ABC INFORMATIQUE

N° 95

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE

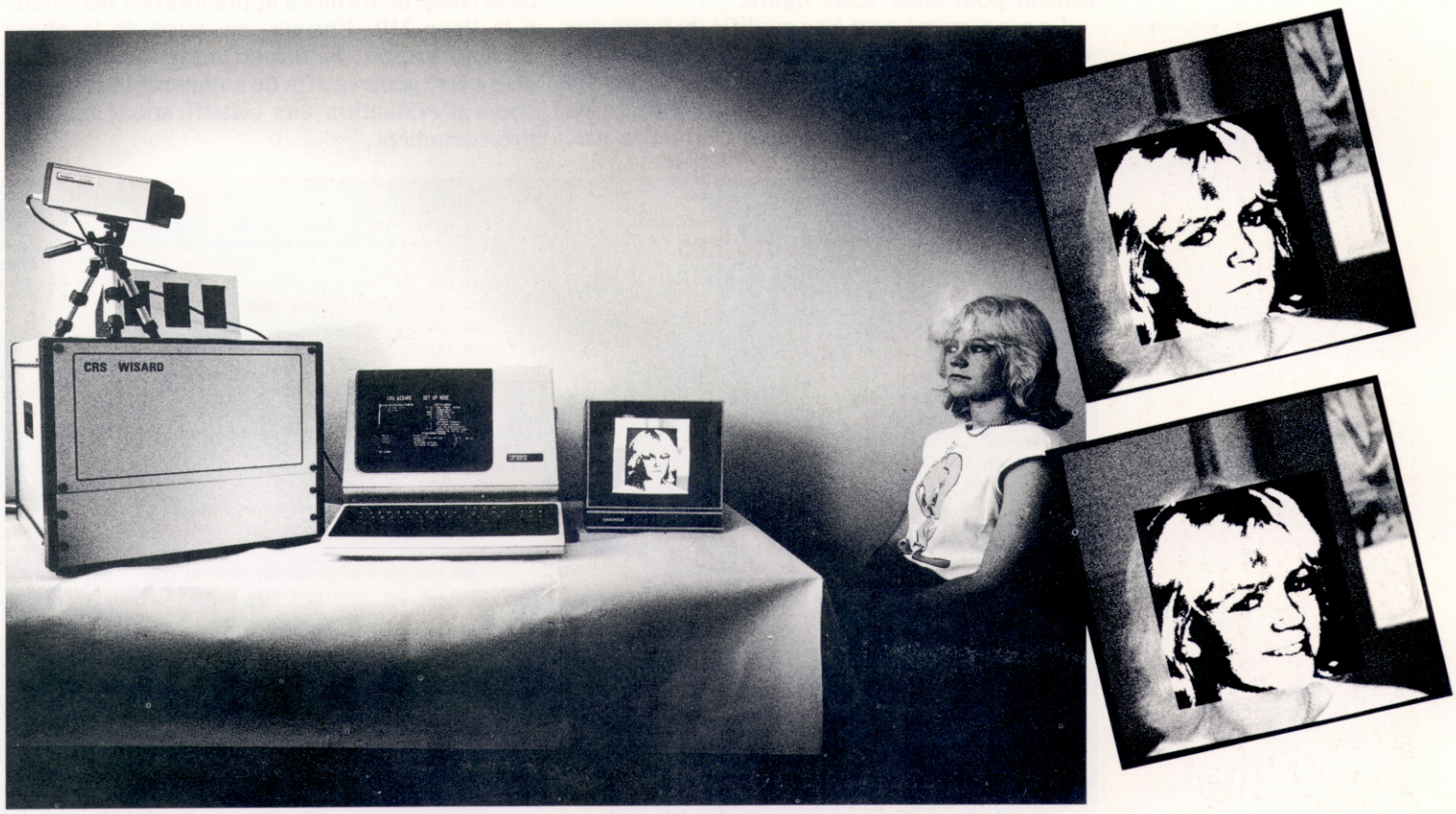
- Une souris pour les hommes
- Voir les objets
- Les crochets du Microdrive
- Les pinces du crabe

EDITIONS ATLAS

M 6062-95-12,50 F 90 FB - 3,80 FS - $ 1.95

Naissance des formes

Continuons notre étude des systèmes informatiques de vision en voyant comment il est possible d'identifier des caractéristiques inscrites à l'intérieur d'un ensemble de formes.



T'en fais une tête ! Le système Wisard d'Igor Aleksander, mis au point à l'Imperial College de Londres, fait usage d'une grille de 512 × 512 et d'une RAM de 1 mégaoctet pour pouvoir reconnaître les formes qui apparaissent sur un écran de télévision. Il est capable de faire des distinctions assez subtiles, par exemple entre le même visage selon qu'il est maussade ou souriant. (Cl. Tony Sleep/Computer Recognition System.)

Les systèmes de reconnaissance des formes, censés procéder « de bas en haut », cherchent à extraire d'une image donnée des traits caractéristiques qui seront à leur tour réduits à quelques éléments plus simples, beaucoup plus abstraits. Une méthode pour y parvenir consiste à faire usage d'opérateurs locaux. Chacun d'eux analyse une zone particulière, et multiplie les diverses intensités de gris qu'il est amené à rencontrer par des facteurs de pondération. Ces derniers sont par ailleurs déterminés de façon à donner des résultats élevés quand la caractéristique recherchée est découverte.

Le programme Wisard d'Igor Aleksander recourt au principe de « génération d'adresses mémoire » pour apprendre à se souvenir d'une forme et à la reconnaître plus tard.

Le grand avantage de Wisard est qu'il travaille en parallèle avec ses blocs de RAM; il est donc très rapide, et peut de surcroît traiter des images télévisées à haute définition. Notre programme BASIC n'est que séquentiel, et sera donc bien plus lent : mais il suffit à donner une bonne idée des principes fondamentaux mis en œuvre.

Les discriminants du système sont des sextuples : ils sont moins puissants que les octuples mais réclament moins d'espace mémoire. Nous aurons besoin de 20 480 bits (40 × 64 × 8) de RAM, soit 2 560 octets. Il y a en effet 40 sextuples, dont chacun peut prendre 64 états différents, ce qui lui permet d'adresser un bloc de RAM de 64 adresses. Chacune d'elles compte 8 bits, et notre programme pourra donc différencier huit classes de données différentes.

Il comporte deux phases successives — la première d'apprentissage, la seconde de reconnaissance. L'ordinateur se voit d'abord soumettre des exemples de chacune des classes de données, puis procède à des identifications par comparaison.

Supposons par exemple que pendant la première phase le sextuple n° 20 donne la réponse 110010 (soit 50) à chaque fois qu'une forme de classe 4 est présente. Le quatrième bit de l'adresse mémoire 50 du bloc de RAM n° 20 prendra dans ces conditions une valeur de 1. Lors de la phase ultérieure d'identification, si le discriminant donne de nouveau une réponse de 50, le quatrième bit de l'adresse correspondante sera

mis à 1, ce qui tendra à confirmer la présence de la forme n° 4.

Un ensemble d'octets, commençant en D, contient les données relatives à la forme considérée ; le tableau tout entier est remis à zéro à chaque nouvelle exécution. C'est là le rôle du sous-programme commençant à la ligne 1000. Celui de la ligne 2000 vous permettra de « peindre » des images très simples (16 × 16) sur l'écran, en vous servant des touches H, B, G et D pour déplacer le curseur, et de * et de la barre d'espacement pour créer votre figure.

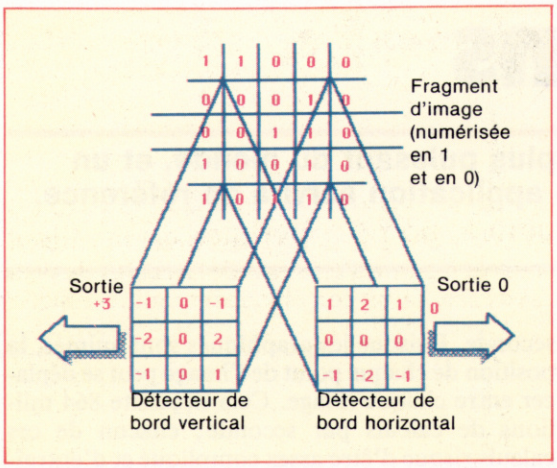
Le programme peut être modifié de façon que vous puissiez stocker le tableau D sur cassette ou sur disquette. Tel qu'il est ici, vous devrez lui réapprendre à chaque fois toutes les règles de classification en partant de zéro ; n'hésitez pas, par conséquent, à lui faire subir toutes les modifications que vous jugerez utiles. Il serait magnifique de pouvoir faire usage d'un convertisseur analogique-numérique rattaché à une caméra, pour relier l'ordinateur au monde extérieur.

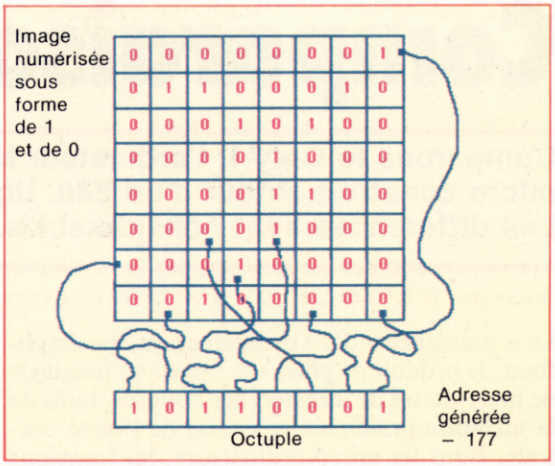
La forme originelle est conservée au sein d'un tableau bidimensionnel, I(,), dans lequel 1 représente un astérisque et 0 un espace. Au cours de la phase d'apprentissage, les adresses sont déterminées à partir d'éléments de I(,) choisis de façon aléatoire. Le bit correspondant (qui est fonction de la classe de formes à apprendre) est déterminé à la ligne 210. Ensuite, au cours de la phase d'identification, la même séquence d'adresses devra être générée afin de comparer la forme en cours d'évaluation aux classifications apprises précédemment.

Programme de reconnaissance de formes

Spectrum

```
10 REM **** RECONNAISSANCE ****
20 REM ****   DE FORMES    ****
25 REM ****   SPECTRUM     ****
30 CLEAR 49999 : GOSUB 4000
56 SX=40 : DS=6 : RB=2^DS
57 ME=SX*RB
60 DIM I(16,16),C(8)
64 LET D=50018
85 PRINT "TOTAL TAILLE RAM =";ME; "OCTETS"
88 GOSUB 1000 :REM TABLEAU D REMIS A ZERO
90 REM **** PHASE D'APPRENTISSAGE ****
92 M$="APPRENTISSAGE"
95 INPUT "DE QUELLE CLASSE (1....,8)"
96 IF C<1 OR>8 THEN 95
100 FOR I=1 TO 16:FOR J=1 TO 16 : I(I,J)=0
101 NEXT J:NEXT I
110 REM **** SAISIE IMAGE UTILISATEUR ****
120 GOSUB2000 : REM CREATION D'IMAGES
130 RANDOMIZE 1
140 FOR I=1 TO SX
150 A=RB*(I-1):REM RECOPIE ADRESSE BLOC RAM
160 FOR J=0 TO(DS-1)
170 R1=INT(RND(1)*16+1):RZ=INT(RND(1)*16+1)
190 A=A+I(R1,R2)*2↑J
200 NEXT J
210 LET X1=PEEK(D+A):LET Y1=2↑(C-1):GOSUB 6000
215 POKE(D+A),R1
220 NEXT I
230 INPUT"UNE AUTRE LEÇON D'APPRENTISSAGE"
233 IF A$="O" THEN 90:REM ON RECOMMENCE
234 REM **** PHASE D'IDENTIFICATION"
235 M$="IDENTIFICATION"
240 FOR I=1 TO 16:FOR J=1 TO 16
244 I(I,J)=0
245 NEXT J:NEXT I
250 GOSUB 2000:REM SAISIE IMAGE POUR IDENTIFICATION
270 RANDOMIZE 1
280 FOR C=1 TO 8 : C(c)=0:NEXT C
290 FOR I=1 TO SX
300 A=RB*(I-1)
310 FOR J=0 TO DS-1
320 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
340 A=A+I(R1,R2)*2↑J
350 NEXT J
355 FOR C=1 TO 8
360 LET X1=PEEK(D+A):LET Y1=2↑(C-1):GOSUB 5000
365 IF R1>0 THEN C(c)=C(c)+1
370 NEXT C
380 NEXT I
381 CLS
382 CX=0:X=0:Y=17:GOSUB 3000
385 FOR C=1 TO 8
388 PRINT"LA CLASSE"C"OBTIENT UN SCORE DE";
C(c)/SX*100
390 IF C(C)>C(CX) THEN CX=C
400 NEXT C
404 PRINT"LA CLASSE LA PLUS PROBABLE EST LA
CLASSE N°";CX
410 INPUT"VOULEZ-VOUS IDENTIFIER   UNE AUTRE
IMAGE (O/N)?";A$
420 IF A$="O" THEN 240:REM ON RECOMMENCE
444 END
999 :
1000 REM *** INITIALISATION MEMOIRE ****
1010 FOR I=0 TO ME:POKE(D+I),0:NEXT I
1030 RETURN
1040 :
2000 REM **** CREATION D'IMAGES ****
2005 CLS
2010 X=0:Y=20 : GOSUB 3000:PRINT"SERVEZ-VOUS DE H,
   B,G,D,X,* OU DE LA BARRE D'ESPACE"
2012 PRINT"POUR CREER VOTRE IMAGE"
2013 PRINT"PHASE D'";M$
2015 X=1:Y=1 : GOSUB 3000
2020 T=0:A$="  ": H=1 : V=1
2040 GET C$
2050 IF C$="H" THEN V=V-1 : IF V< 1 THEN V=16
2060 IF C$="B" THEN V=V+1 : IF V >16 THEN V=1
2070 IF C$="G" THEN H=H-1 : IF H< 1 THEN H=16
2080 IF C$="D" THEN H=H+1 :'IF H >16 THEN H=1
2110 IF C$=" "THEN T=0 : A$=C$
2120 IF C$="*"THEN T=1 : A$=C$
2130 I(H,V)=T
2140 X=H:Y=V: GOSUB 3000:PRINT A$;
2150 IF C$<>"X" THEN 2040
2155 GOSUB 2200
2160 RETURN
2170 :
2200 REM **** S/P AFFICHAGE ****
2210 FOR H=1 TO 16
2220 FOR V=1 TO 16
2230 X=H : Y=V : GOSUB 3000:IF I(H,V)=0 THEN PRINT
   ":";GOTO 2240
2235 PRINT" ";
2240 NEXT V:NEXT H
2250 X=0 : Y=17 : GOSUB 3000
2260 RETURN
2270 :
3000 REM **** TABULATION ****
3010 PRINT CHR$(19);TAB(X);LEFT$(DW$,Y);
3020 RETURN
```





Commodore 64

```
10 REM **** RECONNAISSANCE ****
20 REM **** DE FORMES C64  ****
30 FORI=1 TO 25:DW$=DW$+CHR$(17):NEXT I
56 SX=40:DS=6:RB=2^DS
57 ME=SX*RB
60 DIM I(16,16),C(8)
64 D=12*4096:REM UTILISE  $C000 POUR TABLEAU D
85 PRINT"TOTAL TAILLE RAM=";ME;"OCTETS"
88 GOSUB 1000:REM TABLEAU D REMIS A ZERO
90 REM **** PHASE D'APPRENTISSAGE ****
92 M$="APPRENTISSAGE"
95 INPUT"DE QUELLE CLASSE EST (1...  8):C"
96 IFC<1 OR C>8 THEN 95
100 FOR I=1 TO 16:FOR J=1 TO 16:I(I,J)=0
101 NEXT J:NEXT I
110 REM **** SAISIE IMAGE UTILISATEUR ****
120 GOSUB 2000:REM CREATION D'IMAGES
130 R=RND(-1):REM VALEUR ORIGINELLE ALEATOIRE
140 FORI=1 TO SX
150 A=RB*(I-1):REM RECOPIE ADRESSE BLOC RAM
160 FOR J=0 TO (DS-1)
170 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
190 A=A+I(R1,R2)*2^J
200 NEXT J
210 POKE(D+A),(PEEK(D+A) OR 2^(C-1))
220 NEXT I
230 INPUT"UNE AUTRE LEÇON D'APPRENTISSAGE (O/N):A$
233 IF A$="O" THEN 90:REM REPETITION
234 REM **** PHASE D'IDENTIFICATION ****
235 M$="IDENTIFICATION"
240 FOR I=1 TO 16:FOR J=1 TO 16
244 I(I,J)=0
245 NEXT J:NEXT I
250 GOSUB 2000:REM SAISIE IMAGE POUR CLASSIFICATION
270 R=RND(-1)
280 FORC=1 TO 8:C(C)=0:NEXT C
290 FOR I=1 TO SX
300 A=RB*(I-1)
310 FOR J=0 TO DS-1
320 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
340 A=A+I(R1,R2)*2^J
350 NEXT J
355 FOR C=1 TO 8
360 IF (PEEK(D+A)AND 2^(C-1)) 0 THEN C(C)=C(C)+1
370 NEXT C
380 NEXT I
381 PRINTCHR$(147)
382 CX=0:X=0:Y=17:GOSUB 3000
385 FOR C=1 TO 8
389 PRINT"LA CLASSE" C "OBTIENT UN SCORE DE";C(C)/
SX*100
390 IF C(C)>C(CX) THEN CX=C
400 NEXT C
404 PRINT"LA CLASSE LA PLUS PROBABLE EST LA CLASSE
N°";CX
410 INPUT "VOULEZ-VOUS IDENTIFIER UNE AUTRE IMAGE
(O/N)?";A$
420 IF A$="O"THEN 240:REM ON RECOMMENCE
444 END
999 :
1000 REM ***** INITIALISATION MEMOIRE ****
1010 FOR I=0 TO ME:POKE(D+I),0:NEXT I
1030 RETURN
1040 :
2000 REM **** CREATION D'IMAGES ****
2005 PRINTCHR$(147)
2010 X=0:Y=20:GOSUB 3000:PRINT"SERVEZ-VOUS DE
H,B,G,D,X,* OU DE LA BARRE D'ESPACE"
2012 PRINT "POUR CREER VOTRE IMAGE"
2013 PRINT"PHASE:";M$
2015 X=1:Y=1:GOSUB 3000
2020 T=0:A$=" ":H=1:V=1
2040 GET C$
2050 IF C$="H" THEN V=V-1:IF V<1 THEN V=16
2060 IF C$="B" THEN V=V+1:IF V>16 THEN V=1
2070 IF C$="G" THEN H=H-1:IF H<1 THEN H=16
2080 IF C$="D" THEN H=H+1:IF H>16 THEN H=1
2110 IF C$=" " THEN T=0:A$=C$
2120 IF C$="*" THEN T=1:A$=C$
2130 I(H,V)=T
2140 X=H:Y=V:GOSUB 3000:PRINTA$;
2150 IF C$<>"X" THEN 2040
2155 GOSUB 2200
2160 RETURN
2170 :
2200 REM **** S/P AFFICHAGE ****
2210 FOR H=1 TO 16
2220 FOR V=1 TO 16
2230 X=H:Y=V:GOSUB3000:IF I(H,V)=0 THEN PRINT";":G
OTO2240
2235 PRINT" ";
2240 NEXT V:NEXT H
2250 X=0:Y=17:GOSUB 3000
2260 RETURN
2270 :
3000 REM **** TABULATION ****
3010 PRINTCHR$(19):TAB(X):LEFT$(DW$,Y);
3020 RETURN
```

Opérateurs locaux
Nous vous présentons (à gauche) un exemple de traitement élémentaire de données, tel qu'il peut être mis en œuvre par un système de reconnaissance de formes procédant « de bas en haut ». Les grilles de 3 × 3 sont des « opérateurs locaux » ; elles analysent les données relatives à l'image et enregistrent des résultats élevés aux zones « frontières », ce qui permet de définir, très grossièrement, l'image comme un ensemble de lignes droites. Les opérateurs sont déplacés sur toute l'image et chaque nombre est multiplié par l'intensité du niveau de gris de la section au-dessus de laquelle il se trouve. L'addition des produits ainsi obtenus donne des résultats élevés quand la caractéristique recherchée est présente, faibles dans le cas contraire. Chacun des neuf nombres de l'opérateur local est un facteur de pondération ; la conformation de ces facteurs détermine quelle sera la forme identifiée. (Cl. Caroline Clayton sur Macintosh.)

Un œil octuple
Des groupes de 8 pixels sont connectés arbitrairement à des registres de 8 bits et constituent la base du système de reconnaissance de formes Wisard. Chaque octuple correspond à un bloc de 256 adresses mémoire. Au cours de la phase d'apprentissage, la valeur contenue dans l'octuple détermine une adresse mémoire en RAM, selon la disposition des pixels, et une valeur de 1 est écrite à l'adresse correspondante. Si la même forme est présentée au système au cours de la phase d'identification, la même adresse sera générée et la présence d'un 1 permettra la reconnaissance de la structure en question.

Comparaisons

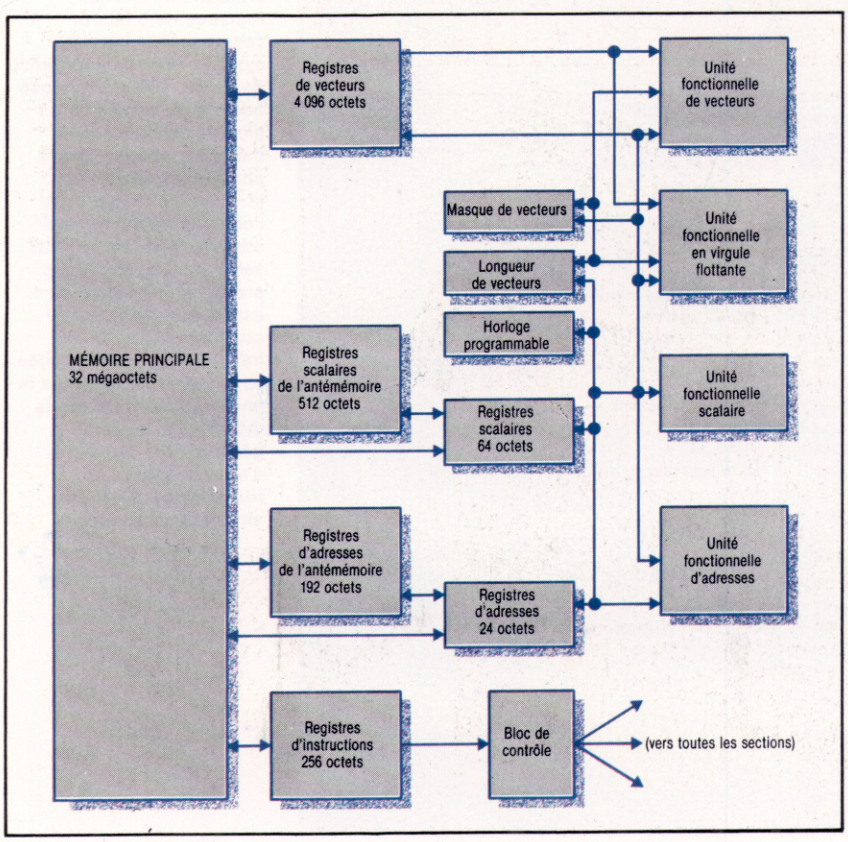
Comparons le Cray-1, l'ordinateur le plus puissant du monde, et un micro construit autour d'un Z80. Une application servira de référence. Les différences sont considérables.

La « puissance » d'un ordinateur est, en simplifiant, le produit de plusieurs facteurs : longueur de mot, vitesse de transfert des données, taille de la mémoire principale et vitesse de l'unité centrale. Pour les micro-ordinateurs, les fonctions des UC traditionnelles sont rassemblées dans un seul microprocesseur. Parmi ceux-ci, mentionnons le Z80, les Intel 8088 et 8086 et le Motorola M68000. Tous utilisent la technologie MOS *(Metal Oxyde Semiconductor)* pour les circuits logiques et une mémoire sur puce. Les données sont transférées et traitées en parallèle sur 8 ou 16 bits à la fois. Les fréquences d'horloge vont de 1 MHz à 12 MHz.

De telles caractéristiques, bien qu'assez impressionnantes, ne permettent pas aux micro-ordinateurs de traiter la quantité colossale de données résultant d'applications comme le traitement des images, la dynamique des fluides et les prévisions météorologiques.

Prenons un exemple concret. Supposons que vous créez un film avec des graphiques générés par ordinateur, et que le film implique une résolution de 6 000 sur 6 000 pixels, et 24 images par seconde. Puisque les graphiques sont animés, la position de chaque point de l'image peut se déplacer entre chaque image. Cela nécessite 864 millions de calculs par seconde; chacun de ces calculs risque d'être assez compliqué et d'entraîner des douzaines ou des centaines d'instructions écrites en code machine. Cela signifie des milliards d'instructions par seconde. L'encadré « Essais chronométrés » donne une comparaison approximative du temps nécessaire pour produire une séquence de film de 10 minutes à la fois sur un super-ordinateur et sur un micro fondé sur un Z80.

La machine
Le Cray-1 (photographié ci-dessus) offre à l'utilisateur une énorme puissance de traitement sous le contrôle d'un ordinateur frontal comme un IBM, un DEC ou un autre ordinateur similaire.



Les super-ordinateurs — ainsi sont surnommés les grands systèmes — fonctionnent un peu de la même manière que les micros; les instructions sont extraites de la mémoire, les données sont manipulées par le processeur selon ses instructions et les résultats sont stockés en mémoire. Principales différences : la vitesse à laquelle ces opérations sont exécutées, et la manière dont sont construites les diverses parties des ordinateurs.

Comme exemple de super-ordinateur, nous avons choisi le Cray-1S/4400, fabriqué par Cray Research. Le premier Cray-1 fut installé en 1976, seulement quatre ans après la création de la société. Depuis lors, il a acquis la réputation d'être l'un des grands systèmes parmi les plus puissants qui existent.

L'UC du Cray-1 occupe une armoire semi-circulaire d'une hauteur de 2 m, dont la forme rappelle celle d'un canapé courbe (le système de refroidissement se trouve sous les « sièges »). Sa vitesse provient de l'utilisation d'une logique à semi-conducteurs bipolaires (les semi-conducteurs bipolaires sont des transistors « ordinaires », par opposition aux MOS, CMOS, NMOS, FET — transistors à effet de champ —, et à d'autres types de semi-conducteurs). La logique bipolaire et la mémoire sont contenues dans plus de 200 000 circuits intégrés sur 3 400 cartes distinctes de circuits imprimés, avec plus de 90 km de fils servant à les interconnecter.

La taille des mots utilisés dans les calculs est de 64 bits (traitement simultané de données huit fois supérieur à celui d'un Z80 à 8 bits) et l'horloge système fonctionne à 80 MHz (80 millions de cycles par seconde). Une addition typique de 64 bits ne met que 37,5 ns. Une addition de 8 bits dans un Z80 fonctionnant à 4 MHz (par exemple, ADD A, n) est effectuée en 1,75 μs.

La mémoire principale du Cray fait partie intégrante de l'UC. Elle renferme 32 mégaoctets disposés en 4 194 304 mots. La mémoire peut trans-

férer jusqu'à 2 560 millions d'octets par seconde. Comme vous pouviez le supposer, le Cray dispose d'un ensemble impressionnant de registres d'UC. Il y a 72 registres d'adresses (24 bits de largeur), 72 registres scalaires (64 bits de largeur) et 8 registres de vecteurs (64 mots de largeur). L'espace total de registre UC est donc de 4 888 octets (celui du Z80 n'est que de 26).

L'ordinateur frontal

Contrairement à la plupart des ordinateurs, qui sont des unités autonomes, les ordinateurs Cray sont conçus pour être utilisés comme extensions à des grands systèmes ou à des mini-ordinateurs. L'ordinateur frontal sert d'interface; il reçoit les entrées provenant des terminaux ou des lecteurs de cartes, et dirige une sortie vers des unités périphériques, telles des imprimantes ou des unités de stockage sur bande magnétique.

Le sous-système d'E/S du Cray se situe entre l'UC et l'ordinateur frontal principal; ce sous-système sert à rendre le fonctionnement du système Cray compatible avec les gros systèmes IBM, DEC, Data General ou autres. Le sous-système d'E/S Cray est composé de deux ou de quatre processeurs d'E/S, chacun ayant la capacité de traitement d'un mini-ordinateur.

Cette description du Cray-1 est très superficielle, bien que le diagramme fonctionnel de l'UC puisse vous donner un aperçu de son perfectionnement.

Nous ne pouvons aborder dans cet article le jeu d'instructions ni les 13 unités fonctionnelles pouvant fonctionner en parallèle, ni le traitement vectoriel qui permet d'utiliser 64 paires d'opérandes sur une seule instruction. Mais tout cela est suffisant pour dire que le Cray-1 est une machine très puissante.

Applications pour super-ordinateurs

Prévisions météorologiques

Les prévisions météorologiques sont le résultat d'une cueillette de données météorologiques à l'échelle mondiale, de liaisons par satellites et d'utilisation de modèles informatiques.
Un modèle météorologique de la planète implique des centaines de millions de calculs sur d'énormes quantités de données. Seuls des ordinateurs très puissants, comme le Cray, peuvent gérer une telle quantité de données aussi rapidement.

Dynamique des fluides

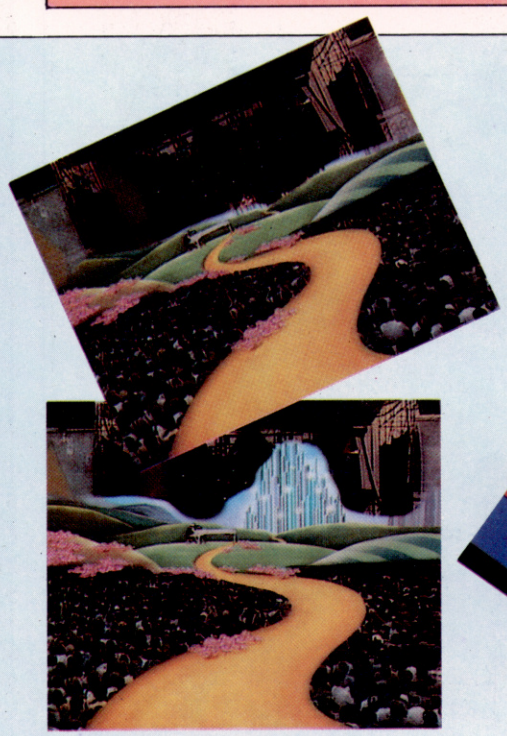
La dynamique des fluides joue un rôle très important dans la conception d'automobiles à faible consommation de carburant, de systèmes de refroidissement de centrales nucléaires, dans la construction aéronautique, etc. L'analyse et la prévision des mouvements nécessitent le traitement d'une quantité de données gigantesque. Les problèmes deviennent énormes lorsqu'il est nécessaire de fournir des résultats en temps réel. La principale difficulté vient du fait que chaque particule du fluide — et il y en a des milliards — a, par son mouvement, un effet sur les autres particules du système. Le calcul du comportement global exige donc une puissance de traitement colossale.

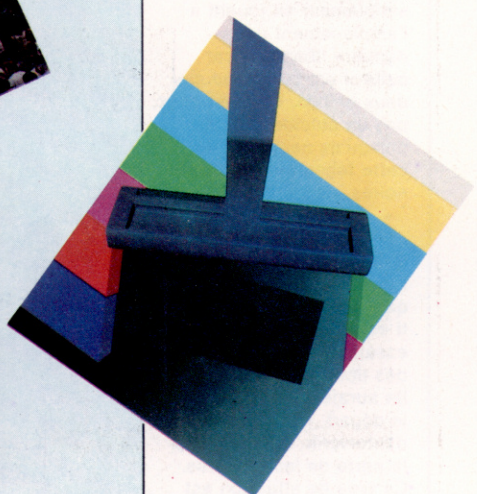
Prévisions économiques

Les modèles économiques ont la réputation d'être très difficiles à construire. Comme en mécanique des fluides, une toute petite variation au niveau d'une composante peut avoir des effets qui touchent l'ensemble du système. La création d'un modèle économique du monde entier n'est pratiquement pas réalisable; mais même des modèles simplifiés sont d'une complexité impressionnante. De nouveau, il est nécessaire d'avoir recours à des ordinateurs extrêmement rapides et puissants afin d'obtenir les résultats en temps utile. L'utilisation des super-ordinateurs s'impose lorsque le facteur temps est essentiel dans une application.

Essais chronométrés

Afin de comparer la puissance de traitement du Cray-1 à celle d'un ordinateur domestique construit autour d'un Z80, comme l'Amstrad CPC 464, prenons par exemple la réalisation d'une séquence de film d'animation de 10 minutes, en graphiques haute résolution. Nous supposons une résolution image de 6 000 × 6 000 points, et 24 images par seconde. Nous supposons également que chaque point doit être calculé séparément pour chaque nouvelle image, et que cent instructions en code machine seront nécessaires pour la version Z80, avec un temps d'exécution moyen de 19 cycles horloge (4,75 μs). Pour le Cray-1, avec ses puissantes instructions de traitement de vecteurs, supposons que nous ayons besoin de vingt-cinq instructions écrites en code machine (estimation au minimum) avec un temps moyen d'exécution de 4 cycles d'horloge (50 ns).
Un Z80 mettrait $6\,000^2 \times 24 \times 10 \times 100 \times 4{,}75 \times 10^{-6} = 2{,}462\,4 \times 10^8$ s, soit 7 ou 8 années. Un Cray-1 mettrait $6\,000^2 \times 24 \times 60 \times 10 \times 50 \times 10^{-9} = 6{,}46 \times 10^5$ s, soit 7,5 jours. (CAL Video for Mike Mansfield Enterprises.)





Révolution visuelle
Des plus grands succès du rock aux clips publicitaires, les graphiques générés par ordinateur ont révolutionné la production vidéo et cinématographique. De telles images peuvent être créées facilement par un mini-ordinateur moderne et en utilisant la technologie micro. A une large échelle, un Cray-1 servit à générer plus de vingt minutes de tournage pour le film *The Last Starfighter*; il effectua des calculs pour lesquels un micro 8 bits aurait mis plus de quinze ans ! (Cl. CAL Video pour B.B.D.O.)

Noir et blanc

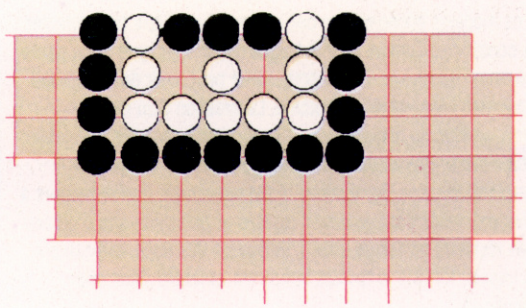
Voici les listages qui permettront d'adapter le programme de go au C64, au Spectrum et à l'Amstrad 464/664. Ils sont consacrés à l'affichage du plateau de jeu et comportent plusieurs modules.

A chaque fois, nous nous sommes efforcés de respecter une architecture d'ensemble reposant sur des principes voisins. Cela nous permet de donner une certaine unité aux listages du Commodore 64, du Spectrum de Sinclair et de l'Amstrad CPC 464/664 relatifs au jeu de go.

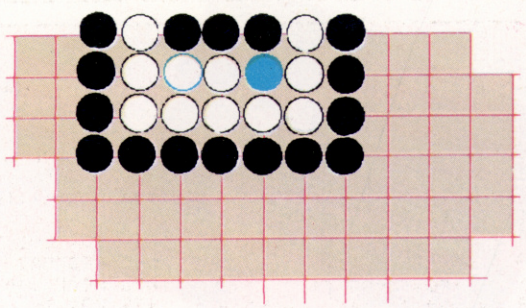
Par souci de clarté, les numéros de ligne sont pratiquement les mêmes. Les limitations de telle ou telle machine, ou les particularités de leur dialecte BASIC, ont parfois imposé des modifications aux algorithmes originaux. C'est par exemple le cas pour tout ce qui concerne l'affichage écran. Retenez pourtant que tous les listages, par-delà leurs différences de détail, suivent une logique analogue; un examen comparatif vous en convaincra sans peine. Ce sera aussi un bon moyen d'étudier sur pièces la conversion d'un programme donné, de façon qu'il puisse tourner sur des ordinateurs différents.

Le jeu de go est une excellente base conceptuelle pour d'autres applications logicielles. Par exemple, les principes de ce jeu permettraient de concevoir de bons programmes de stratégie militaire ou économique.

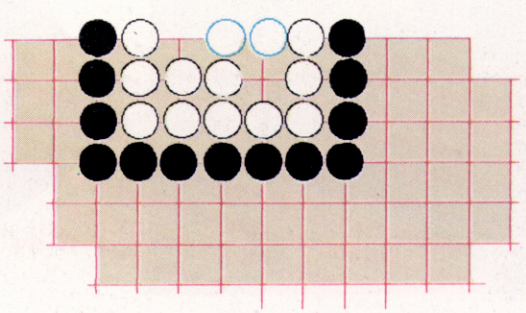
Seki



Qui encercle qui ?
Le groupe blanc est complètement encerclé par le groupe noir, et il ne lui reste que deux libertés. Mais il entoure également trois pierres noires.



Une erreur fatale
Si les noirs entreprennent d'occuper une des deux libertés blanches, en espérant ainsi capturer le groupe tout entier, les blancs répondront simplement en jouant à l'emplacement de la dernière liberté — car celle-ci est commune aux deux camps. Ce n'est donc pas un suicide (illégal), puisqu'ils capturent de cette façon quatre pierres noires.



Sain et sauf
Les blancs n'auront plus qu'à se doter de deux yeux. Il est inutile que les noirs essaient d'occuper l'une des libertés restantes. Ils auraient dû, dès le départ, s'abstenir d'intervenir, et accepter la perte de leurs pierres. Ce type de situation est appelé *seki*.

Caroline Clayton

Module un

Commodore 64

```
10 POKE 56578,PEEK(56578)OR3:POKE 56576,
(PEEK(56576)AND252)OR1
15 POKE648,132:POKE 52,128:POKE 56,128
20 DIM SK%(300)
30 POKE 53280,0:POKE 53281,0:PRINT"[CLEA
R][DOWN][DOWN][DOWN][DOWN][DOWN][DOWN][D
OWN][DOWN][DOWN][DOWN]                [WHITE]
PLEASE WAIT":GOSUB 170
40 GOSUB 1270
50 PRINT"[CLEAR]":GOSUB 1730
60 MOVE%=MOVE%+1:REM WHITE MOVES HERE
70 IF FIN% GOTO 100
80 MOVE%=MOVE%+1:REM BLACK MOVES HERE
90 IF NOT FIN% GOTO 60
100 IP%=22:IM%=9:IW%=1:GOSUB 1990
110 IF A$="Y" GOTO 40
115 IF A$<>"N" GOTO 100
120 MP%=24:MM%=5:O$="":GOSUB 2160
130 GOTO130
170 REM INITIALISE ROUTINE
190 BLACK%=1:WHITE%=2:COLOUR%=3
200 MARKER%=4:LIBERTY%=8
220 BOARD=49152
240 DIM CAPTURE%(2)
250 AXIS$="[c 3]ABCDEFGHIJKLMNO"
260 DIM DIR%(4)
270 FOR L=1 TO 4
280 READ DIR%(L)
290 NEXT
300 DATA 16,1,-16,-1
310 GOSUB 390
350 RETURN
390 REM READ-MESSAGES ROUTINE
420 DIM MESS$(9)
430 FOR M=0 TO 9
440 READ MESS$(M)
450 NEXT
460 DATA "O.K. THINKING..."
470 DATA "ILLEGAL ENTRY: "
480 DATA "STONE ALREADY ON POINT: "
490 DATA "ILLEGAL. KO ON POINT: "
500 DATA "ILLEGAL SUICIDE ON POINT: "
510 DATA "O.K. GAME OVER. PRESS RUN/STOP
"
520 DATA ""
530 DATA "  HOW MANY HANDICAP STONES MAY
 I          HAVE (2-9) ?"
```

```
540 DATA "TYPE YOUR MOVE (EG. H8), PASS
OR          QUIT :"
550 DATA "DO YOU WANT TO PLAY AGAIN (Y/N
)?"
560 RETURN
1270 REM INTRODUCTION ROUTINE
1280 GOSUB 1340
1290 GOSUB 1450
1340 REM GAME-INIT ROUTINE
1345 STACK%=1
1360 A1$="      ":A2$="      "
1370 LOCAT%=0:MOVE%=1
1380 FIN%=0
1390 CAPTURE%(1)=0:CAPTURE%(2)=0
1410 RETURN
1450 REM TITLE-SCREEN ROUTINE
1460 POKE 53280,0:POKE 53281,0
1470 PRINT"[CLEAR][DOWN][DOWN][DOWN][DOW
N]                [c 3][RVSON][c V][c I][c
 C][RVSOFF]  [RVSON][c V][c I][c C]"
1480 PRINT"                [RVSON] [RVSOFF
][c D][c I]  [RVSON] [RVSOFF] [RVSON] "
1490 PRINT"                [RVSON] [RVSOFF
] [RVSON] [RVSOFF]  [RVSON] [RVSOFF] [RV
SON] "
1500 PRINT"                [c C][RVSON][c
 I][RVSOFF][c V]  [c C][RVSON][c I][RVSOF
F][c V]"
1510 PRINT"[DOWN][DOWN]          [CYAN]BY
 MARCUS  JEFFERY"
1520 PRINT"[DOWN][PURPLE]YOU WILL PLAY T
HE [WHITE]WHITE [PURPLE]STONES,  AND"
1530 PRINT"THE  COMPUTER  (BEING WEAKER!
)  WILL"
1540 PRINT"HAVE THE [c 3]RED [PURPLE]STO
NES WITH A  HANDICAP"
1550 PRINT"              ADVANTAGE."
1560 IP%=20:IM%=7:IW%=1:GOSUB 1990
1565 HND%=VAL(A$)
1570 IF HND%<2 OR HND%>9 GOTO 1560
1590 RETURN
1730 REM PRINT-BOARD ROUTINE
1750 PRINT"[HOME][YELLOW]       STONES CA
PTURED BY:";
1760 PRINT"[WHITE]       MOVE"
1770 PRINT"[YELLOW]     WHITE =[CYAN] ";C
APTURE%(2);
1780 PRINT TAB(18);"[YELLOW]BLACK =[CYAN
] ";CAPTURE%(1);
1790 PRINT TAB(31);"[WHITE]";MOVE%
1800 PRINT TAB(12);"[DOWN]";AXIS$
1810 FOR Y=15 TO 1 STEP -1
1820 PRINT TAB(9);"[c 3]";RIGHT$(" "+STR
$(Y),2);" ";
1830 FOR X=1 TO 15
1840 PP%=PEEK(BOARD+16*Y+X)
1850 IF PP%=1 THEN PRINT"[c 3][s Q]";:GO
TO 1880
1860 IF PP%=2 THEN PRINT"[WHITE][s Q]";:
GOTO 1880
1870 PRINT"[c 6][s +]";
1880 NEXT
1890 PRINT"[c 3] ";RIGHT$(" "+STR$(Y),2)
1900 NEXT
1910 PRINT TAB(12);AXIS$
1920 PRINT"[DOWN][YELLOW]    LAST MOVE:";
1930 ITCP%=LOCAT%:GOSUB 2260:PRINT TAB(2
4);A1$
1940 PRINT TAB(18);"[DOWN][CYAN]";A2$
1950 RETURN
1990 REM INPUT ROUTINE
2000 IS%=0
2010 POKE 780,0:POKE 781,IP%+1:POKE 782,
39:SYS 65520
2015 FOR II=1 TO 79:PRINT CHR$(20);:NEXT
2020 A$=""
2030 PRINT"[WHITE]    ";MESS$(IM%);" ";
2060 GET I$:IF I$="" GOTO 2060
2065 IF ASC(I$)=13 GOTO 2120
2070 IF ASC(I$)<>20 GOTO 2100
2080 IF IS%>0 THEN IS%=IS%-1:A$=LEFT$(A$
,IS%):PRINT I$;:GOTO 2060
2085 GOTO 2060
2100 IF IS%<IW% THEN IS%=IS%+1:A$=A$+I$:
PRINT"[CYAN]";I$;
2110 GOTO 2060
2120 RETURN
2160 REM MESSAGE ROUTINE
2190 POKE 780,0:POKE 781,MP%:POKE 782,39
:SYS 65520
2195 FOR ML=1 TO 39:PRINT CHR$(20);:NEXT
2200 PRINT"    [RVSON][c 3]";MESS$(MM%);
O$;"[RVSOFF]";
2220 RETURN
2260 REM INT-TO-CHAR ROUTINE
2270 IF ITCP%=0 THEN PRINT"[CYAN]HANDICA
P";:RETURN
2275 C$=CHR$(ITCP%-16*INT(ITCP%/16)+64)+
MID$(STR$(INT(ITCP%/16)),2)
2280 PRINT"[CYAN]";C$;"      ";:RETURN
```

Amstrad CPC 464/664:

```
10 MEMORY &9FFF
20 DIM s(300):REM stack
30 GOSUB 170:REM init
40 GOSUB 1270:REM introduction
50 CLS:GOSUB 1730:REM print board
60 mve%=mve%+1:REM white moves here
70 IF over% THEN 60
80 mve%=mve%+1:REM black moves here
90 IF NOT over% THEN 60
100 ip%=23:im%=9:iw%=1:GOSUB 1990
110 IF a$="Y" THEN 40
115 IF a$<>"N" THEN 100
120 mp%=25:mm%=5:o$="":GOSUB 216
0:REM message
130 END
170 REM init routine
190 black%=1:white%=2:colour%=3
195 INK 1,26:REM white
197 INK 2,24:REM yellow
200 marker%=4:liberty%=8
220 board=&A000
240 DIM capture%(2)
250 axis$="A B C D E F G H I J K
 L M N O"
260 GOSUB 390:REM read messages
290 DIM dir%(4)
300 RESTORE 340
310 FOR i%=1 TO 4
320 READ dir%(i%)
330 NEXT i%
340 DATA 16,1,-16,-1
350 RETURN
390 REM read message routine
410 RESTORE 460
420 DIM mess$(9)
430 FOR m%=0 TO 9
440 READ mess$(m%)
450 NEXT m%
460 DATA "O.K. THINKING..."
470 DATA "Illegal entry: "
480 DATA "Stone already on point
: "
490 DATA "Illegal. Ko on point: "
500 DATA "Illegal. Suicide on
point: "
510 DATA "    O.K. GAME OVER"
520 DATA ""
530 DATA "How many handicap ston
es may I          have (2-9) ?"
540 DATA "Type your move (eg. H8
) PASS or          QUIT :"
550 DATA "Do you want to play ag
ain (Y/N)?"
560 RETURN
1270 REM introduction routine
1280 GOSUB 1340:REM game init
1290 GOSUB 1450:REM title screen
1300 RETURN
1340 REM game init routine
1345 stack%=1
1360 atari1$="      ":atari2$="
1370 location%=0:mve%=1
1380 over%=0
1390 capture%(1)=0:capture%(2)=0
1410 RETURN
1450 REM title screen routine
1460 CLS
1462 FOR c=1 TO 20 STEP 3
1465 RESTORE 1474
1470 MOVE 260+c,320+c
1472 FOR i=1 TO 8:READ x,y:DRAWR
 x,y,3:NEXT i
1474 DATA -10,10,-50,0,-10,-10,0
,-80,10,-10,50,0,10,10,0,20
1476 MOVER -10,0:DRAWR 20,0
1478 MOVE 380+c,320+c
1480 FOR i=1 TO 8:READ x,y:DRAWR
 x,y:NEXT i
1482 DATA -10,10,-50,0,-10,-10,0
,-80,10,-10,50,0,10,10,0,80
1484 NEXT c
1500 LOCATE 12,13:PEN 3:PRINT
1510 LOCATE 1,16:PEN 2:PRINT "Yo
u will play the ";
1520 PEN 1:PRINT"white ";:PEN 2:
PRINT"stones,  and"
1530 PRINT"and the computer (bei
ng weaker) will"
1540 PRINT"have the ";:PEN 3:PRI
NT"red ";:PEN 2:PRINT"stones wit
```

```
h a  handicap"
1550 PRINT TAB(13)"advantage"
1560 ip%=22:im%=7:iw%=1:GOSUB 19
90:REM input
1565 hand%=ASC(a$)-48
1570 IF hand%<2 OR hand%>9 THEN
1560
1590 RETURN
1730 REM print board routine
1735 INK 0,4:BORDER 9:REM magent
a/green
1750 LOCATE 3,1:PEN 2:PRINT" Sto
nes captured by:";TAB(32)"Move"
1770 PRINT TAB(3)"White =";:PEN
3:PRINT capture%(2);
1780 PEN 2:PRINT"   Black =";:PE
N 3:PRINT capture%(1);
1790 PRINT TAB(33)mve%
1800 PRINT:PRINT TAB(5)axis$
1810 FOR y%=15 TO 1 STEP -1
1820 LOCATE 2,20-y%:PRINT RIGHT$
(" "+STR$(y%),2);
1830 FOR x%=1 TO 15
1835 PRINT" ";
1840 p%=PEEK(board+16*y%+x%)
1850 IF p%=1 THEN PEN 3:PRINT"O"
;:GOTO 1880
1860 IF p%=2 THEN PEN 1:PRINT"O"
;:GOTO 1880
1870 PEN 2:PRINT"+";
1880 NEXT x%
1890 PEN 3:PRINT TAB(34)y%
1900 NEXT y%
1910 PRINT TAB(5)axis$
1920 PEN 2:PRINT TAB(3)"My last
move was:";
1930 itcp%=location%:GOSUB 2260:
PEN 1:PRINT TAB(30) atari1$
1940 LOCATE 16,23:PRINT atari2$
1950 RETURN
1990 REM input routine
2000 is%=0
2010 LOCATE 3,ip%:PRINT SPACE$(62)
2020 a$=""
2030 LOCATE 3,ip%:PEN 3:PRINT me
ss$(im%);
2060 i$="":WHILE i$="":i$=INKEY$
:WEND
2065 IF i$=CHR$(13) THEN 2120
2070 IF i$<>CHR$(127) THEN 2090
2080 IF is%>0 THEN is%=is%-1:a$=
LEFT$(a$,is%):PRINT CHR$(8);" ";
CHR$(8);
2085 GOTO 2060
2090 IF i$>="a" AND i$<="z" THEN
 i$=CHR$(ASC(i$)-32)
2100 IF is%<iw% THEN is%=is%+1:a
$=a$+i$:PEN 2:PRINT i$;
2110 GOTO 2060
2120 RETURN
2160 REM message routine
2190 LOCATE 3,mp%:PRINT SPACE$(3
6);
2200 LOCATE 3,mp%:PEN 3:PRINT me
ss$(mm%);o$
2220 RETURN
2260 REM int-to-char routine
2270 IF itcp%=0 THEN PEN 1:PRINT
"Handicap";:RETURN
2275 c$=CHR$(itcp%-16*INT(itcp%/
16)+64):r$=MID$(STR$(INT(itcp%/1
6)),2)
2280 PEN 1:PRINT c$;r$;:RETURN
```

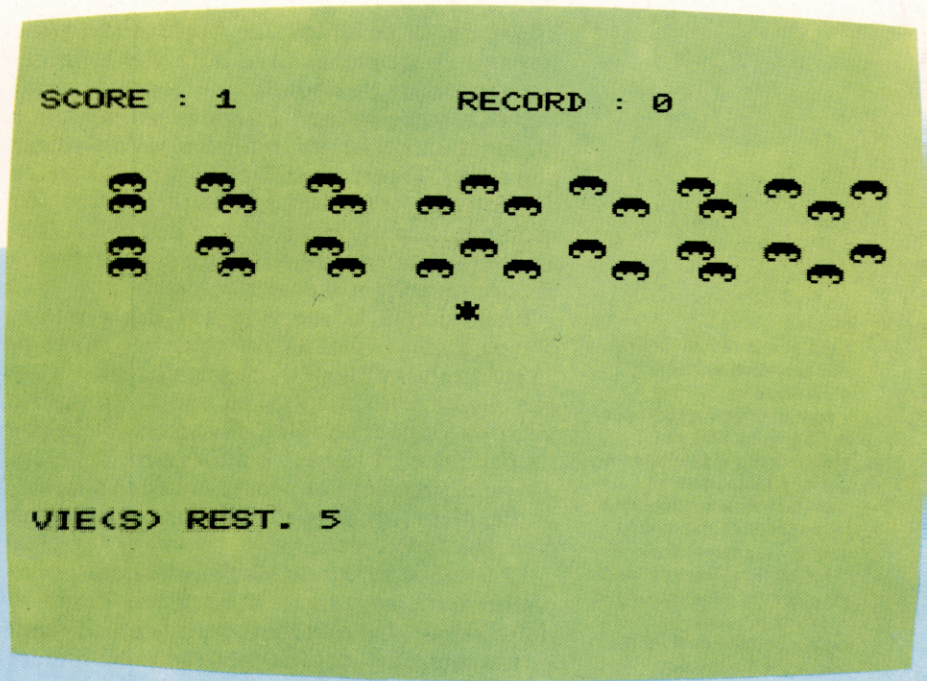
Sinclair Spectrum:

```
  10 CLEAR 63999
  20 DIM s(300)
  30 PRINT AT 10,10;"PLEASE WAIT
": GO SUB 170
  40 GO SUB 1270
  50 CLS : GO SUB 1730
  60 LET move=move+1: REM white
moves here
  70 IF end THEN  GO TO 100
  80 LET move=move+1: REM black
moves here
  90 IF NOT end THEN  GO TO 60
 100 LET ip=20: LET im=10: LET i
w=1: GO SUB 1990
 110 IF a$="Y" THEN  GO TO 40
 115 IF a$<>"N" THEN  GO TO 100
 120 LET mp=21: LET mm=6: LET o$
="": GO SUB 2160
 130 STOP
 170 REM initialise routine
 190 LET black=1: LET white=2: L
ET colour=3
 200 LET marker=4: LET liberty=8
 220 LET board=64000
 240 DIM c(2)
 260 GO SUB 390
 290 DIM d(4)
 300 RESTORE 340
 310 FOR l=1 TO 4
 320 READ d(l)
 330 NEXT l
 340 DATA 16,1,-16,-1
 350 RETURN
 390 REM read-messages routine
 410 RESTORE 460
 420 DIM m$(10,43)
 430 FOR m=1 TO 10
 440 READ m$(m)
 450 NEXT m
 460 DATA "O.K. THINKING..."
 470 DATA "Illegal entry: "
 480 DATA "Stone already on poin
t: "
 490 DATA "Illegal. Ko on point:"
 500 DATA "Illegal. Suicide on p
oint: "
 510 DATA "    O.K. GAME OVER."
 520 DATA ""
 530 DATA "How many handicap sto
nes may I  have (2-9) ?"
 540 DATA "Type your move (eg. H
8), PASS   or QUIT :"
 550 DATA "Do you want to play a
gain       (Y/N) ?"
 560 RETURN
1260:
1270 REM introduction routine
1280 GO SUB 1340
1290 GO SUB 1450
1300 RETURN
1340 REM game-init routine
1345 LET stack=1
1360 LET x$="      ": LET y$="
 "
1370 LET location=0: LET move=1
1380 LET end=0
1390 LET c(1)=0: LET c(2)=0
1410 RETURN
1450 REM title-screen routine
1460 BORDER 0: PAPER 0: CLS
1470 INK 2: PRINT AT 3,12;"sh23s
h1  sh23sh1"
1480 PRINT AT 4,12;"sh84sh3  sh8
 sh8"
1490 PRINT AT 5,12;"sh8  sh8  sh8
 sh8"
1500 PRINT AT 6,12;"132  132"
1510 PRINT AT 8,7; INK 5;"by  Ma
rcus Jeffery"
1520 PRINT : PRINT  INK 3;"You w
ill play the "; INK 7;"white"; I
NK 3;" stones."
1530 PRINT  INK 3;"and the compu
ter (being weaker)"
1540 PRINT  INK 3;"will have the
 "; INK 2;" red  "; INK 3;"stone
s with"
1550 PRINT  INK 3;"      a handic
ap advantage."
1560 LET ip=15: LET im=8: LET iw
=1: GO SUB 1990
1565 LET hand=CODE (a$)-48
1570 IF hand<2 OR hand>9 THEN  G
O TO 1560
1590 RETURN
1730 REM print-board routine
1750 PRINT AT 0,0; INK 6;" Stone
s captured by:";
1760 PRINT  INK 7;"       Move "
1770 PRINT  INK 6;" White="; INK
 5;c(2);
1780 PRINT TAB 10; INK 6;"  Blac
k ="; INK 5;c(1);
1790 PRINT TAB 27; INK 7;move
1810 FOR y=15 TO 1 STEP -1
1820 PRINT AT 17-y,5; INK 2;(" "
+STR$ y)(LEN STR$ y TO );" ";
1830 FOR x=1 TO 15
1840 LET pp=PEEK (board+16*y+x)
1850 IF pp=1 THEN  PRINT  PAPER
4; INK 2;"O";: GO TO 1880
1860 IF pp=2 THEN  PRINT  PAPER
4; INK 7;"O";: GO TO 1880
1870 PRINT  PAPER 4; INK 0;"+";
1880 NEXT x
1890 PRINT  INK 2;" ";y
1900 NEXT y
1910 PRINT TAB 8; INK 2;"ABCDEFG
HIJKLMNO"
1920 PRINT  INK 6;"Last move: ";
1930 LET itcp=location: GO SUB 2
260: PRINT TAB 20; INK 5;x$
1940 PRINT AT 20,20; INK 5;y$: B
EEP 1,0
1950 RETURN
1990 REM input routine
2000 LET is=0
2010 PRINT AT ip,0;: FOR i=1 TO
62: PRINT " ";: NEXT i
2020 LET a$=""
2030 PRINT AT ip,0; INK 7;m$(im)
;
2060 LET i$=INKEY$: IF i$="" THE
N  GO TO 2060
2065 IF CODE i$=13 THEN  GO TO 2
120
2070 IF CODE i$<>12 THEN  GO TO
2090
2080 IF is>0 THEN  LET is=is-1:
LET a$=a$(1 TO LEN a$-1): PRINT
CHR$ 8;" ";CHR$ 8;: GO TO 2060
2085 GO TO 2060
2090 IF i$>="a" AND i$<="z" THEN
  LET i$=CHR$ (CODE i$-32)
2100 IF is<iw THEN  LET is=is+1:
 LET a$=a$+i$: PRINT  INK 5;i$;
2110 GO TO 2060
2120 RETURN
2160 REM message routine
2190 PRINT AT mp,0;: FOR m=1 TO
30: PRINT " ";: NEXT m
2200 PRINT AT mp,0; INK 2;m$(mm,
 TO 26);o$;
2220 RETURN
2260 REM int-to-char routine
2270 IF itcp=0 THEN  PRINT  INK
5;"Handicap";: RETURN
2275 LET c$=CHR$ (itcp-16*INT (i
tcp/16)+64): LET r$=STR$ (INT (i
tcp/16))
2280 PRINT  INK 5;c$;r$;: RETURN
```

Crabes

Pour que ce programme puisse tourner sur votre micro-ordinateur, il est nécessaire que ce dernier soit conforme à la norme MSX (son, couleur, graphisme).

Vous devez maintenant aider une pauvre tortue à regagner la mer en évitant les crabes voraces qui patrouillent sur la plage... Chaque tortue amenée au but rapporte un point. Vous disposez de cinq vies pour tenter de marquer un score maximal. Utilisez les touches du curseur pour avancer et pour reculer.




```
10 REM **********
20 REM * CRABES *
30 REM **********
40 SCREEN 0,0
50 DEFINT A-Z
60 KEY OFF
70 WIDTH 39
80 GOSUB 990
90 GOSUB 1130
100 GOSUB 840
110 LOCATE 0,20,0
120 PRINT "VIE(S) REST.";NP;
130 A$=RIGHT$(A$,1)+LEFT$(A$,38)
140 B$=RIGHT$(B$,38)+LEFT$(B$,1)
150 LOCATE 0,X1,0
160 PRINT A$;
170 LOCATE 0,X2,0
180 PRINT B$;
190 LOCATE 0,X3,0
200 PRINT A$;
210 LOCATE 0,X4,0
220 PRINT B$;
230 D$=INKEY$
240 PY=PY+(STICK(0)=1)-(STICK(0)=5)
250 IF PY>10 THEN PY=10
260 IF PY=2 THEN 370
270 C=VPEEK(PX+PY*40+1)
280 IF C<>32 AND C<>128 THEN 550
290 LOCATE PX,YP,0
300 PRINT N$;
310 LOCATE PX,PY,0
320 PRINT P$;
330 YP=PY
340 T=T+1
350 IF T>500 THEN 660
360 GOTO 110
370 LOCATE PX,YP,0
380 PRINT N$;
390 LOCATE PX,PY,0
400 PRINT P$;
410 BEEP
420 FOR I=1 TO 200
430 NEXT I
440 LOCATE PX,PY,0
450 PRINT N$;
460 PY=10
470 YP=PY
480 S=S+1
490 LOCATE 0,0,0
500 PRINT "SCORE :";S;
510 LOCATE 19,0
520 PRINT "RECORD :";R;
530 GOSUB 1040
540 GOTO 110
550 NP=NP-1
560 LOCATE PX,YP,0
570 PRINT N$;
580 LOCATE PX,PY,0
590 PRINT CHR$(128);
600 GOSUB 1090
610 IF NP=0 THEN 660
620 PY=10
630 YP=PY
640 GOSUB 1040
650 GOTO 110
660 CLS
670 IF S>R THEN R=S
680 IF T<500 THEN 710
690 LOCATE 10,8,0
700 PRINT "** TEMPS ECOULE **";
710 LOCATE 10,12,0
720 PRINT "SCORE :";S;
730 LOCATE 10,16,0
740 PRINT "RECORD :";R;
750 LOCATE 10,20
760 PRINT "UNE AUTRE ?";
770 IF INKEY$<>"" THEN 770
780 D$=INKEY$
790 IF D$="" THEN 780
800 IF D$<>"N" AND D$<>"n" THEN 100
810 CLS
820 LOCATE 0,0,1
830 END
840 CLS
850 COLOR 1,11
860 P$=CHR$(128)
870 N$=CHR$(32)
880 S=0
890 NP=5
900 PX=19
910 PY=10
920 YP=PY
930 X1=4
940 X2=5
950 X3=7
960 X4=8
970 T=0
980 RETURN
990 FOR I=1 TO 39
1000 READ A
1010 A$=A$+CHR$(A)
1020 NEXT I
1030 B$=A$
1040 X=RND(1)*35+2
1050 A$=RIGHT$(A$,X)+LEFT$(A$,39-X)
1060 RETURN
1070 DATA 32,129,130,32,32,129,130,32,32
,32,129,130,32,32,32,32,32,129,130,32
1080 DATA 32,32,129,130,32,32,32,129,130
,32,32,129,130,32,32,129,130,32,32
1090 PLAY "T100O3G1604C2O3G8O4C16E2T150C
8E16G8F+16F8D+16E8C16O3A8G16O4C4C32E16
1100 FOR I=1 TO 4000
1110 NEXT I
1120 RETURN
1130 FOR I=0 TO 23
1140 READ A
1150 VPOKE 3072+I,A*4
1160 NEXT I
1170 RETURN
1180 DATA 12,45,63,30,30,63,45,0
1190 DATA 7,15,31,31,18,16,12,0
1200 DATA 56,60,62,62,18,2,12,0
```

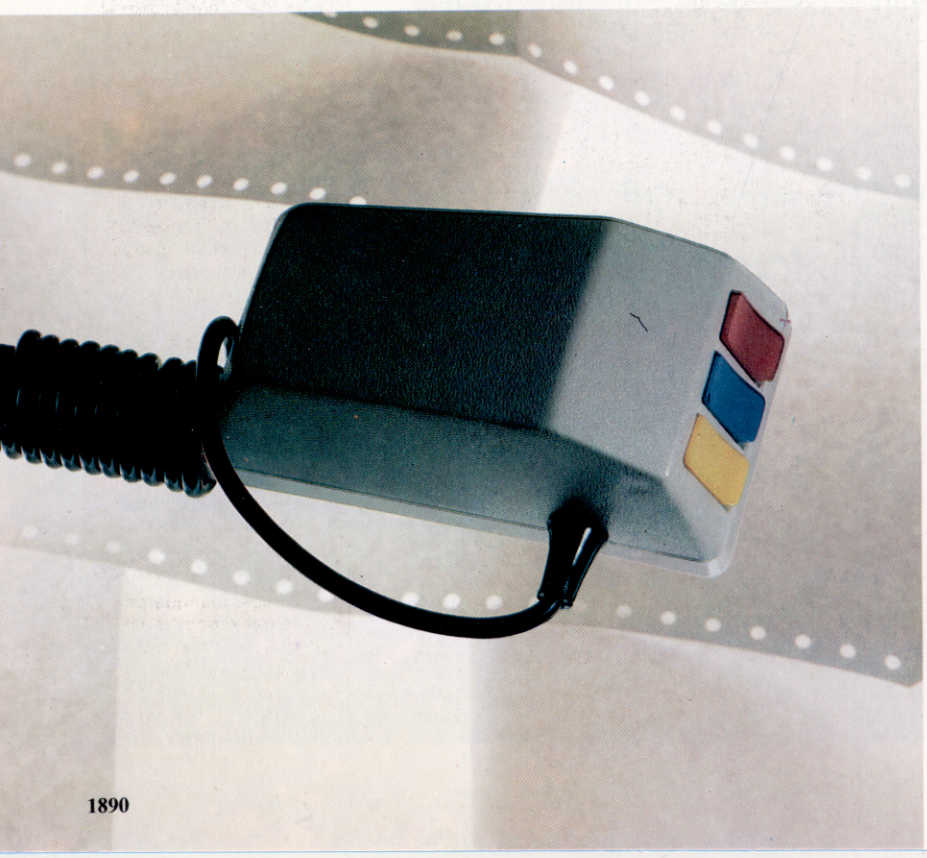
Petites souris

Depuis l'arrivée du Macintosh sur le marché, et de sa souris, cette dernière fait des petits. La SMC Supplies Magic Mouse est la première souris destinée au C64.

Lorsque la firme américaine Apple mit en vente son nouvel ordinateur Lisa, doté d'un périphérique inattendu, la « souris », les journalistes lui firent un accueil mitigé. La souris permettait de faciliter l'approche de l'ordinateur à l'utilisateur non averti. Toutefois, si dans les premiers temps la souris se révèle un auxiliaire précieux du néophyte, la pratique croissante de l'ordinateur amène rapidement à discerner ses limites, en particulier un certain manque de précision.

Étant donné le succès actuel des systèmes d'exploitation organisés autour de la « souris », il peut sembler difficile d'imaginer les polémiques que suscita cette interface au sein de l'industrie micro-informatique : était-elle acceptable, voire souhaitable? L'échec commercial de Lisa d'Apple, premier ordinateur doté de ce système, semblait confirmer que l'utilisateur final ne lui était pas favorable.

Le succès ultérieur du Macintosh devait cependant réduire les critiques et confirmer l'innovation d'Apple. Les attitudes envers la souris changèrent complètement; les fabricants incorporent maintenant systématiquement ce périphérique, et de nombreux fournisseurs mettent sur le marché de nouvelles interfaces graphiques pour des machines existantes.



Magie !
SMC Magic Mouse est l'un des premiers systèmes d'interface graphique « souris » développés pour le C64. Bien qu'il ne s'agisse pas à proprement parler d'un logiciel d'environnement graphique (avec fenêtres, icônes, souris et programmation), c'est certainement un outil précieux de développement pour les programmeurs. La souris se connecte au port manettes de jeu du Commodore. Les icônes à l'écran sont obtenues au moyen des trois boutons situés sur la souris. (Cl. Crispin Thomas.)

Magic Mouse de SMC Supplies est le premier produit de cette sorte disponible pour le C64. Un logiciel est livré avec la souris, sur cassette et sur disque. Il comprend quatre programmes d'application. La souris SMC est plus grosse que la plupart de ses congénères (125 × 66 × 50 mm), le double de celle d'Apple. Elle comporte sur sa face antérieure trois boutons de couleur, contre généralement deux. Ils servent à sélectionner des fonctions selon les applications retenues. Une balle de caoutchouc dur, à l'intérieur, frotte contre deux encodeurs de position angulaire.

Au départ, il s'agissait de faire une boule d'acier, mais cela se révéla irréalisable et le lancement du système fut reporté pour permettre la substitution par une boule de caoutchouc. La souris n'en fonctionne pas moins à la perfection.

Programmes utilitaires

Le progiciel de Magic Mouse n'est pas un système d'exploitation comme celui du Macintosh d'Apple. C'est plutôt un périphérique graphique. Les quatre programmes fournis sont : Hi-Res Designer (Mode haute résolution), Sprite Designer (Figures graphiques), Icon Designer (Icônes) et Mouse Controller (Contrôleur de la souris).

Pour utiliser le logiciel, il faut charger le fichier système et calibrer le curseur de la souris. Le système résidant sur disque est automatiquement chargé avec un menu principal désignant les quatre programmes utilitaires. La version cassette est légèrement différente, le fichier système faisant partie de Hi-Res Designer. Aussi, ce programme doit être chargé préalablement à l'utilisation des trois autres utilitaires. En outre, le curseur doit être positionné dans le coin inférieur droit avant démarrage.

Hi-Res Designer est un programme de type « palette de couleurs » très semblable à celui de Koala Pad. Une fois chargé, l'écran se divise en cases ayant chacune une option : Palette, Dessin ou Remplir. En déplaçant à l'aide de la souris le curseur sur une option, et en appuyant sur le bouton rouge de la souris (en validant), on retient l'option. De même, pour choisir la couleur de fond ou de premier plan, il suffit de déplacer le curseur en regard d'une des 16 lignes indiquant une couleur. Pour passer au fond, vous appuyez sur le bouton bleu.

Les modules Icon et Sprite sont très semblables et ne diffèrent que par leurs applications. Comme la plupart des programmes de cette nature, le module de conception de figures gra-

phiques Sprite Designer vous propose une grille de 24 colonnes par 21 lignes. Un pixel est retenu en appuyant sur le bouton rouge lorsque le curseur recouvre la position voulue. Le lutin apparaît dans une fenêtre située à l'angle supérieur droit de l'écran. Quand le curseur quitte la grille, un certain nombre d'options s'affichent dans le coin inférieur droit. Elles sont destinées à des applications telles qu'agrandir le lutin, en changer la couleur ou passer à une autre figure.

Le module de création d'icône se présente de manière presque identique, avec une grille à l'angle et l'icône dans une fenêtre en haut de l'écran. Sa spécificité tient au fait qu'il génère du graphisme utilisateur (jeux de caractères) susceptible d'être sauvegardé et réutilisé.

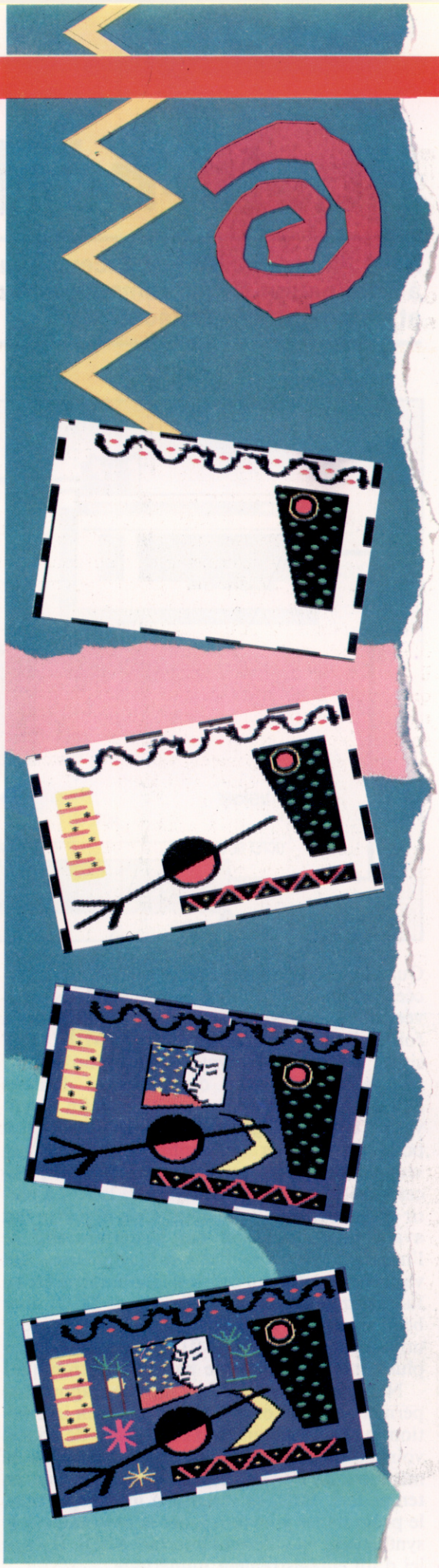
Le dernier programme de Magic Mouse est le contrôleur de la souris, Mouse Controller. Il ne fait pas *tout* à lui tout seul. Il fournit le programme pilote qui permet l'exploitation de l'ensemble. Bien qu'il soit à première vue le moins utile, ce module est en fait le plus riche, dans la mesure où il vous encourage à écrire vos propres logiciels pour la souris.

L'avantage essentiel des systèmes à souris tient à la précision du positionnement du curseur qui favorise le placement des lignes et l'obtention des couleurs désirées. Un chiffreur est constitué d'un ensemble de nœuds interconnectés, mais sa résolution est inférieure à celle de l'écran. Aussi, lorsque le crayon optique est situé entre deux positions, l'ordinateur ne peut détecter la position voulue et donne un résultat décalé. Comme on s'en doute, cela peut avoir des conséquences désastreuses pour le graphisme en cours. Avec une souris, le curseur ne réagit qu'aux déplacements très sensibles de la balle. Il reste donc bien en place tant que la souris n'est pas déplacée.

Le système n'est cependant pas parfait. Certaines caractéristiques de Hi-Res manquent de convivialité. Par exemple, la commande RUB ne permet la suppression d'erreurs que pixel par pixel. Cela prend un temps considérable et peut, en outre, introduire d'autres erreurs si l'on se trompe de pixel. Une meilleure méthode consiste à supprimer tous les ajouts graphiques depuis le dernier recours au menu. Cela suppose de perdre du travail correct, mais vous serez au moins sûr de reprendre sur de bonnes bases.

Le manuel de Magic Mouse est un excellent guide utilisateur. Chaque programme comporte une explication complète du logiciel de développement, des informations sur la manière d'incorporer le module obtenu dans votre propre logiciel, et donne des programmes types. Cette démarche témoigne de la part de SMC d'une grande ouverture d'esprit et doit être mentionnée.

Bien que Magic Mouse ne puisse rivaliser avec le Macintosh Emulator, il n'en constitue pas moins un outil de développement très appréciable, semblable à la souris AMX ou au logiciel DR GEM. Son optique est certainement d'encourager le développement de logiciels utilisateur. Cette approche n'est pas nécessairement commerciale, mais devrait assurer une longue vie au Magic Mouse.



Développement artistique
Ces images ont été obtenues en utilisant le module d'application haute résolution Hi-Res Designer avec la souris SMC Magic Mouse. Le programme permet les commandes habituelles pour ce genre de logiciel, dont LIGNE, TRACER, REMPLIR et CERCLE. Jusqu'à seize couleurs sont disponibles, ainsi que plusieurs modes de coloriage qui font varier l'épaisseur et la texture du trait. (Cl. John Clementson/Images-écran de Dimension Graphics.)

MAGIC MOUSE

INTERFACE

Magic Mouse se connecte au port manettes de jeux du Commodore 64.

LOGICIEL

La souris est livrée avec quatre logiciels d'applications, sur disque ou sur cassette.

DOCUMENTATION

Le manuel donne tous les détails utiles sur l'installation de la souris, l'utilisation des logiciels existants et l'incorporation du module résultant dans le programme utilisateur.

AVANTAGES

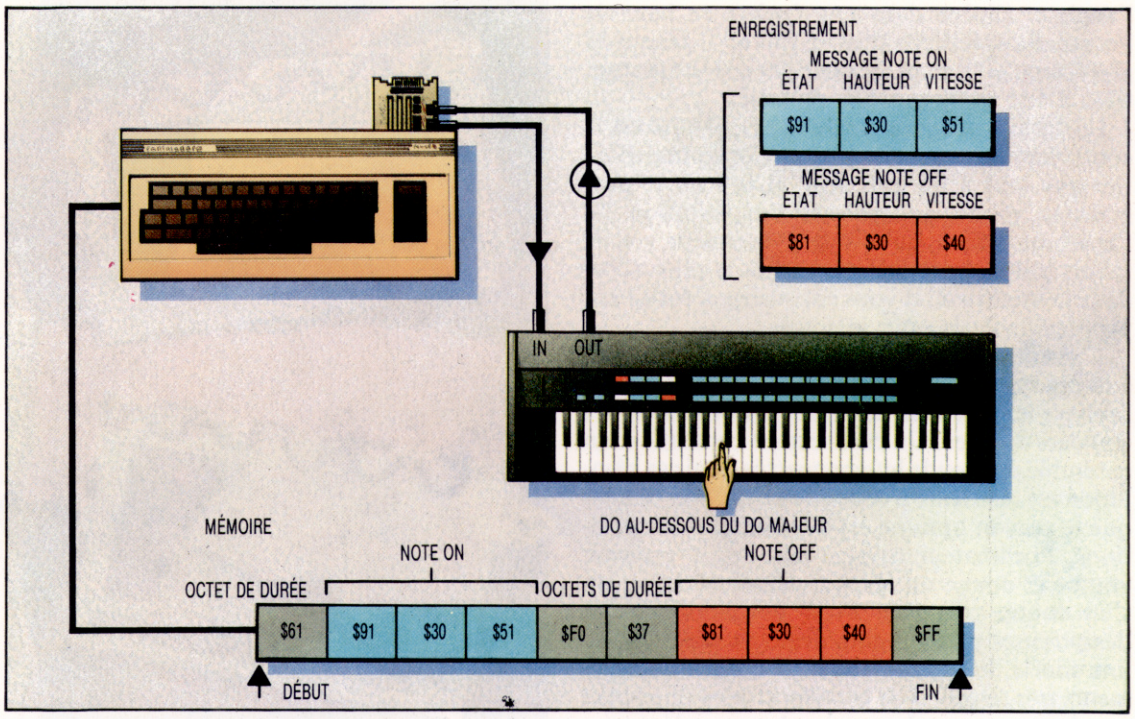
La souris semble robuste et fiable. Les utilisateurs devraient facilement la mettre à profit pour transférer les lutins et leur propre graphisme.

INCONVÉNIENTS

L'ensemble manque de portée par rapport à de nombreux programmes orientés « interface-graphique », et risque d'être très vite dépassé.

Haute-fidélité

Avec l'interface MIDI, concevons un programme permettant à l'ordinateur d'agir comme unité d'enregistrement numérique et comme unité d'exécution.



La voix de son maître Notre schéma montre comment une note est enregistrée à l'aide du programme « Enregistrement numérique ». Pendant la phase d'enregistrement, le synthétiseur envoie un message *note on* quand la touche est appuyée et un message *note off* lorsqu'elle est relâchée. Ces messages sont stockés en mémoire, avec l'information de synchronisation qui enregistre combien d'intervalles de 2 ms se sont écoulés entre les événements. L'information de synchronisation entre les messages *note on* et *note off* est contenue dans deux octets dans notre exemple, pour illustrer comment est traité le dépassement en capacité. Pendant l'exécution, le programme utilise l'information de synchronisation pour retarder la transmission des messages *note* afin de reproduire le même motif musical.

Quand les touches d'un clavier MIDI sont enfoncées, ou quand une unité de commande est activée, ces « événements » sont transmis sous la forme d'octets de données à la prise MIDI OUT située à l'arrière du synthétiseur. L'ordinateur peut servir également de source de données MIDI, transmises à un synthétiseur afin d'activer ses circuits de génération sonore. Il est ainsi possible d'utiliser un ordinateur comme un séquenceur, ce qui permet d'entrer des séquences musicales et de les éditer une à la fois à partir du clavier de l'ordinateur. Cela doit être fait avant la transmission par l'intermédiaire de l'interface MIDI afin d'activer le synthétiseur qui va jouer le morceau musical. Cette tâche est en fait assez simple; les possibilités de MIDI vont beaucoup plus loin. Imaginez un séquenceur utilisant les 16 canaux MIDI pour commander plusieurs claviers et une batterie électronique!

Nous pouvons combiner diverses approches permettant à un ordinateur d'entrer en interaction avec des instruments MIDI afin de concevoir un logiciel pouvant enregistrer dans la mémoire de l'ordinateur un morceau joué en temps réel. Puis, en réponse à une instruction, le programme pourra rejouer le morceau sur le synthétiseur en utilisant l'interface MIDI. Il s'agit ici d'un enregistrement numérique.

Bien qu'il semble similaire à un enregistrement analogique sur bande magnétique, l'enregistrement numérique par l'intermédiaire de MIDI est vraiment différent. Lorsqu'une copie est effectuée sur bande magnétique, les sons musicaux sont codés selon des combinaisons magnétiques sur la surface de la bande, et la position de ces combinaisons détermine l'ordre dans lequel seront reproduits ces sons pendant l'exécution, ainsi que les intervalles entre chacun d'entre eux. Prenons deux notes enregistrées de cette manière et séparées par un intervalle de deux secondes. Pendant cet intervalle, la bande continue à défiler devant la tête de lecture bien que rien ne soit enregistré. Lors de l'exécution, la longueur de la section de bande à blanc détermine l'intervalle entre la production de la première note et celle de la seconde.

Dans un système d'enregistrement numérique qui utilise MIDI, les deux notes correspondent aux deux événements « frappe de touche » et chacun de ces événements génère un message MIDI. L'enregistrement en temps réel pose la difficulté d'enregistrer l'intervalle de temps compris entre la réception du premier message MIDI et celle du second. Par analogie avec l'enregistrement sur bande, la meilleure façon de le faire pourrait consister à stocker les messages dans un tableau

mémoire, tout en stockant les blancs correspondant à chaque unité de temps de l'intervalle. Ainsi, lors de l'exécution, les deux notes pourraient être jouées en respectant le bon intervalle. Il est évident que cette méthode d'enregistrement serait assez gourmande en mémoire.

Il existe une autre solution. Au lieu de stocker les valeurs de chaque blanc, il serait préférable de stocker une information d'intervalle sous la forme d'un seul octet. Par exemple, si les deux notes sont séparées par un intervalle de 50 unités de temps, un octet contenant la valeur d'intervalle 50 serait entré dans le tableau mémoire au lieu des 50 octets blancs.

Cela permet une économie substantielle de mémoire, mais il est difficile de prévoir la longueur maximale pouvant être stockée dans une mémoire donnée. Les morceaux impliquant une exécution très rapide nécessitent plus d'octets pour enregistrer le grand nombre de messages *note on/off* transmis par le synthétiseur qu'un morceau lent de la même longueur.

La mise en œuvre de la roue de variation de registre tend à utiliser de grandes quantités de mémoire en raison du nombre important de messages MIDI qu'elle génère.

Programme d'enregistrement numérique

Dans le programme décrit ici, nous avons besoin d'un compteur pour enregistrer les intervalles de temps compris entre les divers événements. Le compteur que nous utilisons est une variable nommée CLOCKS. La vitesse d'incrémentation du compteur définit la précision avec laquelle la synchronisation peut être déterminée, et par conséquent la fidélité de l'enregistrement.

Un compromis doit être trouvé lors de la sélection de l'intervalle de temps optimal — nous devons choisir un intervalle d'environ 2 ms. Cet intervalle est à la fois plus long que les temps d'exécution des routines et assez court pour ne pas affecter la précision de l'enregistrement. Il est nommé *résolution* de l'enregistrement, puisqu'il est l'intervalle de temps le plus court pouvant être enregistré. L'intervalle est généré par un chronomètre de l'une des puces CIA ou VIA du micro.

Le format du message MIDI doit être légèrement étendu pour tenir compte de la synchronisation. Pour cela, nous plaçons devant chaque message MIDI un seul octet représentant le nombre d'intervalles de temps qui se sont écoulés depuis le dernier message. Cet octet peut enregistrer jusqu'à 239 unités de temps et ses valeurs peuvent être comprises entre $0 et $EF.

Si 240 intervalles s'écoulent sans avoir reçu de message, un message de dépassement de capacité est enregistré. C'est l'octet $F0. Finalement, un message à un octet, $FF, a été choisi pour marquer la fin d'une séquence. Les valeurs d'octet $F1 à $FE sont réservées pour vos propres marqueurs de programme.

Enregistrement numérique

BASIC Loader

```
10 FOR I = 49152 TO 49581
20 READ X:POKE I,X:S=S+X:NEXT
30 READ X:IF S=X THEN PRINT"OK SYS49152 TO RUN":END
40 PRINT"CHECKSUM ERROR"
50 END
100 DATA76,7,192,2,83,201,241,169,3
110 DATA141,0,222,169,21,141,0,222,169
120 DATA255,141,173,193,169,0,141,6
130 DATA220,169,8,141,7,220,169,17,141
140 DATA15,220,32,228,255,201,0,240
150 DATA249,201,69,208,1,96,201,82,240
160 DATA4,201,80,208,236,72,120,169
170 DATA127,141,0,220,169,173,133,247
180 DATA169,193,133,248,169,173,141,5
190 DATA192,169,241,141,6,192,160,0
200 DATA104,201,82,8,240,66,169,2,32
210 DATA69,193,32,44,193,201,255,208,4
220 DATA40,76,6,193,141,4,192,201,240
230 DATA173,4,192,240,12,32,246,192
240 DATA240,251,206,4,192,208,246,176
250 DATA223,32,44,193,72,173,0,222,41
260 DATA2,240,249,104,141,1,222,16,3
270 DATA32,53,193,202,208,233,174,3
280 DATA192,16,195,169,1,32,69,193,140
290 DATA4,192,162,255,32,246,192,240
300 DATA16,238,4,192,173,4,192,201,240
310 DATA144,6,32,21,193,140,4,192,173
320 DATA0,222,41,1,208,6,224,1,48,224
330 DATA16,243,173,1,222,48,11,224,0
340 DATA48,213,208,27,174,3,192,16,11
350 DATA201,248,176,223,201,240,174
360 DATA196,32,53,193,72,173,4,192,32
370 DATA21,193,140,4,192,104,32,21,193
380 DATA202,240,178,208,197,173,1,220
390 DATA73,239,208,18,104,104,40,208,4
400 DATA169,255,145,247,88,169,0,32,69
410 DATA193,76,37,192,173,13,220,41,2
420 DATA96,145,247,238,5,192,208,18
430 DATA238,6,192,208,13,104,104,104
440 DATA169,3,32,69,193,76,6,193,177
450 DATA247,230,247,208,2,230,248,96
460 DATA162,2,201,192,144,5,201,224
470 DATA176,1,202,142,3,192,232,96,10
480 DATA170,189,96,193,133,249,189,97
490 DATA193,133,250,160,0,177,249,240
500 DATA6,32,210,255,200,208,246,160,0
510 DATA96,104,193,134,193,146,193,158
520 DATA193,82,32,61,32,82,69,67,79,82
530 DATA68,44,80,32,61,32,80,76,65,89
540 DATA44,69,32,61,32,69,88,73,84,13
550 DATA0,147,82,69,67,79,82,68,73,78
560 DATA71,13,0,147,80,76,65,89,66,65
570 DATA67,75,32,13,0,79,85,84,32,79
580 DATA70,32,77,69,77,79,82,89,13,0
590 DATA240
600 DATA52178:REM*CHECKSUM*
```

Assembly Listing

```
*=$C000
GETIN  = $FFE4          ;INPUT A CHARACTER
       CHROUT   = $FFD2 ;OUTPUT A CHARACTER
       JMP START
       NOBYTS *=*+1     ;NO. BYTES IN CURRENT MESSAGE
       CLOCKS  *=*+1    ;NO 2MS PERIODS SINCE LAST MESSA

       FREMEM *=*+2     ;- NO. OF FREE BYTES
       DATREG  = $DE01  ;TRANSMIT/RECEIVE REG ON ACIA
       STREG   = $DE00  ;STATUS/CONTROL REG FOR ACIA
       STOPKY = $EF
       MEM     = $F7    ;DATA MEMORY POINTER
       PTR     = $F9    ;STRING READ POINTER
       START   = *
;++++ SET UP ROUTINES ++++
       LDA #$03
       STA STREG        ;INITIALISE 6850
       LDA #$16
       STA STREG
       G10     LDA #$FF
```

Fonctionnement du programme

Le fonctionnement du programme « Enregistrement numérique » pour le Commodore 64 est assez simple. La sélection de l'option « Record » enregistre les événements se produisant sur le port MIDI. Elle cesse lorsque la mémoire est épuisée ou quand on appuie sur la barre d'espacement. L'option « Playback » reproduit sur MIDI OUT la séquence enregistrée. L'exécution se poursuit jusqu'à la fin de la séquence ou jusqu'à ce qu'on appuie sur la barre d'espacement. Tout enregistrement écrase l'enregistrement précédent. (Cl. Marcus Wilson-Smith.)

```
STA     LOWMEM          ;INIT TO EMPTY
        LDA #$00
        STA $DC06
        LDA #$08
        STA $DC07
        LDA #$11
        STA $DC0F       ;START TIMER B
G20     JSR GETIN
        CMP #$00
        BEQ G20
        CMP #69         ;PRESS 'E' TO EXIT
        BNE G22
        RTS
G22     = *             ;NOT EXIT
        CMP #82
        BEQ G30
        CMP #80
        BNE G20
                        ;++++ REAL TIME PROG STARTS HERE
G30     PHA             ;SAVE RECORD/PB COMMAND
        SEI
        LDA #$7F
        STA $DC00       ;SCAN LAST ROW OF KEYS
        LDA #<LOWMEM
        STA MEM
        LDA #>LOWMEM
        STA MEM+1
        LDA #<FRBYTS
        STA FREMEM
        LDA #>FRBYTS
STA     FREMEM+1
        LDY #$00
        PLA
        CMP #82
        PHP             ;SAVE Z FLAG = RECORD
        BEQ R00
                        ;
        P00 = *         ;PLAY ROUTINE
                        ;
        LDA #$02
        JSR STROUT
        P05 JSR READ
        CMP #$FF
        BNE P07         ;EXIT IF END REACHED
        PLP             ;ADJUST STACK
        JMP C20         ;EXIT PROCESS
        P07 STA CLOCKS
        CMP #$F0        ;CARRY SET = $F0 READ
        LDA CLOCKS
        BEQ P30         ;JMP IF NO TIME TO WAIT
        P10 JSR CHECK
        BEQ P10
        DEC CLOCKS
        BNE P10         ;JMP IF NOT TIMED OUT
        BCS P05         ;READ NEXT CLOCK IF $F0
                        ;
        P30 = *         ;READ NEXT MESSAGE BYTE
JSR     READ
PHA                     ;SAVE DATA BYTE
;
P35     = *             ;WAIT FOR TX REGISTER
LDA     STREG
AND     #$02
BEQ     P35             ;JMP IF STILL FULL
;
PLA                     ;GET DATA
STA     DATREG          ;SEND TO MIDI
BPL     P50             ;JMP IF NOT STATUS BYTE
JSR     GETNO           ;GET NO OF DATA BYTES
P50     DEX             ;DEC. BYTE COUNT
BNE     P30             ;GET NEXT DATA BYTE
LDX     NOBYTS
BPL     P05             ;GET CLOCK BYTE
;
R00     = *             ;RECORD ROUTINE
;
LDA     #$01
JSR     STROUT
STY     CLOCKS          ;CLOCKS = 0
R05     LDX #$FF        ;INIT TO NO DATA
R10     JSR CHECK
BEQ     R20
INC     CLOCKS
LDA     CLOCKS
CMP     #$F0
BCC     R20
JSR     STORE
STY     CLOCKS
;
R20     = *             ;CHECK FOR MIDI DATA
LDA     STREG
AND     #$01
BNE     R40             ;JMP IF DATA
CPX     #$01
BMI     R10             ;JMP IF WAITING
BPL     R20             ;JMP IF MID-MESSAGE
;
R40     = *             ;GET THE DATA
LDA     DATREG
BMI     R50             ;JMP IF STATUS BYTE
CPX     #$00
BMI     R10             ;JMP IF NOT CHANNEL DATA
BNE     R80             ;JMP IF MID-MESSAGE
LDX     NOBYTS          ;RESET BYTE COUNT
BPL     R60
;
R50     = *             ;STATUS BYTE PROCESS
CMP     #$F8
BCS     R20             ;IGNORE IF SYS REALTIME
CMP     #$F0
BCS     R05             ;JMP IF NOT CHANNEL STAT.
JSR     GETNO           ;GET NO OF DATA BYTES
;
```

```
R60     = *             ;RECORD THE DATA
PHA
LDA     CLOCKS
JSR     STORE
STY     CLOCKS          ;RESET COUNTER
PLA
;
R80     = *             ;RECORD MIDI DATA
JSR     STORE
DEX
BEQ     R10
BNE     R20
;
CHECK   = *
;
LDA     $DC01
EOR     #STOPKY         ;DOESN'T AFFECT CARRY
BNE     C40
PLA
PLA                     ;ADJUST STACK
PLP                     ;CHECK FOR REC/PB
BNE     C20             ;JMP IF PLAYBACK
LDA     #$FF
STA     (MEM),Y
C20     = *
CLI
LDA     #$00
JSR     STROUT
JMP     G20             ;WAIT FOR NEXT COMMAND
;
C40     = *             ;CHECK FOR TIMER B
LDA     $DC0D
AND     #$02            ;Z FLAG = NOT TIMED OUT
RTS
;
STORE   = *             ;STORE DATA IN MEMORY
;
STA     (MEM),Y
INC     FREMEM          ;CHECK MEMORY FREE
BNE     POINT
INC     FREMEM+1
BNE     POINT
PLA
PLA
PLA
LDA     #$03            ;ERROR MESSAGE
JSR     STROUT
JMP     C20             ;RESET PROGRAM
;
READ    = *             ;READ DATA FROM MEMORY
;
LDA     (MEM),Y
;
POINT   = *             ;UPDATE MEMORY POINTER
;
INC     MEM
BNE     P20
INC     MEM+1
P20     RTS
;
GETNO   = *             ;GET NO OF DATA BYTES
;           FOR STATUS IN ACC.
;
LDX     #$02            ;INIT TO 2 DATA BYTES
CMP     #$C0
BCC     N10
CMP     #$E0
BCS     N10
DEX                     ;1 DATA BYTE IF C0<=A<E0
N10     STX NOBYTS
INX                     ;ADD 1 FOR STATUS
RTS
;
STROUT  = *             ;PRINT STRING
;
ASL     A
TAX
LDA     MESTAB,X
STA     PTR
LDA     MESTAB+1,X
STA     PTR+1
LDY     #$00
M10     LDA (PTR),Y
BEQ     M70
JSR     CHROUT
INY
BNE     M10
M70     LDY #$00
RTS
;
;
MESTAB  = *             ;MESSAGE TABLE
;
.WORD   MESS0
.WORD   MESS1
.WORD   MESS2
.WORD   MESS3
;
MESS0   .BYTE 'R = RECORD,P = PLAY,E = EXIT',$0D,0

MESS1   .BYTE 147,'RECORDING',$0D,0


MESS2   .BYTE 147,'PLAYBACK',$0D,0


MESS3   .BYTE 'OUT OF MEMORY',$0D,0


LOWMEM  = *             ;START OF DATA MEMORY
FRBYTS  = LOWMEM - $D000
```

Liste active

Voyons pourquoi il peut être utile de savoir comment le lisp représente des structures de données et illustrons par quelques exemples diverses fonctions.

Pour disposer d'une notation cohérente qui puisse facilement traiter de manière identique des identificateurs, des caractères et des listes, LISP utilise systématiquement des pointeurs. Ils peuvent être facilement simulés dans tout langage, y compris en BASIC. Par exemple, si vous donnez l'instruction suivante :

```
POKE100,55:POKE99,100 (pour la plupart des micros)
```

ou :

```
?100 = 55 :?99 = 100 (BBC Micro/Electron)
```

les positions 100 et 99 comporteront les valeurs assignées. On peut considérer la valeur située à la position 99 comme un pointeur sur la position 100. Aussi, si nous donnons la commande :

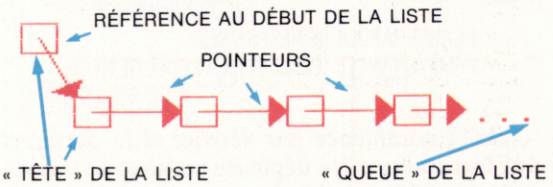
```
PRINT PEEK (PEEK (99))
```

ou

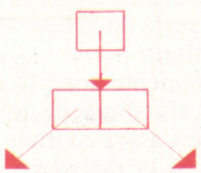
```
PRINT ?(?99)
```

nous obtenons la valeur 55. Remarquez que, pour certains micro-ordinateurs, les positions 99 et 100 peuvent résider en ROM, et ne pourront donc pas changer.

Il est bien sûr possible de créer des pointeurs vers toute position de la mémoire. Mais, si la position visée dépasse 255, il faudra utiliser deux octets afin d'être en mesure de couvrir la fourchette allant de 0 à 65535. Si nous supposons maintenant que la valeur 55 est à son tour un pointeur d'une autre position, et ainsi de suite, nous obtenons la structure de liste indiquée.

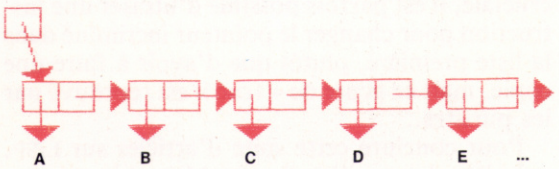


C'est malheureusement de peu d'utilité dans la mesure où la liste ne comporte pas encore d'informations. Pour lui en donner, on combine des positions pour qu'il y ait deux pointeurs à chaque nœud.



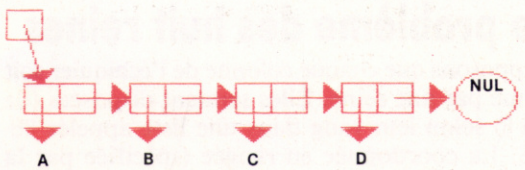
Nous pouvons maintenant représenter la liste :

```
(A B C D E...)
```



C'est parfait pour des listes infinies, mais nous devons pouvoir également les arrêter. On y parvient facilement en adressant le dernier pointeur de la liste à la liste nulle. La syntaxe NUL correspond à la liste vide. La liste finale devient donc :

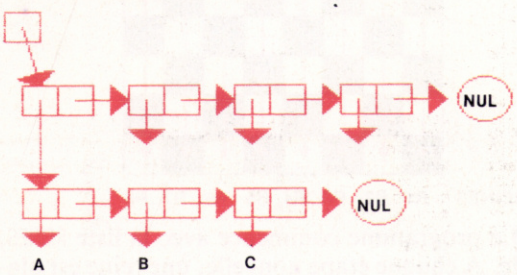
```
(A B C D)
```



Une liste qui contient une autre liste suppose un pointeur gauche vers une nouvelle liste, à la place d'un atome dans le cas précédent. Aussi la liste :

```
((A B C) D E F)
```

serait représentée de la sorte :



Le fonctionnement de CAR et de CDR doit maintenant vous être parfaitement clair. CAR, l'en-tête de l'argument, est le pointeur gauche du premier nœud de l'argument. Inversement, CDR, qui représente tout sauf l'en-tête de son argument, est simplement le pointeur droit du premier nœud. Aussi, si nous appelons la liste complète L :

```
CONS((CAR L) (CDR L))
```

donne la liste d'origine, L.

CONS est donc une méthode d'élaboration de listes. Il met ses deux arguments sous la forme

d'une nouvelle liste, et donne comme résultat le pointeur du premier élément de la nouvelle liste. Avec CONS, on peut créer toute structure de liste, même si le nombre considérable d'appels CONS peut devenir fastidieux — d'où la fonction « sténo » LIST.

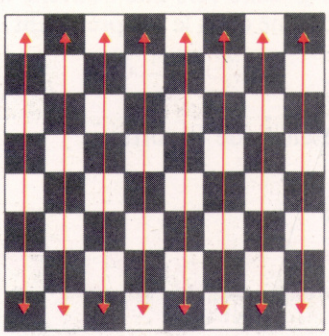
Il faut remarquer qu'il est toujours préférable d'ajouter de nouvelles informations en tête de liste en utilisant CONS. Quand on ajoute des éléments en fin de liste, il est nécessaire de copier la liste d'origine, le pointeur final de la liste première pointant sur l'élément suivant.

Certaines versions LISP disposent d'une instruction spécifique pour cela, APPEND. Mais elle est moins efficace que CONS. Si la vitesse devient cruciale, il est parfois possible d'utiliser une instruction pour changer le pointeur incriminé dans la liste première, plutôt que d'avoir à faire une copie, mais ce genre de « truc » est réprouvé par les puristes...

Pour conclure cette suite d'articles sur LISP, voici l'étude complète d'un problème bien connu. Le problème consiste à placer huit reines sur un échiquier de telle sorte qu'elles se neutralisent (qu'aucune ne puisse attaquer). *A priori,* il doit y avoir exactement une reine par case, en ligne. La difficulté est de les disposer de sorte que deux reines ne se croisent pas.

Le problème des huit reines

Supposons que chaque colonne de l'échiquier soit tenue par une reine. Elles seraient disposées (de 1 à 8) selon leur rang dans une liste appelée REINES. La coordonnée en rangée (spécifiée par la valeur dans REINES) est variable, et calculée par LISP.



REINES R1 R2 R3 R4 R5 R6 R7 R8

Le programme commence avec la liste REINES, vide. A chaque étape nouvelle, une reine est placée à la rangée 1 au début de la liste. Sa position (et celles d'autres reines peut-être) sera changée (fonction CHANGER) jusqu'à ce que toutes les reines soient en sécurité. Cette nouvelle liste devient REINES, des nouvelles étant ajoutées après le départ, et ainsi de suite.

Ce processus se poursuit jusqu'à ce que la liste comporte huit reines. La liste est alors imprimée par AF_ECHIQUIER. La fonction première, que nous appellerons RESOLUTION, s'écrit donc de la sorte :

```
(DEFON RESOLUTION ()
 (ASSIGNERQ REINES () )
 (BOUCLER (UNTIL EQ (LONGUEUR (REINES) 8))
  (ASSIGNERQ REINES (CHANGER (CONS 1 REINES))))
 (AF_ECHIQUIER '(QR QN QB Q K KB KN KR) REINES))
```

On voit que RESOLUTION, sans argument, met initialement en place la liste vide REINES. La fonction BOUCLE assigne alors REINES à la nouvelle valeur de la liste changée (CHANGER) pour la reine de rangée 1, suivie de la liste précédente. Cette BOUCLE est exécutée UNTIL (jusqu'à ce que) LONGUEUR de REINES soit EQuivalent à 8. La liste finale est ensuite affichée.

Nous venons d'utiliser trois nouvelles fonctions : LONGUEUR fournit simplement le nombre d'éléments dans son argument, et elle est définie de la sorte :

```
(DEFON LONGUEUR (L (N))
 (ASSIGNERQ N 0)
 (BOUCLE (WHILE L N)
  (ASSIGNERQ N (AJOUTER1 N))
  (ASSIGNERQ L(CDR L)) ))
```

L'argument N donné entre parenthèses est la manière propre à LISP Acornsoft d'indiquer les arguments optionnels. Ici, N n'est jamais affecté à la fonction, et joue seulement le rôle d'une variable locale.

L'autre fonction, AF_ECHIQUIER, combine les éléments de ses deux listes d'arguments pour générer un état de sortie. La fonction est définie ainsi :

```
(DEFON AF_ECHIQUIER (RANGEES COLS)
  (COND ((NULLE RANGEES) VRAI)
    (VRAI/ (AFFICHERC (CAR RANGEES) (CAR COLS))
      (AF_ECHIQUIER (CDR RANGEES) (CDR COLS)) )))
```

Les deux listes sont censées être de la même longueur, aussi la fonction s'achève lorsque la première condition est VRAIE (liste vide). Faute de quoi, le premier élément de chaque liste est affiché (avec un RC), et la fonction s'appelle elle-même pour le reste de chaque liste.

La troisième fonction, ALTER, est celle qui fait tout le travail. Elle se définit comme suit :

```
(DEFON CHANGER (REINES)
  (COND ((EQ (CAR REINES) 9)
    (CHANGER (CONS (AJOUT1 (CADR REINES))
                   (CDDR REINES) )))
    ((SAUVEGARDER REINES) REINES)
    (VRAI (CHANGER (CONS (AJOUT1 (CAR REINES))
                         (CDR REINES) )))))
```

CHANGER commence par vérifier si la dernière reine introduite a été déplacée en haut de l'échiquer (rangée 9). Si c'est le cas, une ou plusieurs des reines positionnées en dernier devront être déplacées.

Souvenez-vous que cette reine est partie de la rangée 1 et aura essayé toutes les rangées (selon la troisième condition de CHANGER). Cela signifie que cette reine ne peut trouver place dans l'échiquier courant. Aussi CHANGER s'appelle alors lui-même avec le reste de la liste REINES (tout sauf la reine en question).

CHANGER incrémente aussi la rangée de la dernière reine. Cela s'appelle « remonter l'algorithme » et c'est assez courant pour ce genre de programmes. Si la deuxième condition donne

VRAI, toutes les reines en place sont sauvegardées (SAFE), et la liste en est communiquée.

La seule fonction qu'il reste à définir est SAUVEGARDER, qui vérifie si toutes les reines présentes sont en position sûre. Puisque la position dans la liste définit la colonne (deux reines ne peuvent figurer à la même position dans la liste), nul besoin de vérifier si des colonnes se croisent. Il ne reste donc que les rangées et les diagonales à vérifier. Soit la fonction PASDEVISAVIS qui donne VRAI quand aucune reine ne se trouve dans le champ des reines précédentes. SAUVEGARDER se définit ainsi :

```
(DEFON SAUVEGARDER (REINES)
   (COND ((NULLE REINES) VRAI)
      (VRAI (ET (PASDEVISAVIS (CAR REINES)
                              (CDDR REINES) 1)
         (SAUVEGARDER (CDDR REINES)) ))))
```

Il peut être plus facile de définir ainsi SAUVEGARDER : « La première reine de la liste REINE n'est pas *vis-à-vis* des autres reines de la liste, et toutes les reines du reste de la liste sont *sauves.* »

Il ne reste plus qu'à définir la fonction PASDEVISAVIS, qui doit retourner la valeur VRAI s'il n'y a pas d'intersection rangées-colonnes. A savoir :

```
(DEFON PASDEVISAVIS (NOUVELLE RESTE COL)
   (COND ((NULLE RESTE) VRAI)
      (VRAI (ET (NON(EQ NOUVELLE (CAR RESTE)))
         (NON (EQ COL
            (ABS (DIFFERENCE NOUVELLE (CAR RESTE)))))
         (PASDEVISAVIS NOUVELLE (CDR RESTE) (ADD1 COL)) )))))
```

NOUVELLE représente la dernière reine mise en place, RESTE faisant allusion aux autres. S'il y a d'autres reines sur l'échiquier (NUL RESTE donne FAUX), la fonction ET donne VRAI lorsque :

1. La rangée occupée par une reine n'est pas la même que celle de la première reine venant ensuite dans la liste.

2. Les reines ne sont pas sur la même diagonale. On s'en assure en vérifiant que la différence ABSolue de rangée n'est pas égale à la différence de COLonnes. COL est le nombre de colonnes entre une reine et la suivante dans la liste (initialement 1). Nous avons défini ABS de la façon suivante :

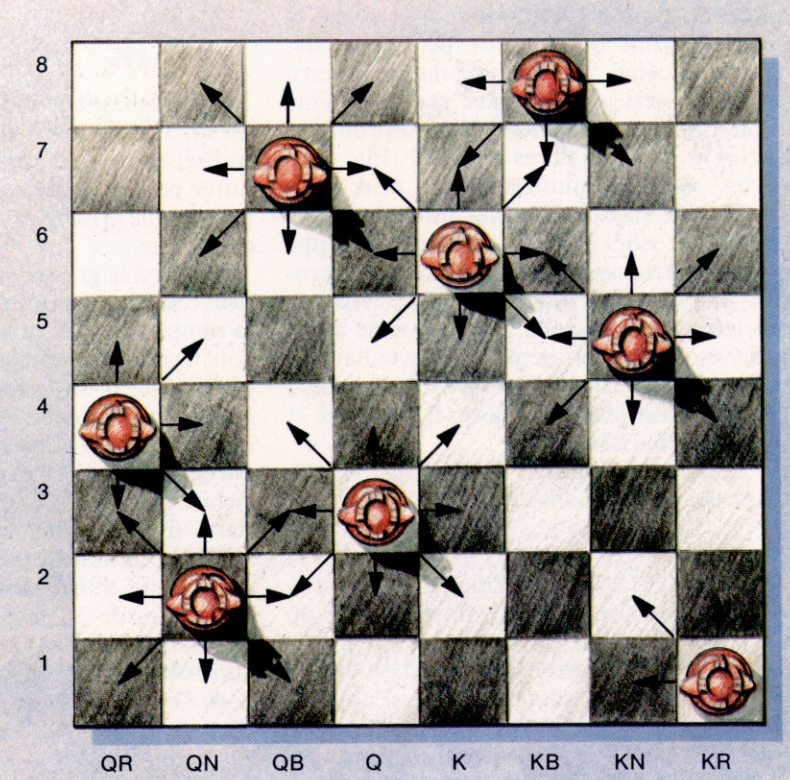
```
(DEFON ABS (NOMBRE)
   (COND ((MINUSP NOMBRE) (MINUS NOMBRE))
      (VRAI NOMBRE) ))
```

3. 1 et 2 restent vrais pour une reine et sa suivante. On le vérifie par la fonction qui s'appelle elle-même avec tous les éléments de la liste RESTE sauf le premier, et en incrémentant la différence de COLonnes.

Les définitions nécessaires sont maintenant en place. L'illustration ci-dessous montre la solution mise en œuvre.

Si vous disposez d'une version LISP, peut-être voudrez-vous essayer ces routines. Souvenez-vous qu'elles ont été écrites en utilisant Acornsoft LISP, mais elles devraient être facilement transférables sur d'autres systèmes. Remonter dans les algorithmes peut se révéler complexe, mais vous devriez y parvenir en imprimant la liste REINES dans la fonction RESOLUTION, avant chaque appel à CHANGER.

Un problème royal
Le problème des huit reines fournit une bonne illustration de résolution d'un problème par l'utilisation des performances de LISP en matière de manipulation de listes et de fonctions récursives. Avant d'étudier la réponse donnée ici, essayez de placer huit reines sur un échiquier de sorte qu'aucune ne menace les autres. Vous n'avez pas besoin de savoir jouer aux échecs, il suffit de savoir qu'une reine peut se déplacer d'un certain nombre de cases en ligne droite (diagonale, verticale et horizontale) et prendre n'importe quel nombre de pièces sur son passage.



Contrôle du Microdrive

Après avoir vu les codes crochets utilisés par l'Interface 1, considérons les fonctions concernant l'exploitation des Microdrive, un périphérique bien utile du Spectrum.

Les accès au Microdrive sont réalisés soit à l'aide des codes crochets, soit en appelant directement les routines ROM de l'Interface 1. Toutefois, cette dernière procédure n'est pas toujours recommandée, puisque plus d'une Interface 1 ROM peut être présente. Nous examinerons donc ici les treize codes crochets qui nous sont fournis pour utiliser le Microdrive.

Auparavant, voyons quelques principes sous-jacents à l'emploi des Microdrive. Après le formatage, une cartouche est divisée en 255 « secteurs », qui peuvent être considérés comme l'unité physique de stockage d'information sur une cartouche. Chacun consiste en un en-tête (donnée contenant, entre autres, le nom de la cartouche), un bloc de données, qui contient éventuellement le nom du fichier utilisant ce secteur, quelques autres données et un enregistrement de 512 octets. L'opération de formatage comporte aussi des marques pour référencer dans la cartouche des secteurs de bande qui ne sont pas utilisables.

Comme toute autre communication avec des dispositifs d'E/S dans le système Spectrum, la communication avec les Microdrive est effectuée *via* un canal. Pour les Microdrive, elle prend la forme d'une zone de mémoire de 595 octets. Associée à ce canal, il existe une mappe Microdrive de 32 octets, qui est fixée par le système d'exploitation pour dire aux différentes routines quels sont les secteurs libres à utiliser. Elle indiquera les secteurs inutilisables et ceux déjà employés pour stocker d'autres informations.

Le canal est stocké dans la zone de la mappe mémoire Spectrum appelée « zone canal Microdrive ». Chaque fois qu'un canal est fixé, la mémoire comprise entre la fin de la zone canal Microdrive et STKEND est déplacée vers le haut en mémoire pour faire de la place. De même, quand un tel canal n'est plus nécessaire, il est fermé et la mémoire rabaissée.

Les canaux Microdrive sont fixés par le système de deux manières : *explicitement,* lorsqu'un fichier est requis par une commande OPEN, ou *implicitement,* quand le système ouvre un canal pour effectuer une opération. C'est le genre de chose faite par la commande SAVE du Microdrive — le canal ouvrant est dit « implicite » parce qu'implicitement un canal est ouvert pour accomplir la commande, même si nous ne l'utilisons pas directement. Un canal ouvert explicitement peut aussi être fermé manuellement, tandis que cela se fait automatiquement pour un canal implicite.

Nous donnons un schéma montrant la structure d'un canal Microdrive. Vous remarquerez la ressemblance entre les premiers octets de ce canal et les canaux que nous avons fixés pour envoyer l'information à l'écran ou lire les données à partir du clavier.

L'adresse de mappe Microdrive est contenue dans le canal, et nous pouvons avoir une idée de l'espace disponible sur une cartouche en examinant la mappe de cette cartouche. En réalité, c'est ainsi que la fonction BASIC CAT vous fournit les données concernant l'espace restant sur une cartouche. Si vous voulez considérer une mappe Microdrive, alors essayez la routine suivante (tapez NEW d'abord) :

```
10 OPEN #5;«M»;1;«testprog»
20 start=PEEK(23870)+256*PEEK(23871)
30 FOR I=start TO start+31
40 LET sector=PEEK(I)
50 FOR J=1 TO 8
60 IF sector/2=INT(sector/2)THEN PRINT «O»;
70 IF sector/2<>INT(sector/2) THEN PRINT «1»;
80 LET sector=sector/2:LET sector=INT(sector)
90 NEXT J
100 PRINT
110 NEXT I
120 CLOSE #5
```

En exécutant ce programme, vous voyez donc qu'un 1 indique un secteur inutilisable ou déjà utilisé, et qu'un 0 indique un secteur libre. Vous aurez peut-être envie de modifier ce programme pour qu'il affiche le total d'espace libre sur la cartouche.

Quant à la grande partie de 512 octets de données dans le diagramme structure de canal, le SE la remplit avec les données à écrire. Les données sont écrites sur cartouche soit si le tampon est plein, soit si on exécute une commande CLOSE# (ou son code crochet équivalent). Dans le dernier cas, le tampon s'inscrit sur la cartouche, et cet enregistrement est marqué comme une « fin de fichier » (EOF, *end of file*). Il peut aussi être écrit dans un troisième cas en utilisant un code crochet, comme nous le verrons plus tard. Normalement, donc, la cartouche sera inscrite à intervalles de 512 octets.

Voyons à présent les codes crochets servant à contrôler le Microdrive.

- *Code crochet 33.* Il fait démarrer le moteur dans une unité de Microdrive spécifiée (processus fréquemment appelé « choix du lecteur »). Les Microdrive sont numérotés de 1 à 8. La routine peut aussi servir à arrêter tous leurs moteurs.

Un point important doit être noté ici : la routine peut retourner avec les interruptions invalidées. Si donc vous utilisez cette routine, la première chose à faire au retour est d'émettre une commande EI.

La routine est simple à utiliser. On commence par mettre dans le registre A le numéro du lecteur souhaité. Ensuite, on utilise RST #8 pour appeler le code crochet. En mettant la valeur 0 dans le registre A on éteint les lecteurs. Il est aussi possible d'éteindre les Microdrive en générant une erreur en BASIC. Cela peut se faire de deux façons : si vous essayez d'allumer un lecteur qui n'est pas connecté, ou bien un lecteur ne contenant pas de cartouche formatée.

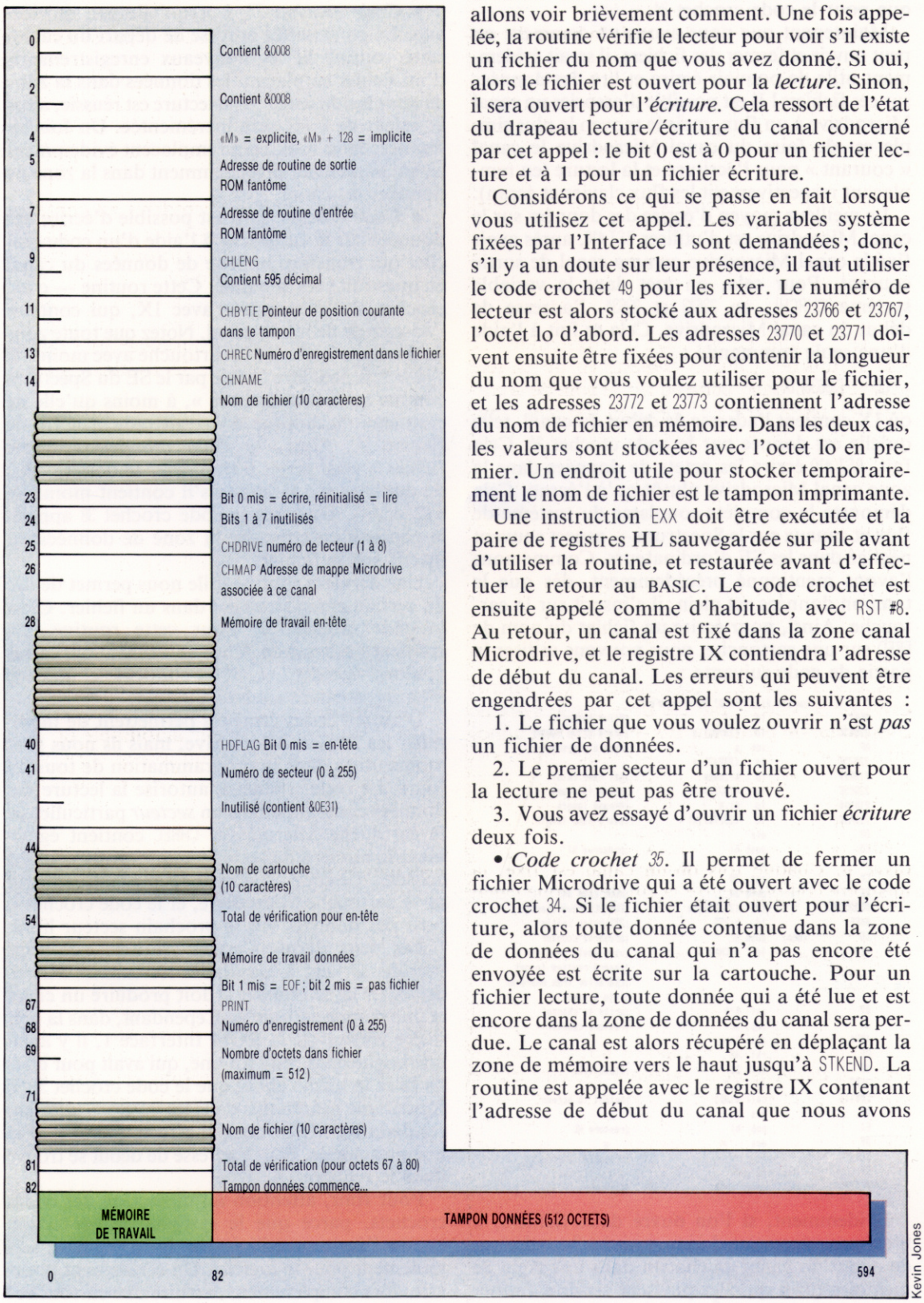
• *Code crochet 34.* Cet appel très utile permet d'ouvrir un fichier sur une cartouche. Nous allons voir brièvement comment. Une fois appelée, la routine vérifie le lecteur pour voir s'il existe un fichier du nom que vous avez donné. Si oui, alors le fichier est ouvert pour la *lecture*. Sinon, il sera ouvert pour l'*écriture*. Cela ressort de l'état du drapeau lecture/écriture du canal concerné par cet appel : le bit 0 est à 0 pour un fichier lecture et à 1 pour un fichier écriture.

Considérons ce qui se passe en fait lorsque vous utilisez cet appel. Les variables système fixées par l'Interface 1 sont demandées; donc, s'il y a un doute sur leur présence, il faut utiliser le code crochet 49 pour les fixer. Le numéro de lecteur est alors stocké aux adresses 23766 et 23767, l'octet lo d'abord. Les adresses 23770 et 23771 doivent ensuite être fixées pour contenir la longueur du nom que vous voulez utiliser pour le fichier, et les adresses 23772 et 23773 contiennent l'adresse du nom de fichier en mémoire. Dans les deux cas, les valeurs sont stockées avec l'octet lo en premier. Un endroit utile pour stocker temporairement le nom de fichier est le tampon imprimante.

Une instruction EXX doit être exécutée et la paire de registres HL sauvegardée sur pile avant d'utiliser la routine, et restaurée avant d'effectuer le retour au BASIC. Le code crochet est ensuite appelé comme d'habitude, avec RST #8. Au retour, un canal est fixé dans la zone canal Microdrive, et le registre IX contiendra l'adresse de début du canal. Les erreurs qui peuvent être engendrées par cet appel sont les suivantes :

1. Le fichier que vous voulez ouvrir n'est *pas* un fichier de données.
2. Le premier secteur d'un fichier ouvert pour la lecture ne peut pas être trouvé.
3. Vous avez essayé d'ouvrir un fichier *écriture* deux fois.

• *Code crochet 35.* Il permet de fermer un fichier Microdrive qui a été ouvert avec le code crochet 34. Si le fichier était ouvert pour l'écriture, alors toute donnée contenue dans la zone de données du canal qui n'a pas encore été envoyée est écrite sur la cartouche. Pour un fichier lecture, toute donnée qui a été lue et est encore dans la zone de données du canal sera perdue. Le canal est alors récupéré en déplaçant la zone de mémoire vers le haut jusqu'à STKEND. La routine est appelée avec le registre IX contenant l'adresse de début du canal que nous avons



Kevin Jones

La structure du canal Microdrive
Le canal Microdrive occupe 595 octets de mémoire. Son emplacement est donné dans le registre IX après ouverture d'un fichier sur cartouche à l'aide du code crochet 34.

spécifié. Il est aussi recommandé de préserver l'autre paire de registres, HL, avant usage. Une erreur est engendrée s'il n'y a pas suffisamment de place sur la cartouche pour fermer le fichier.

● *Code crochet 36.* Il nous permet d'effacer un fichier Microdrive, que ce soit un fichier de données ou un programme. Le nom du fichier, son adresse et sa longueur, ainsi que le numéro de lecteur, sont tous fixés de la même manière que pour le code crochet 34.

Maintenant que nous avons vu comment on peut ouvrir et fermer des fichiers, il serait évidemment utile de pouvoir écrire et lire des données sur ces fichiers ! Il est possible d'associer un canal Microdrive à un flux, mais le moyen le plus simple est de faire du canal Microdrive le canal « courant » pour l'écriture et la lecture (en remplaçant respectivement les flux clavier et écran).

Le meilleur moyen d'écrire des données sur le canal Microdrive est d'utiliser RST #10, après avoir fixé le canal Microdrive comme canal de sortie courant. Pour cela, on met dans la variable système CURCHL, en 23633 et 23634, l'adresse de début du canal Microdrive. Cela se fait à l'aide d'une seule commande :

```
LD    (23633),IX
```

où IX contient l'adresse de début du canal, telle qu'elle est donnée par le code crochet 34. Cela fait, RST #10 écrira les octets de données dans la zone canal Microdrive plutôt qu'à l'écran. Cela démontre la complète souplesse du système de flux et canaux sur le Spectrum — aspect souvent négligé dans les SE d'ordinateurs. Comme nous l'avons mentionné précédemment, dès que le tampon données est plein, il s'inscrit sur la cartouche. Ainsi, pour écrire un fichier du nom de FRED sur une cartouche, nous pouvons utiliser la partie de code suivante :

```
                ;write file "FRED" to microdrive
3E01            ld   a,1                ;drive number
32D65C          ld   (23766),a          ;store drive number
AF              xor  a                  ;zero a register
32D75C          ld   (23767),a
212B29          ld   hl,name            ;get name in hl
22DC5C          ld   (23772),hl         ;store filename
210400          ld   hl,4               ;4=name length
22DA5C          ld   (23770),hl         ;store name length
D9              exx
E5              push hl                 ;preserve hl'
D9              exx
CF              rst  #8
22              defb 34                 ;hook code 34
DD22515C        ld   (23633),ix         ;CURCHL=drive channel
06FF            ld   b,255              ;255 bytes infile
C5        loop: push bc                 ;preserve counter
3E00            ld   a,CHAR             ;user would need to
                                        ;insert routine to get
                                        ;character from wherever
                                        ;into a register
D7              rst  #10                ;write to channel
C1              pop  bc                 ;restore counter
10F6            djnz loop
CF              rst  #8
23              defb 35                 ;close file
3E02            ld   a,2
CD0116          call #1601              ;output to screen
D9              exx
E1              pop  hl                 ;restore HL
D9              exx
FB              ei                      ;reenable interrupts
C9              ret,
46524544  name: defm "FRED"             ;filename
```

Évidemment, si l'on ouvre deux fichiers, il n'est pas nécessaire de sauvegarder les adresses de début du canal de chacun dans une zone de mémoire (afin qu'elles puissent servir à fermer les fichiers, etc.). Notre routine génère un fichier de 255 caractères dans la cartouche.

La lecture des données d'un tel fichier est aussi immédiate ; le fichier est ouvert comme précédemment, et on met dans CURCHL l'adresse de départ du canal Microdrive. La routine Spectrum en #15E6 dans la ROM principale sert alors à lire un octet du canal. L'octet ainsi lu est mis dans le registre A.

● *Code crochet 37.* Lorsqu'elle est appelée, avec IX contenant l'adresse de départ du canal, cette routine lit les nouveaux enregistrements d'un fichier en plaçant les données dans la zone de données du canal. Si la lecture est réussie, alors la valeur de CHREC sera incrémentée. Un fois les données ainsi lues, elles remplacent évidemment ce qui se trouvait précédemment dans la zone de données du canal.

● *Code crochet 38.* Il est possible d'écrire des données sur la cartouche à l'aide d'un code crochet qui transfère la zone de données du canal en question vers la bande. Cette routine — code crochet 38 — est entrée avec IX, qui contient l'adresse de début du canal. Notez que toute zone de données écrite sur la cartouche avec moins de 512 octets peut être traitée par le SE du Spectrum comme un « secteur libre », à moins qu'elle ne contienne la marque d'enregistrement « fin de fichier ». Ainsi, le code crochet « ferme fichier » peut servir à transférer le dernier bloc de données sur la bande s'il contient moins de 512 octets. Une fois le code crochet 38 appelé, le contenu en cours de la zone de données est inscrit sur le fichier.

Une dernière routine utile nous permet de lire un certain enregistrement dans un fichier. C'est le code crochet 39. Pour cette routine, IX contient l'adresse de début du canal auquel nous voulons accéder, et CHREC contient le numéro d'enregistrement souhaité.

D'autres codes crochets permettent de manipuler les fichiers Microdrive, mais ils nous sont moins utiles dans la programmation de tous les jours. Le code crochet 40 autorise la lecture des données contenues dans un *secteur* particulier de la cartouche Microdrive. CHREC contient également le numéro de secteur auquel nous voulons accéder. Le code crochet 41 lit le secteur suivant de la cartouche Microdrive, et le code crochet 42 écrit des données sur le prochain secteur libre.

Les deux derniers codes crochets que nous verrons servent à manipuler les canaux Microdrive. Le code crochet 43 doit produire un canal et une mappe cartouche. Cependant, dans la première version de la ROM Interface 1, il y avait une bogue dans cette routine, qui avait pour effet de faire la même chose que le code crochet 34. Il fonctionne néanmoins dans les versions ultérieures de la ROM. Le code 44 efface un canal et la mappe associée, dont l'adresse de début se trouve dans le registre IX.

Pour finir, attention à l'utilisation des codes crochets, parce que le système de protection contre l'écriture utilisée par le Microdrive sert seulement pour le *logiciel*. Un écrasement pourrait alors complètement détruire une cartouche !