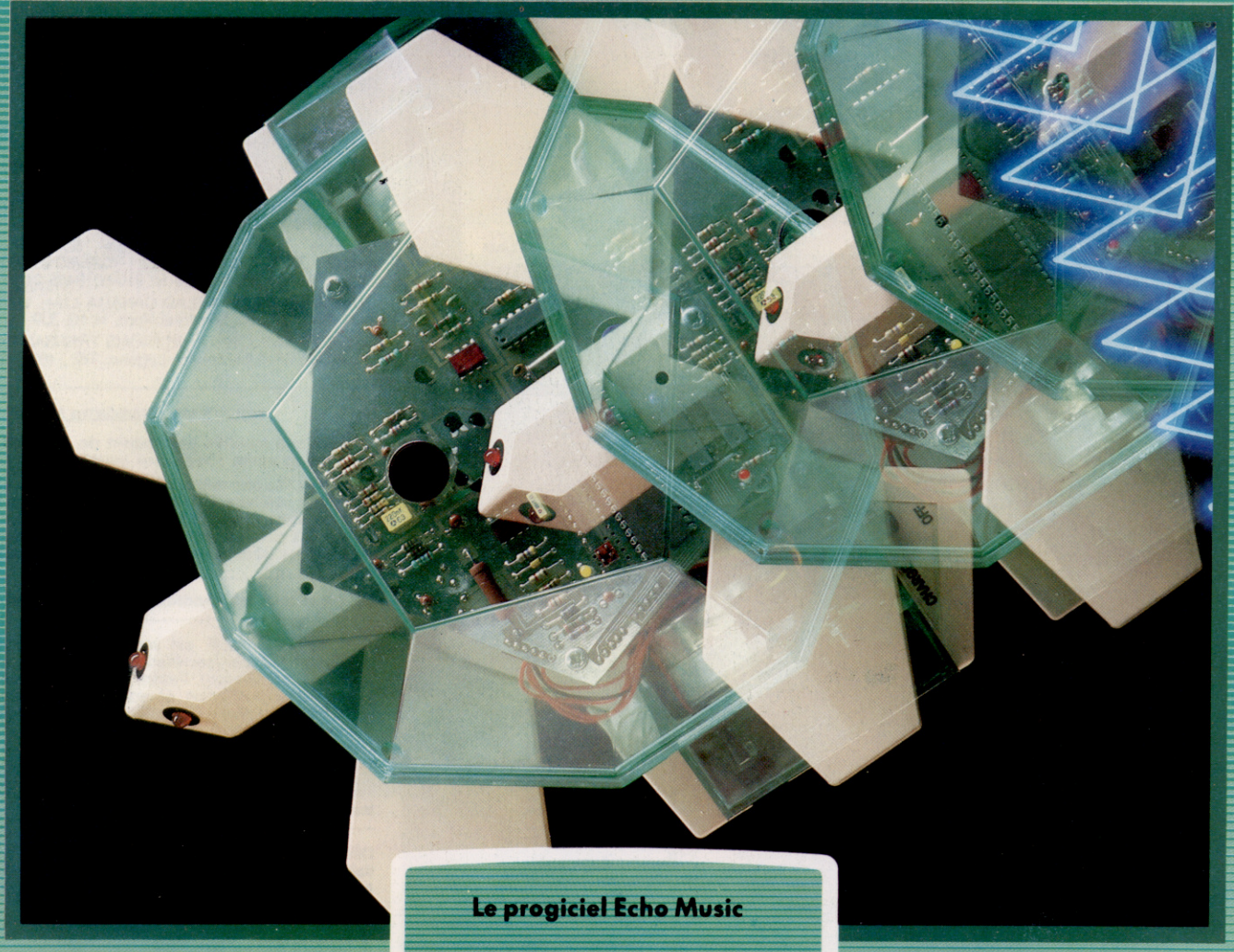
ABC INFORMATIQUE

N° 92

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Le progiciel Echo Music

Circuits suivis pour C64

Nouveau Monde (fin)

DCA pour Commodore

EDITIONS ATLAS

M 6062-92-12,50 F 90 FB - 3,80 FS - $ 1.95

Conservatoire

Le progiciel Echo Music, destiné au Commodore 64, tente de pallier l'absence de produits accédant directement à la puce sonore des micros. Mais ce logiciel laisse un peu à désirer.

L'ensemble Echo, de la société Leasalink Viewdata, est composé d'un véritable clavier à trois octaves et demie et du logiciel permettant de s'en servir. Un câble d'interface permet de connecter le clavier à l'ordinateur par le port utilisateur. Enfin, un manuel est également fourni à l'utilisateur.

Le matériel de l'Echo est d'une conception très simple : la plus grande partie du travail est accomplie par le logiciel fourni avec l'ensemble, ce qui diminue sans aucun doute les coûts de production.

Le logiciel comporte deux modules différents et le premier est un mode orgue. Il suffit d'appuyer sur une des touches du clavier pour sélectionner un des « instruments » listés. Ceux-ci vont de la guitare hawaiienne au violoncelle et au clavecin, même si les réglages de ces deux machines varient légèrement. De nombreux paramètres sont affichés en haut de l'écran ; cela vous permet, selon le paramètre choisi, de modifier le son de l'instrument que vous utilisez.

Notes
Le clavier de l'Echo, présenté sur cette photo, est le premier « véritable » clavier musical qui accède directement aux puces sonores des ordinateurs et qui ne nécessite pas une interface coûteuse.
(Cl. Paul Chave.)

La version Commodore 64 du logiciel offre la possibilité d'ajouter trémolo et vibrato (un à la fois), ou de passer en mode majeur ou mineur. Ces effets sont obtenus en appuyant sur une des touches du pavé numérique.

Dans d'autres versions, on utilise également les touches du micro, mais alors ce sont les touches de fonction qui vous permettent de sélectionner le trémolo et les modes majeur et mineur. Pour créer un effet particulier tout en jouant un morceau au clavier de l'Echo, vous pouvez utiliser l'une des touches de fonction pour solliciter le son approprié — marimbas, contrebasse, cymbales, etc. Malheureusement, aucune fonction ne permet de créer un accompagnement rythmique en jouant au clavier.

Commandes de tonalité

Au bas de chaque écran apparaissent les commandes de tonalité. Sur certaines versions, la tonalité peut être élevée ou abaissée en appuyant sur les touches appropriées du curseur. Cependant, sur celle du Commodore 64, cette fonction est effectuée au moyen des touches < et >. Malheureusement, on peut en dire long sur la qualité de ce logiciel en indiquant que ces deux fonctions portent l'étiquette PITCH MINUS sur l'écran !

Il est également malheureux que les sons produits par les touches instruments ne ressemblent guère aux sons qu'ils doivent imiter. Ce défaut se retrouve très souvent dans toute l'industrie de la musique électronique — pas seulement dans les programmes informatiques. Cependant le son de l'Echo semble particulièrement mauvais. Par exemple, le violon, la viole et le violoncelle sur la version Commodore produisent des sons qui ressemblent à ceux produits par un orgue à des octaves différentes selon les instruments. Ce n'est tout de même pas la même chose !

Le mode synthétiseur est aussi disponible. Les paramètres fournis par ce mode dépendent des fonctions offertes par la puce sonore. Ainsi, la version du Commodore 64 vous permet d'ajuster les paramètres d'enveloppe de la forme d'onde et de programmer le filtre. Comme en mode orgue, les paramètres sont modifiés en appuyant sur les touches appropriées sur le clavier de l'ordinateur jusqu'à ce que soit obtenue la valeur désirée.

De façon similaire, le mode synthétiseur sur le BBC Micro, par exemple, exploite la commande ENVELOPE de la machine. Lorsque cette commande est appelée, le programme vous demande lequel

SYNTHÉTISEUR ECHO

CLAVIER

Touches de piano ordinaires.

INTERFACES

Câble à 20 voies et adaptateur, qui se branchent dans le port utilisateur du Commodore 64.

LOGICIEL

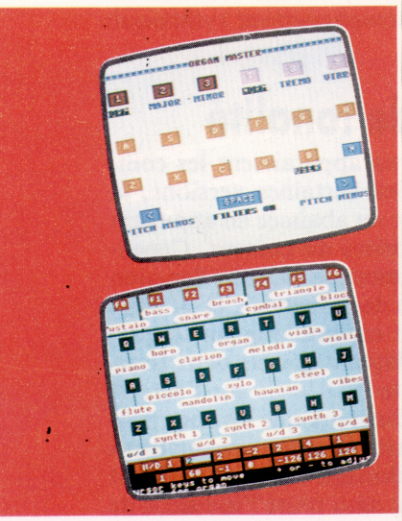
La version du C64 est fournie uniquement sur cassette.

FORCES

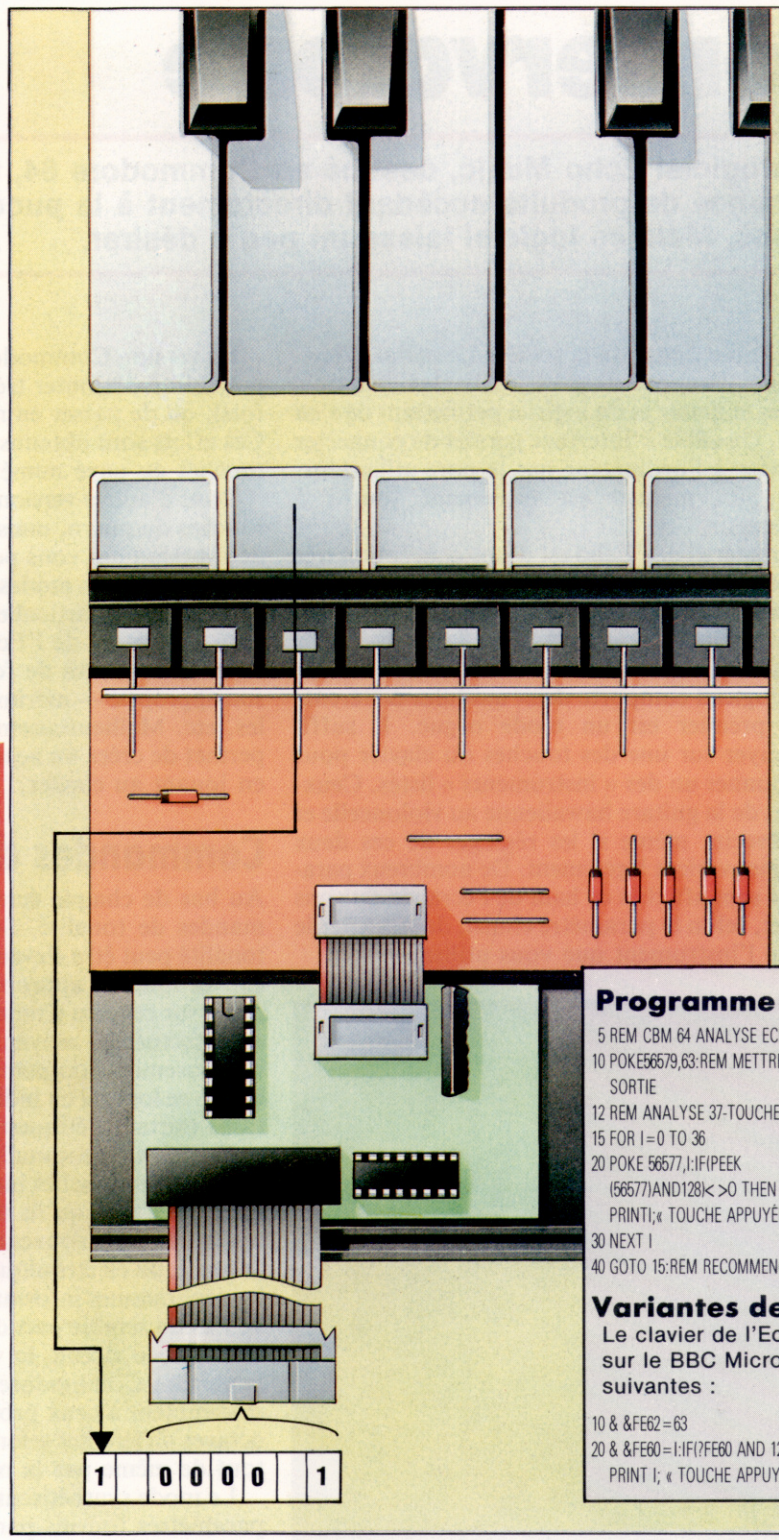
L'ensemble Echo offre des fonctions permettant d'exploiter pleinement le potentiel des puces sonores des micros à partir d'un véritable clavier.

FAIBLESSES

La piètre qualité du logiciel diminue grandement l'efficacité du système, bien qu'une version améliorée soit promise.



Musique sur écran
Voici les affichages écran du logiciel pour le Commodore 64 (gauche) dans le mode orgue, et pour le Commodore 64 dans le mode synthétiseur. Bien qu'elles se ressemblent, les deux versions du logiciel sont assez différentes. On utilise la frappe au clavier pour modifier les sons.



Kevin Jones

Analyse du clavier

Le clavier de l'Echo se branche dans le port utilisateur du Commodore 64 et peut être facilement analysé par logiciel. Le principe de fonctionnement est très simple. Les six bits inférieurs du registre du port utilisateur sont mis en sortie afin que des données puissent être envoyées au clavier, et le bit le plus significatif (bit 7) est mis en entrée pour recevoir des données en provenance du clavier. Pour vérifier la frappe d'une touche sur le clavier, nous devons faire deux interventions : premièrement, POKEr un nombre compris entre 0 et 36 dans le registre du port utilisateur (le clavier a 37 touches numérotées à partir de 0 du côté gauche). Cela définit la touche que nous désirons tester. Deuxièmement, tester le bit 7 du registre de données (avec l'opérateur AND sur la valeur du registre et sur 128). Si ce bit est à 1, la touche en question est alors appuyée; lorsque le bit est à 0, la touche ne l'est pas.

Programme analyse Echo

```
 5 REM CBM 64 ANALYSE ECHO
10 POKE56579,63:REM METTRE BITS 0-5POUR
   SORTIE
12 REM ANALYSE 37-TOUCHES CLAVIER
15 FOR I=0 TO 36
20 POKE 56577,I:IF(PEEK
   (56577)AND128)<>0 THEN
   PRINTI;« TOUCHE APPUYÉE »
30 NEXT I
40 GOTO 15:REM RECOMMENCEZ
```

Variantes de basic

Le clavier de l'Echo peut aussi être utilisé sur le BBC Micro. Faites les modifications suivantes :

```
10 & &FE62=63
20 & &FE60=I:IF(?FE60 AND 128) <>0 THEN
   PRINT I; « TOUCHE APPUYÉE »
```

des quatre synthétiseurs définis par l'utilisateur (par opposition à ceux qui sont prédéfinis dans le programme orgue) vous désirez modifier. L'écran affiche alors les quatorze paramètres requis pour la commande ENVELOPE; chacun de ceux-ci est sélectionné par les touches de curseur gauche et droit et leurs valeurs peuvent être modifiées par les touches de curseur haut et bas.

Le logiciel crée une interface entre le clavier et l'ordinateur en plaçant une valeur de 36 à 0 dans les six bits inférieurs du registre de données du port utilisateur. La méthode d'analyse du clavier par le progiciel a pour résultat que, si on appuie simultanément sur plusieurs touches, ce qui peut arriver souvent, seule la plus élevée sera enregistrée.

Malheureusement, cela interdit tout jeu rapide des doigts sur le clavier de l'Echo car, en d'autres termes, il faut laisser le temps au logiciel de reconnaître les mouvements des doigts : une touche appuyée n'aura d'effet musical qu'une fois la touche précédente relâchée.

Circuits sonores

Construisons pour le C64 une interface lui permettant de communiquer avec la gamme croissante d'équipements compatibles MIDI.

Le principal composant est un adaptateur d'interface de communication asynchrone (ACIA). L'ACIA se charge de la conversion des données parallèles de l'ordinateur en données série requises par MIDI, et vice versa. Nous utiliserons le MC6850 que l'on trouve dans de nombreux micros domestiques pour assurer le transfert de données asynchrones vers des unités cassette et des interfaces de type RS232. Nous aurons besoin d'une horloge d'interface, d'un opto-isolateur et de prises d'entrée et de sortie.

La structure interne de l'unité montrée ici consiste en un certain nombre de registres à 8 bits. La ligne d'entrée du sélecteur de registre (RSEL)

Liste des pièces

N°	Réf.	Article
1	C4	Condensateur polycarbonate 1nF
2	C2/C3	Condensateur polycarbonate 100nF
1	R1	Résistance 270 Ω
3	R2/R3/R4	Résistance 220 Ω
2	R5/R6	Résistance 680 Ω
1	»	Bobine de fil
2	SK1/SK2	Prises DIN 5 broches 180°
1	IC1	Puce ACIA MC68B5
1	IC2	Puce opto-isolateur 6N139
1	IC3	Inverseur hex TTL 74LS04
1	»	Prise DIL à 24 broches
1	»	Prise DIL à 8 broches
1	»	Prise DIL à 14 broches
1	X1	Cristal à 2,0 MHz
1	D1	Diode IN914

N.B. — *La carte DIP utilisée pour monter les composants peut être obtenue chez tous les représentants Tandy; numéro de pièce : 276-164.*

Connexions des circuits

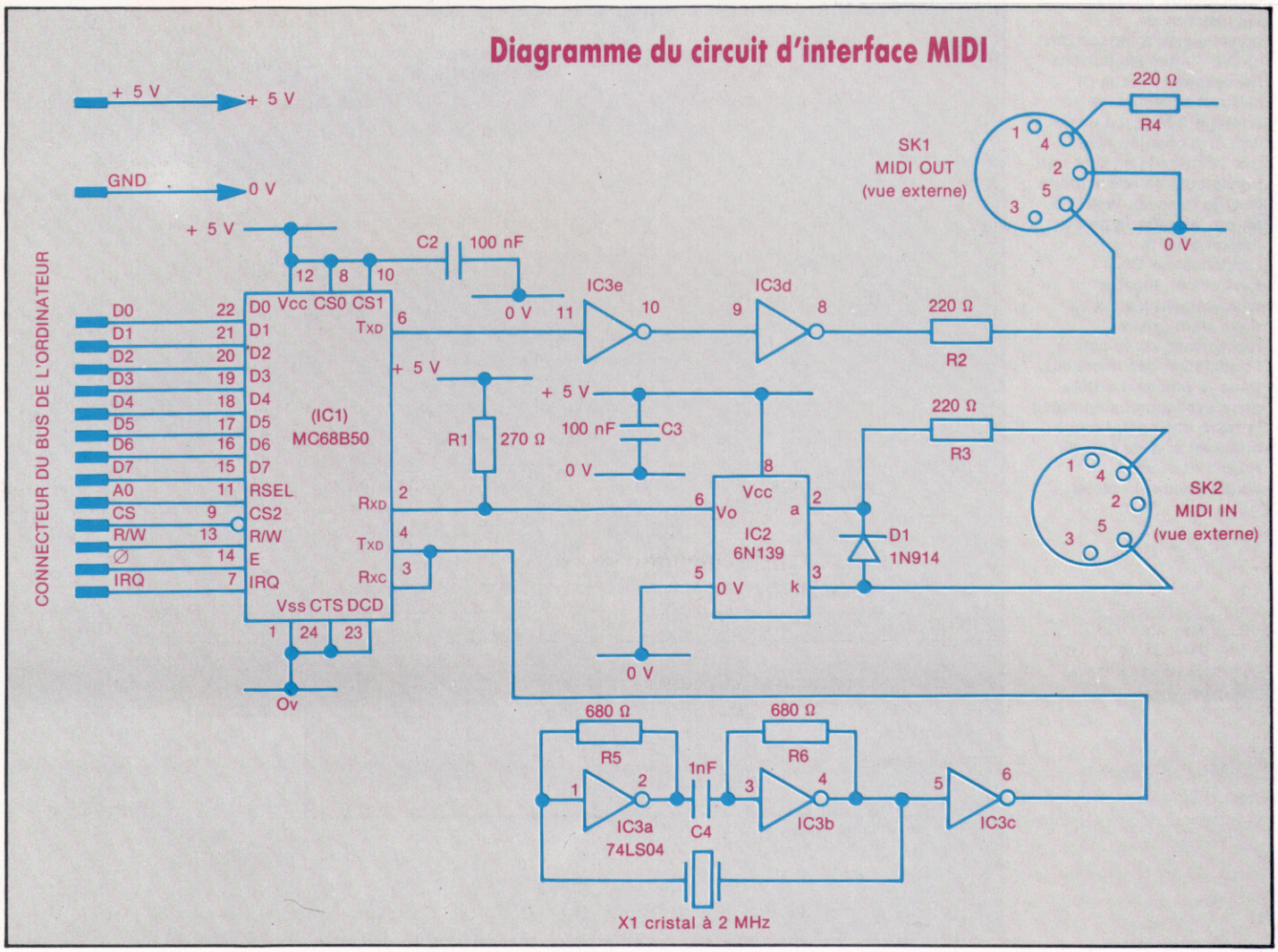
Le couplage opto est assuré par un opto-isolateur (IC2). L'horloge de transmission est dérivée, indépendamment de l'horloge de l'ordinateur, par un oscillateur à cristal de 2 MHz composé d'un IC3a et d'un IC3b.

Les IC3c, d et e assurent chacun des fonctions de tampon pour l'émetteur et pour le générateur d'horloge.

Le MC6850 est compatible au niveau du bus avec l'UC 6502, utilisée dans de nombreux ordinateurs domestiques. Notez que nous avons nommé E l'entrée d'horloge vers l'ACIA pour éviter toute confusion avec les entrées d'horloge de transmission TXD et TXC.

Les deux lignes de sélection de puces, CS0 et CS1, sont connectées à l'alimentation + 5 V et sont donc continuellement au niveau élevé, laissant toujours CS2 comme seule entrée de sélection de puce active (ou unité validée).

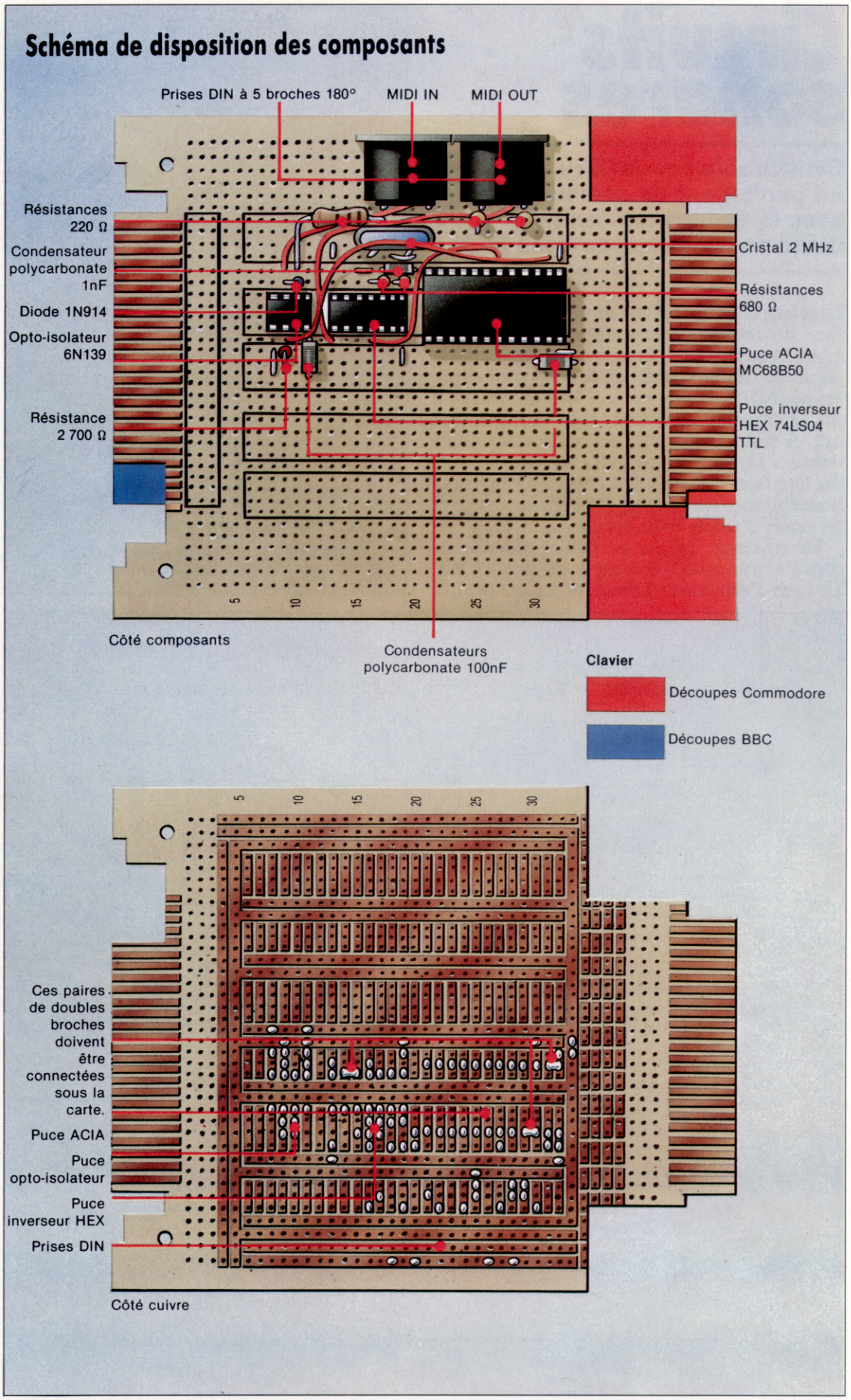
Diagramme du circuit d'interface MIDI



Construction MIDI

Les principaux composants de l'interface MIDI qui doivent être montés sur la carte de montage DIP sont précisés dans la liste des pièces. Cette carte a six connecteurs plats, doubles à ses deux extrémités ; l'extrémité de droite sert à connecter le Commodore 64. Avant de monter les composants, il est nécessaire de faire certaines découpes sur la carte. Notez qu'elles ne doivent être faites qu'à l'une des extrémités de la carte, selon l'ordinateur que vous désirez utiliser. Prenez un couteau pointu et une petite scie à métaux pour faire cette découpe. Toutes les liaisons sont faites sur le haut de la carte à l'aide d'un fil à une voie, sauf les trois liaisons entre les broches adjacentes de CI. Ces liaisons doivent être faites avec de la soudure entre les paires de broches appropriées sur le côté cuivre de la carte. Commencez par monter les composants passifs : les résistances, condensateurs, prises DIN et DIL. Faites les liaisons nécessaires avec le fil (unique) et montez le cristal à 2 MHz. La diode doit être orientée afin que l'extrémité portant une bande colorée soit à droite (vue du dessus). Veillez à ne pas omettre la petite liaison sous le condensateur C3. Finalement, montez délicatement les puces dans leurs prises respectives, en veillant à l'orientation des encoches. Dans le prochain article, nous expliquerons en détail le reste des opérations à effectuer afin de pouvoir relier l'interface à l'ordinateur et la tester.

Schéma de disposition des composants



autorise l'accès aux registres à partir du bus de données. Bien qu'il ne soit pas essentiel de comprendre les fonctions des registres ACIA pour exécuter le logiciel qui sera fourni, les utilisateurs qui désirent programmer l'interface doivent posséder l'information suivante :

• Le *registre d'état* est composé de 8 bits qui décrivent l'état en cours de la puce ACIA. Il est accessible au programmeur en effectuant une lecture sur l'ACIA lorsque la ligne RSEL est au niveau bas (zéro logique).

• Le *registre de contrôle* contient 8 bits qui, comme son nom l'indique, contrôlent le fonctionnement de l'ACIA. On accède au registre de contrôle en écrivant sur l'ACIA quand la ligne RSEL est au niveau bas.

• Le *registre de décalage de transmission* (TSR) effectue la conversion parallèle-série nécessaire pour transmettre un octet de données. Le registre est chargé avec un octet provenant du *registre de transmission de données* (TDR) lorsque le TDR est plein et que la transmission de l'octet précédent est terminée.

Cette opération met à 1 le bit du registre d'état. L'octet est ensuite transmis à un rythme déterminé par l'horloge de transmission. Les bits de départ et d'arrêt sont alors ajoutés aux huit bits de données. Il s'agit d'un registre interne auquel on ne peut accéder directement à partir du bus de données.

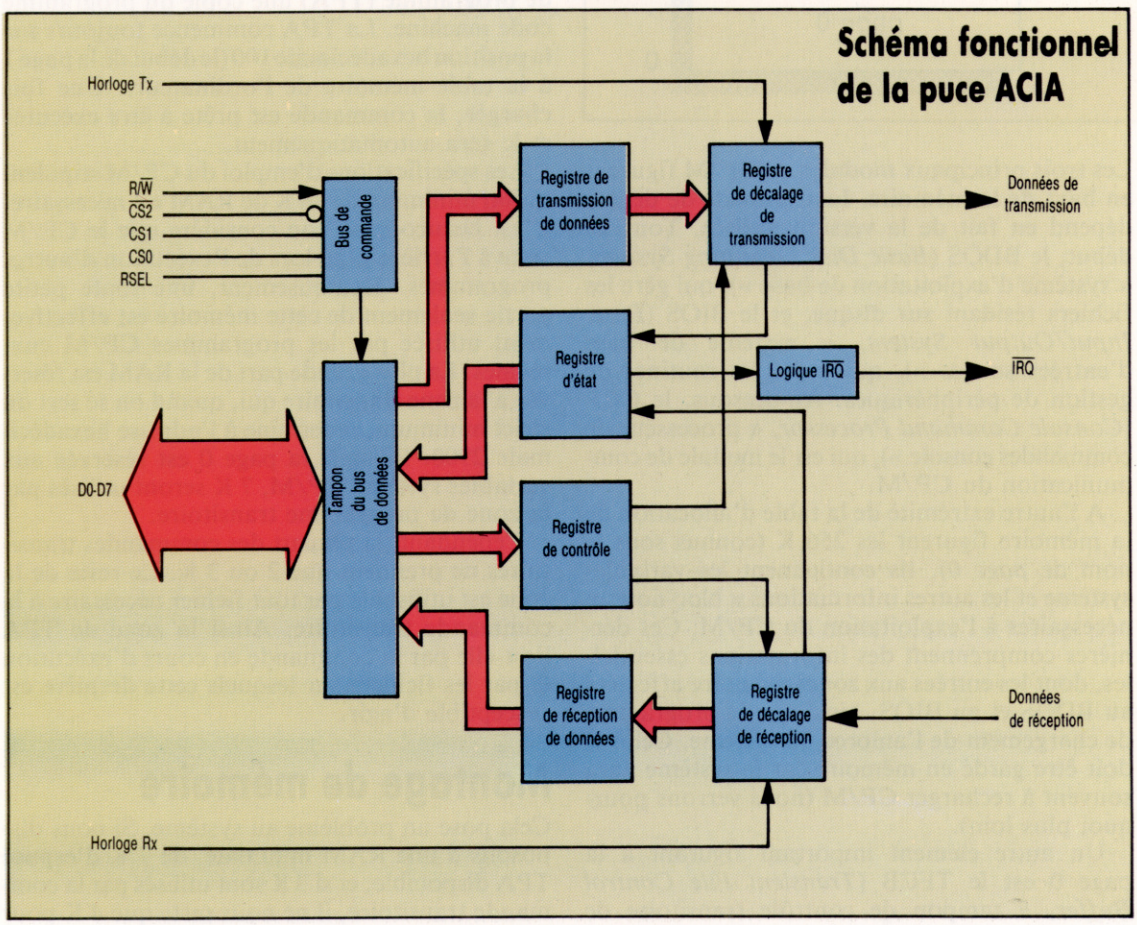
• Le *registre de données de transmission* (TDR) forme un tampon entre le TSR et le bus de données du système. L'octet à transmettre est chargé dans ce registre en écrivant dans l'ACIA quand la ligne de sélection est au niveau élevé. Cette opération met à zéro le bit 1 du registre d'état. Pour ne pas perdre les données, TDR ne doit jamais être chargé lorsque le bit d'état est à zéro. Cela s'explique par le fait que l'octet précédent est toujours dans le TDR en attente de transmission et sera écrasé.

• Le *registre de décalage de réception* (RSR) effectue la conversion de série à parallèle requise lors de la réception de données en provenance du MIDI. Lorsque le registre reçoit un bit complet, il charge les données dans le RDR, mettant à 1 le bit d'état 0. Si ce bit était déjà à 1, le bit 5 est aussi mis à 1, ce qui indique que l'octet précédent n'a pas été lu par l'UC et que les données sont donc perdues.

Parasites

Si l'octet reçu ne comptait pas le nombre désiré de bits de départ et d'arrêt, le bit d'état 4 serait perdu. Cela peut se produire en présence de parasites électriques sur la ligne d'entrée série. On ne peut accéder directement au RSR, pas plus qu'au TSR.

• Le *registre de données de réception* (RDR) n'est chargé que lorsqu'un octet complet est reçu en provenance du RSR. Il contient par conséquent l'octet de données le plus récent. On y a accède en effectuant une lecture sur l'ACIA en maintenant la ligne RSEL à un niveau élevé. Cette opération supprime le bit d'état 0.

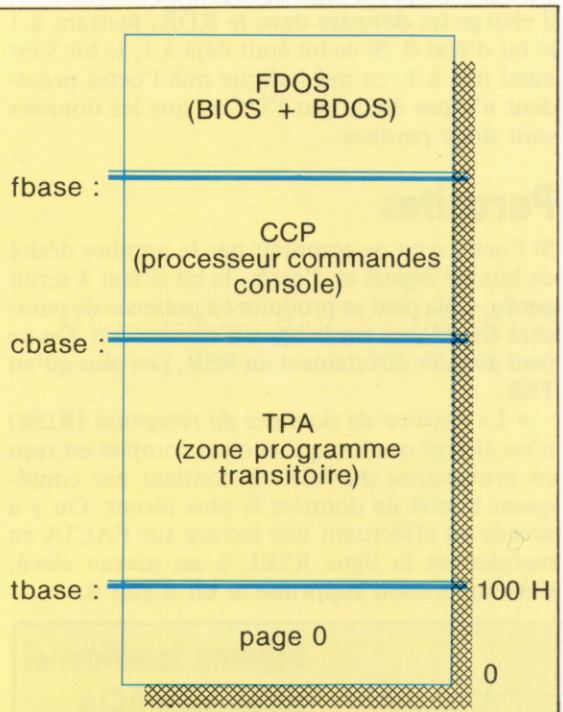


Futur prometteur

Dans ce dernier article sur le système d'exploitation CP/M, nous étudions comment l'organisation interne de la mémoire de l'ordinateur est conçue afin d'optimiser la place disponible.

A l'étroit

Ce schéma montre la disposition du CP/M en mémoire. En bas de la mémoire, la page 0 contient les variables système et les adresses d'entrée, ainsi que le programme chargeur de l'amorce. Au-dessus est la zone de programme transitoire. Sa limite supérieure peut être déplacée pour loger les fichiers qui ne tiendraient pas dans cette zone devant venir recouvrir elle du CCP. En haut de la mémoire, les routines des BIOS et BDOS. Celles-ci ne doivent pas être recouvertes puisqu'elles sont nécessaires à l'exécution des commandes.



Les trois principaux modules de CP/M figurent en haut de la mémoire. Leur adresse de départ dépend en fait de la version utilisée. Tout au début, le BDOS (*Basic Disk Operating System,* « système d'exploitation de base »), qui gère les fichiers résidant sur disque, et le BIOS (*Basic Input/Output System,* « système de base d'entrées/sorties »), qui traite les routines de gestion de périphériques. Au-dessous, le CCP (*Console Command Processor,* « processeur de commandes console »), qui est le module de communication du CP/M.

A l'autre extrémité de la table d'allocation de la mémoire figurent les 256 K (connus sous le nom de *page 0*). Ils contiennent les variables système et les autres informations « bloc-notes » nécessaires à l'exploitation du CP/M. Ces dernières comprennent des informations essentielles, dont les entrées aux zones mémoire affectées au BDOS et au BIOS, ainsi que le programme de chargement de l'amorce du système. Celui-ci doit être gardé en mémoire car le système aura souvent à recharger CP/M (nous verrons pourquoi plus loin).

Un autre élément important figurant à la page 0 est le TFCB (*Transient File Control Buffer,* « tampon de contrôle transitoire de fichier »). Nous avons précédemment vu que, quand nous demandons qu'un fichier soit chargé depuis un disque, le CCP met en place en mémoire un bloc virtuel de contrôle fichier. Lorsque le BDOS rencontre un fichier, il le compare à celui qui figure au CCP et transmet les autres informations de la piste du répertoire au CCP. Cette information est stockée dans le TFCB.

Entre la page 0 et les zones allouées au CCP figure une région de la mémoire appelée TPA (*Transient Program Area,* « région transitoire de programme »). On peut dire qu'il s'agit de l'espace de travail du CP/M. Comme nous l'avons vu, lorsqu'une instruction est soumise au système d'exploitation, le CCP commence par passer en revue la liste des commandes résidentes stockée dans sa zone mémoire. S'il ne trouve pas l'instruction, il la considère comme transitoire et demande au BDOS de l'identifier. Celui-ci suppose que la commande figure au disque système. Il charge au début de la zone transitoire de programme (TPA) une copie du programme code machine. La TPA commence toujours sur la position hexadécimale 100 (le début de la page 1 à la table mémoire de l'ordinateur). Une fois chargée, la commande est prête à être exécutée et le sera automatiquement.

Les spécifications d'emploi du CP/M stipulent qu'un minimum de 16 K de RAM est nécessaire. C'est beaucoup si l'on considère que le CP/M reste à l'arrière-plan lors de l'exécution d'autres programmes. Heureusement, une toute petite partie seulement de cette mémoire est effectivement utilisée par les programmes CP/M eux-mêmes. La plus grande part de la RAM est réservée à la zone transitoire qui, quand on se sert du strict minimum, se termine à l'adresse hexadécimale 2900. Puisque la page 0 est réservée aux variables système CP/M, 7 K seront utilisés par la zone de programme transitoire.

Cependant, la plupart des commandes transitoires ne prennent que 2 ou 3 K. Le reste de la zone est utilisable par tout fichier nécessaire à la commande transitoire. Ainsi la zone de TPA l'est-elle par la commande en cours d'exécution et par les fichiers sur lesquels cette dernière est susceptible d'agir.

Montage de mémoire

Cela pose un problème au système. Si nous disposons d'une RAM minimale, de 7 K d'espace TPA disponible, et si 3 K sont utilisés par la commande transitoire, il ne nous reste que 4 K pour

les autres fichiers. C'est à peine suffisant pour un programme de taille moyenne, si nous excluons les longs fichiers texte. Nous avons vu qu'un fichier CP/M peut comprendre jusqu'à 16 K. On pourrait bien sûr contourner le problème en rajoutant de la mémoire. Toute nouvelle tranche de RAM ajoutée au système est affectée au TPA, jusqu'à un maximum de 64 K directement adressables par le CP/M. Ainsi, le haut de la mémoire occupée par le CP/M est élevé d'une valeur hexadécimale de 4000 pour chaque nouvelle tranche de 16 K.

Mais supposons que nous ne puissions rajouter la mémoire nécessaire. Comment procède alors CP/M pour faire tenir tant de mémoire en si peu de place? La réponse est simple : tout débordement de la zone programme transitoire recouvre en écriture le processeur de commandes console. Ce n'est pas aussi grave que cela en a l'air. D'abord ce dernier, le CCP, n'est pas nécessaire lors de l'exécution d'une commande. Le CP/M, à l'image de tous les systèmes d'exploitation, refuse alors toute nouvelle commande. Aussi le CCP n'a-t-il pas besoin d'être présent pour une éventuelle interruption ou exécution de commandes résidentes superflues.

Bien sûr, lorsque le programme de commande est terminé, il devient à nouveau nécessaire de recharger les programmes du CCP afin que le système soit disponible pour une nouvelle commande. C'est la fin d'une commande transitoire. Le programme appelle alors la routine de lancement du système, qui est située à l'adresse hexadécimale 0005 de la page 0. Cette adresse est l'entrée du système d'exploitation. Elle recharge en mémoire le module du CCP qui est alors prêt à recevoir l'instruction suivante.

Contraintes de conception

Nous sommes maintenant à même de comprendre la nature des commandes transitoires. Lors de la conception du CP/M, les composants (notamment de RAM) étaient très chers, et peu de machines étaient équipées en standard de plus de 16 K. Aussi les logiciels destinés à ces machines devaient-ils se conformer à ces limitations.

Pour que le CP/M soit aussi complet que possible dans un espace mémoire donné, certaines de ses zones devaient se recouper, se recouvrir. Et comme les BIOS et BDOS sont tous les deux nécessaires de manière permanente, ils ne pouvaient être employés à cette fin. De nombreuses commandes les utilisent pour accéder aux fichiers ou pour des opérations d'entrée/sortie (telles que diriger un fichier vers l'imprimante ou l'afficher à l'écran). C'est pourquoi il a été décidé que le CCP et la TPA occuperaient alternativement le même emplacement mémoire : ils ne peuvent en aucun cas être utilisés en même temps.

La zone du processeur peut s'effacer au profit du programme transitoire : celui-ci étant supprimé après exécution, le processeur peut alors être rechargé en toute sécurité. Il est remarquable que Gary Kildall et son équipe soient parvenus à réaliser un système durable et performant malgré les limites des possibilités des matériels de l'époque.

Maintenant que les micros professionnels adoptent le processeur 16 bits, il semblerait que le CP/M soit voué à disparaître. Cependant, même si des systèmes d'exploitation sur 16 bits, tels que le MS-DOS, dominent de plus en plus le marché, l'ordinateur 8 bits utilisant le microprocesseur Z80 a encore de beaux jours devant lui.

Les ordinateurs domestiques et les petits micros professionnels n'ont pas besoin, en effet, de la puissance d'un processeur 16 bits, mais leurs lecteurs de disque auront, eux, besoin d'un système d'exploitation. Le CP/M, qui est, à ce jour, le système d'exploitation de fait pour les 8 bits, devrait constituer le choix idéal pour les fabricants désireux de fournir une vaste gamme de logiciels sans grever les coûts de développement. Certains ont déjà fait ce choix, notamment Memotech et Amstrad.

Mise en œuvre d'Amstrad

Naturellement, les utilisateurs sont éblouis par la perspective d'avoir à leur disposition quelque dix mille logiciels développés sous le standard CP/M. Mais ils se trouvent confrontés à plusieurs problèmes. Notamment, Amstrad leur pose une difficulté particulière : ayant choisi le format Hitachi 3 pouces pour les lecteurs, il n'offre pas une compatibilité immédiate avec les logiciels CP/M développés originellement en $5\frac{1}{4}$ pouces et 8 pouces. Cela signifie que les utilisateurs d'Amstrad devront attendre des applications spécifiques.

Un autre problème avec cet ordinateur vient de sa gestion d'écran qui ne laisse que 38 K de libre pour les programmes, ce qui est très insuffisant pour les logiciels récents écrits sous CP/M. A part certains de ces programmes qui peuvent être aménagés pour correspondre à la mémoire disponible, les utilisateurs d'Amstrad devront se contenter de programmes obsolètes. Cependant, en choisissant d'investir le marché des « bas de gamme », CP/M sur 8 bits semble devoir confirmer son développement.

Cela ne signifie pas que Digital Research ait abandonné le marché des processeurs 16 bits. Mais ce dernier est maintenant très fermé, très concurrentiel, et la conjoncture n'est plus favorable comme au temps où Gary Kildall développa le CP/M. Une version ultérieure, le CP/M-86, essaya de pénétrer le marché des processeurs 8088/6. Mais elle ne connut pas de succès. La tendance est maintenant à la compatibilité IBM et au standard MS-DOS.

Mais, tout récemment, Digital Research a lancé une version multi-utilisateurs de CP/M, appelée « CP/M Concurrent ». Ce système d'exploitation peut être utilisé par un seul utilisateur ou par plusieurs reliés en réseau. Il est peut-être trop tôt pour dire si CP/M Concurrent connaîtra le même succès que son prédécesseur 8 bits, mais il semble bien que CP/M soit en mesure d'occuper le terrain encore longtemps.

A l'intérieur de MP/M et CP/NET

Nous avons abordé presque exclusivement le CP/M version 2.2, de loin la plus répandue. Comme toutes les versions CP/M, elle est mono-utilisateur et monoposte. Mais Digital Research a par ailleurs développé d'autres systèmes (appelés « systèmes multitâches ») construits autour de CP/M et destinés à être multipostes et multi-utilisateurs.
Un système multitâches permet le partage temporel de l'unité centrale entre un certain nombre d'utilisateurs et de périphériques. Le système d'exploitation qui permet cette mise en commun des ressources s'appelle MP/M (*Multi-Processing Monitor Control Programm,* « programme de contrôle de moniteur multitraitements »). Ce système permet de relier jusqu'à seize terminaux entièrement autonomes. Un utilisateur donné n'a du reste pas conscience de la présence des autres utilisateurs sur le réseau. Pour coordonner les interventions des utilisateurs, MP/M comprend un certain nombre de caractéristiques pour gérer le partage des fichiers et des périphériques.
CP/NET est un autre développement de CP/M de Digital Research. C'est la version réseau de CP/M. Elle permet à plusieurs utilisateurs de communiquer. Contrairement à MP/M, CP/NET ne suppose pas une tête de réseau et une hiérarchisation, même si un nœud central fonctionne sous MP/M et dispose de lecteurs.

Code génétique

Dans cet article consacré aux machines capables d'apprendre, nous proposons un programme intitulé GENE, qui permet de trouver de meilleures solutions à des problèmes de réseaux assez complexes.

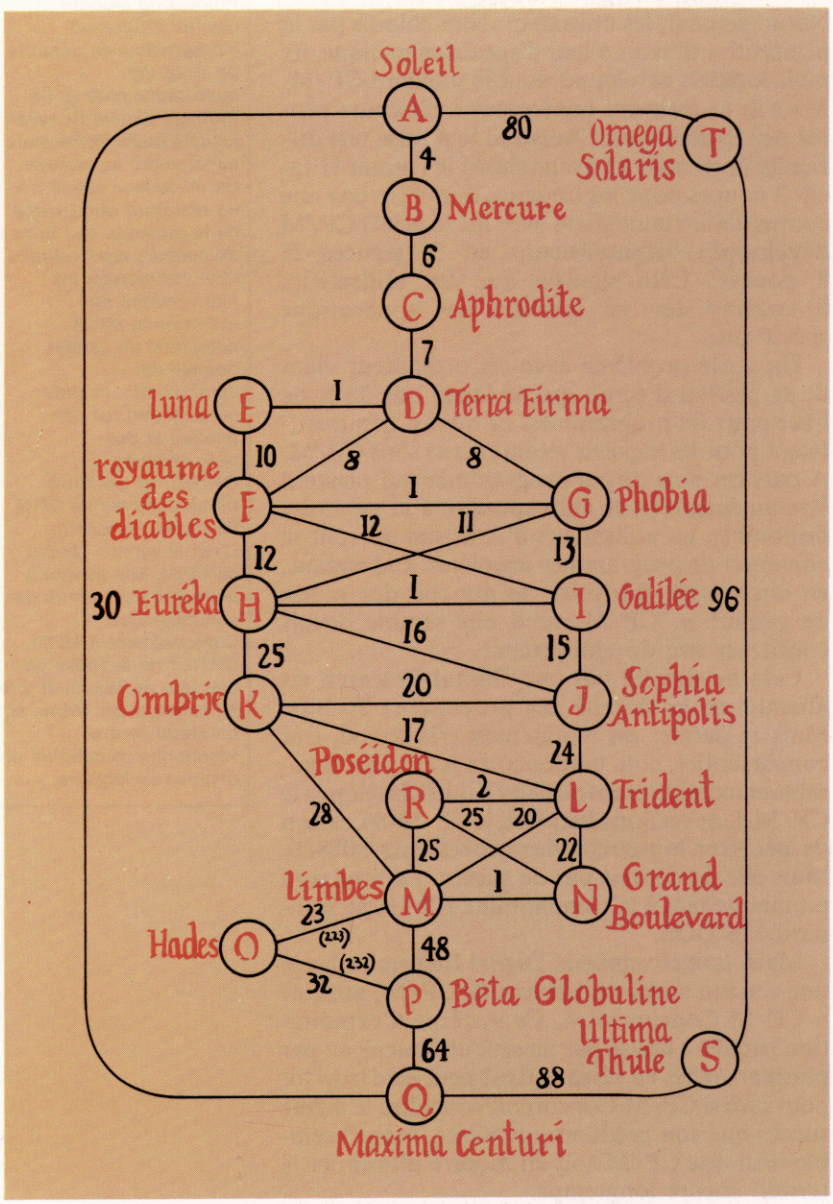
Quand les chercheurs en intelligence artificielle s'efforcent de trouver dans la nature des idées leur permettant de mettre au point des systèmes capables d'apprendre par eux-mêmes, ils rencontrent aussitôt trois bons exemples : le système nerveux, le système immunitaire et l'évolution elle-même. Le premier est un mécanisme extraordinairement efficace, comme le montre le cerveau humain. Mais ses opérations restent plus ou moins mystérieuses, et nous ne les comprenons pas assez pour pouvoir les reproduire. Le deuxième est aussi un mécanisme d'apprentissage, capable de faire la différence entre lui-même et les autres, de telle sorte qu'il peut attaquer et détruire les organismes étrangers. Dans l'espace d'une vie, il apprend à reconnaître des millions de protéines; nous serions vite morts sans cette phénoménale capacité d'adaptation. Mais il procède avant tout par simple accumulation de connaissances ponctuelles; ses pouvoirs de généralisation restent tout à fait rudimentaires.

Le processus évolutif lui-même peut, en revanche, nous aider à mettre au point des systèmes plus élaborés; il est, bien sûr, très lent pour nous, mais nous pouvons toujours recréer sur ordinateur une simulation ayant un rythme plus rapide. Il a surtout l'avantage d'être assez simple, et assez bien compris, pour que nous puissions le reproduire avec une chance raisonnable de succès. Nous avons déjà fait remarquer qu'une approche « darwinienne » du problème de l'apprentissage des machines avait certains avantages théoriques. Voyons maintenant ce qui se passe quand on en vient au stade de la pratique.

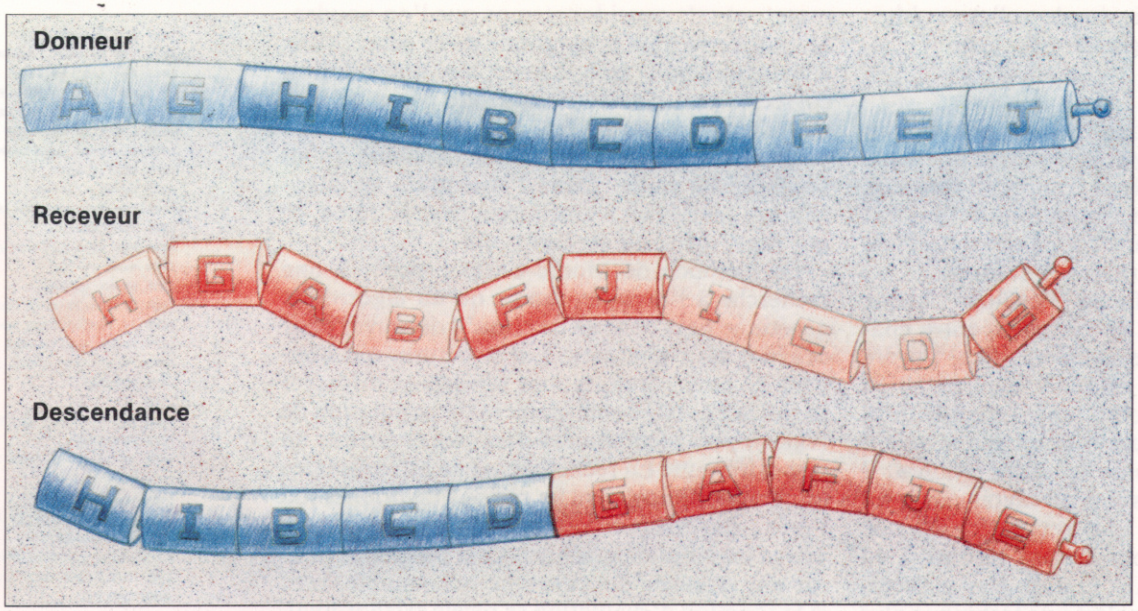
Pour illustrer l'emploi d'un algorithme évolutionniste dans un problème de ce type, examinons la célèbre énigme du voyageur de commerce. Sous sa forme classique, elle met en scène un représentant américain qui doit visiter les capitales de quarante-huit États des États-Unis (Hawaii et Alaska sont exclus pour des raisons de commodité). Il connaît les distances entre elles, et doit se rendre une seule fois dans chacune d'elles; le trajet total doit être le plus court.

Tout cela a l'air très simple, mais on a vite fait de se rendre compte qu'il existe un nombre total de trajets égal à (N-1)! N étant le nombre de villes. Un circuit comprenant 48 villes peut comporter 47 premières étapes différentes, suivies de 46 autres (deuxième étape), puis de 45, etc. A dire vrai, il existe en fait plus de trajets concevables que d'atomes dans l'Univers! Même réduit à vingt villes, notre circuit comprend plus de 20 millions de millions de possibilités!

Le problème peut être abordé de plusieurs façons. On peut le traiter comme une question de recherche, en mettant en œuvre des méthodes analogues à celles que nous avons déjà décrites, ou faire usage de techniques aléatoires de type Monte-Carlo. Nous choisirons pourtant une approche un peu particulière, en l'abordant comme s'il s'agissait d'un problème d'apprentissage. Nous avons donc besoin d'un procédé qui explore, de façon économique, le nombre énorme de solutions potentielles. Il n'est pas nécessaire



Par-delà les étoiles
Notre carte interplanétaire indique les routes hyperspatiales entre chaque planète, et le temps de traversée correspondant. La fonction du programme GENE est de trouver des circuits aussi rapides que possible.
Bernard Jennings



Kevin Jones

Familles, je vous aime
GENE découvre de nouveaux trajets au sein du réseau en appariant deux circuits déjà existants et qui donnent de bons résultats. Ils sont présentés sous forme de chaînes de caractères BASIC ; leur donner un « descendant » se réduit en fait à manipuler ces chaînes. Dans notre exemple, la sous-chaîne HIBCD est prise à l'un des parents (le donneur) et transmise à l'autre (le receveur), en s'assurant qu'aucune lettre n'est répétée deux fois. On est ainsi certain que, comme dans les manipulations génétiques réelles, les caractéristiques des parents passent à leur « rejeton ».

qu'il découvre la meilleure, mais il doit au moins en trouver une bonne.

Nous vous présentons ici le programme GENE *(General Evolutionary Network Explorer),* qui est un système d'apprentissage spécifiquement mis au point pour explorer des réseaux et déterminer des règles évolutives lui permettant de définir des trajets de plus en plus courts. Avant d'en appliquer par le menu le fonctionnement, il n'est peut-être pas inutile de choisir un réseau précis. La plupart des guides galactiques donnent la liste des planètes situées à moins de 80 al et sur lesquelles vous avez une chance d'assister, chaque samedi soir, à une surprise-partie décente. Votre objectif sera donc de participer à vingt « sauteries » différentes, puis de revenir à votre point de départ, le tout en une seule nuit.

La carte qui vous est donnée décrit les grandes routes hyperspatiales qui sillonnent le voisinage de notre système, et précise les temps nécessaires pour aller d'un point à l'autre. Certains de ceux-ci ne sont pas connectés ; on considère que dans ce cas il faut 1 000 unités de temps pour faire le trajet.

Schéma d'apprentissage évolutionniste

Un schéma classique définit d'abord une population de structures qui sont traitées comme autant de « pseudo-organismes ». Chacune d'elles (les « règles ») donne une solution potentielle au problème à traiter. Elle permet aussi la diffusion de « règles filles » grâce à des procédures qui reprennent certains éléments de la reproduction biologique — par exemple les « parents » transmettent certaines de leurs caractéristiques à leurs « enfants ».

Suivant les résultats obtenus, certaines règles sont choisies ; elles ont les plus grandes chances de « survie » et la plus grande probabilité de se « reproduire ». Dans notre programme, leur

Ingénierie alphabétique

Deux structures « parentales » sont d'abord choisies au hasard parmi celles qui ont survécu au processus de choix (ce qui implique que les parents ont une capacité supérieure à la moyenne pour remplir leur tâche). L'un d'eux s'appelle le « donneur » (R1%) et l'autre le « receveur » (R2%) : voir les lignes 3030 à 3060.
Un lot de « matériel génétique » (qui est en fait une sous-chaîne) est extrait du « donneur » de façon aléatoire et placé dans la chaîne S$. Le sous-programme de la ligne 3300 mêle ensuite le tout à la chaîne « receveur » en y insérant les lettres choisies (sauf celles qui font double emploi), dans l'ordre dans lequel elles apparaissent. Le processus d'appariement est en fait asymétrique : X apparié avec Y ne donne pas les mêmes résultats que Y apparié avec X, même si les positions aléatoires P1% et P2% sont identiques. Voyons d'un peu plus près un exemple précis. Les deux parents étant :

Donneur : A G H I B C D F E J.
Receveur : H G A B F J I C D E.

Nous pouvons sélectionner la sous-chaîne

H I B C D

comme étant offerte par le donneur, et y adjoindre les autres lettres du receveur pour obtenir

H I B C D G A F J E

en résultat final.
Il faut bien insister sur le fait que ce n'est pas là la seule façon de procéder autrement. Mais cette méthode, tout comme les croisements génétiques authentiques, assure le transfert de l'information parentale à la nouvelle génération. Chaque appariement produit de même un rejeton « valide ». Cela signifie qu'en fait le sous-programme de la ligne 4000 n'a pas d'utilité véritable, mais nous l'avons conservé afin de vous donner une idée plus précise de la structure d'ensemble du programme.

structure reste tout à fait simple : elles se réduisent à des chaînes de caractère du genre :

ABJHMNCDKTSFRQEGILOP

Chaque lettre correspond à une planète du système stellaire et n'apparaît donc qu'une seule fois dans la chaîne. Celle-ci comprend toujours 20 caractères, dont les permutations définissent l'ordre de visite des planètes, ce qui détermine un trajet particulier à travers le réseau.

Il est extrêmement facile d'évaluer chaque chaîne en additionnant les distances entre chaque planète prise successivement dans l'ordre déterminé. Plus le total est petit, meilleur est le trajet. Nous ferons en sorte que les chaînes dont les totaux sont supérieurs à la moyenne soient supprimées à chaque nouvelle « génération ». Il se pose bien sûr la question de savoir par quoi les remplacer. Comme tout repose ici sur une analogie biologique, nous aurons recours à un procédé qui évoque la mutation des espèces, parce qu'en l'occurrence c'est une technique plus satisfaisante que la reproduction sexuée proprement dite.

Huit pour cent des chaînes « survivantes » subiront donc des mutations. Le sous-programme correspondant commence à la ligne 3500 et se borne à procéder à des échanges aléatoires. Les mutations ne constituent pas en génétique des opérateurs primaires, mais des modifications d'arrière-plan qui empêchent le système d'atteindre un optimum local dans lequel il resterait pris. Il peut, en effet, y avoir une limite supérieure aux améliorations survenues, par les seules méthodes de la reproduction sexuée, à l'issue de plusieurs générations.

Si vous vous livrez à certaines expériences sur le programme, vous constaterez que la norme moyenne de la population des règles s'améliore réellement avec le temps, bien qu'il n'y ait pas de progression continue. Même les meilleures règles peuvent « mourir » avec le temps. En fait, le programme « triche », puisqu'il conserve toujours la meilleure règle, qu'il ne remplace jamais que par une autre encore meilleure.

Vous pourrez procéder à des « réglages » en jouant avec les « taux de décès » de la ligne 2520. Elle comporte un élément aléatoire qui fixe le taux de survie des bonnes règles, d'une génération à l'autre, à 88 % ; mais cela peut être modifié. Vous pouvez également ajuster ligne 3520 le taux de mutation, ou introduire des opérations de mutations nouvelles, par exemple l'inversion complète d'une chaîne.

Deux questions subsistent :

1. Quelle est la valeur de la méthode ?
2. Pourquoi fonctionne-t-elle ?

On peut obtenir une réponse à la première question en se servant de la méthode de Monte-Carlo, puisque tous les algorithmes évolutifs n'en sont qu'une forme légèrement modifiée ; cette méthode consiste en effet à générer des solutions aléatoires et à conserver la meilleure, le tout dans une limite temporelle spécifiée. Ici, cela voudrait dire essayer une chaîne règle au hasard, l'évaluer, la retenir si elle se révèle la meilleure, puis la faire muter, et cela aussi longtemps que l'on voudra. Si vous écrivez un programme de ce genre, vous constaterez qu'il trouve assez vite une solution correcte, puis s'améliore quelque peu à mesure que le temps passe. En fait, au bout de vingt mille essais, les résultats sont à peine meilleurs qu'au bout de dix mille. Dans l'ensemble, les algorithmes génétiques s'améliorent avec la durée, ce qui nous mène au second point.

La méthode de Monte-Carlo procède à une recherche à l'aveugle, tandis qu'un algorithme génétique tient compte de ce qu'il découvre pour orienter ses recherches. Les structures qui donnent de bons résultats sont reprises et diffusées parmi la base de connaissances (la population de règles), puis recombinées dans des contextes un peu différents. La recherche est en effet orientée préférentiellement vers les régions favorables (où l'on a trouvé de bons résultats) d'un « espace problème » multidimensionnel. Cela permet de définir une stratégie de recherche raisonnable, sauf si la fonction d'évaluation se révèle extraordinairement discontinue.

Le programme GENE

Les tableaux suivants contiennent les données essentielles nécessaires à la mise en œuvre de notre système d'apprentissage évolutif.

NOM$ (TAILLE%)	Noms des nœuds du réseau.
LIEN%(TAILLE%, TAILLE%)	Distances entre nœuds.
NX%(TAILLE%)	Utilisé dans les méthodes emmêlées.
R$(NR%)	Les règles elles-mêmes (chaînes).
RV(NR%)	Valeur de chaque règle.

On se sert de NX% quand on cherche à créer une nouvelle règle (ligne 3300), de façon que chaque lettre-nœud n'apparaisse qu'une seule fois. R$ garde en mémoire la population de règles en cours, sous forme de chaînes de longueur TAILLE%. Chacune d'elles définit un trajet particulier. RV donne la « valeur » de chaque règle par rapport à la longueur du trajet qu'elle représente. Si RV(R%)=0, la règle R% « meurt » et devra être remplacée à la génération suivante (les circuits de ce type ne sont, de toute évidence, pas viables). TAILLE% peut être modifié si vous désirez essayer des réseaux un peu différents, auquel cas vous aurez à modifier le programme à partir de la ligne 8000. NR% peut également être altéré de façon à juger de l'effet d'un nombre inférieur, ou supérieur, de règles.

```
  10 REM ************
  11 REM **  GENE  **
  13 REM ************
 100 GOSUB 1000  : REM Mise au point matrice carte
 101 GX=0: SNX=0
 105 B=TAILLEX*1000: B$=""
 110 INPUT "COMBIEN DE GENERATIONS ", MGX
 111 GOSUB 1700  : REM REGLES INITIALES
 115 IF SNX=0 THEN INPUT "NOEUD INITIAL No :", SNX
 120 REM **** BOUCLE PRINCIPALE ****
 130 GX=GX+1
 140 PRINT "Generation ";GX
 150 GOSUB 2000  : REM EVALUATION REGLES
 160 GOSUB 2500  : REM SUPPRESSION MAUVAISES REGLES
 170 GOSUB 3000  : REM APPARIEMENT BONNES REGLES
 180 GOSUB 3500  : REM mutations
 190 GOSUB 4000  : REM RENFORCEMENT
 200 IF GX<MGX THEN 120
 220 GOSUB 5000  : REM expression nouvelles règles
 250 END
 999 :
1000 REM -- S/P MISE EN PLACE RESEAU :
1001 TAILLE X = 20: NRX=24
1010 DIM NOM$(20), LIENX(20,20)
1011 DIM NX%(TAILLE%)
1012 DIM R$(NR%), RV(NR%)
1013 REM les règles et leurs valeurs
1015 FOR IX=1 TO TAILLE X
1020 FOR JX=1 TO TAILLE X
1022 LIENX(IX,JX)=1000  : REM VALEUR P
1023 IF IX=JX THEN LIENX(IX,JX)=0
1025 NEXT : NEXT
1030 NCX=0  : LX=0
1032 FOR IX=1 TO NRX: RV(IX)=0: NEXT
1033 RESTORE
1040 REM **** LIT NOM ET NUMERO DU NOE
1050 READ N$,IDX
1055 PRINT N$,IDX
1060 IF IDX<>0 THEN GOSUB 1500
1070 IF IDX<>0 THEN 1040
1080 PRINT NCX;" LIEUX MIS EN PLACE."
1088 PRINT LX;" LIENS DIFFERENTS."
1090 RETURN
1100
1500 REM -- NOEUD (ET CONNEXIONS) INDI
1510 NCX=NCX+1
1520 IF IDX<>NCX THEN PRINT "ATTENTION
"NON ORDONNE."
1530 NOM$(NCX)=N$
1550 REM **** MISE EN PLACE TABLEAU IN
1560 READ NIX,NTX
1570 LIENX(IDX,NIX)=NTX
1588 REM LIENS NULS : AUCUNE IMPORTANC
1590 LX=LX+1
1600 IF NIX>0 THEN 1550
1610 RETURN
1620 :
1700 REM --règles initiales sans valeu
1710 S$=LEFT$("ABCDEFGHIJKLMNOPQRSTUVW
1715 GOSUB 1780 : REM lit tout fichier
1720 FOR RX= 1 TO NRX
1730 R$(RX)=S$
1740 PRINT S$:RX
1750 IX=INT(RND(1)*TAILLEX+1): JX=INT
1760 GOSUB 6000:REM INVERSION
1770 NEXT
1775 RETURN
1777 :
1780 INPUT "fichier anciennes règles
RFX
1790 IF RF$="" THEN RETURN
1800 OPEN 2,8,2,RF$+"S,R" :FX=2 : REM
1805 OPEN 2,1,0,RF$:FX=2:REM système
1810 INPUT FX,B$,B,SNX
1820 CLOSE 2 REM système à disquettes
1825 S$=B$
1830 REM seulement la meilleure
1840 RETURN
1850 :
2000 REM -- S/P évaluation règles
2010 T=0
2020 FOR RX=1 TO NRX
2030 IF RV(RX)<=0 THEN GOSUB 2200
2040 T=T+RV(RX)
2050 NEXT
2060 AV=T/NRX : REM valeur moyenne
2070 PRINT "SCORE MOYEN =": AV
2080 RETURN
2100 :
2200 REM -- évaluation règle isolée
2210 S$=R$(RX)
2220 P1X=SNX : REM noeud de départ
2230 GTX=0
2240 FOR SX=1 TO LEN(S$)
2250 P2X=ASC(MID$(S$,SX,1))-64
2260 IF P2X=SNX THEN GOTO 2290
2270 GTX=GTX+LIENX(P1X,P2X)
2280 P1X=P2X
2290 NEXT
2300 RV(RX)=GTX+LIENX(P2X,SNX)
2310 RETURN
2320 :
2500 REM -- S/P suppression mauvaises
2510 FOR RX=1 TO NRX
2515 IF RV(RX) < B THEN B=RV(RX): I$=R
2520 IF RV(RX) < AV OR INT(RND(1)*100+
R$(RX)="": RV(RX)=0
2530 NEXT
2540 RETURN
2550 REM -- NB il vaut mieux de petite
2560 :
3000 REM -- S/P appariement
3010 FOR RX=1 TO NRX
3020 IF RV(RX))0 THEN GOTO 3120
3030 R1X=INT(RND(1)*NRX+1):IF RV(R1X)<
3050 R2X=INT(RND(1)*NRX+1):IF RV(R2X)<
3070 REM les "parents" sont choisis
3075 P2X=INT(RND(1)*(TAILLEX-1)+1
3080 P1X=INT(RND(1)*TAILLEX+1) : REM
3090 S$=MID$(R$(R1X),P1X,P2X)
```

```
3100 GOSUB 3300 : REM insertion du reste
3110 R$(RX)=S$
3120 NEXT
3140 RETURN
3150 :
3300 REM -- S/P mélange de gene
3310 FOR SX=1 TO TAILLEX
3320 NX%(SX)=0 : NEXT
3330 FOR SX=1 TO LEN(S$)
3340 SX=ASC(MID$(S$,SX,1))-64
3350 NX%(SX)=NX%(SX)+1
3360 NEXT
3370 FOR SX=1 TO LEN(R$(R2X))
3380 SX=ASC(MID$(R$(R2X),SX,1))-64
3390 IF NX%(SX)>0 THEN GOTO 3420
3400 S$=S$+MID$(R$(R2X),SX,1)
3410 NX%(SX)=NX%(SX)+1
3420 NEXT
3440 RETURN
3450 :
3500 REM -- S/P mutation
3510 FOR RX=1 TO NRX
3520 IF RND(100) > 8 THEN GOTO 3580
3522 S$=R$(RX)
3525 FOR TX=1 TO 7
3530 R1X=INT(RND(1)*TAILLEX+1)
3540 R2X=INT(RND(1)*TAILLEX+1)
3560 IX=R1X:JX=R2X:GOSUB 6000:REM INVERSION
3565 NEXT TX
3570 R$(RX)=S$
3575 RV(RX)=0
3580 NEXT
3590 REM seul échange pour le moment
3595 REM une inversion s'impose
3600 RETURN
3620 :
4000 REM -- S/P RENFORCEMENT
4010 RETURN
4020 REM N'EXISTE PAS ENCORE
5000 REM -- affichage résultats
5010 PRINT"LES TRAJETS SONT :"
5020 BR="TAILLEX+1000
5030 RX=0
5040 FOR IX=1 TO NRX
5050 IF RV(IX)=0 THEN 5100
5060 PRINT IX,RV(IX)
5070 PRINT R$(IX)
5080 IF RV(IX)<BR THEN BR=RV(IX): RX=IX
5100 NEXT
5105 GET A$ : IF A$=""THEN 5105
5110 PRINT
5120 PRINT "LE MEILLEUR EST: "
5130 PRINT R$(RX),RX
5131 S$=R$(RX):GOSUB 6400
5133 PRINT"Distance = ";RV(RX)
5134 GET A$ : IF A$ =""THEN 5134
5135 PRINT"le meilleur jusqu'à présent :"
5136 PRINT B$
5140 S$=B$:GOSUB 6400
5143 PRINT"Distance =";B
5144 GET A$: IF A$="" THEN 5144
5145 PRINT
5148 GOSUB 5500 : REM fichier mise en réserve
5150 RETURN
5160 :
5500 REM -- S/P mise en réserve
5510 INPUT "New Rule File (RETURN if none)", RF$
5520 OPEN 2,1,1, RF$:FX=2 :REM POUR CASSETTES
5530 OPEN 2,8,2, RF$+",S,W":FX=2
5540 PRINT FX,B$,B,SNX
5550 CLOSE 2
5555 REM LE MEILLEUR DU LOT
5560 RETURN
5570 :
6000 REM **** inversion de 2 caractères de S$ ****
6040 IF IX>JX THEN TX=IX: IX=JX: JX=TX
6050 X$=MID$(S$,IX,1)
6060 Y$=MID$(S$,JX,1)
6070 S$=LEFT$(S$,IX-1)+Y$+MID$(S$,IX+1)
6080 S$=LEFT$(S$,JX-1)+X$+MID$(S$,JX+1)
6090 RETURN
6100 :
6400 REM **** VOYAGE ****
6430 PRINT " 0  "NAME$(SNX)
6440 FOR IX=1 TO LEN(S$)
6450 NX=ASC(MID$(S$,IX,1))-64
6460 IF NX<>SNX THEN PRINT IX;" ";NOM$(NX)
6470 NEXT
6480 PRINT IX;" ";NOM$(SNX)
6490 RETURN
6500 :
8000 REM -- DATA carte du système:
8010 DATA SOLEIL, 1
8020 DATA 2,4, 17,30, 20,80, 0,0
8030 DATA MERCURE,2
8040 DATA 1,4, 3,6, 0,0_
8050 DATA APHRODITE,3
8060 DATA 2,6, 5,7, 0,0
8070 DATA LUNE, 4
8080 DATA 5,1  6,10, 0,0
8090 DATA TERRA FIRMA, 5
8100 DATA 3,7, 7,8, 6,8 4,1, 0,0
8110 DATA ROYAUME DES DIABLES, 6
8120 DATA 5,8, 7,1, 9,12, 8,12, 5,10, 0,0
8130 DATA PHOBIA, 7
8140 DATA 5,8, 9,13, 8,11, 6,1, 0,0
8150 DATA EUREKA, 8
8160 DATA 6,12, 7,11, 9,1, 10,16, 11,25, 0,0
8170 DATA GALILEE, 9
8180 DATA 10,15, 8,1, 6,12, 7,13, 0,0
8190 DATA SOPHIA ANTIPOLIS, 10
8200 DATA 9,15, 12,24, 11,20, 8,16, 0,0
8210 DATA OMBRIE, 11
8220 DATA 8,25, 10,20, 12,17, 13,28, 0,0
8230 DATA TRIDENT, 12
8240 DATA 10,24, 14,22, 13,20, 11,17, 18,2, 0,0
8250 DATA LIMBES, 13
8260 DATA 12,20, 14,1, 16,48, 15,23, 11,28, 0,0
8270 DATA GRAND BOULEVARD, 14
8280 DATA 12,22, 13,1, 18,25, 0,0
8290 DATA HADES, 15
8300 DATA 13,223, 16,232, 0,0
8310 DATA BETA GLOBULINE, 16
8320 DATA 13,48, 17,64, 15,32, 0,0
8330 DATA MAXIMA CENTAURI, 17
8340 DATA 16,64, 1,30, 19,88, 0,0
8350 DATA POSEIDON, 18
8360 DATA 12,2, 14,25, 13,25, 0,0
8370 DATA ULTIMA THULE, 19
8380 DATA 17,88, 20,96, 0,0
8390 DATA OMEGA SOLARIS, 20
8400 DATA 1,80, 19,95, 0,0
8410 DATA NULLE PART, 0
```

Variantes de basic

Le programme ci-contre est rédigé pour le Commodore 64.

Pour le Spectrum :

Supprimez les % de toutes les variables, remplacez NOM$() par N$(), LIEN%(,) par L(,), NX%() par N(), RV() par R(), TAILLE% par TA et RF$ par F$ tout au long du programme, et placez entre guillemets les noms de planètes indiqués dans les DATA.

Procédez aux modifications suivantes :

```
1010 DIM N$(20,15),L(20,20)
1012 DIM R$(NR,20),R(NR),D$(3,25)

1710 LET S$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"(TO
     S)

1800 LOAD F$ DATA D$()
1810 LET B$=D$(1)
1820 LET B=VAL(D$(2)):LET SN=VAL(D$(3))

2250 LET P2=CODE S$(S TO S)-64
3090 LET S$=R$(R1)(P1 TO P2-P1)
3340 LET SX=CODE S$(S TO S)-64
3400 LET S$=S$ + R$(RS)(S TO S)

5105 IF INKEY$="" THEN GOTO 5105
5134 IF INKEY$="" THEN GOTO 5134
5144 IF INKEY$="" THEN GOTO 5144

5530 LET D$(1)=B$:LET D$(2)=STR$(B)
5540 LET D$(3)=STR$(SN)
5550 SAVE F$ DATA D$()

6050 LET X$=S$(I TO I)
6060 LET Y$=S$(J TO J)
6070 LET S$=S$(TO I-1)+Y$+S$(I+1 TO)
6080 LET S$=S$(TO J-1)+X$+S$(J+1 TO)

6450 N=CODE S$(I TO I)-64
```

Auto-diagnostic

Prolog possède la propriété de se servir de ses propres programmes comme données. Cela le destine tout particulièrement à être utilisé en programmation pour l'intelligence artificielle.

PROLOG a été choisi par les Japonais comme langage de base pour leurs ordinateurs de cinquième génération. Deux raisons motivent ce choix. D'abord, les termes de PROLOG peuvent prendre une forme très semblable à celle des relations d'une base de données relationnelles. Les Japonais voient leur machine du futur sous la forme d'une base de données sophistiquée.

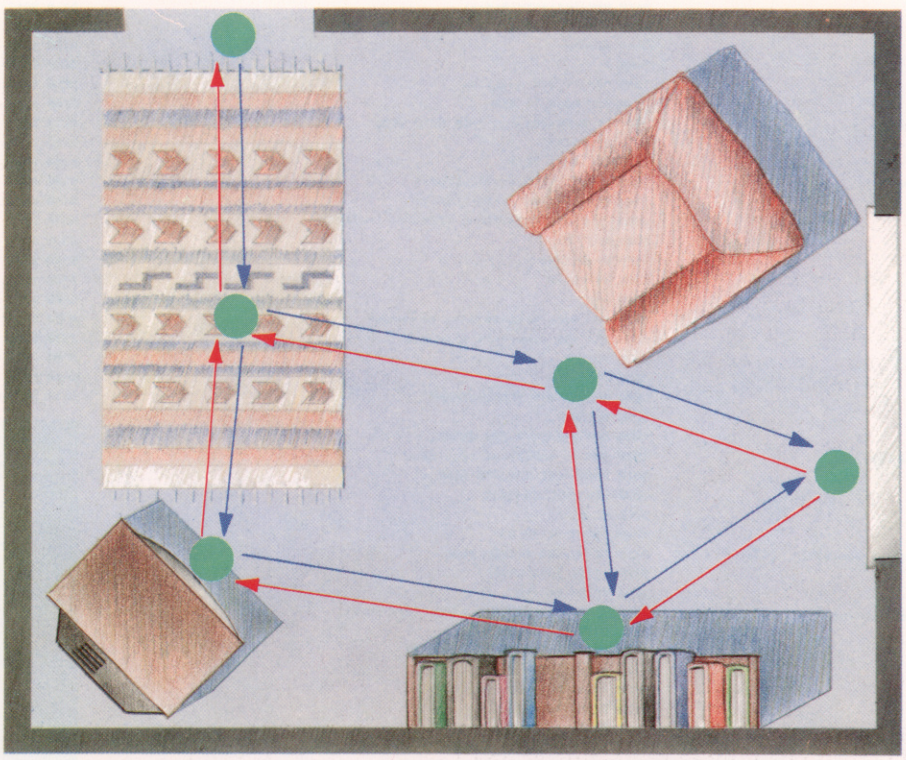
En deuxième lieu, PROLOG est le langage idéal pour l'écriture de programmes d'intelligence artificielle. Cela est principalement dû à sa faculté d'utiliser ses propres programmes comme données, mais aussi au fait que son interpréteur-temps d'exécution est assez semblable à un moteur d'inférences de nombreux systèmes experts modernes.

Nous donnons ici un programme qui illustre de manière simple l'intérêt d'utiliser les instructions de programme comme données. Il s'agit de la simulation des déplacements d'un robot dans une pièce. Il met en évidence quelques-unes des caractéristiques les plus avancées de PROLOG et nous donne une idée sur ses principes généraux.

La pièce est définie comme un ensemble de positions et de trajets. Nous voulons que le robot obéisse à des ordres tels que « va de la télévision au fauteuil ». Il doit trouver de lui-même le trajet le plus court. Le but donné va (Là-bas) signifie « va d'où tu es à Là-bas, par le chemin le plus court ».

Pour exprimer va (Là-bas), le programme utilise deux clauses. La première est destinée à éliminer le cas où l'on demande au robot d'aller là où il est déjà. Le programme a besoin d'un fait dans sa base de données qui lui donne la position en cours du robot (à(la porte)). La seconde clause utilise ce fait pour définir la position de départ. Elle appelle ensuite une autre procédure : va(Position1,Position2).

Du fait que va(Position1,Position2) peut être utilisée séparément de va(Position), elle vérifie d'abord que Ici et Là-bas sont des positions qui lui sont connues, et même qu'Ici est la position en cours.



Si ces vérifications échouent, la clause est abandonnée et PROLOG passe à la suivante. La deuxième clause va(Ici,Là-bas) sert à déplacer le robot sur ici s'il n'y est pas déjà. Pour cela, elle appelle va(Ici), puis, lorsque cela a réussi, elle appelle va(Là-bas). La lecture sous forme de procédure de cette clause est : « Pour aller d'(Ici) à (Là-bas), lorsque le robot n'est pas déjà (Ici), aller d'abord (Ici) puis (Là-bas). » La troisième clause va(Ici, Là-bas) ne sert qu'à afficher un message d'erreur quand aucune des positions n'est connue.

En supposant que les vérifications préalables pour la première clause va(Ici,Là-bas) aient été faites avec succès, l'objectif intermédiaire suivant à vérifier est touttrouver. C'est un prédicat parfois incorporé à PROLOG, mais on peut l'ajouter très facilement (comme FORTH, PROLOG peut être facilement étendu par la définition de nouveaux prédicats, puisque les prédicats utilisateur ont le même statut que ceux du système). Touttrouver comporte trois arguments.

Parcourir une pièce
L'environnement du robot est défini comme un ensemble de positions sur lesquelles il possède des informations : c'est-à-dire qu'elles ont été entrées dans sa base de données sous forme de faits. Elles sont reliées entre elles par un ensemble de chemins que le robot peut parcourir, et qui sont sauvegardés au format chemin(tv,bibli). Le programme de simulation du robot mobile indique la manière dont cette connaissance est utilisée pour trouver le chemin du déplacement de A à B.

Le premier est le nom d'une variable, le deuxième est un objectif, et le troisième une variable liste (RL). PROLOG essaye d'atteindre le but fixé (planifier(Ici,Là-bas,[],Route) dans le cas présent) autant de fois qu'il le peut. La variable du premier argument doit correspondre à l'une de celles de l'objectif. A chaque fois que cela se produit, la valeur prise par la variable est ajoutée à la liste du troisième argument. Par exemple, l'objectif planifier(Position1, Position2,[], Route) trouverait une route entre la Position1 et la Position2, et mettrait ce résultat dans la variable Route. Ici, touttrouver recense toutes les routes possibles entre Ici et Là-bas, et en dresse une

liste. Nous devons remarquer que nous n'avons pas encore eu besoin de définir la procédure pour touttrouver. En fait, il est normal avec PROLOG de développer des programmes de cette manière.

De même, nous n'avons pas encore vu le prédicat lapluscourte(RL,CourteRoute). Sa fonction est de prendre une liste des listes (Liste1), et de créer une nouvelle liste (Liste2) qui ne comporte que la plus courte liste trouvée. Du fait que toutes les listes de Liste1 sont des routes, la plus courte liste sera la plus courte route.

Nous avons atteint le point où notre robot a calculé la plus courte route jusqu'à sa destination. Il ne lui reste qu'à y aller. Cela est possible grâce à une autre procédure virtuelle, avancer(Route). Nous avons défini avancer comme étant l'énumération des routes. Mais une route est en réalité une liste d'objectifs PROLOG. Ces derniers auraient pu lui être communiqués comme un programme à exécuter au lieu d'être affichés à l'écran.

Définissons le prédicat sediriger(Position) pour permettre au capteur de déterminer la Position afin de diriger le robot dans cette direction. déplacer(Position1, Position2) est pour sa part défini comme devant déplacer le robot d'une position à l'autre.

Ce programme de déplacement de robot est écrit par le programme principal lui-même à l'intérieur de la procédure planifier. C'est en réalité assez simple. On pourrait résumer ainsi l'algorithme utilisé : pour trouver une route de A à B, trouver d'abord une route de A à C, puis de C à B. La procédure planifier est récursive. La première clause sert à arrêter la récursion lorsque la destination a été atteinte (il y a toujours une route de A à A).

C'est en fait la deuxième clause qui fait tout le travail. Sa lecture déclarative est qu'il existe un plan pour aller de A à B par la route R, si :

1. il existe un chemin de A à C, *et*
2. C n'est pas sur la liste déjà vue, V (C lui est alors ajoutée pour vous éviter ultérieurement de tourner en rond), *et*
3. il existe un plan pour se rendre de C en B par la route R1.

Quand toutes ces conditions sont vraies, on en conclut que la bonne route, R, est la liste contenant un programme pour aller de A à C, ajoutée à la « route parcourue » (contenue dans R1). De telles procédures récursives sont connues pour être difficiles à suivre (mettez-les par écrit), mais elles donnent une terrible efficacité à PROLOG.

Simulation de robot mobile
Version prolog standard

```
à(la porte).                 /* la position en cours du robot*/
va(Là-bas):-                 à(Là-bas), écrire (« J'y suis déjà ! »), nl,nl,nl.
va(Là-bas):-                 à(Ici),va(Ici,Là-bas).
va(Ici,Là-bas):-             position(Ici),position(Là-bas),à(Ici),touttrouver(Route,planifier(Ici,Là-bas,[],Route,),RL),
                             lapluscourte(RL,CourteRoute),avancer(CourteRoute),retirer(à(Ici)),ajoutera(à(Là-bas)),direoù.
va(Ici,Là-bas):-             non(à(Ici)),écrire(« je ne suis pas à »),écrire(Ici),écrire(« je vais donc d'abord là-bas. »),nl,nl,va(Ici),va(Là-bas).
va(Ici,Là-bas)               écrire(« je ne peux aller que là où je connais ma destination »),nl,nl.
planifier(A,A,-,R).
planifier(A,B,V,R):-         chemin(A,C),non(appartient(C,V)),ajouter([C],V,V1),planifier(C,B,V1,R1),ajouter([face(C),orienter(A,C)]R1,R).
avancer(Route):-             écrire(Route),nl,nl. /*défini plus tard*/
direoù:-                     à(Position),écrire(« je suis à »),écrire(Position),nl,nl,nl.

/*liste des chemins connus par le robot*/    /*et liste des positions qu'il connaît*/

chemin(bibli,chaise).                      position(porte).
chemin(chaise,bibli).                      position(tapis).
chemin(bibli,tv).                          position(tv).
chemin(tv,bibli).                          position(bibli).
chemin(tv,tapis).                          position(fenêtre).
chemin(tapis,tv).                          position(chaise).
chemin(tapis,chaise).
chemin(chaise,tapis).
chemin(porte,tapis).
chemin(tapis,porte).
chemin(fenêtre,chaise).
chemin(chaise,fenêtre).
chemin(fenêtre,bibli).
chemin(bibli,fenêtre).
```

Version micro-prolog

```
(à la porte)          /*position en cours du robot*/
((va X)               (à X) (P j'y suis déjà !) PP,PP,PP)
((va X)               (à Y) (va Y X)
((va X Y)             (position X) (position Y) (à X)
                      (touttrouver Z (planifier XY (!Z)X)
                      (lepluscourt xy)
                      (aller y)
                      (DELC ((à X)) ((ADDCL ((à Y)))
                      (direoù))
((va Y X)             (NONPAS à X)
                      (P Je ne suis pas à) (PX)
                      (P j'y vais donc en premier.) PP PP
                      (va X) (va Y))
((va X Y)             (P Je ne peux aller qu'aux destinations que je connais) PP PP)
(planifier X X x1 Z)
((planifier X Y x Z)  (chemin X X1) (N'appartient pas X1 x) (ajouter (X1) x x1)
                      (planifier X1 Y x1 Z1)
                      (ajouter ((orienter X1) (avancer X X1)) Z1 Z))
((aller Z)            (P Z) PP PP)
((direoù)             (à X) (P Je suis en X) PP PP PP)

/*Liste des chemins que le robot connaît*/    /*et liste des positions qu'il connaît*/

(chemin bibli chaise)                      (position porte)
(chemin chaise bibli.)                     (position tapis)
(chemin bibli tv)                          (position tv)
(chemin tv bibli)                          (position bibli)
(chemin tv tapis)                          (position fenêtre)
(chemin tapis tv)                          (position chaise)
(chemin tapis chaise)
(chemin chaise tapis)
(chemin porte tapis)
(chemin tapis porte)
(chemin fenêtre chaise)
(chemin chaise fenêtre)
(chemin fenêtre bibli)
(chemin bibli fenêtre)
```

Manœuvres programmées

Les informations relatives à l'environnement du robot sont entrées dans la base de données par l'utilisateur. PROLOG utilise la procédure va(position1,position2) pour établir la route entre A et B. Pour cela, il cherche d'abord la route de A à C, puis celle de C à B. Notez l'utilisation du signe de soulignement comme argument à l'intérieur de la clause planifier(A,A,_,_B). Ce symbole est utilisé comme « variable anonyme », ou « joker », et ne prend pas de valeur lors de l'exécution de la clause. Ici, la variable anonyme est utilisée dans la clause qui démontre qu'il y a toujours un chemin entre A et A.

Nouveau Monde II

Nous mettons un terme à notre jeu de simulation consacré à un voyage dans le Nouveau Monde en donnant ici la seconde partie du listage complet.

Ce programme est conçu pour tourner sur un Commodore 64, mais quelques modifications de peu d'ampleur lui permettront de fonctionner sur Amstrad. Il faudra remplacer par CLS toute mention de PRINT CHR$(147). A chaque fois que l'ordinateur attend que vous appuyiez sur une touche, comme dans :

<n° ligne>GET I$:IFI$D=«»THEN<n° ligne>

vous devrez mettre à la place :

<n° ligne> I$=«»:WHILE I$=«»:I$=INKEY$:WEND

Enfin vous devrez introduire la ligne suivante, qui provoquera la sélection d'un affichage en mode 40 colonnes compatible avec le micro-ordinateur Amstrad :

5 MODE 1

```
6530 REM CANOT
6535 IFM(1)=1THENRETURN
6536 PRINTCHR$(147)
6540 M(1)=1
6550 S$="VOUS APERCEVEZ UN CANOT*":GOSUB9100
6552 S$="DERIVANT AU LOIN*":GOSUB9100
6554 PRINT:GOSUB9200
6556 S$="A LA LUNETTE VOUS CONSTATEZ*":GOSUB9100
6558 S$="QU'IL CONTIENT*":GOSUB9100
6560 PRINT:GOSUB9200
6562 S$="4 PERSONNES*":GOSUB9100
6563 GOSUB9200
6563 GOSUB9200
6564 S$="ET UN GROS COFFRE !*":GOSUB9100
6566 PRINT:GOSUB9200
6568 S$="SI VOUS CHANGEZ DE CAP*":GOSUB9100
6570 S$="POUR LES SAUVER*":GOSUB9100
6572 S$="VOUS PERDREZ 2 JOURS DE ROUTE*":GOSUB9100
6574 PRINT:GOSUB9200
6576 S$="VOULEZ-VOUS LES SAUVER ? (O/N)*":GOSUB
9100
6578 INPUTI$:I$=LEFT$(I$,1)
6580 IFI$<>"O"ANDI$<>"N"THEN6578
6585 IFI$="O"THEN6600
6588 PRINT:GOSUB9200
6590 S$="LE CANOT DISPARAIT AU LOIN...*":GOSUB9100
6592 PRINT:GOSUB9200
6594 S$=K$:GOSUB9100
6596 GETI$:IFI$=""THEN6596
6599 RETURN
6600 PRINT:GOSUB9200
6610 EW=EW+2/7
6625 IFCN<>16 THEN6630
6627 S$="VOUS NE POUVEZ LES PRENDRE A BORD*":GOSUB9100
6628 S$="VOUS N'AVEZ PAS LA PLACE*":GOSUB9100
6629 GOTO6592
6630 X=16-CN:IFX>3THEN6635
6632 S$="VOUS N'AVEZ DE PLACE QUE POUR* ":GOSUB9100
6633 PRINTX;" PERSONNES DE PLUS A BORD"
6634 PRINT:GOSUB9200
6635 S$="VOUS SAUVEZ :*":GOSUB9100
6638 X=0
6640 FORT=1TO16
6645 IFTS(T,1)<>0THEN6679
6650 X=X+1
6655 IFX>4THENT=16:GOTO6679
6660 CN=CN+1
6665 TS(T,1)=INT(RND(1)*5)+1
6668 TS(T,2)=INT(RND(1)*50)+50
6670 PRINT"1 ";C$(TS(T,1))
6679 NEXT
6680 PRINT:GOSUB9200
6682 S$="LE COFFRE CONTIENT:*":GOSUB9100
6685 FORT=1TO4
6690 X=INT(RND(1)*10)+10
6692 PRINTX;U$(T);"S DE ";P$(T)
6693 IFPA(T)=-999THENPA(T)=0
6694 PA(T)=PA(T)+X
6695 NEXT
6699 GOTO6592
```

La peste peut frapper l'équipage...

```
6700 REM LA PESTE
6705 IFM(2)=1THENRETURN
6706 PRINTCHR$(147)
6710 M(2)=1
6712 S$="LA PESTE FRAPPE !*":GOSUB9100
6714 PRINT:GOSUB9200
6716 X=1
6718 FORT=1TO16
6720 IFTS(T,1)=2ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=0:T=16
6722 NEXT
6724 Y=1
6726 IFOA(1)<>0ANDOA(1)<>-999THENY=0
6730 Z=((X+Y)*10)+5
```

et les marins résisteront d'autant mieux à l'épidémie que vous aurez embauché un médecin et acheté des médicaments avant de quitter le port.

```
6732 I$="VOUS N'AVEZ":IFX=0ANDY=0THEN6740
6734 IFX=1THENS$="AUCUN MEDECIN ":GOSUB9100:I$="ET"
6736 IFY=1THEN S$=I$+"AUCUN MEDICAMENT*":GOSUB9100
6740 S$="BEAUCOUP D'HOMMES SONT TOUCHES*":GOSUB9100
6745 PRINT:GOSUB9200
6750 X=0
6755 FORT=1TO16
6756 IFRND(1)<.3THEN6775
6760 IFTS(T,2)=0ORTS(T,2)=-999THEN6775
6765 TS(T,2)=TS(T,2)-Z
6770 IFTS(T,2)<1THENS(T,2)=-999:X=X+1
6775 NEXT
6776 IFY=1THEN6780
6777 S$="LA MOITIE DES MEDICAMENTS EST CONSOMMEE*":
GOSUB9100
6778 OA(1)=INT((OA(1)/2)+.5)
6780 PRINT:GOSUB9200
6785 IFX=0THEN6797
6790 PRINT"ET";X;
6792 S$="MEMBRES D'EQUIPAGE MEURENT*"
6794 IFX=1THENS$="MEMBRE D'EQUIPAGE MEURT*"
6795 GOSUB9100
6796 PRINT:GOSUB9200
6797 S$=K$:GOSUB9100
6798 GETI$:IFI$=""THEN6798
6799 RETURN
```

Des pirates peuvent attaquer le navire...

```
6800 REM PIRATES
6805 IFM(3)=1 THENRETURN
6810 X=0
6812 FORT=1TO16
6814 IFTS(T,2)=0ORTS(T,2)=-999THENX=X+1
6815 NEXT
6816 IFX=16 THEN RETURN
6818 M(3)=1
6820 PRINTCHR$(147)
```

Y a-t-il des fusils à bord pour vous défendre de façon à avoir le moins de pertes possible ?

```
6822 S$="DES PIRATES ATTAQUENT LE NAVIRE !*":GOSUB9100
6824 PRINT:GOSUB9200
6825 K=2
6826 IFOA(2)=0OROA(2)=-999THENK=4
6828 S$="EN DEPIT DE VOS FUSILS*"
6830 IFK=4THENS$="VOUS N'AVEZ PAS DE FUSILS*"
6832 GOSUB9100
6835 X=0
6836 FORT=1TO16
6838 IFTS(T,2)=0ORTS(T,2)=-999THEN6845
```

```
6840 X=X+1:TS(T,2)=-999
6842 IFX=KTHENT=16
6845 NEXT
6850 PRINTX:
6855 S$="DE L'EQUIPAGE EST TUE*"
6856 IFX>1THENS$="DE L'EQUIPAGE SONT TUES*"
6860 GOSUB9100
6865 PRINT:GOSUB9200
6890 S$=K$:GOSUB9100
6895 GETI$:IFI$=""THEN6895
6899 RETURN
6900 REM GOUVERNAIL
6905 IFM(4)=1THENRETURN
6910 PRINTCHR$(147)
6915 M(4)=1
6920 S$="PROBLEMES DE GOUVERNAIL!*":GOSUB9100
6925 PRINT:GOSUB9200
6928 X=4
6930 FORT=1TO16
6935 IFTS(T,1)=3ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=1:T=16
6938 NEXT
6940 S$="VOUS AVEZ UN CHARPENTIER, MAIS*"
6945 IFX=4THENS$="VOUS N'AVEZ PAS DE CHARPENTIER, ET*"
6950 GOSUB9100
6955 S$="LE VOYAGE PRENDRA*":GOSUB9100
6960 PRINTX:"SEMAINES DE PLUS."
6965 EW=EW+X
6967 PRINT:GOSUB9200
6969 S$=K$:GOSUB9100
6970 GETI$:IFI$=""THEN6970
6975 RETURN
7000 REM TEMPETE
7005 IFM(5)=1THENRETURN
7010 PRINTCHR$(147)
7015 M(5)=1
7020 S$="UN ORAGE*":GOSUB9100
7022 S$="VOUS SECOUE!*":GOSUB9100
7025 PRINT:GOSUB9200
7028 X=2
7030 FORT=1TO16
7035 IFTS(T,1)=4ANDTS(T,2)<>0ANDTS(T,2)<>-999THENX
=1:T=16
7038 NEXT
7040 S$="VOUS AVEZ UN NAVIGATEUR, MAIS*"
7045 IFX=2THENS$="VOUS N'AVEZ PAS DE NAVIGATEUR ET*"
7049 GOTO6950
```

Si vous commencez à manquer de provisions, vous pourrez aborder dans l'île pour renouveler vos stocks, mais le voyage sera alors plus long.

```
7050 REM L'ILE
7055 IFM(6)=1THENRETURN
7060 PRINTCHR$(147)
7065 M(6)=1
7070 S$="VOS CARTES INDIQUENT UNE ILE*":GOSUB9100
7071 S$="OU VOUS POURREZ*":GOSUB9100
7072 S$="RENOUVELER VOS PROVISIONS*":GOSUB9100
7073 S$="MAIS VOUS Y RENDRE*":GOSUB9100
7074 S$="VOUS PRENDRA DU TEMPS EN PLUS*":GOSUB9
100
7075 PRINT:GOSUB9200
7080 S$="VOULEZ-VOUS Y ALLER ? (O/N)":GOSUB9100
7082 INPUTI$:I$=LEFT$(I$,1)
7084 IFI$<>"O"ANDI$<>"N"THEN7082
7086 PRINT:GOSUB9200
7090 IFI$="N"THEN7145
7100 S$="VOUS ATTEIGNEZ L'ILE*":GOSUB9100
7105 S$="ET Y RECUEILLEZ:*":GOSUB9100
7106 IFB$="N"THEN7110
7107 PRINT:GOSUB9200
7108 PRINT"RIEN DU TOUT!":GOSUB9200
7109 S$="(SOUVENEZ-VOUS DE L'ALBATROS!)":GOSUB9100:
GOTO7130
7110 FORT=1TO4
7112 IFRND(1)<.25THEN7129
7115 X=INT(RND(1)*10)+5
7120 PRINTX;U$(T);"S DE ";P$(T)
7122 IFPA(T)=-999THENPA(T)=0
7125 PA(T)=PA(T)+X
7129 NEXT
7130 S$="LE VOYAGE VA PRENDRE*":GOSUB9100
7135 X=INT(RND(1)*2)+1
7139 PRINTX;:S$="SEMAINES DE PLUS*":GOSUB9100
7140 EW=EW+X
7145 PRINT:GOSUB9200
7150 S$=K$:GOSUB9100
7155 GETI$:IFI$=""THEN7155
7159 RETURN
```

Sept facteurs peuvent se combiner pour provoquer une mutinerie au sein de l'équipage. Ce sous-programme crée un facteur de mutinerie, MF, en analysant les conditions qui règnent à la fin de chaque semaine du voyage, et en ajoutant à MF les facteurs correspondants.

```
7200 REM MUTINERIE
7210 MF=0
7215 IFH$="O"THENMF=MF+30
7220 NC=0
7225 FORT=1TO16
7228 IFTS(T,1)=SANDTS(T,2)<>0ANDTS(T,2)<>-999THENN
C=1:T=16
7230 NEXT
7235 IFNC=0THENMF=MF+30
7240 IFA$="O"THENMF=MF-20
7245 IFB$="O"THEMMF=MF+30
7250 IFCN>12THENMF=MF+30
7255 IFWT>MOTHENMF=MF+30
7260 IFWK>8THENMF=MF+((WK-8)*10)
7275 MF=MF+INT(RND(1)*30)
```

Si le facteur de mutinerie atteint au moins 75 (mais reste au-dessous de 100), le joueur est prévenu qu'une certaine agitation règne.

```
7280 IFMF<75 THENRETURN
7282 PRINTCHR$(147)
7284 IFMF>100THEN7300
7285 S$="LES CONDITIONS DE VIE*":GOSUB9100
7286 S$="SE FONT MAUVAISES*":GOSUB9100
7287 S$="ET CERTAINS PARLENT*":GOSUB9100
7288 S$="DE SE MUTINER!!!":GOSUB9100
7290 PRINT:GOSUB9200
7292 S$=K$:GOSUB9100
7294 GETI$:IFI$=""THEN7294
7299 RETURN
```

Si MF dépasse 100, la rébellion éclate, et l'équipage expose son mécontentement avant d'abandonner le capitaine à bord d'un canot.

```
7300 PRINTCHR$(147)
7305 PRINT:GOSUB9200
7310 S$="L'EQUIPAGE S'EST MUTINE*":GOSUB9100
7312 S$="PARCE QUE:*":GOSUB9100
7313 X=0
7314 IFH$<>"O"THEN7320
7315 GOSUB9200:X=X+1:PRINTX:
7316 S$="DES HOMMES ONT ETE MIS":GOSUB9100
7318 S$="A DEMI-RATION*":GOSUB9100
7320 IFNC<>0THEN7325
7321 GOSUB9200:X=X+1:PRINTX:
7322 S$="IL N'Y A PAS DE CUISINIER*":GOSUB9100
7324 S$="  ET LA NOURRITURE EST INFECTE":GOSUB9100
7325 IFB$<>"O"THEN7330
7326 GOSUB9200:X=X+1:PRINTX:
7327 S$="VOUS AVEZ TUE L'ALBATROS!":GOSUB9100
7330 IFCN<13THEN7335
7331 GOSUB9200:X=X+1:PRINTX:
7332 S$="LE NAVIRE EST SURPEUPLE*":GOSUB9100
7335 IFMO>=WT THEN7340
     GOSUB9200:X=X+1:PRINTX:
7337 S$="IL N'Y A PLUS ASSEZ D'OR*":GOSUB9100
7338 S$="  POUR LES SALAIRES!":GOSUB9100
7340 IFWK<=8 THEN7350
7341 GOSUB9200:X=X+1:PRINTX:
7342 S$="ILS SONT EN MER*":GOSUB9100
7343 S$="  DEPUIS PLUS DE 8 SEMAINES!":GOSUB9100
7350 PRINT:GOSUB9200
7360 S$="L'EQUIPAGE S'EMPARE DU NAVIRE*":GOSUB9100
7362 GOSUB9200
7363 S$="ET VOGUE AU LOIN*":GOSUB9100
7370 GOSUB9200
7372 S$="IL VOUS LAISSE A LA DERIVE*":GOSUB9100
7373 S$="DANS UN CANOT*":GOSUB9100
7374 S$="DIEU AIT PITIE DE VOTRE AME!":GOSUB9100
7375 PRINT:GOSUB9200
7380 PRINT"  FIN DE LA PARTIE"
7382 END
```

Ces brefs sous-programmes sont utilisés tout au long de la partie et produisent divers effets, comme de ralentir l'affichage des messages.

```
9100 REM RALENTIT AFFICHAGE
9110 S=LEN(S$)
9115 S=S*20
9120 FOR H=1 TO S:NEXT
9130 PRINT
9140 PRINT:PRINT
9199 RETURN
9200 REM BOUCLE DE RETARD
9210 FORS5=1TO1000:NEXT
9299 RETURN
9300 REM REDUCTION PAR WF DE LA FORCE DE L'EQUIPAGE
9310 FORS1=1TO16
9315 IFTS(S1,2)=0THEN9340
9320 TS(S1,2)=TS(S1,2)-WF
9330 IFTS(S1,2)<1THENTS(S1,2)=-999
9340 NEXT
9399 RETURN
```

Le navire arrive à destination...

```
10000 REM ARRIVEE DANS LE NOUVEAU MONDE
10001 PRINTCHR$(147):GOSUB9200
10005 S$="VOUS VOICI DANS LE NOUVEAU MONDE+":GOSUB9100
10006 PRINT:GOSUB9200
10007 S$="SUR LES BORDS DU RIVAGE+":GOSUB9100
10009 S$="DES INDIGENES APPARAISSENT+":GOSUB9100
10010 PRINT:GOSUB9200
10015 IFOA(2)=0THEN10050
10017 S$="ILS SONT ARMES ET ONT L'AIR FEROCE!!":GOSUB
9100
10018 PRINT:GOSUB9200
10020 S$="OUVREZ-VOUS LE FEU ? (O/N)+":GOSUB9100
10022 INPUTI$:I$=LEFT$(I$,1)
10024 IFI$<>"N"ANDI$<>"O"THEN10022
10026 IFI$="N"THEN10050
10028 PRINT:GOSUB9200
10030 S$="DE NOMBREUX INDIGENES SONT TUES+":GOSUB9100
10032 S$="MAIS LA NUIT+":GOSUB9100
10034 S$="D'AUTRES REVIENNENT+":GOSUB9100
10036 S$="ET BRULENT VOTRE VAISSEAU!!+":GOSUB9100
10038 PRINT:PRINT:GOSUB9200
10040 S$="  FIN DE LA PARTIE":GOSUB9100
10042 END
10044 GOTO10042
10050 S$="ILS VOUS MENENT A LEUR CHEF+":GOSUB9100
10052 S$="IL A DEJA RENCONTRE+":GOSUB9100
10054 S$="DES EUROPEENS ET SE MONTRE AMICAL+":GOSUB9100
10056 GOSUB9200
10058 S$="L'EQUIPAGE EST ACCUEILLI AVEC JOIE+":GOSUB9100
10060 PRINT:GOSUB9200
10062 S$="LE LENDEMAIN LES ECHANGES COMMENCENT+":GOSUB9100
10064 PRINT:GOSUB9200
10066 S$=K$:GOSUB9100
10068 GETI$:IFI$=""THEN10068
10069 RETURN
```

et les échanges commencent...

```
10070 PRINTCHR$(147):GOSUB9200:REM COMMERCE
10072 IFOA(2)=0THEN10088
10074 S$="LE CHEF NE VEUT PAS+":GOSUB9100
10076 S$="DE VOS FUSILS - TROP DANGEREUX*!":GOSUB9100
10078 PRINT:GOSUB9200
10080 IFOA(3)<>0ORDA(4)<>0ORDA(5)<>0ORDA(6)<>0THEN
10085 S$="IL NE RESTE PLUS RIEN+":GOSUB9100
10090 S$="A ECHANGER+":GOSUB9100
10095 GOTO10038
10100 S$="EN ECHANGE DES COUTEAUX+":GOSUB9100
10102 S$="SEL, TISSUS OU BIJOUX+":GOSUB9100
10104 S$="IL OFFRE DES PERLES, DES STATUES+":GOSUB9100
10106 S$="ET DES EPICES+":GOSUB9100
10108 PRINT:GOSUB9200
10110 S$="AU DEPART DU PORT ELLES+":GOSUB9100
10112 S$="VALAIENT :+":GOSUB9100
10114 S$="PERLES  -2 PCS D'OR CHACUNE+":GOSUB9100
10116 S$="STATUES -2 PCS D'OR CHACUNE+":GOSUB9100
10118 S$="EPICES  -1 PC  D'OR LE GRAMME+":GOSUB9100
10120 PRINT:GOSUB9200
10122 S$="MAIS CELA AURA PEUT-ETRE+":GOSUB9100
10124 S$="CHANGE AU RETOUR+":GOSUB9100
10125 PRINT:GOSUB9200:S$=K$:GOSUB9100
10126 GETI$:IFI$=""THEN10126
10130 FORT=3TO6
10135 IFOA(T)=0THEN10200
10140 PRINTCHR$(147):GOSUB9200
10145 PRINT"VOUS AVEZ:";OA(T);
10150 IFT=3THENS$="SACS DE SEL+"
10151 IFT=4THENS$="BALLOTS DE TISSU+"
10152 IFT=5THENS$="COUTEAUX+"
10153 IFT=6THENS$="BIJOUX+"
10155 GOSUB9100
10156 PRINT:GOSUB9200
10160 S$="LE CHEF VOUS EN OFFRE+":GOSUB9100
10165 PRINT" OU";OA(T)*EQ(T-2,1);"PERLES"
10166 PRINT" OU";OA(T)*EQ(T-2,2);"STATUES"
10167 PRINT" OU";OA(T)*EQ(T-2,3);"GRAMMES D'EPICE"
10168 PRINT:GOSUB9200
10170 S$="VOULEZ-VOUS DES PERLES, DES STATUES+":
GOSUB9100
10172 S$="OU DES EPICES ?+":GOSUB9100
10174 S$="(ENTREZ 1,2 OU 3)+":GOSUB9100
10175 INPUTI$
10176 I=VAL(I$):IFI<1ORI>3THEN10174
10180 AO(I)=AO(I)+(OA(T)*EQ(T-2,I))
10190 PRINT:PRINT"LES ";T$(I);" SONT RAMENEES A BORD"
10192 S$=K$:GOSUB9120
10194 GETI$:IFI$=""THEN10194
10200 NEXT
10210 PRINT:PRINT:GOSUB9200
10215 S$="  FIN DES ECHANGES+":GOSUB9100
10216 PRINT:GOSUB9200
10218 S$="VOUS AVEZ ACQUIS:+":GOSUB9100
10220 PRINTAO(1);"PERLES"
10222 PRINTAO(2);"STATUES"
10224 PRINTAO(3);"GRAMMES D'EPICE"
10226 PRINT:GOSUB9200
10228 S$=K$:GOSUB9100
10229 GETI$:IFI$=""THEN10229
10230 RETURN
```

Voulez-vous participer à un coup de force local ? Le bénéfice est élevé, mais les risques aussi, au cas où les choses ne marcheraient pas bien...

```
10300 REM REVOLUTION
10305 IFOA(2)=0THENRETURN
10310 PRINTCHR$(147):GOSUB9200
10315 S$="AU COURS DE LA NUIT UN RIVAL+":GOSUB9100
10316 S$="DU CHEF VOUS REND VISITE+":GOSUB9100
10317 PRINT:GOSUB9200
10318 S$="IL VEUT ACQUERIR VOS FUSILS+":GOSUB9100
10320 S$="POUR UN COUP DE FORCE+":GOSUB9100
10322 PRINT:GOSUB9200
10324 S$="IL OFFRE 30 PERLES PAR ARME+":GOSUB9100
10326 S$="ACCEPTEZ-VOUS ? (O/N)+":GOSUB9100
10328 INPUTI$:I$=LEFT$(I$,1)
10330 IFI$<>"N"ANDI$<>"O"THEN10328
10332 IFI$="O"THEN10400
10334 PRINT:GOSUB9200
10336 S$="LE CHEF L'APPREND+":GOSUB9100
10338 S$="ET POUR VOUS REMERCIER+":GOSUB9100
10340 S$="VOUS DONNE DES PROVISIONS+":GOSUB9100
10344 GOSUB9200
10345 IF RND(1)<.75 THEN 10350
10346 S$="ET 50 PERLES+":GOSUB9100
10348 AO(1)=AO(1)+50
10350 PRINT:GOSUB9200
10352 S$=K$:GOSUB9100
10354 GETI$:IFI$=""THEN10354
10359 RETURN
10400 PRINTCHR$(147):GOSUB9200
10405 IFRND(1)<.75THEN10450
10410 S$="LE SOULEVEMENT REUSSIT!+":GOSUB9100
10412 PRINT:GOSUB9200
10415 S$="LE NOUVEAU CHEF+":GOSUB9100
10420 S$="VOUS OFFRE DES PROVISIONS+":GOSUB9100
10425 S$="POUR LE RETOUR.+":GOSUB9100
10429 AO(1)=AO(1)+(OA(2)*30):REM ADD PEARLS
10430 OA(2)=0
10431 GOTO10350
10450 S$="LE SOULEVEMENT ECHOUE!+":GOSUB9100
10452 PRINT:GOSUB9200
10455 S$="LE CHEF, FURIEUX+":GOSUB9100
10457 S$="BRULE VOTRE NAVIRE+":GOSUB9100
10458 S$="ET REPREND SON BIEN!+":GOSUB9100
10459 PRINT:GOSUB9200
10460 S$="  FIN DE LA PARTIE!!!+":GOSUB9100
10462 END
10464 GOTO10462
10500 REM FIN DE VOYAGE
```

Le navire revient ensuite à son point de départ et le programme procède à l'évaluation de vos profits, en vous précisant si oui ou non vous êtes un bon commerçant...

```
10501 PRINTCHR$(147):GOSUB9200
10505 S$="UN EQUIPAGE EN FORME ET+":GOSUB9100
10507 S$="DES VENTS FAVORABLES+":GOSUB9100
10508 S$="PERMETTENT UN VOYAGE DE 8 SEMAINES+":
GOSUB9100
10512 WW=0
10514 FORT=1TO5
10516 WW=WW+(8*CC(T)*WG(T))
10518 NEXT
10519 PRINT:GOSUB9200
10520 S$="TOTAL DES SALAIRES DU RETOUR+":GOSUB9100
10522 PRINTWW;"PIECES D'OR"
10524 PRINT:GOSUB9200
10526 S$="AU RETOUR+":GOSUB9100
10528 S$="VOUS VENDEZ VOS TROUVAILLES+":GOSUB9100
10530 S$="ELLES VALENT MAINTENANT:+":GOSUB9100
10532 PRINT"PERLES ";V2(1);"PIECES D'OR"
10534 PRINT"STATUES ";V2(2);"PIECES D'OR"
10536 PRINT"EPICES ";V2(3);"PIECES D'OR"
10538 PRINT:S$=S$+"SOIT UN TOTAL DE+":GOSUB9100
10540 X=(AO(1)*V2(1))+(AO(2)*V2(2))+(AO(3)*V2(3))
10542 PRINTX;"PIECES D'OR"
10545 PRINT:S$=K$:GOSUB 9100:PRINT
10547 GET I$:IF I$="" THEN 10547
10550 S$="VOUS AVEZ MAINTENANT:+":GOSUB9100
10552 PRINTMO+X;"PIECES D'OR"
10555 PRINT:GOSUB9200
10556 S$="LE TOTAL DES SALAIRES POUR LE VOYAGE EST DE+":
GOSUB9100
10557 PRINTWT+WW;"PIECES D'OR"
10559 PRINT:GOSUB9200
10560 S$="IL VOUS RESTE DONC:+"GOSUB9100
10562 Z=MO+X-WT-WW
10565 PRINTZ;"PIECES D'OR"
10566 PRINT:GOSUB9200
10567 PRINT:S$="VOUS ETES VRAIMENT:+":GOSUB9100:PRINT
10568 IF Z>3200 THEN S$="UN SUPER-CAPITALISTE+":GOSUB9
100:END
10569 IF Z>2500 THEN S$="UN MAITRE MARCHAND+":GOSUB9100:
END
10570 IF Z>2000 THEN S$="UN HONNETE CABOTEUR+":GOSUB
9100:END
10571 IF Z>1000 THEN S$="UN CABOTEUR DE PEU DE POIDS+":
GOSUB9100:END
10572 S$="PEU DOUE POUR LE COMMERCE+"
10573 GOSUB9100:END
```

Mesures de bande

Nous étudions les routines de ROM utilisées par le système d'exploitation du Spectrum pour accéder au système de fichier sur bande.

Un Sinclair Spectrum standard n'est pourvu que d'une interface de bande pour lui permettre de sauvegarder et de charger les programmes et les données. L'adjonction de l'Interface 1 nous donne toutefois accès aux Microdrive, à l'interface série et aux réseaux locaux. Ceux-ci peuvent être considérés comme d'autres systèmes de fichiers. Sur le Spectrum, nous utilisons une autre syntaxe pour différencier les commandes concernant le système de fichier sur bande de tous les autres systèmes de fichiers disponibles sur l'ordinateur. Par exemple, la commande :

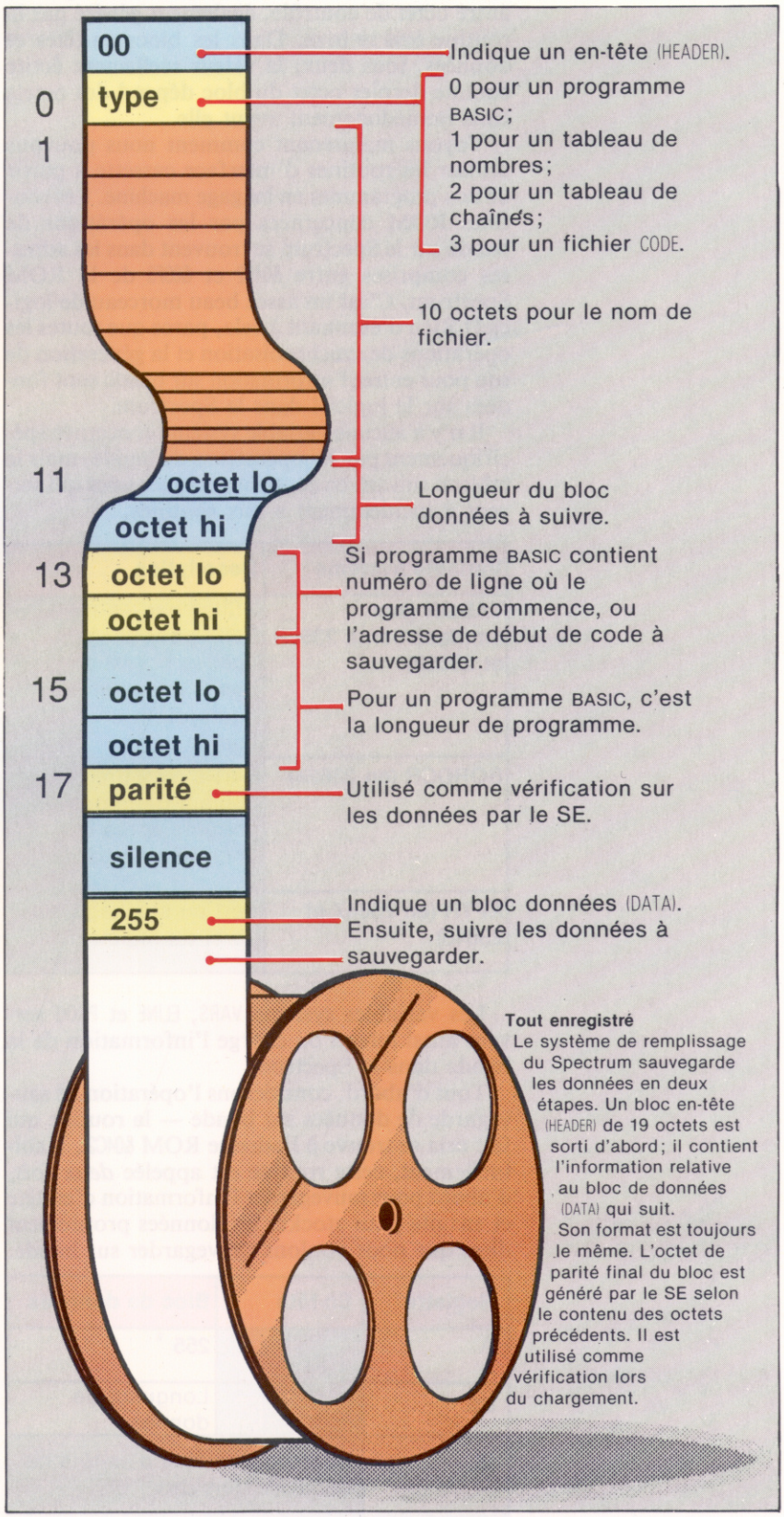
```
SAVE « fred 1» LINE 1
```

sauvegardera un programme BASIC sur bande afin qu'il commence à tourner à partir de la ligne 1 où il est rechargé dans le Spectrum. De même, pour sauvegarder un programme sur le système de fichier Microdrive, il faut employer la commande très compliquée :

```
SAVE *«m»;1;« fred 1 » LINE 1
```

La partie « m » de l'instruction spécifie que le Microdrive est utilisé; le 1 suivant indique le numéro du lecteur requis; et le nom de fichier — « fred 1 » dans ce cas — suit ce dernier. Nous verrons ultérieurement comment fonctionne le système de fichier Microdrive. Ici nous allons examiner l'utilisation du système de fichier bande à partir de nos programmes en langage machine.

Indépendamment des données sauvegardées sur bande, il est toujours organisé comme l'indique la figure. Les 19 premiers octets envoyés à la bande sont considérés comme l'en-tête (HEADER), et ces octets contiennent les données relatives au bloc de données suivant sur la bande. Les premier et dernier octets de l'en-tête sont fournis par les routines de système d'exploitation qui écrivent les octets de données sur la bande. Les autres octets doivent être spécifiés avant d'appeler la routine ROM qui écrit les données sur bande. L'octet « type » indique au SE, au chargement, la nature des données dans le bloc. Ensuite vient un nom de fichier sur 10 octets — c'est pourquoi les noms de fichiers sont limités à 10 caractères sur le Spectrum. Même lorsque le nom de fichier a moins de 10 caractères, les espaces restant dans le nom de fichier sont remplis par le code ASCII pour un espace (32). Les données de l'en-tête, à partir de l'octet 11, fournissent d'autres détails au SE, comme l'information concernant l'endroit où sont chargées les



données en mémoire. L'octet final de l'en-tête sert à vérifier des erreurs au moment où la bande est lue — si quelque chose est détecté, le message d'erreur approprié est généré.

Le bloc de données principales commence lui-même par un seul octet contenant la valeur 255 ; il est également fourni par la routine ROM écrire sur bande. Puis viennent les données, suivies par un autre octet de contrôle, également généré par la routine écrire sur bande. Dans les blocs en-têtes et données, tous deux, la valeur réellement écrite comme dernier octet du bloc dépend des octets émis immédiatement avant elle.

Voyons maintenant comment nous pouvons utiliser les routines d'interface cassette à partir de nos programmes en langage machine. Les routines ROM concernées par les opérations de bande sur le Spectrum se trouvent dans les adresses comprises entre &04C2 et &09F3 de la ROM Spectrum. C'est un assez beau morceau de logiciel ! Rien d'étonnant à cela, parce que toutes les opérations de synchronisation et la génération de son pour entrer l'information sur bande sont fondées sur le logiciel dans le Spectrum.

Il n'y a aucune variable système concernée spécifiquement par les opérations de bande, mais le tableau suivant en contient quelques-unes qui servent « incidemment » aux routines.

Variable système	Description
TADDR (aux emplacements 23668 et 23669)	Utilisé par les routines ROM pour décider quelles opérations de bande exécuter en interprétant une commande BASIC
BORDCR (en 23624)	Sert à remettre la couleur de marge à l'origine après une opération de bande
XPTR (en 23647 et 23648)	Sert au stockage temporel du registre IX

Les variables système VARS, ELINE et PROG servent aussi lorsqu'on charge l'information de la bande dans le Spectrum.

Tout d'abord, considérons l'opération de sauvegarde de données sur bande — la routine qui fait cela se trouve à l'adresse ROM &04C2.. Habituellement, cette routine est appelée *deux fois,* d'abord pour sauvegarder l'information d'en-tête et ensuite pour stocker les données proprement dites que nous voulons sauvegarder sur bande.

Registre	En-tête	Bloc de données
A	0	255
DE	17	Longueur de données
IX	Adresse de début d'en-tête ou début de données	

Comme le montre ce tableau, le registre IX est utilisé pour pointer les données à sauvegarder. Pour expliciter cela, considérons un exemple simple de la manière dont on sauvegarde un bloc de données sur bande. Le programme suivant sauvegardera 100 octets de données (à partir de l'adresse ROM 0000) sur bande.

```
        ;routine to save 100 bytes, starting address
         0000, to tape
        ;
        ;send header
3E00            ld   a,0        ;indicate a HEADER BLOCK
DDE5            push ix         ;save ix on stack
DD21D328        ld   ix,header  ;add. of header block
111100          ld   de,17      ;no of header bytes
CDC204          call #04c2      ;write header to tape
        ;send data
DD210000        ld   ix,0000    ;add of 1st byte to save
116400          ld   de,100     ;no of bytes to be saved
3EFF            ld   a,255      ;indicate a DATA BLOCK
CDC204          call #04c2      ;write data to tape
DDE1            pop  ix         ;restore IX value
C9              ret             ;back to BASIC
03      header: defb 3          ;data type 3=CODE block
54455354        defm "TESTPROG" ;f/name+spaces=10 chars
64              defb 100        ;lo-byte of data-length
00              defb 0          ;hi-byte of data-length
00              defb 00         ;lo-byte of start add.
00              defb 00         ;hi-byte of start add.
00              defb 00         ;used for BASIC only
00              defb 00         ;for start line no.
```

L'appel de cette routine sauvegardera la zone adéquate de mémoire. Toutefois, aucun message ne sera affiché (comme ce serait le cas avec des routines de manipulation de bande à partir du BASIC). En fait, cela ne pose pas de problème puisque les données sur bande peuvent être rechargées pour voir si le programme a correctement fait son travail, à l'aide de cette commande :

```
LOAD « TESTPROG » CODE 40000
```

Comparons maintenant les octets chargés aux octets se trouvant entre 0 et 99 en ROM. Rappelez-vous que, bien que le bloc de données d'en-tête commence par l'octet type, le premier octet réellement émis sur la bande est fourni par le SE (0 pour l'en-tête et 255 pour un bloc de données).

Une modification utile de ce programme peut vous permettre de sauvegarder un programme BASIC à partir de l'intérieur d'un listage en langage machine. Le bloc en-tête (HEADER) à la fin de notre précédent listage doit être modifié pour lire :

```
00      header: defb 0          ;0=BASIC program
73737373        defm "sssss     ;fname filled to 10 chars
                sssss"
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) prog+variables
00              defb nn         ;lo-byte) start line
00              defb nn         ;hi-byte) number
00              defb nn         ;lo-byte) length of
00              defb nn         ;hi-byte) program only
```

La longueur du programme et des variables peut être trouvée en soustrayant la valeur contenue dans PROG de la valeur contenue dans ELINE, et la longueur du programme seul peut être obtenue en soustrayant la valeur contenue dans PROG de celle qui est contenue dans VARS. Si vous ne voulez pas que le programme commence à tourner une fois chargé, il suffit de mettre l'entrée « numéro de ligne auquel le programme doit commencer à tourner » dans le listage à 32768. De même, nous pouvons simuler la commande SAVE SCREEN$ à partir du langage machine en spé-

cifiant l'adresse de départ du code à sauvegarder, 8400, et la longueur, &1B00.

La routine ROM qui charge les données à partir de la bande est aussi appelée deux fois : une fois pour l'en-tête (HEADER), et ensuite pour le bloc de données (DATA) proprement dit qui doivent être chargées. La routine est située à l'adresse &0556 en ROM et ses exigences d'entrée sont données dans le tableau suivant (ainsi que les vérifications de code) :

Registre	Charger	Vérifier
Drapeau	1	0
A	0 pour en-tête, 255 pour bloc données	
IX	Pointe l'emplacement en mémoire dans lequel les octets doivent être chargés	
DE	Nombre d'octets à charger; D doit être compris entre 0 et 254	

Nous aurons aussi besoin de mémoire de travail pour cette routine : environ 34 octets ou suffisamment pour accommoder deux blocs en-tête. La raison en est évidente. Afin de charger un fichier nommé à une adresse en RAM que nous spécifions, il faut d'abord placer un second bloc en-tête contenant des détails du fichier que nous voulons charger. Puis nous comparons les données en-tête des fichiers sur bande avec l'en-tête que nous avons placé en mémoire afin de localiser le fichier dont nous avons besoin.

En appelant la routine (en &0556), le drapeau C devrait être mis comme indiqué dans le tableau. Si l'on vérifie une partie de code, celui-ci devra être placé à l'adresse pointée par le registre IX. En quittant la routine ROM « charger à partir de bande », le drapeau C indiquera le résultat de l'opération. S'il est mis à 1, le chargement se fait alors doucement. Mais si, par exemple, vous tentiez de charger un bloc en-tête, et que la première chose rencontrée sur la bande était un bloc de données, alors le drapeau C sera mis à 0. (Toute bande chargeant des erreurs sera manipulée par le SE Spectrum.) Considérons maintenant un exemple simple qui chargera un en-tête de la bande en RAM.

```
        ;load a header from tape into RAM
        ;
37              scf             ;set carry 1=LOAD
3E00            ld   a,0        ;0 indicates a header
DDE5            push ix         ;save ix on stack
DD2148EE        ld   ix,61000   ;load header to 61000
111100          ld   de,17      ;no of bytes in header
CD5605          call #0556      ;do it
DDE1            pop  ix         ;restore ix
C9              ret
```

Une fois que l'en-tête est chargé, nous pouvons l'examiner. Nous pouvons comparer le nom de fichier à celui qu'il nous faut, nous assurer que c'est le type de fichier correct, et vérifier la longueur du bloc de données si nous en avons besoin. Le programme suivant vérifie seulement le nom, et s'il est correct il charge le bloc de données (DATA) suivant l'en-tête (HEADER) en mémoire à une adresse particulière.

```
        ;locate and load TESTPROG
        ;
DDE5            push ix         ;save IX on stack
37      loop:   scf             ;set carry for LOAD
3E00            ld   a,0        ;0=BASIC file
111100          ld   de,17      ;no of bytes in header
DD21CA2B        ld   ix,head2   ;load header into head2
CD5605          call #0556
D27B2B          jp   nc,loop    ;if no header then again
060A            ld   b,10       ;no of bytes in fname
11BA2B          ld   de,head+1  ;point to desired f/name
21CB2B          ld   hl,head2+1 ;point HL to found fname
1A      name:   ld   a,(de)     ;get header char in a
BE              cp   (hl)       ;comp with found header
2007            jr   nz,no      ;different so exit loop
13              inc  de         ;get next char
23              inc  hl
05              dec  b          ;decrement counter
20F7            jr   nz,name    ;same so check next char
1802            jr   ok         ;all chars checked + ok
18DB    no:     jr   loop       ;check failed, try again
        ;found right header so prepare to load data
DD21B92B ok:    ld   ix,head    ;get head in ix
DD6E0D          ld   l,(ix+13)  ;get address to
DD660E          ld   h,(ix+14)  ;load data into
E5              push hl         ;push address onto stack
DDE1            pop  ix         ;and get it into ix
3EFF            ld   a,255      ;indicate load data
        ;next instruction requires user to insert data
         length
11AF2B          ld   de,nn      ;insert data length
37              scf             ;indicate a LOAD
CD5605          call #0556      ;doit
DDE1            pop  ix         ;restore ix
C9              ret
        ;now follow details of desired f/name
03      head:   defb 3          ;indicates CODE block
54455354        defm "TESTPROG" ;f/name
00              defb 0          ;use for data length
00              defb 0          ;data-length hi-byte
48              defb 72         ;lo-byte for load
EE              defb 238        ;address of 61000
00              defb 0          ;used for BAS. prog only
00              defb 0          ; ditto
        ;now follows space for header loaded for
         checking
        head2:  defs 17
```

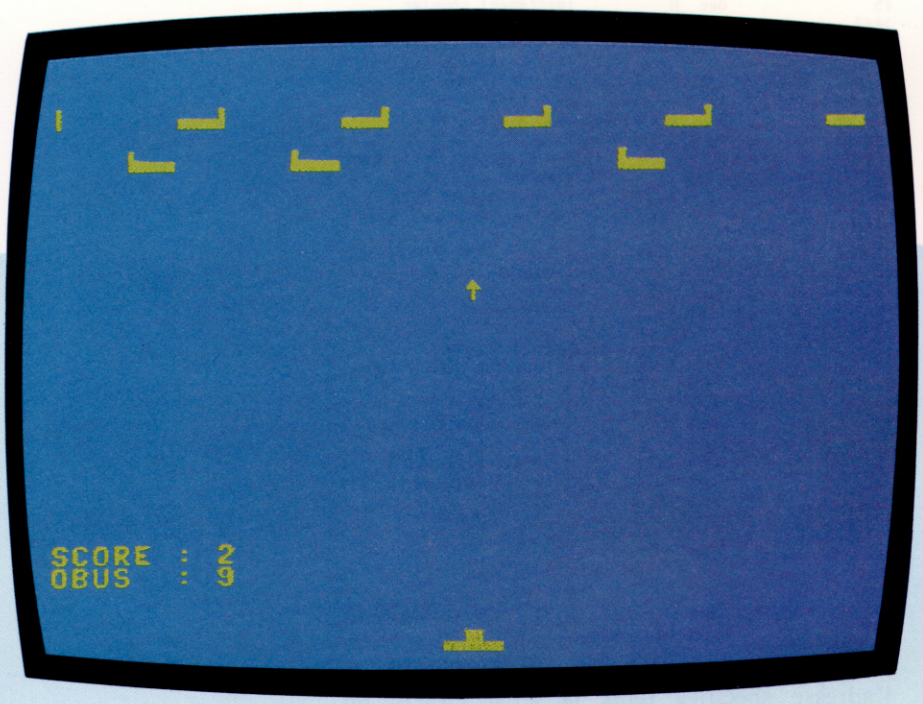
Là encore, aucun message n'est affiché à l'écran pendant cette opération. La routine chargera un bloc de données appelé TESTPROG à l'adresse spécifiée dans la zone en-tête de mémoire. La routine peut servir à charger des fichiers de noms différents, et l'on peut aussi ajouter du code pour vérifier que le type de fichier est correct.

Nous concluons cette discussion des opérations de sauvegarde et de chargement du SE Spectrum par un court programme qui lit les en-têtes de fichiers à partir d'une bande et affiche les détails concernant le fichier — longueur du fichier, adresse de début, etc. — à l'écran. La routine en langage machine est stockée dans l'instruction DATA, et elle charge simplement un bloc en-tête comme indiqué précédemment, puis examine le bloc à partir du BASIC.

```
5 CLEAR 59999
10 FOR I=0 TO 19
20 READ A:POKE (60000+I),A
30 NEXT I
40 RANDOMIZE USR 60000
50 LET type=PEEK 60020
60 LET length=PEEK 60031+256*PEEK 60032
70 LET start=PEEK 60033+256*PEEK 60034
80 LET L$=""
90 FOR I=60021 TO 60030: LET L$=L$+CHR$(PEEK I): NEXT I
100 PRINT "Name: ";L$
110 IF type=0 THEN LET f$="BASIC"
120 IF type=1 THEN LET f$="Number Array"
130 IF type=2 THEN LET f$="String Array"
140 IF type=3 THEN LET f$="CODE"
150 PRINT "Type: ";f$
160 IF type=0 THEN PRINT "Auto Run line number: ";start
170 IF type<>0 THEN PRINT "Start address: ";start
180 PRINT "Length: ";length
190 GOTO 40
200 DATA 221,229,62,0,55,221,33,116,234,17,17,0,205,86,
    5,48,241,221,225,201
```

D.C.A.

Rien de bien difficile dans ce programme écrit par Pierre Monsaut pour le Commodore 64. Mais prudence, n'oubliez pas de l'enregistrer avant de le faire tourner.



Les rôles sont maintenant inversés. Vous manœuvrez la D.C.A. et devez essayer d'abattre les avions qui passent au-dessus de vous. Pour tirer, utilisez n'importe quelle touche. Vous disposez au départ de quinze obus. Si vous abattez dix avions, vous obtenez un bonus de dix points et dix obus supplémentaires.

```
5 REM  xxxxxxxxxx
10 REM x D.C.A. x
15 REM xxxxxxxxxx
20 GOSUB 1000
30 GOTO 720
100 A$=RIGHT$(A$,39)+LEFT$(A$,1)
110 B$=RIGHT$(B$,1)+LEFT$(B$,39)
120 PRINT A$
140 PRINT B$;HH$;
200 GET X$
210 IF X$<>"" AND M=MI THEN M=MI-80:NM=N
M-1:GOTO 230
220 IF M<>MI THEN M=M-80
230 IF M<>M2+80 THEN 250
240 IF PEEK(M2-1)<>32 THEN 500
250 IF M<>M1+80 THEN 270
260 IF PEEK(M1+1)<>32 THEN 600
270 IF M=MI THEN 310
280 POKE M,CM
290 POKE M+N,MC
300 IF M+80<>MI AND M<>M1 THEN POKE M+80
,CR
310 IF M<1064 THEN M=MI:GOTO 720
320 IF NM<1 AND M=MI THEN 4000
330 GOTO 100
500 B$=LEFT$(B$,17)+O1$+RIGHT$(B$,16)
550 GOTO 650
600 A$=LEFT$(A$,17)+O1$+RIGHT$(A$,16)
650 POKE M-80,160
660 POKE (M-80)+N,2
670 POKE M+80,CR
680 S=S+1
700 IF S>1 AND INT(S/10)=S/10 THEN GOSUB
 800
710 M=MI
720 FOR I=1 TO 20
730 PRINT CHR$(17);
740 NEXT I
750 PRINT "SCORE :";S
760 PRINT "OBUS  :";STR$(NM);" "
770 PRINT HH$;
780 GOTO 310
800 A$=A1$:B$=B1$:NM=NM+10
830 FOR I=1 TO 500:NEXT I
850 S=S+10
860 RETURN
1000 A$="":B$="":HH$=CHR$(19)
1030 M1=1044:M2=1124:MI=2004
1060 M=MI:CM=30:MC=5:CR=32:NM=15
1110 O1$="":N=54272:X$="":S=0
1200 FOR I=1 TO 7
1210 O1$=O1$+" "
1220 NEXT I
1230 FOR I=1 TO 40
1240 READ A,B
1250 A$=A$+CHR$(A)
1260 B$=B$+CHR$(B)
1270 IF I/8=INT(I/8) THEN RESTORE
1280 NEXT I
1290 A1$=A$:B1$=B$
1500 PRINT CHR$(147);
2000 FOR I=M-1 TO M+1
2010 POKE I,98:POKE I+N,5
2030 NEXT I
2040 POKE M,160
2050 POKE 53280,0
2060 POKE 53281,6
2070 PRINT CHR$(158);
2080 RETURN
4000 PRINT CHR$(147)
4010 IF S>RE THEN RE=S
4020 FOR I=1 TO 4
4030 PRINT
4040 NEXT I
4050 PRINT TAB(13)"SCORE :";S
4060 FOR I=1 TO 4
4070 PRINT
4080 NEXT I
4090 PRINT TAB(13)"RECORD :";RE
4100 FOR I=1 TO 4
4110 GET X$
4120 PRINT
4130 NEXT I
4140 PRINT TAB(13)"UNE AUTRE ?"
4150 GET X$
4160 IF X$="" THEN 4150
4170 IF X$<>"N" THEN 20
4180 END
10000 DATA 32,182,162,162,162,162,181
10010 DATA 32,32,32,32,32,32,32,32,32
```