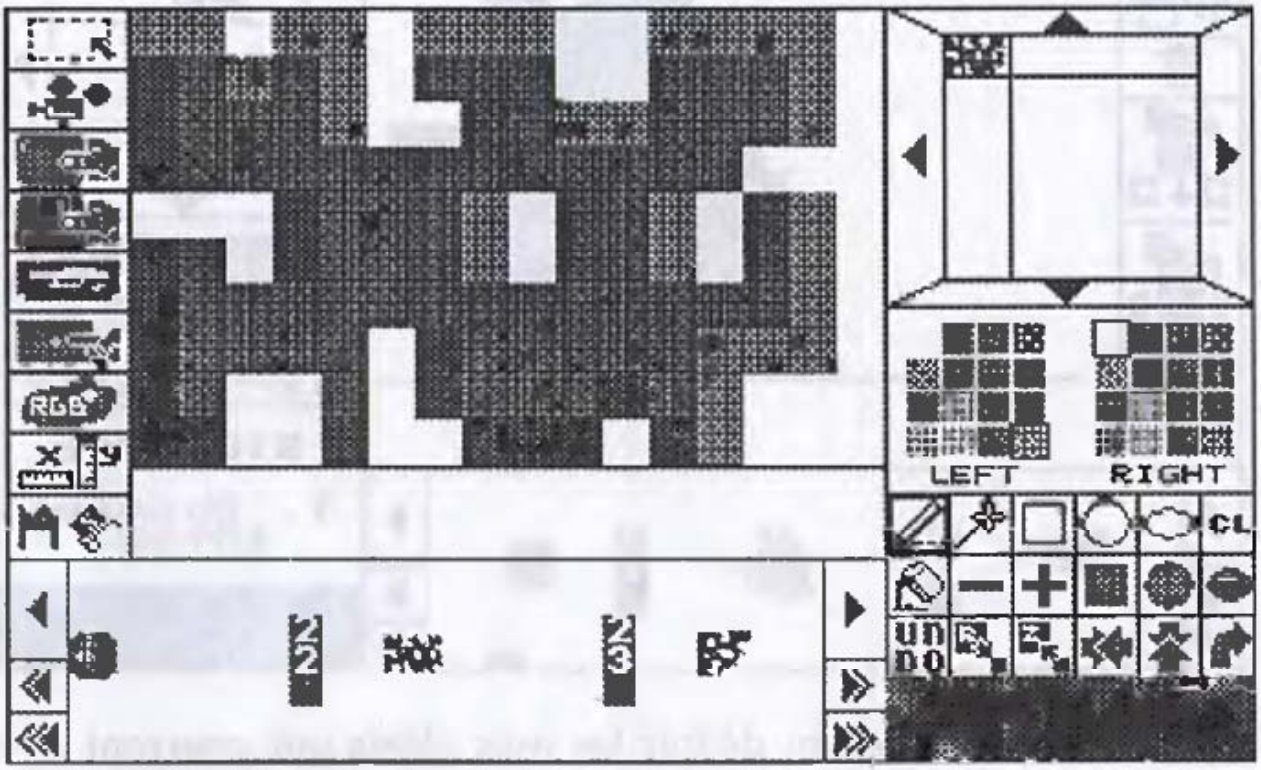


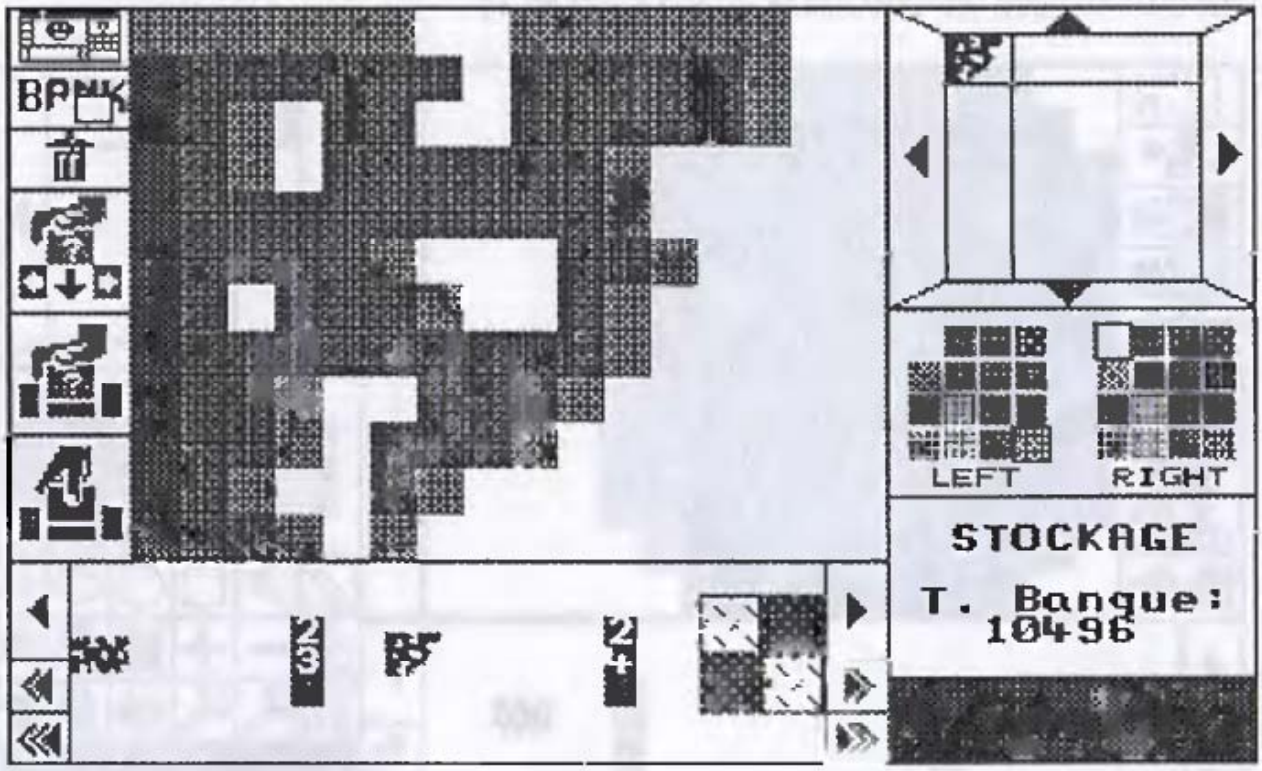


Nous allons maintenant définir les trois objets qui pourront être grignotés par la souris: un biscuit et deux morceaux de gruyère. Le biscuit utilise les couleurs 2, 3, 11, 15 et 16 :

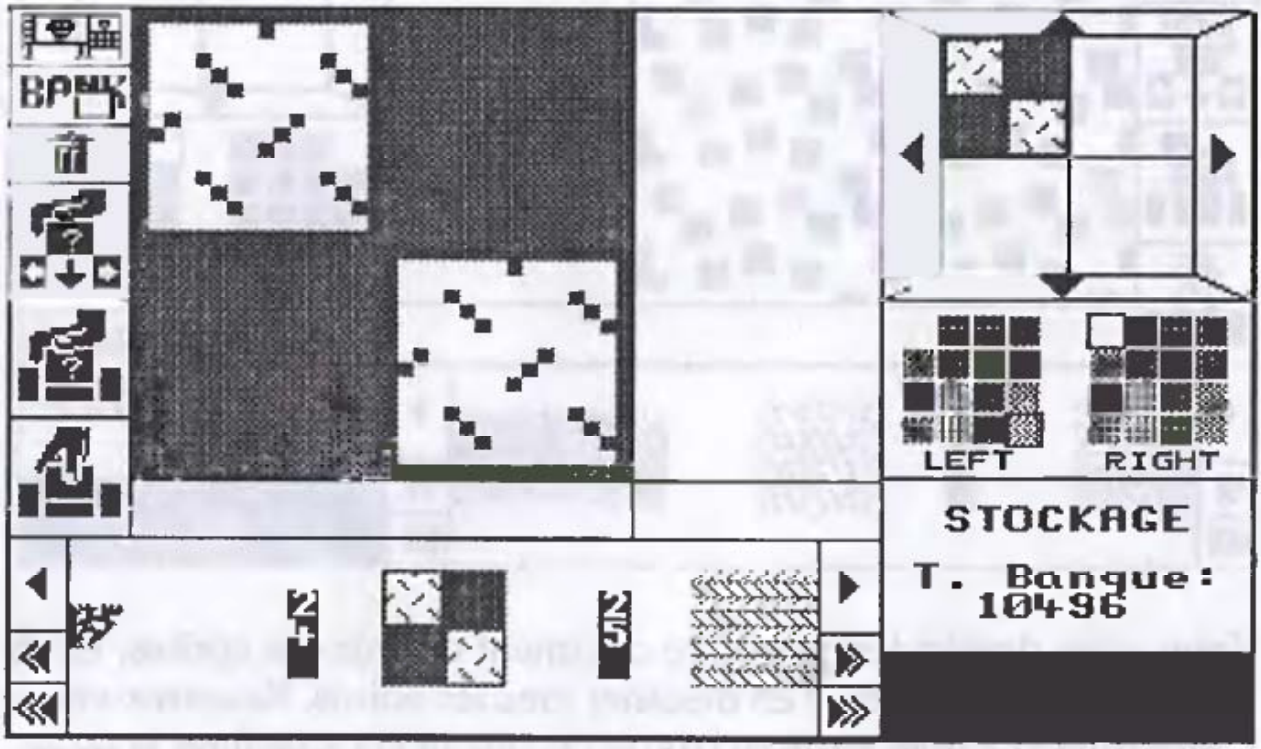


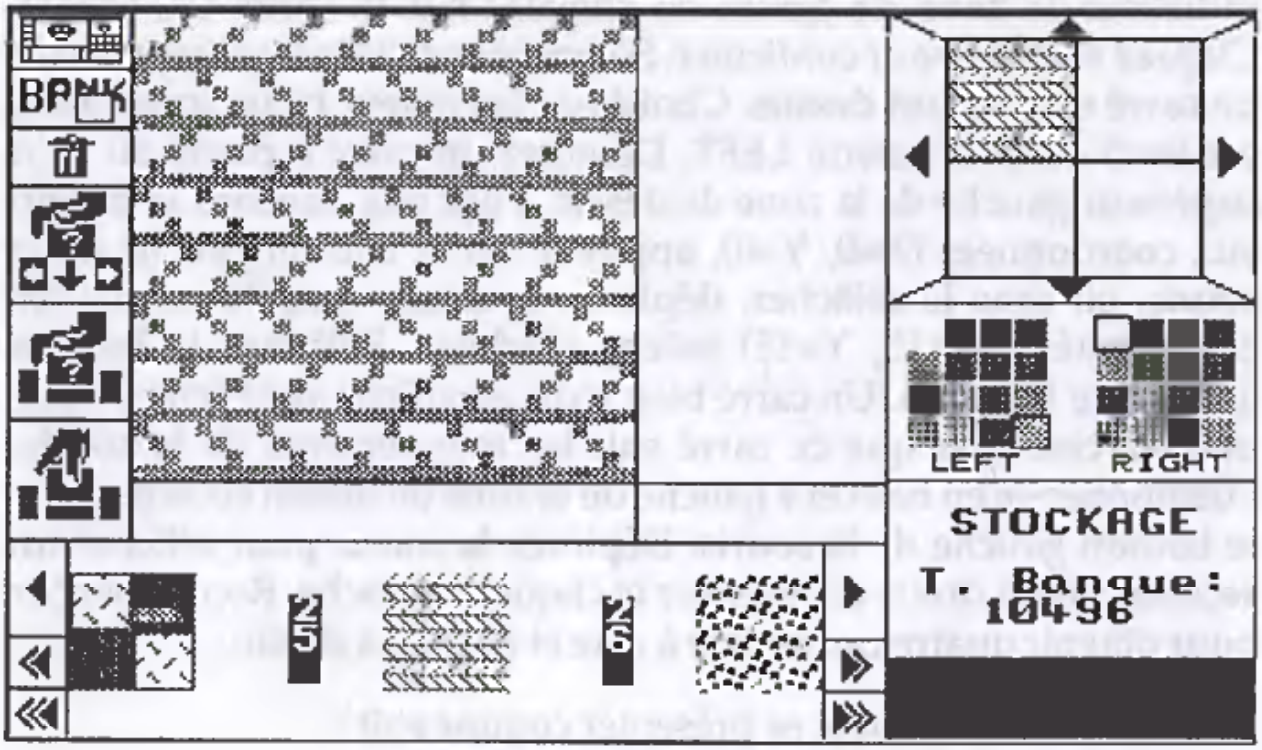
Les deux morceaux de gruyère utilisent les couleurs 2, 3 et 16 :

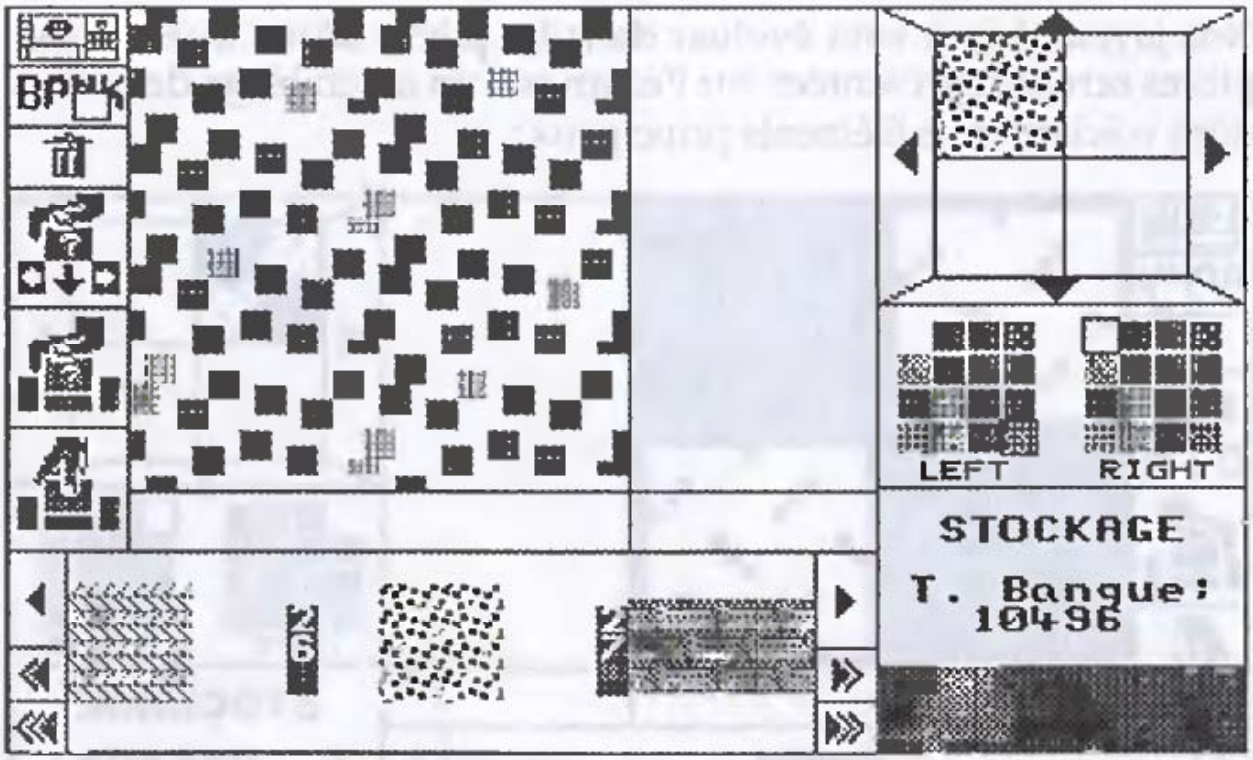




Nos joyeux héros vont évoluer dans les pièces d'une maison. Ces pièces seront représentées sur l'écran par un assemblage de sprites dont voici les trois éléments principaux :



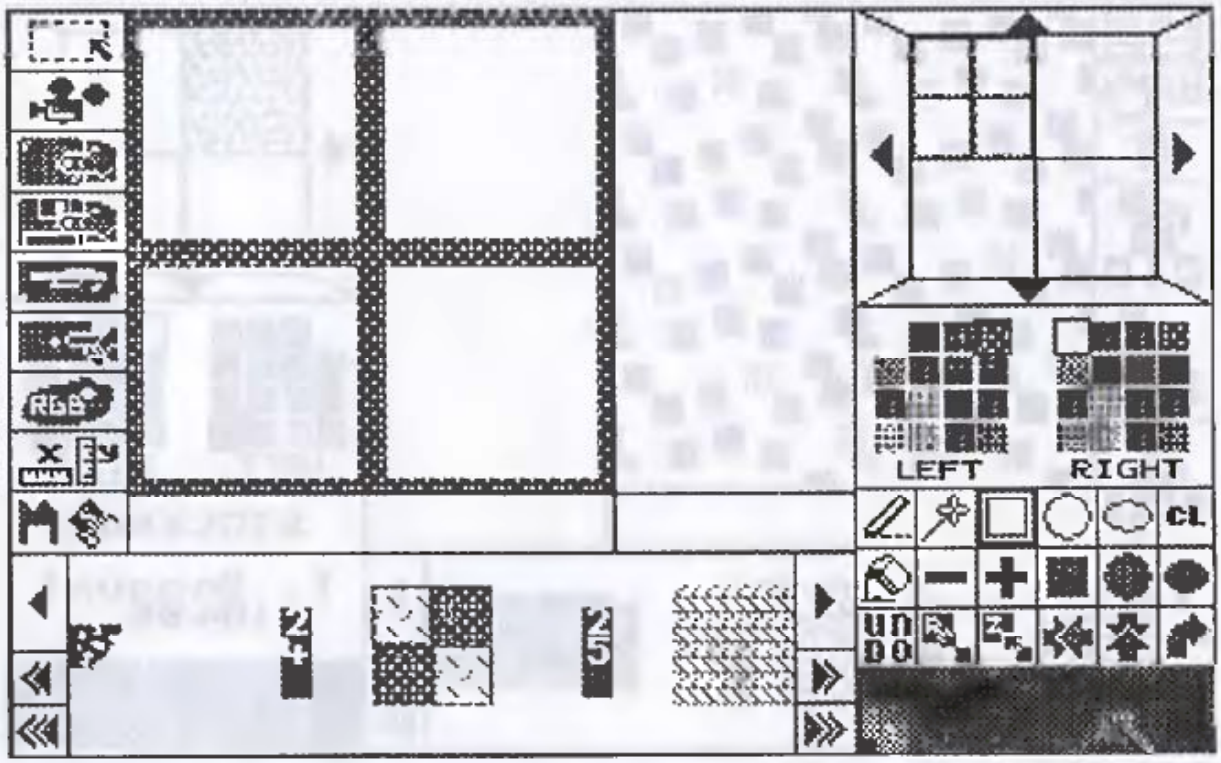




Vous vous demandez peut-être comment définir ces sprites, et en particulier si vous devez en dessiner tous les points. Rassurez-vous. L'ensemble d'icônes situé en bas de l'écran va vous faciliter la tâche.

Initialisez la zone de dessin en cliquant sur le sprite CL (clear). Cliquez sur OK pour confirmer. Sélectionnez l'icône qui représente un carré en cliquant dessus. Choisissez la couleur bleue (quatrième couleur) dans la palette LEFT. Dessinez un carré à partir du coin supérieur gauche de la zone de dessin. Pour cela, amenez le crayon aux coordonnées (X=0, Y=0), appuyez sur le bouton gauche de la souris, et, sans le relâcher, déplacez la souris jusqu'à ce que les coordonnées (X=15, Y=15) soient affichées. Relâchez le bouton gauche de la souris. Un carré bleu vide est affiché sur l'écran. Vous pouvez constater que ce carré suit les mouvements de la souris. Positionnez-le en haut et à gauche de la zone de dessin et cliquez sur le bouton gauche de la souris. Déplacez la souris pour afficher un second carré à droite du premier et cliquez à gauche. Recommencez pour obtenir quatre carrés côte à côte et cliquez à droite.

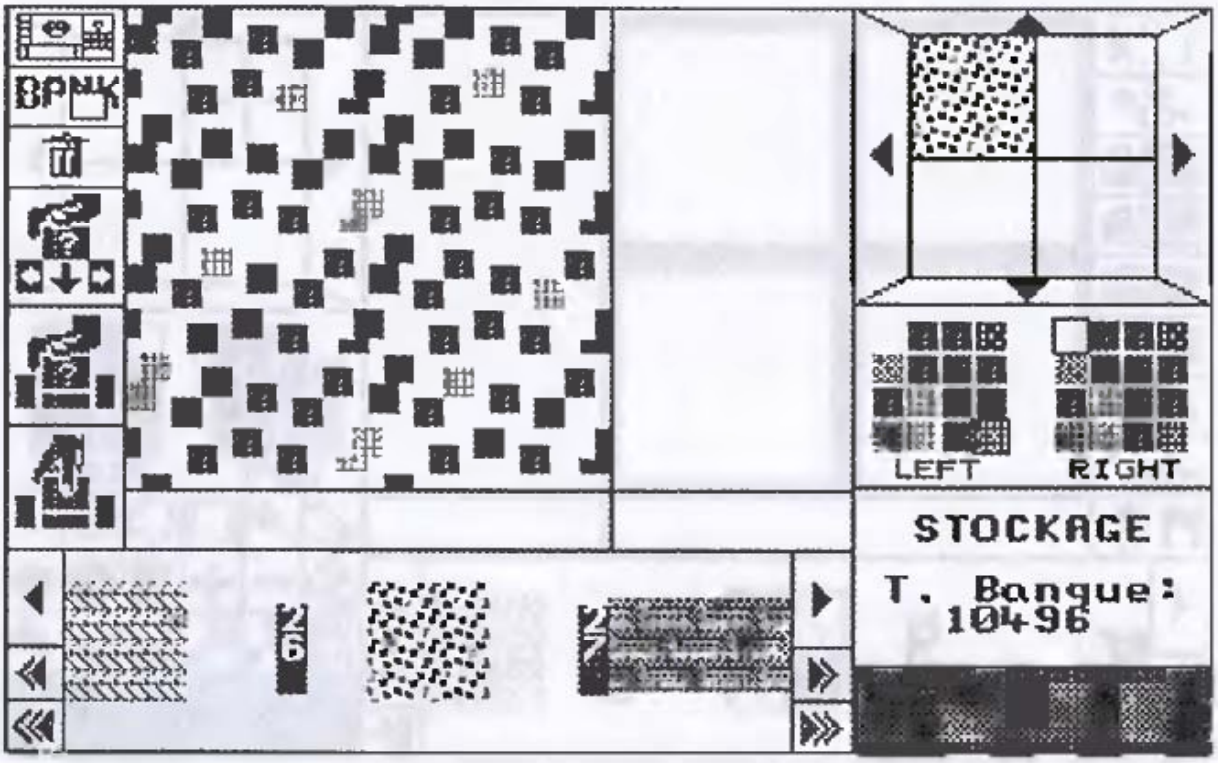
L'écran doit maintenant se présenter comme suit :



Cliquez sur l'icône qui représente un pot de peinture en train de se vider en bas et à droite de l'écran. Positionnez le curseur (qui a maintenant la même forme que l'icône) sur le carré en haut et à droite du sprite, et cliquez à gauche. Le carré se remplit en bleu. Recommencez la même opération sur le carré en bas et à droite du sprite.

Déplacez le curseur sur l'icône qui représente un signe plus (+) en bas et à droite de l'écran. Cliquez dessus cinq fois. Comme vous pouvez vous en douter, cet icône permet de choisir un motif de remplissage. Positionnez le curseur sur le carré en haut et à gauche du sprite et cliquez à gauche. Le carré est rempli avec le motif sélectionné. Recommencez pour remplir le carré en bas et à droite du sprite.

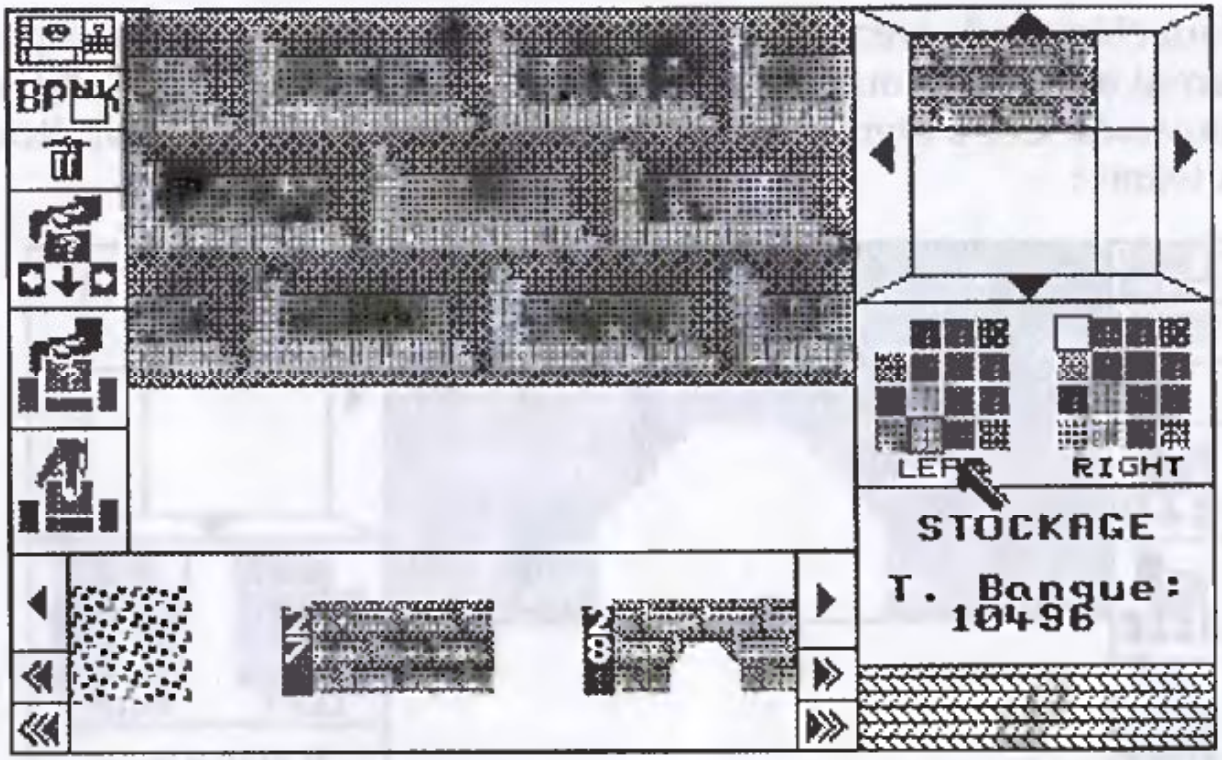
Le sprite doit maintenant avoir l'allure suivante :

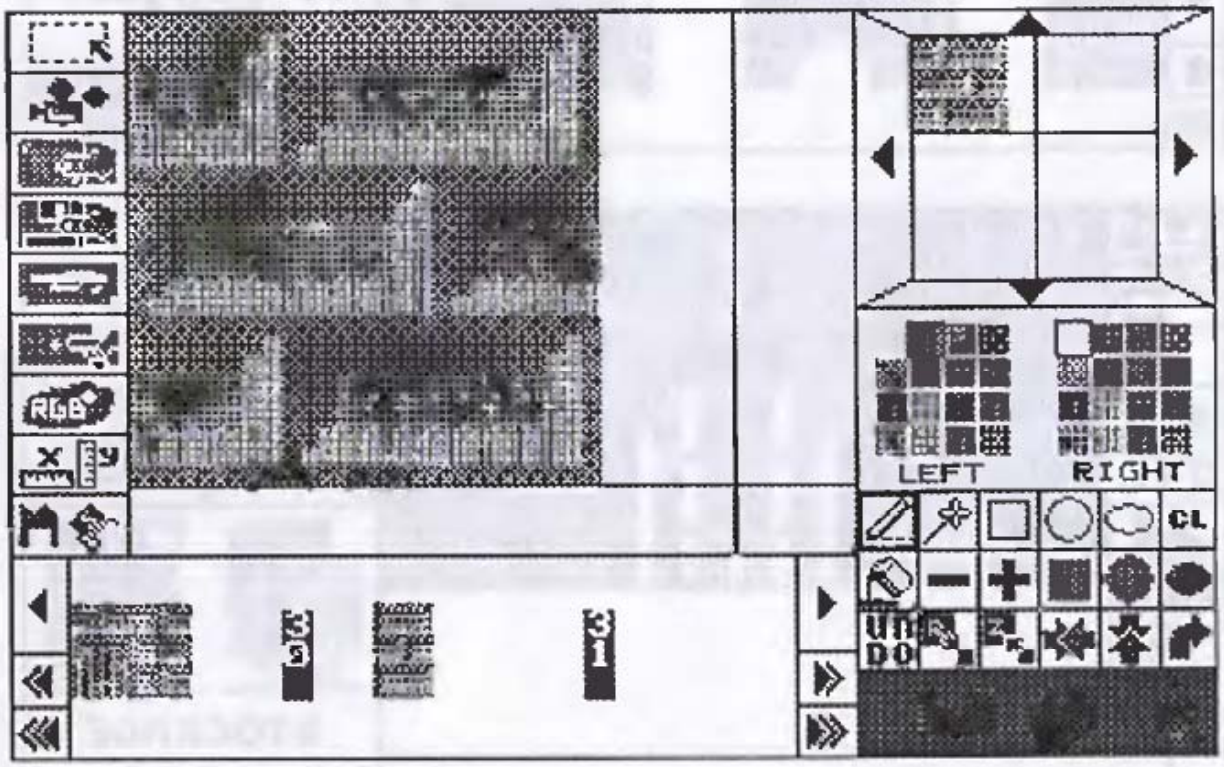


Sauvegardez-le dans la fenêtre du bas de l'écran selon le procédé habituel.

Choisissez les motifs de remplissage appropriés pour créer les deux autres sprites en utilisant la méthode qui vient d'être décrite.

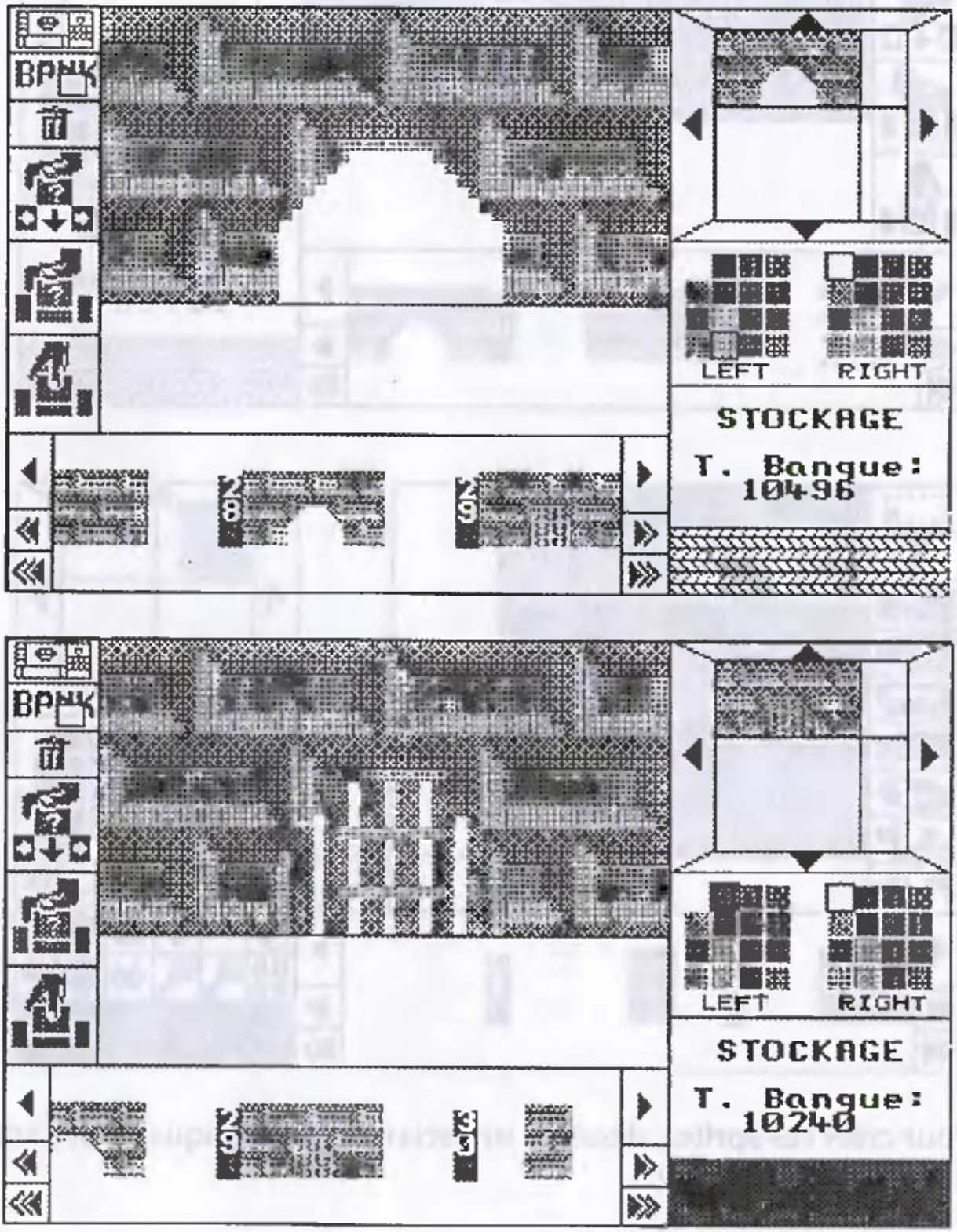
Les pièces affichées sur l'écran seront délimitées par un mur dont voici les deux motifs principaux :





Pour créer ces sprites, dessinez un rectangle, et dupliquez-le autant de fois que nécessaire.

Pour changer de pièce, la souris devra passer par des ouvertures qui seront accessibles ou inaccessibles en fonction du nombre d'objets ramassés. Ces ouvertures seront représentées par les deux sprites suivants :



Lorsque tous les sprites ont été définis, sauvegardez-les sur disque. Pour cela, cliquez sur le cinquième icône du menu principal, puis sur le sixième icône du nouveau menu, et entrez le nom du fichier de sauvegarde. Dans la suite du livre, nous ferons référence à ce fichier sous le nom SPRITE.MBK.

Récapitulatif

Vous savez maintenant utiliser quelques-uns des icônes de la partie basse de l'écran, en particulier :

- ✓ l'icône "CL" efface l'écran après confirmation,
- ✓ l'icône "carré vide" permet de tracer des rectangles vides de toutes dimensions,
- ✓ l'icône "pot de peinture" permet de remplir une surface délimitée par un contour fermé,
- ✓ l'icône "+" (plus) sélectionne le prochain motif de remplissage.
- ✓ L'icône "-" (moins) sélectionne le motif de remplissage précédent,
- ✓ l'icône "double flèche droite/gauche" permet d'obtenir le symétrique du sprite courant selon un axe vertical. Par extension, l'icône "double flèche haut/bas" permet d'obtenir le symétrique du sprite courant selon un axe horizontal, et le sprite "flèche bas vers droite" fait une rotation du sprite de 90 degrés.

Conclusion

Vous connaissez maintenant la plupart des commandes de l'éditeur de sprite. Nous allons terminer ce chapitre en vous donnant des éléments sur les commandes qui n'ont pas encore été utilisées.

Tout d'abord en ce qui concerne les icônes qui se trouvent sur le côté gauche de l'écran.

Le troisième icône en partant du haut vous permet de saisir des sprites à partir d'une disquette. Il donne accès à une nouvelle série d'icônes dont les plus importants sont les suivants :

- ✓ L'icône "pince" vous permet de charger une image au format Degas ou Néochrome et d'affecter une partie de cette image au sprite courant.
- ✓ L'icône INSERER (le quatrième) effectue une sauvegarde automatique du nouveau sprite dans la fenêtre du bas de l'écran.
- ✓ Le dernier icône est une bascule qui indique si la palette de couleurs doit ou ne doit pas être chargée en même temps que l'image.

Revenons au menu principal. Le quatrième icône en partant du haut vous permet de créer un sprite à partir d'un fichier quelconque, qui bien sûr contient des données graphiques. Ce fichier peut être de type exécutable, image, ou encore d'un tout autre type.

Le cinquième icône du menu principal permet de :

- ✓ charger une banque,
- ✓ créer une banque,
- ✓ retourner au STOS sans sauvegarder les sprites en mémoire (QUIT),
- ✓ retourner au STOS en sauvegardant les sprites en mémoire (QUIT & GRAB).

Le sixième icône permet de définir la position du point chaud (reportez-vous au chapitre 2 de la troisième partie) pour avoir plus de détail à ce sujet.

Enfin, l'icône "x y" permet de définir la taille du sprite. La taille horizontale X peut varier entre 16 et 64, et la taille verticale Y entre 2 et 64.

Concernant les icônes situés en bas de l'écran, signalons la présence :

- ✓ d'un réducteur (icône Z) et d'un agrandisseur (icône R) qui permettent respectivement de réduire et d'augmenter la taille du sprite courant,
- ✓ de la touche "UNDO" qui a la même fonction que la touche <Undo> du clavier.

2.2. L'éditeur de caractères

L'éditeur de caractères est un outil qui vous permet de créer votre propre jeu de caractères ou de modifier un jeu de caractères existant.

Pour accéder à l'éditeur de caractères, il vous faut le charger en mémoire. Comme pour l'éditeur de sprites, vous pouvez le charger en tant que programme, ou en tant qu'accessoire. Son nom est "FONTS.ACB", et il se trouve sur la disquette Accessoires. Insérez la disquette Accessoires dans le lecteur A:.

Si vous désirez charger l'éditeur en tant que programme, tapez en mode direct :

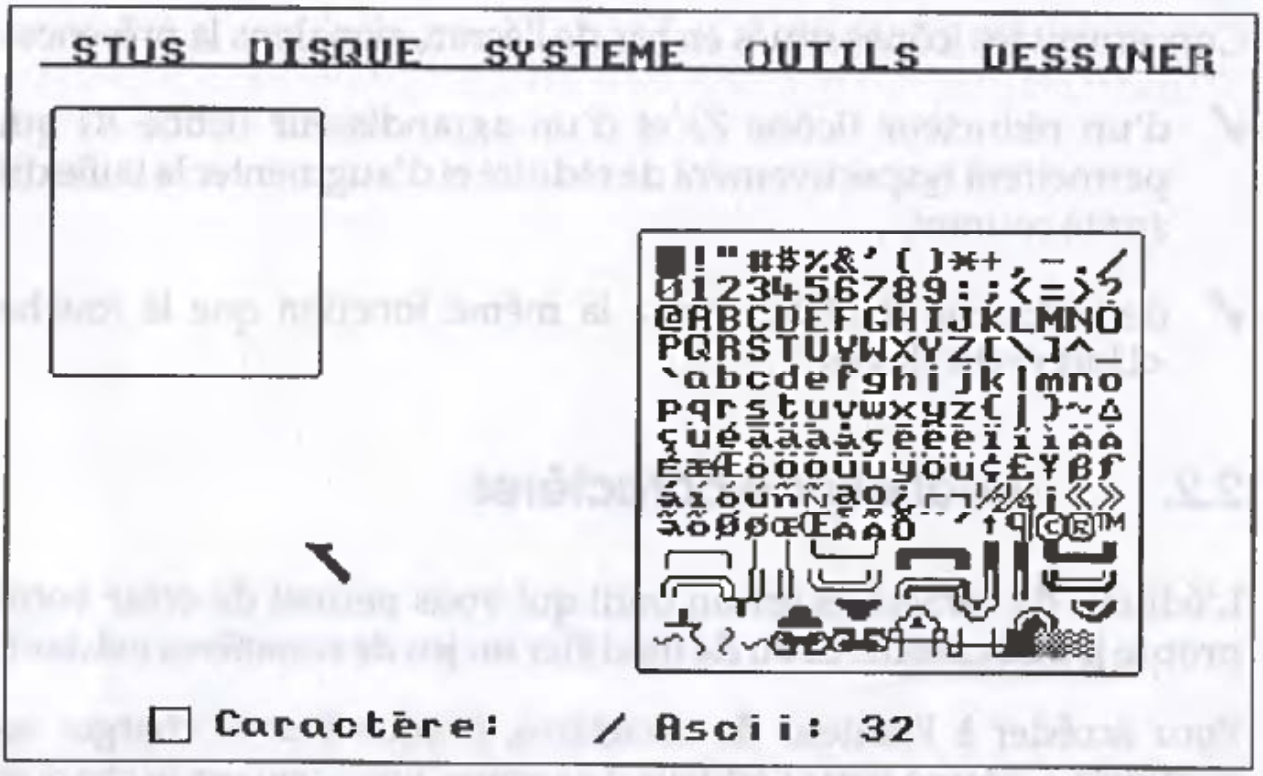
```
load "fonts.acb"<CR>
run<CR>
```

Si vous désirez le charger en tant qu'accessoire, tapez en mode direct :

```
accload"fonts"<CR>
```

et appuyez sur les touches <Help> puis <F1> (ou <F2> si sprite est toujours en F1).

L'écran doit maintenant se présenter comme suit :



Seul le menu disque présente un réel intérêt. Il vous permet de charger un jeu de caractères ou de sauvegarder le jeu de caractères en mémoire.

Le jeu de caractères affiché dans la fenêtre à droite de l'écran est le jeu standard. Pour accéder à un autre jeu de caractères, sélectionnez le menu DISQUE avec la souris et l'option CHARGER UN JEU dans ce menu. Sur la disquette Accessoires, vous disposez de trois jeux de caractères de nom FONT1.MBK, FONT2.MBK et FONT3.MBK. Chargez (par exemple) le jeu FONT1.MBK. Les caractères du nouveau jeu apparaissent dans la fenêtre à droite de l'écran. Pour modifier un des caractères de ce jeu, pointez-le avec la souris et cliquez. Reportez-vous dans la fenêtre à gauche de l'écran pour effectuer vos modifications. Le bouton gauche de la souris "allume" le point qui se trouve sous le curseur de la souris. Le bouton droit de la souris "éteint" le point qui se trouve sous le curseur de la souris. Lorsque vous avez effectué vos modifications, positionnez le curseur de la souris sur le caractère modifié dans la fenêtre de droite et cliquez à droite. Le caractère est recopié. Répétez les étapes qui viennent d'être décrites pour modifier tous les caractères qui ne sont pas à

votre goût, et sauvegardez le nouveau jeu de caractères sur disque. L'option SAUVER LE JEU du menu DISQUE sauvegarde le jeu en mémoire dans le fichier d'où il est issu. L'option SAUVER COMME du même menu vous demande d'entrer le nom du nouveau jeu de caractères.

2.3. L'éditeur d'icônes

L'éditeur d'icônes est très similaire à l'éditeur de caractères. Il permet de créer ou de modifier un jeu d'icônes.

L'éditeur d'icônes se trouve sur la disquette Accessoires sous le nom ICON.ACB. Il peut être exécuté en tant que programme ou en tant qu'accessoire. Pour vous faire une idée de ce que sont les icônes, vous pouvez charger le fichier ICON.MBK de la disquette Accessoires en sélectionnant l'option CHARGER UNE BANQUE du menu BANQUE.

Pour afficher un icône dans un programme STOS, il vous faut charger le fichier d'icônes avec une instruction du type :

```
load "icon.mbk"<CR>
```

L'affichage du Nième icône est obtenu à l'aide de l'instruction :

```
print icon$(N)<CR>
```

2.4. L'éditeur de musique

L'éditeur de musique est un puissant outil qui permet d'exploiter les ressources du générateur sonore de l'ATARI.

Son but premier est l'écriture de morceaux sur une, deux ou trois voies. Ses concepteurs l'ont doté :

✓ d'un générateur d'enveloppes qui permet de moduler le volume des notes émises,

✓ d'un générateur de trémolos qui permet de moduler la fréquence des notes émises.

Les morceaux ainsi créés peuvent bien évidemment être écoutés lorsque le générateur sonore est en fonction, mais ils peuvent également se superposer à l'exécution d'un programme STOS, ce qui en fait tout leur intérêt.

Pour accéder à l'éditeur de musique, il vous faut le charger en mémoire. Comme pour l'éditeur de sprites, vous pouvez le charger en tant que programme, ou en tant qu'accessoire. Son nom est "MUSIC.ACB", et il se trouve sur la disquette Accessoires. Insérez la disquette Accessoires dans le lecteur A:. Si vous désirez charger l'éditeur en tant que programme, tapez en mode direct :

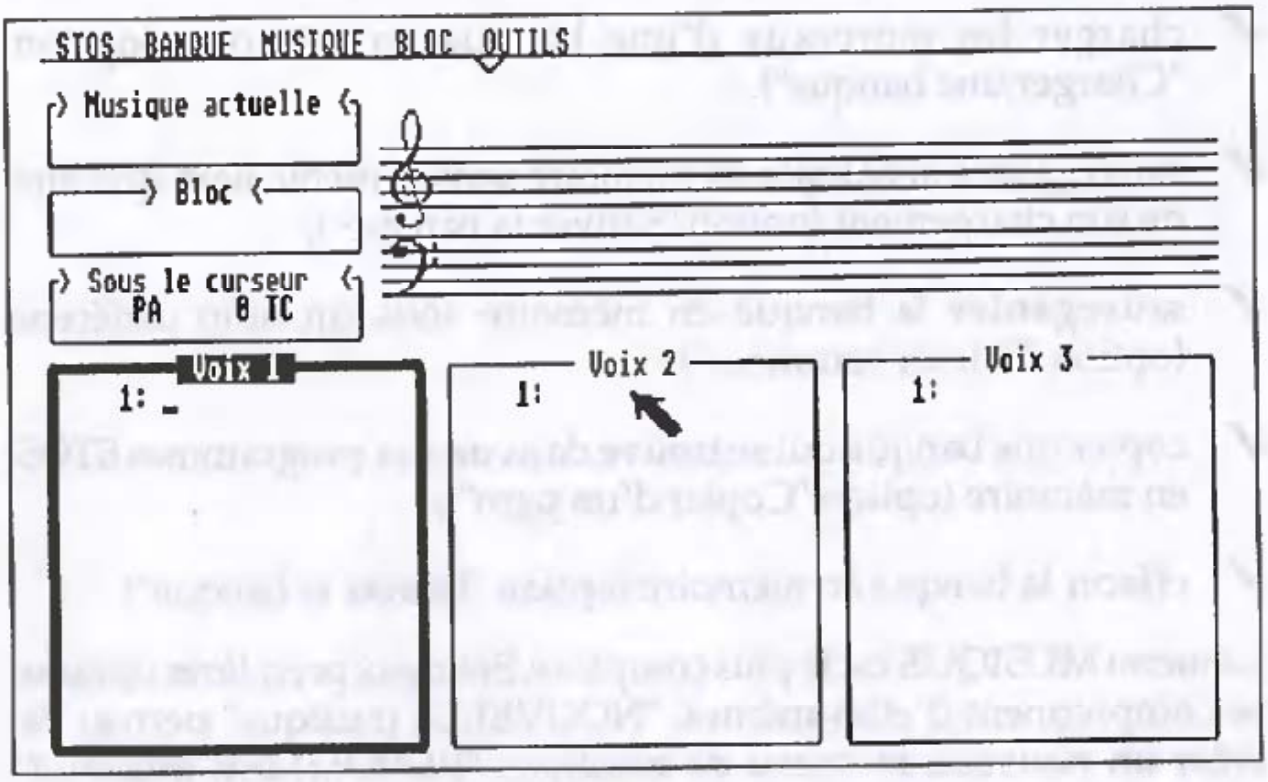
```
load "music.acb"<CR>
run<CR>
```

Si vous désirez le charger en tant qu'accessoire, tapez en mode direct :

```
accload "music"<CR>
```

et appuyez sur les touches <Help> et <F1>.

L'écran doit maintenant se présenter comme suit :



Ne vous affolez pas !

La complexité de cet écran n'est qu'apparente ...

La partie haute de l'écran laisse apparaître cinq menus. Pour sélectionner une option dans un des menus, amenez le curseur de la souris sur le menu de votre choix et sélectionnez l'option qui vous intéresse en cliquant lorsqu'elle apparaît en inverse vidéo.

Le menu STOS :

- ✓ présente l'accessoire (option "A propos de MUSIC"),
- ✓ permet de retourner au BASIC sans sauvegarder la banque en mémoire (option "Quitter"),
- ✓ permet de retourner au BASIC en sauvegardant la banque en mémoire (option "Quitter/récupérer).

Le menu BANQUE permet de :

- ✓ charger les morceaux d'une banque en mémoire (option "Charger une banque").
- ✓ sauvegarder la banque en mémoire sous le même nom que lors de son chargement (option "Sauver la banque"),
- ✓ sauvegarder la banque en mémoire sous un nom différent (option "Sauver comme ..."),
- ✓ copier une banque qui se trouve dans un des programmes STOS en mémoire (option "Copier d'un pgm"),
- ✓ effacer la banque en mémoire (option "Effacer la banque").

Le menu MUSIQUE est le plus complexe. Ses deux premières options se comprennent d'elles-mêmes. "NOUVELLE musique" permet de créer un nouveau morceau de musique. "RENOMMER musique" permet de modifier le nom du morceau en mémoire.

Avant d'expliciter les autres options, il convient de marquer une pause pour bien comprendre un détail fondamental.

Une banque de musique peut contenir PLUSIEURS morceaux, mais UN SEUL morceau peut être édité à la fois. Nous appellerons "morceau courant" le morceau en cours d'édition.

Revenons aux options du menu MUSIQUE.

L'option "STOCKER musique" copie le morceau courant dans la banque.

L'option "PRENDRE musique" copie un des morceaux de la banque dans l'éditeur. Ce morceau devient le morceau courant et ses notes apparaissent dans les fenêtres Voix 1, Voix 2 et Voix 3.

L'option "AJOUTER musique" ajoute le morceau spécifié dans la banque à la suite du morceau courant.

L'option "EFFACER musique" efface un des morceaux de la banque.

Comme vous vous en doutez, l'option "JOUER musique" joue le morceau courant. Attention, les éventuelles modifications apportées ne seront prises en compte que si vous avez sélectionné préalablement l'option "STOCKER musique".

L'option "STOCKER et JOUER" combine les actions des options "STOCKER musique" et "JOUER musique". Nous vous conseillons de l'utiliser systématiquement à la place des deux options équivalentes.

Enfin, l'option "IMPRIMER musique" envoie le morceau courant vers l'imprimante. Les données sont imprimées sous la forme suivante :

```
**************************************************************
*                    Nom du morceau                          *
**************************************************************
1 : ENVEL 2          1 : ENVEL 2          1 : ENVEL 2
2 : REPETER 0 3      2 : REPETER 0 3      2 : REPETER 0 3
3 : LA 0 SC          3 : SOL# 2 SC        3 : MI 2 SC
etc..
```

Chaque colonne représente une des voies sonores.

Les options du menu BLOC se comprennent d'elles-mêmes, surtout si vous avez déjà utilisé un traitement de textes. Pour définir un bloc de notes, amenez la flèche du curseur sur la première note du bloc. Cliquez dessus. Déroulez le menu BLOC et sélectionnez l'option "Debut bloc". Recommencez la même opération pour définir la fin du bloc en cliquant cette fois ci sur "Fin bloc". Le bloc ainsi sélectionné apparaît en inverse vidéo sur l'écran. Il peut être désélectionné, copié, effacé ou transposé (respectivement options "Annuler bloc", "Copier bloc", "Effacer bloc" et "Transposer bloc").

Enfin, le menu OUTILS permet de :

✓ définir des enveloppes (option "Fixer enveloppes"),

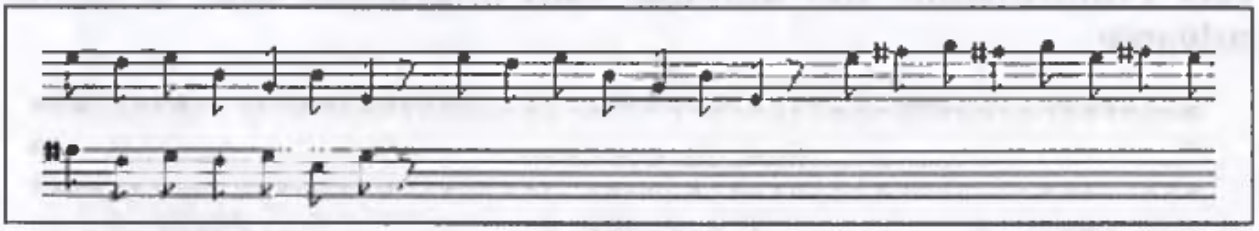
✓ définir des trémolos (option "Fixer trémolos"),

✓ effacer les enveloppes et trémolos définis (option "Effacer env/trem").

Nous y reviendrons en détail dans la suite du chapitre.

Votre premier morceau

Vous connaissez certainement la musique "Pop Corn". Nous allons nous en servir pour comprendre les mécanismes de l'éditeur de musique. Cette musique correspond à la partition suivante, sur une voie :



Pour ceux qui ne connaissent pas la musique, les notes de cette partition sont les suivantes, en supposant que l'octave centrale est la troisième :

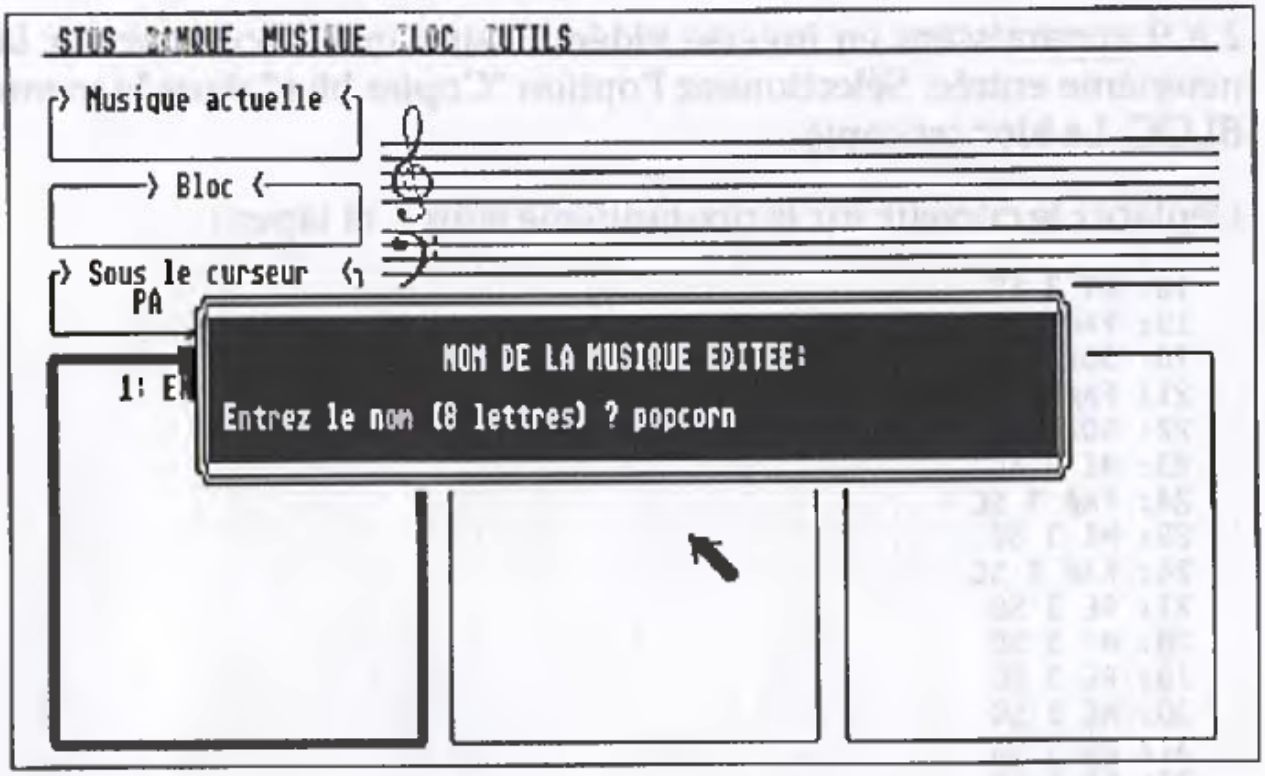
```
MI3 RE3 MI3 SI2 SOL2 SI2 MI2 SILENCE
MI3 RE3 MI3 SI2 SOL2 SI2 MI2 SILENCE
MI3 FA#3 SOL3 FA#3 SOL3 MI3 FA#3
MI3 FA#3 RE3 MI3 RE3 MI3 D03 MI3 SILENCE
```

Avant d'entrer les notes dans l'éditeur musical, nous allons préciser que l'enveloppe utilisée est la première.

Pour cela, tapez :

```
1: ENVEL 1
```

dans la fenêtre correspondant à la voie No 1. Un message est affiché sur l'écran. Il vous demande d'entrer le nom du morceau à éditer. Tapez "popcorn", puis appuyez sur la touche <Return> :



Le nom du morceau en cours d'édition apparaît dans la fenêtre "Musique actuelle", et le curseur se positionne sur la seconde entrée dans la fenêtre "Voix 1".

Toutes ces notes étant des croches, vous les entrerez dans l'éditeur musical comme suit :

```
Voix 1

2: MI 3 SC
3: RE 3 SC
4: MI 3 SC
5: SI 2 SC
6: SOL 2 SC
7: SI 2 SC
8: MI 2 SC
9: PA 0 SC.
```

Les huit notes suivantes sont les mêmes. Nous allons utiliser le menu BLOC pour gagner un peu de temps.

Positionnez le curseur sur la deuxième entrée. Le contenu de cette entrée disparaît et apparaît dans la fenêtre "Sous le curseur". Sélectionnez l'option "Début bloc" dans le menu BLOC. Les entrées

2 à 9 apparaissent en inverse vidéo. Positionnez le curseur sur la neuvième entrée. Sélectionnez l'option "Copier bloc" dans le menu BLOC. Le bloc est copié.

Déplacez le curseur sur la dix-huitième entrée, et tapez :

```
18: MI 3 SC
19: FA# 3 SC
20: SOL 3 SC
21: FA# 3 SC
22: SOL 3 SC
23: MI 3 SC
24: FA# 3 SC
25: MI 3 SC
26: FA# 3 SC
27: RE 3 SC
28: MI 3 SC
29: RE 3 SC
30: MI 3 SC
31: DO 3 SC
32: MI 3 SC
33: PA 0 SC.
```

Ce morceau très simple n'était composé que de croches, mais il est également possible de définir des rondes, des blanches, des noires, des doubles croches et des triples croches. Les sigles utilisés dans l'éditeur de musique sont :

Durée	Sigle	Durée en temps
Ronde	RD	4
Blanche	BL	2
Noire	NR	1
Croche	SC	1/2
Double croche	DC	1/4
Triple croche	TC	1/8

Les notes peuvent également être pointées, c'est-à-dire augmentées de la moitié de leur durée. Pour définir une note pointée dans l'éditeur de musique, il suffit d'écrire un point décimal (.) après le sigle de sa durée :

Durée	Sigle	Durée en temps
Ronde pointée	RD.	6
Blanche pointée	BL.	3
Noire pointée	NR.	1 1/2

Croche pointée	SC.	3/4
Double croche pointée	DC.	3/8
Triple croche pointée	TC.	3/16

Toutes les notes du morceau ont été entrées. Sélectionnez l'option "STOCKER et JOUER" du menu MUSIQUE pour entendre votre oeuvre. Un message est affiché sur l'écran. Il vous demande de cliquer sur un numéro de morceau. Cliquez sur l'entrée 1. Après quelques instants, le morceau est activé.

Comme vous le constatez, il s'arrête après la dernière note. Pour le répéter de façon continue, il suffit de placer une instruction de répétition au début du morceau. Positionnez-vous sur l'entrée 2 à l'aide de la souris. Le contenu de cette entrée disparaît et apparaît dans la fenêtre "Sous le curseur". Appuyez sur la touche <Insert> du clavier de l'ATARI. Entrez REPETER 0,3 en face de cette entrée.

Enregistrez et activez le morceau de la manière habituelle en sélectionnant l'option "STOCKER et JOUER" du menu MUSIQUE. Le morceau s'exécute sans fin.

Le tempo 50 a été sélectionné par défaut. Vous pouvez le modifier en temps réel (c'est-à-dire pendant l'exécution du morceau) en utilisant les touches + et -.

De même, vous pouvez modifier la hauteur des notes jouées en temps réel en utilisant les touches / et *. A titre indicatif, une octave correspond à 12 demi-tons, et donc, le paramètre transposition doit être égal à :

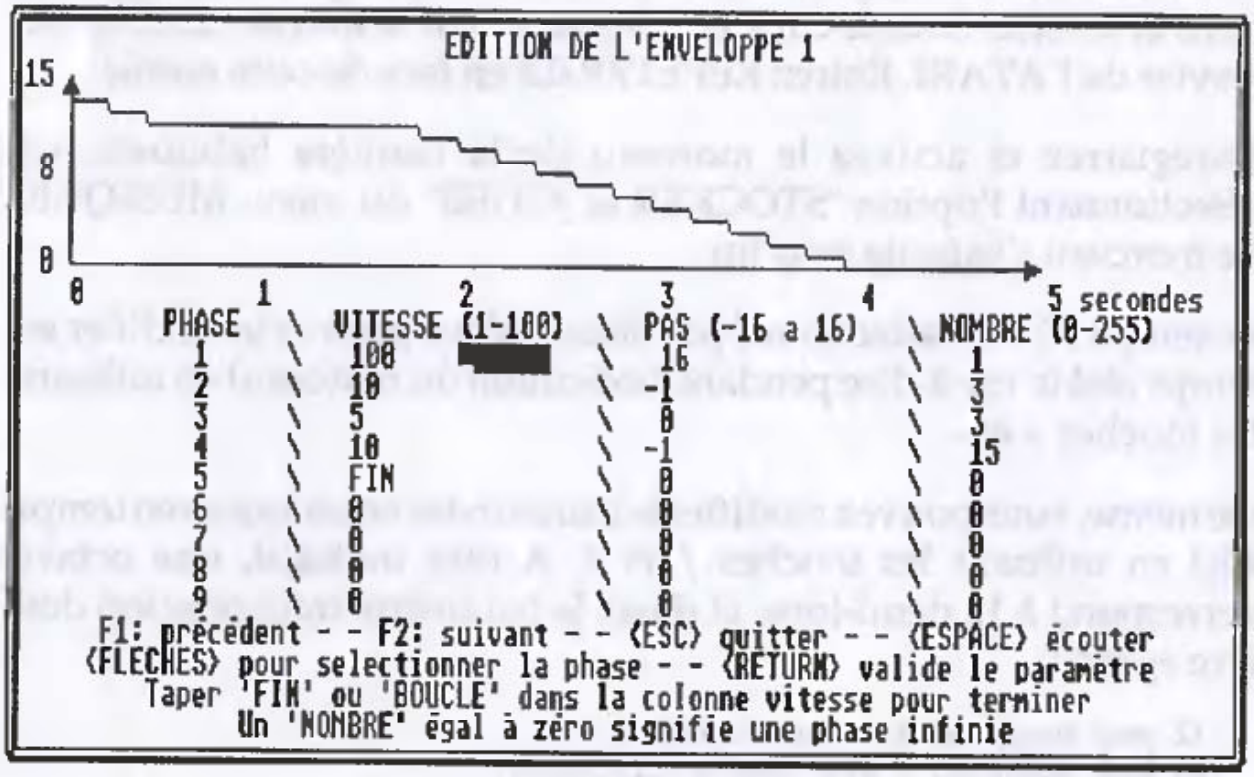
12 pour transposer à l'octave supérieure,
24 pour transposer à deux octaves supérieures,
...
-12 pour transposer à l'octave inférieure,
-24 pour transposer à deux octaves inférieures,
etc..

Appuyez sur la touche <Esc> pour revenir à l'éditeur.

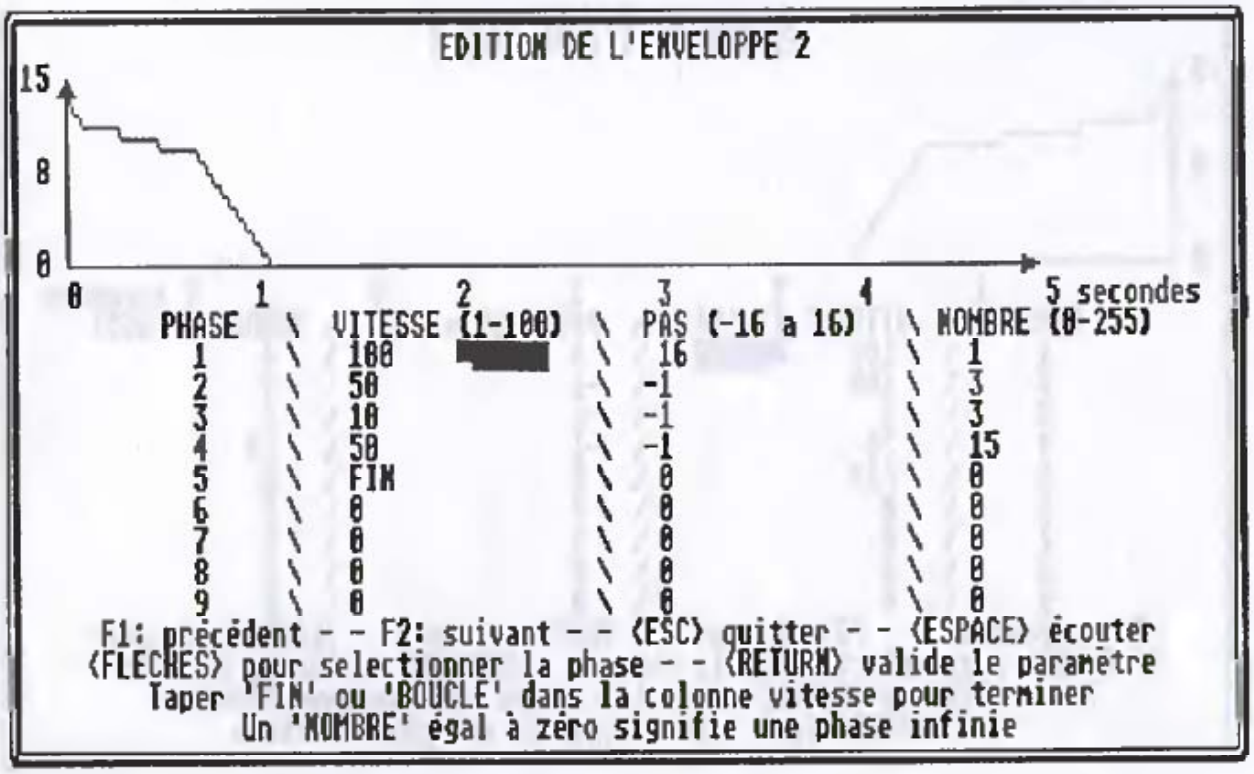
La création d'un morceau sur trois voies n'est guère plus complexe. Il suffit de répéter les opérations qui viennent d'être décrites sur les voies 2 et 3. Reportez-vous au chapitre 3 de la troisième partie du livre pour avoir un exemple concret de musique sur trois voies.

Nous allons maintenant faire quelques essais d'enveloppes et de trémolos sur ce morceau.

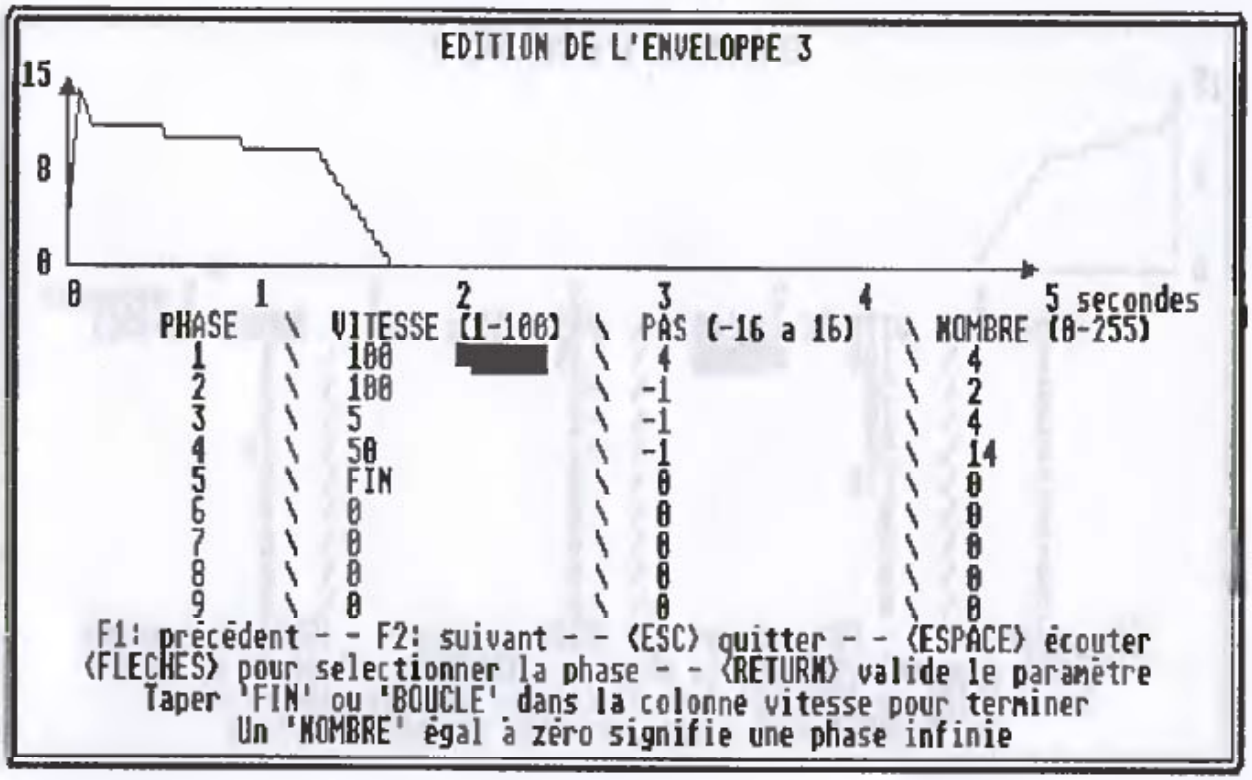
Une enveloppe est une courbe qui détermine la variation temporelle des sons émis. Elle a généralement la forme d'une cloche. L'éditeur de musique dispose de huit enveloppes prédéfinies qui ont les formes suivantes :



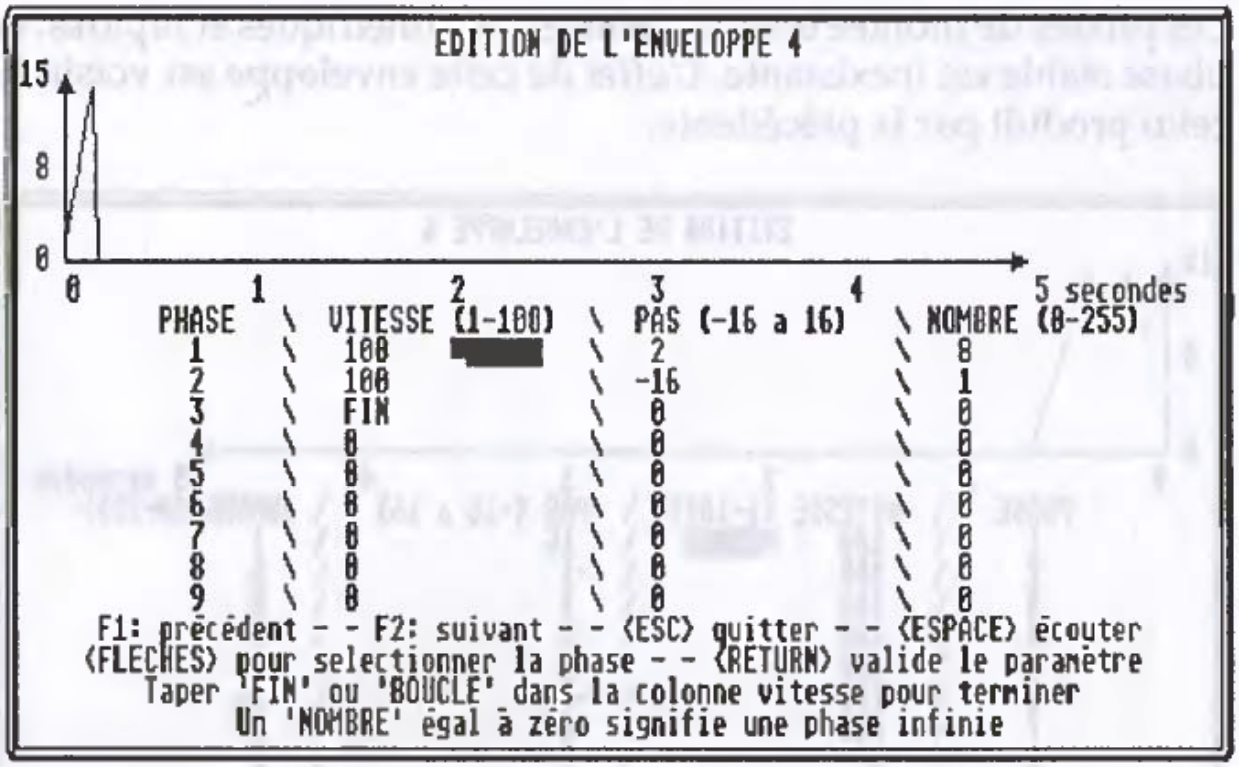
L'enveloppe 1 a un temps de montée nul, une phase stable relativement longue et un temps de descente relativement long, ce qui se traduira par un son immédiatement à son maximum de volume, de longue durée et décroissant lentement.



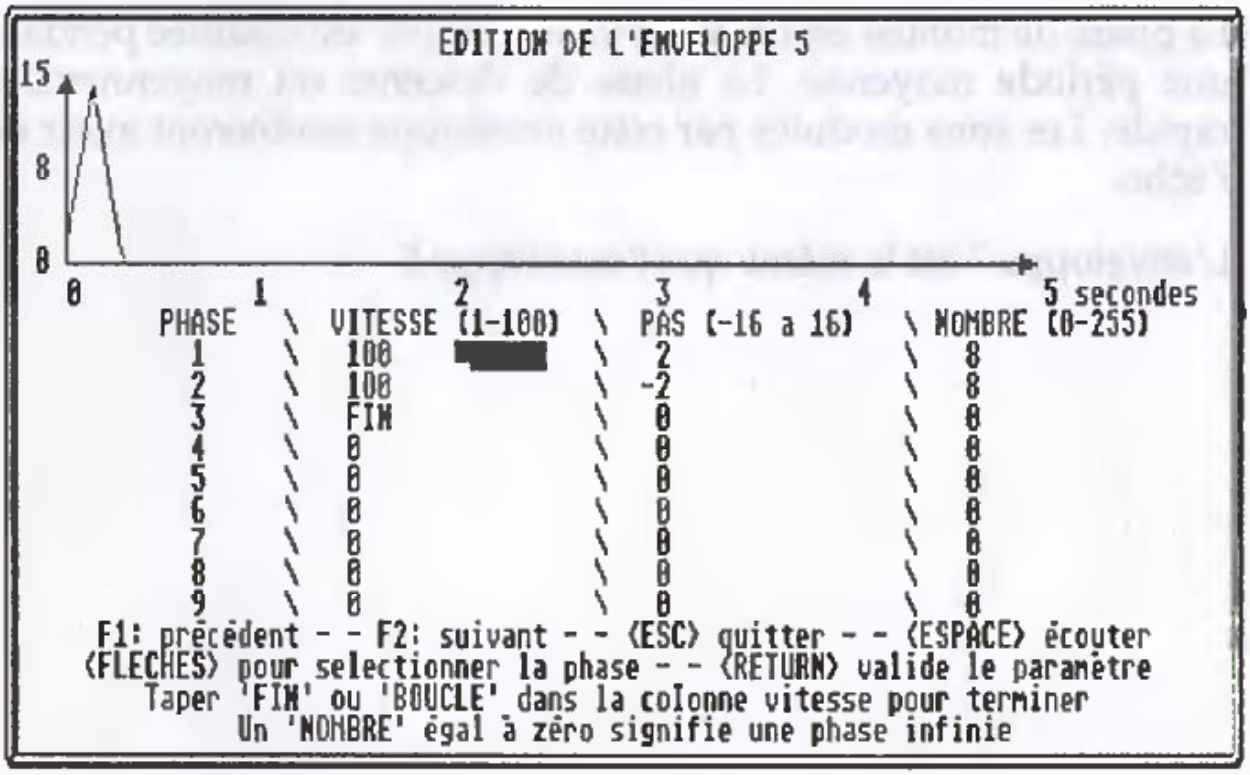
Comme l'enveloppe 1, l'enveloppe 2 a un temps de montée nul. Sa phase stable est également nulle. Elle possède trois phases de décroissance : accentuée, faible et accentuée. Les sons modulés par cette enveloppe seront immédiatement à leur maximum mais auront une décroissance bien plus rapide qu'avec l'enveloppe 1.



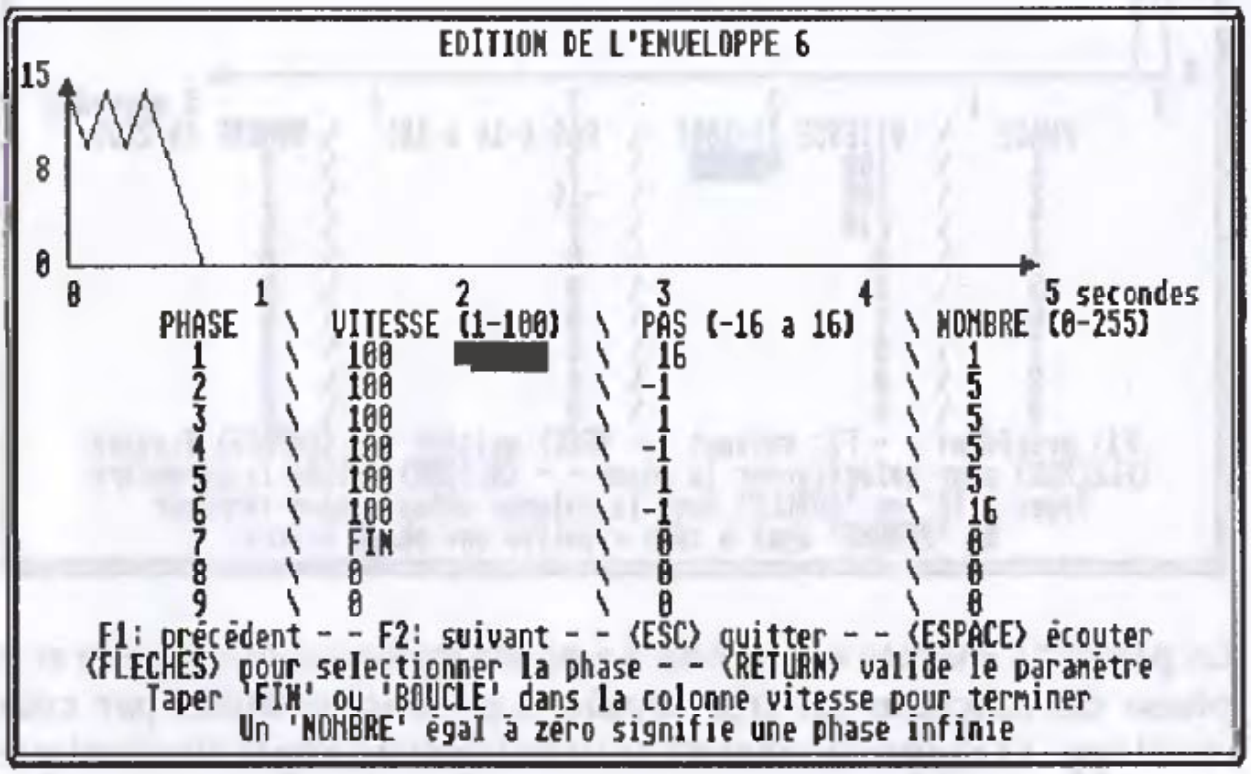
La phase de montée est rapide, mais non nulle. La phase stable est inexistante, et la phase de décroissance est rapide sur une courte durée, lente sur une durée moyenne, et rapide sur une courte durée. Les sons modulés par cette enveloppe se rapprocheront de la sonorité d'une guitare.



La phase de montée est rapide. La phase stable est inexistante et la phase de descente est très rapide. Les sons modulés par cette enveloppe se rapprocheront de ceux émis par un orgue électronique.

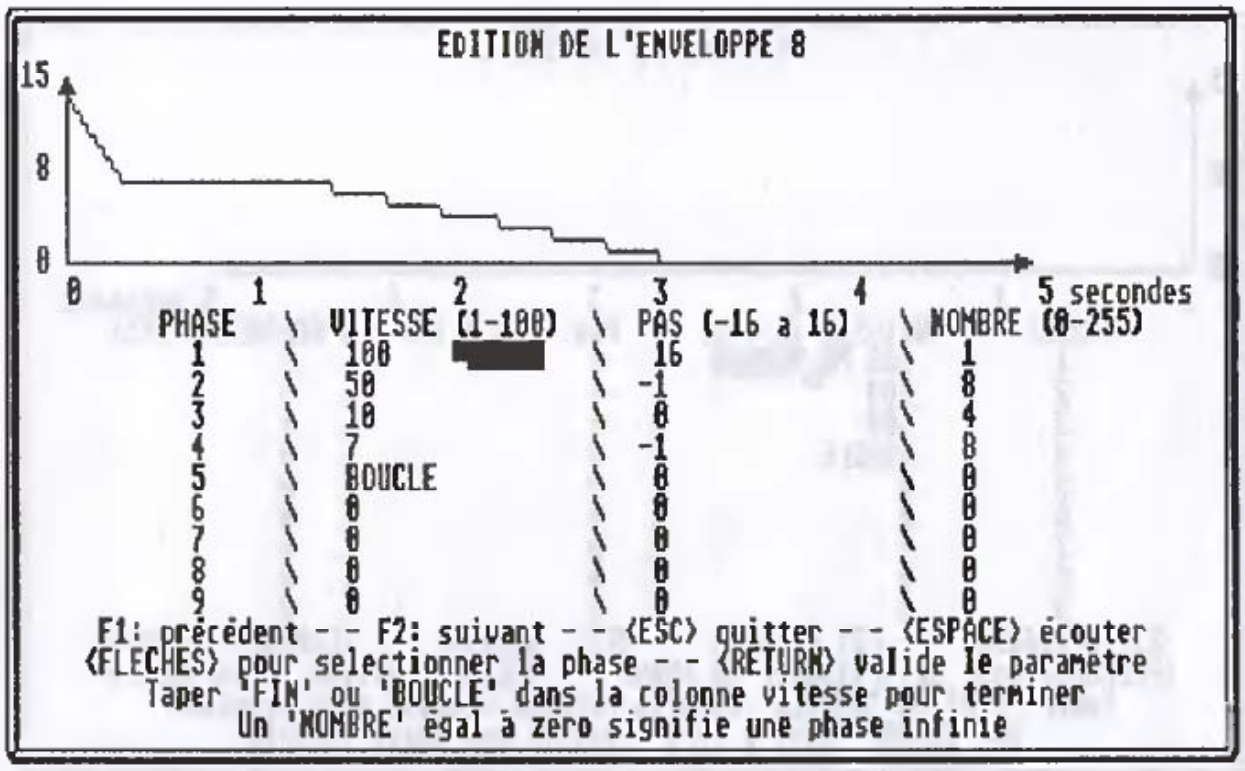


Les phases de montée et de descente sont symétriques et rapides. La phase stable est inexistante. L'effet de cette enveloppe est voisin de celui produit par la précédente.



La phase de montée est nulle. La phase "stable" est ondulée pendant une période moyenne. La phase de descente est moyennement rapide. Les sons modulés par cette enveloppe sembleront avoir de l'écho.

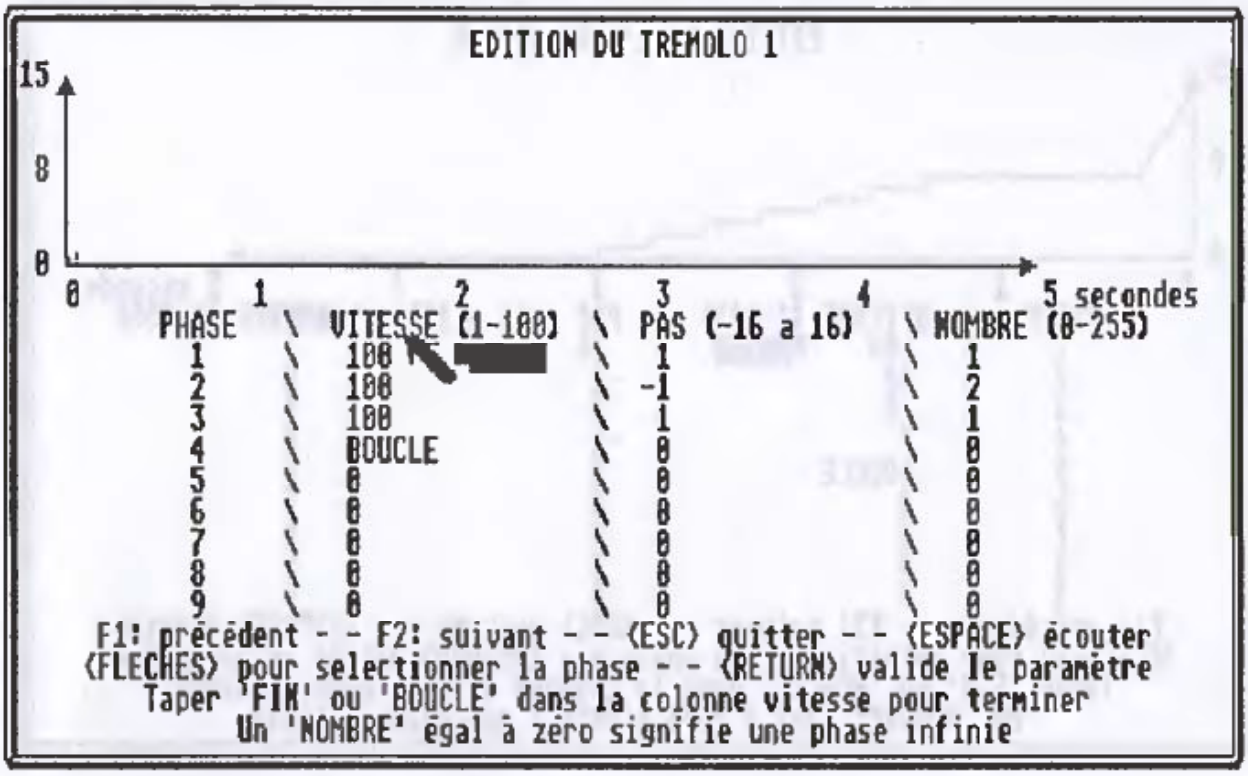
L'enveloppe 7 est la même que l'enveloppe 5.



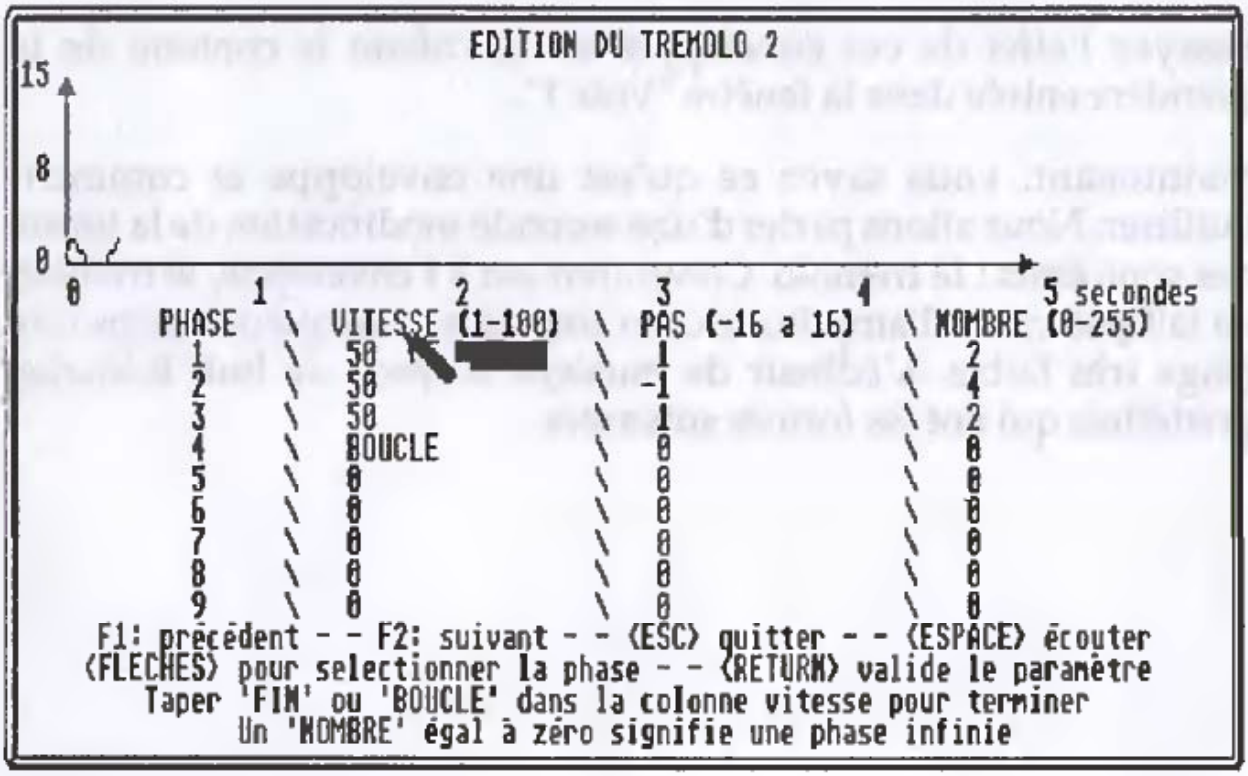
Cette enveloppe est très comparable à la première, mais avec un volume moyen inférieur. Le temps de montée est nul. La phase stable est relativement longue et le temps de descente est long.

Essayez l'effet de ces enveloppes en modifiant le contenu de la première entrée dans la fenêtre "Voix 1".

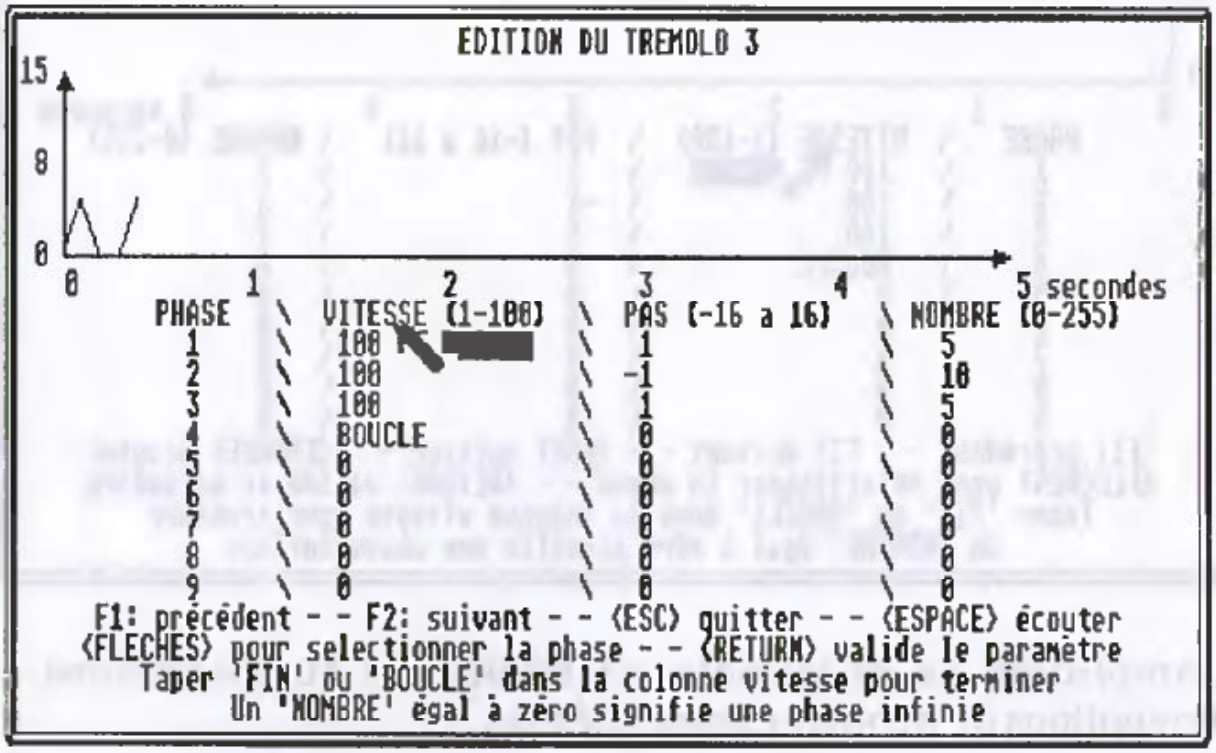
Maintenant, vous savez ce qu'est une enveloppe et comment l'utiliser. Nous allons parler d'une seconde modification de la forme des sons émis : le trémolo. Contrairement à l'enveloppe, le trémolo ne fait pas varier l'amplitude d'un son mais sa fréquence, dans une plage très faible. L'éditeur de musique dispose de huit trémolos prédéfinis qui ont les formes suivantes :



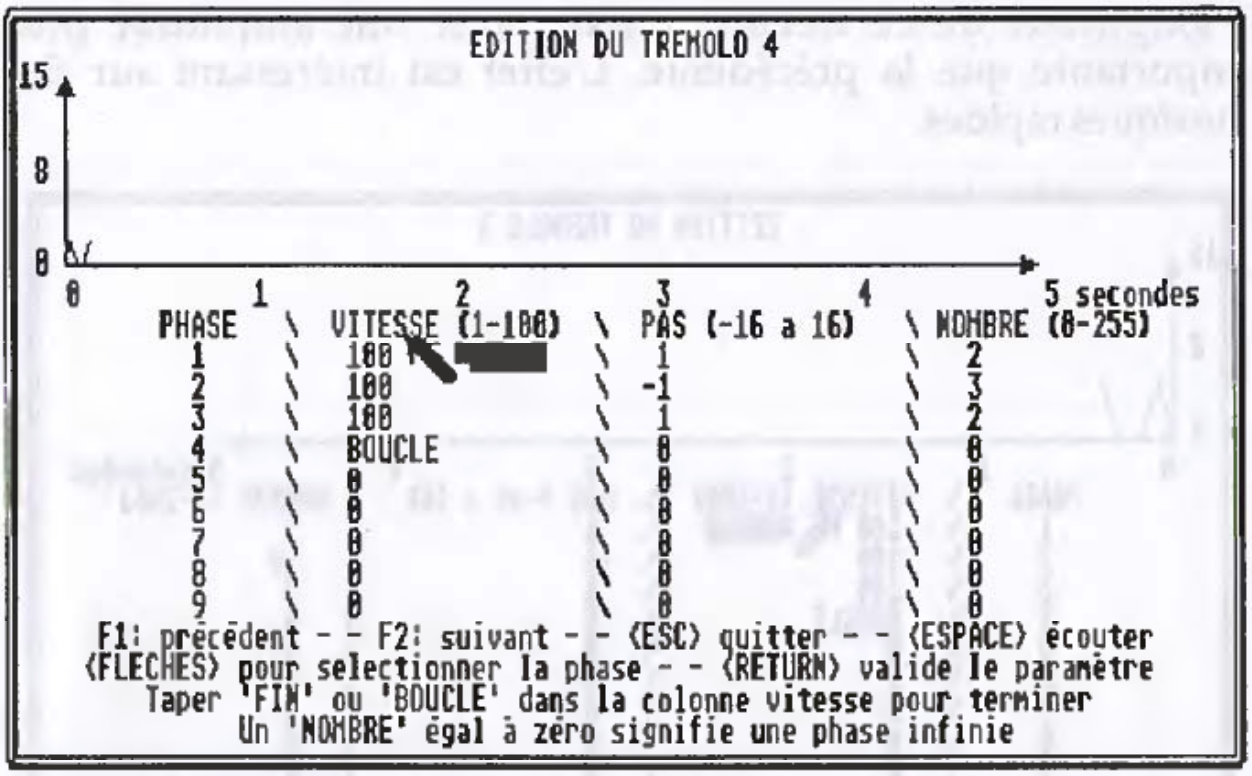
L'amplitude de ce trémolo ondule faiblement. L'effet obtenu est agréable.



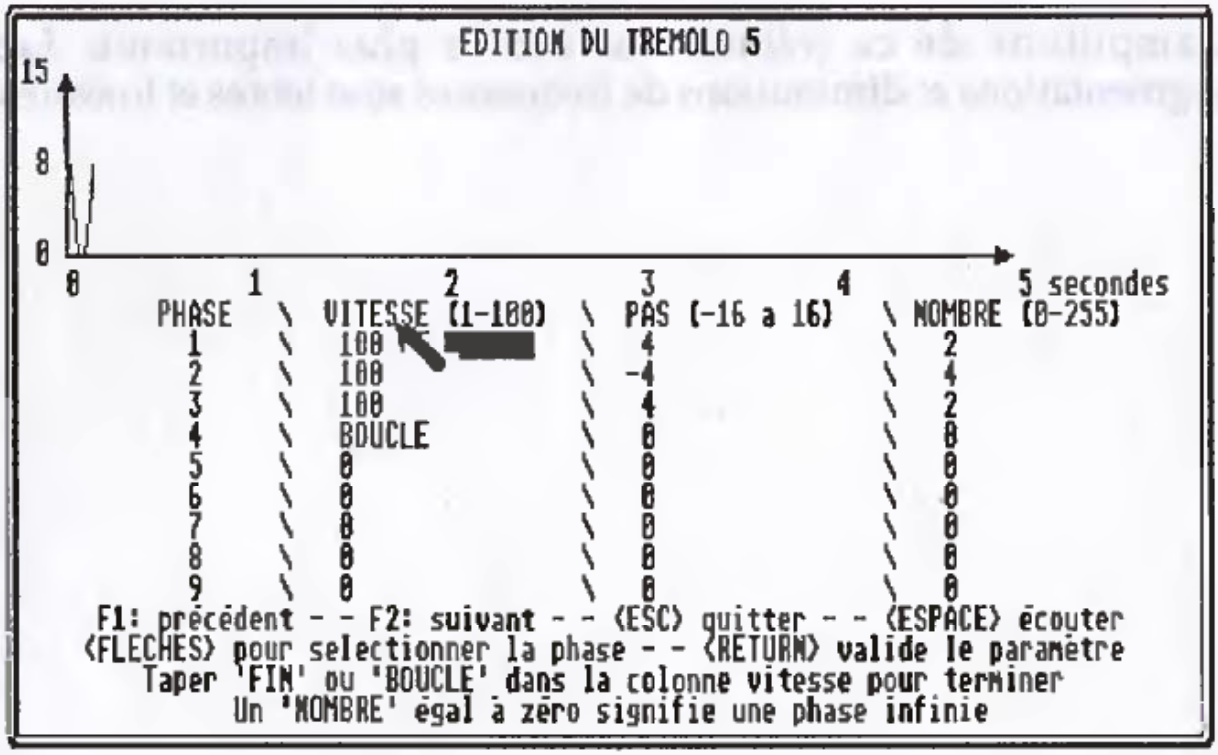
L'amplitude de ce trémolo ondule avec une amplitude plus importante que la précédente. L'effet est intéressant sur des musiques rapides.



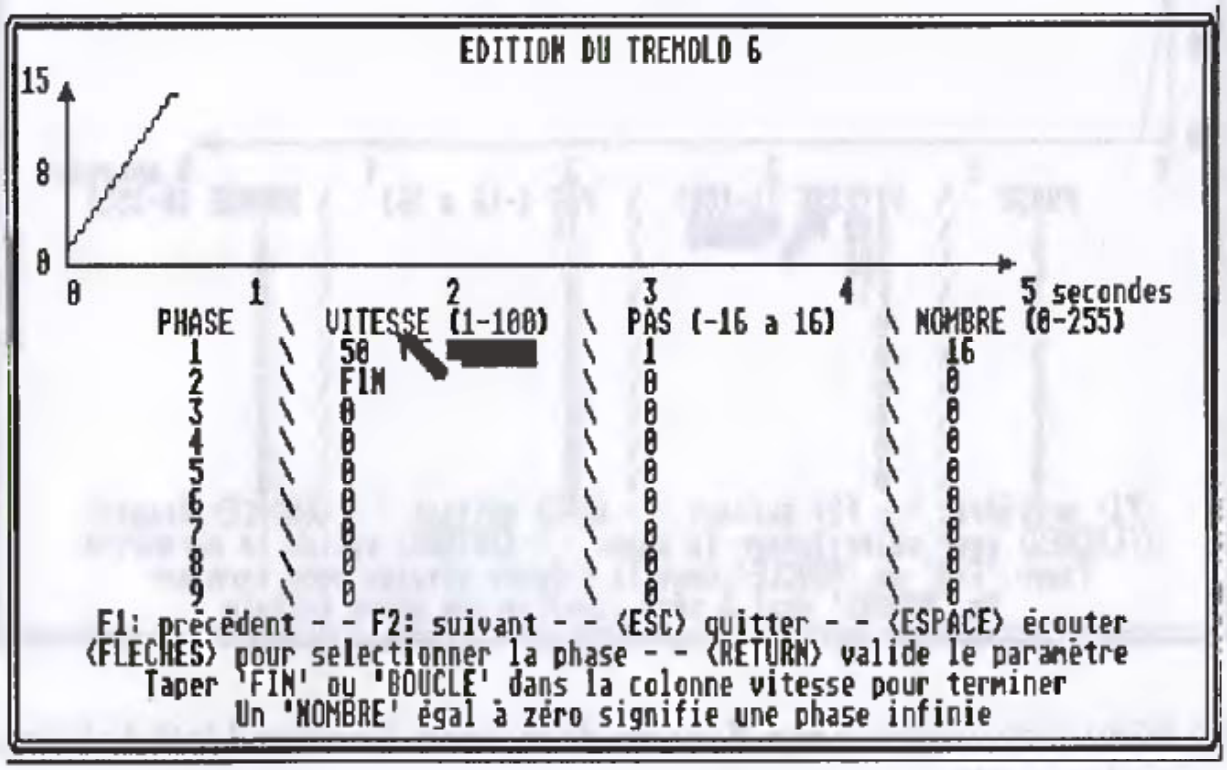
L'amplitude de ce trémolo est encore plus importante. Les augmentations et diminutions de fréquences sont lentes et linéaires.



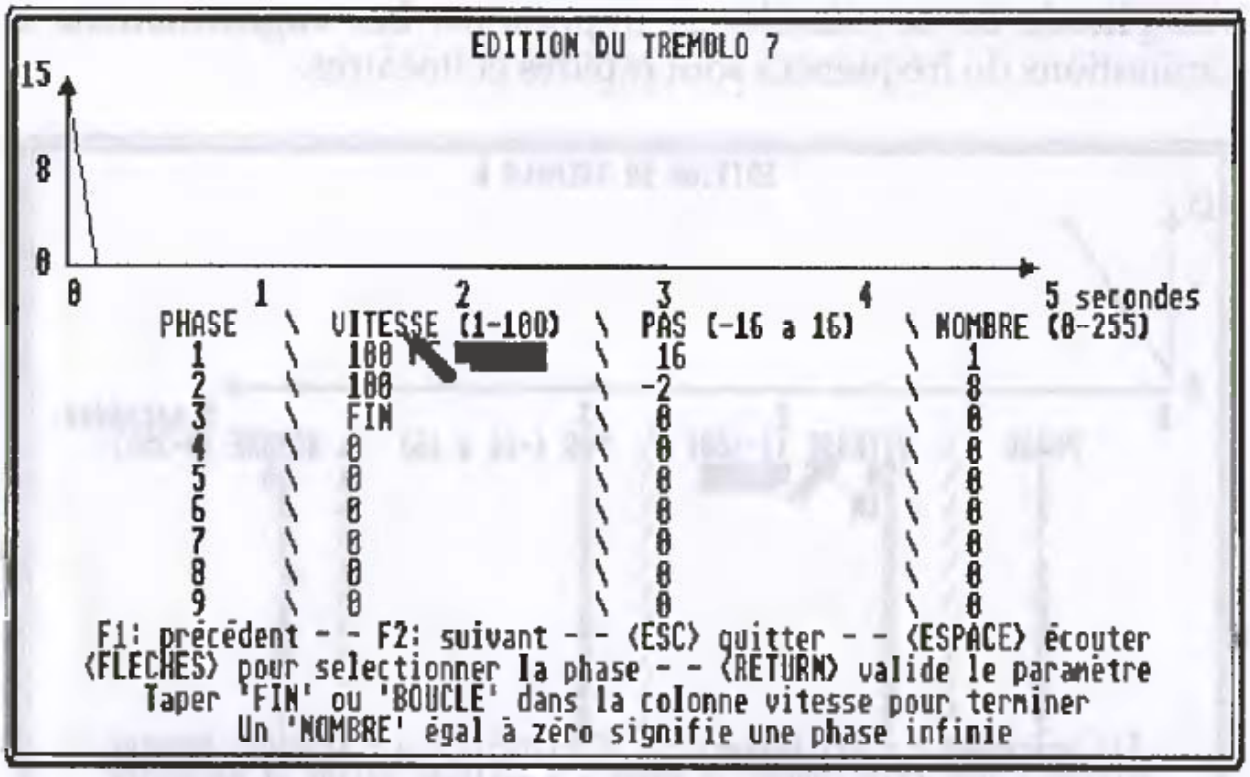
L'amplitude de ce trémolo est faible. Les augmentations et diminutions de fréquences sont linéaires.



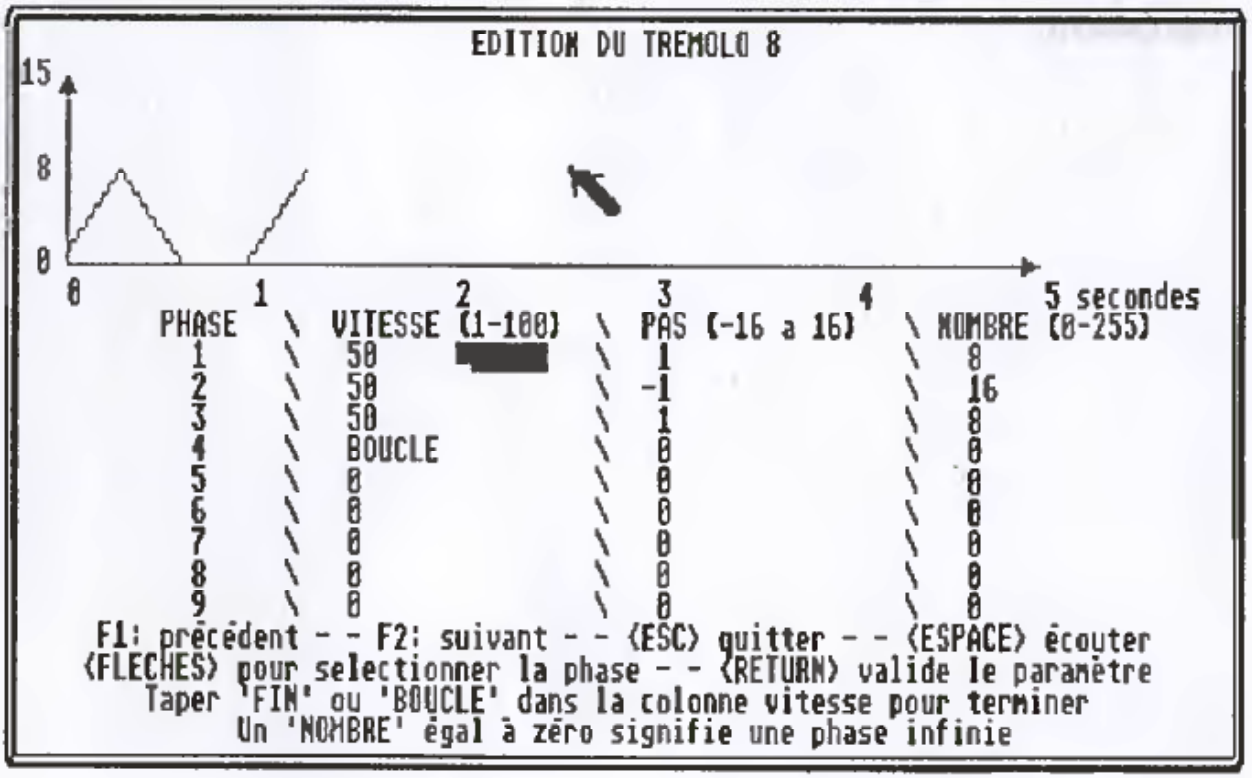
L'amplitude de ce trémolo est importante. Les augmentations et diminutions de fréquences sont rapides et linéaires.



La fréquence augmente lentement par escaliers. L'effet obtenu est intéressant.



La fréquence diminue rapidement de manière linéaire. L'effet obtenu est intéressant.



La fréquence suit une dent de scie tronquée. Les sons modulés par ce trémolo font penser à l'écho d'un sonar dans un sous marin.

Essayez l'effet de ces trémolos en insérant une ligne entre la définition de l'enveloppe et l'ordre de répétition dans la fenêtre Voix 1. Par exemple :

```
1: ENVEL 1
2: TREMOLO 3
3: REPETER 0,3
etc..
```

Signalons encore la commande VOLUME qui permet de régler le volume d'une voie sonore. lille demande un paramètre compris entre 1 et 15. Par exemple :

```
1: VOLUME 12
```

Si vous désirez sauvegarder le morceau que vous venez d'entrer, sélectionnez l'option "Sauver la banque" dans le menu BANQUE. Entrez le nom de la banque dans la fenêtre affichée. Par exemple POPCORN.MBK et appuyez sur la touche <Return>.

Résumé

Les options de menus les plus utilisées sont les suivantes :

Pour charger une banque musicale en mémoire :

✓ BANQUE/Charger une banque

Pour avoir accès à un des morceaux de la banque :

✓ MUSIQUE/PRENDRE musique/No de morceau

Pour activer un morceau après modification :

✓ MUSIQUE/STOCKER et JOUER

Les commandes de l'éditeur de musique sont :

✓ VOLUME <N> pour définir le volume d'une voie,

✓ REPETER <N>,<Pos> pour répéter une partie ou la totalité d'un morceau. <N> est le nombre de répétitions (0 pour une répétition infinie), <Pos> est la position dans la voie de la première note à répéter.

✓ ENVEL <N> pour activer une enveloppe,

✓ TREMOLO <N> pour activer un trémolo.

2.5. Le compacteur d'écran

Le compacteur d'écran est un accessoire de nom "COMPACT.ACB" qui se trouve sur la disquette Accessoires. Il permet de compacter toute image de format Néochrome ou Degas.

Pour compacter une image, il vous suffit :

✓ d'exécuter COMPACT.ACB (en tant que programme ou en tant qu'accessoire),

✓ de sélectionner une des options du menu disque :
- Charger IMAGE.NEO pour une image Néochrome,
- Charger IMAGE.PI1 pour une image Degas basse résolution,
- Charger IMAGE.PI2 pour une image Degas moyenne résolution,
- Charger IMAGE.PI3 pour une image Degas haute résolution,

✓ de sélectionner l'option COMPACTER du menu STOS.

Sous STOS, il est possible de compacter et de décompacter des écrans avec la fonction pack et l'instruction unpack.

L'instruction unpack a la syntaxe suivante :

```
unpack <banque1>,<banque2>
```

où <banque1> est le numéro de la banque dans laquelle se trouve l'écran à décompacter,

et <banque2> est le numéro de la banque où doit être stocké l'écran décompacté.

La fonction pack a la syntaxe suivante :

```
<taille>=pack(<banque1>,<banque2>)
```

où <taille> est la taille de l'écran compacté,

<banque1> est la banque dans laquelle se trouve l'écran à compacter,

et <banque2> est la banque dans laquelle doit être stocké l'écran compacté.

Partie II

Le savoir de base en Basic Stos

Chapitre 3

Instructions de base du STOS

3.1. Instructions d'ordre général

3.1.1. Les instructions généralement utilisées en mode direct

Comme dans la plupart des BASIC, le STOS permet de travailler en mode direct et en mode programmé.

En mode direct, la ou les instructions figurant sur une ligne sont directement interprétées et exécutées.

Par exemple :

```
for I=1 to 10 : print I; : next I<CR>
```

provoque l'affichage suivant :

```
1 2 3 4 5 6 7 8 9 10
```

En mode programmé, les lignes tapées sont stockées en mémoire et exécutées lorsque vous entrez l'instruction run en mode direct. En mode programmé, chaque ligne tapée doit débuter par un numéro de ligne compris entre 1 et 65535. L'exemple donné ci-dessus pourrait s'écrire comme suit en mode programmé :

```
10 for I=1 to 10 : print I; : next I
```

ou encore, de manière plus aérée :

```
10 for I=1 to 10
20 print I;
30 next I
```

Pour commencer notre exploration des instructions d'ordre général du STOS, nous allons passer en revue les instructions utilisées en mode direct.

a) Exécution d'un programme

Comme nous l'avons déjà dit, les instructions entrées en mode programmé sont exécutées lorsque vous tapez run en mode direct. L'instruction run admet deux variantes. Sa syntaxe générale est :

```
RUN [<NL>|<Fichier>]
```

où <NL> est un numéro de ligne et <Fichier> un nom de programme basic.

Dans sa version la plus simple, l'instruction run exécute le programme qui se trouve en mémoire, à partir de sa première ligne.

Lorsqu'un numéro de ligne est spécifié, le programme en mémoire est exécuté à partir de ce numéro de ligne.

Enfin, lorsqu'un nom de fichier est spécifié, le fichier est chargé en mémoire et exécuté. Par exemple

```
RUN "Config<CR>
```

charge et exécute le programme de configuration.

Remarque

Les instructions

```
run "config"<CR>
```

et

```
run "config.bas"<CR>
```

sont équivalentes.

b) Sauvegarde du programme en mémoire

Lorsque vous avez tapé un programme en mode programmé, il se trouve en mémoire. Si vous coupez l'alimentation de l'ordinateur, le programme est irrémédiablement perdu. Pour ne pas avoir à réentrer vos programmes chaque fois que vous voulez les exécuter, vous pouvez les sauvegarder sur disquette. Pour cela, deux instructions peuvent être utilisées.

L'instruction save sauvegarde le programme qui se trouve en mémoire sur disque sous le nom spécifié. Cette instruction peut être utilisée pour effectuer des sauvegardes de données très diverses, mais, pour l'instant, nous nous contenterons de voir comment sauvegarder un programme. La syntaxe de l'instruction est la suivante :

```
save <fichier>[.bas]
```

où <fichier> est un nom de fichier d'au maximum 8 lettres. Par exemple :

```
save "monprog.bas
```

Le nom du fichier peut être précédé d'un nom d'unité et/ou de répertoire. Par exemple, l'instruction :

```
save "b:\monrep\monprog.bas
```

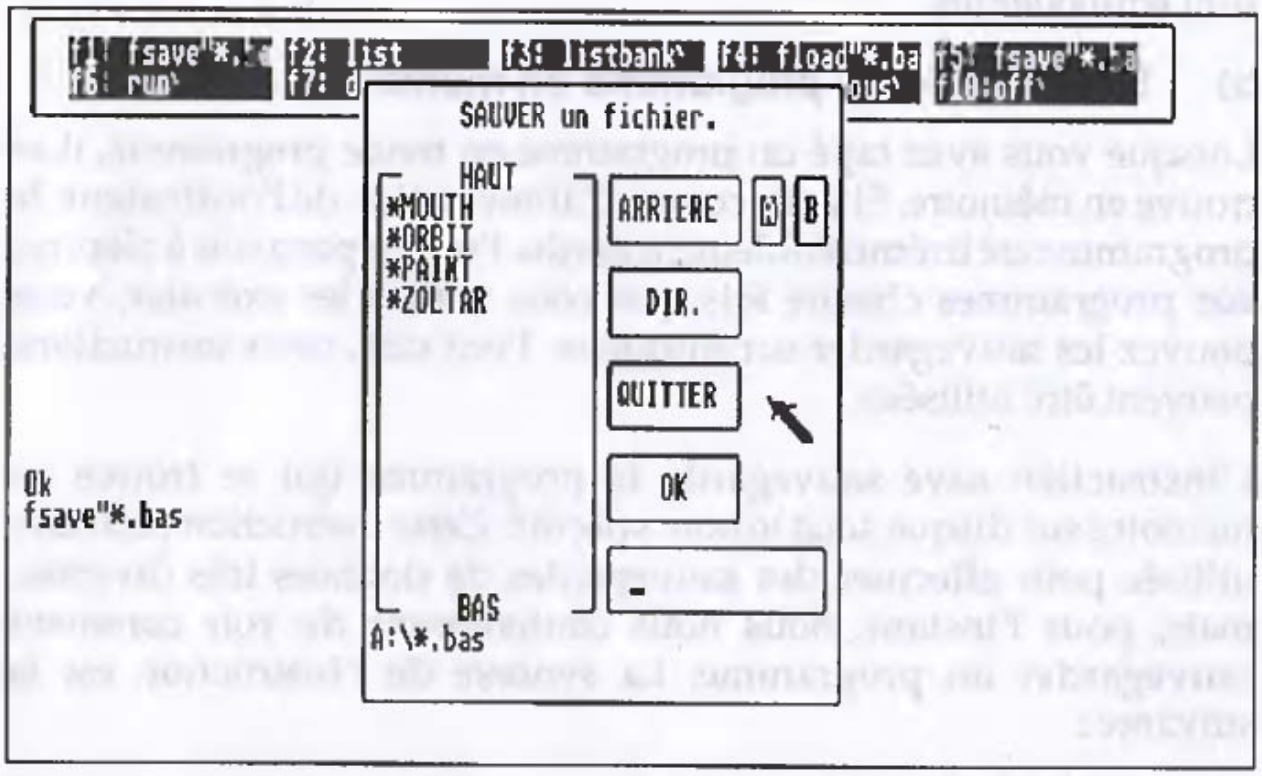
sauvegarde le programme en mémoire sous le nom monprog.bas, dans le répertoire monrep du disque b.

La seconde instruction permettant de charger un fichier est fsave. Cette instruction liste les fichiers dont le type est spécifié, permet d'en choisir un ou d'en créer un autre.

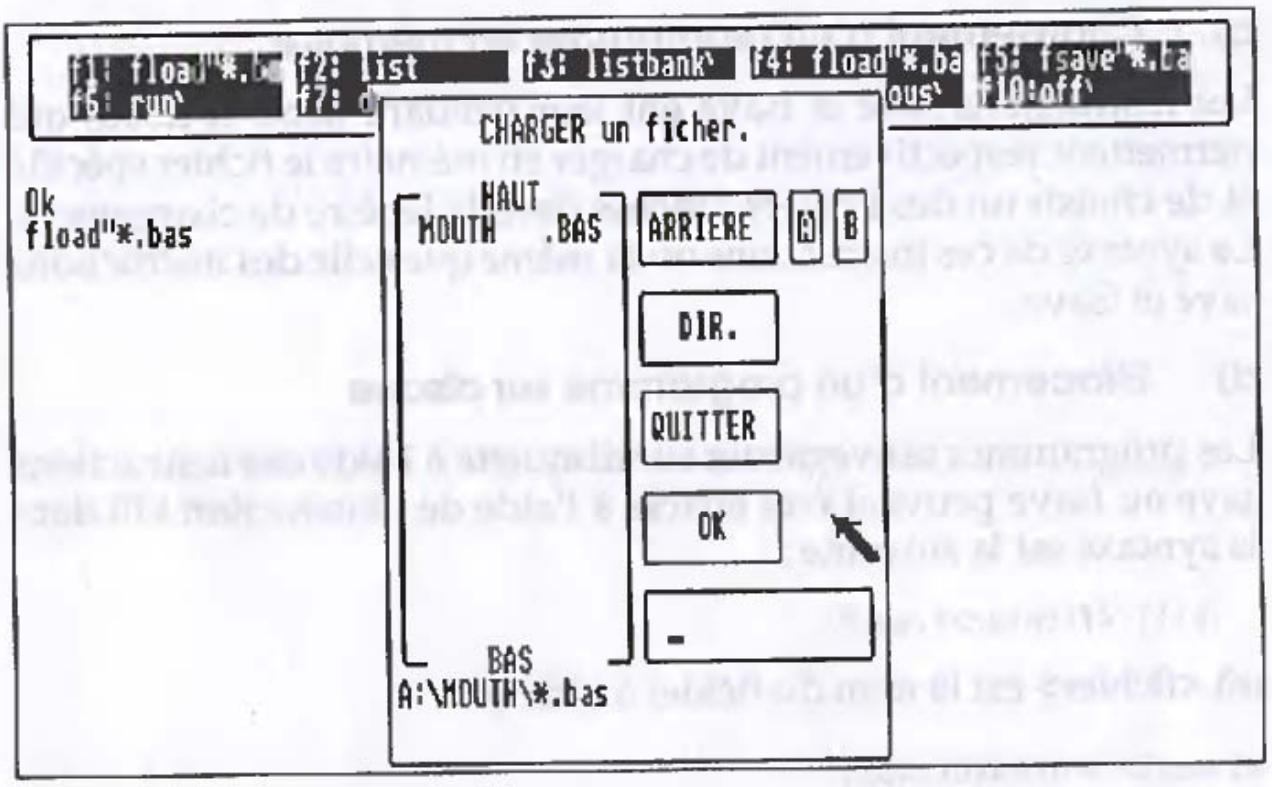
Introduisez la disquette "Jeux" dans le lecteur A: et tapez :

```
fsave "*.bas<CR>
```

Une fenêtre du type suivant apparaît sur l'écran :



Dans le cadre situé à gauche de la fenêtre apparaissent les noms des fichiers qui correspondent au modèle (ici les fichiers d'extension .BAS) et les noms des répertoires, précédés d'un astérisque. Dans notre cas, aucun fichier d'extension .BAS ne se trouve dans le répertoire de base. Cliquez sur le répertoire MOUTH. La fenêtre affiche maintenant les fichiers d'extension .BAS et les sous-répertoires du répertoire MOUTH :



Si vous cliquez sur le bouton :

✓ ARRIERE, vous retournez dans le répertoire père du répertoire courant.

✓ DIR, le répertoire courant est lu et affiché,

✓ QUITTER, vous désactivez la fenêtre de sauvegarde.

Si vous cliquez sur un des noms de fichiers listés, il apparaît dans le cadre situé en bas de la fenêtre. Pour sauvegarder le programme en mémoire sous ce nom, il suffit alors de cliquer sur le bouton OK. Il est également possible de double cliquer sur le nom du fichier pour obtenir le même résultat.

Enfin, il vous est également possible d'entrer le nom du fichier de sauvegarde au clavier et d'appuyer sur la touche <Return>.

c) Chargement d'un programme en mémoire

Les instructions save et fsave ont leur pendant (load et fload) qui permettent respectivement de charger en mémoire le fichier spécifié et de choisir un des fichiers affichés dans la fenêtre de chargement. La syntaxe de ces instructions est la même que celle des instructions save et fsave.

d) Effacement d'un programme sur disque

Les programmes sauvegardés sur disquette à l'aide des instructions save ou fsave peuvent être effacés à l'aide de l'instruction kill dont la syntaxe est la suivante :

```
kill <fichier><.ext>
```

où <fichier> est le nom du fichier à effacer

et <ext> son extension.

Par exemple, l'instruction suivante efface le fichier monprog.bas :

```
kill "monprog.bas
```

Remarques

- ✓ l'instruction kill ne permet d'effacer qu'un fichier qui se trouve dans le disque/répertoire courant. Pour effacer un fichier qui se trouve dans un autre disque/répertoire, rendez ce disque/répertoire courant à l'aide de l'instruction dir$.
- ✓ les jokers ? et * peuvent être utilisés pour (respectivement) remplacer une ou un nombre quelconque de lettres dans le nom du fichier.

e) Numérotation des lignes

Comme vous le savez maintenant, les programmes saisis en mode programmé sont précédés de numéros de lignes. Ces numéros de lignes peuvent être affichés de manière automatique à l'aide de l'instruction auto dont voici la syntaxe :

```
auto [<NL>,[<Esp>]]
```

où <NL> est un numéro de ligne,

et <Esp> la différence entre deux numéros de ligne consécutifs.

Dans sa forme la plus simple

```
auto
```

la première ligne saisie porte le numéro 10 et l'incrémentation entre deux numéros de lignes consécutifs est de 10.

Lorsqu'un numéro de ligne est spécifié, la numérotation commence à partir de la ligne spécifiée et l'incrémentation entre deux numéros de lignes consécutifs est de 10. Par exemple, auto 100 affiche consécutivement les numéros de ligne 100, 110, 120, etc..

Lorsqu'un numéro de ligne et un espacement sont spécifiés, la saisie commence à la ligne spécifiée et l'écart entre deux lignes consécutives correspond à <Esp>. Par exemple, auto 100,5 affiche consécutivement les numéros de ligne 100, 105, 110, etc..

Pour uniformiser un programme, ou déplacer les divers sous-programmes qu'il contient à des numéros de lignes faciles à retenir, il est possible de le renuméroter avec l'instruction renum dont la syntaxe est la suivante :

```
renum [<NL>[,<Esp>[,<Deb>-<Fin>]]]
```

où <NL> est le numéro de la première ligne renumérotée,

<Esp> est l'espacement entre deux lignes consécutives,

<Deb> et <Fin> sont les numéros de première et dernière ligne à renuméroter.

Pour mieux comprendre le fonctionnement de cette instruction, nous allons partir du programme exemple suivant :

```
1 rem Première ligne
2 rem Seconde ligne
3 rem Troisième ligne
4 rem Quatrième ligne
```

L'instruction renum modifie le programme comme suit :

```
10 rem Première ligne
20 rem Seconde ligne
30 rem Troisième ligne
40 rem Quatrième ligne
```

L'instruction renum 100 modifie le programme comme suit :

```
100 rem Première ligne
110 rem Seconde ligne
120 rem Troisième ligne
130 rem Quatrième ligne
```

L'instruction renum 100,1 modifie l'espacement entre deux lignes consécutives :

```
100 rem Première ligne
101 rem Seconde ligne
102 rem Troisième ligne
103 rem Quatrième ligne
```

Enfin l'instruction renum 500,10,102- renumérote les lignes situées entre 102 et la fin du programme à partir du numéro 500, avec un espacement de 10 :

```
100 rem Première ligne
101 rem Seconde ligne
500 rem Troisième ligne
510 rem Quatrième ligne
```

f) Effacement d'un programme

L'instruction new efface le programme qui se trouve en mémoire. Si le programme a été effacé par erreur, il peut être restitué avec l'instruction unnew à condition :

- ✓ qu'aucune nouvelle ligne n'ait été saisie,
- ✓ qu'aucune instruction modifiant l'agencement de la mémoire n'ait été exécutée en mode direct.

L'instruction delete permet d'effacer les lignes dont les numéros sont compris entre les deux limites spécifiées. Sa syntaxe est la suivante :

```
delete <premier>-<dernier>
```

où <premier> est le numéro de la première ligne à supprimer,

et <dernier> le numéro de la dernière ligne à supprimer.

Par exemple, si le programme en mémoire est le suivant :

```
1 rem Première ligne
2 rem Seconde ligne
3 rem Troisième ligne
4 rem Quatrième ligne
5 rem Cinquième ligne
```

l'instruction delete 2-4 effacera les lignes comprises entre 2 et 4. Les lignes restantes seront alors les suivantes :

```
1 rem Première ligne
5 rem Cinquième ligne
```

g) Type d'écriture dans l'éditeur

Par défaut, les instructions entrées dans un programme apparaissent en minuscules, et les variables en majuscules. Il est possible d'inverser ce format avec l'instruction upper. L'instruction lower ramène l'éditeur à son mode par défaut.

h) Remise à zéro de l'éditeur

L'instruction reset (ou l'appui de la touche undo deux fois de suite) initialise l'éditeur. L'écran est effacé, le rappel des touches de fonctions est réaffiché, et la fenêtre courante occupe tout l'écran.

i) Listage d'un programme

L'instruction list permet d'afficher en totalité ou en partie le programme qui se trouve en mémoire. Sa syntaxe est la suivante :

```
list [<début>][-][<Fin>]
```

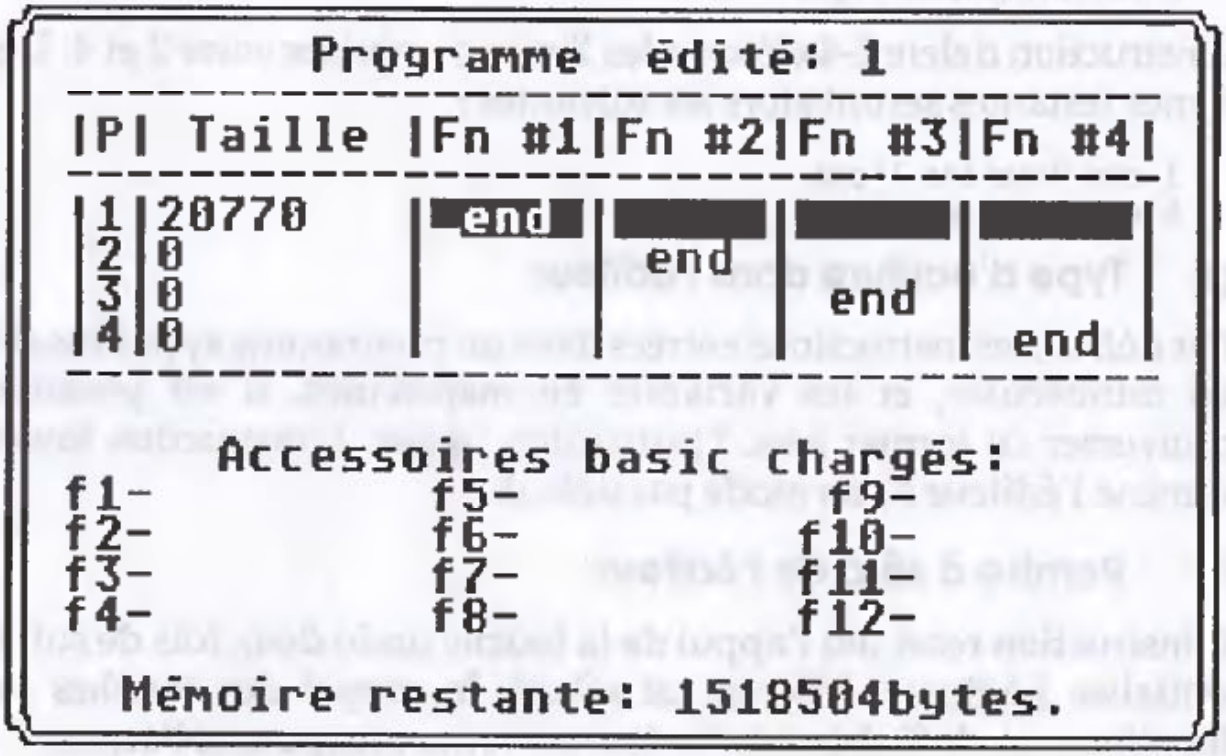
où <début> est le numéro de la première ligne à afficher,

et <fin> est le numéro de la dernière ligne à afficher.

Lorsqu'aucun numéro de ligne n'est spécifié, tout le programme est listé.

j) Edition simultanée de plusieurs programmes

Le BASIC STOS permet d'éditer simultanément jusqu'à quatre programmes en mémoire. Après le lancement du STOS, les instructions que vous tapez sont toujours affectées au premier programme. Pour accéder à un des trois autres programmes possibles, appuyez sur la touche <Help>. Une fenêtre du type suivant est affichée :



En haut de la fenêtre apparaît le numéro du programme en cours d'édition. Pour passer à un autre programme, déplacez le curseur en utilisant les touches flèches vers la bas ou vers le haut. Lorsque le programme que vous désirez éditer est choisi, appuyez à nouveau sur la touche <Help>. A partir de cet instant, toutes les lignes entrées seront affectées à ce programme.

Si vous le désirez, vous pouvez afficher les deux, trois ou quatre programmes édités sur le même écran. Pour cela, vous devez utiliser l'instruction multi qui divise l'écran en deux, trois ou quatre zones. Chaque zone contient un des programmes édités. La syntaxe de l'instruction multi est la suivante :

```
multi {2|3|4)
```

Pour passer d'un des programmes édités à un autre, la méthode sera la même que précédemment.

L'effet de l'instruction multi peut être annulé avec les instructions full, reset ou par l'appui sur la touche <Undo> deux fois de suite.

k) Mixage de deux programmes

Supposons que vous désiriez utiliser les sous-programmes d'un programme existant dans un programme en cours d'élaboration. Vous pouvez recopier les lignes correspondantes, ou utiliser des instructions STOS pour mixer les deux programmes.

Deux méthodes sont possibles.

Première méthode

Les deux programmes se trouvent en mémoire

L'instruction grab copie la totalité ou une partie du programme N (N compris entre 1 et 4) dans le programme en cours d'édition. La syntaxe de l'instruction grab est la suivante :

```
grab <N>[,<début>-<fin>]
```

où <N> est le numéro du programme à copier,

<début> est la première ligne du programme N à copier

et <fin> la dernière ligne du programme N à copier.

Seconde méthode

Le programme à copier n'est pas en mémoire

L'instruction merge copie le programme disque spécifié dans le programme en cours d'édition. Attention: les éventuelles lignes du programme en cours d'édition qui portent le même numéro que certaines des lignes du programme disque sont écrasées par ces dernières. La syntaxe de l'instruction merge est la suivante :

```
merge <fichier>
```

où <fichier> est le nom du programme disque à mixer.

Selon si vous désirez intégrer la totalité ou une partie d'un programme au programme courant, vous emploierez la première ou la seconde méthode.

D) Aide à la recherche d'erreur

Le BASIC STOS possède des instructions qui pourront s'avérer bien utiles lorsque vos programmes ne fonctionneront pas comme vous le souhaitez.

❶ L'instruction search recherche l'occurrence d'une chaîne dans le programme. Sa syntaxe est la suivante :

```
search [<chaîne>]
```

où <chaîne> est la chaîne recherchée.

Par exemple, pour rechercher la chaîne "utilisateur" dans le programme, il faudra taper :

```
search "utilisateur"
```

Le programme affichera la première ligne contenant la chaîne "utilisateur" ou le message "La recherche a échoué." s'il n'a pas pû localiser cette chaîne. La prochaine occurrence de cette même chaîne sera recherchée par la simple instruction search, sans préciser la chaîne recherchée.

❷ L'instruction change recherche l'occurrence d'une chaîne dans le programme et la remplace par une autre chaîne. Elle peut agir sur tout le programme ou seulement entre deux lignes dont les numéros sont spécifiés. Sa syntaxe est la suivante :

```
change <ch1> to <ch2> [,<debut>-<fin>]
```

où <ch1> est la chaîne à remplacer,

<ch2> est la chaîne de remplacement,

<debut> et <fin>, s'ils sont précisés, spécifient les numéros de lignes entre lesquels agit l'instruction change.

❸ L'instruction follow permet de :

✓ détecter le passage du programme à une ligne ou un ensemble de lignes successives particulier,

✓ suivre l'évolution d'une ou de plusieurs variables pendant l'exécution du programme.

Quel que soit le type de résultat recherché, le programme s'exécute en "pas à pas", c'est-à-dire instruction par instruction, ce qui permet par exemple d'exécuter une ou plusieurs instructions en mode direct pour identifier d'où vient l'erreur. La syntaxe de l'instruction follow est la suivante :

```
follow <début>-<fin>
```

ou

```
follow <v1>[,...[,<vN>]...]
```

ou

```
follow <v1>[,...[,<vN>]...],<début>-<fin>
```

ou

```
follow off
```

Dans la première version, le programme s'exécute pas à pas entre les lignes <début> et <fin>.

Dans la seconde version, les valeurs des variables <v1> à <vN> sont affichées en haut de l'écran pendant toute l'exécution pas à pas du programme.

Dans la troisième version, les valeurs des variables <v1> à <vN> sont affichées en haut de l'écran pendant l'exécution pas à pas du programme entre les lignes <début> et <fin>.

Enfin, la quatrième version supprime l'effet de l'instruction follow.

❹ Les dernières instructions destinées à faciliter la recherche d'erreurs dans un programme sont stop et cont. Elles permettent respectivement d'arrêter et de reprendre l'exécution d'un programme. L'instruction stop sera insérée en un point stratégique du programme. Vous pourrez ainsi tester en mode direct toutes les variables qui vous intéressent afin de déterminer la cause du mauvais fonctionnement. L'exécution reprendra à l'instruction qui suit le stop lorsque vous taperez cont en mode direct.

m) Langue des messages système

Nous allons terminer notre tour d'horizon des instructions utilisées en mode direct par les instructions francais et english. Ces instructions permettent de choisir la langue (Français ou Anglais) d'expression des "messages système". Les messages système sont tous les messages renvoyés par le STOS, à savoir, les messages d'erreur et d'information, les messages informatifs affichés dans les fenêtres STOS.

3.1.2. Instructions utilisées en mode programmé

Avant d'être en mesure d'écrire vos propres jeux en BASIC STOS, vous devez acquérir la technique de programmation de ce langage. Cet apprentissage passe par la découverte, puis la maîtrise des instructions de base de la programmation. La progression vous paraîtra lente, surtout si vous débutez en BASIC, mais cela en vaut la peine car le STOS est un langage puissant et agréable qui mérite le mal que vous allez vous donner ...

a) Débranchements

Les instructions qui figurent dans un programme écrit en BASIC STOS s'exécutent séquentiellement, des numéros de lignes faibles vers les numéros de lignes élevés, à moins qu'une instruction de débranchement ne soit rencontrée.

Les instructions de débranchement sont de deux types :

❶ débranchement vers une étiquette sans retour,

❷ débranchement vers une étiquette avec retour.

Débranchement vers une étiquette sans retour

L'instruction de base du débranchement sans retour est goto. Elle provoque la poursuite du programme à la ligne indiquée. La ligne peut être spécifiée en clair, comme par exemple dans :

```
goto 1000
```

ou sous forme calculée, comme par exemple dans :

```
goto 120*I+10
```

Une variante très puissante de l'instruction goto est l'instruction on goto qui permet le débranchement multiple du programme en fonction de la valeur d'une variable. La syntaxe de cette instruction est la suivante :

```
on <var> goto <ligne1>[,<ligne2>[,...[,<ligneN>]...]]
```

où <var> est une variable entière,

et les <ligneI> sont des numéros de lignes.

La Ième ligne spécifiée est exécutée si <var> vaut I. Il est donc nécessaire d'avoir autant de <ligneI> que de valeurs possibles de <var>.

Débranchement vers une étiquette avec retour

L'instruction de base du débranchement avec retour est gosub. Elle donne le contrôle au sous-programme dont la ligne est spécifiée. Dans ce sous-programme doit figurer l'instruction return qui redonne le contrôle à l'instruction qui suit le gosub.

Comme pour l'instruction goto, gosub peut faire référence à un numéro de ligne spécifié en clair, comme par exemple dans :

```
gosub 1000
```

ou calculé, comme par exemple dans :

```
gosub 120*I+10
```

De même que l'instruction on goto, l'instruction on gosub donne le contrôle à un programme dont le numéro de ligne dépend de la valeur d'une variable. La syntaxe de cette instruction est la suivante :

```
on <var> gosub <ligne1>[,<ligne2>[,...[,<ligneN>]...]]
```

où <var> est une variable entière,

et les <ligneI> sont des numéros de lignes.

La Ième ligne spécifiée est exécutée si <var> vaut I. Il est donc nécessaire d'avoir autant de <ligneI> que de valeurs possibles de <var>.

La rencontre d'une instruction return donne le contrôle à l'instruction qui suit le gosub.

Citons enfin l'instruction pop qui efface l'information de retour générée par les instructions gosub et on gosub. Sauf dans des cas très particuliers, cette instruction devra être évitée, car elle nuit à la clarté du programme.

b) Tests et répétitions

L'instruction if est très utile en programmation. Elle permet de tester une condition logique et, en fonction du test, exécute un bloc d'instructions ou un autre, ou donne le contrôle à une ligne ou à une autre. Sa syntaxe est la suivante :

```
if <condition> then <bloc1> else <bloc2>
```

où <condition> est une expression qui peut être évaluée à vrai ou à faux, par exemple b=2 ou encore (c>0) and (a+b=10)

<bloc1> est un numéro de ligne ou une ou plusieurs instructions,

et <bloc2> est un numéro de ligne ou une ou plusieurs instructions.

Considérons l'exemple suivant :

```
10 input "Entrez votre age : ";age
20 if age<20 then 30 else 40
30 print "bonjour mon petit":goto 50
40 print "bonjour mon grand"
50 end
```

Exécutez le programme. Si vous entrez une valeur inférieure à 20, le programme affichera "bonjour mon petit", sinon, il affichera "bonjour mon grand".

Lorsque vous désirez effectuer une action répétitive, vous pouvez répéter les lignes correspondant à l'action autant de fois que nécessaire :

```
10 print "meubles tarnal"
20 print "meubles tarnal"
...
100 print "meubles tarnal"
110 print "meubles tarnal"
```

Vous pouvez aussi inclure les lignes correspondant à l'action dans une boucle, par exemple :

```
10 for I=1 to 20
20 print "meubles tarnal"
30 next I
```

Vous venez de découvrir le premier type de boucle du BASIC STOS. Sa syntaxe est :

```
for <var>=<début> to <fin> [step <pas>]
[<instruction1>[....[.<InstructionN>]...]]
next <var>
```

où <var> est une variable numérique qui varie entre <début> et <fin>. Lorsqu'un pas est spécifié, il est ajouté à <var> à chaque boucle. Dans le cas contraire, le pas est par défaut égal à 1.

La boucle prend fin lorsque :

✓ <var> a une valeur supérieure à <fin> (si le pas est positif),

✓ var a une valeur inférieure à <fin> (si le pas est négatif).

Les boucles du type for next sont très pratiques lorsque le nombre d'itérations à effectuer est connu à l'avance. Mais il peut également être nécessaire d'effectuer une boucle tant qu'une condition logique est vraie ou jusqu'à ce qu'une condition logique soit vraie. C'est là le but des boucles while wend et repeat until.

La boucle while wend exécute une ou plusieurs instructions tant qu'une condition logique est vraie. Sa syntaxe est :

```
while <condition>
[<instruction1>[....[.<InstructionN>]...]]
wend
```

où <condition> est une condition logique qui peut être évaluée à vrai ou à faux, par exemple Z=0 ou (I > 4) or (T < 0)

et les <instructionI> sont des instructions BASIC.

Pour mieux comprendre le fonctionnement de la boucle while wend, considérez le programme suivant qui affiche une suite croissante de nombres tant qu'une touche du clavier n'est pas pressée :

```
10 CH$="" : I=0
20 while CH$=""
30 I=I+1
40 print I;
```

```
50 CH$=inkey$
60 wend
70 end
```

La boucle repeat until exécute une ou plusieurs instructions jusqu'à ce qu'une condition logique soit vraie. Sa syntaxe est :

```
repeat
[<instruction1>[,...[,<InstructionN>]...]]
until <condition>
```

où <condition> est une condition logique qui peut être évaluée à vrai ou à faux, par exemple Z=0 ou (I > 4) or (T < 0)

et les <instruction1> sont des instructions BASIC.

L'exemple qui suit montre comment réaliser un programme équivalent au précédent à l'aide d'une boucle repeat until.

```
10 I=0
20 repeat
30 I=I+1
40 print I;
50 CH$=inkey$
60 until CH$<>""
70 end
```

c) Gestion des erreurs

Au cours de votre vie d'informaticien, vous aurez fatalement l'occasion d'écrire des programmes bogués (erronés) qui produiront des erreurs lors de leur exécution. Le langage BASIC STOS possède plusieurs instructions, fonctions et variables liées à la gestion des erreurs d'exécution. Nous allons les étudier en détail.

Lorsqu'une erreur se produit pendant l'exécution d'un programme, les variables errl et errn contiennent respectivement le numéro de ligne où s'est produite l'erreur et le numéro de l'erreur. Reportez-vous à l'annexe C pour prendre connaissance de la liste des erreurs possibles et de leur cause.

L'instruction error permet de générer une erreur dans le but de tester un sous-programme de gestion d'erreur. Sa syntaxe est :

```
error <n>
```

où <n> est un numéro d'erreur compris entre 1 et 86.

L'instruction on error goto placée au début d'un programme déroute ce dernier lorsqu'une erreur se produit vers le programme de traitement dont la ligne est spécifiée. L'instruction resume permet de reprendre l'exécution du programme après le passage dans le programme de traitement.

La syntaxe de ces deux instructions est la suivante :

```
on error goto <ligne>
```

où <ligne> est le numéro de la première ligne du programme de traitement des erreurs

```
resume
```

ou

```
resume next
```

ou

```
resume <ligne>
```

Dans sa première version, l'instruction resume réexécute le programme à partir de l'instruction qui a provoqué l'erreur.

Dans sa seconde version, le contrôle est donné à l'instruction qui suit l'instruction qui a causé l'erreur.

Enfin, dans sa troisième version, l'instruction resume donne le contrôle à la ligne dont le numéro est spécifié.

Pour concrétiser ce qui vient d'être dit sur les instructions relatives à la gestion des erreurs, voici un programme qui affiche en clair le type de chaque erreur et permet à l'utilisateur :

- ✓ de réexécuter l'instruction qui a provoqué l'erreur,
- ✓ d'exécuter l'instruction suivante,
- ✓ d'arrêter l'exécution du programme.

```
10 on error goto 10000
20 dim ER$(86)
30 for I=1 to 86
40 read ER$(I)
50 next I
60 CLS: print"début du programme"
70 error 3
80 CLS: print"fin du programme"
90 end
10000 print ER$(errn);" ligne";erl
10010 print "1) Instruction ayant provoqué l'erreur, 2)
Instruction suivante"
10020 input "3) Fin du programme : ";A
10030 if A=1 then resume
10040 if A=2 then resume next
10050 end
20000 data "Mauvais format de fichier","Mémoire pleine"
20010 data "Cette ligne n'existe pas","Cette ligne existe déjà"
20020 data "La recherche a échoué","Ligne trop
longue","Impossible de continuer"
20030 data "Mémoire pleine","Follow trop long","L'imprimante
n'est pas prête"
20040 data "Renumérotation impossible","Erreur de syntaxe"
20050 data "Appel illégal de fonction","Instruction interdite en
mode direct"
20060 data "Instruction interdite en mode programme","Erreur
d'entrée/sortie"
20070 data "Stop","Tableau non déclaré","Types de variables
incompatibles"
20080 data "Fonction non implémentée","Dépassement de
capacité","For sans next"
20090 data "Next sans for","While sans wend","Wend sans while"
20100 data "Repeat sans until","Until sans repeat","Tableau déjà
défini"
20110 data "Numéro de ligne non défini","Chaîne trop
longue","Erreur de bus"
20120 data "Erreur d'adresse","Pas de 'data' sur cette
ligne","Plus de donnée"
20130 data "Trop de gosub","Return sans gosub","Pop sans gosub"
20140 data "Resume sans erreur","Fonction utilisateur non
définie"
20150 data "Mauvais appel de fonction utilisateur"
20160 data "Banque mémoire déjà réservée","Banque mémoire non
écran"
20170 data "Mauvaise adresse d'écran","Banque mémoire non
réservée"
20180 data "Résolution non autorisée","Division par
zéro","Opérande négatif"
20190 data "Fichier introuvable","Lecteur pas prêt","Disquette
protégée"
```

```
20200 data "Disquette pleine","Erreur disquette","Mauvais nom de
fichier"
20210 data "Mauvaise heure","Mauvaise date","Erreur de sprite"
20220 data "Mauvais appel de MOVE","Mauvais appel
d'ANIM","Fichier non ouvert"
20230 data "Mélange de types de fichiers","Chaîne en entrée trop
longue"
20240 data "Fichier déjà ouvert","Fichier déjà fermé","Fin de
fichier"
20250 data "Chaîne en entrée trop longue","Champ trop long"
20260 data "Mauvais appel de FLASH","Paramètre de fenêtre trop
grand"
20270 data "Fenêtre déjà ouverte","Fenêtre non ouverte","Fenêtre
trop petite"
20280 data "Fenêtre trop grande","Jeux de caractères non défini"
20290 data "Buffer texte plein","Musique non définie"
20300 data "Appel d'une fenêtre système","Appel d'un jeu de
caractères système"
20310 data "Jeu de caractères introuvable","Menu non défini"
20320 data "La banque 15 est déjà réservée"
20330 data "La banque 15 est réservée pour les
menus","Instruction illégale"
20340 data "Lecteur non connecté","Extension non
chargée","Indice trop grand"
20350 data "Appel illégal de fonction"
```

d) Date et heure

Le BASIC STOS possède quelques instructions dédiées à la gestion des date et heure. Nous allons les passer en revue.

Le mot clé date$ permet d'accéder à la date système, en lecture et en écriture.

Pour lire la date système, tapez :

```
print date$
```

La date apparaît sous la forme jj/mm/aaaa, par exemple 01/04/1990.

Pour initialiser la date système, tapez par exemple :

```
date$="01/06/1990"
```

L'entrée d'une date incorrecte provoque l'affichage du message "Mauvaise date" et le refus de la date entrée.

Le mot clé time$ permet d'accéder à l'heure système, en lecture et en écriture.

Pour lire l'heure système, tapez :

```
print time$
```

L'heure apparaît sous la forme hh:mm:ss, par exemple 10:01:08.

Pour initialiser l'heure système, tapez par exemple :

```
time$="10:00:00"
```

L'entrée d'une heure incorrecte provoque l'affichage du message "Mauvaise heure" et le refus de l'heure entrée.

Il est également possible d'accéder à un compteur qui s'incrémente automatiquement d'un tous les 1/50 secondes à l'aide du mot clé timer. Tout comme pour date$ et time$, le compteur peut être accédé en lecture et en écriture.

Par exemple :

✓ l'instruction timer=0 initialise le compteur à 0,

✓ l'instruction print timer affiche la valeur actuelle du compteur.

Pour clore ce paragraphe, citons enfin l'instruction wait qui arrête le fonctionnement du programme pendant un délai paramétrable en 1/50 seconde. La syntaxe de cette instruction est :

```
wait <délai>
```

où <délai> est le délai d'attente en 1/50 seconde.

Remarque

Les instructions qui fonctionnent sous interruptions ne sont pas affectées par wait.

e) Entrée/sortie

Dans ce paragraphe, nous allons étudier un groupe très important d'instructions. Leur but est :

- ✓ l'affichage de données sur l'écran,
- ✓ l'entrée de données au clavier,
- ✓ la lecture de données dans le programme.

Affichage de données sur l'écran

L'instruction de base pour afficher des données sur l'écran est l'instruction print dont voici la syntaxe :

```
print <v1>[{,;}<v2>[...[{,;}<vN>]...]]
```

où les <vI> sont des constantes, des variables ou des expressions, par exemple "bonjour", upper$(A$), I#, A, C*10, etc..

Les données affichées peuvent être formatées à l'aide du mot clé using rajouté après le print. La syntaxe d'une instruction print using est la suivante :

```
print using "<ch>";<var>
```

où <ch> est une chaîne de formatage qui peut également comporter une partie texte,

et <var> est un champ texte ou numérique à formater.

La chaîne de formatage peut être composée des signes suivants :

```
~    remplace une lettre
#    remplace un chiffre
+    signe + devant un nombre positif, signe - devant un nombre
     négatif
-    espace devant un nombre positif, signe - devant un nombre
     négatif
```

Exemples

Le petit programme suivant donne un aperçu des possibilités de l'instruction print using :

```
10 A$="allez" : A#=sqr(2)
20 print using "comment ~~~~~~ vous";A$
30 print using "comment ~~~~ vous";A$
40 print using "#.##";A#
50 print using "+#.###";A#
60 print using "+#.###";A#
70 print using "-#.###";A#
80 print using "-#.###";A#
90 print using "#;###";A#
```

Lancez le programme. Les données suivantes sont affichées sur l'écran :

```
comment allez vous
comment all vous
1.41
+1.414
-1.414
 1.414
-1.414
1 414
```

Lecture de données au clavier

L'instruction de base pour lire une ou plusieurs données au clavier est input. En voici la syntaxe :

```
input [<texte>;]<v1>[{,;}<v2>[...[{,;}<vN>]...]]
```

où <texte> est un message alphanumérique quelconque. S'il est absent, il est automatiquement remplacé par un point d'interrogation.

et les <vI> sont des variables de tous types dans lesquelles seront stockées les données lues.

Exemples

```
10 input "Entrez votre âge : ";AGE
20 input "Entrez trois chaînes : ";A$,B$,C$
```

Si vous désirez lire une chaîne dont la longueur est fixe, vous pouvez également utiliser la fonction input$(n) où n est le nombre de caractères à lire. Par exemple, pour lire un mot de passe sur 5 caractères :

```
10 a$=input$(5)
```

L'inconvénient de cette fonction est qu'elle n'affiche pas les caractères tapés sur l'écran.

Lorsque vous désirez lire une chaîne de caractères qui risque de comporter une ou plusieurs virgules (,), vous devez utiliser l'instruction line input qui n'interprète pas ces caractères comme des séparateurs. La syntaxe de l'instruction line input est la même que celle de l'instruction input.

Si vous désirez tester la frappe d'une touche du clavier, vous devez utiliser la fonction inkey$. Tant qu'aucune touche n'est pressée, inkey$ renvoie la valeur "". Lorsqu'une touche est pressée, inkey$ renvoie son code ASCII.

Certaines touches du clavier (comme les touches du curseur ou les touches de fonction) ont un code ASCII nul. Pour les identifier, un code clavier leur est associé. Ce code peut être obtenu à l'aide de la fonction scancode.

Le petit programme ci-dessous illustre les fonctions inkey$ et scancode :

```
10 A$=inkey$ : if A$="" then 10
20 print asc(A$),scancode
30 if scancode <> 28 then 10
```

La ligne 10 attend la frappe d'une touche du clavier. La ligne 20 affiche le code ASCII et le code clavier de la touche pressée. La ligne 30 redonne le contrôle à la ligne 10 si la touche pressée n'est pas la touche Return.

Pour terminer le paragraphe dédié aux instructions d'entrée au clavier, signalons également la fonction fkey et l'instruction wait key. La fonction fkey a un fonctionnement comparable à la fonction

inkey$, mais elle est surtout utilisée avec les touches de fonction. Lorsqu'une touche de fonction est pressée, elle renvoie son numéro (entre 1 et 20). Lorsqu'une autre touche est pressée, fkey renvoie son code clavier.

Exemple

```
10 FK=fkey : if FK=0 then 10
20 print FK
30 if FK<>10 then 10
```

La ligne 10 attend la frappe d'une touche. La ligne 20 affiche le numéro de la touche de fonction ou le code clavier de la touche pressée. La ligne 30 redonne le contrôle à la ligne 10 si la touche pressée est différente de <F10>.

L'instruction wait key suspend l'exécution du programme jusqu'à ce qu'une touche du clavier soit pressée. Cette instruction n'affecte en rien les instructions qui fonctionnent sous interruptions.

Lecture de données dans un programme

Les instructions data, read et restore sont particulièrement utiles lorsque vous désirez initialiser des variables dans un programme. L'instruction read lit les données situées après l'instruction data et les stocke dans les variables spécifiées. L'exemple suivant initialise le tableau JOUR$ avec le nom des jours de la semaine :

```
10 dim JOUR$(7)
20 for I:=1 to 7
30 read JOUR$(I)
40 next I
50 data "Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi",
"Dimanche"
```

L'instruction restore permet de relire deux ou plusieurs fois les données situées après les instructions data. Sa syntaxe est la suivante :

```
restore [<ligne>]
```

où <ligne> est le numéro d'une ligne de data, par exemple 1000 ou encore I*10.

Si aucun numéro de ligne n'est spécifié dans l'instruction, le restore se réfère à la première ligne de data du programme.

Pour mieux comprendre le fonctionnement de l'instruction restore, considérez l'exemple suivant.

Entrez ce court programme et exécutez-le.

```
10 read A,B,C,D
20 data 10,20
```

Une erreur se produit en ligne 10. Le STOS affiche le message suivant :

```
Plus de donnée en ligne 10.
```

Modifiez le programme comme suit :

```
10 read A,B
20 restore
30 read C,D
40 print A,B,C,D
100 data 10,20
```

Le programme s'exécute maintenant sans erreur et affiche

```
10   20    10   20
```

L'instruction restore en ligne 20 a repositionné le pointeur de lecture des data sur la ligne 100.

f) Divers

Dans ce paragraphe, nous avons rassemblé toutes les instructions d'ordre général qui n'entraient pas dans les sous-groupes définis précédemment.

L'instruction rem permet d'insérer des remarques dans un programme pour en clarifier la lecture ou pour souligner un détail particulier.

Par exemple :

```
1000 rem ******************************
1010 rem Sous-programme de déplacement
1020 rem ******************************
```

Lorsque plusieurs données de même type doivent être manipulées, il est pratique d'utiliser un tableau. Dans le langage BASIC STOS, un tableau est une structure uni ou multi-dimentionnelle composée d'autant d'éléments que vous le désirez, dans la limite de l'espace mémoire disponible. Une contrainte cependant : chacune des dimensions du tableau doit comporter moins de 65536 éléments. La syntaxe de cette instruction est :

```
dim <nom>(<d1>[,<d2>[...[<dN>]...]])
```

où <nom> est le nom du tableau,

et les <dI> sont les dimensions du tableau <nom>.

Supposons que vous ayez défini le tableau T suivant :

```
dim T(10)
```

Ce tableau comporte 11 éléments : T(0), T(1), ..., T(10).

De même, si vous définissez le tableau N$(2,3), il sera constitué de 12 éléments (2+1 * 3+1) : N$(0,0), N$(0,1), N$(0,2), N$(0,3), N$(1,0), N$(1,1), N$(1,2), N$(1,3), N$(2,0), N$(2,1), N$(2,2), N$(2,3).

L'instruction sort permet de classer les éléments d'un tableau dans l'ordre ascendant. Sa syntaxe est :

```
sort <nom>(0)
```

où <nom> est le nom d'un tableau.

L'exemple suivant montre clairement comment utiliser l'instruction sort :

```
10 dim T(10)
20 for I=0 to 10
30 read T(I)
40 next I
50 for I=0 to 10 : print T(I); : next I : print
```

```
60 sort T(0)
70 for I=0 to 10 : print T(I);:next I
80 data 13,0,4,6,123,76,12,9,34,6,2
```

Exécutez ce programme. Les données suivantes s'affichent sur l'écran :

```
13 0 4 6 123 76 12 9 34 6 2  --> avant classement
0 2 4 6 6 9 12 13 34 76 123  --> après classement
```

L'instruction match recherche dans un tableau la valeur la plus proche de la valeur spécifiée. Sa syntaxe est :

```
match (<nom>(0),<val>)
```

où <nom> est le nom d'un tableau,

et <val> la valeur de test.

La valeur renvoyée est positive si <val> a été trouvée dans <nom>. Dans le cas contraire, la valeur renvoyée est négative, et sa valeur absolue indique le premier élément dans <nom> supérieur à <val>.

Remarque

Avant d'utiliser l'instruction match, le tableau où s'effectuera la recherche doit être trié à l'aide de l'instruction sort.

Pour mieux comprendre le fonctionnement de cette instruction, considérez l'exemple suivant :

```
10 dim T(10)
20 for I=0 to 10
30 read T(I)
40 next I
50 sort T(0)
60 for I=0 to 10 : print T(I); : next I : print
65 input "Entrez un nombre : ";N
70 print "La première valeur supérieure ou égale à ";N;"est ";
80 I=match(T(0),N) : print T(abs(I))
80 data 13,0,4,6,123,76,12,9,34,6,2
```

La fonction asc donne le code ASCII du caractère qui lui est donné. Pour avoir une idée des codes ASCII des caractères affichés sous STOS, reportez-vous à l'annexe A. La syntaxe de la fonction asc est :

```
<res> = asc(<ch>)
```

où <res> est une variable de type entier,

et <ch> est un caractère ou une chaîne de caractères.

Si <ch> est une chaîne de caractères, asc retourne le code ASCII du premier caractère de la chaîne.

La fonction chr$ est la réciproque de asc. Elle renvoie le caractère correspondant au code ASCII qui lui est donné. Sa syntaxe est la suivante :

```
<res> = chr$(<asc>)
```

où <asc> est un code ASCII compris entre 0 et 255,

et <res> est le caractère correspondant.

Lorsqu'un programme ne spécifie pas le contraire, il peut à tout moment être interrompu par l'appui simultané sur les touches Ctrl et C. L'instruction break off peut être utilisée pour dévalider la fonction des touches Ctrl C. Inversement, l'instruction break on valide les touches Ctrl C.

L'instruction clear :

- ✓ initialise toutes les variables du programme : les variables numériques sont mises à 0, les variables alphanumériques sont mises à "",
- ✓ réinitialise le pointeur de data (comme le ferait une instruction restore),
- ✓ efface toutes les banques utilisées par le programme.

Les instructions dec et inc (respectivement) décrémentent et incrémentent d'un les variables entières qui leur sont passées. Leur syntaxe est :

```
dec <var>
inc <var>
```

où <var> est le nom d'une variable entière.

L'instruction end marque la fin d'un programme. Elle est facultative.

Enfin, l'instruction system permet de quitter le STOS pour retourner au système d'exploitation, après confirmation.

3.2. Instructions à caractère mathématique

Comme tous les BASIC, le STOS possède un ensemble d'instructions à caractère mathématique. Cet ensemble est néanmoins plus étendu que sur la plupart des BASIC. Il permet entre autres :

- ✓ d'utiliser les fonctions trigonométriques et hyperboliques,
- ✓ d'accéder au niveau bit des variables entières,
- ✓ de changer de base de numération (2, 10 et 16),
- ✓ de définir des fonctions utilisateur.

La constante pi contient une approximation du nombre PI (3.14159...).

La fonction abs renvoie la valeur absolue de la quantité qui lui est donnée. Sa syntaxe est la suivante :

```
<x>=abs(<y>)
```

où <y> est une variable ou un nombre entier ou réel,

et <x> est la valeur absolue de <y>.

La fonction sgn renvoie le signe de la variable ou de l'expression qui lui est donnée. Sa syntaxe est :

```
<res> = sgn(<var>)
```

où <var> est la variable ou l'expression dont vous voulez connaître le signe,

et <res> est le signe de <var> :

```
1 si <var> est positif,
0 si <var> est nul,
-1 si <var> est négatif.
```

Fonctions trigonométriques et hyperboliques

Les fonctions deg et rad permettent de convertir des angles (respectivement) en degrés et en radians. Rappelons que :

```
2 pi radians = 360 degrés
```

d'où :

```
1 radian = 180 / pi degrés,
```

et :

```
1 degré = pi / 180 radians
```

La syntaxe de ces instructions est :

```
<x> = deg(<y>)
<z> = rad(<t>)
```

où <y> et <z> sont des mesures en radians,

et <x> et <t> sont des mesures en degrés.

Les fonctions cos, sin et tan renvoient (respectivement) le cosinus, le sinus et la tangente de l'angle en radians qui leur est donné. Le résultat est de type réel. La syntaxe de ces instructions est la suivante :

```
<x> = cos(<y>)
<x> = sin(<y>)
<x> = tan(<y>)
```

où <y> est un angle exprimé en radians,

et <x> est le cosinus/sinus/tangente de <y>.

Les fonctions acos, asin et atan renvoient (respectivement) l'arc cosinus, l'arc sinus et l'arc tangente de la mesure d'angle qui leur est donnée. Le résultat est de type réel. La syntaxe de ces instructions est la suivante :

```
<y> = acos(<x>)
<y> = asin(<x>)
<y> = atan(<x>)
```

où <x> est une mesure d'angle,

et <y> est l'angle (en radians) dont le cosinus/sinus/tangente est <x>.

Les fonctions hcos, hsin et htan renvoient (respectivement) le cosinus hyperbolique, le sinus hyperbolique et la tangente hyperbolique de la valeur qui leur est donnée. Le résultat est de type réel. La syntaxe de ces instructions est la suivante :

```
<x> = hcos(<y>)
<x> = hsin(<y>)
<x> = htan(<y>)
```

où <y> est un angle exprimé en radians,

et <x> est le cosinus/sinus/tangente hyperbolique de <y>.

Les constantes true et false représentent (respectivement) les valeurs -1 et 0. Elles peuvent être utilisées dans des tests et des affectations binaires.

L'opérateur not inverse la valeur d'un booléen. Supposons que la variable A ait pour valeur true (-1). Alors not(A) aura pour valeur false (0) et not(not(a)) aura pour valeur true (-1).

Cet opérateur peut par exemple être utilisé dans des boucles, ou dans des calculs logiques. Comme par exemple dans :

```
100 repeat
...
600 until not STOP
```

ou encore :

```
A = (B and C) or not(D)
```

L'opérateur and effectue un ET logique entre les deux paramètres qui lui sont donnés. Le résultat est du type des paramètres (booléen ou entier). Lorsque les paramètres sont entiers, la fonction and est appliquée sur chacun des bits de même rang des paramètres. La table logique de l'opérateur and est la suivante :

and	0	1
0	0	0
1	0	1

Exemple

```
print bin$(%11001100 and %00110011)
%00000000
```

Dans la suite logique de l'opérateur and, le STOS dispose des opérateurs or et xor qui effectuent respectivement un OU logique et un OU EXCLUSIF logique. Les tables de vérité de ces deux opérateurs sont les suivantes :

or	0	1
0	0	1
1	1	1

xor	0	1
0	0	1
1	1	0

Exemple

```
print bin$(%11001100 or %00110011)
%11111111
print bin$(%10101010 xor %11111111)
%01010101
```

Changement de base

Les nombres entiers du BASIC STOS sont manipulés sous forme décimale, c'est-à-dire dans la base utilisée dans la vie de tous les jours. Le microprocesseur de l'ATARI, par contre, utilise la base 2 pour effectuer tous ses calculs.Cette base est dite binaire. Pour faciliter l'accès à certaines variables système ou aux opérations qui touchent de près le microprocesseur, le STOS contient deux fonctions (bin$ et hex$) qui renvoient la valeur du nombre qui leur est passé respectivement en base 2 et en base 16. Leur syntaxe est la suivante :

```
<res> = bin$(<var>[,<prec>])
<res> = hex$(<var>[,<prec>])
```

où <var> est une donnée ou une variable entière,

<prec> est le nombre de chiffres souhaité pour la conversion,

et <res> est une variable chaîne qui contient le résultat de la conversion.

Remarque

Lorsque le paramètre <prec> n'est pas spécifié, la précision utilisée pour la conversion est la précision strictement nécessaire.

Exemple

```
10 A=32765
20 print hex$(A)
30 print hex$(A,8)
40 print bin$(A)
50 print bin$(A,32)
$7FFD
$00007FFD
%111111111111101
%00000000000000001111111111111101
```

Instructions opérant sur des bits

Tout comme dans le langage du microprocesseur, le STOS permet d'accéder à des informations binaires (bits).

Le Bit est l'information élémentaire (0 ou 1). L'octet (ou byte) est formé de 8 bits, le mot (ou WORD) de 16 bits et le mot long (ou LONG WORD) de 32 bits.

Les instructions bchg, bclr, bset (respectivement) changent, annulent et initialisent à 1 la valeur du bit spécifié dans la variable spécifiée. Leur syntaxe est :

```
bchg <bit>,<var>
bclr <bit>,<var>
bset <bit>,<var>
```

où <bit> est le numéro du bit à inverser/annuler/mettre à 1 dans <var>.

La fonction booléenne btst renvoie la valeur true si le bit spécifié est à 1 dans la variable spécifiée. Dans le cas contraitre, elle renvoie la valeur false. Sa syntaxe est :

```
<test> = btst(<bit>,<var>)
```

où <bit> est le numéro du bit à tester dans <var>,

et <test> est le résultat du test.

Les instructions rol et ror effectuent (respectivement) une rotation de N bits vers la gauche et vers la droite de la variable spécifiée. Leur syntaxe est la suivante :

```
rol.<f> <N>,<var>
ror.<f> <N>,<var>
```

où <f> est le format de la rotation (b pour byte, w pour word et l pour long word),

et <N> est le nombre de bits de rotation de <var>.

L'effet d'une rotation d'un bit vers la gauche d'une variable codée sur 8 bits apparaît dans le schéma ci-dessous :

```
Avant la rotation : 7 6 5 4 3 2 1 0
Après la rotation : 6 5 4 3 2 1 0 7
```

Exemples

```
100 A=%10101011 : B=A
110 print "A=";bin$(A)
120 bchg 2,A
130 print "Valeur de A après un bchg 2,A : ";bin$(A)
140 A=B : bclr 6,A
150 print "Valeur de A après un bclr 6,A : ";bin$(A)
160 A=B : bset 6,A
170 print "Valeur de A après un bset 6,A : ";bin$(A)
180 A=B
190 print "La fonction btst(2,A) renvoie la valeur ";btst(2,A)
200 print "La fonction btst(3,A) renvoie la valeur ";btst(3,A)
210 A=B : rol .b 3,A
```

```
220 print "Valeur de A après un rol.b 3,A : ";bin$(A)
230 A-B : ror .b 3,A
240 print "Valeur de A après un ror.b 3,A : ";bin$(A)
A-%10101011
Valeur de A après un bchg 2,A : %10101111
Valeur de A après un bclr 6,A : %10101011
Valeur de A après un bset 6,A : %11101011
La fonction btst(2,A) renvoie la valeur 0
La fonction btst(2,A) renvoie la valeur -1
Valeur de A après un bchg 2,A : %1011101
Valeur de A après un bchg 2,A : %1110101
```

Fonctions utilisateur

L'instruction def fn vous permet de définir une fonction mathématique. Sa syntaxe est la suivante :

```
def fn <nom>[(<var>)] = <exp>
```

où <nom> est le nom de la fonction,

<var> représente une ou plusieurs variables passées à la fonction,

<exp> est une expression mathématique utilisant les variables données et/ou les fonctions mathématiques du STOS et/ou d'autres fonctions utilisateur.

Une fonction utilisateur étant définie à l'aide de l'instruction def fn, il est possible d'y accéder avec fn.

Voici un exemple pour éclaircir le fonctionnement de def fn et de fn :

```
10 MA#=-1E+10 : MI#=1E+10
20 def fn UTIL (X)=sin(X)*cos(X)
30 for X#=0 to 2*pi step 0.05
40 Y#=fn UTIL (X#)
50 if Y#>MA# then MA#=Y#
60 if Y#<MI# then MI#=Y#
70 next X#
80 print "maximum : ";MA#
90 print "minimim : ";MI#
```

L'opérateur de mise à la puissance est (^). L'exposant peut être entier ou réel. Par exemple, tapez en mode direct :

```
print 2^8
256
print 1.4^3.25
2.98481
```

La fonction sqr permet d'extraire la racine carrée d'un nombre. Sa syntaxe est :

```
<res> = sqr(<nb>)
```

où <nb> est une expression ou un nombre positif,

et <res> est la racine carrée de <nb>.

Les fonctions exponentielle et logarithme (népérien et base 10) n'ont pas été oubliées. En voici la syntaxe :

```
<res> = exp(<nb>)
<res> = ln(<nb>)
<res> = log(<nb>)
```

où <nb> est le nombre ou l'expression dont vous désirez connaître l'exponentielle/le logarithme népérien/le logarithme décimal,

et <res> est le résultat renvoyé par la fonction.

Prenez bien garde de fournir un argument positif aux fonctions ln et log, sans quoi, le programme vous renverra une erreur "nombre négatif".

Si vous désirez connaître le plus grand (ou le plus petit) de deux nombres A et B, vous pouvez faire le test suivant :

```
if A>=B then C=A else C=B  --> pour extraire le plus grand
if A<B  then C=A else C=B  --> pour extraire le plus petit
```

ou encore utiliser les fonctions STOS max et min comme suit :

```
C=max(A,B)  --> pour extraire le plus grand
C=min(A,B)  --> pour extraire le plus petit
```

Si vous désirez échanger les valeurs contenues dans deux variables A et B, la méthode classique consiste à passer par une troisième variable :

```
C=A : A=B : B=C
```

Vous pouvez aussi utiliser l'instruction swap qui est bien plus pratique :

```
swap A,B
```

Pour extraire la partie entière d'un nombre réel, le BASIC STOS vous permet d'utiliser la fonction int dont la syntaxe est :

```
<ne> = int(<nr>)
```

où <nr> est une expression ou un nombre réel,

et <ne> est la partie entière de <nr>.

Que le nombre réel soit positif ou négatif, sa partie entière est obtenue en prenant l'entier inférieur ou égal le plus proche.

Exemples

```
print int(1.25)
1
print int(1.75)
1
print int(-1.25)
-2
print int(-1.75)
-2
```

L'instruction fix détermine le nombre de chiffres après la virgule des nombres réels. Sa syntaxe est :

```
fix(<n>)
```

où <n> est un nombre compris entre 1 et 7 pour une notation classique,

et <n> est un nombre compris entre -7 et -1 pour une notation exponentielle.

Lorsque le nombre de chiffres après la décimale est inférieur au nombre spécifié dans fix, il est complété par autant de 0 que nécessaire.

Exemples

```
fix(2) : print sqr(2),3.5
1.41    3.50
fix(5) : print sqr(2),3.5
1.41421    3.50000
fix(-2) : print sqr(2),3.5
1.41E+00    3.50E+00
fix(-5) : print sqr(2),3.5
1.41421E+00    3.50000E+00
```

3.3. Instructions dédiées aux chaînes

Les chaînes de caractères peuvent être composées d'un ou de plusieurs caractères (lettres, chiffres, signes). Les variables de type chaîne ont une longueur maximale de 65536 caractères. Leur nom se termine toujours par le signe $. Par exemple A$ ou CHAINE$ sont des variables chaîne.

Les opérateurs + et - permettent respectivement d'"additionner" et de "soustraire" deux chaînes. Supposons que les chaînes A$ et B$ contiennent les valeurs suivantes :

```
A$="Bonjour"
B$="jour"
```

Alors A$+B$ sera égal à

```
Bonjourjour
```

et A$-B$ sera égal à

```
Bon
```

En plus de ces opérations élémentaires, il est possible d'extraire :

- ✓ les N premiers caractères avec la fonction left$,
- ✓ les N derniers caractères avec la fonction right$,
- ✓ les N caractères situés à partir d'une position donnée avec la fonction mid$.

La syntaxe de ces fonctions est la suivante :

```
<res> = left$(<ch>,<len>)
<res> = right$(<ch>,<len>)
```

où <ch> est la chaîne source

et <len> le nombre de caractères à extraire.

```
<res>=mid$(<ch>,<deb>,<len>)
```

où <ch> est la chaîne source,

<deb> est la position du premier caractère à extraire,

et <len> est le nombre de caractères à extraire.

Le petit programme suivant montre comment utiliser ces fonctions :

```
10 A$="Opérations sur les chaînes"
20 print "Les 5 premiers caractères de A$ sont :";left$(A$,5)
30 print "Les 10 derniers caractères de A$ sont :";right$(A$,10)
40 print "Les caractères compris entre le 6ème et le 10ème
sont";mid$(A$,6,5)
Les 5 premiers caractères de A$ sont Opéra
Les 10 derniers caractères de A$ sont es chaînes
Les caractères compris entre le 6ème et le 10ème sont tions
```

Si vous désirez redéfinir tous les caractères d'une chaîne en majuscule ou en minuscule, vous pouvez utiliser les fonctions upper$ et lower$.

```
<res> = upper$(<ch>)
<res> = lower$(<ch>)
```

où <ch> est la chaîne à convertir,

et <res> la chaîne convertie.

Ces fonctions sont en particulier utiles pour éviter les tests multiples lorsque des données sont entrées au clavier. Par exemple :

```
100 input "Votre choix (O/N) :";a$
110 if upper$(A$)="O" then goto 1000 else goto 2000
```

Sans utiliser la fonction upper$, le test aurait été plus complexe :

```
100 input "Votre choix (O/N) :":a$
110 if (A$="O") or (A$="o") then goto 1000 else goto 2000
```

La longueur d'une chaîne est renvoyée par la fonction len dont la syntaxe est :

```
<res> = len(<ch>)
```

où <ch> est une chaîne de caractères,

et <res> le nombre de caractères de <ch>.

Par exemple, tapez :

```
A$="Test de len" : print len(A$)
11
```

Si vous désirez créer une chaîne composée uniquement d'espaces, vous pouvez utiliser la fonction space$ dont la syntaxe est :

```
<res> = space$(<N>)
```

où <N> est la longueur de la chaîne à créer,

et <res> est la chaîne résultante.

Dans le même ordre d'idées, la fonction string$ crée une chaîne composée de N fois le même caractère. Sa syntaxe est :

```
<res> = string$(<car>,<N>)
```

où <N> est la longueur de la chaîne à creér,

<car> est le caractère dont sera constituée la chaîne,

et <res> est la chaîne résultante.

Si vous désirez rechercher la présence d'une sous-chaîne dans une chaîne, la fonction instr est faite pour vous. En voici la syntaxe :

```
<pos> = instr(<ch>,<sch>)
```

où <ch> est la chaîne dans laquelle s'effectue la recherche,

<sch> est la chaîne recherchée,

et <pos> est la position de <sch> dans <ch>.

Si <sch> n'est pas trouvée dans <ch>, la fonction instr renvoie la valeur 0.

Par exemple :

```
print instr("coucou","cou")
1
print instr("coucou","coup")
0
```

La fonction flip$ inverse tous les caractères d'une chaîne donnée. Ainsi, si A$ contient la chaîne "Bonjour", flip$(A$) renverra la chaîne "ruojnoB".

Pour terminer les instructions et fonctions dédiées aux chaînes de caractères, voici deux fonctions bien utiles qui convertissent une chaîne de caractères en un nombre (val) et un nombre en une chaîne de caractères (str$). Leur format est le suivant :

```
<nb>=val(<ch>)
```

où <ch> est la chaîne à convertir,

et <nb> est la version numérique de <ch>.

```
<ch>=str$(<nb>)
```

où <nb> est le nombre ou l'expression numérique à convertir,

et <ch> est la version alphanumérique de <nb>.

Exemples

```
print val("azer")
 0
print val("12.65 francs")
 12.65
print "Le total est"+str$(650)
Le total est 650
```

Conclusion

Nous allons conclure ce chapitre par un jeu de réflexion qui utilise de nombreuses instructions d'ordre général. En voici les règles.

N lettres sont affichées sur l'écran. Le but du jeu est de classer ces lettres dans l'ordre alphabétique en un minimum de coups en opérant des retournements. Un retournement de P lettres consiste à inverser les P premières lettres affichées. Le nombre de retournements effectués est affiché en permanence. Le jeu prend automatiquement fin lorsque toutes les lettres sont bien classées.

Le listing du programme est le suivant :

```
1000 rem --
1010 rem Démonstration des instructions d'ordre général
1020 rem --
1025 dim T(26),T2(26)
1030 repeat
1040 gosub 2000 remChoix du nombre de lettres au départ
1050 gosub 3000 remInitialisation de la partie
1060 gosub 4000 remDéroulement d'une partie
1070 until BIS=0
1080 end
2000 rem - - - - - - - - - - - - - - - - - - - -
2010 rem Choix du nombre de lettres au départ
2020 rem - - - - - - - - - - - - - - - - - - - -
2030 cls
2040 print "Jeu du renversé"
2050 print ""
2060 print
2070 print "N lettres vont apparaître sur l'écran dans un ordre
quelconque."
2080 print "Le but du jeu est de classer ces lettres dans
l'ordre alphabétique"
2090 print "en un minimum de coups en opérant des retournements."
2100 print "Un retournement de N lettres consiste à inverser les
N premières"
2110 print "lettres affichées."
2120 print "Le nombre de retournements effectués est affiché en
permanence."
2130 print "Le jeu prend automatiquement fin lorsque toutes les
lettres sont"
2140 print "bien classées."
2150 print
2160 repeat
2170 input "Entrez le nombre de lettres en jeu (3 à 26) :";NL
```

```
2180 if (NL>26) or (NL<3) then print "entre 3 et 26 s.v.p."
2190 until (NL>-3) and (NL<-26)
2200 return
3000 rem - - - - - - - - - - - - - - -
3010 rem Initialisation de la partie
3020 rem - - - - - - - - - - - - - - -
3040 rem ....................................
3050 rem Initialisation du tableau des lettres
3060 rem ....................................
3070 for I=1 to NL
3080 T(I)=I
3090 next I
3100 rem ..................
3110 rem Mélange du tableau
3120 rem ..................
3130 for I=1 to 40*NL
3140 A=int(rnd(NL-1))+1 : B=int(rnd(NL-1))+1
3150 swap T(A),T(B)
3160 next I
3170 return
4000 rem - - - - - - - - - - - - - -
4010 rem Déroulement d'une partie
4020 rem - - - - - - - - - - - - - -
4030 cls
4040 COUPS=0 remNombre de coups joués
4050 print "Jeu du renversé"
4060 print ""
4070 repeat
4080 BP=0 remNombre de lettres bien placées
4090 for I=1 to NL
4100 if T(I)=I then BP=BP+1
4110 next I
4120 locate 1,6
4130 print "Nombre de lettres bien placées :";BP
4140 locate 1,10
4150 for I=1 to NL
4160 print chr$(T(I)+64);" ";
4170 next I
4180 repeat
4190 locate 1,15
4200 print "Nombre de lettres à retourner (2 à";NL;") : ";
4210 input NR
4220 if (NR<2) or (NR>NL) then print "entre 2 et";NL;" s.v.p."
else locate 1,16 : print space$(40)
4230 locate 1,15 : print space$(60)
4240 until (NR>-2) and (NR<-NL)
4250 rem ............
4260 rem Retournement
4270 rem ............
4280 for I=1 to NR
```

```
4290 T2(NR-I+1)=T(I)
4300 next I
4310 for I=1 to NR
4320 T(I)=T2(I)
4330 next I
4340 rem ......................
4350 rem Test de fin de partie
4360 rem ......................
4370 BP=0 remNombre de lettres bien placées
4380 for I=1 to NL
4390 if T(I)=I then BP=BP+1
4400 next I
4410 COUPS=COUPS+1
4420 until BP=NL
4430 cls
4440 print "Partie terminée en";COUPS;" coup(s)."
4450 print
4460 input "Une autre partie (O/N) : ";R$
4470 if upper$(R$)="O" then BIS=1 else BIS=0
4480 return
Le programme principal occupe les lignes 1000 à 1080. Il
consiste en l'activation séquentielle des sous programmes situés
aux lignes 2000 (choix du nombre de lettres), 3000
(initialisation d'une partie) et 4000 (partie).
```

Le sous-programme permettant de choisir le nombre de lettres affiche la règle du jeu (lignes 2030 à 2140) et stocke le nombre de lettres à classer dans la variable NL (lignes 2160 à 2190).

Le sous-programme d'initialisation stocke dans le tableau T les NL lettres du jeu (lignes 3070 à 3090) et les mélange (lignes 3130 à 3160).

Le sous-programme responsable du déroulement d'une partie débute par un test des lettres bien placées (lignes 4080 à 4110). Il demande ensuite au joueur d'entrer le nombre de lettres à retourner NR (lignes 4190 à 4240). Les NR lettres demandées sont inversées dans le tableau T (lignes 4280 à 4330). Le sous programme se termine par un test de fin de partie. Le nombre de lettres bien placées est comptabilisé (lignes 4370 à 4400). S'il est égal au nombre de lettres en jeu (NL), le nombre de coups nécessaires à l'achèvement de la partie est affiché (ligne 4440) et le programme demande au joueur s'il désire faire une autre partie (lignes 4460 et 4470).

Chapitre 4

Organes de sortie

4.1. L'écran

Dans cette section, nous allons étudier les diverses instructions, fonctions et variables système liées à l'écran. Nous les répartirons en cinq groupes :

❶ ***Instructions texte*** :
utilisation du curseur texte et affichage de données alphanumériques sur l'écran.

❷ ***Instructions graphiques*** :
tracé de points, droites, rectangles, cercles, etc..

❸ ***Manipulation de l'écran*** :
accès direct aux écrans, zoom, réduction, déplacements de blocs, effets spéciaux, etc..

❹ ***Fenêtres*** :
création et utilisation des fenêtres et des jeux de caractères.

❺ *Menus* :
création et utilisation des menus texte.

4.1.1. Instructions texte

Toutes les instructions relatives à l'affichage de données alphanumériques opèrent à partir de la position courante du curseur. Avant de parler des instructions d'affichage, nous allons voir comment manipuler le curseur.

L'instruction home positionne le curseur en haut et à gauche de l'écran ou de la fenêtre courante.

Comme leur nom l'indique, les instructions cdown, cup, cright et cleft déplacent le curseur d'une position vers (respectivement) le bas, le haut, la droite et la gauche de l'écran.

La position courante du curseur sur l'écran se trouve en permanence dans les variables système :

✓ xcurs : abscisse du curseur comprise entre 0 et 39 en mode 0, et entre 0 et 79 en modes 1 et 2.

✓ ycurs : ordonnée du curseur comprise entre 0 et 24.

Le curseur peut être positionné sur l'écran à l'aide de l'instruction locate dont la syntaxe est la suivante :

```
locate <x>,<y>
```

où <x> est l'abscisse du curseur (entre 0 et 39 en mode 0, entre 0 et 79 en modes 1 et 2),

et <y> l'ordonnée du curseur (entre 0 et 24).

Comme leur nom l'indique, les instructions curs on et curs off permettent d'activer et de désactiver l'affichage du curseur sur l'écran.

Enfin, l'instruction set curs modifie la taille du curseur. Le curseur s'inscrit dans une matrice de 8x8 pixels. L'instruction set curs permet de définir le nombre de lignes "allumées" dans cette matrice. Sa syntaxe est la suivante :

```
set curs <H>,<B>
```

où <H> est le numéro de la ligne la plus haute allumée (entre 0 et 8),

et <B> est le numéro de la ligne la plus basse allumée (entre 0 et 8).

Par défaut, les deux dernières lignes du curseur sont allumées, ce qui correspond à l'instruction set curs 6,8. Si vous désirez que le curseur soit à sa taille maximum, tapez set curs 0,8.

Vous savez maintenant comment manipuler le curseur sur l'écran. Passons aux instructions relatives à l'affichage alphanumérique.

La première instruction, que vous connaissez déjà (cf Partie 2, Chap 3) est l'instruction print. Elle vous permet d'afficher du texte ou le contenu de variables, de manière formatée ou littérale. Reportez-vous au chapitre 3 pour avoir plus de détail à son sujet.

L'écran peut être considéré comme une feuille de papier sur laquelle on écrit en utilisant un stylo. La couleur de cette feuille peut être choisie à l'aide de l'instruction paper. Celle du stylo à l'aide de l'instruction pen. La syntaxe de ces instructions est la suivante :

```
paper <C>
pen <C>
```

où <C> est la couleur à affecter au papier ou au stylo. Elle doit être comprise entre 0 et 15 en mode 0, entre 0 et 3 en mode 1, et égale à 0 ou à 1 en mode 2 et correspond aux couleurs de la palette courante.

Les quantités de rouge, vert et bleu affectées à une des couleurs de la palette peuvent être définies à l'aide de l'instruction colour dont la syntaxe est la suivante :

```
colour <C>,<$RVB>
```

où <C> est le numéro de la couleur à définir,

et <$RVB> est un nombre hexadécimal qui exprime les quantités de rouge, de vert et de bleu affectées à la couleur <C>. Les composantes R, V et B doivent être comprises entre 0 et 7.

L'instruction colour peut également être utilisée en tant que fonction. Dans ce cas, elle renvoie une valeur numérique qui représente (en hexadécimal) les quantités de rouge, de vert et de bleu affectées à une couleur. La syntaxe de la fonction colour est la suivante :

```
<res>=colour(<C>)
```

où <C> est le numéro de la couleur à inspecter,

et <res> est le dosage RVB de cette couleur.

Le petit programme ci-dessous affiche la composante RVB des 16 couleurs courantes de la palette :

```
10 for I=0 to 15
20 print I,hex$(colour(I))
30 next I
```

Reportez-vous également à l'instruction palette qui permet de définir globalement les 16 couleurs de la palette.

Dans certains cas, il peut être intéressant de lire le caractère qui se trouve à une position donnée de l'écran. La fonction scrn renvoie le code ASCII du caractère qui occupe les coordonnées d'écran qui lui sont données. Sa syntaxe est la suivante :

```
<res>=scrn(<x>,<y>)
```

où <x> et <y> sont les coordonnées du caractère à lire,

et <res> est le code ASCII du caractère lu.

L'affichage d'un caractère en ligne L, colonne C sur l'écran provoque (par défaut) l'effacement de l'ancien caractère qui se trouvait à cette position. L'instruction writing permet de modifier ce mode par défaut. Sa syntaxe est :

```
writing <Mode>
```

où <Mode> vaut 1 pour le mode remplacement (mode par défaut),

- 2 pour le mode OR,
- 3 pour le mode XOR.

Lorsque les modes OR ou XOR sont actifs, chacun des points élémentaires qui constituent le caractère affiché subissent une opération OU logique (OR) ou OU EXCLUSIF logique (XOR) avec les points élémentaires qui occupent la même position au moment de l'affichage.

Pour bien comprendre le fonctionnement de cette instruction, exécutez le petit programme suivant :

```
10 mode 0
20 print "writing 1 : mode remplacement"
30 writing 1
40 locate 1,5 : print "bonjour"
50 locate 1,5 : print "coucou"
60 wait key
70 cls
80 print "writing 2 : mode OR"
90 writing 2
100 locate 1,5 : print "bonjour"
110 locate 1,5 : print "coucou"
120 wait key
130 cls
140 print "writing 3 : mode XOR"
150 writing 3
160 locate 1,5 : print "bonjour"
170 locate 1,5 : print "bonjour"
180 writing 1
```

Nous allons terminer les instructions liées à la gestion de l'écran "texte" par les attributs d'affichage.

L'instruction inverse permet d'afficher des caractères en inverse vidéo. Lorsqu'elle est suivie du mot clé "on", les couleurs du texte et du fond sont inversées. Lorsqu'elle est suivie du mot clé "off", l'affichage redevient normal. Sa syntaxe est :

```
inverse {on|off}
```

L'instruction under permet d'afficher des caractères soulignés. Lorsqu'elle est suivie du mot clé "on", les caractères apparaissent soulignés sur l'écran. Lorsqu'elle est suivie du mot clé "off", l'affichage redevient normal. Sa syntaxe est :

```
under {on|off}
```

L'instruction shade permet d'afficher des caractères en relief. Lorsqu'elle est suivie du mot clé "on", les caractères sont affichés en relief. Lorsqu'elle est suivie du mot clé "off", l'affichage redevient normal. Sa syntaxe est :

```
shade {on|off}
```

Conclusion

Pour bien assimiler les instructions qui viennent d'être décrites, je vous propose un petit programme. Il s'agit d'un jeu de pendu. Un mot tiré au hasard dans une liste doit être découvert par le joueur. Pour cela, il propose des lettres de l'alphabet. Lorsqu'elles existent dans le mot à découvrir, elles sont affichées sur l'écran. Le but du jeu est évidemment de découvrir le mot en proposant le moins de lettres possible.

Le listing du programme est le suivant :

```
100 rem ---
110 rem Jeu du pendu
120 rem ---
130 rem
140 cls
150 dim T$(20)
160 for I=1 to 20
170 read T$(I)
180 next I
190 for I=1 to 200
200 J=int(rnd(19))+1
210 K=int(rnd(19))+1
220 swap T$(J),T$(K)
230 next I
240 MAT=1 : rem Premier mot à trouver
250 TAMP$=T$(MAT) : rem Variable tampon
260 UTIL$=space$(len(TAMP$))
270 print "Jeu du pendu"
```

```
280 print "--"
290 print
300 locate 10,10
310 for I=1 to len(T$(MAT))
320 print "- ";
330 next I
340 COUPS=0
350 repeat
360 locate 1,15
370 input "Entrez une lettre : ":L$
380 L$=upper$(L$)
390 if instr(TAMP$,L$)=0 then goto 470
400 repeat
410 J=instr(TAMP$,L$)
420 if J<>0 then locate 8+J*2,10 : print L$
430 if J<>0 then
UTIL$=left$(UTIL$,J-1)+L$+right$(UTIL$,len(UTIL$)-J)
440 if J<>0 then TAMP$=space$(J)+right$(TAMP$,len(TAMP$)-J)
450 until J=0
460 TAMP$=T$(MAT)
470 COUPS=COUPS+1
480 rem
490 until TAMP$=UTIL$
500 cls
510 print "Le mot ";TAMP$;" a été trouvé en";COUPS;" coups."
520 data "ORDINATEUR","PLANTATION","CLASSEUR","AQUARIUM",
"VEGETATION"
530 data "DEMENAGEMENT","DEPLACEMENT","REPONDEUR","VERDURE",
"CROCODILE"
540 data "SINISTRE","INFORMATION","PLUSIEURS","DISQUETTE","SALON"
550 data "TABULATION","TELEVISEUR","SECTION","CALME","MARTEAU"
```

Après son dimensionnement (ligne 150), le tableau T$ est initialisé (lignes 160 à 180) puis mélangé (lignes 190 à 230).

La variable TAMP$ contient le mot à trouver (ligne 250). La variable UTIL$ contient les lettres proposées par le joueur, positionnées dans le mot à découvrir (ligne 260).

Les lettres à découvrir apparaissent sur l'écran sous la forme de tirets (lignes 300 à 330) à partir de la position 10,10. Le joueur doit proposer répétitivement des lettres (ligne 370) jusqu'à la découverte du mot (ligne 490).

Lorsque la lettre proposée se trouve dans le mot à découvrir, elle est affichée sur l'écran (ligne 420) et mémorisée dans UTIL$ (ligne 430).

Lorsque le mot à découvrir est trouvé, le nombre de lettres proposées est affiché (ligne 510).

Les lignes 520 à 550 contiennent la liste des mots à découvrir.

4.1.2. Instructions graphiques

L'écran graphique de l'ATARI peut être représenté par une matrice de points dont le nombre dépend du mode d'affichage : 320x200 en mode 0, 640x200 en mode 1, 640x400 en mode 2. Le plus petit élément graphique accessible sur l'écran s'appelle un pixel.

Avant de passer en revue les instructions graphiques, nous allons nous intéresser à l'instruction ink qui définit la couleur utilisée par défaut par toutes les instructions de tracé. Sa syntaxe est :

```
ink <Coul>
```

où <Coul> est la couleur qui sera utilisée par les instructions graphiques.

Accès au pixel

L'instruction plot permet d'affecter une couleur à un pixel de l'écran. Sa syntaxe est la suivante :

```
plot <X>,<Y>[,<C>]
```

où <X> et <Y> sont les coordonnées du pixel à initialiser,

et <C> est la couleur à affecter au pixel de coordonnées <X>,<Y>. Si le paramètre <C> n'est pas spécifié, la couleur affectée au point sera celle qui a été fixée par l'instruction ink.

Inversement, la fonction point renvoie la couleur du point dont les coordonnées lui sont passées. Sa syntaxe est la suivante :

```
<Coul>=point(<X>,<Y>)
```

où <X> et <Y> sont les coordonnées du point dont vous voulez connaître la couleur,

et <Coul> est la couleur du point de coordonnées <X>,<Y>.

Tracé de lignes, rectangles et polygones

L'instruction de base pour tracer une ligne droite est draw. Cette instruction admet deux variantes :

L'instruction

```
draw <x1>,<y1> to <x2>,<y2>
```

où <x1> et <x2> sont les coordonnées d'une des extrémités de la droite, et <x2> et <y2> les coordonnées de l'autre extrémité de la droite, trace une droite entre les points de coordonnées <x1>,<y1> et <x2>,<y2>.

L'instruction

```
draw to <x2>,<y2>
```

où <x2> et <y2> sont les coordonnées de la seconde extrémité de la droite, trace une droite entre la position courante du curseur graphique et le point de coordonnées <x2>,<y2>.

Pour concrétiser ce qui vient d'être dit sur draw, essayez le petit programme suivant :

```
10 cls
20 for I=0 to 10
30 draw 50,50+I*10 to 50+I*20,150
40 next I
```

Si vous désirez tracer des rectangles, vous pouvez bien sûr utiliser draw, mais plusieurs instructions spécialement destinées au tracé de rectangles ont été implémentées dans le STOS. Ces instructions rendent évidemment le tracé plus simple. En voici la liste :

- ✓ box trace un rectangle vide,
- ✓ rbox trace un rectangle vide aux coins arrondis,
- ✓ bar trace un rectangle plein,

✓ rbar trace un rectangle plein aux coins arrondis.

Ces instructions ont toutes la même syntaxe. Par exemple pour rbox :

```
rbox <x1>,<y1> to <x2>,<y2>
```

où <x1> et <x2> sont les coordonnées d'un des coins du rectangle,

et <x2> et <y2> sont les coordonnées du coin opposé du rectangle.

Si vous en éprouvez le besoin, voici un petit programme pour vous aider à assimiler l'utilisation des instructions de tracé de rectangles :

```
10 cls
20 for I=1 to 30
30 box 10,50 to 10+4*I,50+4*I
40 box 300,50 to 300-4*I,50+4*I
50 next I
```

Maintenant, vous savez tracer des lignes et des rectangles vides ou pleins. Le STOS vous permet également de tracer des polygones vides ou pleins en une seule instruction :

✓ polyline trace un polygone vide,

✓ polygon trace un polygone plein.

La syntaxe de ces instructions est la suivante :

```
polyline <x1>,<y1> to <x2>,<y2> to ... to <xN>,<yN>
polygon <x1>,<y1> to <x2>,<y2> to ... to <xN>,<yN>
```

où les <xI> et <yI> sont les coordonnées des coins du polygone.

Voici un exemple d'utilisation de l'instruction polyline :

```
10 cls
20 for I=1 to 6
30 polyline 100,50+I*4 to 195-8*I,75 to 195-8*I,125
to 100,150-4*I to 5+8*I,125 to 5+8*I,75 to 100,50+4*I
40 next I
```

Tracé de cercles, arcs de cercles, ellipses et camemberts

Comme vous vous en doutez, l'instruction arc permet de tracer des arcs de cercles. Sa syntaxe est la suivante :

```
arc <x>,<y>,<r>,<dep>,<fin>
```

où <x> et <y> sont les coordonnées du centre de l'arc de cercle,

<r> est le rayon de l'arc de cercle,

<dep> est l'angle de départ de l'arc de cercle en degrés multipliés par dix,

<fin> est l'angle de fin de l'arc de cercle en degrés multipliés par dix.

Un angle égal à 0 correspond à un point situé à droite du centre. Un angle égal à 900 correspond à un point situé au-dessus du centre. Un angle égal à 1800 correspond à un point situé à gauche du centre. Enfin, un angle égal à 2700 correspond à un point situé au-dessous du centre.

Si vous désirez afficher des cercles pleins, vous devrez utiliser l'instruction circle dont la syntaxe est :

```
circle <x>,<y>,<r>
```

où <x> et <y> sont les coordonnées du centre du cercle,

et <r> est le rayon du cercle.

Le petit programme ci-dessous montre comment utiliser les instructions arc et circle :

```
10 cls
20 for I=1 to 20
30 arc 30+I*3,30+I*3,30,0,3600
40 circle 200-I*3, 30+I*3,30
50 next I
```

Dans la suite logique des instructions arc et circle, nous trouvons les instructions earc et ellipse qui permettent de tracer (respectivement) des arcs d'ellipses et des ellipses pleines. Leur syntaxe est la suivante :

```
earc <x>,<y>,<r1>,<r2>,<dep>,<fin>
```

où <x> et <y> sont les coordonnées du centre de l'arc d'ellipse,

<r1> est le premier rayon de l'ellipse,

<r2> est le second rayon de l'ellipse,

<deb> est l'angle de départ de l'arc d'ellipse en degrés multipliés par dix,

<fin> est l'angle de fin de l'arc d'ellipse en degrés multipliés par dix.

```
ellipse <x>,<y>,<r1>,<r2>
```

où <x> et <y> sont les coordonnées du centre de l'ellipse,

<r1> est le premier rayon de l'ellipse,

<r2> est le second rayon de l'ellipse.

Enfin, les instructions pie et epie permettent de tracer (respectivement) une portion de camembert pleine et une portion de camembert elliptique pleine. Leur syntaxe est la même que celle des instructions (respectivement) arc et earc.

Tramage des lignes et des surfaces

Les objets tracés avec les instructions arc, earc, box, rbox et polyline sont par défaut composés de lignes et courbes continues. L'instruction set line permet de définir un tramage de ces lignes :

```
set line <masque>,<épaisseur>,<deb>,<fin>
```

où <masque> est un nombre compris entre 0 et 65535 dont chaque bit à zéro représente un point éteint et chaque bit à un représente un bit allumé. Par défaut, le masque est égal à %1111111111111111. Un

masque égal à %1010101010101010 produit des droites composées de pointillés serrés, et un masque égal à %1111000011110000 produit des droites composées de pointillés nettement plus larges,

<épaisseur> est l'épaisseur du trait qui doit être comprise entre 1 (très mince) et 40 (très épais),

<deb> est la forme de la première extrémité de la droite. 0 représente une forme carrée, 1 une flèche et 2 une forme arrondie.

<fin> est la forme de la deuxième extrémité de la droite. 0 représente une forme carrée, 1 une flèche et 2 une forme arrondie.

Exemple

```
10 mode 1
20 set line %1100110011001100,1,0,1
30 box 100,100 to 150,150
40 set line %1111111100000000,2,1,1
50 polyline 300,100 to 350,150 to 250,150
60 set line %1111000011110000,10,0,2
70 arc 500,125,50,0,2700
```

La dernière instruction examinée est set paint. Elle permet de choisir un motif de remplissage pour les instructions circle, ellipse, bar, rbar, pie, epie et polygon. Sa syntaxe est :

```
set paint <type>,<motif>,<bord>
```

où <type> est le type de remplissage souhaité :

0 pour ne pas remplir la surface,
1 pour remplir la surface avec la couleur ink,
2 pour remplir la surface avec un motif à base de points
3 pour remplir la surface avec un motif à base de hachures

<motif> est le numéro du motif choisi,

Il existe 24 motifs de points et 12 motifs de hachures prédéfinis.

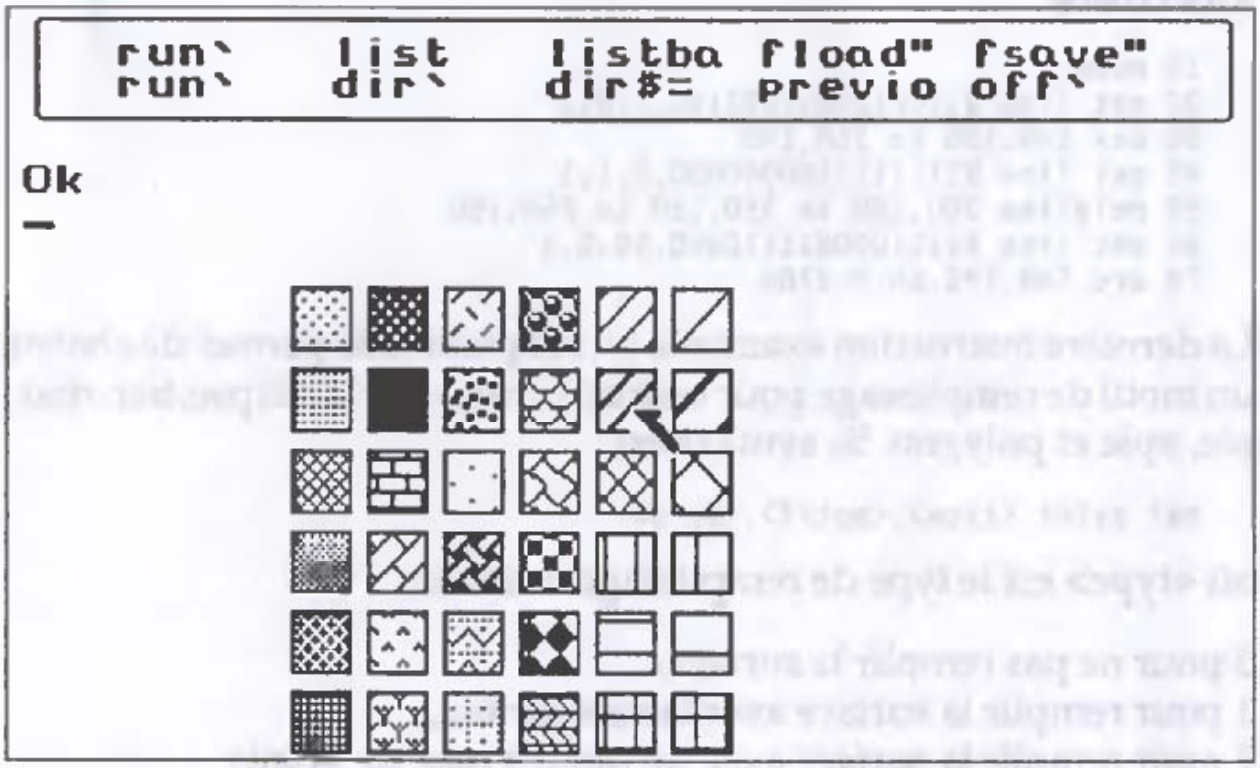
et <bord> détermine si la surface doit (1) ou ne doit pas (0) être entourée par une ligne de couleur ink.

Exemple

Le petit programme suivant donne une idée des possibilités de l'instruction set paint.

```
10 mode 0
20 for I=1 to 6
30 for J=1 to 6
40 if ((J-1)*6+I)<25 then set paint 2,(J-1)*6+I,1 else set paint
3,(J-1)*6+I-24,1
50 bar 50+J*20,50+I*20 to 50+J*20+15,50+I*20+15
60 next j
70 next i
```

Ce programme produit l'affichage suivant :



Conclusion

Nous allons terminer ce paragraphe sur un petit programme qui montre une des utilisations possibles des instructions ink et plot. Il s'agit d'un télécran. Les touches numériques (0 à 9) permettent de choisir la couleur de tracé. Les touches flèches donnent la direction du tracé sur l'écran.

Voici le listing du programme :

```
100 rem ----
110 rem Telecran
120 rem ----
130 X=100 : Y=100 : rem Coordonnées au départ
140 mode 0
150 repeat
160 repeat
170 K$=inkey$
180 until K$<>""
190 if asc(K$)=0 then 210
200 if (asc(K$)>47) and (asc(K$)<58) then ink asc(K$)-48 : goto
270
210 SC=scancode
220 if SC=72 then Y=Y-1
230 if SC=80 then Y=Y+1
240 if SC=75 then X=X-1
250 if SC=77 then X=X+1
260 plot X,Y
270 until asc(K$)=27
```

Au début du programme, le curseur est positionné aux coordonnées 100,100 (ligne 130). Le programme se met en attente d'une frappe au clavier (lignes 160 à 180). Si la touche frappée est un chiffre, la couleur graphique courante est modifiée (ligne 200). S'il s'agit d'une des touches flèches, la position du curseur graphique est modifiée en conséquence, ce qui provoque un tracé sur l'écran (lignes 220 à 260). Le programme prend fin lorsque vous appuyez sur la touche Escape (ligne 270).

4.1.3. Manipulation de l'écran

Jusqu'ici, nous avons parlé des instructions d'affichage texte et graphique, mais le STOS comporte également quelques instructions puissantes qui permettent de manipuler l'écran.

Avant de les passer en revue, vous devez savoir que, dès son activation, le STOS manipule trois écrans appelés l'écran logique (logic), l'écran physique (physic) et le décor (back).

Les écrans physique et logique contiennent la même image, sauf si vous spécifiez le contraire. Seul l'écran physique est visible. Cela vous permet entre autres de travailler sur l'écran logique et de l'afficher instantanément lorsque vous le désirez en utilisant l'instruction screen swap.

L'écran physique et le décor contiennent la plupart du temps la même image, excepté lorsque vous utilisez des sprites. En effet, les sprites n'apparaissent que sur l'écran physique. Le STOS se sert du décor pour restituer la partie décor qui se trouvait sous un sprite lors de son déplacement ou de son animation.

Chargement d'un écran en mémoire et utilisation de ses couleurs

Si vous désirez charger plusieurs écrans en mémoire, une des possibilités offertes par le STOS consiste à les stocker dans des banques mémoire.

Pour stocker un écran dans une banque, il faut la réserver de manière temporaire avec l'instruction reserve as screen ou de manière permanente avec l'instruction reserve as datascreen. Pour cela exécutez une des instructions suivantes :

```
reserve as screen <n>
reserve as datascreen <n>
```

où <n> est un numéro de banque compris entre 1 et 15.

La différence entre ces deux instructions est la suivante :

✓ tout écran réservé avec reserve as datascreen sera sauvegardé avec le programme,

✓ tout écran réservé avec reserve as screen ne sera pas sauvegardé avec le programme.

Lorsque vous avez réservé une banque, vous pouvez charger une image avec l'instruction load :

```
load "<Nom>.NEO",<écran> pour une image Néochrome,
load "<Nom>.PI1",<écran> pour une image Degas basse résolution,
load "<Nom>.PI2",<écran> pour une image Degas moyenne résolution,
load "<Nom>.PI3",<écran> pour une image Degas haute résolution.
```

<écran> représente le numéro d'une banque précédemment réservée.

Par exemple, les instructions suivantes réservent un écran temporaire dans la banque 7 et y chargent une image Degas basse résolution de nom "image.pi1" :

```
reserve as screen 7
load "image.pi1",7
```

Vous vous rappelez certainement des instructions palette et colour qui permettent de définir les couleurs de la palette courante. Nous allons compléter ces instructions par get palette qui initialise la palette courante à partir des couleurs d'un écran stocké dans une des banques. Sa syntaxe est :

```
get palette <n>
```

où <n> est une banque écran.

Effacement d'un écran

Si vous connaissez un autre BASIC que le STOS, l'instruction CLS ne vous est certainement pas inconnue. Mais, dans le STOS, cette instruction est bien plus complète que celles que vous avez pu rencontrer jusque-là. Elle permet d'effacer totalement ou partiellement un des écrans en mémoire, en lui affectant ou non une couleur. Sa syntaxe admet trois variantes :

```
cls [<écran>]
```

où <écran> est une adresse d'écran (back, physic ou logic) ou un numéro de banque dans laquelle se trouve un écran. Si le paramètre <écran> n'est pas spécifié, les écrans logique, physique et décor sont effacés.

```
cls <écran>,<couleur>
```

où <écran> est une adresse d'écran (back, physic ou logic) ou un numéro de banque dans laquelle se trouve un écran,

et <couleur> est la couleur d'effacement.

```
cls <écran>,<couleur>,<<x1>,<y1> to <x2>,<y2>
```

où <écran> est une adresse d'écran (back, physic ou logic) ou un numéro de banque dans laquelle se trouve un écran,

<couleur> est la couleur d'effacement,

<x1> et <y1> sont les coordonnées du coin supérieur gauche du rectangle à effacer,

et <x2> et <y2> sont les coordonnées du coin inférieur droit du rectangle à effacer.

Scrolling d'écran

L'écran ou une partie de l'écran peut être "scrollé" (déplacé) vers le haut, le bas, la droite ou la gauche. Cela permet par exemple d'écrire un texte sur l'écran en le faisant défiler, comme dans la présentation des acteurs d'un film.

Le scrolling d'un écran ou d'une partie d'écran se fait en deux temps. Dans un premier temps, vous devez définir la zone de l'écran à scroller et la direction du scrolling en utilisant l'instruction def scroll:

```
def scroll <No>,<x1>,<y1> to <x2>,<y2>,<dirx>,<diry>
```

où <No> est un numéro de zone compris entre 1 et 16,

<x1> et <y1> sont les coordonnées du coin supérieur gauche du rectangle à scroller,

<x2> et <y2> sont les coordonnées du coin inféreur droit du rectangle à scroller,

<dirx> indique le nombre de pixels du scrolling sur l'axe horizontal. Les nombres positifs produisent un scrolling vers la droite, et les nombres négatifs un scrolling vers la gauche.

<diry> indique le nombre de pixels du scrolling sur l'axe vertical. Les nombres positifs produisent un scrolling vers le bas, et les nombres négatifs un scrolling vers le haut.

Lorsqu'une zone de scrolling est définie, vous devez utiliser l'instruction scroll pour produire le scrolling. Le seul paramètre de l'instruction scroll est le numéro de la zone à scroller.

Exemple

```
10 mode 1
20 def scroll 1,1,1 to 300,35,1,0
30 for I=1 to 300
40 scroll 1
50 next I
```

Zoom, réduction et copie de portions d'écran

Si vous n'êtes pas encore conscient de la puissance du BASIC STOS, utilisez les instructions zoom, reduce et screen copy. Vous serez vite charmé.

Comme son nom l'indique, l'instruction zoom permet d'agrandir une portion de l'écran. Sa syntaxe est :

```
zoom <écran1>,<x1>,<y1>,<x2>,<y2> to
[<écran2>,]<x3>,<y3>,<x4>,<y4>
```

où <écran1> est l'adresse de l'écran source (back, physic ou logic) ou un numéro de banque dans laquelle se trouve l'écran source,

<x1> et <y1> sont les coordonnées du coin supérieur gauche du rectangle à agrandir,

<x2> et <y2> sont les coordonnées du coin inférieur droit du rectangle à agrandir,

<écran2>, s'il est spécifié est l'adresse ou le numéro de banque de l'écran dans lequel doit se faire l'agrandissement. S'il n'est pas spécifié, l'agrandissement se fait dans l'écran physique,

<x3> et <y3> sont les coordonnées du coin supérieur gauche du rectangle résultant,

<x4> et <y4> sont les coordonnées du coin inférieur droit du rectangle résultant.

Exemple

Introduisez une disquette qui comporte une image au format PI1 dans le lecteur A:. Supposons que cette image s'appelle IMAGE.PI1. Exécutez les instructions suivantes, en mode direct :

```
mode 0:load "image.pi1":zoom physic,1,1,30,30 to 100,1,300,150
```

Le rectangle situé entre les coordonnées 1,1 et 30,30 est agrandi sur le même écran dans le rectangle situé entre les coordonnées 100,1 et 300,150.

L'inverse de l'instruction zoom est reduce qui permet de réduire un écran selon les proportions spécifiées. Sa syntaxe est :

```
reduce <écran1> to [<écran2>,]<x1>,<y1>,<x2>,<y2>
```

où <écran1> est l'adresse de l'écran à réduire (back, physic ou logic) ou un numéro de banque dans laquelle se trouve l'écran à réduire,

<écran2> est l'écran dans lequel doit s'effecter la réduction. S'il n'est pas spécifié, la réduction s'effectue dans l'écran physique,

<x1> et <y1> sont les coordonnées du coin supérieur gauche du rectangle résultant,

<x2> et <y2> sont les coordonnées du coin inférieur droit du rectangle résultant.

Exemple

Introduisez une disquette qui comporte une image au format PI1 dans le lecteur A:. Supposons que cette image s'appelle IMAGE.PI1. Exécutez les instructions suivantes, en mode direct :

```
mode 0:load "image.pi1":reduce physic to 1,1,100,100
```

Après l'exécution de ces instructions, l'écran contient l'image initiale sur laquelle se superpose sa réduction entre les coordonnées 1,1 et 100,100.

La dernière instruction examinée dans ce paragraphe permet de recopier un écran dans un autre ou une portion d'écran dans une autre. Il s'agit de l'instruction screen copy qui admet deux variantes :

```
screen copy <écran1> to <écran2>
```

recopie <écran1> dans <écran2>

```
screen copy <écran1>,<x1>,<y1>,<x2>,<y2> to <ecran2>,<x3>,<y3>
```

où <écran1> est le nom de l'écran source,

<x1>, <y1>, <x2> et <y2> délimitent une surface rectangulaire dans <écran1>

<écran2> est le nom de l'écran destination,

<x3> et <y3> sont les coordonnées du coin supérieur gauche du rectangle destination.

Exemple

Introduisez une disquette qui comporte une image au format PI1 dans le lecteur A:. Supposons que cette image s'appelle IMAGE.PI1. Entrez les instructions suivantes, en mode direct, et déplacez la souris sur l'écran. L'image stockée dans le décor apparaîtra petit à petit :

```
reserve as screen 7
load"image.pi1",7
screen copy 7 to back
```

Effets spéciaux

Nous allons terminer notre tour d'horizon des instructions dédiées à l'écran sur deux instructions remarquables.

L'instruction appear fait apparaître de manière progressive une image sur un écran. Sa syntaxe est :

```
appear <x>[,<y>]
```

où <x> est le numéro d'une banque dans laquelle se trouve un écran,

et <y> est un nombre compris entre 1 et 79 qui spécifie le type d'apparition du nouvel écran.

Introduisez une disquette qui contient deux images au format PI1 dans le lecteur A:. Supposons que ces images s'appellent IMAGE1.PI1 et IMAGE2.PI1. Tapez :

```
10 mode 0
20 reserve as screen 7
30 load"image1.pi1",7
40 load"image2.pi1"
50 appear 7,1
```

L'instruction fade modifie progressivement une ou plusieurs couleurs. Elle admet trois variantes :

L'instruction

```
fade <vitesse>
```

fait disparaître une image de l'écran. <vitesse> représente le nombre de balayages d'écran avant de modifier les couleurs. Plus <vitesse> est grand, plus la disparition de l'image est lente.

L'instruction

```
fade <vitesse> to <banque>
```

fait passer progressivement des couleurs courantes aux couleurs de la banque spécifiée.

Enfin, l'instruction

```
fade <vitesse>,<c1>[,<c2>...[,<cN>]...]
```

remplace la ou les couleurs de rang 1, 2, N par les couleurs <c1>, <c2>, ..., <cN> à la vitesse <vitesse>.

Conclusion

Nous avons fait le tour des instructions dédiées à la manipulation de l'écran . Vous savez désormais qu'il existe trois types d'écran dont un seul est visible (physic). Cet écran peut être effacé (cls), scrollé (SCROLL).... Vous connaissez l'existence et le fonctionnement de l'instruction screen copy qui vous permet de copier un écran ou une partie d'écran dans un autre écran. Vous savez également faire des zooms ou des réductions d'image en utilisant les instructions zoom et reduce. Enfin, vous connaissez deux instructions intéressantes pour enchaîner progressivement deux images ou modifier les couleurs d'une image.

4.1.4. Fenêtres et jeux de caractères

Si vous désirez afficher plusieurs types de données sur l'écran, par exemple des graphiques et des textes, ou encore si vous désirez avoir accès à plusieurs styles de caractères (polices), vous devez utiliser le fenêtrage du STOS.

Fenêtres

Une fenêtre est une zone rectangulaire totalement indépendante du reste de l'écran dans laquelle des données de tous types peuvent être affichées. Le STOS permet d'afficher jusqu'à 13 fenêtres simultanément.

Pour utiliser une fenêtre, il faut l'ouvrir avec l'instruction windopen dont la syntaxe est la suivante :

```
windopen <n>,<x>,<y>,<larg>,<haut>[,<bord>[,<police>]]
```

où <n> est le numéro de la fenêtre à ouvrir (entre 1 et 13),

<x> et <y> sont les coordonnées du coin supérieur gauche de la fenêtre. En mode 0, <x> doit être compris entre 0 et 37, et <y> entre 0 et 22.

<larg> et <haut> sont la largeur et la hauteur de la fenêtre en nombre de caractères. Ils ont pour valeur minimale 3.

<bord> est le type du contour de la fenêtre (entre 1 et 16),

<police> est le numéro de la police de caractères à utiliser dans la fenêtre.

Exemple

```
mode 0:windopen 1,0,0,40,25    ----> fenêtre plein écran
mode 1:windopen 1,0,0,80,25,3 ----> fenêtre plein écran
                                    délimitée par une bordure
                                    de type 3.
mode 1:windopen 1,0,0,10,10,1,3 --> fenêtre de 8 caractères sur
                                    8 délimitée par une bordure
                                    de type 1, et utilisant
                                    le jeu de caractères 3.
```

Pour effacer une fenêtre, il suffit d'utiliser l'instruction windel en précisant le numéro de la fenêtre à effacer, par exemple, l'instruction :

```
windel 3
```

efface la fenêtre référencée 3.

Si vous le désirez, vous pouvez affecter un titre à une fenêtre avec l'instruction

```
title <t$>
```

où <t$> est le titre à donner à la fenêtre.

Lorsque vous avez défini plusieurs fenêtres sur le même écran, vous pouvez rendre la fenêtre N active en utilisant l'instruction :

```
window N          si les fenêtres se chevauchent,
qwindow N         si les fenêtres ne se chevauchent pas.
```

Si vous le désirez, vous pouvez déplacer la fenêtre courante aux coordonnées de votre choix en utilisant l'instruction windmove :

```
windmove <x>,<y>
```

où <x> et <y> sont les nouvelles coordonnées du coin supérieur gauche de la fenêtre.

Signalons enfin les instructions suivantes :

- ✓ l'instruction clw efface le contenu de la fenêtre courante,
- ✓ l'instruction scroll {on | off} interdit ou autorise le scrolling dans la fenêtre courante,
- ✓ l'instruction scroll {up | down} produit un scrolling de la fenêtre courante d'une ligne vers le haut ou vers le bas.
- ✓ la variable système WINDON contient le numéro de la fenêtre courante.

Jeux de caractères

Si vous désirez utiliser une police de caractères non standard, vous devez effectuer les trois actions suivantes :

❶ réservation de place dans une banque mémoire pour la nouvelle police avec l'instruction reserve as set dont la syntaxe est la suivante :

```
reserve as set <No>,<Taille>
```

où <No> est le numéro d'une banque mémoire,

et <Taille> est la taille du jeu de caractères à utiliser, arrondie au millier supérieur.

❷ Chargement de la police de caractères avec une instruction load du type :

```
load "<police>.mbk"
```

où <police> est le nom d'un fichier police de caractères (par exemple font1, font2 ou font3).

❸ Déclaration de la police à utiliser dans l'instruction windopen <n>,<x>,<y>,<larg>,<haut>,<bord>,<police>

Reportez-vous en début de paragraphe pour avoir de plus amples renseignements sur l'utilisation de l'instruction windopen.

Si vous le désirez, vous pouvez créer votre propre police de caractères. Le paragraphe 2.2 de la première partie du livre vous explique comment.

4.1.5. Menus

Un programme qui fonctionne autour de menus d'écran est toujours plus agréable et plus facile à utiliser. Le BASIC STOS dispose de quelques instructions qui permettent de créer des menus à moindres frais. Nous allons voir comment les utiliser à partir d'un exemple.

Supposons que vous désiriez créer le menu suivant :

```
Fichier
  Lire
  Sauvegarder
Dessin
  Couleur
  Point
  Ligne
  Rectangle
  Cercle
Symétrie
  Horizontale
  Verticale
```

Définissez les options principales du menu avec l'instruction menu$ dont la syntaxe est la suivante :

```
menu$(<N>)=<Ch>[,<papier>,<crayon>]
```

où <N> est le numéro de l'option principale (entre 1 et 10),

<Ch> est la chaîne à affecter au menu,

<papier> est la couleur de l'arrière-plan,

et <crayon> la couleur de l'écriture sur l'arrière-plan.

```
10 menu$(1)="Fichier"
20 menu$(2)="Dessin"
30 menu$(3)="Symétrie"
```

Définissez ensuite les options secondaires du menu en utilisant l'instruction menu$ comme suit :

```
menu$(<N>,<M>)=<Ch>[,<papier>,<crayon>]
```

où <N> est le numéro de l'option principale (entre 1 et 10),

<M> est le numéro de l'option secondaire,

<Ch> est la chaîne à affecter au menu,

<papier> est la couleur de l'arrière-plan,

et <crayon> la couleur de l'écriture sur l'arrière-plan.

```
40 menu$(1,1)="Lire"
50 menu$(1,2)="Sauvegarder"
60 menu$(2,1)="Couleur"
70 menu$(2,2)="Point"
80 menu$(2,3)="Ligne"
90 menu$(2,4)="Rectangle"
100 menu$(2,5)="Cercle"
110 menu$(3,1)="Horizontale"
120 menu$(3,2)=" Verticale"
```

Validez enfin le menu qui vient d'être défini avec l'instruction menu on :

```
130 menu on
```

Lancez le programme. Lorsque la souris passe sur une des options principales, le menu correspondant est affiché. Pour sélectionner une des options secondaires, il vous suffit de cliquer lorsque cette option apparaît en inverse vidéo. Pour exécuter une séquence d'instructions liée à une sélection dans un menu, vous pouvez utiliser l'instruction on menu goto et la variable système mnselect.

L'instruction on menu goto débranche le programme vers une des lignes spécifiées en fonction de l'option principale sélectionnée. Sa syntaxe est celle des instructions on error goto ou on goto :

```
on menu goto <l1>[,<l2>[...[,<lN>]...]]
```

où les <lI> sont des numéros de ligne. on menu goto donne le contrôle au Ième numéro de ligne si vous sélectionnez l'option principale I.

Pour fonctionner correctement, l'instruction on menu goto doit être suivie d'un on menu on.

Entrez les lignes suivantes :

```
140 on menu goto 500,600,700
150 on menu on
160 goto 160
```

Pour connaître l'option secondaire sélectionnée, vous devez utiliser la variable système mnselect. Entrez les instructions suivantes :

```
500 locate 0,0 : print "Option principale 1"
510 print "Option secondaire ";mnselect
520 goto 130
600 locate 0,0 : print "Option principale 2"
610 print "Option secondaire ";mnselect
620 goto 130
700 locate 0,0 : print "Option principale 3"
710 print "Option secondaire ";mnselect
720 goto 130
```

Résumé

Vous venez de voir sur un exemple précis comment créer un menu. La première chose à faire consiste à créer les options du menu avec l'instruction menu$. Validez ensuite le menu avec l'instruction menu on. Définissez les adresses de traitement avec l'instruction on menu goto. Validez le on menu avec une instruction on menu on. L'option secondaire choisie se trouve dans la variable système mnselect.

Sachez également que vous pouvez :

✓ effacer un menu avec menu off,

- ✓ arrêter temporairement le fonctionnement du menu avec menu freeze,
- ✓ suspendre et rétablir une des options secondaires avec menu$(<N>,<M>) {on | off}
- ✓ connaître l'option principale avec la variable système mnbar.

4.2. Imprimante

Les six instructions de base pour communiquer avec l'imprimante en BASIC STOS sont :

- ✓ hardcopy pour réaliser une copie graphique de l'écran courant,
- ✓ windcopy pour réaliser une copie texte de la fenêtre courante,
- ✓ llist pour imprimer le listing du programme courant,
- ✓ listbank pour imprimer la liste des banques utilisées par le programme courant,
- ✓ ldir pour imprimer les fichiers du répertoire spécifié. Reportez-vous à l'instruction dir pour connaître la syntaxe et le fonctionnement de cette instruction,
- ✓ lprint pour imprimer le texte ou le contenu de la/des variable(s) spécifiée(s). Reportez-vous à l'instruction print pour avoir plus de détails sur le fonctionnement de cette instruction

4.3. Lecteurs de disques

Le support de sauvegarde de masse de votre ordinateur est un lecteur de disquettes ou de disque dur. Le STOS contient de nombreuses instructions qui facilitent l'accès au support de masse. Nous diviserons ces instructions en deux groupes :

✓ les instructions liées à la gestion des répertoires,

✓ les instructions liées à la gestion des fichiers.

Gestion des répertoires

Les lecteurs de disquettes et de disque dur ont des possibilités de stockage sans cesse croissantes. Si vous n'êtes pas prudent, vous pouvez finir par ne plus savoir ce qui se trouve sur vos disquettes. Heureusement, il est possible de créer un ou plusieurs répertoires. Dans chacun de ces répertoires seront stockés tous les fichiers qui correspondent à un domaine d'utilisation précis.

Nous allons maintenant voir quelles sont les instructions dédiées à la gestion des disques et comment les utiliser.

L'instruction de base pour accéder à un lecteur de disque est dir. Elle liste le contenu du disque/répertoire spécifié. Par exemple

```
dir "a:"<CR>
```

liste le nom des fichiers du disque A:. Insérez la disquette "Langage" et tapez

```
dir "a:"<CR>
```

Les données suivantes apparaîssent sur l'écran :

```
Lecteur A, dir:
* ->              AUTO
* ->              STOS
BASIC205.PRG      2208
CONFIG  .BAS     19826
CONVERT .BAS     21588
PROTECT .BAS      1150
VERSION .102       106

44878 octets utilisés
```

Examinons les données affichées.

Les fichiers sont listés sous la forme :

```
nom    .ext    taille
```

Les fichiers précédés du sigle "* ->" sont en fait des répertoires. Les fichiers qu'ils contiennent ne sont pas listés.

La liste des fichiers se termine par la taille totale des fichiers listés.

Pour lister les fichiers d'un des répertoires du disque, par exemple le répertoire AUTO, tapez :

```
dir "a:\auto"<CR>
```

Les données affichées sont les suivantes :

```
Lecteur A, dir: \AUTO
BASIC205.PRG     2208
2208 octets utilisés.
```

Ce répertoire est composé d'un seul fichier. Il ne contient aucun sous répertoire. S'il avait contenu un sous-répertoire de nom DATA (par exemple), les fichiers de ce sous-répertoire pourraient être listés avec l'instruction :

```
dir "a:\auto\data"<CR>
```

Si vous travaillez surtout sur un des répertoires d'un disque, il peut être intéressant de le rendre "courant", c'est-à-dire de ne pas avoir à spécifier le nom de ce répertoire lors de chaque opération disque. Pour ce faire, vous devez utiliser la variable système dir$.

Par exemple,

```
dir$="a:\auto"<CR>
```

rend le répertoire AUTO du disque A: courant. Si vous demandez la liste du répertoire courant en entrant

```
dir
```

c'est le répertoire AUTO (et non pas le répertoire principal) qui sera listé.

La variable système dir$ est également accessible en lecture. Dans ce cas, elle renvoie le nom du répertoire courant. Si vous avez changé le nom du répertoire courant comme indiqué dans l'exemple ci-dessus, l'instruction :

```
print dir$
```

affichera :

```
\AUTO
```

Les sous-répertoires d'un répertoire donné sont qualifiés de sous-répertoires "fils". Le répertoire dont ils dépendent est le répertoire "père". Pour rendre celui-ci courant, vous pouvez bien sûr utiliser l'instruction dir$, mais il sera plus pratique d'utiliser l'instruction previous.

Exemple

```
Dir$ = "\AUTO\DATA\BACK"
print dir$
\AUTO\DATA\BACK
previous
print dir$
\AUTO\BACK
previous
print dir$
\AUTO
```

Pour créer un nouveau sous-répertoire, utilisez l'instruction mkdir en spécifiant le nom du sous-répertoire. Par exemple :

```
mkdir "monrep"
```

Le répertoire "monrep" deviendra un sous-répertoire du répertoire courant.

Pour effacer un sous-répertoire, procédez comme suit :

- ✓ effacez tous les fichiers qu'il contient,
- ✓ positionnez-vous dans le répertoire père,
- ✓ utilisez l'instruction rmdir en spécifiant le nom du sous-répertoire à effacer.

Pour effacer un fichier, utilisez l'instruction kill en spécifiant le nom du fichier à effacer, éventuellement précédé du nom du disque et/ou du sous-répertoire qui le contient.

Par exemple, l'instruction

```
kill "b:\auto\monprog.prg"<CR>
```

efface le fichier MONPROG.PRG du répertoire AUTO du disque B:.

Si deux ou plusieurs lecteurs de disquettes ou de disque dur sont reliés à votre ordinateur, vous pourrez rendre le lecteur <N> courant en tapant :

```
drive$-"<N>"
```

Si vous entrez une lettre qui ne correspond à aucun lecteur connecté, un message d'erreur sera affiché :

```
Lecteur non connecté.
```

Enfin, vous pouvez renommer un fichier en utilisant l'instruction rename comme suit :

```
rename <ancien> to <nouveau>
```

où <ancien> est le nom du fichier à renommer,

et <nouveau> est le nouveau nom du fichier.

Résumé

Vous connaissez maintenant les instructions de base relatives à la gestion des répertoires :

- ✓ dir affiche le nom des répertoires et les fichiers d'un lecteur,
- ✓ dir$ permet de modifier ou de connaître le nom du répertoire courant,
- ✓ drive$ permet de modifier ou de connaître le nom du disque courant,
- ✓ previous rend le répertoire père du sous-répertoire actuel courant,
- ✓ mkdir crée un sous-répertoire,

✓ rmdir efface un sous-répertoire vide,

✓ kill efface un fichier,

✓ rename renomme un fichier.

Gestion des fichiers

Sur ATARI ST, les fichiers peuvent être de deux types :

✓ à accès séquentiel,

✓ à accès direct.

Les premiers ne peuvent être accédés qu'en lecture ou en écriture, et pas en lecture/écriture comme les seconds. Ils contiennent généralement des informations lues en une seule fois, comme par exemple des données de redéfinition pour une imprimante, des données de configuration, etc.. Les fichiers à accès direct, par contre, sont tout à fait indiqués pour stocker des banques de données (fichier d'adresses, de téléphone, etc..)

Les instructions STOS concernant ces deux types de fichiers sont assez différentes. Aussi, nous les étudierons indépendamment.

Fichiers à accès séquentiel

Les fichiers sont comparables à des livres: Pour les lire ou les écrire, il faut tout d'abord les ouvrir. L'ouverture d'un fichier séquentiel se fait avec l'instruction open dont la syntaxe dépend du type d'accès souhaité :

```
Lecture : open in #<No>,"<Nom>"
Ecriture: open out #<No>,"<Nom>"
```

où <No> est un numéro d'accès entre 1 et 10,

et <Nom> est le nom du fichier à accéder.

Pour écrire des données dans un fichier séquentiel, vous devez utiliser l'instruction print# suivie du numéro affecté à l'ouverture et de la donnée à écrire :

```
print #<No>,<data>
```

où <No> est le numéro d'accès affecté au fichier dans l'instruction open out,

et <data> est une donnée ou le nom d'une variable de tout type.

Pour lire des données dans un fichier séquentiel, vous devez utiliser l'instruction input# suivie du numéro affecté à l'ouverture et d'un nom de variable dans laquelle sera stockée la donnée lue :

```
input #<No>,<var>
```

où <No> est le numéro d'accès affecté au fichier dans l'instruction open in,

et <var> est la variable dans laquelle doit être stockée la donnée lue.

Enfin, lorsque le fichier séquentiel a été totalement lu ou écrit, il doit être fermé avec une instruction close :

```
close #<No>
```

Exemple

Création d'un fichier séquentiel:

```
10 open out #1,"data.abc"
20 for I=1 to 5
30 read A$
40 print #1,A$
50 next I
60 close #1
60 data "Premier","Second","Troisième","Quatrième","Cinquième"
```

Lecture d'un fichier séquentiel :

```
10 open in #1,"data.abc"
20 for I=1 to 5
30 input #1,A$
```

```
40 print A$
50 next I
60 close #1
```

Exécutez ces deux programmes. Le premier stocke les chaînes "Premier", "Second", "Troisième", "Quatrième" et "Cinquième" dans le fichier DATA.ABC. Le second lit ces données et les affiche sur l'écran.

Fichiers à accès direct

Les fichiers à accès direct sont légèrement plus complexes à utiliser que les fichiers à accès séquentiel.

Un fichier à accès direct est composé d'enregistrements de longueur fixe, eux même composés de champs de longueur fixe. Raisonnons sur un exemple. Un pharmacien doit gérer son stock de produits. Pour cela, il utilise un fichier à accès direct dans lequel les enregistrements ont la structure suivante :

```
Nom       --- texte sur 15 caractères
Quantité  --- entier sur 4 chiffres.
```

Pour accéder à son fichier, il doit l'ouvrir avec une instruction open du type :

```
open #1,"R","MEDIC.DAT"
```

Il doit ensuite définir la structure d'un enregistrement avec l'instruction field :

```
field #1,15 as nom$,4 as qte$
```

Vous l'avez peut-être remarqué dans cet exemple, le champ quantité (qte$) est de type chaîne alors qu'il contient une donnée numérique. En fait, tous les champs d'un fichier à accès direct sont de type chaîne.

Pour écrire un enregistrement dans le fichier, notre pharmacien doit stocker des valeurs dans les variables nom$ et qte$, puis utiliser l'instruction put#. Par exemple :

```
nom$="Mérisone"
qte$="120"
put #1,5
```

La syntaxe de l'instruction put est la suivante :

```
put #<No>,<Enreg>
```

où <No> est le numéro d'accès affecté au fichier dans l'instruction open,

et <Enreg> est le numéro de l'enregistrement à écrire.

Pour extraire un enregistrement du fichier, il suffit d'utiliser l'instruction get# selon le même format que l'instruction put# :

```
get #1,5
```

Les données lues se trouvent dans les variables nom$ et qte$.

Lorsque le fichier ne doit plus être utilisé, il doit être fermé avec une instruction close en précisant son numéro d'accès. Par exemple :

```
close #1
```

Exemple

Le petit programme ci-dessous permet de lire et d'écrire dans un fichier à accès direct. Veillez bien à écrire des données dans le fichier avant d'essayer de les lire, car cela produirait une erreur.

Le lecteur par défaut doit contenir une disquette formatée non protégée en écriture.

```
10 open #1,"R","medic.dat"
20 field #1,15 as NOM$,4 as QTE$
30 input "L)ecture, E)criture ou F)in : ";R$
40 if upper$(R$)="L" then 100
50 if upper$(R$)="E" then 200
60 if upper$(R$)="F" then 300
70 goto 30
100 input "No d'enregistrement : ";NO
110 get #1,NO
120 print "Nom = ";NOM$
130 print "Quantité = ";QTE$
140 goto 30
200 input "No d'enregistrement : ";NO
210 input "Nom : ";NOM$
220 input "Quantité : ";QTE$
```

```
230 put #1,NO
240 goto 30
300 close #1
```

Conclusion

La gestion des fichiers séquentiels et à accès direct ne doit plus avoir de secrets pour vous. Vous savez :

- ✓ ouvrir un fichier avec l'instruction open,
- ✓ fermer un fichier avec l'instruction close,
- ✓ définir la structure d'un fichier à accès direct avec l'instruction field,
- ✓ lire une donnée avec l'instruction input (fichier séquentiel) ou get (fichier à accès direct),
- ✓ écrire une donnée avec l'instruction print (fichier séquentiel) ou put (fichier à accès direct).

Chapitre 5

Organes d'entrée

5.1. Le clavier

Le clavier est de loin l'organe d'entrée le plus utilisé dans un ordinateur. Sur les ATARI ST, chaque appui sur une touche du clavier se traduit par l'émission d'un ou de deux codes dans une mémoire spéciale appelée "buffer clavier". Les touches classiques (lettres, chiffres, signes) sont codées sur un seul code appelé code ASCII. Les touches spéciales (touches de fonction, touches flèches, help, Undo, etc.) ont un code ASCII nul. Pour pouvoir les différentier, un second code leur est affecté : le SCANCODE.

La fonction inkey$ renvoie le caractère tapé. Pour mieux comprendre le fonctionnement de cette fonction, considérez le programme suivant :

```
10 A$=inkey$ : if A$="" then 10
20 print asc(A$)
30 if asc(A$)<>27 then 10
```

La ligne 10 attend qu'une touche du clavier soit frappée. La ligne 20 affiche le code ASCII de la touche frappée. Si ce code est différent de 27 (ESCape), la ligne 30 redonne le contrôle à la ligne 10.

Lancez ce programme. Il marche parfaitement bien pour les touches classiques. Mais que se passe-t-il si vous appuyez sur une touche de fonction ou sur la touche <Help> ?. Le code ASCII affiché est nul. Modifiez le programme comme suit :

```
20 if asc(A$)<>0 then print "touche classique de code ASCII
";asc(A$)
25 if asc(A$)=0 then print "touche spéciale de scancode
";scancode
```

Exécutez le programme. Toutes les touches du clavier sont maintenant décodées.

Si vous désirez attendre l'appui sur une touche du clavier, sans pour cela avoir le code de la touche pressée, vous pouvez utiliser l'instruction wait key.

Exemple

```
10 print "Appuyez sur une touche ..."
20 wait key
30 print "merci."
```

La vitesse de répétition automatique des touches peut être modifiée avec l'instruction key speed dont la syntaxe est la suivante :

```
key speed <v>,<d>
```

où <v> est la vitesse de répétition. Plus <v> est petit, plus la répétition est rapide,

et <d> est le délai (en 1/50 secondes) entre l'appui sur une touche et le début de la répétition automatique.

Enfin, l'instruction key permet d'affecter une chaîne de caractères (64 caractères au maximum) à l'une des touches de fonction. Son format est le suivant :

```
key(<No>)=<ch>
```

où <No> est le numéro de la touche de fonction (entre 1 et 20). Les numéros compris entre 1 et 10 correspondent aux touches <F1> à <F10>. Les numéros compris entre 11 et 20 correspondent aux touches <F1> à <F10> shiftées (touche <Shift> enfoncée).

et <ch> est une chaîne ou une variable chaîne à affecter à la touche de fonction <No>.

5.2. La souris

Sur les ATARI ST, la souris est tellement utilisée que l'on pourrait presque dire qu'elle fait office de second clavier. Sous STOS, sa position sur l'écran se trouve en permanence dans les variables système x mouse et y mouse, et l'état de ses touches dans la variable système mouse key. Cette variable peut avoir les valeurs suivantes :

mouse key	signification
0	Aucune touche enfoncée
1	Touche gauche enfoncée
2	Touche droite enfoncée
3	Touches droite et gauche enfoncées

Les déplacements de la souris peuvent être limités à une portion rectangulaire de l'écran avec l'instruction limit mouse dont la syntaxe est la suivante :

```
limit mouse <x1>,<y1> to <x2>,<y2>
```

où <x1> et <y1> sont les coordonnées du coin supérieur gauche du rectangle,

et <x2> et <y2> les coordonnées du coin inférieur droit du rectangle.

Nous allons terminer ce paragraphe par l'instruction change mouse qui permet de modifier la forme du curseur de la souris.

change mouse 1 est la forme habituelle du curseur,

change mouse 2 transforme le curseur en une main dont l'index est pointé,

change mouse 3 transforme le curseur en une horloge.

5.3. Le joystick

Le STOS étant avant tout un langage de jeu, vous serez très certainement amené à utiliser les instructions relatives au joystick. Etant donné que ce périphérique est des plus simples, les instructions qui lui sont rattachées ne sont pas nombreuses.

Les fonctions jleft, jright, jup, jdown et jfire retournent la valeur -1 lorsque (respectivement) le joystick est manoeuvré vers la gauche, la droite, le haut, le bas, et lorsque le bouton feu a été pressé. Ces fonctions retournent la valeur 0 dans le cas contraire.

Si la manipulation des bits ne vous rebute pas, vous pouvez également utiliser la fonction joy qui équivaut à toutes les fonctions précédentes. Les bits 0 à 4 du nombre renvoyé par joy renseignent sur les mouvements du joystick comme le montre le tableau suivant :

```
bit  mouvement
0    joystick vers le haut
1    joystick vers le bas
2    joystick à gauche
3    joystick à droite
4    bouton feu actionné
```

Chapitre 6

Les sprites

Un des grands avantages du STOS est qu'il permet de créer et d'animer très facilement des sprites. Dans ce chapitre, nous allons surtout parler de l'animation. Si vous ne savez pas encore utiliser l'éditeur de sprite, reportez-vous au paragraphe 2.1. de la première partie du livre où vous apprendrez comment le mettre en oeuvre pour créer des sprites de tous types.

Dans le langage STOS, les sprites (parfois aussi appelés esprits ou lutins) sont des objets graphiques quelconques qui s'inscrivent dans une grille dont les dimensions X Y sont comprises entre 16x2 et 64x64 pixels. Ces objets peuvent être utilisés pour créer le décor ou les éléments (objets, personnages, etc.) d'un jeu. Leur animation est totalement contrôlée sous interruptions. En d'autres termes, un sprite peut se déplacer sur l'écran alors qu'un programme STOS est en train de s'occuper d'une autre tâche, comme par exemple tester la souris ou le joystick.

6.1. Affichage d'un sprite

Lorsque vous avez créé un fichier .MBK contenant un ou plusieurs sprites avec l'éditeur de sprites, la première chose à faire avant de visualiser le fruit de vos efforts est de charger ce fichier en mémoire. Dans la suite, nous allons travailler avec le fichier SPRITE.MBK créé dans la première partie du livre.

Pour charger ce fichier en mémoire, insérez la disquette sur laquelle il se trouve dans le lecteur A: et tapez en mode direct :

```
load "sprite.mbk"<CR>
```

Quelques instants après, la banque de sprites est en mémoire. Pour vous en convaincre, listez les banques en mémoire en tapant, toujours en mode direct :

```
listbank<CR>
```

Une information du type suivant doit s'afficher sur l'écran:

```
Banques mémoire réservées:
1  sprites S:$1D1C00 E:$1D8100 L:$006500
```

Les sprites du fichier SPRITE.MBK sont bien en mémoire. Mais comment les afficher sur l'écran. Et bien tout simplement en utilisant l'instruction sprite dont la syntaxe est la suivante :

```
sprite <No>,<Abs>,<Ord>,<Sp>
```

où <No> est un numéro de référence compris entre 1 et 15 qui sera utilisé par toutes les autres instructions pour accéder à ce sprite,

<Abs> est l'abscisse du point chaud (par défaut la partie gauche du sprite) sur l'écran. Elle doit être comprise entre -640 et 1280. En mode 0 (basse résolution), l'abscisse 0 correspond au côté gauche de l'écran, et l'abscisse 320 au côté gauche de l'écran.

<Ord> est l'ordonnée du point chaud (par défaut la partie supérieure du sprite) sur l'écran. Elle doit être comprise entre -400 et 800. En mode 0, l'ordonnée 0 correspond au bord supérieur de l'écran, et l'abscisse 200 au bord inférieur de l'écran.

<Sp> est le numéro du sprite à afficher dans la banque.

Essayez par exemple de taper en mode direct:

```
mode 0<CR>
sprite 1,100,100,1
```

La souris définie dans la première partie du livre apparaît sur l'écran.

Quelques remarques avant de poursuivre :

- ✓ Déplacez le curseur du clavier vers le sprite, et tapotez sur votre clavier. Le texte entré apparaît derrière le sprite.
- ✓ Déplacez le curseur de la souris ATARI. Il passe devant le sprite.
- ✓ Les couleurs de la souris ne sont pas celles utilisées dans l'éditeur de sprites. N'ayez crainte, il est facile de les restituer. Tapez en mode direct :

```
palette 0,$777,$762,$057,$432,$444,$555,$666,$555,$222,
$733,$072,$421,$311,$753,$531<CR>
```

Les couleurs sont maintenant celles qui avaient été utilisées dans l'éditeur de sprites.

6.2. Déplacement d'un sprite

La première étape est franchie. Mais comment déplacer un sprite sur l'écran ?

Tout simplement en utilisant les instructions "move x", "move y" et "move on" qui, comme leur nom l'indique permettent (respectivement) de déplacer un sprite horizontalement, verticalement et de valider son déplacement. La syntaxe de move x et de move y est similaire :

```
move x <No>,"<Ch>"
move y <No>,"<Ch>"
```

où <No> est le numéro de référence (compris entre 1 et 15) qui a été défini dans une instruction sprite précédemment exécutée

et <Ch> est une chaîne d'animation qui peut être composée d'un ou de plusieurs des éléments suivants :

① Déplacement de base

```
(<Pér>,<Pas>,<Mouv>)
```

où <Pér> est la période de déplacement du sprite. Cette période est comprise entre 1 et 32767. La période 1 est donc très rapide et la période 32767 très lente.

<Pas> est l'amplitude du déplacement en pixels, entre -10 et 10. Les valeurs négatives produisent des déplacements vers la gauche pour move x et vers le haut pour move y. Les valeurs positives des déplacements vers la droite pour move x et des déplacements vers le bas pour move y.

<Mouv> représente le nombre de déplacements à effectuer, entre 0 et 32767. La valeur 0 correspond à un déplacement perpétuel.

Par exemple, les instructions :

```
move x 1,"(10,1,50)"
move on
```

provoqueront cinquante déplacements rapides d'amplitude 1 pixel vers la droite, et les instructions :

```
move x 1,"(10,1,50)(20,-1,50)"
move on
```

provoqueront cinquante déplacements rapides d'amplitude 1 pixel vers la droite suivis de cinquante déplacements plus lents vers la gauche d'amplitude 1 pixel.

② Bouclage

La lettre "L" active indéfiniment la commande de déplacement qui la précède. Par exemple, les instructions

```
move x 1,"(6,4,10)(6,-4,10)L"
move on
```

font déplacer rapidement et indéfiniment la souris de droite à gauche et de gauche à droite.

③ Arrêt en un point particulier

La lettre "E" suivie d'une position en abscisse arrête la commande de déplacement qui la précède si la position spécifiée est exactement atteinte. Par exemple, les instructions :

```
move x 1,"(6,4,100)E120"
move on
```

arrêtent prématurément les 100 déplacements demandés lorsque l'abscisse du sprite est égale à 120.

④ Arrêt en un point particulier et reprise

La lettre "L" suivie d'une position en abscisse arrête la commande de déplacement qui la précède si la position spécifiée est exactement atteinte et la réexécute. Cette commande n'a de réel intérêt que si une abscisse est précisée en début d'instruction. Par exemple, les instructions :

```
move x 1,"80(6,4,100)L120"
move on
```

déplacent le sprite vers la droite à partir de l'abscisse 80, arrêtent prématurément son déplacement à l'abscisse 120, et réexécutent le même déplacement indéfiniment.

Résumé

Pour déplacer un sprite, il faut :

- ✓ charger la banque qui le contient en mémoire avec l'instruction load,
- ✓ redéfinir la palette des couleurs (si nécessaire) avec l'instruction palette,
- ✓ lui affecter un numéro et une position de départ avec l'instruction sprite,
- ✓ le déplacer avec une ou plusieurs instruction(s) move x et/ou move y,
- ✓ valider son déplacement avec l'instruction move on.

Exemple

Le petit programme suivant montre comment faire déplacer la souris de plus en plus rapidement pour lui faire traverser l'écran de la droite vers la gauche :

```
10 load"sprite.mbk"
20 mode 0
30 palette 0,$777,$762,$057,$432,$444,$555,$666,$555,$222,$733,
$072,$421,$311,$753,$531
40 sprite 1,300,100,1
50 move x 1,(5,-1,10)(5,-2,10)(5,-3,10)(5,-4,10)(5,-5,10)
(5,-6,10)(5,-7,10)"
60 move on
```

Pour en terminer avec les instructions relatives au déplacement des sprites, signalons également :

move freeze[<No>] qui suspend le mouvement de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

move on [<No>] qui autorise le mouvement de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

move off [<No>] qui inhibe le mouvement de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

<Var>=movon(<No>) qui renvoie un résultat différent de zéro si le sprite <No> est en train de se déplacer, et un résultat égal à zéro dans le cas contraire.

<Var>=x sprite(<No>) qui renvoie l'abscisse du point chaud du sprite <No>.

<Var>=y sprite(<No>) qui renvoie l'ordonnée du point chaud du sprite <No>.

limit sprite <x1>,<y1> to <x2>,<y2> qui définit une zone rectangulaire dans laquelle les sprites seront visibles. <x1>,<y1> représentent les coordonnées du coin supérieur gauche du rectangle, et <x2>,<y2> les coordonnées du coin inférieur droit du rectangle.

6.3. Animation d'un sprite

Maintenant, vous savez déplacer un sprite sur l'écran. Mais avouez que les déplacements seraient bien plus spectaculaires si les sprites étaient animés. Dans ce paragraphe, nous allons étudier les mécanismes de l'animation.

L'instruction de base pour animer un sprite est anim dont la syntaxe est la suivante :

```
anim <No>,<Ch>
```

où <No> est le numéro de référence (compris entre 1 et 15) qui a été défini dans une instruction sprite précédente,

et <Ch> est une chaîne d'animation qui peut être composée d'un ou de plusieurs des éléments suivants :

① Animation de base

(<Im>,<Pér>)

où <Im> est le numéro du sprite à afficher,

et <Pér> est la période d'animation du sprite. Cette période est comprise entre 1 et 32767. La période 1 est donc très rapide et la période 32767 très lente.

Par exemple, les instructions :

```
anim 1,"(1,10)(2,10)(3,10)"
anim on
```

provoqueront l'enchaînement moyennement rapide de trois sprites. L'instruction anim on est obligatoire. Sans elle, les animations spécifiées ne sont pas exécutées.

② Bouclage

Comme vous l'avez remarqué dans l'exemple précédent, la séquence d'animation n'est exécutée qu'une fois. Pour l'exécuter en permanence, il suffit de placer un "L" à la fin de la chaîne d'animation. L'exemple précédent devient :

```
anim 1,"(1,10)(2,10)(3,10)L"
anim on
```

L'animation est permanente.

Essayez le petit programme suivant qui déplace et anime en permanence notre souris :

```
10 load"sprite.mbk"
20 mode 0
30 palette 0,$777,$762,$057,$432,$444,$555,$666,$555,$222,$733,
$072,$421,$311,$753,$531
40 sprite 1,300,100,1
50 move x 1,"300(4,-2,100)L" : move on
60 anim 1,"(1,10)(2,10)(3,10)L" : anim on
```

Tout comme pour l'instruction move, anim peut être activée, désactivée ou suspendue à l'aide des attributs on, off et freeze. Les instructions sont alors :

```
anim on[<No>]
```

qui autorise l'animation de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

```
anim off[<No>]
```

qui inhibe l'animation de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

```
anim freeze[<No>]
```

qui suspend momentanément l'animation de tous les sprites si <No> n'est pas spécifié, ou du sprite <No> si <No> est spécifié.

6.4. Tests de collision

Les test de collisions sont de trois types :

❶ Entrée d'un sprite dans une zone rectangulaire,

❷ Test de la couleur du décor sous le point chaud,

❸ Collision entre deux sprites.

① Entrée d'un sprite dans une zone rectangulaire

Pour tester l'entrée d'un sprite dans une zone rectangulaire particulière, il faut procéder en deux temps :

Dans un premier temps, la zone à tester doit être créée avec l'instruction set zone dont la syntaxe est la suivante :

```
set zone <NZ>,<X1>,<Y1> to <X2>,<Y2>
```

où <NZ> est un numéro de zone compris entre 1 et 128,

<X1> et <Y1> sont les abscisse et ordonnée du coin supérieur gauche de la zone rectangulaire,

<X2> et <Y2> sont les abscisse et ordonnée du coin inférieur droit de la zone rectangulaire.

Dans un second temps, la fonction zone permet de tester la présence d'un sprite dans la zone définie. La syntaxe de cette fonction est la suivante :

```
<Var>=zone(<No>)
```

où <No> est un numéro de sprite, compris entre 1 et 15

et <Var> est le résultat de la fonction zone : 0 si le sprite <No> ne se trouve dans aucune zone définie, le numéro de la zone pénétrée dans le cas contraire.

Si l'utilisation de ces deux instructions ne vous semble pas évidente, considérez l'exemple suivant :

```
10 mode 0
20 reset zone
30 set zone 1,1,1 to 30,30
40 sprite 1,10,10,1 : wait 5
50 print zone(1)
60 sprite 1,100,10,1 : wait 5
70 print zone(1)
```

La ligne 20 définit une zone rectangulaire en haut et à gauche de l'écran. La ligne 30 positionne le sprite dans cette zone et attend qu'il apparaisse. La fonction zone(1) renvoie donc la valeur 1. La ligne 50 positionne le sprite hors de la zone et attend qu'il soit affiché. La fonction zone(1) renvoie donc la valeur 0.

Avant de passer au test de collision entre deux sprites, signalons l'instruction reset zone qui permet d'effacer la définition d'une zone particulière ou de toutes les zones définies avec set zone. Sa syntaxe est la suivante :

```
reset zone [(<NZ>)]
```

où <NZ> est un numéro de zone. Si <NZ> n'est pas précisé, toutes les zones précédemment créées par set zone sont effacées. Si <NZ> est précisé, seule la zone <NZ> est effacée.

② Test de la couleur du décor sous le point chaud

La fonction detect renvoie la couleur du décor situé sous le point chaud d'un sprite. Sa syntaxe est :

```
<Var> = detect(<No>)
```

où <No> est un numéro de sprite compris entre 1 et 15,

et <Var> est la couleur du décor situé sous le point chaud.

③ Collision entre deux sprites

Le moyen le plus simple pour tester si deux ou plusieurs sprites sont entrés en collision consiste à utiliser la fonction collide qui renvoie sous forme binaire le numéro du ou des sprites qui est/sont en contact avec un sprite donné. Le format de cette fonction est le suivant :

```
<Rés>=collide(<No>,<Lar>,<Hau>)
```

où <No> est le numéro du sprite dont vous voulez tester la collision, entre 1 et 15,

<Lar> est le nombre de pixels horizontaux autour du point chaud à partir desquels toute pénétration est considérée comme une collision,

<Hau> est le nombre de pixels verticaux autour du point chaud à partir desquels toute pénétration est considérée comme une collision.

et <Rés> est un nombre binaire qui contient le numéro du ou des sprites qui a(ont) pénétré(s) dans la zone de collision. Lorsque le bit N de <Rés> vaut 1, cela signifie que le sprite N est entré dans la zone de collision du sprite <No>.

Vous trouverez un exemple d'utilisation de cette fonction dans le programme d'exemple donné dans la conclusion. Dans cet exemple, collide(1,10,10) vaut 4, ce qui signifie que le bit 2 est à un, et donc que le sprite 2 est entré en collision avec le sprite 1.

Conclusion

Nous allons conclure ce chapitre par un petit programme qui utilise le joystick pour diriger la souris sur l'écran et teste sa rencontre avec le chat. Ce programme utilise les instructions liées à la manipulation du joystick. Reportez-vous au chapitre 5.3 de la seconde partie du livre si nécessaire. Le listing du programme est le suivant :

```
1000 mode 0
1010 palette 0,$777,$762,$57,$432,$444,$555,$666,$555,$222,$733,
$72,$421,$311,$753,$531
1020 sprite 1,300,100,1 : sprite 2,300,150,15
1030 move x 2,"300(4,-2,100)L" : move on
1040 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
1050 G=0 : D=0 : H=0 : B=0
1060 repeat
1070 if jright and D=0 then move y 1,"(4,0,1)" : move x
1,"(4,2,10)L" : move on : anim 1,"(4,10)(5,10)(6,10)L" : anim on
: D=1
1080 if jleft and G=0 then move y 1,"(4,0,1)" : move x
1,"(4,-2,10)L" : move on : anim 1,"(1,10)(2,10)(3,10)L" : anim
on : G=1
1090 if jup and H=0 then move x 1,"(4,0,1)" : move y
1,"(4,-2,10)L" : move on : anim 1,"(7,10)(8,10)(9,10)(10,10)L" :
anim on : H=1
1100 if jdown and B=0 then move x 1,"(4,0,1)" : move y
1,"(4,2,10)L" : move on : anim 1,"(11,10)(12,10)(13,10)(14,10)L"
: anim on : B=1
1110 if joy=0 then G=0 : D=0 : H=0 : B=0 : move freeze (1) :
anim freeze (1)
1120 until collide(1,10,10)=4
1130 boom : mode 1
```

Avant d'exécuter le programme, chargez le fichier qui contient les sprites en mémoire. Tapez en mode direct :

```
load "sprite.mbk"<CR>
```

Voici la signification détaillée de chaque ligne du programme :

ligne 1000 : initialisation de l'écran en mode basse résolution,

ligne 1010 : initialisation de la palette des couleurs aux valeurs utilisées dans l'éditeur de sprites,

ligne 1020 : définition des sprites (1 = souris, 2 = chat),

ligne 1030 : définition du déplacement du chat,

ligne 1040 : définition de l'animation du chat,

ligne 1050 : initialisation des variables liées à la direction du joystick,

ligne 1070 : si le joystick est actionné vers la droite alors qu'il était au repos (D=0), tout éventuel déplacement vertical est éliminé, un déplacement horizontal vers la droite est initialisé et une animation à l'aide des sprites 4, 5 et 6 est créée. La variable D est mise à 1 pour éviter une nouvelle initialisation du mouvement et de l'animation lors du prochain passage dans la boucle,

lignes 1080, 1090 et 1100 :

similaires à la ligne 1070, mais pour un déplacement vers la gauche, le haut et le bas.

ligne 1110 : si le joystick n'est pas actionné, les variables G, D, H et B sont mises à 0, ce qui permettra de valider un autre déplacement, et tous les mouvements et animations courants sont suspendus,

ligne 1120 : si la souris rencontre le chat, la boucle d'animation prend fin,

ligne 1130 : un boom est émis et l'écran passe en moyenne résolution.

Chapitre 7

Musique et effets sonores

Sur la carte mère de votre ATARI se trouve un circuit intégré spécialisé appelé PSG (Programmable Sound Generator). Comme son nom l'indique, ce circuit est utilisé par l'ordinateur pour générer des sons.

Les sons générés sont riches et variés. Ils peuvent être émis sur trois voix indépendantes. Sur chacune de ces voix peut se superposer un souffle, aussi appelé bruit blanc.

Les sons générés peuvent être modulés par des enveloppes , ce qui élargit considérablement leur variété.

Pour produire des effets sonores dans vos programmes, vous avez deux possibilités :

❶ intégrer des commandes sonores directement dans le programme,

❷ utiliser l'éditeur de musique puis l'instruction music pour activer les morceaux créés.

Nous allons voir comment procéder dans les deux cas.

7.1. Intégration de commandes sonores dans un programme

Le STOS dispose de trois effets sonores prédéfinis :

- ✓ bell : son de cloche,
- ✓ boom : explosion,
- ✓ shoot : coup de feu.

Pour activer un de ces effets sonores, il suffit d'écrire le mot clé correspondant sur une ligne BASIC, par exemple :

```
10 boom
```

Remarque

Si vous craignez des interférences avec le cliquetis du clavier, utilisez de part et d'autre de l'activation du son les instructions click off et click on. Par exemple :

```
10 click off
20 boom
30 click on
```

Si vous trouvez que ces trois bruits prédéfinis sont insuffisants, vous pouvez utiliser les instructions volume, noise, envel et play pour en créer de nouveaux.

L'instruction volume fixe le volume des voix sonores. Elle demande un ou deux paramètre(s). Le premier indique la voix sur laquelle s'applique l'instruction volume. S'il est absent, l'instruction s'applique aux trois voix. Le second paramètre est compris entre 1 et 15. Il indique le volume souhaité.

Par exemple :

volume 13 pour fixer le volume des 3 voix à 13,
volume 2,10 pour fixer le volume de la voix 2 à 10.

L'instruction noise produit un souffle pur. Elle s'utilise comme suit :

```
noise [<voix>,]<hauteur>
```

où <voix> est une des voix sonores (1, 2 ou 3),

et <hauteur> est la hauteur du son généré (entre 1 et 31).

Lorsque la voix n'est pas précisée, le souffle est émis sur les trois voix.

Essayez ce court programme qui simule le bruit d'une locomotive à vapeur :

```
10 noise 1 : wait 10 : noise 20 : wait 10 : goto 10
```

Maintenant, vous savez comment générer des souffles. Voyons comment créer des sons purs à l'aide de l'instruction play. La syntaxe de cette instruction est :

```
play [<voix>,]<hauteur>,<durée>
```

où <voix> est le numéro de la voix à activer (entre 1 et 3). Si ce paramètre est omis, le son est joué simultanément sur les trois voix,

<hauteur> est une valeur entière comprise entre 1 et 96 qui spécifie la note à jouer selon les conventions suivantes :

	Note	Do	Do#	Ré	Ré#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
Octave													
0		1	2	3	4	5	6	7	8	9	10	11	12
1		13	14	15	16	17	18	19	20	21	22	23	24
2		25	26	27	28	29	30	31	32	33	34	35	36
3		37	38	39	40	41	42	43	44	45	46	47	48
4		49	50	51	52	53	54	55	56	57	58	59	60
5		61	62	63	64	65	66	67	68	69	70	71	72
6		73	74	75	76	77	78	79	80	81	82	83	84
7		85	86	87	88	89	90	91	92	93	94	95	96

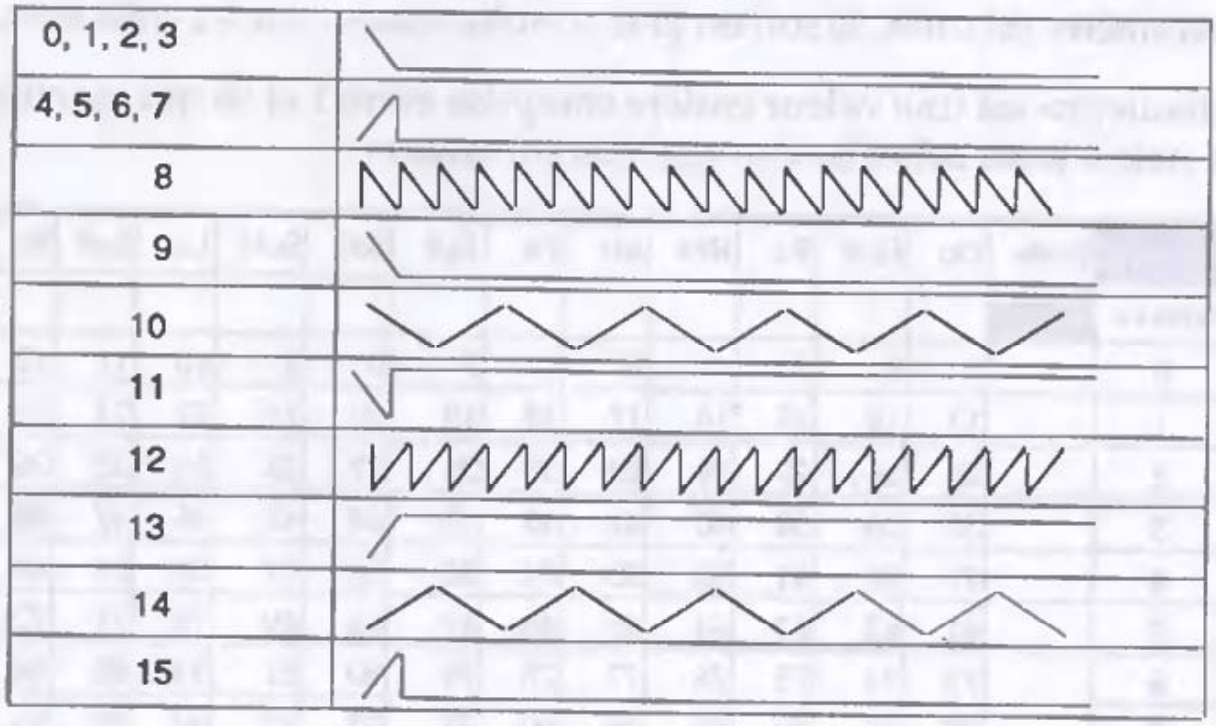
et <durée> est la durée de la note en 1/50 seconde.

Le petit programme ci-dessous montre comment jouer la gamme avec l'instruction play :

```
10 for I=1 to 7
20 read A
30 play A,20
40 next I
50 data 49,51,53,54,56,58,60
```

Les souffles purs générés par noise et les notes pures générées par play peuvent être modulés selon une des 16 enveloppes de l'ATARI en utilisant l'instruction envel.

Une enveloppe est une courbe qui définit les temps de montée, de stabilisation et de descente d'un son. En faisant varier ces trois paramètres, les sons émis sont très différents. Les enveloppes prédéfinies de l'ATARI sont les suivantes :



L'instruction envel a la syntaxe suivante :

```
envel <env>,<vitesse>
```

où <env> est le numéro de l'enveloppe à activer : entre 0 et 15,

et <vitesse> est la longueur de l'enveloppe : entre 1 (très courte) et 65535 (très longue).

Pour vous rendre compte de l'effet des diverses enveloppes, exécutez le programme suivant, tout en examinant le schéma ci-dessus :

```
10 click off : rem Suppression du cliquetis des touches
20 volume 16
30 for I=0 to 15
40 print "Enveloppe";I : rem Enveloppe actuelle
50 envel I,15000
60 play 60,100
70 wait key : rem Attente de l'appui sur une touche
80 next I
90 click on : rem Restitution du cliquetis des touches
```

L'instruction envel agit également sur noise. Pour vous en convaincre, remplacez la ligne 60 du programme précédent par :

```
60 noise 20
```

et exécutez le programme en examinant le schéma ci-dessus.

Le générateur sonore peut également être accédé à travers ses registres par l'instruction psg. Mais ce type de programmation dépasse le cadre du livre.

7.2. Utilisation de l'éditeur de musique

Reportez-vous au paragraphe 2.4. de la première partie du livre pour prendre connaissance du fonctionnement de l'éditeur de musique et pour avoir un exemple pratique de son utilisation.

Avant d'être en mesure d'exécuter un morceau créé par l'éditeur de musique, il vous faut le charger en mémoire.

Si vous connaissez le nom du fichier d'extension .mbk auquel il se rattache, chargez-le en mémoire en tapant :

```
load <nom>
```

où <nom> est le nom du fichier d'extension .mbk créé par le générateur sonore.

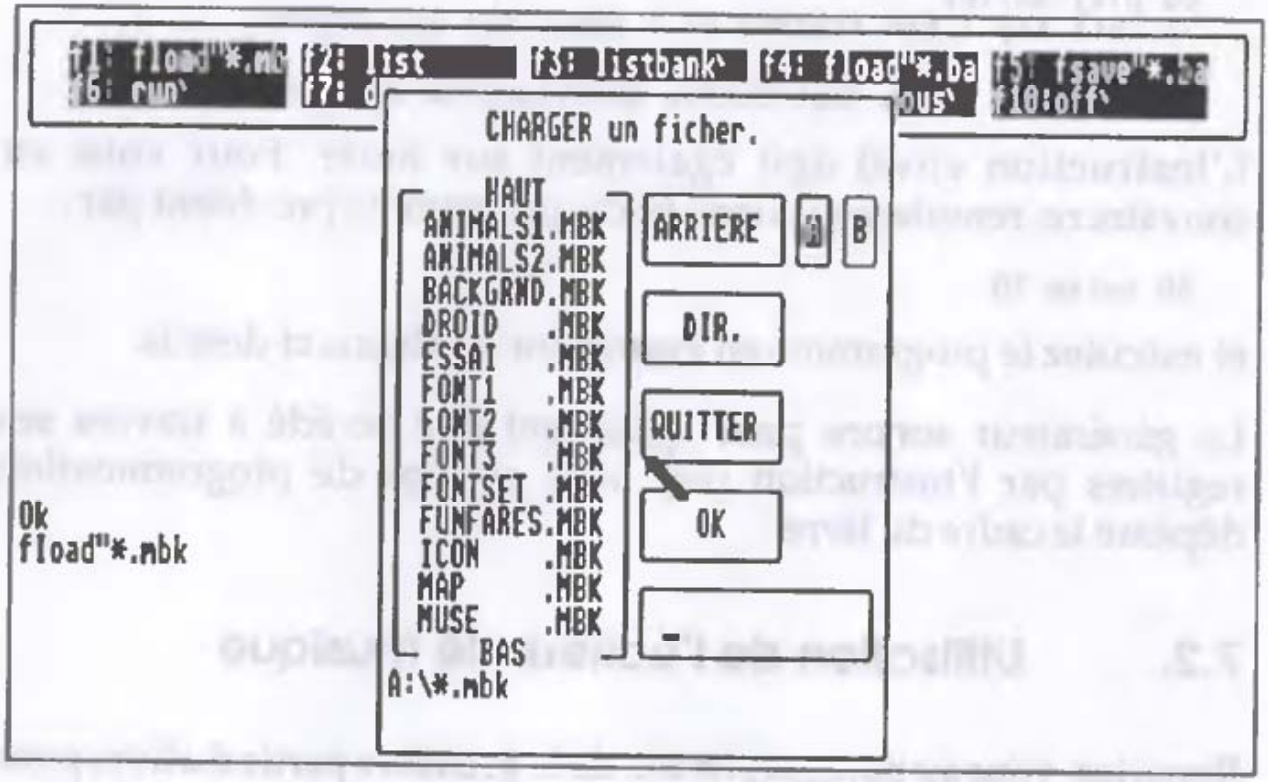
Pour charger le fichier "music.mbk" de la disquette Accessoires, insérez la disquette dans le lecteur et tapez :

```
load "music.mbk"<CR>
```

Si vous n'avez plus en tête le nom du fichier .mbk, tapez :

```
fload "*.mbk"<CR>
```

Une fenêtre du type suivant apparaît sur l'écran.



Choisissez le fichier musical qui vous intéresse en cliquant dessus deux fois de suite avec le bouton gauche de la souris. Au besoin, faites-le apparaître en cliquant autant de fois que nécessaire sur le label "BAS" situé en bas de la fenêtre des noms de fichiers.

Lorsque le fichier musical est en mémoire, activez un de ses morceaux en tapant :

```
music <N>
```

où <N> est le morceau à exécuter.

Par exemple, l'instruction

```
music 1<CR>
```

active le morceau numéro 1.

Comme vous avez pu le remarquer, l'exécution d'une instruction music n'affecte en rien l'accessibilité au BASIC. En effet, les morceaux exécutés par music utilisent les interruptions (comme les sprites).

Comme nous venons de le voir, l'instruction music active un des morceaux en mémoire. Mais son rôle ne se limite pas là. music permet aussi :

- ✓ de suspendre l'exécution du morceau : music freeze,
- ✓ de reprendre l'exécution d'un morceau suspendu par music freeze : music on,
- ✓ d'arrêter l'exécution d'un morceau : music off.

Pour bien comprendre la différence entre music freeze et music off, chargez un morceau en mémoire, activez-le avec une instruction music, puis tapez les instructions suivantes en mode direct :

```
music freeze<CR>
```

Le morceau s'arrête.

```
music on<CR>
```

Le morceau reprend à l'endroit où il s'est arrêté.

```
music off<CR>
```

Le morceau s'arrête

```
music on<CR>
```

Le morceau reprend au début.

Si la vitesse d'exécution d'un morceau ne vous convient pas, utilisez l'instruction tempo comme suit :

```
tempo <vit>
```

où <vit> est la vitesse d'exécution du morceau : entre 1 (très lent) et 100 (très rapide).

Cette instruction fonctionne en temps réel : si un morceau était en cours d'exécution lorsque tempo a été exécutée, sa vitesse varie instantanément.

De même, il vous est possible de transposer un morceau, c'est-à-dire de modifier la fréquence des notes émises, en tapant :

```
transpose <df>
```

où <df> est un nombre (offset) qui est ajouté à chaque note avant son émission. <df> peut être compris entre -90 et +90. A titre indicatif, une octave est représentée par un offset de 12.

Cette instruction fonctionne en temps réel : si un morceau était en cours d'exécution lorsque transpose a été exécutée, la fréquence de ses notes varie instantanément.

Nous terminerons ce chapitre avec l'instruction voice qui permet d'activer (voice on) ou de désactiver (voice off) une des voix d'un morceau en cours d'exécution.

Résumé

Pour créer des sons dans un programme BASIC STOS, utilisez les instructions suivantes :

- ✓ bell, boom et shoot pour produire des bruits prédéfinis,
- ✓ noise pour créer des souffles purs,
- ✓ play pour créer des sons purs,
- ✓ envel pour moduler les sons émis par une enveloppe de volume.

Pour activer un morceau qui se trouve dans une banque musicale, il suffit d'utiliser l'instruction music. Le tempo de la musique peut être modifié avec l'instruction tempo, et la hauteur de ses notes avec l'instruction transpose.

Chapitre 8

Les instructions particulières du STOS

8.1. Les accessoires

Le STOS n'est pas un simple langage. Il doit plutôt être considéré comme un environnement de développement puissant, d'une part par la variété et la richesse des instructions qui le composent, d'autre part par un ensemble d'outils d'aide à la conception appelés accessoires.

Un accessoire n'est ni plus ni moins qu'un programme écrit en BASIC STOS et sauvegardé sur disque avec une extension ACB. Les accessoires disponibles dans le STOS de base sont essentiellement les suivants :

- ✓ éditeur de sprites (SPRITE.ACB),
- ✓ éditeur de caractères (FONTS.ACB),
- ✓ éditeur de musique (MUSIC.ACB),
- ✓ éditeur d'icônes (ICON.ACB),
- ✓ compacteur d'écran (COMPACT.ACB).

Ils se trouvent sur la disquette "Accessoires".

Pour charger un accessoire en mémoire, vous devez utiliser l'instruction STOS accload, en précisant le nom de l'accessoire à charger, par exemple :

```
accload"music<CR>
```

Si vous désirez charger tous les accessoires présents sur une disquette, tapez simplement :

```
accload"*<CR>
```

Si la mémoire disponible n'est pas suffisante pour charger un accessoire, il est possible d'effacer tous les accessoires en mémoire avec l'instruction accnew.

Les cinq accessoires de base du STOS (cités plus haut) sont analysés dans le chapitre 2 de la première partie du livre. Je vous invite à vous y reporter si vous avez des difficultés à faire fonctionner un de ces accessoires.

8.2. Les banques mémoire

Les banques mémoire sont directement liées aux accessoires. Elles représentent un moyen simple et efficace pour interfacer les données créées par les accessoires avec les programmes BASIC STOS. Les banques mémoire sont de deux types :

- ✓ permanentes,
- ✓ temporaires.

Les banques permanentes font partie du programme STOS auquel elles sont rattachées. Elles sont sauvegardées et chargées avec lui. Elles concernent essentiellement les sprites, les icônes, les données musicales, et les écrans de type datascreen.

Les banques temporaires sont créées par les programmes STOS. Elles ne font pas partie intégrante du programme. Elles concernent essentiellement les écrans non permanents de type screen, et les menus créés par les programmes.

Sous STOS, il est à tout instant possible de connaître l'état des banques mémoire avec l'instruction list bank. Le jeu développé dans la dernière partie du livre utilise deux banques mémoire. Si vous le chargez et demandez la liste des banques définies, les informations suivantes seront affichées sur l'écran :

```
Banques mémoire réservées :
1 sprites S:$1ED100 E:$1EF900 L:$002800
3 music   S:$1EF900 E:$1EFF00 L:$000600
```

Les nombres hexadécimaux affichés après les lettres S, E et L sont (respectivement) l'adresse de début de banque, l'adresse de fin de banque, et la longueur de la banque.

Ces données peuvent être affichées en décimal avec l'instruction hexa off. L'instruction hexa on provoque à nouveau l'affichage en hexadécimal.

Les fonctions start et length permettent de connaître l'adresse de début et la longueur de la banque spécifiée. Leur syntaxe est la suivante :

```
<rés>=start(<b>)
```

et

```
<rés>=length(<b>)
```

où <b> est une banque mémoire,

et <rés> est l'adresse de début ou la longueur de la banque <b>.

Les accessoires SPRITE, MUSIC et ICON réservent automatiquement les banques qu'ils utilisent. Si, par contre, vous désirez utiliser une banque de type screen ou datascreen, vous devez la réserver avec une instruction reserve en précisant le numéro de la banque à réserver.

Par exemple :

```
reserve as screen 5
reserve as datascreen 6
```

Nous allons terminer ce paragraphe avec des instructions qui permettent de copier et d'effacer des banques mémoire.

Pour copier une banque dans une autre, il suffit d'utiliser l'instruction bcopy :

```
bcopy <source> to <dest>
```

où <source> est le numéro de la banque à copier,

et <dest> le numéro de la banque dans laquelle doit être copié <source>.

Si vous utilisez les facultés multi-programme du STOS, il vous sera peut-être utile de récupérer les banques d'un des programmes en mémoire et de les stocker dans le programme courant. L'instruction bgrab rend ce transfert aisé. Il vous suffit de préciser le numéro du programme où se trouvent les banques à transférer. Si vous ne désirez transférer qu'une banque, précisez laquelle :

```
bgrab <progsource>[,<b>]
```

où <progsource> est le numéro du programme où se trouvent les banques à transférer,

et <b> est une des banques de <progsource>.

Cette instruction peut également être étendue aux accessoires en mémoire. Pour charger dans le programme courant les banques de l'accessoire N, il suffit d'ajouter 4 au numéro de l'accessoire :

```
bgrab 4+N
```

Enfin, l'instruction erase vous permet d'effacer la banque spécifiée. Par exemple :

```
erase 4
```

efface la banque numéro 4.

Partie III

Création d'un programme de jeu de A à Z

Chapitre 9

Première approche. Généralités

Dans cette partie, nous allons appliquer tout ce qui a été vu jusqu'ici pour créer un jeu complet intégrant décors, musique, effets sonores et sprites animés au joystick et par l'ordinateur. Avant de passer au côté pratique, voici quelques considérations d'ordre général qui pourront vous servir lorsque vous concevrez vous-même vos jeux.

Avant de créer un jeu, vous devez avoir une idée précise :

- ✓ des personnages du jeu,
- ✓ des objets du jeu,
- ✓ de la règle du jeu, c'est-à-dire de l'interaction entre les divers personnages et objets du jeu.

Si vous vous mettez devant le clavier sans avoir réfléchi à ces éléments, vous risquez d'être assez vite déçu, voire même découragé...

Voici une méthode qui vous permettra de progresser sans ambiguïté et qui réduira la phase de programmation à son strict minimum.

9.1. Première étape : Réflexion subliminale.

Concevoir un jeu est facile. Concevoir un bon jeu est une autre paire de manches. Souvent, un petit détail suffit pour différencier un jeu d'un bon jeu : une musique agréable, un stress mesuré (le jeu ne doit être ni trop facile, ni trop compliqué pour "accrocher" le joueur), un bon graphisme, etc..

Réfléchissez le temps nécessaire pour trouver ce petit détail.

Choisissez le nombre de personnages à animer et leur quête : retrouver un diamant perdu, éliminer un maximum de cannibales, collectionner les objets perdus, ... A vous de trouver. Plus le nombre de personnages animés sera important, plus le jeu sera intéressant, mais aussi, ... plus il sera difficile à écrire. Dans vos premiers jeux, l'animation simultanée de trois ou quatre personnages sera bien suffisante.

Pensez au décor dans lequel vont évoluer les personnages. Souvent, un décor totalement abstrait est plus simple à concevoir qu'un décor plus réaliste, car l'oeil est plus critique sur les choses qu'il a l'habitude de voir que sur les autres...

Tout ce que je viens de dire peut se résumer en une seule phrase : concevez le scénario du jeu. Cette phase est très importante. Vous devez lui consacrer le temps nécessaire, sous peine de créer un jeu banal, voire inintéressant.

9.2. Seconde étape : Création du décor et des personnages

Si vous optez pour la création d'un décor abstrait répétitif, comme celui de "mouth trap" par exemple, je vous conseille de créer les éléments de base de votre décor sous la forme de sprites. Ce type de décor est celui utilisé par notre programme "Le chat et la souris". Il est décrit dans le second chapitre de cette partie.

Si vous désirez créer un décor non répétitif, vous pouvez utiliser un programme de dessin quelconque, à condition que les images créées soient au format Degas ou Néochrome. Ces images seront manipulées par le programme à travers des banques mémoire.

En ce qui concerne la création des personnages, vous devrez utiliser l'éditeur de sprites. Reportez-vous au second chapitre de la première partie pour apprendre à l'utiliser. Voici quelques conseils qui vous éviteront des pertes de temps :

✓ lorsque vous avez défini un sprite, réduisez au maximum la taille de la fenêtre dans laquelle il se trouve. Cela pour deux raisons :

❶ la taille du fichier MBK en sera d'autant réduite,

❷ ses déplacements et animations seront plus rapides.

✓ pensez à uniformiser la position du point chaud de chaque image constitutive de l'animation d'un personnage. Si vous ne le faites pas, vos personnages ... auront des gestes inconsidérés.

✓ utilisez plusieurs couleurs voisines, par exemple trois teintes de rouge, quatre teintes de gris, etc. pour améliorer la résolution apparente des personnages et objets dessinés,

✓ utilisez au moins trois sprites différents pour chaque séquence d'animation. Au dessous, les animations sont trop saccadées.

✓ récupérez dans l'éditeur de sprites des portions d'images qui vous intéressent, et regardez comment elles sont construites. Vous pourrez ainsi reproduire les techniques graphiques qu'elles utilisent,

✓ sauvegardez périodiquement vos sprites sur disque, et utilisez l'option "QUIT & GRAB" du menu DISQUE pour retourner au STOS. Les sprites seront ainsi disponibles sans aucune autre manipulation.

9.3. Troisième étape : Création de la musique de fond et des effets sonores

Si vous ne vous sentez pas l'âme d'un musicien, vous pouvez toujours récupérer les musiques implantées dans d'autres jeux écrits en STOS, si, bien sûr, vous ne destinez pas vos oeuvres à des opérations commerciales. Rien de plus simple : il suffit de charger le programme en mémoire et de sauvegarder la banque musicale qu'il utilise dans un fichier MBK.

Si vous êtes à même d'écrire des morceaux, évitez les airs trop répétitifs, surtout s'ils sont exécutés pendant toute la durée du jeu. Les instructions transpose et tempo peuvent être des moyens simples et économiques pour modifier l'allure de vos morceaux, par exemple, lorsque le temps maximum imparti pour réaliser une action est pratiquement écoulé...

En ce qui concerne les effets sonores, les instructions boom, shoot et bell devraient suffire dans bien des cas. Utilisez au besoin les instructions noise et envel pour créer des bruits plus spécifiques. Reportez-vous au chapitre 5 de la seconde partie pour avoir plus de détails.

9.4. Quatrième étape : La programmation

Contrairement à ce que vous pensez, cette étape n'est pas forcément la plus longue, surtout si vous avez une idée très précise du scénario. Au besoin, dessinez l'organigramme du jeu pour y voir plus clair.

Avant de commencer à écrire la première ligne de votre programme, chargez tous les fichiers qui doivent être implantés dans des banques (écrans, sprites, musique, etc.), en mode direct, à l'aide d'instructions load, par exemple :

```
load "sprite.mbk"<CR>
load "musique.mbk"<CR>
reserve as datascreen 5<CR>
load "ecran.mbk",5<CR>
```

Sauvegardez le programme sur disque à l'aide d'une instruction save. Par exemple :

```
save "jeu<CR>
```

ceci, pour sauvegarder les banques mémoire sur disque.

Structurez le plus possible votre programme en utilisant les possibilités du STOS au niveau des sous-programmes. De cette façon, les éventuels mauvais fonctionnements ou erreurs de programmation seront facilement identifiés.

Testez votre programme fonction par fonction, par exemple en mettant d'abord au point le déplacement des personnages, puis l'affichage des décors et des objets, cela pour éviter une trop longue mise au point des erreurs qui ne manqueront pas d'arriver.

Chapitre 10

Décor et personnages

10.1. Utilisation de l'éditeur de sprites

Tous les sprites utilisés dans le programme "Le chat et la souris" ont été définis dans le chapitre 2 de la partie 1. Si ce n'est pas encore fait, créez tous ces sprites et stockez-les sur disque dans le fichier "sprite.mbk".

Veillez en particulier :

✓ à utiliser les couleurs préconisées,

✓ à définir les sprites dans l'ordre indiqué :
- 1 à 3 = souris de profil gauche,
- 4 à 6 = souris de profil droit,
- 7 à 10 = souris de dos,
- 11 à 14 = souris de face,
- 15 à 17 = chat de profil gauche,
- 18 à 20 = chat de profil droit,
- 21 = biscuit,
- 22 et 23 = fromage,
- 24 à 26 = décor des pièces,

- 27 et 30 = éléments de mur,
- 28 = mur avec un trou ouvert,
- 29 = mur avec un trou fermé.

✓ à bien définir le point chaud de chaque sprite :
- les sprites 24, 25, 26, 27 et 30 ont un point chaud quelconque,
- les sprites 28 et 29 ont un point chaud en bas et au centre,
- les autres sprites ont un point chaud en leur centre.

10.2. Intégration des sprites dans le programme

Dans le programme "Le chat et la souris", trois types de sprites sont utilisés :

✓ les sprites de décor qui sont les éléments de base du décor,

✓ les sprites objets qui représentent des objets à ramasser. Ces sprites ne sont ni déplacés ni animés,

✓ les sprites chat et souris. Le chat est déplacé et animé par l'ordinateur. La souris est déplacée et animée par le joueur.

Le paragraphe 10.3. détaille la méthode utilisée pour créer le décor du jeu.

Les sprites objets sont placés sur l'écran grâce à l'instruction sprite. Ils sont toujours au nombre de 10.

Les sprites chat et souris sont placés sur l'écran avec des instructions sprite. Ils sont animés à l'aide d'instructions anim et déplacés sur l'écran grâce à des instructions move. Les collisions entre les divers sprites (objets et personnages) sont détectées par l'instruction collide.

10.3. Création du décor

Le décor du programme "Le chat et la souris" est créé à partir de sprites stockés dans le fichier SPRITE.MBK. Nous allons étudier en détail la technique utilisée.

La constitution du décor s'effectue en trois temps.

Dans un premier temps, les couleurs utilisées dans le générateur de sprites sont affectées à la palette courante à l'aide d'une instruction palette.

Dans un second temps, la partie centrale du décor est constituée à partir d'un des trois sprites réservés à cet usage (24 à 26). Le sprite choisi est placé sur l'écran à l'aide d'une instruction sprite. Il est recopié autant de fois que nécessaire, verticalement, puis horizontalement à l'aide d'instructions screen copy. Le sprite utilisé pour créer le décor est alors déplacé à l'extérieur de l'écran et une instruction screen copy le remplace par son équivalent non sprite.

Dans un troisième temps, la partie centrale du décor est encadrée par le motif en briques représenté dans les sprites 27 à 30. La même technique est utilisée: un sprite est placé sur l'écran, il est recopié autant de fois que nécessaire. Il est enfin placé hors de l'écran et remplacé par sa copie non sprite.

Chapitre 11

Musique

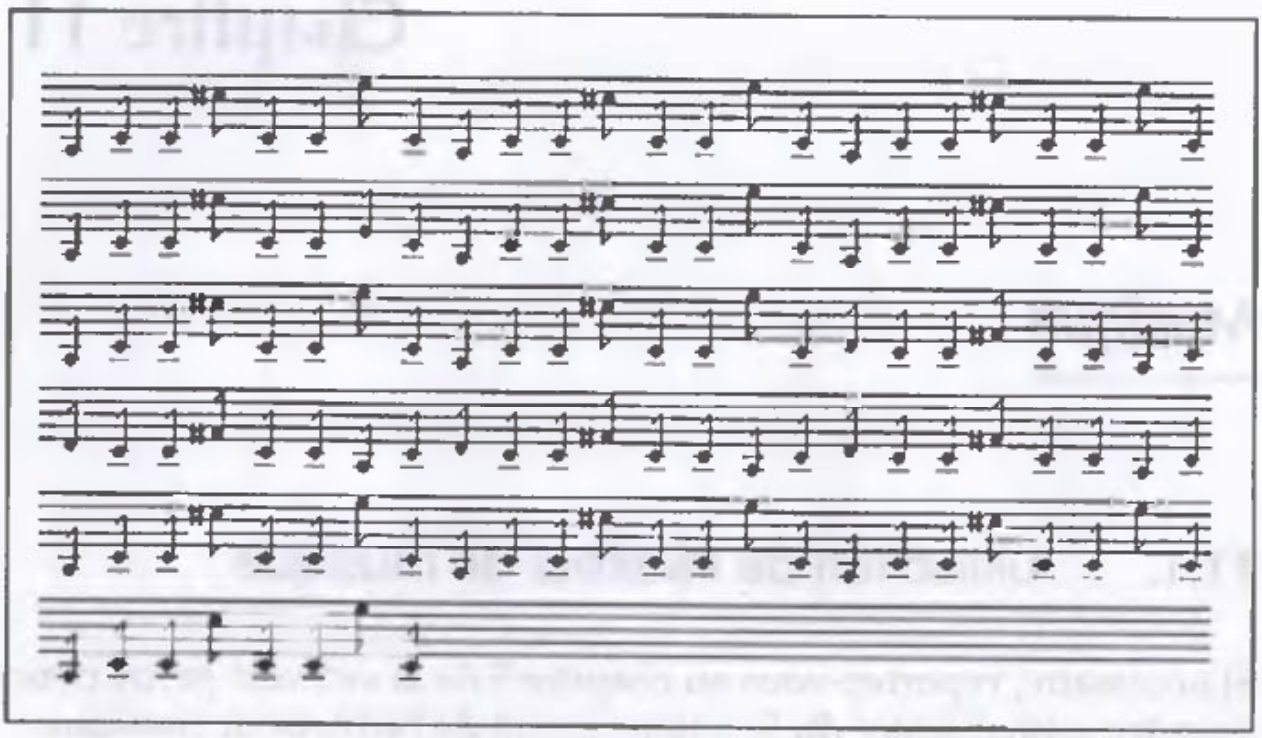
11.1. Utilisation de l'éditeur de musique

Si nécessaire, reportez-vous au chapitre 5 de la seconde partie pour prendre connaissance du fonctionnement de l'éditeur de musique.

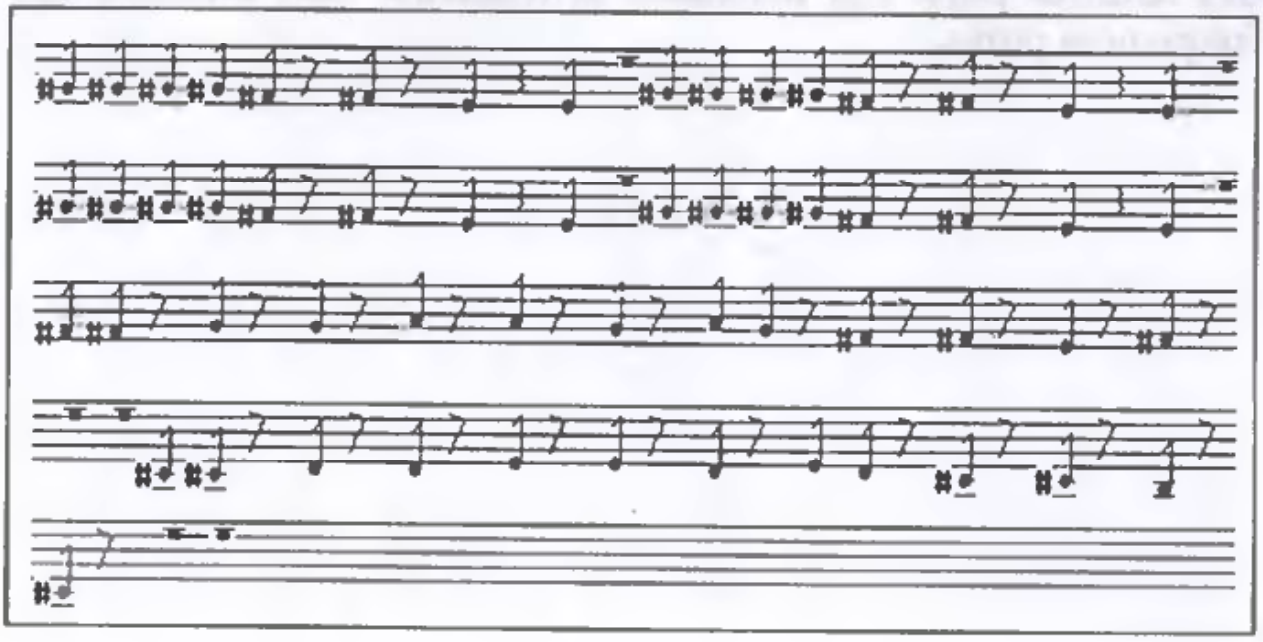
Le morceau d'accompagnement de notre jeu utilise les trois voix du générateur sonore. La première voix génère des fréquences pures ou des souffles purs. Les deuxième et troisième voix génèrent des fréquences pures.

Pour les amateurs, voici la partition du morceau :

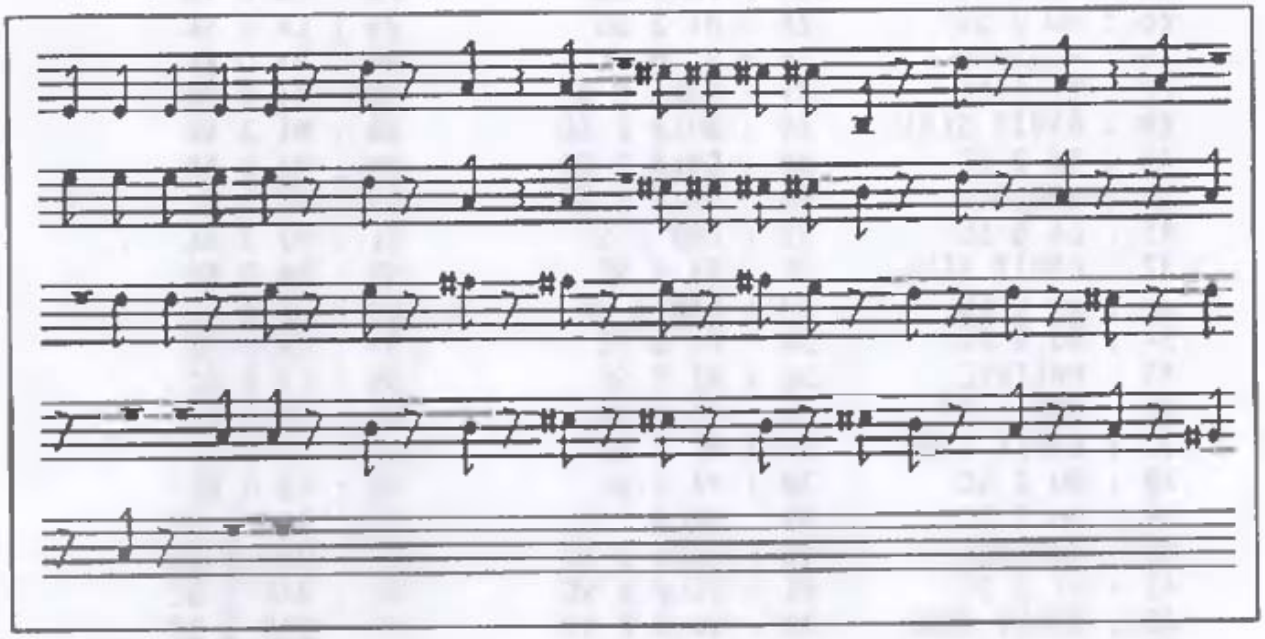
Voix 1



Voix 2



Voix 3



Entrez les notes correspondantes dans l'éditeur de musique :

Voix 1	Voix 2	Voix 3
1 : ENVEL 2	1 : ENVEL 2	1 : ENVEL 2
2 : REPETER 0 3	2 : REPETER 0 3	2 : REPETER 0 3
3 : LA 0 SC	3 : SOL# 2 SC	3 : MI 2 SC
4 : BRUIT SEUL	4 : SOL# 2 SC	4 : MI 2 SC
5 : DO 0 SC	5 : SOL# 2 SC	5 : MI 2 SC
6 : DO 0 SC	6 : SOL# 2 SC	6 : MI 2 SC
7 : MUSIQUE	7 : FA# 2 SC	7 : MI 2 SC
8 : DO# 1 SC	8 : PA 0 SC	8 : PA 0 SC
9 : BRUIT SEUL	9 : FA # 2 SC	9 : RE 3 SC
10 : DO 0 SC	10 : PA 0 SC	10 : PA 0 SC
11 : DO 0 SC	11 : MI 2 SC	11 : LA 2 SC
12 : MUSIQUE	12 : PA 0 NR	12 : PA 0 NR
13 : MI 1 SC	13 : MI 2 SC	13 : LA 2 SC
14 : BRUIT SEUL	14 : PA 0 BL	14 : PA 0 BL
15 : DO 0 SC	15 : SOL# 2 SC	15 : DO# 3 SC
16 : MUSIQUE	16 : SOL# 2 SC	16 : DO# 3 SC
17 : LA 0 SC	17 : SOL# 2 SC	17 : DO# 3 SC
18 : BRUIT SEUL	18 : SOL# 2 SC	18 : DO# 3 SC
19 : DO 0 SC	19 : FA# 2 SC	19 : SI 2 SC
20 : DO 0 SC	20 : PA 0 SC	20 : PA 0 SC
21 : MUSIQUE	21 : FA# 2 SC	21 : RE 3 SC
22 : DO# 1 SC	22 : PA 0 SC	22 : PA 0 SC
23 : BRUIT SEUL	23 : MI 2 SC	23 : LA 2 SC

	Voix 1	Voix 2	Voix 3
24	DO 0 SC	PA 0 SC	PA 0 NR
25	DO 0 SC	MI 2 SC	LA 2 SC
26	MUSIQUE	PA 0 BL	PA 0 BL
27	MI 1 SC	SOL# 2 SC	MI 3 SC
28	BRUIT SEUL	SOL# 2 SC	MI 3 SC
29	DO 0 SC	SOL# 2 SC	MI 3 SC
30	MUSIQUE	SOL# 2 SC	MI 3 SC
31	LA 0 SC	FA# 2 SC	MI 3 SC
32	BRUIT SEUL	PA 0 SC	PA 0 SC
33	DO 0 SC	FA# 2 SC	RE 3 SC
34	DO 0 SC	PA 0 SC	PA 0 SC
35	MUSIQUE	MI 2 SC	LA 2 SC
36	DO# 1 SC	PA 0 NR	PA 0 NR
37	BRUIT SEUL	MI 2 SC	LA 2 SC
38	OO 0 SC	PA 0 BL	PA 0 BL
39	DO 0 SC	SOL# 2 SC	DO# 3 SC
40	MUSIQUE	SOL# 2 SC	DO# 3 SC
41	MI 1 SC	SOL# 2 SC	DO# 3 SC
42	BRUIT SEUL	SOL# 2 SC	OO# 3 SC
43	DO 0 SC	FA# 2 SC	SI 2 SC
44	MUSIQUE	PA 0 SC	PA 0 SC
45	LA 0 SC	FA# 2 SC	RE 3 SC
46	BRUIT SEUL	PA 0 SC	PA 0 SC
47	DO 0 SC	MI 2 SC	LA 2 SC
48	DO 0 SC	PA 0 NR	PA 0 NR
49	MUSIQUE	MI 2 SC	LA 2 SC
50	DO# 1 SC	PA 0 SC	PA 0 SC
51	BRUIT SEUL	PA 0 SC	PA 0 SC
52	DO 0 SC	PA 0 SC	PA 0 SC
53	DO 0 SC	PA 0 SC	PA 0 SC
54	MUSIQUE	FA# 2 SC	RE 3 SC
55	MI 1 SC	FA# 2 SC	RE 3 SC
56	BRUIT SEUL	PA 0 SC	PA 0 SC
57	DO 0 SC	SOL 2 SC	MI 3 SC
58	MUSIQUE	PA 0 SC	PA 0 SC
59	LA 0 SC	SOL 2 SC	MI 3 SC
60	BRUIT SEUL	PA 0 SC	PA 0 SC
61	DO 0 SC	LA 2 SC	FA# 3 SC
62	DO 0 SC	PA 0 SC	PA 0 SC
63	MUSIQUE	LA 2 SC	FA# 3 SC
64	DO# 1 SC	PA 0 SC	PA 0 SC
65	BRUIT SEUL	SOL 2 SC	MI 3 SC
66	DO 0 SC	PA 0 SC	PA 0 SC
67	DO 0 SC	LA 2 SC	FA# 3 SC
68	MUSIQUE	SOL 2 SC	MI 3 SC
69	MI 1 SC	PA 0 SC	PA 0 SC
70	BRUIT SEUL	FA# 2 SC	RE 3 SC

Voix 1	Voix 2	Voix 3
71 : DO 1 SC	71 : PA 0 SC	71 : PA 0 SC
72 : MUSIQUE	72 : FA# 2 SC	72 : RE 3 SC
73 : LA 0 SC	73 : PA 0 SC	73 : PA 0 SC
74 : BRUIT SEUL	74 : MI 2 SC	74 : DO# 3 SC
75 : DO 0 SC	75 : PA 0 SC	75 : PA 0 SC
76 : DO 0 SC	76 : FA# 2 SC	76 : RE 3 SC
77 : MUSIQUE	77 : PA 0 RD	77 : PA 0 RD
78 : DO# 1 SC	78 : PA 0 SC	78 : PA 0 SC
79 : BRUIT SEUL	79 : DO# 2 SC	79 : LA 2 SC
80 : DO 0 SC	80 : DO# 2 SC	80 : LA 2 SC
81 : DO 0 SC	81 : PA 0 SC	81 : PA 0 SC
82 : MUSIQUE	82 : RE 2 SC	82 : SI 2 SC
83 : MI 1 SC	83 : PA 0 SC	83 : PA 0 SC
84 : BRUIT SEUL	84 : RE 2 SC	84 : SI 2 SC
85 : DO 0 SC	85 : PA 0 SC	85 : PA 0 SC
86 : MUSIQUE	86 : MI 2 SC	86 : DO# 3 SC
87 : LA 0 SC	87 : PA 0 SC	87 : PA 0 SC
88 : MUSIQUE	88 : MI 2 SC	88 : OO# 3 SC
89 : DO 0 SC	89 : PA 0 SC	89 : PA 0 SC
90 : DO 0 SC	90 : RE 2 SC	90 : SI 2 SC
91 : MUSIQUE	91 : PA 0 SC	91 : PA 0 SC
92 : DO# 1 SC	92 : MI 2 SC	92 : DO# 3 SC
93 : BRUIT SEUL	93 : RE 2 SC	93 : SI 2 SC
94 : DO 0 SC	94 : PA 0 SC	94 : PA 0 SC
95 : DO 0 SC	95 : DO# 2 SC	95 : LA 2 SC
96 : MUSIQUE	96 : PA 0 SC	96 : PA 0 SC
97 : MI 1 SC	97 : DO# 2 SC	97 : LA 2 SC
98 : BRUIT SEUL	98 : PA 0 SC	98 : PA 0 SC
99 : DO 0 SC	99 : SI 1 SC	99 : SOL# 2 SC
100 : MUSIQUE	100 : PA 0 SC	100 : PA 0 SC
101 : LA 0 SC	101 : DO# 2 SC	101 : LA 2 SC
102 : BRUIT SEUL	102 : PA 0 RD	102 : PA 0 RD
103 : DO 0 SC	103 : PA 0 SC	103 : PA 0 SC
104 : DO 0 SC		
105 : MUSIQUE		
106 : DO# 1 SC		
107 : BRUIT SEUL		
108 : DO 0 SC		
109 : DO 0 SC		
110 : MUSIQUE		
111 : MI 1 SC		
112 : BRUIT SEUL		
113 : DO 0 SC		
114 : MUSIQUE		
115 : RE 0 SC		
116 : BRUIT SEUL		
117 : OO 0 SC		

Voix 1

```
118 : DO 0 SC
119 : MUSIQUE
120 : FA# 0 SC
121 : BRUIT SEUL
122 : OO 0 SC
123 : DO 0 SC
124 : MUSIQUE
125 : LA 0 SC
126 : BRUIT SEUL
127 : DO 0 SC
128 : MUSIQUE
129 : RE 0 SC
130 : BRUIT SEUL
131 : DO 0 SC
132 : DO 0 SC
133 : MUSIQUE
134 : FA# 0 SC
135 : BRUIT SEUL
136 : DO 0 SC
137 : DO 0 SC
138 : MUSIQUE
139 : LA 0 SC
140 : BRUIT SEUL
141 : DO 0 SC
142 : MUSIQUE
143 : RE 0 SC
144 : BRUIT SEUL
145 : DO 0 SC
146 : DO 0 SC
147 : MUSIQUE
148 : FA# 0 SC
149 : BRUIT SEUL
150 : DO 0 SC
151 : DO 0 SC
152 : MUSIQUE
153 : LA 0 SC
154 : BRUIT SEUL
155 : OO 0 SC
156 : MUSIQUE
157 : RE 0 SC
158 : BRUIT SEUL
159 : DO 0 SC
160 : DO 0 SC
161 : MUSIQUE
162 : FA# 0 SC
163 : BRUIT SEUL
164 : DO 0 SC
```

Voix 1

```
165 : DO 0 SC
166 : MUSIQUE
167 : LA 0 SC
168 : BRUIT SEUL
169 : DO 0 SC
170 : MUSIQUE
171 : LA 0 SC
172 : BRUIT SEUL
173 : DO 0 SC
174 : DO D SC
175 : MUSIQUE
176 : DO# 1 SC
177 : BRUIT SEUL
178 : DO 0 SC
179 : DO 0 SC
180 : MUSIQUE
181 : MI 1 SC
182 : BRUIT SEUL
183 : DO 0 SC
184 : MUSIQUE
185 : LA D SC
186 : BRUIT SEUL
187 : DO 0 SC
188 : DO 0 SC
189 : MUSIQUE
190 : DO# 1 SC
191 : BRUIT SEUL
192 : DO 0 SC
193 : DO 0 SC
194 : MUSIQUE
195 : MI 1 SC
196 : BRUIT SEUL
197 : DO D SC
198 : MUSIQUE
199 : LA 0 SC
200 : BRUIT SEUL
201 : DO 0 SC
202 : DO 0 SC
203 : MUSIQUE
204 : DO# 1 SC
205 : BRUIT SEUL
206 : DD 0 SC
207 : DO 0 SC
208 : MUSIQUE
209 : MI 1 SC
210 : BRUIT SEUL
211 : DO 0 SC
```

Voix 1

```
212 : MUSIQUE
213 : LA 0 SC
214 : BRUIT SEUL
215 : DO 0 SC
216 : DO 0 SC
217 : MUSIQUE
218 : DO# 1 SC
219 : BRUIT SEUL
220 : DO 0 SC
221 : DO 0 SC
222 : MUSIQUE
223 : MI 1 SC
224 : BRUIT SEUL
225 : DO 0 SC
```

Sauvegardez ces données dans la banque "muse.mkb" en sélectionnant l'option "Sauver la banque" du menu BANQUE.

Quelques conseils

✓ n'hésitez pas à faire plusieurs sauvegardes pendant la saisie en sélectionnant MUSIQUE/STOCKER musique, puis BANQUE/Sauver la banque dans le menu de l'éditeur de musique.

✓ vérifiez périodiquement que les notes entrées se trouvent à la même position que celles du livre.

✓ soyez très scrupuleux sur les durées des pauses. Une seule erreur peut en effet dénaturer le morceau.

11.2. Intégration du morceau dans le programme de jeu

Lorsque vous sauvegardez un programme STOS avec une commande du type :

```
save "monprog.bas"<CR>
```

toutes les banques qui se trouvaient en mémoire au moment de la sauvegarde sont également sauvegardées.

Tout chargement ultérieur du programme provoquera également le chargement des banques sauvegardées.

Pour vous en persuader, tapez les instructions suivantes, en mode direct :

```
new<CR>                --> Efface la mémoire (programme et banques),
load"muse.mbk"<CR>     --> Charge la banque mucicale
```

Entrez ensuite les instructions ci-dessous en mode programme. Leur but est l'activation du premier morceau de la banque musicale muse.mbk :

```
10 music 1
20 transpose 12
```

Sauvegardez le programme sur disque sous le nom "essai.bas" :

```
save"essai.bas"<CR>
```

Effacez la mémoire :

```
new<CR>
```

A ce niveau, le programme d'activation de la musique et la banque musicale ne sont plus en mémoire. Lancez le programme "essai.bas" en tapant en mode direct :

```
run "essai.bas<CR>
```

La musique est activée. Listez le contenu de la mémoire en tapant list en mode direct. Le listing du programme et les informations concernant la banque musicale sont affichés sur l'écran :

```
list
10 music 1
20 transpose 12

Banques mémoire réservées :
3 music  S:$1DF500 E:$1DFB00 L:$000600
```

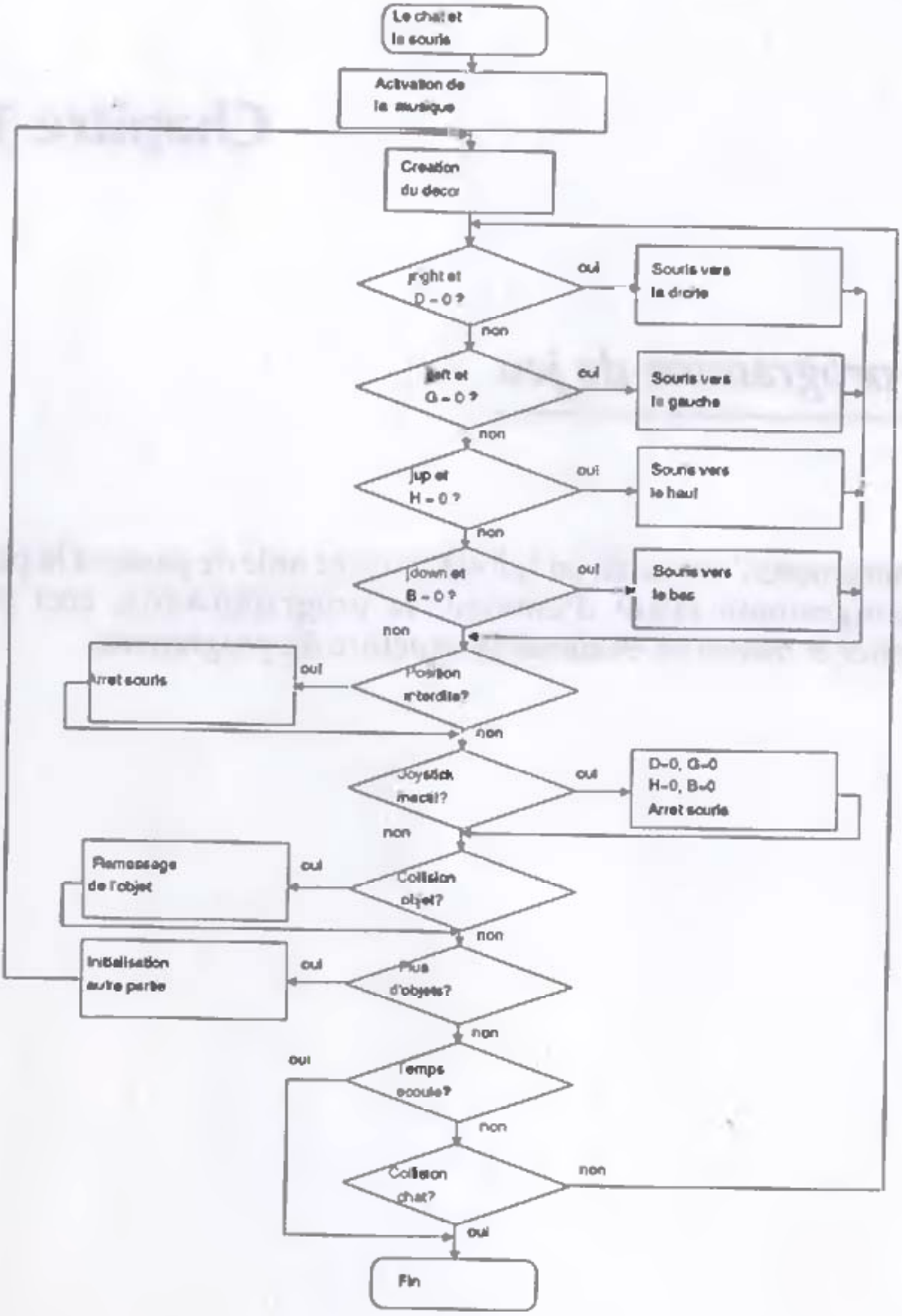
De ce qui précède, nous pouvons en conclure que l'intégration d'une banque musicale à un programme STOS est un jeu d'enfant. Avant de passer au programme proprement dit, précisons encore un point. Si vous désirez émettre des bruits prédéfinis tels que boom ou shoot ou encore des bruits créés de toutes pièces avec les instructions noise et envel, il vous suffit de les intégrer à l'endroit voulu dans le programme. La musique sera automatiquement suspendue le temps nécessaire à l'émission du bruit.

Chapitre 12

Le programme de jeu

Comme nous l'avons dit en 1, il est souvent utile de passer à la phase organigramme avant d'entamer la programmation, ceci pour clarifier et mettre en évidence la structure du programme.

Voici l'organigramme du programme "Le chat et la souris" :



Avant de saisir le programme, chargez les deux banques mémoire en tapant en mode direct :

```
load "sprite.mbk"<CR>
load "muse.mbk"<CR>
```

Le listing du programme découle directement de l'organigramme ci-dessous :

```
1000 rem -
1010 rem Le chat et la souris
1020 rem -
1030 key off
1040 mode 0
1050 hide : curs off : click off
1060 SC=0 : rem Score en début de partie
1070 music 1 : transpose 12
1080 dim T(14),SX(14),SY(14)
1090 rem --
1100 rem Constitution du décor
1110 rem --
1120 palette 0,$777,$762,$57,$432,$444,$555,$666,$555,
$222,$733,$72,$421,$311,$753,$531
1130 tempo 50
1140 Z=int(rnd(2))+1
1150 sprite 1,16,8,23+Z : wait 2
1160 for I=1 to 4
1170 screen copy logic,16,8,48,40 to logic,16,32*I+8
1180 next I
1190 for I=1 to 8
1200 screen copy logic,16,8,48,168 to logic,32*I+16,8
1210 next I
1220 rem --
1230 rem Cadre autour du décor
1240 rem --
1250 sprite 1,-100,0,27 : wait 2
1260 screen copy logic,16,40,48,72 to logic,16,8
1270 sprite 1,0,0,27 : wait 2
1280 for I=1 to 6
1290 screen copy logic,0,0,48,25 to logic,48*I,0
1300 next I
1310 sprite 1,-100,0,30 : wait 5
1320 screen copy logic,48,0,96,25 to logic,0,0
1330 sprite 1,0,150,30 : wait 2
1340 for I=1 to 5
1350 screen copy logic,0,150,16,200 to logic,0,25*I
1360 next I
1370 for I=1 to 6
```

```
1380 screen copy logic,0,25,16,50 to logic,310,25*I
1390 next I
1400 sprite 1,-100,0,27 : wait 2
1410 screen copy logic,0,25,16,50 to logic,0,150
1420 sprite 1,0,150,27 : wait 2
1430 for I=1 to 6
1440 screen copy logic,0,150,48,175 to logic,48*I,150
1450 next I
1460 sprite 1,-100,0,1 : wait 2
1470 screen copy logic,48,150,96,175 to logic,0,150
1480 screen copy logic,0,180,48,205 to logic,144,6
1490 screen copy physic to back
1500 timer=0 : TEMPS=0 : OK=0
1510 gosub 1880 : rem Initialisation des paramètres de la partie
1520 locate 7,23 : print "Temps              Score";SC
1530 NAR=10 : rem Nombre d'objets à ramasser
1540 for I=4 to 13
1550 sprite I,SX(I),SY(I),T(I)+20
1560 next I
1570 sprite 1,160,76,1 : sprite 15,167,24,29
1580 anim 1,"(1,10)(2,10)(3,10)L"
1590 G=0 : D=0 : H=0 : B=0 : rem Indicateurs de déplacement
1600 rem
1610 rem Boucle principale du jeu
1620 rem
1630 repeat
1640 if jright and D=0 then move y 1,"(3,0,1)" : move x
1,"(3,2,1)L" : move on : anim 1,"(4,10)(5,10)(6,10)L" : anim on
: D=1
1650 if jleft and G=0 then move y 1,"(3,0,1)" : move x
1,"(3,-2,1)L" : move on : anim 1,"(1,10)(2,10)(3,10)L" : anim on
: G=1
1660 if jup and H=0 then move x 1,"(3,0,1)" : move y
1,"(3,-2,1)L" : move on : anim 1,"(7,10)(8,10)(9,10)(10,10)L" :
anim on : H=1
1670 if jdown and B=0 then move x 1,"(3,0,1)" : move y
1,"(3,2,1)L" : move on : anim 1,"(11,10)(12,10)(13,10)(14,10)L"
: anim on : B=1
1680 if jright and x sprite(1)>283 then move freeze (1)
1690 if jleft and x sprite(1)<36 then move freeze (1)
1700 if jup and y sprite(1)<28 then move freeze (1)
1710 if jdown and y sprite(1)>136 then move freeze (1)
1720 if joy=0 then G=0 : D=0 : H=0 : B=0 : move freeze (1) :
anim freeze (1)
1730 A=collide(1,10,10) : C=0
1740 for I=0 to 15
1750 if btst(I,A)=-1 then C=I
1760 next I
1770 if (C<>0) and (C<14) then sprite C,-100,0,0 : NAR=NAR-1 :
SC=SC+10*T(C) : locate 29,23 : print SC
```

```
1780 if NAR=0 then sprite 15,167,24,28 : OK=1
1790 XX=int(timer/50) : if XX<>TEMPS then TEMPS=XX : locate
12,23 : print 40-XX;"  "
1800 CC=collide(1,16,12)
1810 if (CC=3276B) and (OK=1) then OK=2
1820 if TEMPS>30 then tempo 60
1830 until (CC=4) or (CC=16384) or (OK=2) or (TEMPS=40)
1840 if OK=2 then cls : goto 1130
1850 boom : default
1860 print "Scrore final";SC;" au tableau":NIV
1870 end
1880 rem --
1890 rem Initialisation des paramètres de la partie en cours
1900 rem --
1910 NIV=NIV+1 : rem Passage au prochain niveau
1920 if NIV>1 then 2070
1930 rem ----
1940 rem Niveau 1
1950 rem ----
1960 for I=4 to 14 : T(I)=1 : next I
1970 for I=0 to 4
1980 for J=0 to 1
1990 SX(4+J+I*2)=50+I*50
2000 SY(4+J+I*2)=50+J*50
2010 next J
2020 next I
2030 sprite 2,270,95,15
2040 move x 2,"270(2,-2,125)L" : move on
2050 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
2060 goto 2610
2070 if NIV>2 then 2230
2080 rem ----
2090 rem Niveau 2
2100 rem ----
2110 for I=4 to 14 : T(I)=2 : next I
2120 SX(4)=100 : SX(5)=200 : SX(6)=50 : SX(7)=150 : SX(8)=250
2130 SX(9)=100 : SX(10)=200 : SX(11)=50 : SX(12)=150 : SX(13)=250
2140 SY(4)=30 : SY(5)=30 : SY(6)=60 : SY(7)=60 : SY(8)=60
2150 SY(9)=90 : SY(10)=90 : SY(11)=120 : SY(12)=120 : SY(13)=120
2160 sprite 2,270,95,15
2170 move x 2,"270(2,-2,125)L" : move on
2180 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
2190 sprite 14,40,40,18
2200 move x 14,"40(2,2,125)L" : move on
2210 anim 14,"(18,10)(19,10)(20,10)(19,10)L" : anim on
2220 goto 2610
2230 if NIV>3 then 2390
2240 rem ----
2250 rem Niveau 3
2260 rem ----
```

```
2270 for I=4 to 14 : T(I)=3 : next I
2280 SX(4)=50 : SX(5)=250 : SX(6)=100 : SX(7)=200 : SX(8)=150
2290 SX(9)=100 : SX(10)=200 : SX(11)=50 : SX(12)=250 : SX(13)=150
2300 SY(4)=50 : SY(5)=50 : SY(6)=70 : SY(7)=70 : SY(8)=90
2310 SY(9)=110 : SY(10)=110 : SY(11)=130 : SY(12)=130 : SY(13)=90
2320 sprite 2,270,95,15
2330 move x 2,"270(2,-2,125)L" : move on
2340 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
2350 sprite 14,40,40,18
2360 move x 14,"40(2,2,125)L" : move on
2370 anim 14,"(18,10)(19,10)(20,10)(19,10)L" : anim on
2380 goto 2610
2390 rem -
2400 rem Niveau supérieur à 3
2410 rem -
2420 for I=4 to 14 : T(I)=int(rnd(2))+1 : next I
2430 for I=4 to 8
2440 BAD=0
2450 CX=int(rnd(100))+25 : CY=int(rnd(80))+50
2460 if I=4 then 2500
2470 for J=4 to I-1
2480 if (abs(SX(J)-CX)<10) or (abs(SY(J)-CY)<10) then BAD=1
2490 next J
2500 if BAD=0 then SX(I)=CX : SY(I)=CY else 2440
2510 next I
2520 for I=9 to 13
2530 SX(I)=SX(I-5)+150 : SY(I)=SY(I-5)
2540 next I
2550 sprite 2,270,95,15
2560 move x 2,"270(2,-2,125)L" : move on
2570 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
2580 sprite 14,40,40,18
2590 move x 14,"40(2,2,125)L" : move on
2600 anim 14,"(18,10)(19,10)(20,10)(19,10)L" : anim on
2610 return
```

Nous allons maintenant décortiquer ce programme.

Les premières lignes du programme préparent les périphériques : Le menu des touches de fonctions est effacé, l'écran est initialisé en mode basse résolution, les curseurs souris et clavier sont effacés, et le cliquetis du clavier est supprimé :

```
1030 key off
1040 mode 0
1050 hide : curs off : click off
```

Le programme se poursuit par l'initialisation et la déclaration de certaines variables, et par le déclenchement du fond sonore :

```
1060 SC=0 : rem Score en début de partie
1070 music 1 : transpose 12
1080 dim T(14),SX(14),SY(14)
```

Le décor du jeu est ensuite créé. Dans un premier temps, la palette courante est initialisée aux couleurs utilisées dans l'éditeur de sprites :

```
1120 palette 0,$777,$762,$57,$432,$444,$555,$666,$555,$222,
$733,$72,$421,$311,$753,$531
```

Le sprite à partir duquel sera créé la partie centrale du décor est choisi aléatoirement et affiché en haut et à gauche de l'écran :

```
1140 Z=int(rnd(2))+1
1150 sprite 1,16,8,23+Z : wait 2
```

Ce sprite est recopié autant de fois que nécessaire pour remplir la partie centrale de l'écran :

```
1160 for I=1 to 4
1170 screen copy logic,16,8,48,40 to logic,16,32*I+8
1180 next I
1190 for I=1 to 8
1200 screen copy logic,16,8,48,168 to logic,32*I+16,8
1210 next I
```

Le sprite qui a permis de créer le décor est placé hors de l'écran. Il est remplacé par la partie du décor correspondante.

```
1250 sprite 1,-100,0,27 : wait 2
1260 screen copy logic,16,40,48,72 to logic,16,8
```

Les éléments constitutifs du mur sont alors copiés sur l'écran selon la même méthode (lignes 1270 à 1490).

Les principales variables du programme sont alors initialisées:

```
1500 timer=0 : TEMPS=0 : OK=0
1530 NAR=10 : rem Nombre d'objets à ramasser
1590 G=0 : D=0 : H=0 : B=0 : rem Indicateurs de déplacement
```

et la procédure responsable de l'initialisation des paramètres de la partie est appelée :

```
1510 gosub 1880 : rem Initialisation des paramètres de la partie
```

Les dix objets à ramasser sont affichés sur l'écran :

```
1540 for I=4 to 13
1550 sprite I,SX(I),SY(I),T(I)+20
1560 next I
```

Les sprites souris et porte fermée sont affichés sur l'écran. La chaîne d'animation de la souris est définie :

```
1570 sprite 1,160,76,1 : sprite 15,167,24,29
1580 anim 1,"(1,10)(2,10)(3,10)L"
```

Le programme se poursuit par la boucle principale.

Dans cette boucle sont testés :

- ✓ le joystick. S'il est actionné vers le haut, le bas, la droite ou la gauche, la souris est déplacée et animée (lignes 1640 à 1670). S'il est en position de repos, les mouvements de la souris sont suspendus (lignes 1680 à 1710) et les variables de déplacement sont remises à zéro (ligne 1720).

- ✓ les collisions de la souris avec un autre sprite.

```
1730 A=collide(1,10,10) : C=0
```

Si une collision s'est produite avec un objet à ramasser, cet objet disparaît de l'écran :

```
1740 for I=0 to 15
1750 if btst(I,A)=-1 then C=I
1760 next I
1770 if (C<>0) and (C<14) then sprite C,-100,0,0 : NAR=NAR-1 :
SC=SC+10*T(C) : locate 29,23 : print SC
```

Si le nombre d'objets à ramasser est nul, la porte permettant d'accéder aux autres tableaux est ouverte :

```
1780 if NAR=0 then sprite 15,167,24,28 : OK=1
```

Le compteur de temps est mis à jour et affiché en bas de l'écran :

```
1790 XX=int(timer/50) : if XX<>TEMPS then TEMPS=XX : locate
12,23 : print 40-XX;"  "
```

Si une collision s'est produite avec un des sprites "chat", la variable OK est initialisée pour provoquer la fin du pogramme :

```
1800 CC=collide(1,16,12)
1810 if (CC=32768) and (OK=1) then OK=2
```

Si le compteur de temps est supérieur à 30, c'est à dire s'il reste moins de dix secondes avant la fin du temps imparti, le tempo de la musique est accéléré :

```
1820 if TEMPS>30 then tempo 60
```

La boucle principale prend fin dans l'un des cas suivants :

✓ la souris est entrée en collision avec un des chats,

✓ la souris est passée dans le trou pour accéder au tableau suivant,

✓ le temps imparti est écoulé :

```
1830 until (CC=4) or (CC=16384) or (OK=2) or (TEMPS=40)
```

Si la souris est passée au tableau suivant, l'écran est effacé, et le contrôle est donné à la ligne 1130 :

```
1840 if OK=2 then cls : goto 1130
```

Dans le cas contraire, la partie prend fin. Un bruit prédéfini est émis, l'écran est initialisé et le score final est affiché.

```
1850 boom : default
1860 print "Scrore final";SC;" au tableau";NIV
1870 end
```

Le sous-programme qui débute en ligne 1910 calcule la position des sprites objets et provoque l'affichage, le déplacement et l'animation du ou des sprites chats.

La première action qu'il effectue consiste à incrémenter la variable NIV (niveau de jeu) :

```
1910 NIV=NIV+1 : rem Passage au prochain niveau
```

La partie du sous-programme exécutée dépend de la valeur de la variable NIV. Au niveau 1, c'est-à-dire lors du lancement du programme, les coordonnées des sprites objets sont calculées par les formules suivantes :

```
1990 SX(4+J+I*2)=50+I*50
2000 SY(4+J+I*2)=50+J*50
```

Un seul chat est affiché sur l'écran. Il est affecté au sprite 2 :

```
2030 sprite 2,270,95,15
```

Les séquences de déplacement et d'animation sont les suivantes :

```
2040 move x 2,"270(2,-2,125)L" : move on
2050 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
```

Si la souris arrive à ramasser tous les objets affichés sur l'écran dans le temps imparti, le niveau 2 est atteint.

Les coordonnées des sprites objets sont directement stockées dans les tableaux SX et SY :

```
2120 SX(4)=100 : SX(5)=200 : SX(6)=50 : SX(7)=150 : SX(8)=250
2130 SX(9)=100 : SX(10)=200 : SX(11)=50 : SX(12)=150 : SX(13)=250
2140 SY(4)=30 : SY(5)=30 : SY(6)=60 : SY(7)=60 : SY(8)=60
2150 SY(9)=90 : SY(10)=90 : SY(11)=120 : SY(12)=120 :
SY(13)=120
```

Deux chats sont affichés sur l'écran. Ils sont affectés aux sprites 2 et 14 :

```
2160 sprite 2,270,95,15
```

Les séquences de déplacement et d'animation sont les suivantes :

```
2170 move x 2,"270(2,-2,125)L" : move on
2180 anim 2,"(15,10)(16,10)(17,10)(16,10)L" : anim on
2200 move x 14,"40(2,2,125)L" : move on
2210 anim 14,"(18,10)(19,10)(20,10)(19,10)L" : anim on
```

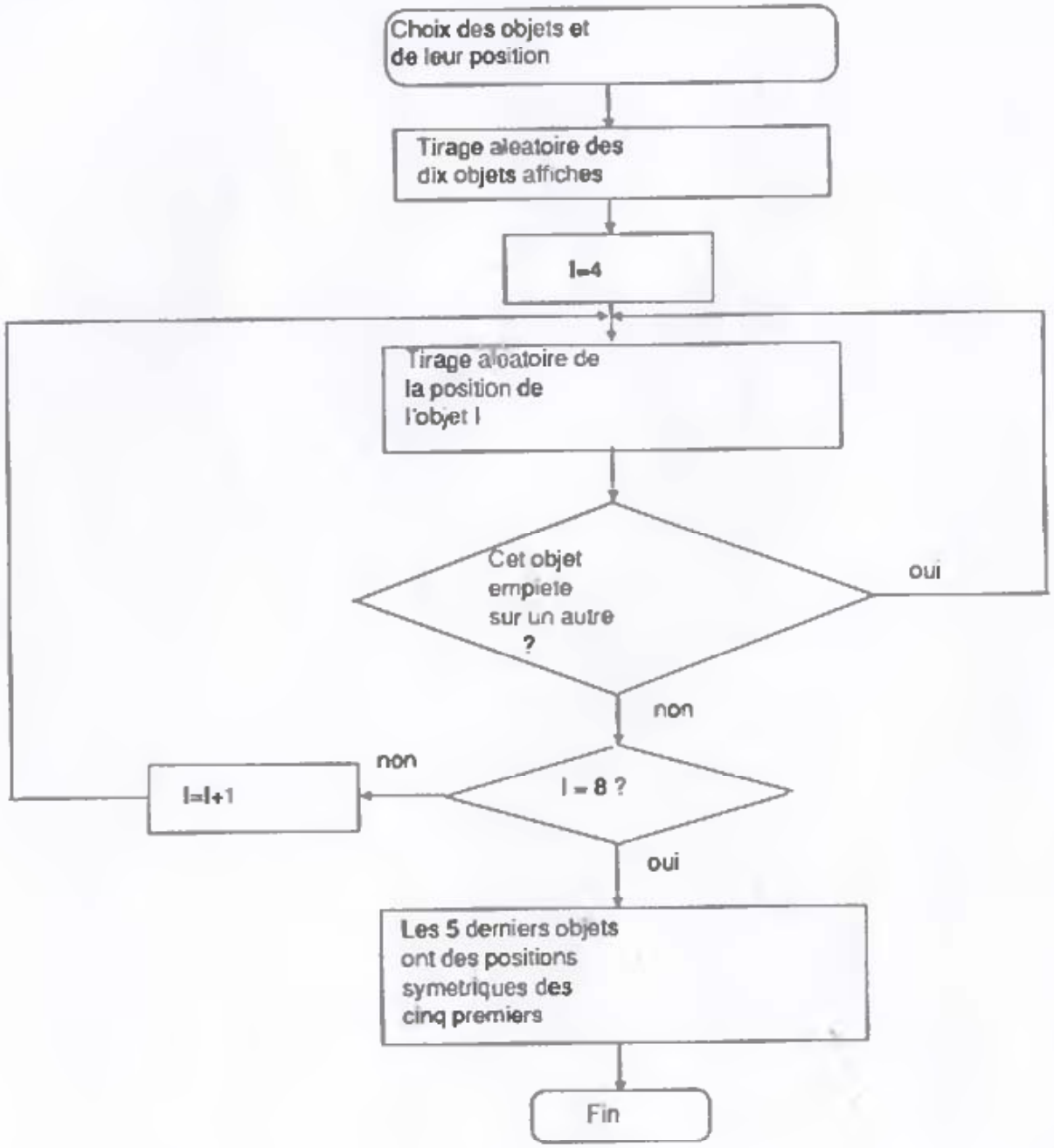
Si tous les objets affichés dans le tableau 2 sont ramassés par la souris dans le temps imparti, le niveau 3 est atteint.

Comme pour le niveau 2, les coordonnées des sprites objets sont directement stockées dans les tableaux SX et SY :

```
2280 SX(4)=50 : SX(5)=250 : SX(6)=100 : SX(7)=200 : SX(8)=150
2290 SX(9)=100 : SX(10)=200 : SX(11)=50 : SX(12)=250 : SX(13)=150
2300 SY(4)=50 : SY(5)=50 : SY(6)=70 : SY(7)=70 : SY(8)=90
2310 SY(9)=110 : SY(10)=110 : SY(11)=130 : SY(12)=130 : SY(13)=90
```

Comme pour le niveau 2, les sprites 2 et 14 sont utilisés pour déplacer et animer les deux chats sur l'écran. Nous n'y reviendrons pas.

Lorsque le niveau de jeu est supérieur à 3, la forme et la position des objets est choisie aléatoirement selon la logique de l'ordinogramme suivant (lignes 2420 à 2540) :



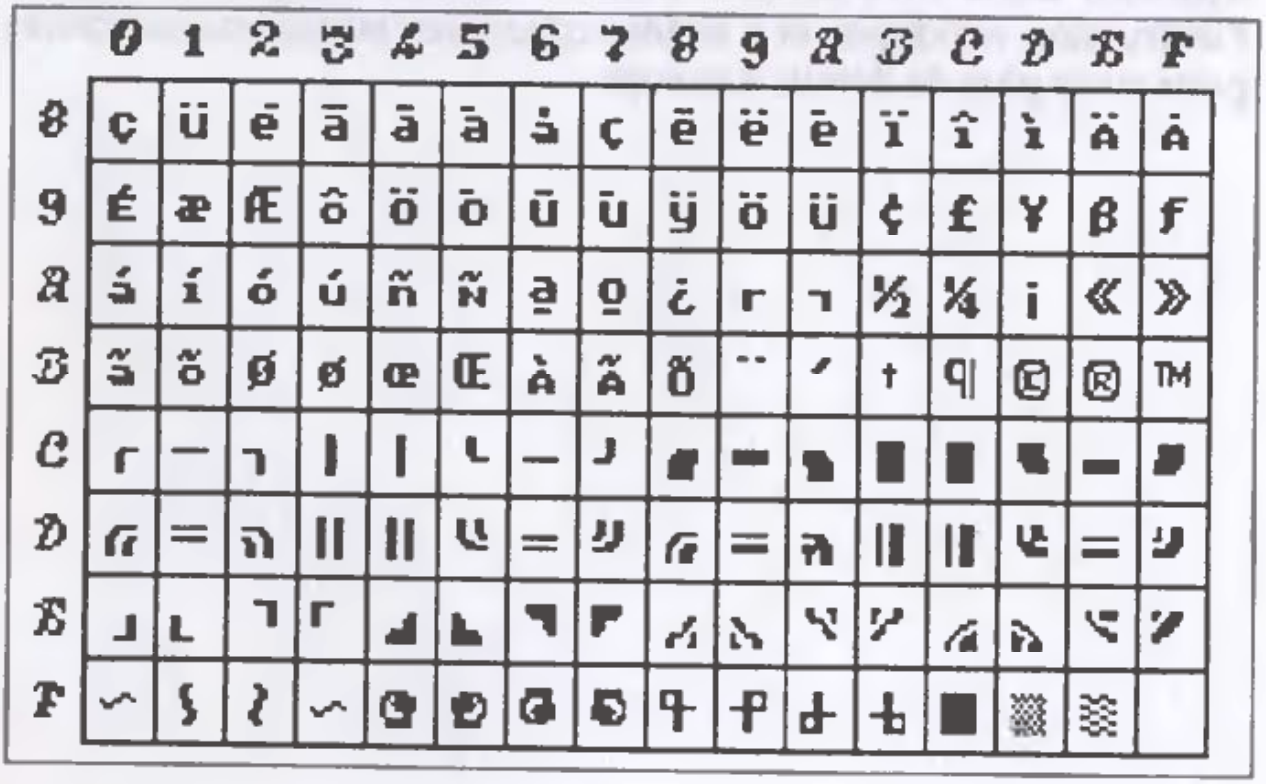
Comme pour les niveaux 2 et 3, les sprites 2 et 14 sont utilisés pour déplacer et animer les deux chats sur l'écran.

Annexe A

Les polices de caractères

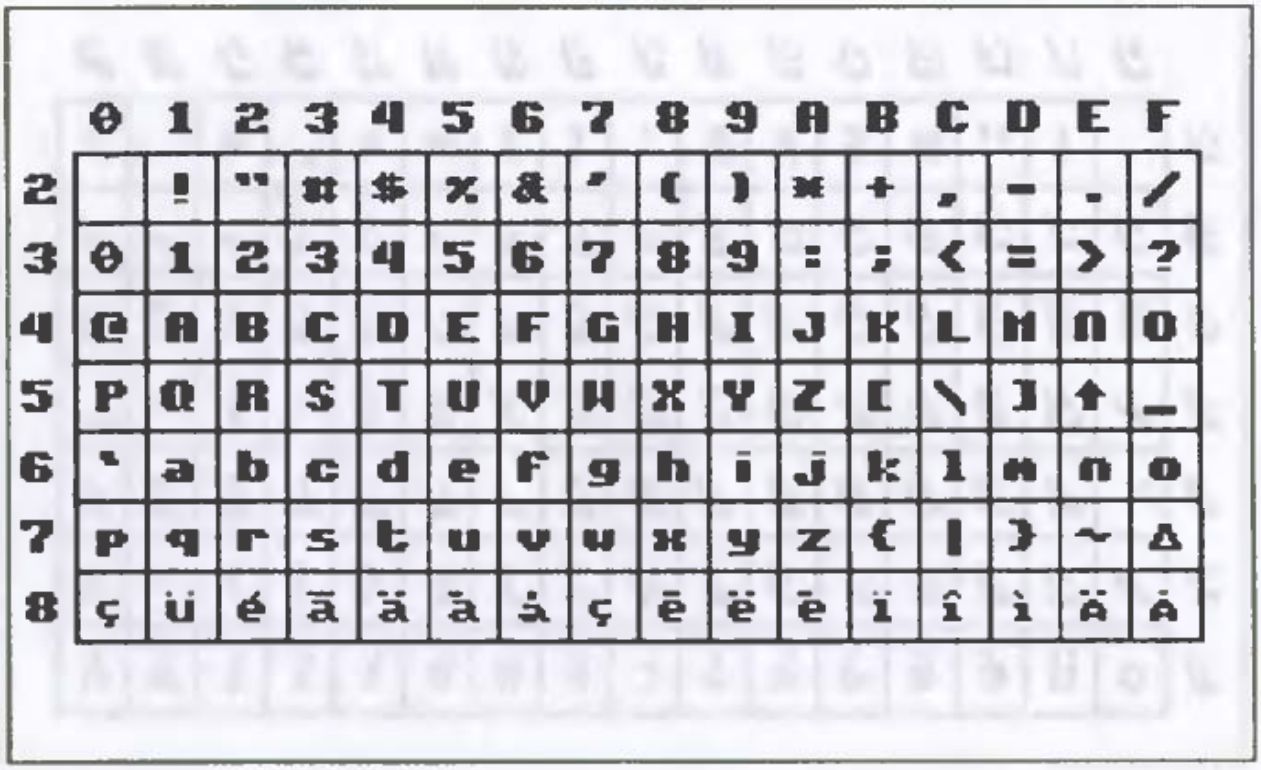
En plus de la police de caractères standard disponible dès l'entrée sous STOS, le disque Accessoires contient trois autres polices de nom FONT1.MBK, FONT2.MBK et FONT3.MBK. Ces polices peuvent être utilisées dans tout environnement fenêtre. Reportez-vous à l'instruction windopen et à la description de l'éditeur de caractères pour avoir plus de détails à ce sujet.

Les caractères disponibles dans FONT1.MBK sont les suivants :



Les caractères disponibles dans FONT2.MBK sont les suivants :

Les caractères disponibles dans FONT3.MBK sont les suivants :



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	Δ
8	ç	ü	é	ā	ä	à	å	ç	ē	ë	ē	ï	î	ì	Ä	Å

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	ç	ü	é	ā	ä	à	å	ç	ē	ë	ē	ï	î	ì	Ä	Å
9	É	æ	Æ	ô	ö	ò	ū	ù	ÿ	ö	ü	¢	£	¥	ß	f
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	ã	õ	ø	ø	œ	Œ	À	Ã	Õ	¨	´	†	¶	©	®	™
C	┌	─	┐	|	|	└	─	┘	▛	▬	▜	▌	▐	▙	▬	▟
D	╓	═	╖	‖	‖	╙	═	╜	╓	═	╖	‖	‖	╙	═	╜
E	┘	└	┐	┌	◢	◣	◥	◤	◌	◌	◌	◌	◌	◌	◌	◌
F	◌	◌	◌	◌	◉	◉	◉	◉	✝	✝	✝	✝	■	▒	▒	

Annexe B

Les mots-clés du basic STOS classés par genres

Mode direct

auto, change, cont, delete, english, fload, follow, francais, fsave, full, grab, kill, list, load, lower, merge, multi, renum, reset, run, save, search, stop, upper

Ordre général

asc, break, chr$, clear, data, date$, dec, dim, end, errl, errn, error, fkey, for to, gosub, goto, if then else, inc, inkey$, input, input$, line input, match, on error goto, on gosub, on goto, pop, print, print using, read, rem, repeat until, restore, resume, scancode, sort, system, time$, timer, wait, wait key, while wend

Mathématique

abs, acos, and, asin, atan, bchg, bclr, bin, bset, btst, cos, deg, def fn, exp, false, fix, fn, hcos, hex$, hsin, htan, int, ln, log, max, min, not, or, pi, rad, rol, ror, sgn, sin, sqr, swap, tan, true, xor

Chaînes

flip$, instr, left$, len, lower$, mid$, right$, space$, str$, string$, upper$, val

Ecran

appear, back, cls, def scroll, fade, get palette, load, logic, physic, reduce, reserve as datascreen, reserve as screen, screen copy, scroll, zoom,

Ecran texte

cdown, cleft, colour, cright, cup, crurs, home, inverse, locate, paper, pen, scrn, set curs, shade, under, xcurs, ycurs, writing

Ecran graphique

arc, bar, box, circle, draw, earc, ellipse, epie, ink, pie, plot, point, polygon, polyline, rbar, rbox, set line, set paint

Fenêtres

clw, menu, menu$, mnbar, mnselect, on goto on, on menu goto, qwindow, reserve as set, scroll, title, windel, windmove, windopen, window

Imprimante

hardcopy, ldir, listbank, llist, mprint, windcopy

Disque

close, dir, dir$, drive$, field, get, input, kill, mkdir, open in, open out, previous, print, put, rename, rmdir

Clavier

inkey$, key, key speed, scancode, wait key

Souris

change mouse, limit mouse, mouse key, x mouse, y mouse

Joystick

Sprites

anim, collide, detect, limit sprite, listbank, mode, move, move x, move y, movon, palette, reset zone, set zone, sprite, x sprite, y sprite, zone

Musique

bell, boom, click, envel, music, noise, play, pvoice, shoot, tempo, transpose, voice, volume

Accessoires

accload, accnew

Banques mémoire

bcopy, bgrab, erase, hexa, length, listbank, reserve, start

Annexe C

Vue d'ensemble sur le BASIC STOS

Lorsque vous aurez assimilé quelques uns des mots-clés du STOS, vous aurez certainement envie d'écrire vos propres jeux.

Si la syntaxe d'un des mots-clés du STOS ne vous est plus familière, vous pouvez :

- ✓ vous reporter à l'index qui vous renverra vers la partie du livre où il est expliqué en détail,
- ✓ examiner cette annexe qui donne la syntaxe et la fonction des mots-clés examinés dans le livre, par ordre alphabétique.

Si le nom d'une des instructions du STOS vous est sortie de la mémoire, vous pouvez également consulter l'annexe B qui liste les divers mots-clés du STOS, classés par types.

abs(<y>)

Valeur absolue de <y>.

accload <nom>

Chargement d'un accessoire en mémoire.

accnew

Effacement des accessoires en mémoire.

acos(<x>)

Arc cosinus de <x>.

and

ET logique.

anim <No>,<Ch>

Définition de la chaîne d'animation d'un sprite.

Appear <x>(,<y>)

Apparition progressive d'une image sur l'écran.

arc <X>,<Y>,<>,<Dep>,<Fin>

Tracé d'un arc de cercle de centre <X>,<Y>, de rayon <> entre les angles <Dep> et <Fin>.

asc(<ch>)

Valeur ASCII d'un caractère.

asin(<x>)

Arc sinus de <x>.

atan(<x>)

Arc tangente de <x>.

auto (<NL>,(<Esp>))

Numérotation automatique des lignes d'un programme.

back

Ecran de décor.

bar <x1>,<y1> to <x2>,<y2>

Tracé d'un rectangle plein.

bchg <bit>,<var>

Complémentation du bit <bit> dans la variable <var>.

bclr <bit>,<var>

Mise à zéro du bit <bit> dans la variable <var>.

bcopy <source> to <dest>

Copie d'une banque mémoire dans une autre.

bell

Son de cloche.

bgrab <prog>(,<b>)

Récupération des banques mémoires d'un programme ou d'un accessoire.

boom

Explosion.

bset <bit>,<var>

Mise à un du bit <bit> dans la variable <var>.

btst(<bit>,<var>)

Test de la valeur du bit <bit> dans la variable <var>.

bin$(<var>(,<prec>))

Conversion binaire d'un nombre.

box <x1>,<y1> to <x2>,<y2>

Tracé d'un rectangle vide.

break {on | off}

Validation/Dévalidation du Ctrl C.

cdown

Déplacement du curseur d'une ligne vers le bas.

change <ch1> to <ch2> (<début>-<fin>)

Recherche et remplacement d'une chaîne par une autre dans une partie ou la totalité du programme.

change mouse <N>

Modification de l'apparence du curseur de la souris.

chr$(<asc>)

Caractère de code ASCII <asc>.

circle <X>,<Y>,<R>

Tracé d'un cercle plein de centre <X>,<Y> et de rayon <R>.

clear

Initialisation des variables et banques d'un programme et du pointeur de données.

cleft

Déplacement du curseur d'une ligne vers la gauche.

click {on | off}

Validation/dévalidation du cliquetis du clavier.

close #<No>

Fermeture d'un fichier séquentiel ou à accès direct.

cls (<écran>)

cls <écran>,<couleur>

cls <écran>,<couleur>,<x1>,<y1> to <x2>,<y2>

Effacement partiel ou total d'un des écrans en mémoire.

collide(<No>,<Lar>,<Hau>)

Test de collision entre deux ou plusieurs sprites.

colour(<C>)

colour <C>,<$RVB>

Lecture ou définition des composantes RVB d'une couleur.

cont

Reprise de l'exécution du programme.

cos(<y>)

Cosinus de l'angle <y>.

cright

Déplacement du curseur d'une ligne vers la droite.

cup

Déplacement du curseur d'une ligne vers le haut.

curs {on | off}

Activation/désactivation du curseur.

data <d1>{,<d2>...{,<dN>}...}

Déclaration de données.

date$

Date système.

dec <var>

Décrémentation de la variable <var>.

def fn <nom>{(<var>)}=<expression>

Definition d'une fonction utilisateur.

def scroll <No>,<x1>,<y1> to <x2>,<y2>,<dirx>,<diry>

Définition d'une zone de scrolling.

deg{<y>}

Conversion d'un angle en degrés.

delete <premier>-<dernier>

Effacement des lignes comprises entre <premier> et <dernier> inclus.

detect{<No>}

Couleur du point situé sous le point chaud du sprite <No>.

dim <nom>(<d1>{,<d2>{...{<dN>}...}})

Dimensionnement d'un tableau.

dir <répertoire>

Affichage du contenu d'un répertoire.

dir$="<répertoire>"

Modification du répertoire courant.

draw <x1>,<y1> to <x2>,<y2>

draw to <x2>,<y2>

Tracé de lignes droites.

drive$="<Lect>"

Définition du lecteur courant.

earc <X>,<Y>,<R1>,<R2>,<Dep>,<Fin>

Tracé d'un arc d'ellipse de centre <X>,<Y>, de rayons <R1> et <R2>, entre les angles <Dep> et <Fin>.

ellipse <X>,<Y>,<R1>,<R2>

Tracé d'une ellipse pleine de centre <X>,<Y> et de rayons <R1> et <R2>.

end

Fin d'un programme.

english

Expression des messages système en Anglais.

envel <env>,<vitesse>

Choix d'une des enveloppes prédéfinies.

eple <X>,<Y>,<R1>,<R2>,<Deb>,<Fin>

Tracé d'une portion de camembert elliptique. Le centre du camembert est en <X>,<Y>. Les rayons de l'ellipse sont <R1> et <R2>. Le camembert est tracé entre les angles <Deb> et <Fin>.

erase <N>

Effacement de la banque <N>.

erl

Ligne où s'est produite la dernière erreur.

ern

Dernier numéro d'erreur.

error <n>

Génération volontaire d'une erreur.

exp(<nb>)

Exponentielle de <nb>.

fade <vitesse>

Disparition progressive d'une image.

false

Opérateur logique FALSE (0).

field #<No>,<l1> as <v1>(...(<lN> as <vN>)...)

Définition de la structure d'un fichier à accès direct.

fix(<N>)

Nombre de chiffres après la virgule des nombres réels.

fkey

Attente de l'appui sur une touche de fonction.

flip$(<ch>)

Inversion des caractères de la chaîne <ch>.

fload <chemin>

Charge un programme à travers un sélecteur de fichiers.

fn((<var>))

Accès à une fonction utilisateur.

follow<début>-<fin>

follow <v1>(,...(,<vN>)...)

follow <v1>(,...(,<vN>)...),<début>-<fin>

Exécution pas à pas du programme avec suivi de la valeur d'une ou de plusieurs variables.

for <var>=<début> to <fin> step <pas>

Exécution répétitive d'un bloc d'instructions.

francais

Expression des messages système en Français.

fsave <chemin>

Sauvegarde d'un programme à travers un sélecteur de fichiers.

full

Annulation de l'affichage multi-programme.

get #<No>,<enreg>

Lecture d'un enregistrement dans un fichier à accès direct.

get palette <N>

Initialisation de la palette courante à partir des couleurs de l'écran stocké dans la banque <N>.

gosub <ligne>

Débranchement du programme vers un sous-programme en <ligne>.

goto <ligne>

Débranchement du programme en <ligne>.

grab<N>(,<début>-<fin>)

Intégration partielle ou totale d'un programme dans le programme courant.

hardcopy

Copie graphique de l'écran sur l'imprimante.

hcos(<y>)

Cosinus hyperbolique de l'angle <y>.

hex$(<var>(,<prec>))

Conversion héxadécimale d'un nombre.

hexa {on | off}

Affichage hexadécimal/décimal des données concernant les banques mémoire.

home

Curseur en haut et à gauche de l'écran.

hsin(<y>)

Sinus hyperbolique de l'angle <y>.

htan(<y>)

Tangente hyperbolique de l'angle <y>.

if <condition> then <bloc1> else <bloc2>

Test d'une condition logique et exécution d'un bloc d'instructions ou d'un autre en fonction du résultat du test.

inc <var>

Incrémentation de la variable <var>.

ink <couleur>

Couleur du crayon pour les instructions graphiques.

inkey$

Test de la frappe d'une touche au clavier.

input (<texte>;)<v1>([,;]<v2>(...([,;]<vN>)...))

Lecture d'une ou de plusieurs données au clavier.

input #<No>,<var>

Lecture de données dans un fichier séquentiel.

input$(<N>)

Lecture de <N> caractères au clavier sans écho.

instr(<ch>,<sch>)

Position de <sch> dans <ch>.

int(<nr>)

Valeur entière de <nr>.

inverse

Affichage en inverse vidéo.

jdown

-1 si le joystick est actionné vers le bas, 0 sinon.

jfire

-1 si le bouton jeu du joystick est actionné, 0 sinon.

left

-1 si le joystick est actionné à gauche, 0 sinon.

joy

Etat du joystick.

jright

-1 si le joystick est actionné à droite, 0inon.

jup

-1 si le joystick est actionné vers le haut, 0 sinon.

key(<No>)=<ch>

Affectation d'une chaîne à une touche de fonction.

key speed <v>,<d>

Définition de la vitesse de répétition des touches et du délai avant répétition.

kill <fichier>.<ext>

Effacement d'un fichier.

ldir <répertoire>

Impression du contenu d'un répertoire.

left$(<ch>,<len>)

Extraction des <len> caractères à gauche dans <ch>.

len(<ch>)

Longueur de la chaîne <ch>.

length(<b>)

Longueur d'une banque mémoire.

limit mouse <x1>,<y1> to <x2>,<y2>

Limitation des déplacements de la souris à une zone réctangulaire.

limit sprite <x1>,<y1> to <x2>,<y2>

Limitation de la zone d'affichage des sprites.

line input(<texte>;)<v1>([,;]<v2>(...([,;]<vN>)...))

Lecture d'une ou de plusieurs chaînes au clavier comportant des caractères séparateurs.

list(<début>)(-)(<fin>)

Listage partiel ou total du programme en mémoire.

list bank

Affichage des banques utilisées par le programme courant.

llist

Impression du listing du programme courant.

ln(<nb>)

Logarithme népérien de <nb>.

load <fichier>(.bas|.acb|.pi1|.pi2|.pi3|.neo|.mbk)

Chargement d'un programme, d'un accessoire, d'une image ou d'une banque mémoire.

locate <x>,<y>

Positionnement du curseur sur l'écran.

log(nb>)

Logarithme décimal de <nb>.

logic

Ecran logique.

lower

Mots-clés STOS en minuscules, variables en majuscules.

lower$(<ch>)

Version minuscule de <ch>.

lprint<v1>({,;}<v2>(...({,;}<vN>)...))

Impression de la valeur d'une ou de plusieurs variables.

match (<nom>(0),<val>)

Recherche de la valeur la plus proche de <val> dans le tableau <nom>.

max(<A>,<B>)

La plus grande des deux valeurs passées.

menu$(<N>)=<Ch>(,<papier>,<crayon>)

menu$(<N>,<M>)=<Ch>(,<papier>,<crayon>)

Définition des options d'un menu.

menu freeze

Dévalidation temporaire d'un menu.

menu off

Effacement d'un menu.

menu on

Validation d'un menu.

merge <fichier>

Intégration partielle ou totale d'un programme dans le programme courant.

mid$(<ch>,<deb>,<len>)

Extraction des <len> caractères à partir du caractère <deb> dans <ch>.

min(<A>,<B>)

La plus petite des deux valeurs passées.

mkdir <répertoire>

Création d'un sous-répertoire du répertoire courant.

mnbar

Option principale de menu sélectionnée.

mnselect

Option secondaire de menu sélectionnée.

mode <N>

Définition du mode d'affichage courant.

mouse key

Etat des touches de la souris.

move {on | off} {<No>}

Validation/Dévalidation du déplacement d'un ou de tous les sprites.

move freeze {<No>}

Suspension du déplacement d'un ou de tous les sprites.

move x <No>,"<Ch>"

Déplacement d'un sprite selon un axe horizontal.

move y <No>,"<Ch>"

Déplacement d'un sprite selon un axe vertical.

movon(<No>)

Valeur non nulle si le sprite <No> est en cours de déplacement.

multi {2|3|4}

Affichage multi-programme.

music {on | off}

Activation/désactivation d'un morceau.

music <N>

Activation du morceau <N>.

music freeze

Désactivation temporaire d'un morceau.

noise (<voie>,)<hauteur>

Production d'un souffle pur.

not

Inversion d'une condition logique.

on <var> gosub <ligne1>(,<ligne2>(,...(,<ligneN>)...))

Débranchement calculé vers un sous-programme en fonction de la valeur de <var>.

on <var> goto <ligne1>(,<ligne2>(,...(,<ligneN>)...))

Débranchement calculé en fonction de la valeur de <var>.

on error goto <ligne>

Débranchement en <ligne> lorsqu'une erreur se produit.

on goto on

Validation du "on menu goto".

on menu goto <l1>(,<l2>...(,<lN>)...)

Débranchement calculé en fonction du menu sélectionné.

open in #<No>,"<Nom>"

open out #<No>,"<Nom>"

Ouverture en lecture ou en écriture d'un fichier séquentiel.

open #<No>,"R","<Nom>"

Ouverture en lecture/écriture d'un fichier à accès direct.

or

OU logique.

Palette <c0>,<c1>,<c2>,<c3>,<c4>,<c5>,<c6>,
<c7>,<c8>,<c9>,<c10>,<c11>,<c12>,<c13>,<c14>,<c15>

Définition des couleurs de la palette courante.

paper <C>

Définition de la couleur du papier.

pen <C>

Définition de la couleur du stylo.

physic

Ecran physique.

pi

Constante transcendentale PI.

ple <X>,<Y>,<R>

,<Dep>,<Fin>

Tracé d'une portion de camembert pleine. Le centre du camembert est en <X>,<Y>. Son rayon est <R>. La portion du camembert est comprise entre les angles <Dep> et <Fin>.

play (<voie>,)<hauteur>,<durée>

Production d'un son pur.

plot <X>,<Y>(,<C>)

Affichage d'un point de couleur <C> en <X>,<Y>.

point(<X>,<Y>)

Couleur du point situé en <X>,<Y>.

polygon <x1>,<y1> to <x2>,<y2> to ... to <xN>,<yN>

Tracé d'un polygone plein.

polyline <x1>,<y1> to <x2>,<y2> to ... to <xN>,<yN>

Tracé d'un polygone vide.

pop

Effacement de l'information de retour des instructions gosub et on gosub.

previous

Passage au répertoire père du répertoire courant.

print <v1>([,;]<v2>(...([,;]<vN>)...))

Affichage de données sur l'écran.

print #<No>,<data>

Stockage de données dans un fichier séquentiel.

print using "<ch>";<var>

Affichage formaté de données sur l'écran .

put #<No>,<enreg>

Stockage d'un enregistrement dans un fichier à accès direct.

qwindow <N>

Rend la fenêtre <N> active.

rad(<t>)

Conversion d'un angle en radians.

rbar <x1>,<y1> to <x2>,<y2>

Tracé d'un rectangle plein aux coins arrondis.

rbox <x1>,<y1> to <x2>,<y2>

Tracé d'un rectangle vide aux coins arrondis.

renum (<NL>(,<Esp>(,<Deb>-<Fin>)))

Renumérotation partielle ou totale des lignes d'un programme.

read <Var1>(...,(<varN>)...)

Lecture de données définies en data.

reduce <écran1> to (<écran2>,)<x1>,<y1>,<x2>,<y2>

Réduction d'une portion d'écran.

rem <remarque>

Remarque.

repeat <Bloc> until <condition>

Exécution répétitive d'un bloc d'instructions.

rename <ancien> to <nouveau>

Modification du nom d'un fichier.

reserve as datascreen

Réservation d'un écran permanent.

reserve as screen

Réservation d'un écran temporaire.

reserve as set <No>,<Taille>

Réservation d'une banque de caractères.

reset

Initialisation de l'éditeur du STOS.

reset zone ((<NZ>))

Effacement d'une zone définie avec set zone.

restore (<ligne>)

Relecture de données définies en data.

resume (next | <ligne>)

Retour d'un programme de gestion des erreurs.

right$(<ch>,<len>)

Extraction des <len> caractères à droite dans <ch>.

rmdir <répertoire>

Effacement d'un répertoire.

rol.<f> <N>,<var>

Rotation de <N> bits vers la gauche de la variable <var>.

ror.<f> <N>,<var>

Rotation de <N> bits vers la droite de la variable <var>.

run (<NL> | <Fichier>)

Exécution d'un programme

save <fichier>(.bas|.acb|.pi1|.pi2|.pi3|.neo|.mbk)

Sauvegarde d'un programme, d'un accessoire, d'une image ou d'une banque mémoire.

scancode

Code clavier d'une touche.

screen copy <écran1>,(<x1>,<y1>,<x2>,<y2>) to <écran2>(,<x3>,<y3>)

Copie partielle ou totale d'un écran dans un autre écran.

scrn(<x>,<y>)

Caractère affiché sur l'écran en <x>,<y>.

scroll <zone>

Scrolling d'une zone d'écran.

scroll (on|off)

scrolling validé/dévalidé dans la fenêtre courante.

scroll (up|down)

Scrolling d'une ligne vers le haut/bas dans la fenêtre courante.

search (<chaîne>)

Recherche d'une chaîne de caractères dans le programme.

set curs <H>,<B>

Définition de la taille du curseur.

set line <masque>,<épaisseur>,<deb>,<fin>

Définition du tramage des lignes et courbes tracées avec les instructions arc, earc, box, rbox et polyline.

set paint <type>,<motif>,<bord>

Définition du motif de remplissage des instructions circle, ellipse, bar, rbar, pie, epic et polygon.

set zone <NZ>,<x1>,<y1> to <x2>,<y2>

Définition d'une zone rectangulaire relative aux sprites.

sgn(<var>)

Signe de <var>.

shade

Affichage des caractères en relief.

shoot

Coup de feu.

sin(<y>)

Sinus de l'angle <y>.

sort <nom>(0)

Classement des éléments du tableau <nom> dans l'ordre croissant.

space$(<N>)

Chaîne de <N> espaces.

sprite <No>,<x>,<y>,<Sp>

Affectation d'un numéro à un sprite et affichage sur l'écran.

sqr(<nb>)

Racine carrée de <nb>.

start(<b>)

Adresse d'implantation d'une banque mémoire.

stop

Arrêt de l'exécution du programme.

str$(<nb>)

Conversion chaîne du nombre <nb>.

string$(<car>,<N>)

Chaîne de <N> caractères <car>.

swap <A>,<B>

Echange des valeurs <A> et <B>.

system

Retour au TOS.

tan(<y>)

Tangente de l'angle <y>.

tempo <vitesse>

Définition de la vitesse d'exécution du morceau courant.

time$

Heure système.

timer

Compteur système au 1/50 secondes.

transpose <val>

Transposition du morceau en cours d'exécution.

true

Opérateur logique TRUE (-1).

under

Affichage des caractères soulignés.

upper

Mots-clés STOS en majuscules, variables en minuscules.

upper$(<ch>)

Version majuscule de <ch>.

val(<ch>)

Conversion numérique de la chaîne <ch>.

volume (<voies>,)<vol>

Définition du volume sonore d'une ou de plusieurs voies.

wait <délai>

Attente d'un délai paramétrable.

wait key

Attente de la frappe d'une touche.

while <condition> <Bloc> wend

Exécution répétitive d'un bloc d'instructions.

windcopy

Copie texte de la fenêtre courante.

windel <n>

Effacement de la fenêtre <n>.

windmove <x>,<y>

Déplacement de la fenêtre courante en <x>,<y>.

Windopen <n>,<x>,<y>,<larg>,<haut>(,<bord>(,<police>))

Définition et ouverture d'une fenêtre.

window <n>

Rend la fenêtre <n> active.

writing <Mode>

Mode d'affichage des caractères sur l'écran.

x mouse

Abscisse de la souris sur l'écran.

xcurs

Abscisse du curseur sur l'écran.

xor

ou exclusif logique.

x sprite(<No>)

Abscisse du point chaud du sprite <No>.

ycurs

Ordonnée du curseur sur l'écran.

y mouse

Ordonnée de la souris sur l'écran.

y sprite(<No>)

Ordonnée du point chaud du sprite <No>.

zone(<No>)

Numéro de l'éventuel sprite qui a pénétré dans une zone prédéfinie.

zoom <écran1>,<x1>,<y1>,<x2>,
<y2> to (<écran2>,)<x3>,<y3>,<x4>,<y4>

Agrandissement d'une portion d'écran.

Annexe D

Les erreurs système

N° de l'erreur	Message d'erreur	Cause de l'erreur
1	Mauvais format de fichier	Vous tentez de charger un fichier qui a un format inconnu du STOS
2	Mémoire pleine	Le STOS n'a plus assez de mémoire pour fonctionner correctement. Supprimez un ou plusieurs accessoires et/ou programmes pour continuer
3	Cette ligne n'existe pas	Vous essayez de supprimer une ligne qui n'existe pas
4	Cette ligne existe déjà	L'instruction auto a généré un numéro de ligne déjà existant dans le programme
5	La recherche a échoué	L'instruction search n'a pas trouvé la chaîne spécifiée
6	Ligne trop longue	Vous tentez d'entrer une ligne de plus de 700 caractères
7	Impossible de continuer	Vous tentez d'exécuter l'instruction cont après un stop alors que vous avez modifié le programme.
8	Mémoire pleine	Le STOS n'a plus assez de mémoire pour fonctionner correctement. Supprimez un ou plusieurs accessoires et/ou programmes pour continuer

N° de l'erreur	Message d'erreur	Cause de l'erreur
9	Follow trop long	Vous essayez de suivre un nombre de variables trop important avec l'instruction follow.
10	L'imprimante n'est pas prête	Vous tentez d'accéder à l'imprimante alors qu'elle n'est pas connectée, pas prête ou pas sous tension.
11	Renumérotation impossible	Les paramètres entrés dans une instruction renum produisent des conflits. L'instruction n'est pas exécutée
12	Erreur de syntaxe	Vous tentez d'exécuter une instruction dont la syntaxe est incorrecte
13	Appel illégal de fonction	Vous tentez d'appeler une fonction en lui transmettant un ou plusieurs paramètres incorrects.
14	Instruction interdite en mode direct	L'instruction que vous tentez d'exécuter en mode direct ne peut être utilisée qu'en mode programme.
15	Instruction interdite en mode programme	L'instruction que vous tentez d'exécuter en mode programme ne peut être utilisée qu'en mode direct.
16	Erreur d'entrée/sortie	Une opération d'entrée/sortie a produit une erreur fatale au programme.
17	Stop	Les touches Ctrl et C ont été pressées simultanément.
18	Tableau non déclaré	Vous tentez d'accéder à un élément de tableau alors qu'il n'a pas été dimentionné.
19	Types de variables incompatibles	Les affectations ne peuvent se faire qu'entre variables et données de même type. Vous essayez d'affecter une chaîne à une variable numérique ou inversement.
21	Dépassement de capacité	Un calcul a donné un résultat trop grand.
22	For sans next	Le STOS n'arrive pas à localiser l'instruction next correspondant à un for qu'il tente d'exécuter.
23	Next sans for	Le STOS a rencontré une instruction next qui ne se rattache à aucune instruction for précédemment exécutée.
24	While sans wend	Le STOS n'arrive pas à localiser l'instruction wend correspondant à un repeat qu'il tente d'exécuter.

N° de l'erreur	Message d'erreur	Cause de l'erreur
25	Wend sans while	Le STOS a rencontré une instruction wend qui ne se rattache à aucune instruction while précédemment exécutée.
26	Repeat sans until	Le STOS n'arrive pas à localiser l'instruction until correspondant à un repeat qu'il tente d'exécuter.
27	Until sans repeat	Le STOS a rencontré une instruction until qui ne se rattache à aucune instruction repeat précédemment exécutée.
28	Tableau déjà défini	Vous tentez de redimentionner un tableau. Vérifiez que le programme ne boucle pas sur une ligne qui comporte une instruction dim.
29	Numéro de ligne non défini	Une instruction de débranchement (goto, gosub, restore) fait référence à un numéro de ligne inexistant.
30	Chaîne trop longue	Vous essayez de créer une chaîne de plus de 65000 caractères.
31	Erreur de bus	Erreur interne au STOS. Peut-être avez vous lu ou écrit une donnée à une adresse interdite ou incorrecte.
32	Erreur d'adresse	Vous essayez d'accéder à une adresse mémoire impaire à l'aide d'une instruction deek, doke, leek ou loke.
33	Pas de 'data' sur cette ligne	La ligne référencée dans une instruction restore ne contient pas de data
34	Plus de donnée	Vous tentez d'utiliser l'instruction read alors que toutes les données du programme ont été lues
35	Trop de gosub	Vous avez effectué un trop grand nombre de gosub et le STOS n'a plus assez de place pour stocker les adresses de retour.
36	Return sans gosub	Vous essayez d'exécuter une instruction return alors qu'aucun gosub n'a été précédemment exécuté.
37	Pop sans gosub	Utilisation illicite de l'instruction pop. Aucun sous-programme n'a été appelé. L'instruction pop ne peut donc pas être utilisée.

N° de l'erreur	Message d'erreur	Cause de l'erreur
38	Resume sans erreur	Vous essayez d'exécuter une instruction resume alors qu'aucune erreur n'a été détectée.
39	Fonction utilisateur non définie	Vous essayez d'appeler une fonction utilisateur alors qu'elle n'a pas été définie.
40	Mauvais appel de fonction utilisateur	Tentative d'appel d'une fonction utilisateur en lui passant un nombre de paramètres différent de celui qui apparaît dans sa définition.
41	Banque mémoire déjà réservée	Vous tentez de réserver une banque mémoire déjà réservée.
42	Banque mémoire non écran	Vous essayez d'accéder à une banque non écran avec une instruction dédiée aux écrans.
43	Mauvaise adresse d'écran	Vous essayez d'accéder à l'écran avec une adresse incorrecte
44	Banque mémoire non réservée	Vous tentez d'accéder à une banque mémoire non réservée.
45	Résolution non autorisée	Paramétrage incorrect de l'instruction mode. Le mode 2 (haute résolution) ne peut être utilisé que sur un moniteur noir et blanc. Les modes 0 et 1 (basse et moyenne résolution) ne peuvent être utilisés que sur un moniteur couleur.
46	Division par zéro	Vous tentez d'effectuer une division par zéro.
47	Opérande négatif	Un paramètre négatif est illégalement passé à une fonction.
48	Fichier introuvable	Vous tentez de lire ou d'écrire dans un fichier qui ne se trouve pas sur le lecteur accédé
49	Lecteur pas prêt	Vous tentez d'accéder à un lecteur qui ne contient aucune disquette.
50	Disquette protégée	Vous tentez d'accéder en écriture à une disquette protégée. Déprotégez ou changez de disquette.
51	Disquette pleine	Vous essayez de stocker des données sur un disque qui n'a plus assez de place libre.

N° de l'erreur	Message d'erreur	Cause de l'erreur
52	Erreur disquette	Erreur lors d'un accès disque. Cette erreur se produit lorsque le TOS rencontre une erreur disque sans pouvoir l'identifier.
53	Mauvais nom de fichier	Vous tentez d'accéder à un fichier dont le nom est illégal.
54	Mauvaise heure	Vous tentez d'initialiser la variable TIME$ avec des données incorrectes.
55	Mauvaise date	Vous tentez d'initialiser la variable DATE$ avec des données incorrectes.
56	Erreur de sprite	Mauvais paramétrage de l'instruction sprite.
57	Mauvais appel de MOVE	Le paramétrage d'une instruction move est incorrect.
58	Mauvais appel d'ANIM	La chaîne d'animation d'une instruction anim est incorrecte
59	Fichier non ouvert	Vous tentez de lire ou d'écrire des données dans un fichier qui n'est pas ouvert.
60	Mélange de types de fichiers	Vous tentez d'accéder à un fichier séquentiel avec des instructions réservées aux fichiers à accès direct.
61	Chaîne en entrée trop longue	Deux possibilités : 1) vous tentez d'affecter une chaîne trop longue à une variable dimentionnée,, vous tentez de lire une chaîne de plus de 500 caractères dans un fichier séquentiel ou sur un périphérique
62	Fichier déjà ouvert	Vous tentez d'ouvrir un fichier déjà ouvert.
63	Fichier déjà fermé	Vous tentez de fermer un fichier qui est déjà fermé.
64	Fin de fichier	Vous essayez de lire une donnée dans un fichier alors que la fin du fichier a été détectée.
65	Chaîne en entrée trop longue	Deux possibilités : vous tentez d'affecter une chaîne trop longue à une variable dimentionnée, vous tentez de lire une chaîne de plus de 500 caractères dans un fichier séquentiel ou sur un périphérique

N° de l'erreur	Message d'erreur	Cause de l'erreur
66	Champ trop long	Erreur typique d'un fichier à accès direct : nombre de champs supérieur à 16 ou un des champ a une dimension supérieure à 65535 octets.
67	Mauvais appel de FLASH	Vous tentez d'utiliser l'instruction flash avec un paramétrage incorrect.
68	Paramètre de fenêtre trop grand	Un des paramètres d'une instruction windopen est incorrect.
69	Fenêtre déjà ouverte	Vous tentez d'ouvrir une fenêtre déjà ouverte.
70	Fenêtre non ouverte	Vous faites appel à une fenêtre qui n'a pas été ouverte.
71	Fenêtre trop petite	Vous tentez d'ouvrir une fenêtre dont la taille est plus petite que la taille minimale autorisée (3x3).
72	Fenêtre trop grande	Vous tentez d'ouvrir une fenêtre dont la taille est supérieure à la taille maximale autorisée.
73	Jeu de caractères non défini	Vous faites référence à un jeu de caractères inexistant.
74	Buffer texte plein	Vous tentez d'ouvrir simultanément 11 fenêtres de la taille d'un écran en mode 1 ou 2.
75	Musique non définie	Vous tentez d'exécuter une musique non définie.
76	Appel d'une fenêtre système	Vous tentez d'utiliser une fenêtre système (0, 14, 15) comme une fenêtre normale.
77	Appel d'un jeu de caractères système	Vous tentez de remplacer un jeu de caractères système par un jeu de caractères personnel.
78	Jeu de caractères introuvable	Vous tentez d'accéder à un jeu de caractères qui n'a pas été défini.
79	Menu non défini	Vous tentez d'exécuter l'instruction menu on alors qu'aucun menu n'a été défini.
80	La banque 15 est déjà réservée	Vous tentez d'utiliser la banque 15 alors qu'elle est déjà réservée. Si c'est bien ce que vous voulez faire, effacez-là avant de la réserver.

N° de l'erreur	Message d'erreur	Cause de l'erreur
81	La banque 15 est réservée pour les menus	Vous tentez de réserver la banque 15 alors que le programme courant utilise des menus.
82	Instruction illégale	Un code instruction illégal a été détecté dans un programme écrit en assembleur.
83	Lecteur non connecté	Vous tentez d'accéder à un lecteur non connecté ou hors tension.
84	Extension non chargée	Vous faites référence à une extension non chargée en mémoire.
85	Indice trop grand	Vous essayez d'accéder à un élément de tableau dont l'indice ou un des indices est trop grand
86	Appel illégal de fonction	Vous tentez d'appeler une fonction en lui transmettant un ou plusieurs paramètres incorrects.
87	Appel illégal de fonction	Vous tentez d'appeler une fonction en lui transmettant un ou plusieurs paramètres incorrects.

Index

A

Abs 3-106
Accessoires 2-19, 8-189
Accload 8-190
Accnew 8-190
Acos 3-107
Affichage d'un sprite 6-166
Aide à la recherche d'erreur 3-86
And 3-109
Anim 6-171
Animation d'un sprite 6-171
Appear 4-144
Arcs de cercles 4-133
Asc 3-104
Asin 3-107
Atan 3-107
Auto 3-80

B

Back 4-138
Banques
➥ mémoire 8-190
➥ permanentes 8-190

➥ temporaires 8-191
Bar 4-131
Bchg 3-110
Bclr 3-110
Bcopy 8-192
Bell 7-180
Bgrab 8-192
Bin$ 3-109
Boom 7-180
Box 4-131
Break off 3-105
Bset 3-110
Btst 3-111
Buffer clavier 5-161

C

Camemberts 4-133
Cdown 4-124
Cercles 4-133
Change 3-86
Change mouse 5-163
Changement de base 3-109
Chargement
➥ d'un écran 4-138
➥ d'un programme en mémoire 3-80
Chr$ 3-105
Circle 4-133
Clavier 3-99, 5-161
Clear 3-105
Cleft 4-124
Click off 7-180
Click on 7-180

Close 4-157
Cls 4-139
Code ASCII 5-161
Collide 6-175
Collision entre deux sprites 6-175
Colour 4-125
COMPACT.ACB 2-70, 8-189
Compacteur d'écran 2-70
Cont 3-88
Copie d'ECRAN 4-143
Copie et décompactage des disquettes 1-13
Couleur du décor sous le point chaud 6-175
Création
➥ d'un programme de jeu 9-193
➥ du décor 10-203
Cright 4-124
Cup 4-124
Curs off 4-124
Curs on 4-124

D

Data 3-101
Date et heure 3-96
Date$ 3-96
Débranchements 3-89
Dec 3-105
Def fn 3-112
Def scroll 4-140
Deg 3-107
Delete 3-82
Déplacement d'un sprite 6-168
Detect 6-175

Dim 3-103
Dir 4-152
Dir$ 4-153
Disquette
➥ "Accessoires" 1-15
➥ "Jeux" 1-14
➥ "Langage" 1-14
Draw 4-131
Drive$ 4-155

E

Earc 4-134
Ecran 3-98, 4-123, 4-138
Editeur
➥ d'icônes 2-49
➥ de caractères 2-47
➥ de musique 2-49, 7-183, 11-205
➥ de sprites 2-19, 10-201
➥ du STOS 1-15
Edition simultanée de plusieurs programmes 3-84
Effacement
➥ d'un écran 4-139
➥ d'un programme 3-82
➥ d'un programme sur disque 3-80
Effets
➥ sonores 7-179
➥ spéciaux 4-144
Ellipse 4-134
End 3-106
English 3-88
Entrée/sortie 3-98
Envel 7-182

Enveloppe 2-58, 7-182
Erreurs système D-263, D-265
Error 3-93
Exécution d'un programme 3-76
Exp 3-113

F

Fade 4-144
False 3-108
Fenêtres 4-145
Fichiers
➥ à accès direct 4-158
➥ à accès séquentiel 4-156
Field 4-158
Fix 3-114
Fkey 3-100
Flip$ 3-118
Fload 3-80
Fn 3-112
Follow 3-87
Fonctions
➥ trigonométriques et hyperboliques 3-107
➥ utilisateur 3-112
FONT1.MBK 2-48, A-228
FONT2.MBK 2-48, A-229
FONT3.MBK 2-48, A-230
FONTS.ACB 2-47, 8-189
For 3-91
Formater une disquette 1-13
Francais 3-88
Fsave 3-78
Full 3-85

G

Gestion
➥ des erreurs 3-93
➥ des fichiers 4-156
Get palette 4-139
Get# 4-159
Gosub 3-90
Goto 3-89
Grab 3-85

H

Hardcopy 4-151
Hcos 3-108
Help 3-84
Hex$ 3-109
Hexa off 8-191
Hexa on 8-191
Home 4-124
Hsin 3-108
Htan 3-108

I

ICON.ACB 2-49, 8-189
Imprimante 4-151
Inc 3-105
Ink 4-130

Inkey$ 3-100, 5-161
Input 3-99
Input # 4-157
Input$ 3-100
Instr 3-117
Instructions
➥ dédiées aux chaînes 3-115
➥ graphiques 4-130
➥ mathématique 3-106
➥ opérant sur des bits 3-110
➥ texte 4-124
➥ utilisées en mode direct 3-75
➥ utilisées en mode programmé 3-88
Int 3-114
Intégration de commandes sonores dans un programme 7-180
Inverse 4-127

J

Jdown 5-164
Jeux de caractères 4-145
Jfire 5-164
Jleft 5-164
Joy 5-164
Joystick 5-164
Jright 5-164
Jup 5-164

K

Key 5-162
Key speed 5-162
Kill 3-80, 4-154

L

Langue des messages système 3-88
Ldir 4-151
Lecteurs de disques 4-151
Lecture de données 3-101
Left$ 3-115
Len 3-117
Lignes 4-131
Limit mouse 5-163
Line input 3-100
List 3-83
List bank 8-191
Listage d'un programme 3-83
Listbank 4-151
Llist 4-151
Ln 3-113
Load 3-80
Log 3-113
Logic 4-138
Lower 3-83
Lower$ 3-116
Lprint 4-151

M

Match 3-104
Max 3-113
Menu freeze 4-151
Menu off 4-150
Menu on 4-149
Menu$ 4-148
Menus 4-148
Merge 3-85
Mid$ 3-115
Min 3-113
Mixage de deux programmes 3-85
Mkdir 4-154
Mnbar 4-151
Mnselect 4-150
Mode 6-167
Mots clés du basic STOS B-231
Mouse key 5-163
Move on 6-168
Move x 6-168
Move y 6-168
Multi 3-85
Music 7-184
Music freeze 7-185
Music off 7-185
Music on 7-185
MUSIC.ACB 2-50, 8-189
Musique 7-179

N

New 3-82
Noise 7-181
Not 3-108
Numérotation des lignes 3-80

O

On error goto 3-94
On gosub 3-90
On goto 3-89
On goto on 4-150
On menu goto 4-150
Open 4-158
Open in 4-156
Open out 4-156
Or 3-109

P

Palette de couleurs 4-139
Paper 4-125
Pen 4-125
Physic 4-138
Pi 3-106
Pixel 4-130
Play 7-181
Plot 4-130

Point 4-130
Polices de caractères A-227
Polygon 4-132
Polygones 4-131
Polyline 4-132
Previous 4-154
Print 3-98, 4-125
Print # 4-157
Print using 3-98
Put# 4-158

Q

Qwindow 4-146

R

Rad 3-107
Rbar 4-132
Rbox 4-131
Read 3-101
Rectangles 4-131
Reduce 4-142
Réduction 4-141
Rem 3-102
Remise à zéro de l'éditeur 3-83
Rename 4-155
Renum 3-81
Repeat until 3-92
Répertoires 4-152

Reserve as datascreen 4-138
Reserve as screen 4-138
Reserve as set 4-147
Reset 3-83, 3-85
Reset zone 6-174
Restore 3-101
Resume 3-94
Right$ 3-115
Rmdir 4-154
Rol 3-111
Ror 3-111
RUN 3-76

S

Sauvegarde du programme en mémoire 3-77
Save 3-77
SCANCODE 5-161
Screen copy 4-143
Scrn 4-126
Scroll 4-141
Scrolling d'écran 4-140
Search 3-86
Set curs 4-125
Set line 4-134
Set paint 4-135
Set zone 6-173
Sgn 3-106
Shade 4-128
Shoot 7-180
Sort 3-103
Souris 5-163
Space$ 3-117

Sprite 6-166
Sprite.acb 2-19, 8-189
Sprites 6-165
Sqr 3-113
Stop 3-88
Str$ 3-118
String$ 3-117
Swap 3-114

T

Tempo 7-186
Tests
➥ de collision 6-173
➥ et répétitions 3-91
Time$ 3-97
Timer 3-97
Title 4-146
Touche Help 3-84
Touche Undo 3-85
Tramage 4-134
Transpose 7-186
Trémolo 2-63
True 3-108
Type d'écriture dans l'éditeur 3-83

U

Under 4-128
Upper 3-83
Upper$ 3-98, 3-116

V

Val 3-118
Volume 7-180

W

Wait 3-97
Wait key 3-101, 5-162
While wend 3-92
Windcopy 4-151
Windel 4-146
Windmove 4-147
Windopen 4-145
Window 4-146
Writing 4-126

X

X mouse 5-163
Xcurs 4-124
Xor 3-109

Y

Y mouse 5-163
Ycurs 4-124

Z

Zone 6-174
Zoom 4-141

Dans la même collection...

Référence	Titre		Prix T.T.C.
...			
ML156	Bien débuter avec l'ATARI ST et STe		129.00
ML527	Bien débuter en GFA BASIC 2.0 et 3.0		129.00
ML561	Bien débuter Le Rédacteur		129.00
ML631	Boîte à Outils ST	disquette incluse	299.00
ML688	Dév. s/SUPERBASE PRO/PROIII	disquette incluse	299.00
ML172	Disquette et Disque Dur		179.00
ML272	Disquette et Disque Dur	disquette incluse	279.00
GL102S	Guide SOS GFA BASIC 2.0 à 3.0		99.00
ML530	Le Grand Livre de l'ATARI ST		199.00
ML530 OS	Pack Le Grand Livre de l'ATARI ST +Additif + freeware		199.00
ML556	Le livre de CALAMUS		199.00
ML573	Le livre de SUPERBASE (versions II, PRO, PRO III)		169.00
ML550	Le livre du Développeur sur ATARI ST		299.00
ML589	Le livre du Développeur sur ATARI ST (T2)		199.00
ML689	Le livre du Dév. sur ATARI ST (T2)	2 disquettes incluses	299.00
ML502	Le livre du Graphisme		199.00
ML602	Le livre du Graphisme	2 disquettes incluses	299.00
ML141	Le livre du Langage Machine		149.00
ML193	Le livre de l'Intelligence Artificielle		179.00
ML616	Le livre de 1ST WORD PLUS	disquette incluse	299.00
ML185	Le livre du GFA BASIC 2.0		199.00
ML285	Le livre du GFA BASIC 2.0	disquette incluse	299.00
ML571	Le livre du GFA BASIC 3.0		199.00
ML671	Le livre du GFA BASIC 3.0	disquette incluse	265.00
ML299	Trucs et Astuces en GFA 2.0	disquette incluse	269.00
ML651	Trucs et Astuces ATARI ST	disquette incluse	299.00
...			

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Mai 1990
N° d'impression: 5305

ATARI ST+STE

BIEN DEBUTER STOS

Michel MARTIN

Une horde de sprites foncent en direction de votre navette, une brusque poussée sur le joystick vous permet d'éviter de justesse la barrière d'astéroïdes... Les tirs se rapprochent et le temps diminue dangereusement...
Avec STOS, programmer un bon jeu d'arcade devient l'enfance de l'art. Quelques instructions suffisent pour définir les bases de votre nouveau "Shoot'em up" ou "Donjon Master"...
N'attendez plus pour découvrir ce fabuleux langage. Grâce à cet ouvrage, apprenez à construire de A à Z un véritable jeu d'arcade: bâtir un scénario, dessiner des décors et des sprites, enchaîner des animations ou différents tableaux, et enfin composer une bande musicale et des effets spéciaux.
Etape après étape, vous bénéficierez d'explications claires sur les principaux outils et instructions de l'environnement STOS, ainsi que de la liste complète des fonctions mots-clés et messages du langage.

Principaux sujets traités:

- Installation du système, décompactage des fichiers.
- Les différents outils de l'éditeur STOS et de l'éditeur d'icones...
- Les instructions de base du STOS: gestion des textes, graphiques, écrans...
- Contrôle des accessoires et périphériques: joystick, souris, imprimante...
- Gestion des sprites: affichage, déplacement, animation, collision...
- Création de musiques et d'effets spéciaux, intégration dans un programme.
- Eléments, conseils et astuces pour construire un programme de jeu complet.
- Listing complet du programme de jeu proposé dans l'ouvrage.

9 782868 993212

Réf. ML 717/Prix 129 F.
ISBN: 2-86899-321-4 / ISSN: 0980-1928

EDITIONS MICRO APPLICATION
58 RUE DU FAUBOURG POISSONNIERE
75010 PARIS TEL (1) 47 70 32 44