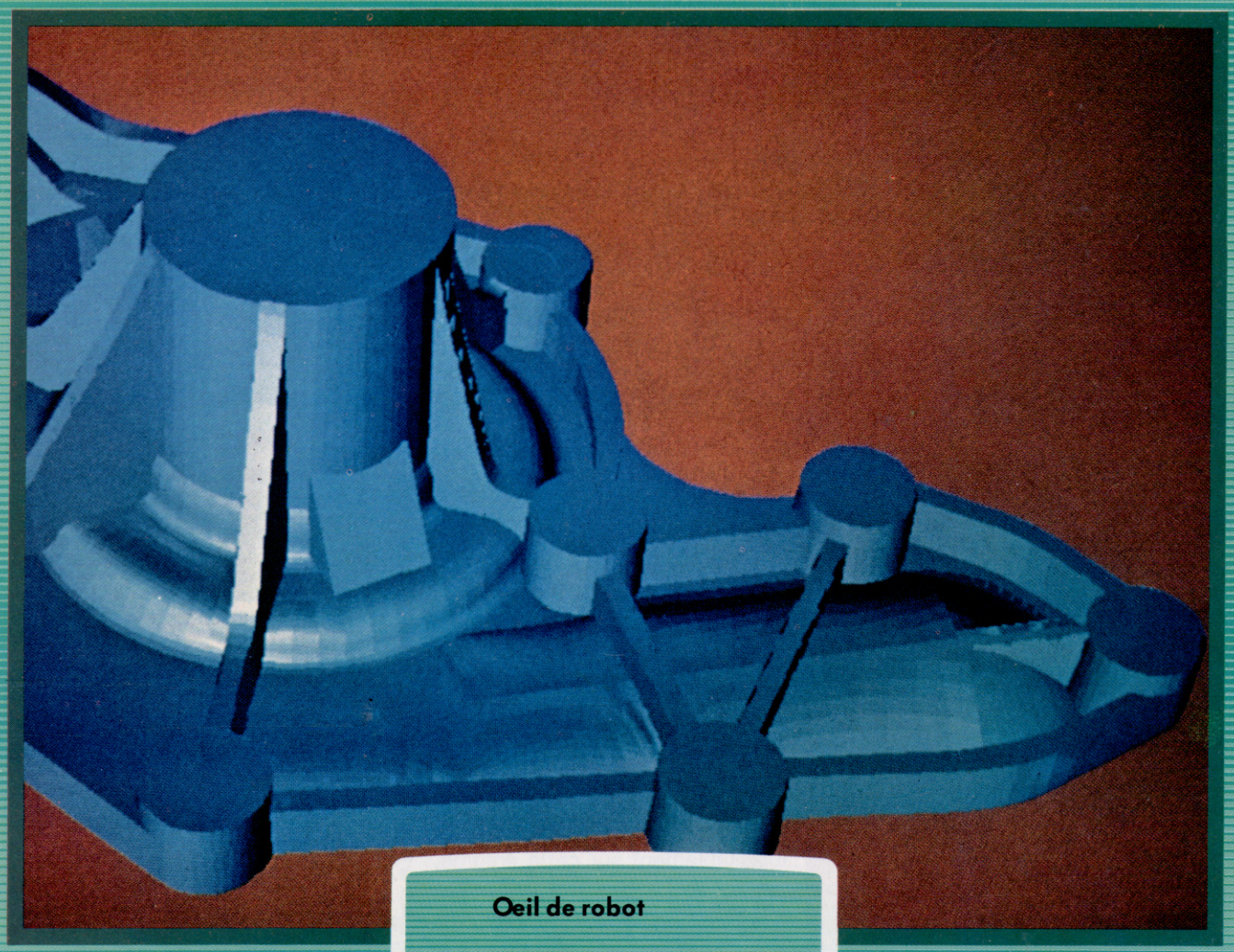
ABC INFORMATIQUE

N° 62

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Oeil de robot

VG 5000 de Philips

Suivez le son

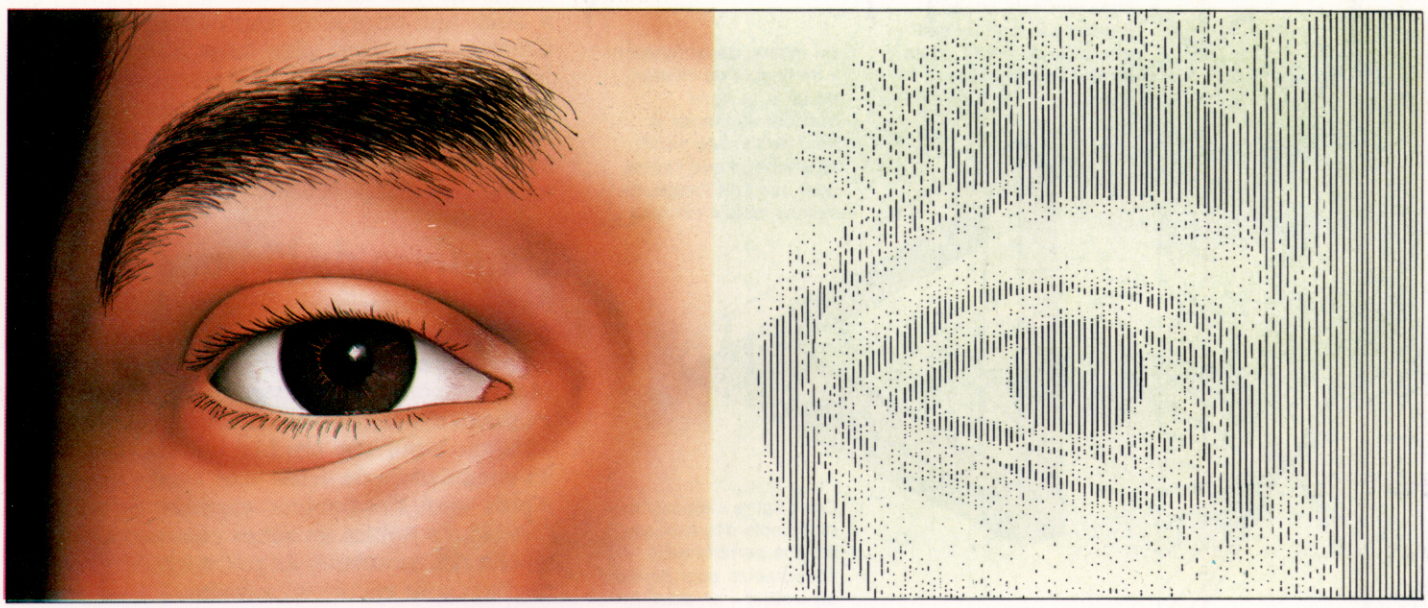
Des chiffres sur Alice

EDITIONS ATLAS

M 6062-62-12,50 F 90 FB - 3,80 FS - $ 1.95

Œil de robot

Après avoir rendu le robot « intelligent » par une programmation adéquate, examinons maintenant s'il lui est possible de « voir » les objets qui l'entourent.



L'homme et le robot
La vision chez l'homme est étroitement dépendante des interactions entre un système très complexe de nerfs et de récepteurs sensoriels et le cerveau, qui traite les informations qu'il reçoit d'eux. Une image visuelle est une structure composée de zones claires et de zones sombres, telle qu'elle s'imprime sur la rétine; mais la vision proprement dite est l'œuvre du cerveau. Un robot ne procède pas autrement, mais sa précision reste, bien sûr, de beaucoup inférieure.
(Cl. Steve Cross.)

L'homme dispose de cinq sens, et la vision est sans doute le plus important; sans elle, notre connaissance du monde extérieur resterait extrêmement limitée. Il en va de même pour un robot. Nous avons vu, précédemment, comment il peut faire usage de capteurs pour détecter la présence d'un objet placé sur son chemin. Cherchons maintenant à mettre au point un système qui puisse le doter d'une perception visuelle aussi efficace que celle d'un être humain.

L'œil dispose d'un iris qui joue à peu près le rôle d'un objectif; il contrôle le niveau de lumière qui pénètre l'œil et focalise une image sur la rétine. Mais, en fait, le globe oculaire ne « voit » rien du tout; il se borne à recueillir un signal, à lui donner une nouvelle forme, plus acceptable, et à le transmettre au cerveau, qui procède alors à une analyse des données reçues; c'est ce traitement d'informations qui constitue réellement la vision.

Pour rendre un robot capable de performances analogues, nous devrons donc diviser le problème en deux parties bien distinctes. Il nous faudra d'abord faire usage d'un « œil » qui puisse servir de détecteur; les signaux qu'il recevra devront ensuite être traités par ordinateur, faute de quoi ils n'auront aucune utilité.

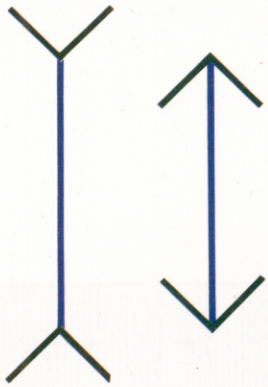
Le dispositif de réception ne pose pas de gros problèmes : à la limite, une simple cellule photoélectrique peut suffire. Elle produira un signal correspondant à l'intensité lumineuse totale du champ de vision. Cette technique est assez bien adaptée si nous cherchons simplement à guider notre robot vers une source lumineuse, ou à lui faire suivre une ligne blanche peinte sur un fond sombre. Le programme de traitement n'a pas besoin d'être très complexe, puisqu'il traite une information limitée, et que celle-ci ne provoque chez le robot qu'une série très limitée de réactions.

Toutefois, on ne peut pas encore parler de « vision », au sens propre du terme. Il nous faudrait un système capable de donner une image en deux dimensions du monde réel, que le « cerveau » de notre engin pourrait ensuite examiner, et à partir de laquelle il parviendrait à certaines conclusions.

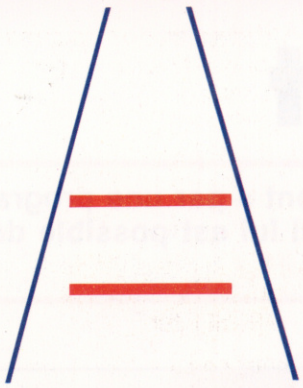
Une solution consiste à réutiliser une cellule photoélectrique, mais cette fois équipée d'un objectif placé devant elle. Un déplacement successif de l'ensemble permettrait de balayer tout le champ de vision, jusqu'à obtention d'une image complète. Elle serait ensuite gardée en mémoire par l'ordinateur. En pratique, cette méthode est malheureusement très lente et peu fiable.

Dans la plupart des cas, l'« œil » du robot sera une caméra vidéo, comme on en voit dans les studios de télévision; il pourra s'agir également d'un appareil spécialement construit à cet effet. C'est ainsi que certains modèles font usage de RAM optiques, c'est-à-dire de mémoires RAM dans lesquelles chaque octet prend une valeur donnée correspondant à l'intensité lumineuse qu'il reçoit. De

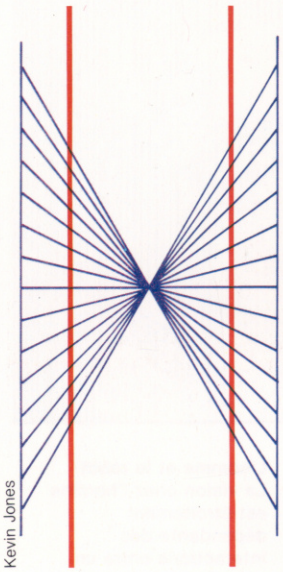
Ce que vous croyez voir



L'illusion d'optique dite « de Muller-Lyer » vous amène à croire que la ligne verticale située à gauche est plus longue que celle de droite, alors qu'en fait elles sont égales.

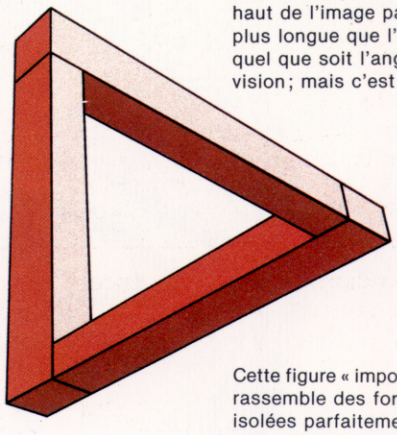


De même, dans l'illusion « de la voie de chemin de fer », la ligne horizontale placée en haut de l'image paraît plus longue que l'autre, quel que soit l'angle de vision ; mais c'est faux...

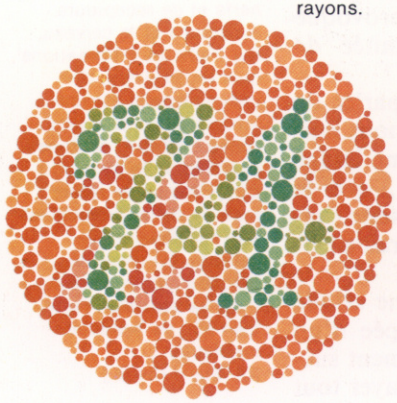


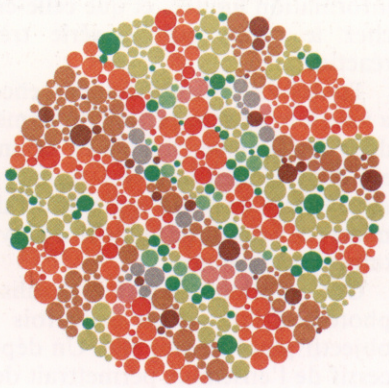
Kevin Jones

Les deux lignes verticales semblent courbes, alors qu'elles sont parfaitement droites — mais l'œil les regarde par rapport aux rayons.



Cette figure « impossible » rassemble des formes isolées parfaitement acceptables pour donner un « objet » en deux dimensions que les sujets soumis à des tests sont incapables de reconnaître en tant que tel.





Les deux motifs colorés ci-dessus sont tirés des tests d'Ishihara, destinés aux daltoniens. Les sujets normaux verront nettement à gauche le nombre 74, et n'identifieront à droite tout au plus un vague 2. Les daltoniens verront respectivement, à gauche, le nombre 21 et, à droite, un 2 parfaitement discernable.

tels dispositifs sont de moins en moins coûteux ; ils peuvent par surcroît stocker toute l'information visuelle relative à la scène observée par le robot.

En général, les données ainsi recueillies sont rassemblées au sein d'un tableau à deux dimensions, dont chaque élément contient une valeur qui est liée à la lumière tombant sur tel ou tel point du décor. Le nombre d'éléments contenus dans le tableau nous donnera la *résolution* de l'image elle-même, et l'éventail des valeurs au sein de chaque tableau nous permettra de définir une *échelle des gris,* donc des nuances discernables. En règle générale, on appelle « pixel » (abréviation de *picture element*) tout élément de base du tableau. Une résolution de 500 × 250 signifie qu'une image se compose de 125 000 pixels différents, dont chacun s'est vu allouer un octet. La résolution horizontale est de 500 pixels, contre 250 pour la résolution verticale, et chacun d'eux peut prendre en tout 256 valeurs différentes, soit varier du noir au blanc pur. Pour avoir une idée de ce que cela représente, rappelons-nous qu'une image télévisée est composée de 625 lignes : elle a donc une résolution verticale de 625 pixels. Comme l'écran est plus large que haut, on a besoin horizontalement de 1 000 pixels environ ; et le tout peut faire usage de 256 niveaux de luminosité. Un robot peut très bien être équipé d'un appareillage de ce genre, afin de fournir à l'ordinateur une image acceptable.

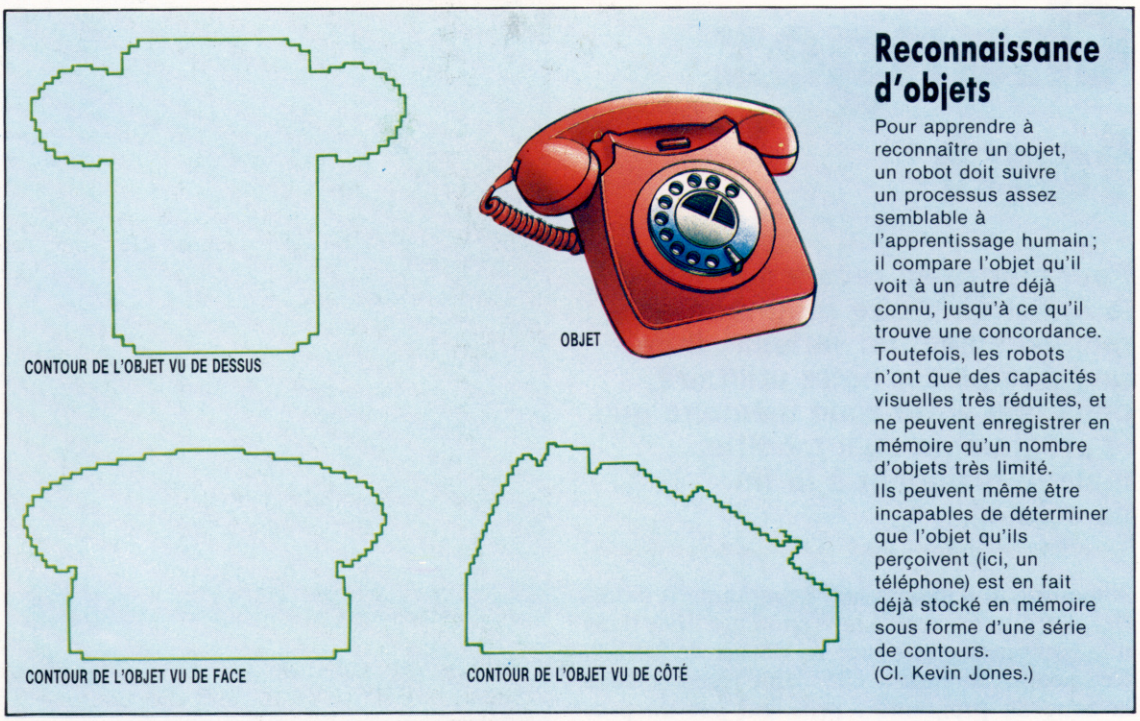
L'analyse de l'image

Pour que le « cerveau » du robot puisse « voir » une image donnée, il doit mener à bien, dans un ordre déterminé, toute une série d'opérations. La première consiste à ajuster les niveaux de gris ; les pixels ayant des valeurs voisines subissent un « lissage » qui leur attribue une valeur unique ; les irrégularités de détail sont ainsi supprimées. Après quoi l'ordinateur examine de nouveau l'image, en notant les zones adjacentes qui présentent des constrastes de gris marqués, et ces différences sont encore accentuées. L'idée de base est évidemment que les caractéristiques essentielles de l'image sont presque toujours indiquées par des limites (lignes, bords) marquées par de brusques changements des niveaux de gris, que l'ordinateur renforce pour mieux les prendre en compte.

Il lui faut alors examiner le tout une troisième fois, de façon à repérer les grandes zones correspondant à tel ou tel niveau de gris. Puis il les traite un peu de la façon dont un être humain joint des points numérotés pour faire apparaître une image ; mais, précisément, l'ordinateur ne dispose d'aucune numérotation de ce type et se borne à suivre certaines règles de vraisemblance très générales. Il est alors en possession d'une image interne de la scène qui est devant lui ; il l'a « lissée », et a tracé des traits autour des objets qui lui paraissent importants.

Mais est-ce bien cela « voir ? » Jusqu'à présent le robot a procédé à diverses transformations, sans « savoir » encore ce qu'il regarde.

Il y a deux solutions à ce problème. La première consiste à programmer l'ordinateur en lui fournissant une série d'instructions très simples, qui prendront la forme de remarques élémentaires concernant le monde visuel. C'est ce qu'on appelle volontiers une approche « de bas en haut », parce que le robot part du niveau le plus limité, et s'efforce ensuite, à partir de ce qu'il « voit », de parvenir à un degré de compréhension plus élevé. La seconde méthode consiste à lui fournir une série prédéfinie des objets qu'il sera amené à voir : il devra les comparer à ce qui



Reconnaissance d'objets

Pour apprendre à reconnaître un objet, un robot doit suivre un processus assez semblable à l'apprentissage humain; il compare l'objet qu'il voit à un autre déjà connu, jusqu'à ce qu'il trouve une concordance. Toutefois, les robots n'ont que des capacités visuelles très réduites, et ne peuvent enregistrer en mémoire qu'un nombre d'objets très limité. Ils peuvent même être incapables de déterminer que l'objet qu'ils perçoivent (ici, un téléphone) est en fait déjà stocké en mémoire sous forme d'une série de contours. (Cl. Kevin Jones.)

lui est présenté pour s'assurer que tel ou tel d'entre eux est bien présent. Elle est donc l'inverse de la première, et on y voit d'ailleurs une approche « de haut en bas », puisque le robot a dès le départ une idée très précise de ce qu'il peut voir, et se borne à vérifier certaines concordances.

La reconnaissance d'objets

Pour mieux saisir la différence entre les deux procédés, considérons un robot placé devant une table. L'approche « de bas en haut » consiste à analyser l'image ainsi produite, et à identifier quatre parties verticales surmontées d'un grand élément horizontal. Le programme de l'ordinateur lui permettra d'identifier une grande surface reposant sur quatre pieds — c'est-à-dire un type de structure appelé table. L'approche « de haut en bas » amène d'abord le robot à se demander : « Est-ce que c'est une table? » Il peut se poser cette question parce qu'il dispose déjà en mémoire d'un modèle de table auquel il comparera les données visuelles qu'il enregistre.

La première méthode lui permet généralement d'aborder des objets inconnus, et de tirer de cette rencontre certains renseignements — encore faut-il le pourvoir de règles de reconnaissance très détaillées, afin qu'il puisse faire face à toutes les circonstances. Inversement, la seconde méthode, si elle résout ce problème, ne pourra lui faire reconnaître que certains objets définis à l'avance : toute nouveauté inattendue fera naître des difficultés.

Les concepteurs de robots ont recours aux deux types d'approche, parfois réunies. Il semble bien que les humains ne se comportent pas autrement; mais la perception visuelle est chez nous automatique, et nous n'avons pas conscience des processus qui la régissent.

Cependant, nous sommes encore loin du but, et pour diverses raisons. La plus importante est qu'il faut, pour traiter une seule image, une puissance de traitement considérable. Une image de 125 000 pixels est stockée sur autant d'octets, ce qui représente plus de 122 K de mémoire. Par ailleurs, les procédures mathématiques mises en œuvre à l'occasion du « lissage » sont très complexes — nous n'en avons donné qu'une présentation simplifiée. Si l'on veut qu'un robot observe le monde « en temps réel » (donc en suivant les événements au moment même où ils surviennent), il devra recevoir 25 images par seconde, comme sur les caméras de télévision; cela signifie que le robot devra analyser pas moins de 3 050 K d'informations par et chaque seconde. A titre de comparaison, c'est à peu de chose près l'équivalent du contenu de plus d'une dizaine de disques souples!

On peut contourner la difficulté de deux façons. Certains dispositifs spécialisés, désormais disponibles, peuvent se charger du traitement d'images. De tels matériels commencent à devenir disponibles sur le marché. Inversement, il est également possible de réduire la résolution de l'image et la gamme des gris. Les opérations d'analyse seront plus rapides, mais évidemment la qualité de reproduction en souffrira.

A vrai dire, le problème de la « vision » des robots n'est pas mieux résolu que celui de la vision humaine. Tous les engins équipés de systèmes de ce type commettent des erreurs. Il se peut que la seule solution consiste à mettre au point un robot capable d'« apprendre » à voir les choses, plutôt que d'emmagasiner ce qu'il peut, ou ne peut pas, voir. Il est même vraisemblable qu'il ne pourra « voir » ces choses sans qu'il ait reçu les moyens de prendre une plus large connaissance du monde extérieur — en fait, au moins comparable à la nôtre.

Changez tout

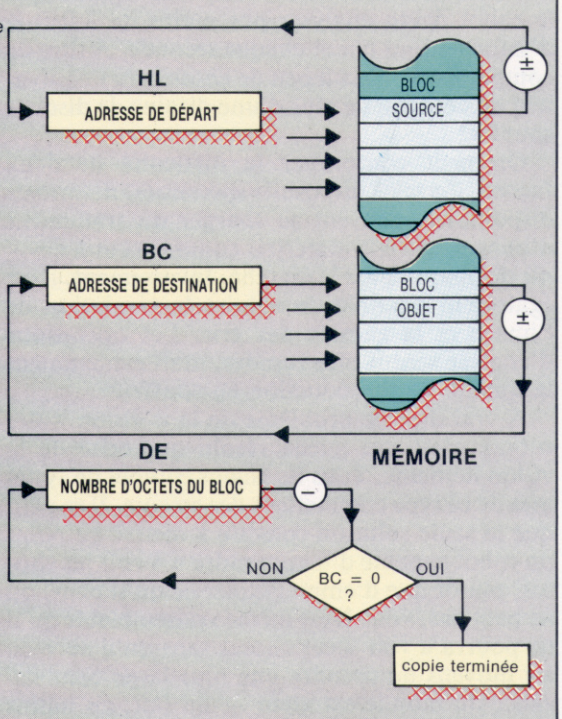
Pour mettre en œuvre sur le Spectrum notre programme de remplacement de variable, il ne faut pas placer notre utilitaire dans une autre zone mémoire que le programme qu'il modifie, mais le fusionner à la fin de celui-ci.

A mesure qu'il manipule le programme à modifier, l'utilitaire recopie la version modifiée dans une zone mémoire située au-dessus de RAMTOP. Cette copie est ensuite elle-même recopiée dans la zone du programme principal par un programme en langage machine qui ajuste l'espace disponible au cas où la longueur de l'ensemble aurait changé. La nouvelle version peut ainsi tenir à l'aise dans la zone accordée au BASIC.

La première partie du programme est très semblable au programme de recherche de variable, vu précédemment. Elle compte cependant quelques variables supplémentaires, comme Altprog, qui pointe le début de la zone affectée à la copie,

Écriture automatique

LDIR et LDDR font partie du jeu d'instructions du Z80; elles permettent le transfert d'un bloc mémoire par incrémentation ou décrémentation automatique. Le registre HL pointe le départ du bloc source, DE l'adresse de destination, et BT contient le nombre d'octets qui composent le bloc. LDIR et LDDR recopient alors octet par octet, et incrémentent ou décrémentent automatiquement HL et DE, et décrémentent BC jusqu'à ce qu'il atteigne zéro, ce qui marque la fin de l'opération. A noter que LDIR est une routine de copie « mécanique »; c'est le programmeur qui procède aux ajustements nécessaires.
(Cl. Kevin Jones.)



Remplacement de variable

```
9000 INPUT "Nom à rechercher ?"; LINE t$
9005 INPUT "A remplacer par ?"; LINE r$
9010 FOR i=1 TO LEN (t$)
9020 IF t$(i)>="a" AND t$(i)<="z" THEN
LET t$(i) = CHR$ (CODE (t$(i))- 32)
9030 NEXT i
9040 LET TokenpourREM =234
9050 LET Guillemet=34
9060 LET Nouvligne=13
9070 LET Soulignement=95
9080 LET Nombre=14
9090 LET PROG=23635
9100 LET Textpointeur=PEEK (PROG)+256*PEEK(PROG+1)
9102 LET Altprog=45000
9105 LET Altpointeur=Altprog
9110 LET Numligne=256*PEEK(Textpointeur)+PEEK
(Textpointeur+1)
9111 PRINT Numligne
9120 IF Numligne>=9000 THEN GOTO 9600
9130 LET q=2 : GOSUB 9800
9135 LET Longadres=Altpointeur
9140 LET Lignesuiv=Textpointeur+2+PEEK(Textpointeur)
+256*PEEK(Textpointeur+1)
9150 LET q=2 : GOSUB 9800
9160 LET Octet=PEEK (Textpointeur) : LET
q=1 : GOSUB 9800
9170 IF Octet=Nouvligne THEN GOTO 9110
9180 IF Octet<>TOKENpourREM THEN GOTO 9220
9190 REM Copie les REM sans changement
9200 LET q=Nouvligne-Textpointeur : GOSUB
9800
9210 GOTO 9110
9220 IF Octet<>Guillemet THEN GOTO 9200
9230 REM Copie tout ce qui est entre guillemets,
mais s'arrête en fin de ligne en cas de paren-
thèse non assortie
9235 LET q=1
9240 IF PEEK(Textpointeur+q-1)=Nouvligne
THEN GOSUB 9800 : GOTO 9110
9250 IF PEEK(Textpointeur+q-1)=Guillemet
THEN GOSUB 9800 : GOTO 9160
9260 LET q=q+1
9270 GOTO 9240
9280 REM Copie un nombre binaire à 5 octets
9290 IF Octet=Nombre THEN LET q=5 : GOSUB
9800 : GOTO 9160
9310 REM Le premier caractère du nom doit
être une lettre majuscule ou minuscule
9320 IF Octet>=CODE ("A") AND Octet<=CODE("Z")
THEN LET c$=CHR$ (Octet) : GOTO 9370
9330 REM Emploie une majuscule au lieu d'une
minuscule
9340 IF Octet>=CODE ("a") AND Octet<=CODE("zz")
THEN LET c$=CHR$(Octet-32): GOTO 9370
9360 GOTO 9160
9370 LET n$=""
9380 LET n$=n$+C$
9400 REM Une lettre, un chiffre ou un trait
de soulignement après le premier caractère
du nom
9410 PEEK (Textpointeur)>=CODE("A") AND PEEK
(Textpointeur)<=CODE("Z") THEN LET c$=CHR$
(PEEK(Textpointeur)) : LET Texpointeur=Textpointeur+1:
GOTO 9380
9420 REM Emploie une lettre majuscule au
lieu d'une minuscule
9430 IF PEEK(Textpointeur)>=CODE("a") AND
PEEK(Textpointeur)<=CODE("z") THEN LET c$=CHR$
(PEEK(Textpointeur)-32) : LET Textpointeur=
Textpointeur+1 : GOTO 9380
9440 IF PEEK(Textpointeur>=CODE("0") AND
PEEK (Textpointeur)<=CODE("9")THEN
LET Textpointeur=Textpointeur+1 : GOTO 9380
9450 IF PEEK (Textpointeur)=Soulignement
THEN LET c$=CHR$ (PEEK (Textpointeur)) : LET
Textpointeur=Textpointeur+1 : GOTO 9380
9460 REM Termine par $ pour une variable
de chaîne
9470 IF PEEK (Textpointeur)=CODE("$") THEN
LET n$=n$+"$" : LET Textpointeur=Textpointeur+1:
GOTO 9500
9480 REM (En cas de tableau ou de fonction
9490 IF PEEK (Textpointeur)=CODE("(") THEN
LET n$=n$+CHR$ (PEEK(Textpointeur)) : LET
Textpointeur=Textpointeur+1
```

```
9500 IF n$=t$ THEN LET n$=r$
9505 LET Alpointeur=Altpointeur-1
9510 FOR p==1 TO LEN (n$)
9520 POKE Altpointeur, CODE (n$(p))
9530 LET Altpointeur=Altpointeur+1
9540 NEXT p
9550 IF n$<>r$ THEN GOTO 9160
9560 LET LongBas=PEEK (Longadres)+LEN(r$)-
LEN (t$)
9570 IF LongBas > 255 THEN LET LongBas=LongBas-
256 : POKE Longadres+1, 1+PEEK(Longadres+1)
9580 POKE Longadres, LongBas
9590 GOTO 9160
9599 REM Se prépare à ramener le programme
modifié dans la zone du programme principal
9600 LET Anclong=Textpointeur-(PEEK(PROG)+256*PEEK
(PROG+1))
9610 LET Nouvlong = Altpointeur-Altprog
9620 POKE 45060,Nouvlong-256*INT(Nouvlong/256)
9630 POKE 45061, INT (Nouvlong/256)
9660 POKE 45056, Textpointeur-256*INT(Textpointeur/25
9670 POKE 45057, INT (Textpointeur/256)
9680 IF Anclong=Nouvlong THEN RANDOMIZE USR
(45084)
9690 IF Anclong < Nouvlong THEN LET X=Nouvlong-
Anclong
9700 IF Anclong > Nouvlong THEN LET X=Textpointeur-
(Anclong-Nouvlong)
9710 POKE 45058, X-256*INT (X/256)
9720 POKE 45059, INT (X/256)
9730 IF Anclong < Nouvlong THEN RANDOMIZE
USR 45062
9740 IF Anclong > Nouvlong THEN RANDOMIZE
USR 45074
```

Chargeur langage machine

```
 10 CLEAR 45055
 20 LET a=45062
 30 FOR i=1000 TO 1040 STEP 10
 40 LET s=0
 50 FOR a=a TO A+7
 60 READ b
 70 POKE a,b
 80 LET s=s+b
 90 NEXT a
100 READ c
110 IF s <> c THEN PRINT "ERREUR DE DATA A LA
LIGNE";i: STOP
120 NEXT i
1000 DATA 42,0,176,237,75,2,176,205,913
1010 DATA 85,22,24,10,42,0,176,237,596
1020 DATA 91,2,176,205,229,25,33,176,937
1030 DATA 179,237,91,83,92,237,75,4,998
1040 DATA 176,237,176,207,255,0,0,0,1051
```

Programme en assembleur

```
0000              MKROOM  EQU     $1655
0000              RCLAM1  EQU     $19E5
0000              T0      EQU     $B000
0000              T1      EQU     $B002
0000              T2      EQU     $B004
0000              ALTPRG  EQU     $B3B0
0000              PROG    EQU     $5C53
B006                      ORG     $B006
B006  2A00B0      UP      LD      HL, (T0)
B009  ED4B02B0            LD      BC, (T1)
B00D  CD5516              CALL    MKROOM
B010  180A                JR      COPY
B012  2A00B0      DOWN    LD      HL, (T0)
B015  ED5B02B0            LD      DE, (T1)
B019  CDE519              CALL    RCLAM1
B01C  21B0B3      COPY    LD      HL, ALTPRG
B01F  ED5B535C            LD      DE, (PROG)
B023  ED4B04B0            LD      BC, (T2)
B027  EDB0                LDIR
B029  CF                  RST     8
B02A  FF                  DB      $FF
```

ou Altpointeur, qui s'assure de la destination du prochain octet du programme modifié. La principale modification consiste en fait à recopier le programme, au lieu de se borner à le lire. Cette tâche est accomplie par la sous-routine qui commence à la ligne 9800. Elle copie le nombre d'octets spécifiés par q, et réactualise les pointeurs de l'ancien et du nouveau programme.

Une fois l'ensemble du programme à modifier transformé et recopié dans une autre zone mémoire, le programme BASIC calcule les valeurs dont le code machine aura besoin pour recopier de nouveau cette nouvelle version, puis il les POKE aux adresses mémoires nécessaires.

Il se peut que l'ancien et le nouveau programme aient la même longueur; dans ce cas, le second peut être recopié à l'emplacement occupé par le premier. La seule information indispensable est alors cette longueur elle-même.

Si en revanche le nouveau programme se révèle plus long, il faudra libérer de l'espace en déplaçant vers le haut la routine de déplacement de variable que nous désirons conserver. Il faudra donc faire appel à la sous-routine MAKE-ROOM, implantée en ROM à l'adresse $1655. Le registre HL doit dans ce cas contenir l'adresse située après le lieu où l'on fait de la place, tandis que le registre BC doit enregistrer la longueur exacte de l'espace nécessaire. Ces deux valeurs sont assez faciles à trouver; celle de HL est la valeur finale de la variable Textpointeur; celle de BC est la différence entre les longueurs de l'ancien et du nouveau programme.

Si ce dernier est plus court que son prédécesseur, nous devrons déplacer vers le bas la routine de remplacement de variable. Pour cela, nous ferons usage de la routine ROM RECLAIM-1. HL doit contenir l'adresse du premier octet à laisser tel quel, et DE celle du premier octet à modifier. Là encore, la valeur de HL est celle de Textpointeur, et celle de DE est calculée en soustrayant la différence entre les deux longueurs de programme de Textpointeur.

Le programme modifié est recopié dans la zone mémoire principale par le biais de l'instruction LDIR (LoaD with Increment and Repeat, « charger avec incrémentation et répéter »). L'adresse de départ du nouveau programme est placée en HL, celle du programme d'origine en DE, tandis qu'en BC est mémorisée la longueur du nouveau programme, qui est alors déplacé, octet par octet, par LDIR.

Les deux dernières lignes du programme recourent à une autre routine implantée en ROM, à l'adresse 8. C'est elle qui a pour fonction d'afficher les messages d'erreur et autres. Elle est appelée par l'instruction RST 8, et le message qu'elle produit est déterminé par l'octet qui suit l'instruction elle-même. En effet, il a pour valeur le chiffre affecté au message, moins 1. En hexadécimal FF, ou −1, donnera OK ou Program Finished. 0 donne NEXT without FOR, et ainsi de suite. Le programme en langage machine ne prend pas fin, comme il est de règle d'habitude, par l'instruction RET, afin de ne pas revenir au programme BASIC qui vient juste d'être déplacé.

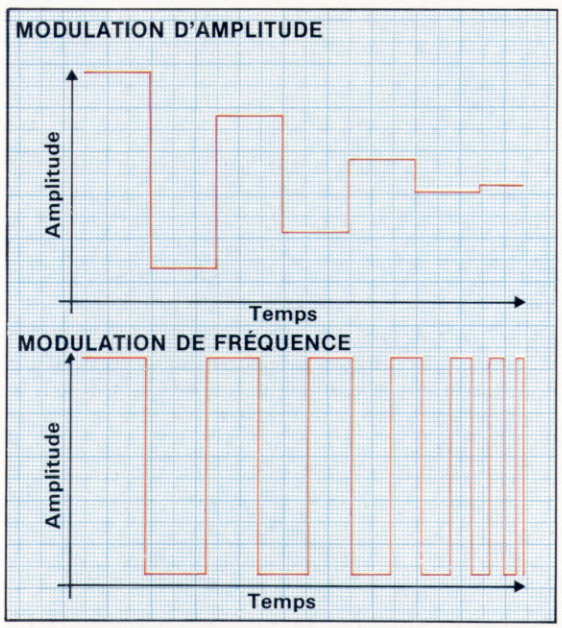
Définition du son

Nous avons déjà conçu un code machine servant à générer trois types d'ondes sonores : carrées, en dents de scie et sinusoïdales. Examinons deux autres paramètres sonores : volume et hauteur.

Le volume est fonction de la différence qui existe entre la valeur maximale de la forme d'onde et sa valeur minimale. Cette propriété d'une onde sonore est nommée amplitude.

En utilisant un programme BASIC simple pour faire osciller des valeurs placées dans le registre du port utilisateur, nous pouvons montrer qu'il est facile de commander l'amplitude d'une onde numérique.

```
10 REM **** FREQ/AMPLITUDE CBM ****
20 :
25 DDR=56579:DATREG=56577
30 POKE DDR,255:REM TOUTES SORTIES
40 FOR I=255 TO 0 STEP-15
50 FOR J=1 TO 100
60 POKE DATREG,I:POKE DATREG,0
70 NEXT J,I

800 REM **** PROGRAMME EXEMPLE ****
805 :
806 UP$=CHR$(145
810 DIV=49798: REM ADRESSE FACTEUR AMPLITUDE
820 DEL=49799: REM ADRESSE FACTEUR TEMPORISATION
830 TME=49800: REM ADRESSE FACTEUR DUREE
840 CALL=49801:REM ADRESSE DEPART PROGRAMME
850 :
860 DDR=56577:POKEDDR,255:REM TOUTES SORTIES
870 :
880 PRINTCHR$(147):REM EFFACE ECRAN
890 PRINT:INPUT"AMPLITUDE FACTOR 0-7";AF
900 IF AF<0 OR AF>7  THEN PRINT UP$:IP$::GOTO890
910 POKE DIV,AF
920 :
930 PRINT:INPUT"DELAY FACTOR 1-101":DF
940 IF DF<1 OR DF>101 THEN PRINT UP$:UP$::GOTO930
945 POKE DEL,DF
950 :
960 PRINT:INPUT"FACTEUR DUREE 0-15":TF
970 IF TF<0 OR TF>15 THEN PRINTUP$:UP$::GOTO960
980 POKE TME,TF
990 :
1000 SYS CALL
1010 GETA$:IFA$="" THEN 1010
1020 IFA$="X" THEN 880:REM DEPART
1030 GOTO 1000REM AUTRE SIGNAL SONORE
```



Construction modulaire Les ondes peuvent être modulées soit au niveau de la fréquence, soit au niveau de l'amplitude. La fréquence modifie la hauteur du son et est déterminée par le nombre de cycles produits par seconde. L'amplitude modifie le volume du son et correspond à la différence entre les valeurs minimales et maximales d'un cycle. Les diagrammes montrent comment l'amplitude peut être modifiée et comment la fréquence peut être modulée pour rendre un son de plus en plus aigu.

Au début du programme, une onde carrée grossière est générée et oscille entre 255 et zéro. Cela signifie que l'amplitude de l'onde est 255. Lors de l'exécution du programme, la valeur supérieure placée dans le registre de données diminue par étapes de 15. L'amplitude diminue donc également — cela a pour effet, lorsqu'on contrôle le son produit à l'aide d'un amplificateur stéréo ou d'un casque d'écoute, que le volume du son diminue progressivement pour finalement s'éteindre. Le volume d'un son numérique synthétisé peut donc être commandé en limitant l'ensemble des valeurs placées dans le registre de données du port utilisateur. La hauteur d'une note est fonction de la fréquence de l'onde qui l'a produite ; cette valeur représente le nombre de cycles par seconde.

La fréquence peut être commandée numériquement de deux façons. La première consiste à l'augmenter à partir d'une limite inférieure en utilisant quelques échantillons d'onde de moins. Si, par exemple, une onde était composée de cent échantillons, un programme en code machine prendrait un certain temps pour placer successivement chaque valeur dans le registre de données ; un certain nombre d'ondes complètes pourraient alors être produites par seconde. Le nombre d'échantillons définit donc la fréquence du son produit. Pour la doubler, le programme en code machine pourrait ne prendre qu'une valeur sur deux dans la table de données qui définit l'onde, et ainsi de suite, pour tripler, quadrupler... Cette méthode présente deux inconvénients : d'abord, il est difficile d'effectuer de petits ajustements de fréquence sans créer de distorsion de l'onde ; ensuite, lorsque la fréquence d'onde augmente, l'onde générée est de plus en plus éloignée de l'onde initiale, puisque moins d'échantillons sont utilisés.

Une autre méthode consiste à démarrer avec une boucle où sont lues les données de l'onde aussi rapidement que possible, ce qui donne une

fréquence maximale. La fréquence peut être ajustée en insérant de courtes temporisations dans la boucle. Cela permet un contrôle beaucoup plus précis sur la fréquence, mais signifie aussi que le nombre d'échantillons qui composent l'onde doit être petit afin de pouvoir obtenir une fréquence maximale raisonnablement élevée. Nous utiliserons la seconde méthode pour produire un programme en code machine qui nous permettra de commander à la fois la fréquence et le volume.

La meilleure méthode pour réduire l'amplitude de l'onde, tout en conservant sa forme globale, consiste à diviser chaque valeur de la table de l'onde par une constante. Cela peut être réalisé de deux manières : après le changement de chaque valeur dans l'accumulateur, mais avant que la valeur soit placée dans le registre de données ou avant d'entrer dans la boucle principale du programme. La première méthode augmente le temps d'exécution de chaque cycle de la boucle principale, mais comme ce facteur limite la fréquence maximale, nous devrions opter pour l'autre méthode. Une deuxième table est produite à partir de la table de l'onde initiale, en divisant chaque valeur par une constante et en plaçant le résultat dans une nouvelle table. La nouvelle table fournit alors les données de l'onde à l'aide de la boucle principale du programme. Cette méthode de division est rudimentaire. Une amplitude constante indique le nombre de fois que la valeur de la table est décalée à droite. Comme chaque décalage à droite est une division entière par 2, l'utilisation d'un facteur d'amplitude *n* a pour effet de diviser chaque table par $2n$.

Une méthode assez ingénieuse est employée lorsque le facteur d'amplitude est zéro. Dans ce cas, nous utilisons la table initiale et le programme se modifie lui-même afin de spécifier l'adresse de base de la table initiale, et non celle de la table des valeurs divisées.

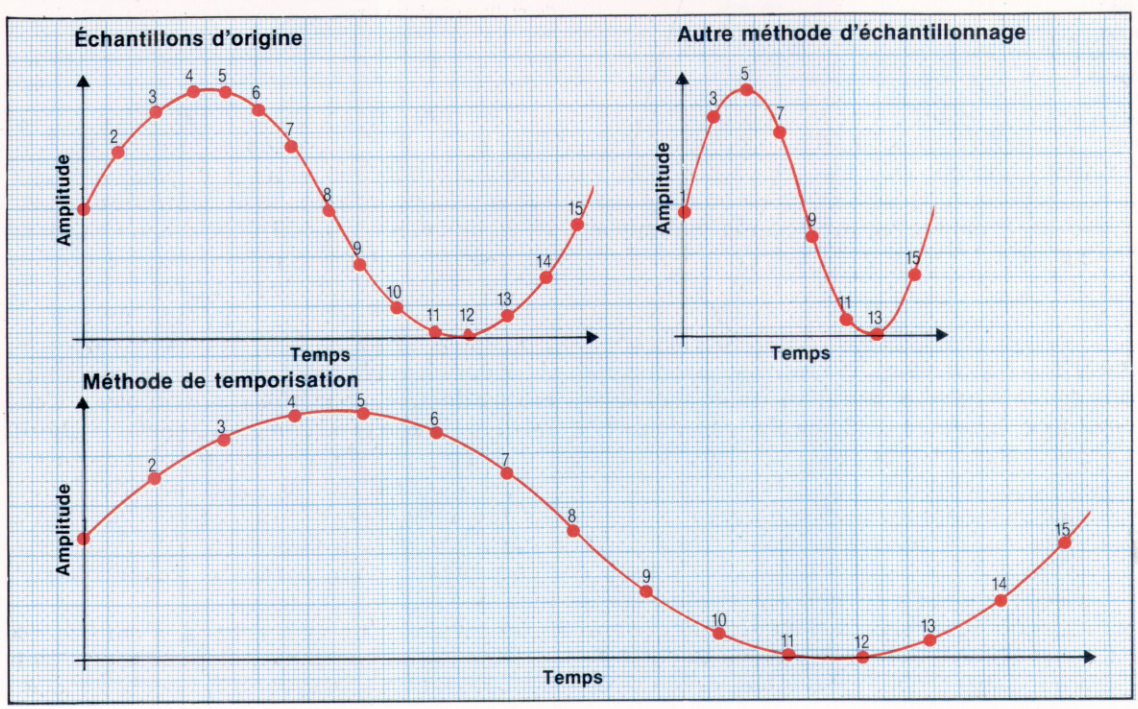
Le principal problème n'est pas de créer une temporisation suffisante, — c'est-à-dire la fréquence la plus basse — mais de créer la temporisation minimale; c'est-à-dire la fréquence maximale. Le code supplémentaire requis pour la temporisation ralentit à un tel point l'exécution qu'il n'est plus possible d'avoir ce nombre d'étapes.

```
                                        Temps en cycles machine
MAIN    LDX # $00                       2
NEXVAL
        LDA AMPTAB, X                   4
        LDY DELAY                       4
MORDEL
        DEY                             2
        BNE MORDEL                      3 (2 si boucle est quittée)
        STA PORT                        4
        INX                             2
        CPX # STEPS                     2
        BNE NEXVAL                      3 (2 si boucle est quittée)
```

Le nombre total de cycles machines nécessaires est donné par : $2 + (4 + 4 + (2 + 3) \times$ temporisation $- 1 + 4 + 2 + 2 + 3) \times$ étapes $- 1 = 1 + (18 + 5 \times$ temporisation$) \times$ étapes; et une valeur minimale de temporisation de 1 donne la fréquence maximale selon cette formule : fréquence maximale = 1 000 000/(1 + 23 × étapes). Pour une fréquence maximale d'environ 3 000 Hz, cette formule donne 15 comme nombre d'étapes.

Tel est le nombre d'échantillons de forme d'onde que nous devrions utiliser pour produire une fréquence maximale raisonnable. En utilisant 15 étapes, la formule initiale peut être écrite ainsi : nombre de cycles machine = 271 + 75 × temporisation.

Le dernier problème qui se présente pour modifier une fréquence est que, pour un certain nombre de répétitions de la boucle de temporisation, la durée du son produit augmentera avec la fréquence. Cela s'explique par le fait que la boucle principale est exécutée plus rapidement. Aussi,



Étirement des courbes
La fréquence d'une onde numérique échantillonnée peut être modifiée en prenant un peu moins d'échantillons ou en insérant une temporisation entre chaque valeur. Si la table de référence initiale de l'onde renferme quinze échantillons, la fréquence de sortie peut être doublée en ne prenant qu'un échantillon sur deux. Les quinze valeurs peuvent aussi être émises en insérant une temporisation pour doubler le temps de sortie, ce qui réduit de moitié la fréquence. La première méthode permet d'utiliser de nombreux échantillons à de faibles fréquences, mais n'autorise qu'un contrôle rudimentaire. La seconde donne un contrôle beaucoup plus précis mais implique qu'un moins grand nombre d'échantillons soient utilisés. (Cl. Liz Dixon.)

pour équilibrer ce phénomène, nous devons prévoir un calcul qui déterminera le nombre de répétitions requises pour produire un son d'une certaine durée quelle que soit la fréquence du son. Si l'unité de durée est fixée à 1/50 de seconde, le nombre de répétitions pour une valeur de temporisation donnée est 1 000 000/(50 × 271 + 75 × temporisation). Un calcul en code machine serait long et difficile; nous pouvons, à la place, composer une table de référence à l'aide de BASIC, en donnant une valeur pour chaque valeur de temporisation selon la formule précédente (c'est-à-dire de 1 à 109). La routine en code machine n'a alors qu'à rechercher la valeur de répétition et à la stocker dans un décrémenteur, afin de produire un son d'une durée de 1/50 de seconde.

Les propriétaires de Commodore possédant un assembleur taperont le listage en code source et l'assembleront pour produire un programme-objet pouvant être sauvegardé. Le programme d'appel rechargera le programme-objet à partir d'une disquette ou d'une cassette et composera les tables de référence. Tapez ce programme et exécutez-le. Tapez NEW et entrez l'exemple de programme BASIC qui donne les diverses adresses de mémoire utilisées par le programme en code machine. Exécutez ce programme avec le boîtier tampon et avec le convertisseur N/A.

```
 10 REM CHARGEUR BASIC POUR PROGRAMME FREQ/AMP
 20 :
 30 FOR I=49801 TO 49911
 40 READ A:POKEI,A
 50 CC=CC+A
 60 NEXT I
 70 READ CS:IF CC<>CS THENPRINT
    "CONTROLE DES ERREURS":STOP
100 DATA172,134,194,208,8,169,0,141
110 DATA204,194,76,174,194,169,15,141
120 DATA204,194,162,15,172,134,194,189
130 DATA0,194,24,106,157,15,194,136
140 DATA208,248,202,16,239,120,169,0
150 DATA141,133,194,174,135,194,189,30
160 DATA194,172,136,194,240,7,10,46
170 DATA133,194,136,208,249,141,132
180 DATA194,162,0,189,15,194,172,135
190 DATA194,136,200,253,141,1,221,232
200 DATA224,15,208,239,173,132,194,56
210 DATA233,1,141,132,194,173,133,194
220 DATA233,0,141,133,194,208,218,169
230 DATA0,205,132,194,208,211,88,96
240 DATA16186:REM*CONTROLE TOTAL*

 10 REM ****      PROGRAMME APPEL      ****
 20 REM ****            ET             ****
 30 REM **** COMPOSITION DE LA TABLE ****
 40 :
 45 DN=8:REM SI CASSETTE DN=1
 50 IF A=0 THEN A=1:LOAD"FREQ.HEX",DN,1
 60 :
 70 REM ****        TABLE ONDE         ****
 75 :
 80 S=15:TB=12*4096+2*256
 90 FOR I=0 TO S-1
100 Y=127*SIN(X)+127
110 POKE TB+I,Y
120 X=X+2/S
130 NEXT I
140 :
150 REM ****    TABLE FREQ/TEMPOR    ****
160 :
170 TB=TB+2*S
180 FOR D=0 TO 101
190 TV=10^6/(50*(271175*D))
200 POKETB+D,TV
210 NEXT D
```

```
**********************
**********************
**                  **
**      CBM 64      **
**    GENERATION    **
**    FREQUENCE     **
**   ET AMPLITUDE   **
**                  **
**********************
**********************

STEPS  =  15      NO. ETAPES PAR CYCLE
PORT   =  56577

**      $C200
SHAPTB ***+STEPS       TABLE ONDE
AMPTAB ***+STEPS       TABLE AMPLITUDE
LOOPTB ***+102         TABLE FREQ/TEMPOR
COUNT  ***+2           COMPTEUR DE BOUCLES
DIVISN ***+1           FACTEUR DIV ONDE
DELAY  ***+1           FACTEUR TEMPOR
TIME   ***+1           FACTEUR DUREE NOTE

**** TABLE AMPLITUDE ****

        LDY DIVISN
        BNE CONT
        LDA #<SHAPTB   MODIFIE PROGRAMME
        STA NEXVAL+1   CHARGE SHAPTB

CONT
        LDA #<AMPTAB
        STA NEXVAL+1
        LDX #STEPS
NEXT
        LDY DIVISN
        LDA SHAPTB,X
MORE
        CLC
        ROR A
        STA AMPTAB,X
        DEY
        BNE MORE
        DEX
        BPL NEXT

**** VALEUR COMPTE ****

        SEI
        LDA #$00
        STA COUNT+1       INIT COMPTE OCTET SUP
        LDX DELAY
        LDA LOOPTB,X
        LDY TIME
        BEQ NOMULT
MULT
        ASL A
        ROL COUNT+1
        DEY
        BNE MULT
NOMULT
        STA COUNT

**** BOUCLE PRINCIPALE ****

MAIN
        LDX #$00
NEXVAL
        LDA AMPTAB,X
        LDY DELAY
MORDEL                    BOUCLE TEMPOR
        DEY
        BNE MORDEL

        STA PORT
        INX
        CPX #STEPS
        BNE NEXVAL

**** DECREMENTE COMPTE ****

        LDA COUNT
        SEC
        SBC #$01
        STA COUNT
        LDA COUNT+1
        SBC #$00
        STA COUNT+1
        BNE MAIN
        LDA #$00
        CMP COUNT
        BNE MAIN
        CLI
        RTS
```

```
15REM ********************************
20REM **                            **
25REM **          BBC               **
30REM **        GENERATION          **
40REM **   FREQUENCE & AMPLITUDE    **
50REM **                            **
60REM ********************************
90MODE 7
95steps=15:port=&FE60
97ddr=&FE62:? ddr=255:REM TOUTES SORTIES
100HIMEM=HIMEM-&101:REM RESERVE ESPACE TABLE
110forme table
112amplitude table=forme table+étapes
114boucle table=amplitude table+étapes
115PROCset tables
120PROCassemble code
140REM **** PROGRAMME TEST BASIC ****
160CLS
170PRINT:INPUT"FACTEUR AMPLITUDE 0-7";AF
180IF AF<0 OR AF>7 THEN 170
```

```
190?div_factor=AF
200PRINT:INPUT"FACTEUR TEMPOR 1-101";DF
201IF DF<1 PR DF>101  THEN 200
220?tempor_facteur=DF
230PRINT:INPUT"FACTEUR DUREE 0-15";TF
240IF TF<0 OR TF>15 THEN 230
250? temps_facteur=TF
265REPEAT
270CALL freq
280A$=GET$
290UNTIL A$="X"
300GOTO 160:REM DEPART
900END
999:
1000DEF PROCassemble_code
1005DIM MC% &FF
1010FOR optX=0 TO 3 STEP 3
1020P%=MC%
1030compte=P%:P%=P%+2
1040div_facteur=P%:P%=P%+1
1050tempor_facteur=P%:P%=P%+1
1060temps_facteur=P%:P%=P%+1
1070[
1075OPT optX
```

```
1080 **** TABLE AMPLITUDE ****
1090
1095.freq
1100      LDY div facteur
1110      BNE cont
1120      LDA #forme_table MOD 256
1130      STA nexval+1
1140      JMP initc
1150
1160.cont
1170      LDA # amplitude_table MOD 256
1180      STA nexval+1
1190      LDX #steps
1195.next
1200      LDY div_facteur
1210      LDA forme_table
1220.more
1230      CLC
1240      ROR A
1250      STA amplitude_table
1260      DEY
1270      BNE more
1280      DEX
1290      BPL next
1300
1310 **** SET VALEUR COMPTE ****
1320
1330.initc
1340      SEI
1350      LDA #0
1360      STA COMPTE+1
1370      LDX tempor_facteur
1380      LDA boucle_table,X
1390      LDY temps_facteur
1400      BEQ nomult
1410.mult
1420      ASL A
1430      ROL compte+1
1440      DEY
1450      BNE mult
1460.nomult
1470      STA compte
1480
1490 **** BOUCLE PRINCIPALE ****
1500
1510.main
1520      LDX #0
1530.nexval
1540      LDA amplitude_table,X
1550      LDY tempor_facteur
1560.mordel
1570      DEY
1580      BNE mordel
1590
1600      STA port
1610      INX
1620      CPX #steps
1630      BNE nexval
1640
1650 **** DECREMENTE COMPTE ****
1660
1670      LDA compte
1680      SEC
1690      SBC #1
1700      STA compte
1710      LDA compte+1
1720      SBC #0
1730      STA compte+1
1740      BNE main
1750      LDA #0
1760      CMP compte
1770      BNE main
1780      CLI
1790      RTS
1800]
1810NEXT optX
1820ENDPROC
1999:
2000DEF PROCset_tables
2005x=0
2010FOR I=forme_table TO forme_table+étapes 1
2020y=127*SIN(x)+127
2030? I=y
2040x=x+2*PI/étapes
2050NEXT I
2060:
2070FOR tempor=0 TO 101
2080boucle_val=10^6/(50*(271+75*delay))
2090boucle_table?tempor=boucle_val
2100NEXT tempor
2120ENDPROC
```

Le VG 5000

Philips se lance dans la micro-informatique avec un micro-ordinateur construit par sa filiale française RTC, destiné surtout aux jeunes et aux petits budgets : le VG 5000.

Développé en France, dans les laboratoires de Radiotechnique (RTC), qui en assure la fabrication, le VG 5000 est le premier micro-ordinateur grand public conçu par Philips. Construit autour du microprocesseur Z80 de Zilog (8 bits), il comporte 18 K de mémoire morte (ROM) et 24 K de mémoire vive (RAM).

Contrairement à son grand frère, le VG 8000, destiné à d'autres pays européens, tels que l'Allemagne et l'Italie, le VG 5000 n'est pas au standard MSX. Il accepte toutefois un certain nombre de périphériques MSX, notamment l'imprimante, mais d'autres caractéristiques du standard n'ont pas été retenues afin que le VG 5000 reste, avant tout, un micro à la portée de toutes les bourses. Son prix est en effet de 1 590 F TTC, alors que celui d'un micro MSX en configuration de base se situe aux alentours de 3 000 F, ce qui place Philips face à la concurrence de Sinclair, Oric, Atari, Thomson, Matra, etc.

Le VG 5000 est essentiellement un outil d'éducation pour l'apprentissage de la pratique des ordinateurs. Il permet la programmation grâce aux 18 K de mémoire morte intégrant un BASIC Microsoft puissant. Afin de faciliter la mise au point des programmes BASIC, des messages d'erreur sont affichés en langage clair, et en français.

Outre le BASIC Microsoft résident, disponible en standard, d'autres langages, tel le LOGO, pourront être développés sous forme de cartouches d'extension; dans ce cas, à l'introduction d'un nouveau langage, le BASIC résident est déconnecté par le moniteur et permet ainsi à l'utilisateur de disposer à nouveau de ses 48 K de RAM adressables.

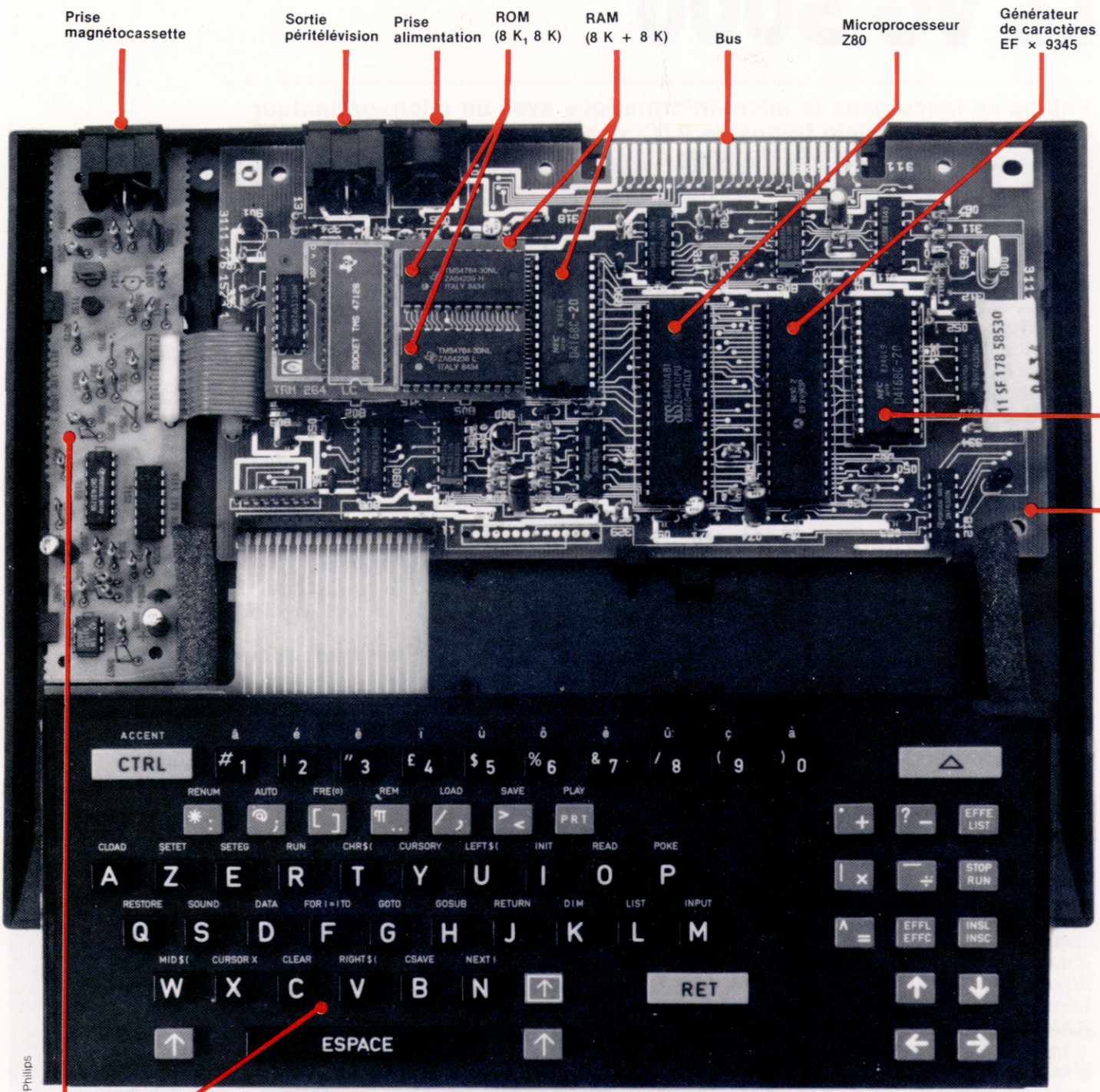
La disponibilité sur cassette audio du programme de conversion BASICODE 2 permettra, après chargement de ce logiciel (quinze secondes environ), d'enregistrer des programmes universels qui pourraient être émis par certaines stations de radio. De plus, le VG 5000 pourra accéder à un parc de logiciels disponibles en BASICODE 2 (éducation, jeux...) et permettra par ce biais l'échange de programmes avec tout autre micro-ordinateur disposant également du programme de conversion correspondant.

Au-delà de l'initiation et de la création de programmes, le VG 5000 dispose d'un catalogue de logiciels diversifiés (voir encadré) qui en font également un outil de distraction. La définition de l'écran — 80 000 points — en fait l'un des ordinateurs grand public parmi les plus performants dans ce domaine, lui donnant la possibilité de disposer de programmes de jeux particulièrement attractifs.

L'affichage de 25 lignes de 40 caractères se fait sur un écran de télévision, chaque caractère étant défini dans une matrice de 10 × 8 pixels; le nombre de couleurs disponibles est de huit. Un affichage en mode texte de 15 lignes de 80 caractères est accessible en assembleur. L'éditeur pleine page du VG 5000 permet la manipulation sous toutes ses formes des caractères et des lignes. En plus des cent vingt-huit caractères semi-graphiques disponibles en standard, l'utilisateur a la possibilité de redéfinir ses cent quatre-vingt-douze caractères graphiques par la fonction SETEG préprogrammée. La fonction INIT préprogrammée permet à l'utilisateur le choix des couleurs de fond et de bord de page, et ainsi la création de dessins artistiques. De plus, les possibilités d'adressage point par point autorisées par l'utilisation de sous-programmes en assembleur permettent la réalisation d'écrans à haute définition.

Dix fonctions musicales sont accessibles directement par touches préprogrammées et permettent, l'une (SOUND), de créer les sons les plus divers, l'autre (PLAY), de programmer des airs de musique sur quatre octaves avec dièses et bémols en notation internationale.

Le VG 5000
Le premier micro-ordinateur de Philips a été conçu pour un public de jeunes passionnés de micro-informatique. Son BASIC Microsoft très performant, une bonne définition d'affichage, un vaste catalogue de logiciels très variés et une compatibilité avec certains périphériques MSX en font un outil d'initiation très honorable pour un prix de moitié inférieur à celui des micros MSX.



Le clavier AZERTY, conçu et fabriqué en France dans l'usine RTC d'Évreux, est en parfaite synergie avec le clavier des terminaux Minitel. Bien que de disposition française, ce clavier est mal adapté au traitement de texte, étant donné la disposition des touches des minuscules accentuées. Celles-ci se trouvent séparées des touches alphabétiques normales par une rangée de touches de fonctions, et ne sont accessibles qu'en appuyant simultanément sur la touche CTRL/ACCENT. En outre, l'accent circonflexe, comme les autres accents, n'existe qu'associé à certaines voyelles (a, e, o). Il est par conséquent impossible d'écrire î ou û, ce qui est plutôt gênant dans la langue française. Il en est de même des trémas qui n'existent que sur le i (ï).

Quoi qu'il en soit, le VG 5000 atteint un très bon rapport spécifications/prix : c'est là le premier souci de ses concepteurs. Ceux-ci ont aussi voulu en faire un micro-ordinateur ouvert sur l'avenir et ses évolutions technologiques. En 1985, il disposera d'une extension qui pourra étendre sa mémoire vive à 56 K. Ce module permettra la connexion d'imprimante, de lecteurs de disquettes, d'interfaces vidéodisque, de carte à mémoire, de synthétiseur musical, etc. Par ailleurs, des versions avec écran intégré noir et blanc ou couleur pourraient être proposées au public.

Instructions basic du VG 5000

&"..." - ABS - ACTION - AND - ASC - ATN - AUTO - CALL - CHR$ - CLEAR - CLOAD - CONT - COS - CSAVE - CURSORY - CURSORX - DATA - DEF FN - DELIM - DIM - DISPLAY - EG - END - ET - EXP - FRE - FOR ... STEP - NEXT - GOSUB - RETURN - GOTO - GR - IF ... GOTO - IF ... THEN - INIT - INPUT - INT - LEFT$ - LEN - LET - LIST - LLIST - LOAD - LOG - LPOS - LPRINT - MID$ - NEW - NOT - ON ... GOSUB - ON ... GOTO - OR - PAGE - PEEK - PLAY - POKE - POS - PRINT - READ - REM - RENUM - RESTORE - RIGHT$ - RND - RUN - SAVE - SCREEN - SCROLL - SETEG - SETET - SGN - SIN - SOUND - SPC - SQR - STICKX - STICKY - STOP - STORE - STR$ - TAB - TAN - TX - USR - VAL.

Les logiciels

Le catalogue de logiciels, particulièrement équilibré, se répartit en jeux, programmes éducatifs et logiciels d'application. Parmi les jeux, citons la Moto infernale, le Fou volant, US Rallye, Glouton, le Monstre, Tortues, Football, l'Abeille, Bris de glace, Citadelle, Viking, Backgammon, Reversi. Parmi les didacticiels, on compte deux cassettes d'initiation au BASIC; Compléments et multiples, ainsi que Mathématiques proposent des exercices dans cette matière; Clé des chants et la Carotte malicieuse sont destinés aux plus petits, Carte de France, Mots croisés, etc. complètent cette gamme de didacticiels, développés en collaboration avec l'éditeur Vifi-Nathan, spécialisé dans l'édition de logiciels éducatifs. Vous trouverez également des logiciels utilitaires tels que Carnet d'adresses, Fichier, Budget et bien d'autres titres en préparation. Des cartouches extension permettront bientôt d'utiliser de façon souple et performante des logiciels de jeux très sophistiqués, des programmes de saisie ou de traitement de texte, des didacticiels, etc.

RAM 8 K vidéo

Platine principale

Philips

Dites-le avec des touches

Le clavier AZERTY du VG 5000 possède soixante-trois touches mécaniques comprenant les touches alphanumériques classiques (majuscules ou minuscules par utilisation des touches CAPS ou SHIFT), trente-trois fonctions BASIC préprogrammées accessibles par la touche CTRL, dix minuscules accentuées accessibles par la touche CTRL/ACCENT et reprogrammables par l'utilisateur, des touches arithmétiques, d'édition, de gestion de curseur.

Philips

VG 5000 de Philips

PRIX

★ ★

DIMENSIONS

280 × 210 × 42 mm.

POIDS

820 g.

UC

Z80, 4 MHz.

MÉMOIRE

24 K RAM,18 K ROM.

AFFICHAGE ÉCRAN

Texte : 25 lignes, 40 colonnes; Graphique : 80 000 points; 8 couleurs; 110 caractères ASCII texte, 128 graphiques, 192 caractères définissables par l'utilisateur.

INTERFACES

Port d'entrée/sortie pour lecteur-enregistreur de cassettes (prises DIN 8 broches). Sortie vidéo pour raccordement à la prise Péritel d'un téléviseur ou pour raccordement d'un codeur-modulateur SECAM (prise DIN 8 broches). Connecteur bord de carte de 2 × 25 voies permettant le raccordement des extensions (entrée/sortie bus parallèle). En option : interface poignée permettant l'utilisation de deux poignées de jeux.

LANGAGES DISPONIBLES

BASIC Microsoft, LOGO (en cartouche).

CLAVIER

AZERTY, soixante-trois touches mécaniques comprenant les touches alphanumériques, trente-trois touches de fonctions BASIC préprogrammées accessibles par CTRL, dix minuscules accentuées accessibles par CTRL/ACCENT et reprogrammables par l'utilisateur, des touches arithmétiques, d'édition et de gestion de curseur.

DOCUMENTATION

Manuel d'utilisation en français, de quatre-vingt-seize pages, reprenant l'ensemble des instructions BASIC et des exemples de programmes.

POINTS FORTS

Microprocesseur à 4 MHz, donc très rapide. Bonne version de BASIC.

POINTS FAIBLES

Les minuscules accentuées sont difficilement accessibles, ce qui compromet l'utilisation en traitement de texte.

Faire des plans

Malgré de nombreuses ressemblances, les logiciels de modèles financiers gardent chacun leurs particularités. C'est le cas de Graph Plan, qui combine tableur et logiciel graphique.

Contrairement aux autres progiciels étudiés jusqu'ici, Graph Plan est un programme résidant sur disque. Acorn fournit gratuitement ce logiciel aux acheteurs d'un ensemble de programmes destinés au deuxième processeur Z80. Comme la plupart des programmes rencontrés dans cet ensemble, Graph Plan apparaît comme un progiciel particulièrement pratique et « solide ».

Graph Plan n'a pas autant de style qu'Abacus, de Psion, le tableur/logiciel graphique donné aux utilisateurs du QL de Sinclair. Mais il a l'avantage énorme de permettre l'accès et les sauvegardes rapides, grâce à sa présence sur disque, alors que le système basé sur cassette du QL reste désespérément lent.

Ajoutons encore que cette forme « spéciale » de vente sur le marché devrait lui assurer une grande popularité, même si l'addition des coûts du processeur Z80 et du lecteur de disques Acorn peut dissuader certains acheteurs potentiels.

Nous étudierons dans cet article essentiellement l'aspect graphique de ce générateur de modèles financiers. Graph Plan, comme son nom l'indique, dispose de ressources graphiques complètes, ainsi que d'une très vaste gamme de fonctions de calcul (formules mathématiques et commerciales).

Le style original de Graph Plan vient de sa manière très particulière de communiquer au moyen des nombres. Toutes les interactions de l'utilisateur avec le programme passent par cent quarante-quatre commandes numérotées. Au chargement, l'affichage est celui d'un tableur standard, avec des lignes et des colonnes, sur tout l'écran ou presque. Dans l'angle inférieur droit, les vingt commandes de base sont affichées avec leur numéro correspondant. L'utilisateur choisit une commande et frappe son numéro en réponse au message ENTREZ VOTRE COMMANDE qui s'affiche sur la troisième ligne des « états » en haut de l'écran.

S'il paraît très simple de choisir une commande, soit depuis le menu à l'écran, soit depuis la liste complète des commandes de l'excellent manuel de cent vingt-quatre pages, cette méthode comporte plusieurs inconvénients manifestes. La plupart des tableurs, et notamment les plus réputés comme Lotus 1-2-3, vous demandent simplement la première lettre de la commande à entrer (ou encore de valider un choix dans un menu fléché). Les progiciels comme Lotus 1-2-3 affichent des informations sur la fonction de chaque commande. Par contre, Graph Plan suppose que vous connaissez la fonction de toutes les commandes, même si le programme permet l'affichage de la liste des commandes détaillées à la place de la liste de commandes, en bas et à droite de l'écran. Si vous choisissez, par exemple, la commande numéro 2, données, le menu des commandes change pour afficher les commandes 29 à 48 (les commandes de saisie et de manipulation de données). Il existe également une commande d'AIDE (commande numéro 7) destinée à fournir des explications sur une commande déterminée.

Outre son système de commandes numérotées, Graph Plan comporte certaines caractéristiques uniques. La plupart des tableurs, par exemple, reposent sur la conception des cases intersections entre les lignes et les colonnes. Graph Plan, pour sa part, traite les lignes et les colonnes comme des entités distinctes. Le pointeur de données, sur la deuxième ligne d'état, représente l'indicateur de la position du curseur. Sa fonction est d'identifier le carré courant du curseur et d'indiquer le mode courant, colonne ou ligne.

Cette distinction serait sans signification pour un système où la seule adresse de la position est le point de référence centrale. Mais elle est de toute première importance pour Graph Plan, pour lequel les points d'une courbe ou d'un graphique doivent être spécifiés soit en référence aux colonnes, soit en référence aux lignes. Pour dire au système que vous choisissez le mode graphique colonne ou le mode graphique ligne, vous devez modifier le pointeur de données. Vous utilisez pour cela les touches fléchées pour placer le curseur en haut d'une colonne ou au début d'une ligne, ce qui précise automatiquement le mode correspondant.

Un modèle simple

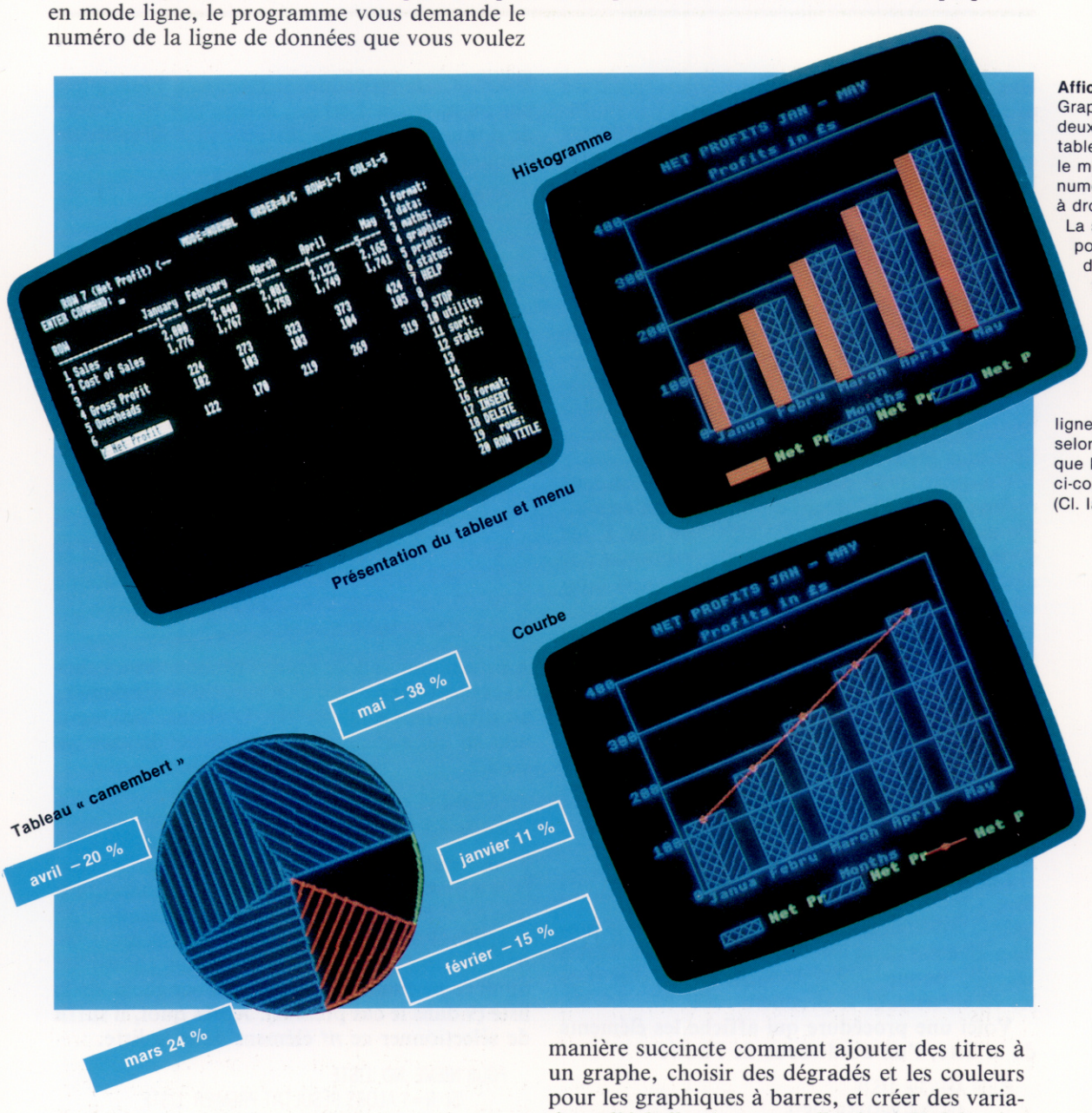
Pour illustrer la technique graphique du logiciel, prenons le cas d'un modèle de tableau simple. Il comporte cinq colonnes dont les en-têtes représentent les mois de janvier à mai, et cinq lignes, Ventes, Coûts, Profit brut, Frais généraux et Profit net. Pour un tableau de ce type, le mode ligne et le mode colonne signifieront des choses différentes. Ainsi, on pourra créer un tableau très simple en mode ligne pour obtenir le chiffre d'affaires de janvier à mai.

Le tableau affichera les en-têtes de colonnes (JAN à MAI) sur l'axe des x, et les barres graphiques seront mises pour les valeurs données sur la première ligne. Un tableau en mode colonne générerait un graphique très différent. La ligne des titres Ventes, Coûts, etc. serait l'axe des x, les barres graphiques représentant les valeurs

étant en colonne 1 (janvier). A partir des données appropriées, Graph Plan peut donc générer de très nombreux graphes. Tous ces tableaux peuvent être affichés successivement à l'écran, sans d'autre intervention de la part de l'utilisateur. Graph Plan utilise pour cela la commande 62 CHOISIR. Si vous répondez 62 au message ENTREZ VOTRE COMMANDE, un autre message s'affiche à la troisième ligne d'état. Si vous êtes par exemple en mode ligne, le programme vous demande le numéro de la ligne de données que vous voulez voir représentée sous forme de graphe. Vous choisissez une ligne, et le programme vous demande le type de graphe que vous désirez : « Choisissez (A barres = 1, A ligne = 2, A «camembert» = 3 ». Vous pouvez alors visionner immédiatement le graphe par la commande 61 AFFICHER. Et vous pouvez passer à volonté d'un graphe à barres à un graphe à ligne ou à « camembert ».

Si vous voulez plus de souplesse dans la disposition et dans la conception d'un graphe, utilisez la commande 63 OPTIONS. Le manuel montre un très bon exemple de table de décision qui expose très bien le processus à suivre avec cette commande. La commande 63 vous présente un menu avec six options : Afficher la table, Définir les options pour la table, Définir les options pour les axes, Définir les options pour le « camembert », Impression de la table et Tracé de la table. Le manuel comprend en outre une annexe spéciale, Guide des sous-menus graphiques, à lire absolument pour un novice. Cette annexe explique de manière succincte comment ajouter des titres à un graphe, choisir des dégradés et les couleurs pour les graphiques à barres, et créer des variations d'échelles (pouvant aller jusqu'à faire des axes logarithmiques et non linéaires).

De bonnes caractéristiques graphiques incorporées à ce tableur (qui comprend lui-même des fonctions mathématiques et statistiques) font de Graph Plan un outil bien adapté à une vaste gamme d'applications relativement simples dans les domaines scientifiques et d'ingénierie. Le progiciel fournit d'excellentes présentations des données pour des rapports ou des conférences. Il s'adresse donc à des techniciens, à des scientifiques, tout aussi bien qu'à des commerciaux.



Affichage tableur
Graph Plan diffère de deux manières des autres tableurs. La première est le menu de commandes numérotées qui s'affiche à droite de l'écran. La seconde est la possibilité de prendre des données au tableur et de les présenter sous la forme d'un graphe à l'écran. Graph Plan peut afficher des données en provenance d'une ligne ou d'une colonne selon trois formats, ainsi que le montre la figure ci-contre.
(Cl. Ian McKinnell.)

Poésie en mouvement

Le traitement de liste représente une notion centrale pour la compréhension de LOGO. En ajoutant une pincée de récursion, nous allons écrire un peu de poésie avec LOGO.

Une liste est un ensemble d'objets logiques ordonnés; elle est identifiée en LOGO par l'utilisation de crochets. Aussi, [CEYLAN MADRAS VINDALOO] est une liste. Nous avons déjà rencontré des listes. Il est impossible de ne pas en rencontrer avec LOGO, ce langage étant fondé sur cette notion. Nous avons vu comment la définition d'un carré — RÉPÈTE 4 [AV 50 DR 90] — comprend une liste d'instructions (entre crochets), comme deuxième élément de saisie. De même, FAIT « SAISIE DEMANDE assigne à SAISIE une liste constituée de la saisie à partir du clavier.

Les listes peuvent être assignées à des variables globales, par exemple : FAIT «CURRY [CEYLAN MADRAS VINDALOO]. La commande AFFICHE :CURRY affiche la liste sans les crochets.

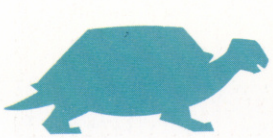
Un objet LOGO peut être un nombre, un mot où une liste. Une liste est définie comme un simple ensemble d'objets. Il s'agit bien sûr d'une définition récursive : une liste peut comporter une autre liste, ou encore une liste de listes, et ainsi de suite. [[BOUILLABAISSE] SALADE NIÇOISE] est une liste autorisée dont le premier élément est une liste ([BOUILLABAISSE]).

La plus grande partie de la programmation que nous avons effectuée avec LOGO a été consacrée jusqu'ici à un nombre ou à un mot à la fois. Lorsque nous voulons traiter simultanément des groupes d'objets logiques, il faut les regrouper en une seule unité. LOGO considère la liste comme sa méthode de base de regroupement des objets logiques simples.

Les deux opérations fondamentales sur une liste sont PREMIER et SAUFPREMIER. PREMIER [CEYLAN MADRAS VINDALOO] donne CEYLAN, c'est-à-dire le premier élément de la liste. SAUFPREMIER [CEYLAN MADRAS VINDALOO] donne MADRAS VINDALOO, c'est-à-dire tous les éléments sauf le premier.

Voici une procédure qui affiche les éléments de la liste, l'un au-dessous de l'autre :

```
POUR AFFICHE :LISTE
      AFFICHE PREMIER :LISTE
      AFFICHE SAUFPREMIER :LISTE
FIN

Aussi, AFFICHE [CEYLAN MADRAS VINDALOO] donne :
CEYLAN
MADRAS
VINDALOO
```



La première commande affiche le premier élément et passe la main, pour la poursuite du reste de l'affichage, à un autre appel de la commande AFFICHE. Cette procédure se termine à l'exécution par un message d'erreur, lorsqu'elle est à court de données. Voici une manière plus élégante de terminer :

```
POUR AFFICHE :LISTE
      SI VIDE? :LISTE ALORS STOP
      AFFICHE PREMIER :LISTE
      AFFICHE SAUFPREMIER :LISTE
FIN
```

VIDE? vérifie si les données en entrée correspondent à la « liste vide » –[]. Certaines versions MIT ne comportent pas la primitive VIDE?, mais vous pouvez la définir de la sorte :

```
POUR VIDE? :LISTE
      SI :LISTE = [] ALORS RÉSULTAT «VRAI
      RÉSULTAT «FAUX
FIN
```

DERNIER et SAUFDERNIER sont similaires à PREMIER et SAUFPREMIER. DERNIER [CEYLAN MADRAS VINDALOO] donne VINDALOO, et SAUFDERNIER [CEYLAN MADRAS VIDALOO] donne CEYLAN MADRAS.

A présent nous allons essayer d'imiter les « babillages aléatoires » que l'on associe généralement au divan de l'analyste (!). D'abord, assignons tous les mots que nous connaissons à la variable MOTS :

```
FAIT «MOTS [MÈRE PÈRE SEXE MEURTRE
JALOUSIE FEU LA.MER LA.MORT RÊVER]
```

Nous voulons susciter un débit de parole constant à partir de ces mots; l'expérience montrant que ce sont les mots qui attirent l'attention du psychanalyste... Pour obtenir un élément au hasard sans cette liste, il nous faut choisir un nombre n, compris entre 1 et la longueur de la liste (9 dans le cas présent). Après quoi, il suffit de sélectionner ce n^e élément dans la liste.

```
POUR NIÈME :NO :LISTE
      SI :N=1 ALORS RÉSULTAT PREMIER :LISTE
      RÉSULTAT NIÈME :NO–1 SAUFPREMIER :LISTE
FIN
```

Illustrons cette procédure de quelques exemples. Si vous tapez NIÈME 1 :MOTS, l'expression conditionnelle de la première ligne est vraie et la procédure donne PREMIER :MOTS, ce qui correspond ici à MÈRE. Avec NIÈME 2 :MOTS, l'expression conditionnelle est fausse, et la procédure donne NIÈME 1 SAUFPREMIER :MOTS. Ce qui signifie que le premier élément a été ignoré au profit du premier mot dans le reste de la liste, PÈRE.

Notre procédure destinée à afficher un mot à partir de notre lexique sera donc :

```
POUR PRENDREHASARD :LISTE
      :RÉSULTAT NIÈME ((HASARD 9) + 1 :LISTE)
FIN
```

Pour l'utiliser, tapez PRENDREHASARD :MOTS.

Notre procédure se limite à des listes de neuf éléments. Nous pourrions l'améliorer si nous pouvions déterminer combien il y a d'éléments dans une liste donnée. Voici une procédure à cette fin :

```
POUR LONGUEUR :LISTE

      SI VIDE? :LISTE ALORS RÉSULTAT 0
      RÉSULTAT 1 + LONGUEUR SAUFPREMIER :LISTE
FIN
```

Pour expérimenter cette procédure, faites LONGUEUR [SCIENCE FICTION]. Comme la liste contient des mots, la première condition n'est pas vérifiée et la procédure donne 1 + LONGUEUR [FICTION]. LONGUEUR [FICTION] donnera 1 + LONGUEUR []. A l'appel de LONGUEUR avec une donnée égale à [], la condition à la ligne 1 est vérifiée et la procédure donne 0. LONGUEUR [FICTION] donne 0 + 1 = 1, et finalement, LONGUEUR [SCIENCE FICTION] donne 1 + 1 = 2. Une procédure plus générale pour prendre des mots dans une liste de toute longueur est :

```
POUR PRENDREHASARD :LISTE
      RÉSULTAT NIÈME ((HASARD LONGUEUR :LISTE) + 1
      : LISTE
FIN
```

De nombreuses versions de LOGO ont une primitive, ÉLÉMENT, qui a la même fonction que NIÈME, et une primitive appelée COMPTE, semblable à LONGUEUR. Nous pouvons réécrire la procédure :

```
POUR PRENDREHASARD :LISTE
      RÉSULTAT ÉLÉMENT ((HASARD COMPTE :LISTE) + 1)
      :LISTE
FIN
```

Pour afficher dix mots qui tiendront votre psychanalyste en éveil, tapez simplement :

```
RÉPÈTE 10 [AFFICHE PRENDREHASARD :MOTS]
```

La structure de ces programmes de gestion de listes est la même que celle de nombreux programmes graphiques récursifs que nous avons déjà utilisés pour la tortue. En voici la structure :

- Si la tâche à effectuer est extrêmement simple, faites-la et arrêtez-vous.
- Sinon, faites-en une petite partie.
- Puis transmettez le reste à une autre procédure (souvent un double de la procédure d'origine).

Cette stratégie est très rentable et nous la rencontrerons partout pour la gestion des listes. Comparez ce programme de dessin d'un polygone :

```
POUR POLY :N
      SI :N = 0 ALORS STOP
      AV 30 DR (360 / :N)
      POLY :N - 1
FIN
```

avec la version AFFICHER donnée plus haut. La structure des deux procédures est identique. Essayons de nous exercer à la poésie... Il nous faudra, cette fois, produire des phrases entières et non plus des mots isolés.

```
POUR POÈME1 :LONGUEUR
      SI :LONGUEUR = 0 ALORS AFFICHE «STOP
      (AFFICHE1 «" PRENDREHASARD :MOTS)
      POÈME1 :LONGUEUR - 1
FIN
```

AFFICHE1 «" est destiné à mettre un espace entre les mots. Pour utiliser cette procédure, tapez POÈME 1 6, pour une phrase à six mots.

Il serait utile de pouvoir augmenter notre liste de mots sans avoir à la réécrire en entier. Une

Abréviations

SAUFPREMIER	SP
SAUFDERNIER	SD
PHRASE	PH

Sir John Tenniel

Chant de la fausse tortue
Un extrait du « Chant de la Tortue railleuse » d'*Alice au pays des merveilles* de Lewis Carroll. Le rythme métrique, difficile à retrouver sur ordinateur, provient d'une vieille chanson populaire et est également utilisé par Mary Howitt dans le célèbre poème *L'Araignée et la Mouche.*

« Le merlan dit au bigorneau "Pourriez-vous
vous presser un peu ?
Il y a, là derrière, un gros thon qui me marche
sur la queue.
Voyez avec quelle ardeur les homards
et les tortues s'avancent !
Ils attendent sur les galets... Voulez-vous entrer
dans la danse ?
Voulez-vous, ne voulez-vous pas, voulez-vous, ne
voulez-vous pas, voulez-vous entrer dans la
danse ?
Voulez-vous, ne voulez-vous pas, voulez-vous,
ne voulez-vous pas, voulez-vous entrer dans
la danse ?

Vous n'avez pas la moindre idée du plaisir que
cela peut faire
Lorsqu'on vous prend et qu'on vous jette, avec
les homards, à la mer !"
Le bigorneau répondit "Trop loin, trop loin !"
et, le toisant avec méfiance,
Dit qu'il remerciait le merlan mais qu'il ne
voulait pas entrer dans la danse.
Ne voulait pas, ne pouvait pas, ne voulait pas,
ne pouvait pas, ne pouvait entrer dans la danse.
Ne voulait pas, ne pouvait pas, ne voulait pas,
ne pouvait pas, ne pouvait entrer dans la danse ».

manière d'augmenter une liste est d'utiliser l'opération logique PHRASE, qui prend deux entrées et les listes. Aussi : PHRASE «CONFITURE [JARRE DE MIEL] donne [CONFITURE JARRE DE MIEL].

```
POUR MOTSAJOUTÉS1 :LISTE
    FAIT «MOTS PHRASE :LISTE :MOTS
FIN
```

Nous pouvons maintenant augmenter MOTS, de MOTSAJOUTÉS [ANXIÉTÉ RÉPRESSION [PEUR DE L'AVION]]. Mais la variable MOTS n'a pas reçu de valeur. La primitive QUELQUECHOSE? est conçue à cet effet, elle teste si une variable a reçu une valeur. Elle donne un résultat « vrai » lorsque l'entrée comporte une valeur associée. Nous pouvons maintenant augmenter notre liste des mots supplémentaires de MOTSAJOUTÉS1 :

```
POUR MOTSAJOUTÉS1 : LISTE
    SI NON QUELQUECHOSE? «MOTS ALORS FAIT «MOTS
      [ ]
    FAIT «MOTS PHRASE :LISTE :MOTS
FIN
```

En utilisant une autre liste de mots, avec cette procédure, nous obtenons le « poème » suivant :

```
APPARITION BRUYAMMENT PARLE SPLENDIDE PARANOÏDE
PLANÈTE TERRIFIE LE AVEC VERTE APPARITION FLOTTANT
ROBOT PARANOÏDE L'HOMME VOLA PARLA FLOTTANT
LOURDEMENT
```

L'une des lacunes les plus évidentes de notre poésie sur ordinateur est son manque total de cohérence vis-à-vis de la grammaire. Les poèmes auraient davantage de sens s'ils respectaient une syntaxe simple, du type : sujet-verbe-complément. Une manière d'y parvenir est d'avoir plusieurs listes, une par genre grammatical. Il suffirait alors de prendre successivement un mot par genre et dans l'ordre d'enchaînement des genres.

Nous vous laissons étudier la question et nous verrons ultérieurement comment améliorer les qualités littéraires de la tortue...

Variantes de logo

Certaines versions LOGO MIT n'ont pas VIDE?, ÉLÉMENT ni COMPTE.
Pour toutes les versions LCSI, utilisez :

VIDEP pour VIDE?
LISTEP pour LISTE?
TAPE pour AFFICHE1

La primitive ÉGALESP teste si deux entrées sont égales. Vous l'utiliserez pour comparer des listes et des mots, à la place du signe égal (=). Le signe égal existe pour certaines versions LCSI.
Souvenez-vous de la syntaxe particulière SI :

SI VIDEP :LISTE [RÉSULTAT 0]

Pour LOGO ATARI, utiliser PH pour PHRASE et sachez qu'ÉLÉMENT n'existe pas.

Exercices

1. Écrire une procédure qui affiche une liste à l'envers (utilisez DERNIER et SAUFDERNIER). Modifiez-la afin qu'elle redonne la liste dans le bon ordre.
2. Écrire une procédure qui retire un élément d'une liste, afin que SUPPRIME «NOURRITURE [BOIRE NOURRITURE] donne [BOIRE], et SUPPRIME «VIN [BOIRE NOURRITURE] donne [BOIRE NOURRITURE].

Réponses aux exercices

Réponses aux exercices précédents :

1. Calcul des puissances :

```
POUR PUISSANCE :A :N
    SI NON ((ENTIER :N) = :N) ALORS AFFICHE
    [INDICES NOMBRES ENTIERS SEULEMENT STOP
    SI :N =0 ALORS RÉSULTAT 1
    RÉSULTAT :A * PUISSANCE :A – 1
FIN
```

2. Conversion en hexadécimal

```
POUR AFF.HEX :NO
    SI :NO<10 ALORS RÉSULTAT :NO
    SI :NO=10 ALORS RÉSULTAT «A
    SI :NO=11 ALORS RÉSULTAT «B
    SI :NO=12 ALORS RÉSULTAT «C
    SI :NO=13 ALORS RÉSULTAT «D
    SI :NO=14 ALORS RÉSULTAT «E
    SI :NO=15 ALORS RÉSULTAT «F
FIN

POUR HEX :NO
    SI :NO=0 ALORS STOP
    HEX QUOTIENT :NO 16
    AFFICHE1 AFF.HEX RESTE: NO 16
FIN
```

3. Test pour savoir si un nombre est pair

```
POUR PAIR? :NO
    SI ((RESTE :NO 2)=0) ALORS RÉSULTAT
    «VRAI RÉSULTAT «FAUX
FIN
```

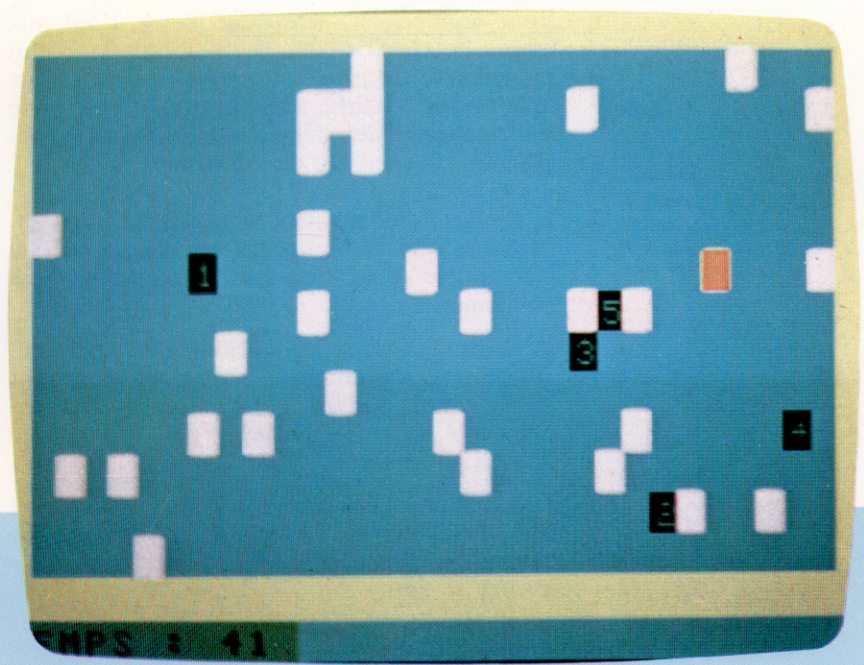
4. Trouver une surface par la méthode de Monte-Carlo

```
POUR MC
    TRACE CL FAIRE «DEDANS 0
    MC1 1000 10 100
    (AFFICHE [LA SURFACE EST] ( :DEDANS))
FIN

POUR MC1 :NO :XNO :YNO
    SI :NO=0 ALORS STOP
    POINT.HASARD :XNO :YNO
    SI :DEDANS? ALORS FAIRE «DEDANS
    :DEDANS + 1
    MC1 :NO–1 :XNO :YNO
FIN

POUR POINT.HASARD :XNO :YNO
    DONNEXY HASARD :XNO HASARD :YNO
FIN

POUR DEDANS?
    SI YCOR<XCOR ALORS RÉSULTAT
    «VRAI
    RÉSULTAT «FAUX
FIN
```

Chiffres sur Alice

Voici un jeu d'adresse plus difficile qu'il n'y paraît. Son auteur, Pierre Monsaut, l'a conçu pour le micro-ordinateur Alice, le premier de la génération de chez Matra.

Vous allez tenter de marquer le plus grand nombre de points possible en effaçant à l'aide du carré rouge les chiffres affichés par l'ordinateur. Les touches à utiliser sont W (bas), Q (gauche) et S (droite). La barre d'espacement vous permet de vous arrêter. *Attention :* vous devez effacer les chiffres dans l'ordre (du plus petit au plus grand) en évitant les obstacles placés au hasard sur l'écran, et ceci en un temps limité. (Le temps qui vous reste est affiché sur la dernière ligne.) Lorsque tous les chiffres sont effacés, le jeu reprend avec un chiffre de plus. Au-delà de neuf chiffres, une difficulté supplémentaire s'ajoute : vous devez effacer des symboles en vous souvenant de l'ordre dans lequel ils sont apparus (à moins que vous ne connaissiez le code ASCII par cœur !).




```
5 REM  ************
10 REM * CHIFFRES *
15 REM ************
20 S=0
30 X=0
40 P$=CHR$(191)
50 GOSUB 2000
100 FOR I=1 TO X
110 D$=INKEY$
120 D=(D$="Q")-(D$="S")+32*((D$=
"Z")-(D$="W"))
130 IF D<>0 THEN D0=D
140 IF D$=" " THEN D0=0
150 T=T-0.1
160 PRINT@ 480,"TEMPS :";INT(T+1
);
170 IF T<0 THEN 500
180 P=P+D0
190 C=PEEK(16384+P)
200 IF C=I+48 THEN S=S+I:SOUND 1
,1:GOTO 260
210 IF C<>223 THEN P=P1
220 PRINT@ P1,CHR$(223);
230 PRINT@ P,P$;
240 P1=P
250 GOTO 110
260 PRINT@ P1,CHR$(223);
270 PRINT@ P,P$;
280 P1=P
290 NEXT I
300 GOSUB 2000
310 GOTO 100
500 IF R<S THEN R=S
510 PRINT@ 166,"TEMPS ECOULE";
520 PRINT@ 234,"SCORE :";S;
530 PRINT@ 266,"RECORD :";R;
540 PRINT@ 326,"UNE AUTRE ?";
550 D$=INKEY$
560 IF D$="" THEN 550
570 IF D$<>"N" THEN 20
580 END
2000 CLS 6
2010 X=X+1
2020 FOR I=0 TO 31
2030 PRINT@ I,CHR$(159);
2040 PRINT@ 448+I,CHR$(159);
2050 NEXT I
2060 FOR I=1 TO 13
2070 PRINT@ I*32,CHR$(159);
2080 PRINT@ I*32+31,CHR$(159);
2090 NEXT I
2100 FOR I=1 TO 30
2110 GOSUB 3000
2130 PRINT@ P,CHR$(239);
2140 NEXT I
2150 FOR I=1 TO X
2160 GOSUB 3000
2180 POKE 16384+P,I+48
2190 NEXT I
2200 GOSUB 3000
2210 PRINT@ P,P$;
2220 P1=P
2230 T=50
2240 D0=0
2250 RETURN
3000 P=RND(414)+32
3010 IF PEEK(16384+P)<>223 THEN
3000
3020 RETURN
```

Chasse aux bogues

Pour illustrer les techniques de conception « de haut en bas » en langage d'assemblage, voici un programme de « débogage ». La première chose à faire est de développer le module de contrôle.

B Insérer point d'interruption
U Supprimer (déplacer) point d'interruption
D Afficher points d'interruption en cours
S Démarrer le programme
G « Go » (repartir de là où le programme a été laissé)
R Afficher contenu des registres
M Inspecter et changer emplacement mémoire
Q Quitter

Pour commencer, jetons un coup d'œil aux étapes de spécification et de conception d'un programme « débogueur ». La spécification est relativement simple; nous avons déjà examiné les fonctions que nous attendons d'un tel programme. Les entrées du débogueur seront :

1. *Un programme pour « déboguer » :* nous supposerons que le débogueur est chargé avec le programme — qu'il est censé déboguer — déjà en mémoire.

2: *Commandes :* il faut décider si les commandes doivent être entrées directement ou en tant qu'options d'un menu. Nous entrerons des commandes à un seul caractère à partir de la liste donnée en marge.

3. *Adresses :* elles devront vraisemblablement être entrées en hex; il faudra donc convertir une chaîne de chiffres hex ASCII en un nombre binaire 16 bits.

Les sorties du débogueur seront :

1. *Les « échos » des caractères entrés :* rappelez-vous que les touches tapées ne génèrent pas automatiquement de caractères à l'écran — l'ordinateur doit être programmé pour cela (c'est ce qu'on appelle l'« écho »).

2. *Des nombres à 8 et 16 bits :* ils sont acceptés comme chaînes de chiffres hex.

3. *Des chaînes :* elles sont utilisées pour étiqueter les précédents.

Il y a bien des façons de découper un programme en modules, puis en sous-programmes, mais il doit toujours y avoir un module externe qui relie ensemble tous les autres. Pour notre programme de « débogage », il prendra la forme suivante :

Le module principal

Données :
Adresse de début du programme (16 bits).
Demande d'entrée de commande (un seul caractère ASCII « > »).
Caractère de commande : un seul caractère ASCII (les minuscules sont-elles autorisées?)
Adresse d'interruption? adresse de la routine effectuant l'interruption SWI.

Traitement :
Met l'interruption
ENTRE adresse de début
RÉPÈTE
    AFFICHE demande
RÉPÈTE
    ENTRE commande
JUSQUE commande valide
AFFICHE commande (écho)
SI commande = « B » ALORS
    Insère point d'interruption
SINON SI commande = « U » ALORS
    Enlève point d'interruption
SINON SI...
JUSQUE commande = « Q »
Fin du module principal

Cela nous donne une bonne idée des routines dont on aura besoin. Un module n'est pas identique à un sous-programme. De toute évidence, il y a différents sous-programmes qui s'associent logiquement en groupes, en partageant les données. Un tel module, par exemple, doit manier des points d'interruption. Le stade suivant montre comment nous pourrions concevoir un tel module.

Points d'interruption du module

Données :
Tables-d'interruptions : c'est un tableau d'adresses 16 bits où peuvent être stockées les adresses des interruptions.

Valeurs-déplacées : c'est un tableau de valeurs 8 bits correspondant à la table ci-dessus. Les opc qui sont remplacés par une instruction SWI à l'interruption peuvent être stockés.

Nombre-d'interruptions : valeur 8 bits contenant le nombre d'interruption actives.
valeur 8 bits qui contient la prochaine interruption qui sera rencontrée opc-SWI : opc à 8 bits pour l'instruction SWI.

Traitement 1 : insertion d'interruptions SI nombre-d'interruptions < MAX ALORS ENTRER adresse.
Additionner 1 à nombre-d'interruptions. Stocker adresse dans table d'interruption (Nombre-d'interruptions).
FINSI
Fin du traitement 1

Traitement 2 : Met-interruptions (N)
(N nous dit quel point d'interruption du tableau doit être mis.)
ENTRER adresse dans table-d'interruptions (N)
ENTRER opc à cette adresse
STOCKER opc dans valeurs-déplacées (N)
STOCKER opc-SWI à adresse
Fin de traitement 2

Le traitement 2 en est au stade où nous pouvons commencer à le coder. Il y a quatre valeurs de données à manipuler. N, paramètre qui nous dit quelle interruption utiliser, est un nombre 8 bits compris entre 1 et (nombre-d'interruptions – 1), que nous utilisons comme décalé dans les deux tableaux. Notez toutefois qu'un tableau est formé de valeurs 16 bits et l'autre de 8 bits. Nous supposerons que N est passé en A. L'adresse du point d'interruption obtenu à partir de cette table sera mis en X. L'opc déplacé sera mis en B pour transfert vers la table valeurs-déplacées. B peut alors servir à mettre l'opc SWI à l'adresse appropriée.

Nous donnons ici la forme finale codée du traitement 2 (Module-mettre-interruptions); il nous faudra ensuite développer un module pour manipuler les entrées et sortie. Comme vous l'avez vu à partir de la conception du débogueur, il y a de nombreuses tâches d'E/S à accomplir pour le programme. Pour l'instant nous supposerons l'existence de deux sous-programmes : INCH, qui entre un seul caractère dans le registre A à partir du clavier; et OUTCH, qui envoie un caractère à partir de A vers l'écran à la position actuelle du curseur. Les routines nécessaires sont :

1. **Entrecommande** : entrer la commande suivante à partir du clavier.
2. **Entradresse** : entrer une adresse hex (de un à quatre caractères de longueur) à partir du clavier.
3. **Entrevaleur** : entrer une valeur (un à deux caractères de longueur) pour modifier la valeur d'un emplacement mémoire.
4. **Affichevaleur** : afficher une valeur hex sur l'écran.
5. **Affichadresse** : affiche une adresse de 4 caractères hex à l'écran.

Notre approche illustre la différence entre les méthodes de haut en bas et de bas en haut. La première doit nous conduire à définir et à coder indépendamment ces opérations, pour terminer par un certain nombre de routines séparées qui font essentiellement la même chose. La seconde peut entraîner un gain de temps, d'effort et d'espace, simplement en écrivant quelques routines utiles qui servent dans de nombreuses circonstances. Voici ces routines :

GETCH : pour entrer un caractère dans A, en vérifiant par rapport à une liste de caractères valides (lettres de commande ou chiffres hex), répétant les caractères valides et ignorant les autres.
GETHX2 : pour utiliser GETCH pour entrer deux chiffres hex et les convertir en un nombre 8 bits.
GETHX4 : pour entrer quatre chiffres hex pour former un nombre 16 bits.
PUTHEX : pour afficher un nombre 8 bits comme deux chiffres hex. (Cela équivaut à afficher un nombre 16 bits.)
PUTCR : pour sortir un retour chariot (ou retour chariot et interligne si nécessaire).

Ces cinq routines doivent être développées à leur tour. Considérons la conception de GETCH.

Routine d'entrée de caractère (GETCH)

Données :
Inchar est une entrée de caractère ASCII à partir du clavier (dans A).
Valid-char comprend l'adresse 16 bits de la table des caractères valides.
Nombre-de-car.-valides est une valeur à 8 bits.
Car.-cherché est un compteur 8 bits.
Traitement :

```
RÉPÈTE
    ENTRE Inchar suivant
    Met Car.-cherché à (Nombre-de-car.-valides – 1)
    TANT QUE Valid-chars(car.-cherché)<>Inchar
      ET Car.-cherché >=0
      Décrémente Car.-cherché
JUSQUE Car.-cherché >=0
AFFICHE Inchar
```

Pour coder, nous devons utiliser A pour stocker Inchar, et la valeur 16 bits de Valid-Chars peut être mise en X. Le Nombre-de-car.-valides peut être mis en B, mais aura besoin d'y être maintenu, au moyen d'une pile. B peut servir pour Car.-cherché. Notez que B remettra le décalé dans la table, ce qui sera utile dans l'interprétation de la commande et la conversion hex.

Voici la forme finale de cette routine. Nous développerons ultérieurement les autres routines requises par le module d'entrée-sortie.

Routine GETCH

```
GETCH   PSHS    B       Sauvegarde B
REPTOO  BSR     INCH    Entre Inchar suivant
        LDB     1,S     Met Car.-cherché
        DECB            Soustrait 1 de décalé max
WHILOO  BLT     ENDWOO  Tant que Car.-cherché >=0
        CMPA    B,X     ET
        BEQ     ENDWOO  Inchar<> Valid-chars(Car.-cherché)

        DECB            Décrémente Car.-cherché
        BRA     WHILOO
ENDWOO  TSTB
UNTLOO  BLT     REPTOO  Jusque Car.-cherché >=0
        BSR     OUTCH   Affiche Inchar
        LEAS    1,S     Incrémente S pour « oublier » la
        RTS             valeur originale de B
```

Module Met-interruptions

```
                        Déclarations de données
BPTAB   RMB     32      Table-d'interruptions
REMTAB  RMB     16      Valeurs-déplacées
NUMPP   FCB     0       Nombre-d'interruptions
NEXTBP  FCB     0       Interruption-suivante
SWIOP   FCB     $3F     Opc-SWI
MAXBP   FCB     16      Nombre max. d'interruptions
                        Traitement 2 — Met-interruptions
BP02    PSHS    B,X     Sauvegarde les registres que
                        nous voulons modifier
        LSLA            Multiplie le décalé par 2
        LEAX    BPTAB,PCR  Adresse de base de la table
        LDX     A,X     Entre adresse dans table-
                        d'interruptions (N)
        LDB     ,X      Entre opc à cette adresse
        LSRA            Restaure A à valeur initiale
        STB     A,X     Stocke opc à valeur-déplacées (N)
        LDB     SWIOP,PCR  Entre opc-SWI
        STB     ,X      Le stocke à l'adresse
        PULS    B,X,PC  Restaure et retour
                        Fin de traitement 2
```

Alerte rouge

Les programmes de simulation de vol sont devenus chose courante, mais Flyerfox, de Tymac, a sur eux un avantage évident : il parle, et ce, sans avoir besoin de périphérique supplémentaire.

Les jeux pour ordinateurs ont fait bien des progrès depuis le vénérable Space Invaders. Mais seul le graphisme semble en avoir bénéficié : les programmeurs ont poussé jusqu'aux dernières limites les techniques de compression des données afin de loger en RAM un nombre toujours croissant d'écrans. Les remarquables capacités sonores de bien des micro-ordinateurs sont restées, dans l'intervalle, largement sous-exploitées.

La compagnie américaine Tymac vient de lancer sur le marché toute une série de jeux qui font usage de la synthèse de parole, sans qu'il y ait à utiliser une interface spécialisée. Le premier d'entre eux est Flyerfox, destiné en particulier au Commodore 64.

C'est un programme de simulation de vol : le joueur se retrouve aux commandes d'un chasseur escortant un jumbo-jet dans lequel a pris place un homme politique, et l'appareil survole un territoire disputé... des avions ennemis qui cherchent à l'abattre devront être détruits après divers combats. C'est là qu'intervient la synthèse de parole : le jumbo jet fait parvenir une série de messages au pilote du chasseur.

Le synthétiseur de parole occupe à lui seul près de 11 K du programme et stocke les données nécessaires à la recréation des phrases prononcées. Flyerfox emploie la méthode de codage dite « à prédiction linéaire » : les mots sont convertis en signaux numériques qui sont installés en RAM. Tel ou tel mot est ensuite, à la demande, extrait de la mémoire et reproduit par l'intermédiaire de la puce sonore SID.

Le graphisme est en haute résolution. L'écran montre la vue du ciel qu'on peut avoir lorsqu'on regarde à travers le cockpit de l'avion, au-dessus du tableau de bord. Celui-ci comporte un compas de navigation et un écran radar qui indique l'arrivée des avions ennemis, ce qui laisse au joueur un certain temps pour se préparer. De plus, deux lumières clignotantes, disposées de chaque côté de l'horizon artificiel, précisent si les attaquants sont au-dessus ou au-dessous du chasseur lui-même.

Parlez-moi...

Les combats aériens sont rapides et très réalistes. A chaque apparition ennemie sur l'écran, un bip d'avertissement se fait entendre, et le joueur doit manœuvrer le Flyerfox de façon que l'avion adverse se retrouve en plein centre de son réticule de visée.

Ce n'est pas chose facile, car l'appareil esquive et plonge à toute allure. Une fois qu'il est dans la ligne de tir, il faut ouvrir le feu en lâchant des missiles à infrarouges ; mais ils ne sont pas infaillibles, et la cible parviendra plus d'une fois à s'échapper.

Si le graphisme est de haute qualité, il n'est malheureusement pas très varié. Pour donner l'illusion du mouvement, les auteurs se sont bornés à modifier la forme des nuages, et le sol se réduit à une grille qui défile sous l'avion.

De plus, le jumbo-jet n'ajoute pas grand-chose au jeu. Il est même difficile de comprendre ce qu'il fait là — sauf allusion morbide au Boeing coréen. Il n'est visible que de l'arrière, et en cas d'attaque, le Flyerfox ne peut le dépasser en passant au-dessus de lui ! A la différence de tout appareil placé dans des circonstances analogues, il ne cherche même pas à s'échapper lorsqu'il est menacé.

En dépit de ce stupide côté « guerre froide », Flyerfox marque sans doute les débuts d'une nouvelle forme de jeu vidéo. On pensait depuis longtems à les doter de la parole ; voilà qui est fait, et de surcroît sans équipement supplémentaire. En dépit de ses limites évidentes, Flyerfox restera peut-être un précurseur en ce domaine.

Chevalier du ciel
Le pilote du Flyerfox, s'il veut repérer les assaillants, doit autant inspecter le ciel que consulter son tableau de bord. Les points sur l'écran radar indiquent des appareils ennemis (mais tous n'attaqueront pas). Leur hauteur relative est précisée par les deux carrés blancs placés de chaque côté de l'altimètre. Le compas de navigation aide à retrouver le jumbo-jet. (Cl. Ian McKinnell.)

L'attaque ennemie

Flyerfox : pour le Commodore 64.

Éditeur : Tymac Corporation.

Auteurs : Gregory Carbonaro, Charles Teufert, Ronald Pintus, Arthur Aspromadis.

Manche à balai : indispensable.

Format : disquette ou cassette.