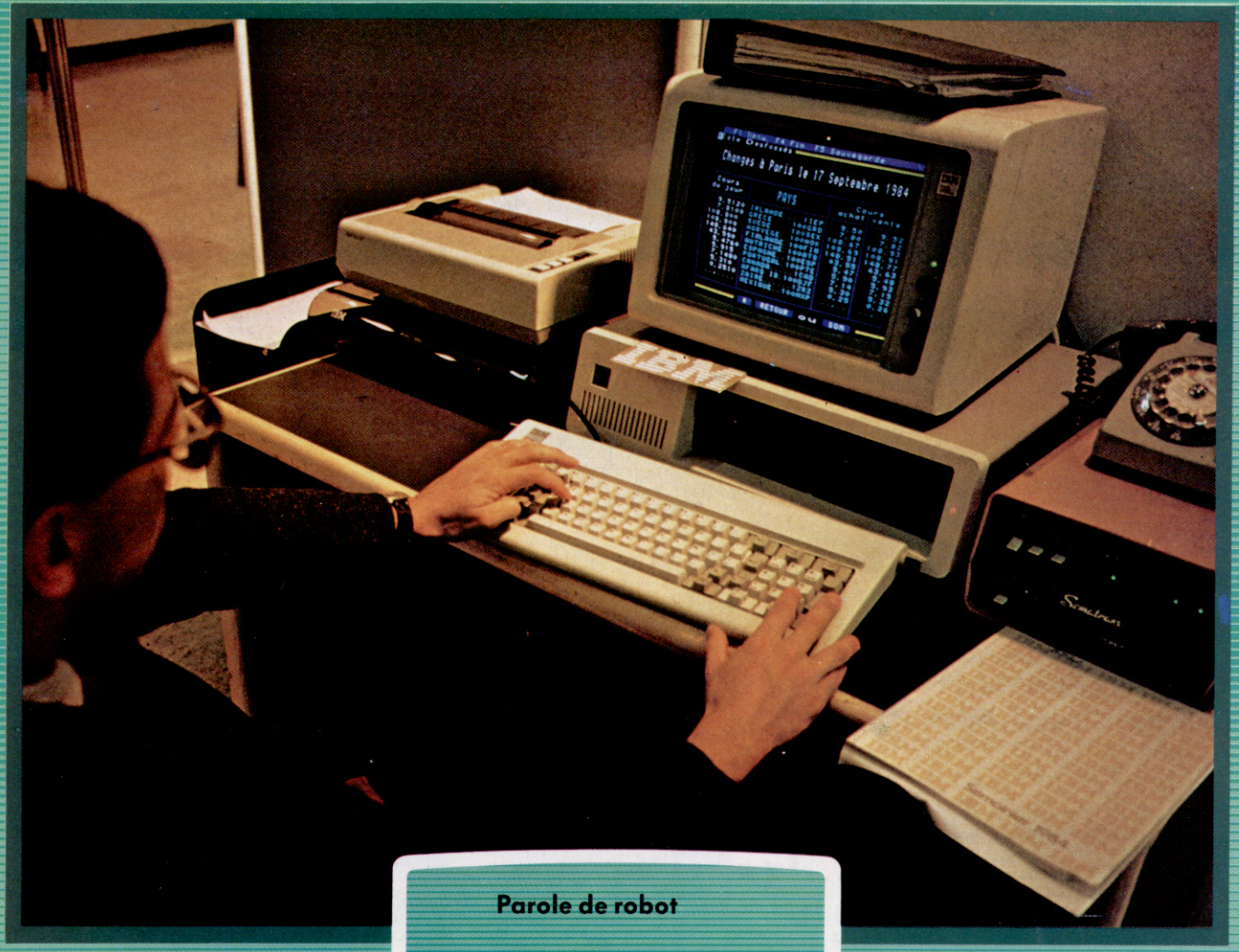
ABC INFORMATIQUE

N°63

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Parole de robot

Tableur Multiplan

Jeu de mots logo

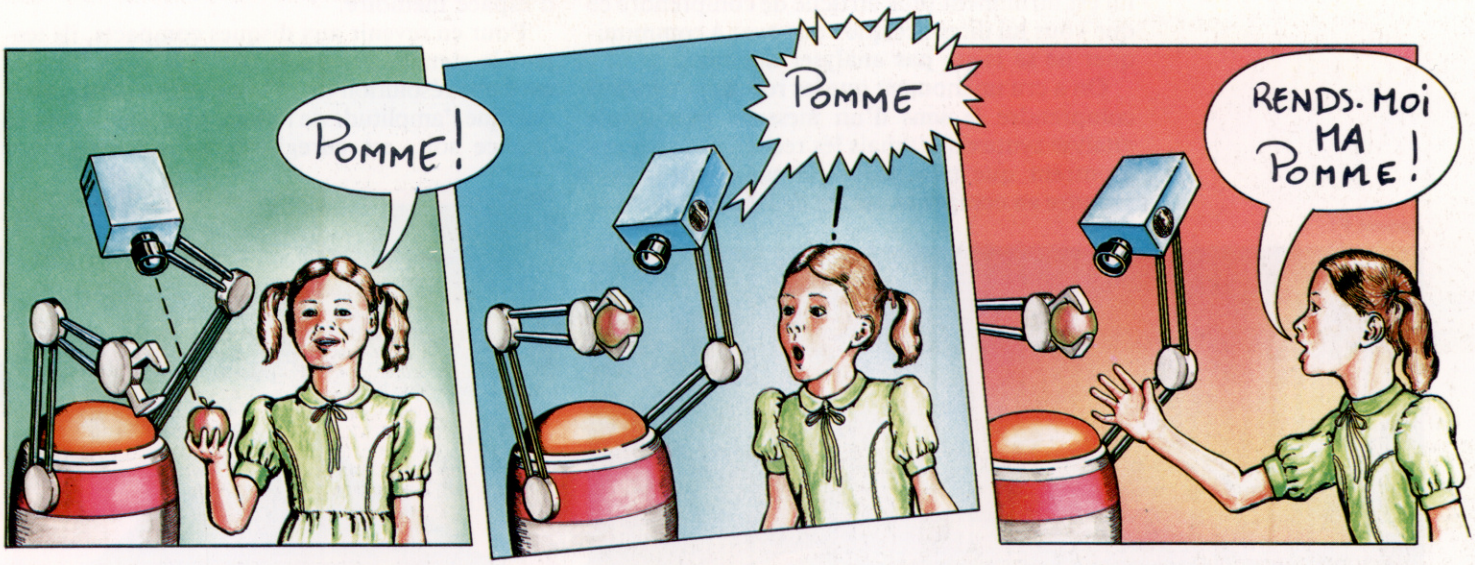
« Déboguons » encore

EDITIONS ATLAS

M 6062-63-12,50 F 90 FB - 3,80 FS - $ 1.95

Parlons franchement

Il est difficile de doter un robot de la parole. Pour comprendre les problèmes liés au langage, il est nécessaire d'examiner les théories qui décrivent l'apprentissage de la parole chez l'homme.



La linguistique peut être divisée en deux grandes écoles : il y a ceux (Chomsky, par exemple) qui pensent que la capacité à produire un langage est innée, et ceux qui estiment au contraire qu'elle est acquise ou apprise.

Les premiers font remarquer que l'homme est le seul animal qui fasse usage du langage, c'est-à-dire des signes vocaux (paroles) et éventuellement des signes graphiques (écriture), pour communiquer. Les autres citent les expériences réalisées sur certains animaux (dauphins ou grands singes) auxquels on a appris à communiquer avec des humains par l'intermédiaire d'un langage gestuel.

Si, pour apprendre un langage, il suffisait de l'entendre, on pourrait envisager cette solution pour un robot : il n'aurait qu'à vous écouter pour retenir votre langage, et les choses en seraient simplifiées d'autant.

Certaines expériences, de portée limitée, visaient à accroître les connaissances grammaticales d'un ordinateur en lui présentant de nouvelles structures de phrases qu'il devait analyser.

D'autres avaient pour but de donner au robot la faculté d'emmagasiner de nouveaux mots et de nouveaux phonèmes (éléments de base sonore du langage articulé), dans quelque langage que ce soit, simplement en les lui faisant entendre. Mais, jusqu'ici, personne n'est parvenu à lui apprendre à parler.

C'est ainsi que, pour des raisons purement pratiques, on a bien dû considérer que les capacités langagières du robot étaient « innées », c'est-à-dire qu'il fallait les lui implanter en mémoire dès le départ, comme s'il était réellement « né » avec. De ce point de vue, il faut distinguer deux niveaux différents : celui de l'analyse syntaxique et celui de l'analyse sémantique.

L'analyse syntaxique étudie la grammaire de ce qui est dit ; elle décode la structure superficielle du langage ou donne au message une forme grammaticale que le robot peut transmettre directement. Pour cela, on a généralement recours à une structure en arbre, qui décompose ou recompose une phrase, à partir des divers éléments du discours. Ce n'est pas là chose facile, mais les résultats obtenus en ce domaine sont déjà appréciables.

L'analyse sémantique est autrement plus complexe et cherche à définir le sens d'un message (ainsi lorsque le robot vous écoute), ou à établir ce qu'il doit transmettre exactement (quand le robot veut vous parler). Ici, la principale difficulté se trouve dans le fait que le langage n'est jamais indépendant du contexte dans lequel il s'inscrit (au contraire, il lui donne son sens) ; et cela ne s'applique pas seulement à la conversation mais aussi au contexte tout entier — qui englobe aussi bien l'environnement dans lequel on parle que la connaissance que chaque interlocuteur possède de l'autre.

C'est d'ailleurs le point de vue adopté par un chercheur en intelligence artificielle américain, Terry Winograd, auteur du programme SHRDLU, qui permettait à un robot de comprendre ce qu'on lui disait et d'obéir à certaines instructions. Toutefois, ce robot (en fait simulé par ordinateur) ne pouvait agir que dans un monde très étroitement défini, réduit à quelques éléments d'un jeu de construction qu'il pouvait manipuler.

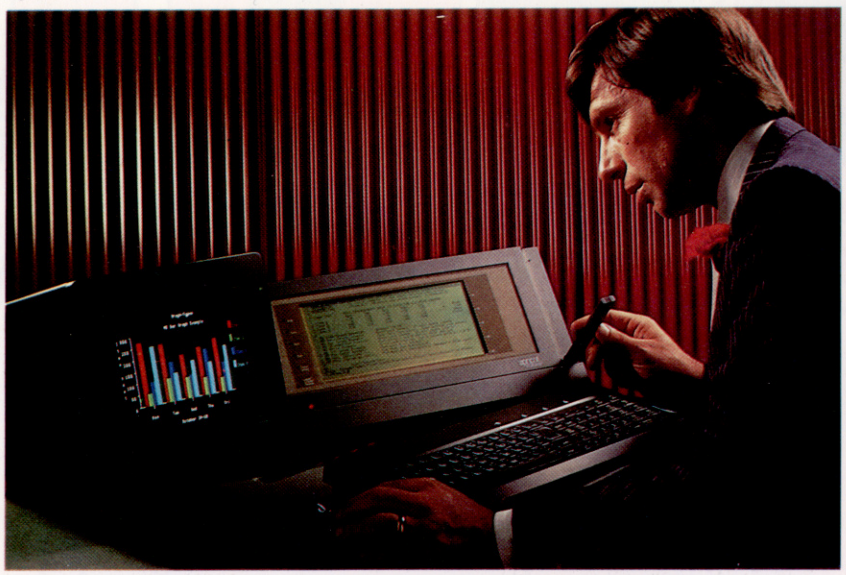
Voir, c'est croire
Quand un être humain voit un objet, une pomme par exemple, et qu'il lui donne un nom, il a une certaine compréhension du mot « pomme ». Le robot peut reconnaître visuellement une pomme s'il en a déjà l'image en mémoire, et il peut aussi répéter le mot, s'il est, lui aussi, enregistré. Mais il ignore tout à fait que c'est un fruit comestible et, plus important encore, qu'il « appartient » à l'être humain considéré — ce dont ce dernier, en revanche, est parfaitement conscient ! (Cl. Steve Cross.)

Le programme avait de bonnes capacités d'analyse sémantique, mais il régissait un univers très simplifié. Face au monde réel, plein de bruits et de fureurs, le robot aurait eu bien plus de mal à comprendre ce qu'on lui disait.

Pour qu'il puisse employer le langage de façon réellement profitable, il faut que le message circule dans les deux sens, entre lui et vous. Pour lui, parler est assez facile — ses connaissances restent très limitées, il a donc assez peu à dire. Il lui est infiniment plus difficile de comprendre ce que vous lui dites : ce que vous avez à communiquer ne se laisse pas analyser aisément.

On a cru un moment que les robots pourraient comprendre le sens d'un message par simple analyse syntaxique; mais les recherches récentes montrent que cela ne suffit pas et qu'il faut prendre en compte la nature de l'environnement et le contexte linguistique dans lequel s'inscrit — oralement bien sûr — le message.

Apricot F1



On a ainsi tenté de procéder de la façon suivante : le robot fait d'abord une analyse syntaxique, puis applique aux résultats obtenus les connaissances qu'il peut avoir du monde extérieur. L'analyse de départ est alors modifiée dans l'espoir de parvenir progressivement à une compréhension correcte de ce qui a été réellement dit.

Aucun des robots disponibles sur le marché n'est encore capable de telles performances. Voyons un peu comment les systèmes actuels parlent et comprennent la parole.

La synthèse de la parole

Le dispositif le plus simple se réduit à un magnétocassette sur lequel un humain a enregistré un message, qui sera diffusé par le robot au moment voulu. C'est, bien entendu, un système assez primitif, mais toutes les méthodes de synthèse de la parole en reprennent le principe de base. Son plus gros handicap est qu'un lecteur de cassette est un appareil mécanique, encombrant, parfois très onéreux et, de plus, sujet aux défaillances techniques. L'étape suivante consiste à convertir le message en signaux numériques, de façon qu'il puisse être enregistré dans la mémoire du robot. On se sert pour cela d'un convertisseur analogique/numérique grâce auquel les ondes sonores de la parole, qui se modifient sans cesse, sont transformées en nombres. Le procédé est d'ailleurs d'usage courant pour enregistrer de la musique (sur les disques compacts, par exemple). Toutefois, ce procédé présente certains inconvénients. En particulier, un signal numérisé occupe énormément d'espace mémoire.

Pour en revenir aux disques compacts, ils testent le signal environ 44 000 fois par seconde, avec une résolution de 16 bits environ (ce qui veut dire que l'amplitude de l'onde, à quelque moment que ce soit, est enregistrée sous forme d'un nombre de 16 bits, ce qui permet de distinguer 2^{16} niveaux différents, soit 65 536 en tout). Avec une telle méthode, une seconde d'enregistrement occuperait près de 88 K de mémoire, ce qui, d'un point de vue pratique, ne peut être envisagé pour un micro-ordinateur.

Big Ears

Currah

Certes, il est bien évident que seuls les systèmes haute-fidélité peuvent apporter une telle précision dans la reproduction du son; on pourrait concevoir un dispositif plus simple, d'une résolution de 8 bits, qui, à raison de 3 000 tests par seconde, n'occuperait plus que 3 K de mémoire!

Reste encore, pour libérer le maximum d'espace, à en économiser le plus possible. Les linguistes se sont rendu compte que le langage parlé peut être décomposé en unités de base, les phonèmes. On s'accorde à penser que la plupart des langues en comportent une quarantaine (par exemple 46 pour l'anglais, mais seulement 22 pour le japonais); il est donc tentant de ne stocker en mémoire que la description acoustique exacte correspondant à chacun d'eux, et de l'utiliser ensuite pour faire parler le robot. Il existe déjà sur le marché des puces spécialisées

La parole et l'ouïe
Il est relativement facile de faire parler les robots et les ordinateurs. Les synthétiseurs de parole (comme celui de Currah, sur la droite) sont déjà largement disponibles. Mais la reconnaissance de la parole pose des problèmes infiniment plus ardus. Les êtres humains ont des prononciations très variées, et pour avoir un vocabulaire supérieur à quelques mots, il faut énormément de mémoire et de puissance de traitement. Des systèmes comme Bigs Ears ou l'Apricot F1 sont bien capables d'identifier certaines injonctions, mais elles sont en nombre trop limité pour pouvoir se révéler vraiment utiles.
(Cl. William Stuart System.)

de ce type, et le seul travail de notre robot consisterait donc à lier entre eux les phonèmes nécessaires, afin d'émettre des messages.

La plupart des synthétiseurs de parole de ce genre peuvent être programmés en rédigeant les phrases à prononcer sous une forme plus ou moins phonétique. « Comment allez-vous? » peut par exemple être noté « Komantalévou? », ce qui est tout à fait suffisant du point de vue du robot, même si les linguistes font usage d'une notation infiniment plus précise.

L'appareil ne répète donc plus un message préenregistré; il crée le sien. Il paraît donc possible de lui faire dire ce que l'on veut, sans avoir à tout stocker dans sa mémoire.

Il n'est pas vraiment nécessaire de le programmer dans le dessein de lui faire manipuler des règles de grammaire très complexes. Répétons-le : il ne pourra jamais dire qu'un nombre assez limité de choses et n'aura besoin que du nécessaire (sauf bien sûr si l'on veut se livrer à des expériences).

Un problème plus important est celui de l'intonation. Si vous avez déjà entendu un synthétiseur de parole, vous avez sans doute remarqué que les paroles prononcées sont à peu près compréhensibles, mais qu'elles sont loin de paraître « naturelles ».

Il y a deux raisons à cela : la première est qu'un phonème n'est pas toujours prononcé de la même façon — cela est fonction des phonèmes qui le précèdent et qui le suivent; la seconde est que le sens même d'une phrase est parfois lié à des phénomènes d'accentuation (ce qu'on appelle en linguistique la prosodie). « Asseyez-vous donc! » peut être dit très aimablement (quand on reçoit un visiteur) ou avec exaspération (par un professeur à des élèves chahuteurs); pourtant le texte écrit reste le même. On a bien essayé de reproduire certaines marques d'intonation, mais cela reste assez difficile dans la mesure où, par définition, un robot ne comprend pas le sens de ce qu'il est amené à dire.

La reconnaissance de la parole

Ce problème est des plus épineux. Un système efficace doit pouvoir reconnaître des messages très variés, émis sous des formes très différentes. On peut, certes, apprendre au robot à identifier certains d'entre eux grâce à des échantillons sonores : lorsqu'il écoute, il pourra chercher parmi les messages qu'il a en mémoire sur tous ses enregistrements magnétiques celui qui se rapproche le plus de ce qu'il vient d'entendre.

On obtient de tels échantillons en lui faisant subir un apprentissage : une phrase ou un mot sont répétés plusieurs fois de suite, de façon à parvenir à une « moyenne » de ce que l'on a cherché a lui faire apprendre. L'un des inconvénients de ce type d'approche est qu'on ne peut ainsi apprendre au robot qu'un nombre réduit de messages, et qu'on est contraint de respecter la prononciation enregistrée! En règle générale, le vocabulaire ne s'accroît qu'aux dépens du nombre de locuteurs identifiables.

Cependant, de tels systèmes peuvent avoir un usage pratique, même s'ils ne comprennent que des ordres très simples comme « en avant », « vers la gauche », et ainsi de suite, ces ordres devant, de plus, être dits pratiquement de la même façon à chaque fois.

On en reste encore au stade de ce qu'on appelle « la reconnaissance de parole par unités discrètes » — discrètes, parce que chaque élément est séparé des autres par une brève pause lorsqu'il est prononcé.

Il est autrement plus complexe de parler au robot de façon continue, donc en employant la méthode de communication verbale habituelle : les mots et les sons se chevauchent, certains sont « avalés » (on ne dit pas « venir », mais « vnir », etc.). Quiconque écoute parler quelqu'un d'autre évalue sans cesse mentalement ce que le locuteur cherche à dire, et décode le message transmis grâce à cette estimation permanente qu'il possède en lui-même.

Mais pour faire de même, un robot devrait en savoir beaucoup plus sur ce qui peut être dit, et sur ce que cela peut bien vouloir dire. On n'en est pas encore là. La tâche paraît encore trop complexe.

La synthèse de la parole est désormais chose banale pour les robots, bien que la qualité des paroles prononcées puisse être encore largement améliorée. La reconnaissance de la parole, en revanche, reste un objectif lointain, et le mieux qu'on puisse faire actuellement est de doter le robot d'une capacité de compréhension à peu près analogue à celle d'un chien bien dressé, pouvant obéir à certains stimuli verbaux, dès lors que ceux-là ne sont pas trop nombreux. Mais les recherches se poursuivent activement dans de nombreux laboratoires dans le monde et les années qui viennent verront sans doute d'importants progrès en ce domaine.

La Voix de son Maître
Le « Voicemate » est un bras mécanique commandé à la voix, et que l'Institut polytechnique de Newcastle a mis au point pour des applications scientifiques et industrielles. (Cl. Newcastle Polytechnic.)

Sur tous les tableaux

Nous nous sommes déjà intéressés aux tableurs. Examinons maintenant Multiplan, un modèle particulièrement sophistiqué destiné au Commodore 64 et dû à Microsoft.

Multiplan comprend de nombreuses commandes évoluées, déjà présentes sur certains logiciels du même type. Il est ainsi possible de travailler sur des groupes de « cases » après leur avoir donné des noms, d'avoir plusieurs écrans — de façon à voir simultanément plusieurs zones du tableur; d'effectuer des recherches rapides à travers les données pour retrouver une information spécifique, d'employer la structure conditionnelle IF...THEN. Multiplan n'était jusqu'à présent disponible que pour l'IBM PC et l'Apple II, mais une version destinée au CBM 64 vient de sortir.

Nous ferons usage de ce logiciel pour établir des statistiques. Nous partirons de données relatives au football américain (exemple choisi à dessein pour son caractère exotique).

Une fois Multiplan chargé en mémoire, le format standard de la feuille de travail est de 255 par 63 cases, toutes définies par des coordonnées horizontales et verticales; ainsi, celle qui se trouve tout en haut à gauche de l'écran est désignée R1C1 (R pour rangée, C pour colonne). Un menu offrant plusieurs options apparaît en bas de l'écran; un curseur clignote au niveau de la première d'entre elles, Alpha. On peut soit appuyer sur la barre d'espace pour déplacer le curseur, soit taper la première lettre de chaque commande.

En choisir une entraîne souvent l'affichage d'un menu auxiliaire qui propose diverses possibilités (formatage des données, gestion de l'espace mémoire, etc.). Vous devrez taper A (pour Alpha) avant de pouvoir entrer du texte. En revanche, les nombres peuvent être enregistrés directement, mais toute formule doit être précédée des signes plus (+) ou égal (=).

Les deux premières rangées de la feuille de travail sont consacrées aux titres. Pour des raisons de commodité, nous avons formaté les cases allant de R1C1 à R2C5 afin qu'elles acceptent un texte continu, ce qui permet de dépasser les limites fixées. Il suffit pour cela de taper :

F(ormat) C(ells) R1C1:R2C5

puis de placer le curseur sur Cont et d'appuyer sur Return. Les deux points indiquent une série de cases. Certaines colonnes sont élargies ou rétrécies de façon à accueillir leurs données.

La feuille de travail comporte deux grandes parties : l'ensemble des informations relatives à une équipe donnée, sur une période de neuf semaines, d'une part; le relevé des parties gagnées ou perdues par toutes les équipes faisant partie de la même « division », d'autre part. Une fois les grandes lignes du modèle définies, toutes les données hebdomadaires devront être entrées « à la main », à l'aide de quelques formules qui permettront de calculer les totaux et les moyennes.

Tableau général des résultats

Performances de chaque équipe

Ian McKinnell

Le premier élément de notre programme est un tableau des résultats, lesquels doivent bien sûr être mis à jour chaque semaine lorsque les rencontres ont eu lieu. Cette manière de procéder offre l'avantage, important, d'utiliser l'une des commandes les plus puissantes de Multiplan, SORT (tri). Nous avons précisé le nom des équipes ainsi que les chiffres correspondants. Chaque équipe est ici placée en fonction de son classement dans sa division, mais rien n'empêche de la classer par rapport à d'autres catégories : le logiciel peut en effet trier plusieurs cases d'une colonne donnée, et ce, par ordre numérique croissant ou décroissant. Tout ce qui est texte est, on s'en doute, trié par ordre alphabétique.

Pour illustrer la manière dont SORT fonctionne, nous classerons chaque équipe par ordre alphabétique. Nous tapons S, et Multiplan affiche :

SORT par colonne:_entre rangées:_et:_ordre:><

Ici, nous voulons effectuer cette opération sur la colonne 1, des rangées 7 à 21, en ordre ascendant (>). Après appui de Return, les noms sont replacés dans l'ordre alphabétique, et les données correspondantes sont elles aussi déplacées. Par exemple, tous les chiffres correspondant à Miami de notre première liste se déplacent vers leur nouvelle position, en changeant simplement la colonne clé de la commande SORT.

Tableau après tri

Maintenant, faisons défiler l'écran par simple appui de la touche curseur vers le bas pour passer à la seconde partie de notre programme, consacrée aux résultats d'ensemble d'une équipe particulière. Nous aurons à utiliser deux formules. La première, SOM, très simple, se borne à faire le total des valeurs hebdomadaires. On passe d'abord à la colonne intitulée TOTAUX (R24C12). Comme nous voulons que la formule s'applique à toutes nos catégories (qui vont de R25C12 à R32C12), nous aurons besoin de pointer chaque case concernée, ce qui se fait à l'aide du curseur.

Pour entrer la formule il suffit de taper :

=SOM(

puis d'appuyer sur une touche qui représente une flèche pointant vers la gauche, jusqu'à ce que le curseur se retrouve sur R25C3. Ensuite, nous tapons deux points pour indiquer que nous allons spécifier un ensemble de cases. Le curseur revient alors automatiquement sur la case où nous entrons la formule; nous appuyons sur la touche « flèche de gauche » (le curseur étant en R25C11), puis sur Return. La formule prend alors la forme :

=SOM (R[-9]C:R[-1]C)

et les totaux des valeurs correspondantes sont affichées. Recopions maintenant la formule dans les cases allant de R26 à R32C12, en maintenant le curseur sur la formule et en recourant à la commande COPY :

C(opy) d(own) 7 rangées

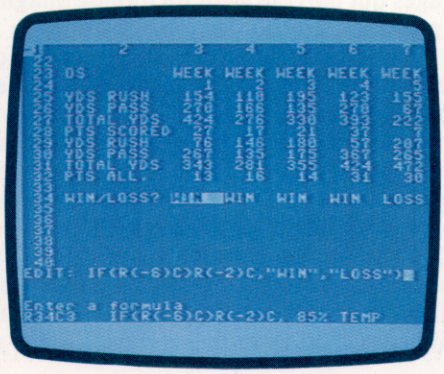
Le même procédé sert à calculer les autres valeurs dont on a besoin. Elle est placée dans la cellule R27C3 pour les gains d'espace et en R31C3 pour ceux concédés. SOM est aussi copiée dans les huit colonnes de droite, pour couvrir l'intégrité de la période de neuf semaines.

La seconde formule fait usage de l'instruction IF; elle est un peu plus complexe, mais très utile. Dans notre modèle, nous demanderons à Multiplan de déterminer si une partie a été gagnée ou perdue, en comparant les totaux de deux catégories : Points Marqués par notre équipe et Points Concédés à l'adversaire. L'instruction est : LE Points Marqués > Points Concédés THEN PRINT «VICTOIRE» ELSE PRINT «DÉFAITE».

Là encore, nous aurons besoin de références relatives, et c'est pourquoi nous nous servirons du curseur pour pointer les emplacements contenant les deux valeurs. Plaçons-le en R34C3 (intitulé «VIC/DÉF»), et entrons la formule :

IF(R[-6]C>R[-2]C,«VIC»,«DÉF»)

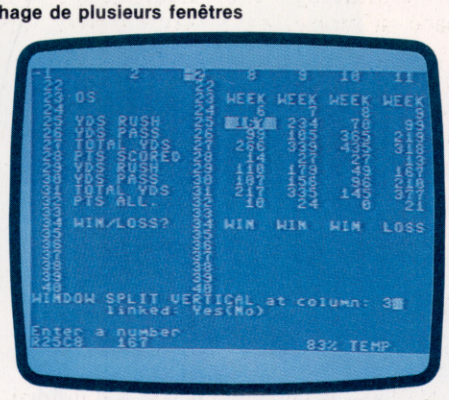
Instruction conditionnelle (IF...THEN)



Notez bien que le texte employé dans la formule doit être placé entre guillemets, et que les conditions doivent être impérativement entourées de parenthèses. Copions maintenant la formule le long de la rangée, comme auparavant. Le modèle que nous avons mis au point renferme des données couvrant neuf semaines. Étant donné les dimensions de l'écran, nous ne pouvons voir les libellés placés du côté gauche, ni les totaux sur la droite. Nous pouvons diviser l'écran en deux fenêtres, qui défilent ensemble ou séparément.

Nous voulons « fendre » l'écran verticalement à partir de la colonne 3, aussi taperons-nous

Affichage de plusieurs fenêtres



W(indow) S(plit) V(ertical), ce qui fera apparaître à l'écran le message WINDOW SPLIT VERTICAL at column:_ linked YES/NO. Si les fenêtres ne sont pas « reliées » *(linked),* elles défileront indépendamment l'une de l'autre. Ici, nous préciserons le numéro de la colonne (3), répondrons NO à la question et appuierons sur Return. Les libellés seront désormais bien visibles, quelle que soit la portion de la feuille affichée.

Le football américain
En dépit de son nom, ce sport ressemble moins à notre football (appelé *soccer* aux États-Unis) qu'au rugby : la balle et les façons de marquer sont en gros les mêmes. Chaque équipe, toutefois, ne comporte que onze joueurs. Mais, peut-être ne pouvez-vous pas regarder la chaîne de télévision Canal Plus qui donne les explications de ce sport.
Sachez que chacune des équipes a droit au cours d'un match à quatre tentatives (appelées *downs*) pour faire avancer la balle de 10 yards (un peu plus de 9 m) en direction des buts adverses. La distance parcourue à chaque fois intervient dans le calcul, assez complexe, des scores. L'American Football League organise les rencontres, et répartit les équipes en deux « divisions », l'American Conference et la National Conference. La saison commence en septembre et comporte seize matches. Le point culminant se situe en janvier, lorsque le Super Bowl oppose la meilleure équipe de chaque division. Signalons que Mind Games vient de lancer un jeu nommé American Football destiné au Spectrum 48 K et au Commodore 64.
(Cl. Kevin Jones.)

Partir à l'aventure

Les jeux d'aventures sont très appréciés des possesseurs d'ordinateurs. Mais, s'il est passionnant d'y jouer, il est plus intéressant d'en écrire soi-même. Voici comment faire.

Les jeux d'aventures sont nés au début des années soixante-dix, avec les jeux de type « Donjon et Dragon ». Les joueurs y incarnaient divers personnages qui parcouraient un monde imaginaire créé par le maître du donjon. Cet univers était pour l'essentiel un labyrinthe de pièces très complexe, où l'on trouvait des objets et où il fallait affronter divers dangers.

En règle générale, il convenait avant tout de sortir de là, en sauvant une belle personne, ou en amassant des trésors. Les programmeurs de gros systèmes furent les premiers à adapter ce jeu sur ordinateur. Ce n'était encore qu'un passe-temps, d'ailleurs très apprécié des professionnels pendant et en dehors de leur travail! Mais la formule avait un avantage certain : il n'y avait plus besoin de réunir plusieurs personnes, et on pouvait jouer quand on voulait. Donjon et Dragon a, depuis, encore gagné en ampleur et en complexité. Nous avons déjà parlé du système MUD qui est un des meilleurs exemples du niveau de complexité atteint par ce type de jeu.

Certains jeux d'aventures sont purement rédactionnels, d'autres sont agrémentés d'illustrations. Les puristes font remarquer que celles-là font perdre de l'espace mémoire qui serait mieux employé à rendre encore plus touffue la structure de l'intrigue.

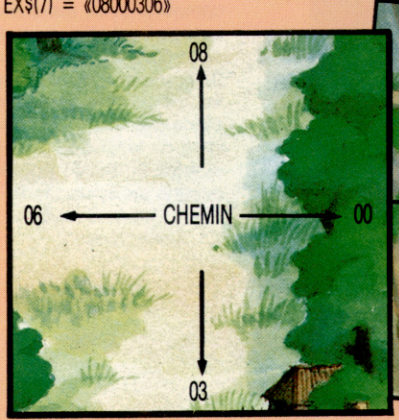
Et, ajoutent-ils, les images seront toujours plus pauvres que l'imagination. Sans vouloir prendre parti, il est bien évident que la popularité actuelle du genre doit beaucoup à l'attrait visuel du graphisme. Si certains jeux sont assez limités de ce

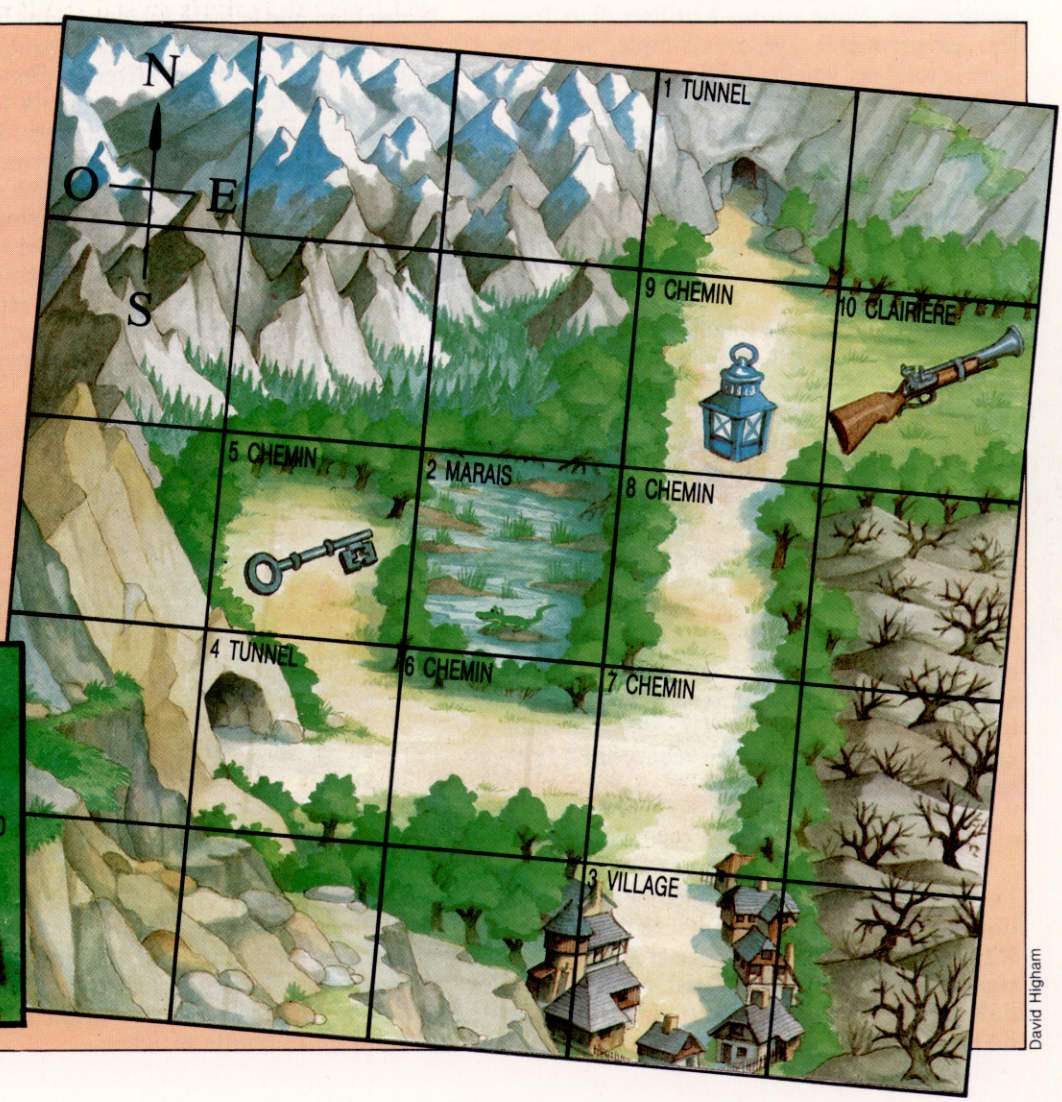
Mise en carte

Quand on rédige un jeu d'aventures, la première chose à faire est de tracer une carte représentant les différents lieux que le joueur pourra visiter. Chacun d'eux comporte une brève description qui précise l'importance éventuelle que prend le lieu pour la résolution de l'énigme, et indique s'il contient ou non un ou plusieurs objets. La numérotation des emplacements permet de les intégrer sans grande difficulté au programme lui-même.

Déplacements possibles à partir de l'emplacement 7

EX$(7) = «08000306»





David Higham

point de vue, d'autres ont fait des efforts considérables en ce sens.

Nous nous intéressons ici aux techniques nécessaires à la création d'un jeu d'aventures. L'un d'eux, dont le titre est « Digitaya », vous sera fourni par fragments jusqu'à ce qu'il soit complet. Vous y jouerez le rôle d'un agent secret et vous descendrez dans un micro-ordinateur pour y retrouver une mystérieuse « personne » nommée Digitaya, perdue quelque part dans les entrailles de la machine.

Les difficultés et les périls ne manqueront pas, et vous aurez besoin de toutes vos connaissances en informatique pour vous en tirer. Dans la mesure du possible, le programme sera rédigé en BASIC « standard »; les « variantes » individuelles seront données si cela est nécessaire. Le seul problème sera en fait celui de l'espace mémoire dont vous disposez sur votre ordinateur.

Nous entendons présenter en détail les techniques de programmation requises et, chemin faisant, il nous sera certainement difficile de ne pas vous donner trop d'indices sur le jeu, ce qui plus tard risque de vous gâcher une partie du plaisir de jouer. C'est pourquoi un second jeu nommé « la Forêt hantée » nous servira de base de travail : il est bien plus simple, et permettra d'exposer les procédés et les algorithmes mis en œuvre dans Digitaya.

Faire une carte

Quand on écrit un jeu d'aventures, il faut avant tout dresser la carte du monde imaginaire où il se déroule. Nous y signalerons tous les endroits qui le composent, l'emplacement de départ des objets à trouver, et nous signalerons aussi certains lieux « spéciaux ». La plupart du temps, le joueur se borne à passer d'un point à un autre, en ramassant ou en abandonnant des objets. Mais il y a des endroits qui peuvent receler des dangers (un marais, la tanière d'un dragon), dans lesquels on ne peut entrer (ou dont on ne peut sortir !) qu'après avoir accompli telle ou telle action.

La meilleure façon de commencer consiste à établir le nombre de lieux différents. La Forêt hantée, par exemple, en compte dix, situés sur une grille de cinq carrés de côté (voir l'illustration). Digitaya en a près de soixante, répartis sur une grille quatre fois plus grande.

Au départ, la grille n'est pas numérotée, et il convient d'abord d'y placer les lieux. Dans la Forêt hantée se trouvent un chemin, deux tunnels, un marais, un village et une clairière. La position des objets est précisée en bas des carrés qui les contiennent. Les endroits « spéciaux » sont signalés par des astérisques et seront traités à part des autres positions.

Une fois le plan général bien défini, il faut numéroter les carrés, en vous occupant d'abord des lieux « spéciaux ». C'est la seule considération spéciale qu'il convient de prendre en compte dès le départ. Le reste n'a qu'une importance relative dans le déroulement des opérations; mais une fois que vous aurez attribué vos nombres, n'en changez plus par la suite.

La programmation des données

Les informations de la carte doivent ensuite être converties en données utilisables par le programme. Il y a bien des manières de procéder, mais, ici, nous ferons usage d'un tableau à deux dimensions. Le premier d'entre eux, LN$(), contient la description de chaque emplacement. C'est ainsi que l'emplacement n° 7, LN$(7), a pour contenu « Sur un chemin ». Chaque utilisation d'une donnée spécifique sera précédée de la mention « Vous êtes ».

Le second tableau, EX$(), abrite des informations sur tous les déplacements possibles à partir d'un emplacement donné. Nos deux jeux se limiteront à quatre directions : nord, est, sud et ouest. EX$(0) contient les données de chacun des lieux auxquels on peut accéder suivant chacune des quatre directions. Les données sont stockées en mémoire sous forme d'une chaîne de huit chiffres, à raison de deux chiffres par direction (l'ordre suivi étant celui de NESO - nord, est, sud, ouest).

Le lieu n° 7 a, par exemple, des sorties vers le nord, le sud et l'ouest, mais aucune vers l'est. Les deux premiers chiffres de EX$(7) seront donc 08 (et non simplement 8), ce qui indique que le lieu n° 8 est au nord. Vient ensuite 00 : il n'y a pas de sortie vers l'est. 03 et 06 précisent quels sont les endroits situés au sud et à l'ouest de 7. On peut de cette façon classer jusqu'à trente-neuf endroits; il est toujours possible d'en faire intervenir davantage, mais il faudrait alors employer des données pour EX$() sous la forme de groupes de trois chiffres.

C'est un autre tableau — IV$() —, qui gère les trois objets se déplaçant dans la Forêt hantée : il est à deux dimensions et prend note de la position de chaque objet après leur changement de lieu; ils sont aussi très brièvement décrits, et leur position originelle est précisée.

Le « fusil » correspond ainsi à IV$(C,1); IV$(C,2) indique l'endroit où il se trouve lorsque le jeu commence. S'il est emmené ailleurs, ce changement sera dûment enregistré.

Éviter les erreurs

Dans nos deux listages, un « total de contrôle » *(checksum)* est placé après les données se rapportant à la carte du jeu. Cela permet de vous assurer qu'elles ont été tapées sans erreurs. En effet, la valeur de ce total est égal à la somme des données; si les deux nombres diffèrent, c'est qu'une erreur a été commise, et le programme interrompra alors son exécution.

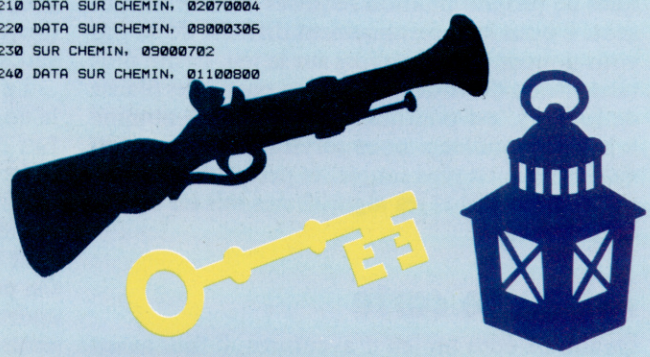
On notera que Digitaya compte deux totaux de contrôle différents. Cela s'explique parce que l'addition de toutes les données aurait donné un chiffre trop grand pour tenir dans un seul total de contrôle, et nous avons préféré regrouper séparément l'ensemble des quatre nombres de gauche, et celui des quatre nombres de droite. Nous verrons prochainement la manipulation et l'affichage des données présentées ici qui forment la carte du jeu.

Digitaya

```
6090 REM ***** LECTURE DONNEES *****
6100 REM ***** LECTURE INVENTAIRE *****
6110 DIM IV$(8,2), IC$(4)
6120 FOR C=1 TO 8
6130 READ IV$(C,1), IV$(C,2)
6140 NEXT C
6150 :
6160 REM ***** LECTURE DATA LIEUX ET SORTIES
6170 DIM LN$(55), EX$(55)
6180 C1=0 : C2=0 : REM INITIALISATION DES
DEUX TOTAUX DE CONTROLE
6190 FOR C=1 TO 54
6200 READ LN$(C), EX$(C)
6210 C1=C1+VAL (LEFT$(EX$(C),4))
6220 C2=C2+VAL (RIGHT$(EX$(C),4))
6230 NEXT C
6240 READ CA : IF CA<>C1 THEN PRINT "TOTAL
DE CONTROLE ERRONE" : STOP
6250 READ CB : IF CB<>C2 THEN PRINT" TOTAL
DE CONTROLE ERRONE" : STOP
6260 RETURN
6270 REM ***** DATA INVENTAIRE *****
6280 DATA UN NOMBRE D'ADRESSE MEMOIRE, 45,
UNE CLE, 34, UN BOUCLIER LASER, 25
6290 DATA UN TICKET POUR LES TROIS DIMENSIONS, 26,
UNE CARTE DE CREDIT DATA, 28
6300 DATA DIGITAYA, 30, UN LIVRE DE CODE,
19, UN DISPOSITIF D'ACTIVATION DU TAMPON, 13
6310 :
6320 REM ****** DATA LIEUX ET SORTIES *****
6330 DATA DANS LA SORTIE TELE, 00000000
6340 DATA DANS LE PORT UTILISATEUR, 00090100
6350 DATA DANS LE PORT CASSETTE, 00110000
6360 DATA DANS LE PORT MANCHE A BALAI, 00130000
6370 DATA DANS UN DISPOSITIF TROIS DIMENSIONS, 00170000
6380 DATA DANS L'UNITE ARITHMETIQUE ET LOGIQUE,
00310016
6390 DATA A L'ENTREE DE LA MEMOIRE, 00490000
6400 DATA SUR L'AUTOROUTE E/S, 09000001
6410 DATA SUR L'AUTOROUTE E/S, 10000802
6420 DATA SUR L'AUTOROUTE E/S, 11000900
6430 DATA SUR L'AUTOROUTE E/S, 12001003
6440 DATA SUR L'AUTOROUTE E/S, 13531100
6450 DATA SUR L'AUTOROUTE E/S, 14001204
6460 DATA SUR L'AUTOROUTE E/S, 15001300
6470 SUR L'AUTOROUTE E/S. UN PANNEAU ANNONCE
'S U D', 00001400
6480 DATA DANS LE REGISTRE DES DONNEES, 00061700
6490 DATA SUR UNE AUTOROUTE A 8 VOIES, 16001805
6500 DATA SUR UNE AUTOROUTE A 8 VOIES, 17001900
6510 DATA SUR UNE AUTOROUTE A 8 VOIES, 18002000
6520 DATA SUR UNE AUTOROUTE A 8 VOIES, 19292100
6530 DATA SUR UNE AUTOROUTE A 8 VOIES, 20282200
6540 DATA SUR UNE AUTOROUTE A 8 VOIES, 21272300
6550 DATA SUR UNE AUTOROUTE A 8 VOIES, 22262400
6560 DATA SUR UNE AUTOROUTE A 8 VOIES, 23250000
6570 DATA DANS LA MATRICE DE CARACTERES, 26360024
6580 DATA EN MEMOIRE HAUTE, 27352523
6590 DATA EN MILIEU DE MEMOIRE, 28342622
6600 DATA EN MILIEU DE MEMOIRE, 29332721
6610 DATA EN MEMOIRE BASSE, 00542820
6620 DATA DANS LA TANIERE DE L'ACCUMULATEUR,
00000600
6630 DATA DANS UN LONG CORRIDOR, 00420005
6640 DATA DANS UN REGISTRE D'INDEX, 31000000
6650 DATA EN MEMOIRE BASSE, 54403428
6660 DATA EN MILIEU DE MEMOIRE, 33393527
6670 DATA EN MEMOIRE HAUTE, 34383626
6680 DATA DANS LA MATRICE DE CARACTERES,
35370025
6690 DATA DANS UNE TABLE DE VECTEURS ALEATOIRES,
00000000
6700 DATA EN MEMOIRE HAUTE, SURPLOMBANT UNE
AUTOROUTE, 39003735
6710 DATA EN MILIEU DE MEMOIRE, 40003834
6720 DATA EN MEMOIRE - A L'EST IL Y A UNE
PORTE, 41003933
6730 DATA EN MEMOIRE BASSE, 00004054
6740 DATA DANS UN CORRIDOR, 00430031
6750 DATA DANS UN CORRIDOR, 00440042
6760 DATA DANS UN CORRIDOR, 00004543
6770 DATA DANS LE REGISTRE ADRESSES, 00004600
6780 DATA SUR UNE AUTOROUTE A 16 VOIES, 45004700
6790 DATA SUR UNE AUTOROUTE A 16 VOIES, 46004800
6800 DATA SUR UNE AUTOROUTE A 16 VOIES, 47004900
6810 DATA SUR UNE AUTOROUTE A 16 VOIES -
IL Y A UNE GRANDE PORTE A L'OUEST, 48005007
6820 DATA SUR UNE AUTOROUTE A 16 VOIES, 49005100
6830 DATA SUR UNE AUTOROUTE A 16 VOIES, 50005200
6840 DATA SUR UNE AUTOROUTE A 16 VOIES, 51000000
6850 DATA DANS UN VECTEUR MENANT A LA MEMOIRE,
00290012
6860 DATA EN MEMOIRE BASSE, 00413329
6870 REM ***** TOTAUX DE CONTROLE *****
6880 DATA 100169, 103973
```

La Forêt hantée

```
6000 REM **** LECTURE DATA CARTE ET OBJETS ****
6010 DIM IV$(3,2), LN$(10), EX$(10), IC$(2)
6020 FOR C=1 TO 3
6030 READ IV$(C,1), IV$(C,2)
6040 NEXT C
6050 :
6060 FOR C=1 TO 10
6065 READ LN$(C), EX$(C)
6070 CC=CC+VAL (EX$(C)) : REM TOTAL DE CONTROLE
6080 NEXT C
6090 :
6100 READ CD : IF CD<>CC THEN PRINT "ERREUR
DE DATA, TOTAL DE CONTROLE INEXACT": STOP
6110 :
6120 REM ***** DATA OBJETS *****
6130 DATA FUSIL, 10, LAMPE, 9,
CLE, 5
6140 :
6150 REM ***** DATA CARTE LIEUX
6160 DATA PRES ENTREE TUNNEL, 00000900
6170 DATA DANS MARAIS, 00000000
6180 DATA DANS VILLAGE, 07000000
6190 DATA PRES ENTREE TUNNEL, 05060000
6200 DATA SUR CHEMIN, 00020400
6210 DATA SUR CHEMIN, 02070004
6220 DATA SUR CHEMIN, 08000306
6230 SUR CHEMIN, 09000702
6240 DATA SUR CHEMIN, 01100800
```



Variantes de basic

Nos programmes sont écrits pour le Commodore 64.
Pour le Spectrum, il convient de procéder aux modifications suivantes :

Spectrum

Pour Digitaya, voici les changements à effectuer :

```
6110 DIM V$(8,2,24):DIM I$(4,24)
6170 DIM L$(55,52):DIM E$(55,8)
6200 READ L$(C),E$(C)
6210 LET C1=C1+VAL(E$(C)(TO 4))
6220 LET C2=C2+VAL(E$(C)(LEN(E$(C))-3 TO))
```

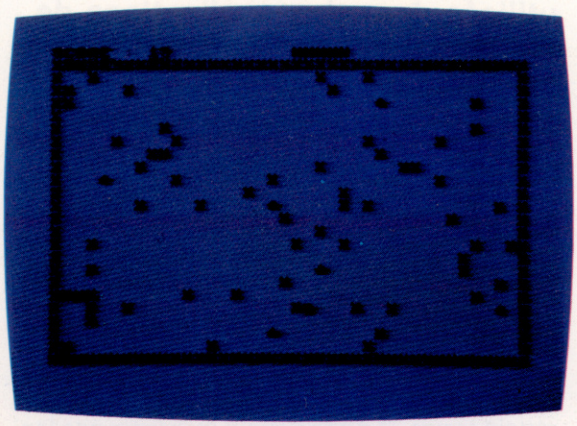
Par ailleurs, tous les DATA doivent être mis entre guillemets, à l'exception de la ligne 6880 (total de contrôle). Pour la Forêt hantée, modifiez les lignes suivantes :

```
6010 DIM V$(3,2,5):DIM L$(10,22)
6015 DIM E$(10,8):DIM I$(2,5)
6030 READ V$(C,1) V$(C,2)
6065 READ L$(C),E$(C)
6067 LET CC=0
6070 LET CC=CC+VAL(E$(C))
```

Là encore, toutes les valeurs et expressions contenues dans les DATA doivent être mises entre guillemets, à l'exception de celle de la ligne 6270 qui correspond au total de contrôle.

Robots sur C 64

Vous voici seul, abandonné sur une planète défendue par des robots meurtriers. Le sol est truffé de mines. C'est « Robots » écrit par Pierre Monsaut pour le Commodore 64.



Les mines sont représentées sur l'écran par des « X ». Au début du jeu, cinq robots (représentés par des losanges) sont présents sur le terrain. Sans perdre une seconde, ils se précipitent sur vous en suivant toujours le plus court chemin. Par chance, les robots sont aveugles et ne voient pas les mines qui se trouvent entre eux et vous, ce qui vous permet, en vous déplaçant judicieusement, de les éliminer. Utilisez pour cela les touches :

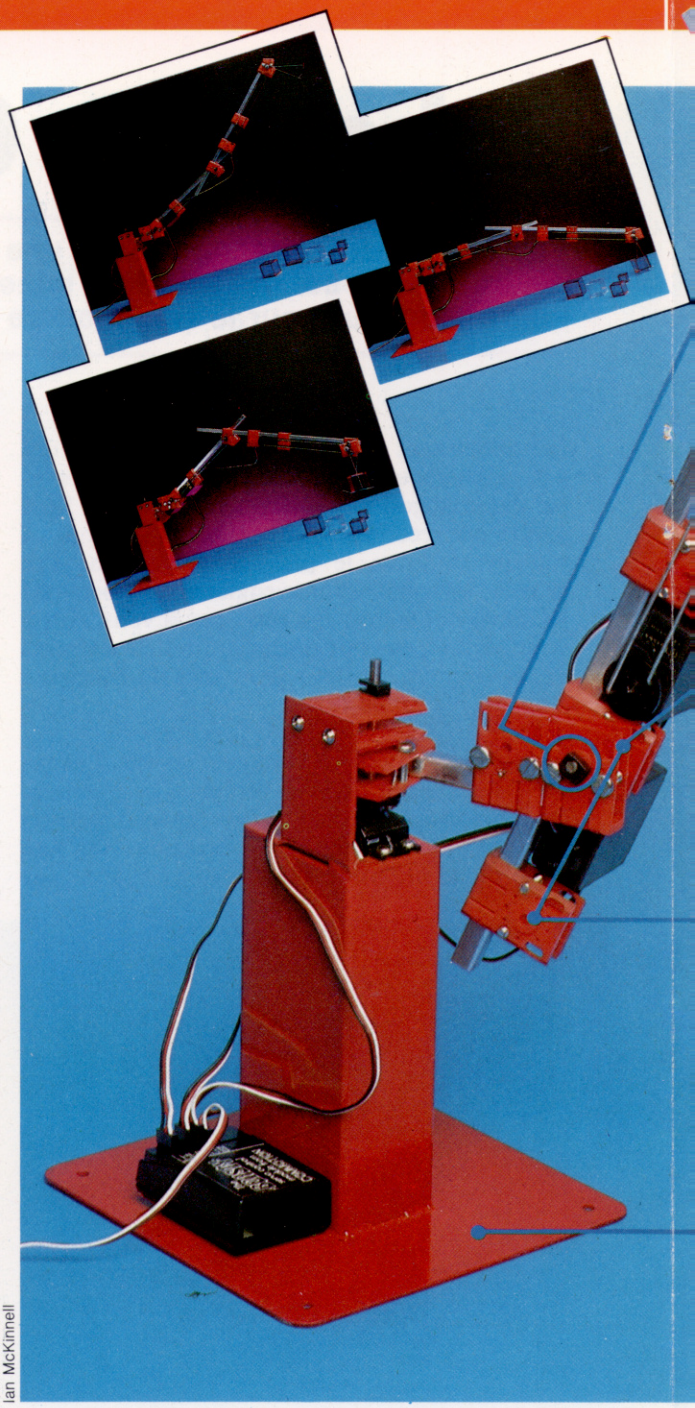
Q, W, E, A, S, D, Z, X, C

suivant la direction que vous avez choisie. La touche S vous servira à vous arrêter. Lorsque tous les robots sont éliminés, le jeu reprend avec un robot supplémentaire. Si vous sautez sur une mine ou si un robot vous tue, tout n'est pas perdu. Vous disposez en effet de cinq vies. Si vous désirez changer le nombre de mines, modifiez la valeur de la variable NM à la ligne 1270.

```
5 REM  ××××××××××
10 REM × ROBOTS ×
15 REM ××××××××××
20 DIM R(30)
25 NH=5
30 GOSUB 1000
100 GET X$
110 GOSUB 900
120 J=J+D
130 C=PEEK(J)
140 IF C<>CN AND C<>CJ THEN 600
150 POKE J1,CN
160 POKE J,CJ
170 POKE J+M,JC
180 J1=J
200 JY=INT((J-1024)/40)
210 JX=(J-1024)-JY*40
220 T=0
230 FOR I=1 TO NR
240 IF R(I)=0 THEN 380
250 T=1
260 RY=INT((R(I)-1024)/40)
270 RX=(R(I)-1024)-RY*40
280 R1=RY+SGN(JY-RY)
290 R2=RX+SGN(JX-RX)
300 RR=40*R1+R2+1024
310 C=PEEK(RR)
320 IF C=CR OR C=CM THEN S=S+1:POKE R(I)
,CN:R(I)=0:GOTO 380
330 IF C=CJ THEN 600
340 POKE R(I),CN
350 POKE RR,CR
360 POKE RR+M,RC
370 R(I)=RR
380 NEXT I
390 IF T=0 THEN 500
400 FOR I=1 TO 500
410 NEXT I
420 GOTO 100
500 S=S+10
510 IF NR<30 THEN NR=NR+1
520 GOSUB 1030
530 GOTO 100
600 NH=NH-1
610 POKE J,CN
620 FOR I=1 TO 4
630 POKE 53281,0
640 FOR K=1 TO 50
650 NEXT K
660 POKE 53281,5
670 FOR K=1 TO 50
680 NEXT K
690 NEXT I
700 IF NH>0 THEN NZ=N1:GOTO 30
710 IF S>RE THEN RE=S
720 PRINT CHR$(147)
730 FOR I=1 TO 4
740 PRINT
750 NEXT I
760 PRINT TAB(13)"SCORE :";
775 PRINT S
780 FOR I=1 TO 4
785 PRINT
790 NEXT I
795 PRINT TAB(13)"RECORD :";
800 PRINT RE;
805 FOR I=1 TO 4
810 GET X$
815 PRINT
820 NEXT I
825 PRINT TAB(13)"UNE AUTRE ?"
830 GET X$
835 IF X$="" THEN 830
840 IF X$<>"N" THEN 25
845 END
900 IF X$="Q" THEN D=-41
910 IF X$="W" THEN D=-40
920 IF X$="E" THEN D=-39
930 IF X$="A" THEN D=-1
940 IF X$="S" THEN D=0
950 IF X$="D" THEN D=1
960 IF X$="Z" THEN D=39
970 IF X$="X" THEN D=40
980 IF X$="C" THEN D=41
990 RETURN
1000 CN=32
1010 N1=5
1020 NR=N1
1030 GOSUB 2360
1040 D=0
1050 CM=24
1060 CR=90
1070 CJ=81
1080 J=1024
1090 M=54272
1100 JC=0
1200 RC=0
1210 RY=0
1220 RX=0
1230 R1=0
1240 R2=0
1250 RR=0
1260 MC=0
1270 NM=60
1980 POKE 53280,6
1990 POKE 53281,6
2010 FOR I=0 TO 39
2020 POKE 1064+I,CM
2030 POKE 1064+M+I,MC
2040 POKE 1984+I,CM
2050 POKE 1984+M+I,MC
2060 NEXT I
2070 FOR I=1 TO 22
2080 POKE 1064+I*40,CM
2090 POKE 1064+M+I*40,MC
2100 POKE 1103+I*40,CM
2110 POKE 1103+M+I*40,MC
2120 NEXT I
2130 FOR I=1 TO NM
2140 GOSUB 3000
2150 POKE P,CM
2160 POKE P+M,MC
2170 NEXT I
2180 FOR I=1 TO NR
2190 GOSUB 3000
2200 R(I)=P
2210 POKE P,CR
2220 POKE P+M,RC
2230 NEXT I
2240 GOSUB 3000
2250 J=P
2260 POKE J,CJ
2270 POKE J+M,JC
2280 J1=J
2290 FOR I=1 TO 5
2300 POKE J,CJ+128
2310 FOR K=1 TO 100
2320 NEXT K
2330 POKE J,CJ
2340 FOR K=1 TO 100
2345 NEXT K
2350 NEXT I
2355 RETURN
2360 PRINT CHR$(144);
2370 PRINT CHR$(147);
2380 PRINT "SCORE :";S,
2390 FOR I=1 TO NH
2400 PRINT "H";
2410 NEXT I
2420 D=0
2430 GET X$
2440 RETURN
3000 P=INT(RND(TI)*960)+1064
3010 IF PEEK(P)<>32 THEN 3000
3020 RETURN
```

Beasty

La robotique est devenue un élément essentiel de l'industrie informatique. Mais, jusqu'à présent, les robots ont surtout été utilisés dans des applications de l'industrie lourde. Avec Beasty, le robot se domestique.



Ian McKinnell

Beasty est vendu en kit; il peut être assemblé à l'aide d'une paire de tournevis. Un logiciel sur cassette accompagne aussi le kit; il s'agit du système d'exploitation Robol qui sert à commander le bras robot.

Deux manuels sont fournis. Le débutant peut être intimidé par le nombre de pièces et par la complexité des instructions; mais il réussira à assembler le bras, même s'il doit mettre un certain temps. Il devra surtout ingurgiter une longue introduction sur l'histoire de la robotique avant d'aboutir aux détails de la construction proposée. Toujours est-il qu'il pourra se servir astucieusement des illustrations. On notera également qu'il n'est pas possible de se procurer le Beasty totalement assemblé; mais le constructeur déclare accepter de le faire si on le lui demande!

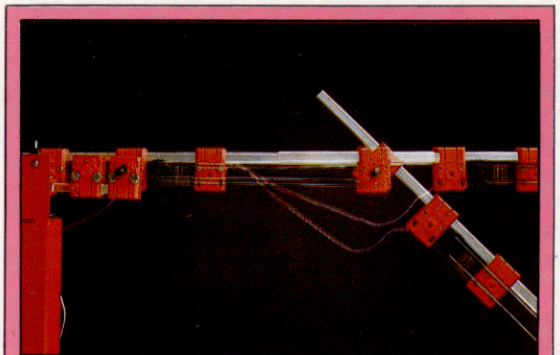
Assemblé, Beasty est composé d'une base qui supporte une articulation permettant un mouvement latéral. A cette articulation est reliée une courte tige d'aluminium, elle-même attachée à la partie supérieure du bras par une deuxième articulation. Une troisième articulation concerne l'avant-bras. Ces articulations sont actionnées par des servomoteurs qui commandent des fils métalliques reliés au squelette du bras. Lorsqu'un servomoteur tourne, il tire un fil et pousse l'autre, ce qui fait tourner l'articulation et déplace le bras. Un servomoteur fonctionne en traduisant des impulsions numériques en un mouvement. Quand la fréquence demeure constante, le moteur maintient le bras dans sa position; un changement de la fréquence d'impulsion indique au processeur qu'un nouvel angle est demandé, ce qui entraîne un déplacement du bras.

Les servos utilisés dans Beasty peuvent effectuer une traction de 3,500 kg/cm. Cela signifie qu'à une distance de 1 cm le long d'un axe le moteur peut soulever 3,500 kg, tandis qu'il ne peut soulever que 350 g à 10 cm. Il est important de tenir compte de ce fait lorsque des poids sont soulevés (le servo de l'« épaule », étant le plus éloigné du poids à soulever, aura à fournir l'effort le plus grand).

Le processeur du servomoteur est logé dans un petit boîtier noir situé à l'extérieur du bras. Ce boîtier est muni de prises pouvant connecter jusqu'à quatre servomoteurs (la quatrième connexion est destinée à un moteur optionnel qui peut servir à actionner une pince ou un dispositif de saisie similaire à l'extrémité de l'avant-bras). Il existe aussi une prise d'entrée, qui crée une interface avec le port utilisateur du micro-ordinateur, et un conducteur d'alimentation qui se branche dans la prise d'alimentation auxiliaire.

Après avoir chargé le logiciel stocké sur cassette, l'écran affiche un message pour rappeler à l'utilisateur que le système est en mode édition. Une ligne de programme en Robol est composée d'un numéro de ligne, d'une commande et d'une série de nombres, chacun de ceux-ci correspondant à l'une des quatre options de servomoteur qui sont autorisées.

Si la ligne renferme la commande MOVE, les nombres correspondent à la fréquence des impulsions qui maintiennent les servomoteurs dans leur position. Ces nombres peuvent être modifiés par l'utilisateur.



Servomoteurs
Beasty est muni en standard de trois moteurs connectés à une boîte d'interface. Les servomoteurs sont très populaires en robotique puisqu'ils peuvent fournir un couple, correspondant à la fréquence du signal numérique qu'ils reçoivent.

Pendant que le système est en mode édition, le servomoteur effectuera les opérations nécessaires aux placements de façon à mettre le bras dans la position que l'utilisateur désire. Dès que l'opérateur est satisfait des différentes positions, il doit appuyer sur RETURN; une nouvelle ligne Robol est alors affichée et une nouvelle série de mouvements peut être programmée.

Le bras peut effectuer une séquence complète de mouvements si la touche de fonction F0 est pressée. Le programme peut être exécuté à partir de toute ligne en appuyant sur F1 suivi de F0 (pour commencer l'exécution au début du programme) ou en changeant le numéro de ligne à l'aide des touches du curseur. A la fin du programme, la séquence d'interventions se répétera automatiquement. Si l'utilisateur désire interrompre le programme, une instruction MOVE peut être changée en STOP.

Temps de retard

Lors de l'exécution d'une série d'instructions MOVE, il est possible d'immobiliser le bras en incorporant une instruction WAIT suivie d'un nombre. La broche TIMER 1 du port utilisateur est ainsi sollicitée, ce qui génère une interruption. Puisque l'horloge utilise des unités de 1/100 de seconde, WAIT 100 produira un retard d'une seconde avant l'exécution de l'instruction suivante.

L'action du bras peut être considérablement accélérée en changeant la commande MOVE en JUMP. Deux instructions de synchronisation sont aussi incluses — JDELAY et MDELAY. Beasty a une temporisation intégrée qui survient avant l'exécution de chaque ligne. Celle-ci a une valeur par défaut de 20, c'est-à-dire 1/5 de seconde — mais cette valeur peut être changée en utilisant JDELAY pour les instructions JUMP, et MDELAY pour les instructions MOVE.

Le système d'exploitation Robol est facile à utiliser. Il est très simple de programmer le bras afin de lui faire effectuer des mouvements complexes. Le manuel d'utilisation est bref mais suffisant; toutefois, des programmeurs expérimentés trouveront sans doute qu'il ne donne pas assez d'information pour une programmation plus complexe. Beasty peut être commandé à partir de BASIC en utilisant le programme Driver.

La société Commotion a aussi inclus un court programme qui permet de faire des copies de programmes Robol. Malheureusement, Beasty et un lecteur de disquettes ne peuvent pas être connectés simultanément.

Cependant, malgré ces problèmes mineurs, Beasty représente une introduction valable à l'univers de la robotique. On peut dire que ce dispositif est toujours à la recherche d'une véritable application, puisqu'on ne peut pas vraiment qualifier le bras robot d'utile. Il sera probablement uniquement acheté par quelques amateurs enthousiastes. Il est cependant valable en milieu pédagogique où il pourra servir à enseigner aux étudiants les principes de la commande robotique.

BEASTY

LOGICIEL

Routines Robol et driver disponibles sur cassette.

DOCUMENTATION

Beasty est livré avec un manuel d'assemblage et un guide de programmation.

FORCES

Une introduction aisée et relativement peu coûteuse à la robotique.

FAIBLESSES

Le guide d'assemblage manque de clarté et le bras souffre d'absence d'applications, bien que le fabricant promette certaines améliorations devant permettre d'élargir son potentiel.

Où est la sortie?

Nous avons déjà mis au point le matériel et le logiciel servant à piloter un véhicule à deux moteurs. Voyons un programme devant piloter ce véhicule à l'intérieur d'un labyrinthe.

La première étape dans la construction d'un labyrinthe consiste à choisir la surface. Ce peut être un dessus de table ou une section de plancher. La zone choisie doit être divisée en carrés; la dimension des carrés dépend de la dimension du véhicule qui circulera à l'intérieur. Chaque carré doit être assez grand pour permettre au véhicule de pivoter sur 360°. Des objets comme des livres, des tasses et des morceaux de bois doivent être placés dans la zone pour former le labyrinthe.

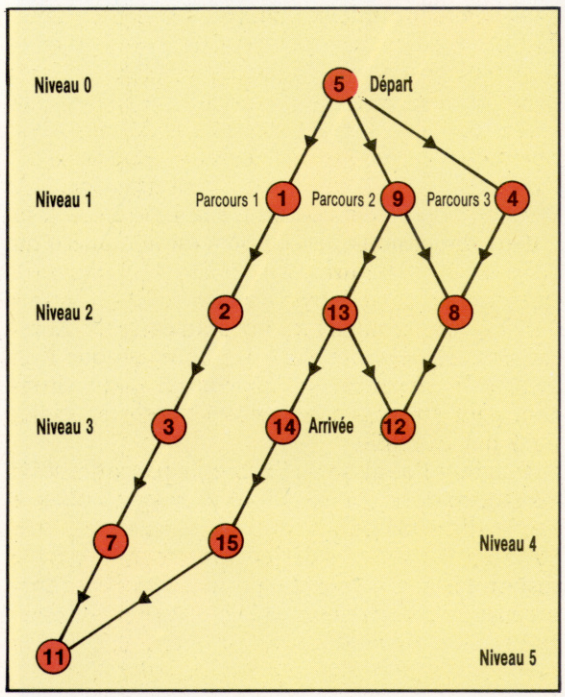
Le programme vous demande de préciser les dimensions du labyrinthe, l'emplacement des carrés occupés et de ceux qui sont libres. La méthode la plus simple est d'utiliser un code binaire : 1 indique qu'un carré est partiellement ou entièrement occupé par un objet, 0 indique qu'il est libre. Afin que les données du labyrinthe n'aient pas à être entrées lors de chaque exécution du programme, cette information doit être écrite sous la forme d'une série d'instructions DATA. Les quatre derniers éléments de données sont les coordonnées des points de départ et d'arrivée. Nous pouvons imaginer que le point d'origine du système de coordonnées se trouve au coin supérieur gauche; la ligne supérieure est la ligne 0 et la colonne à l'extrême gauche est la colonne 0. Ce labyrinthe correspond aux données DATA suivantes :

```
DATA 4,4:REM DIMENSIONS DU LABYRINTHE
DATA 1,0,0,0,0,0,1,0
DATA 0,0,1,0,0,0,0,0
DATA 1,1:REM COORD DE DÉPART
DATA 2,3:REM COORD D'ARRIVÉE
```

Trouver un parcours dans un labyrinthe ne présente pas beaucoup de difficultés. Nous pouvons concevoir un programme qui tracera un parcours, en revenant sur ses pas dans les culs-de-sac et en progressant par étapes jusqu'à ce que le point d'arrivée soit atteint. Le parcours éventuellement trouvé (sans les détours dans les culs-de-sac) peut ne pas être le plus court possible. Si nous désirons trouver le meilleur, nous devons adopter une méthode qui teste tous les parcours possibles existant entre les deux points. Nous devons signaler que notre programme interprète le meilleur parcours comme étant celui qui utilise le moins de carrés.

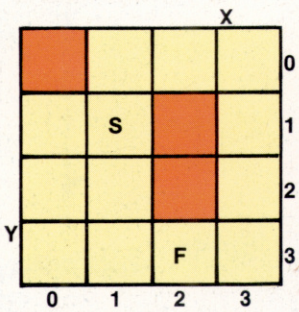
Nous pouvons rendre plus faciles les tests en créant dans les données du labyrinthe une structure qui représente les relations existant entre les carrés. La structure de données qui se prête le mieux à cette application est un arbre hiérarchique. En définissant le point de départ comme la « racine » de l'arbre, nous pouvons construire une deuxième génération de carrés (ou « nœuds ») qui se trouvent à un carré de la racine. Une troisième génération peut être construite à partir de ceux de deuxième génération, et ainsi de suite. Nous pouvons dessiner un arbre, pour tout labyrinthe, en numérotant chaque carré et en suivant la règle qui veut que les descendants de tout nœud soient dessinés de gauche à droite dans l'ordre nord, est, sud et ouest du nœud parent dans le labyrinthe initial.

Ce labyrinthe simple peut être résolu de cinq manières sans retour sur ses pas. Trois d'entre elles sont illustrées ci-dessus. Il est évident que le parcours 2 est le plus court, mais c'est parce que nous sommes en mesure d'évaluer l'arbre latéralement, c'est-à-dire que nous pouvons percevoir le labyrinthe de façon globale. L'ordinateur doit pouvoir résoudre l'arbre de façon linéaire, en examinant systématiquement chaque



Structure d'arbre
Avant de pouvoir trouver le meilleur parcours, un « arbre » représentant les relations entre les carrés du labyrinthe doit être construit. Chaque nœud est examiné séparément, créant ainsi divers niveaux. Les nœuds de niveau 1 sont situés à un carré du départ; les nœuds de niveau 2 à deux carrés du départ, etc. Il est assez facile pour nous de dessiner l'arbre, mais traduire cette structure en BASIC est plus difficile.

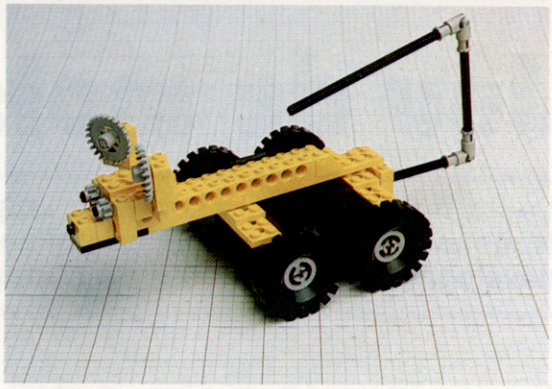
Double traitement
Le programme interprète le labyrinthe de deux manières. Comme ce dernier est lu à partir d'instructions de données, il est stocké dans un tableau à deux dimensions; les points de départ et d'arrivée sont eux aussi mémorisés sous la forme de coordonnées. Afin de résoudre le labyrinthe, le programme doit traiter chaque carré comme un « nœud » dans un arbre. Plutôt que d'utiliser le système de coordonnées initial, chaque carré est donc numéroté séquentiellement, en commençant au coin supérieur gauche du labyrinthe.
(Cl. Liz Dixon.)



parcours possible jusqu'à ce que le nœud final ou un cul-de-sac soit atteint. Dans le premier cas, il doit mémoriser le parcours positif; dans le deuxième cas, le parcours pris doit être marqué comme un cul-de-sac avant de redémarrer au nœud de départ. Le programme continuera à parcourir l'arbre jusqu'à ce que toutes les branches partant du nœud racine aient été essayées.

Le BASIC ne se prête pas facilement à ce type d'algorithme de recherche et la programmation peut souvent paraître peu élégante et difficile à manier. Des langages comme le LOGO et l'ALGOL sont beaucoup mieux adaptés. En BASIC, nous avons deux tâches principales à effectuer : d'abord construire notre arbre à partir des données du labyrinthe; puis prévoir pour chaque carré du labyrinthe quatre pointeurs qui indiquent quel carré se trouve dans chacune des quatre directions. La meilleure façon de stocker ce système de pointeurs consiste à utiliser un tableau à deux dimensions, TR(N,D), où N est le numéro du carré et D est la direction de 1 à 4. Par conséquent, dans notre labyrinthe simple, TR(9,1) serait 5 (le carré se trouvant au nord du carré 9). Lorsque le carré désigné n'est pas libre, ou que l'on atteint la bordure du labyrinthe, cette situation peut être signalée par une valeur spéciale, – 1 par exemple.

Pendant l'analyse de l'arbre, le parcours suivi est stocké dans une pseudo-pile, mise en place à l'aide d'un tableau à une dimension et d'une variable, D, servant à désigner le prochain espace disponible sur la pile. Le parcours le plus court est aussi stocké dans un tableau à une dimension; le premier élément de ce tableau renferme le nombre d'étapes que compte ce parcours.



Ian McKinnell

Quand le programme a entièrement analysé l'arbre, le meilleur parcours est mémorisé sous la forme d'une série de numéros de carrés. En supposant que le véhicule ait été d'abord dirigé vers le nord dans le carré de départ, il peut être déplacé en utilisant de simples relations mathématiques entre la direction à prendre et la différence entre deux numéros de carrés consécutifs dans le tableau de parcours. Par exemple, dans notre labyrinthe, une différence de + 4 indique le nord, – 4 indique le sud, etc. Nous devons alors calculer l'angle de rotation nécessaire pour changer de direction avant d'avancer d'un carré. Comme le véhicule utilise de simples moteurs électriques CC, les angles de rotation et les distances parcourues sont fonction de la durée de mise en œuvre d'une combinaison de moteurs. Pour mettre le programme au point, certaines expériences initiales doivent être effectuées pour déterminer les intervalles requis pour tourner de 90° et pour avancer d'un carré. Cette information doit être entrée respectivement dans les variables AF et FF.

Résoudre le labyrinthe

```
900 REM ************************
910 REM ************************
920 REM **                    **
930 REM ** LABYRINTHE CBM 64 **
940 REM **                    **
950 REM ************************
960 REM ************************
970 REM ************************
980 :
990 REM **** PROGRAMME PRINCIPAL D'APPEL ****
1000 GOSUB 1600:REM LIRE DONNEES
1100 GOSUB3700:REM AFFICHE LAB
1200 GOSUB5400:REM CONSTRUCTION ARBRE
1210 GOSUB 7700 REM ANALYSE ARBRE
1215 IF S(0)=9999 THEN PRINT "FIN":
1220 GOSUB8200:REM DIRIGE VEHICULE
1400 END
1500 :
1600 REM **** LIRE DONNEES LAB./INIT ****
1700 READ SX,SY
1800 DIM MZ(SX,SY),TR(SX*SY,4),DR(4),RT(SX*SY),
     LN(SX*SY),CN(SX*SY)
1900 FOR Y=0 TO SY-1
2000 FOR X=0 TO SX-1
2100 READ A:MZ(X,Y)=A
2200 NEXT X,Y
2300 :
2400 READ XS,YS,XF,YF
2500 DATA 4,4
2600 DATA 1,0,0,0,0,0,1,0
2700 DATA 0,0,1,0,0,0,0,0
2800 DATA 1,1:REM COORD.DEPART
2900 DATA 2,3:REM COORD.ARRIVEE
3000 :
3100 DR(1)=-SX:DR(2)=1:DR(3)=SX:DR(4)=-1
3110 ID(1)=3:ID(2)=4:ID(3)=1:ID(4)=2
3120 SR(0)=9999:REM INIT.PARCOURS
3130 DDR=56579:DATREG=56577:POKE DDR,255
3140 AF=30:FF=45:REM PLUS COURT
3200 FOR I=1 TO 25:CD$=CD$+CHR$(17):NEXT I FACTEURS
     TEMPS ROTATION ET AVANCE
3500 RETURN
3600 :
3700 REM **** AFFICHE LAB ****
3800 PRINTCHR$(147):REM EFFACE ECRAN
3900 FOR X=0 TO SX-1
4000 FOR Y=0 TO SY-1
4100 GOSUB 4900:REM PLACE CURSEUR
4200 PRINT CHR$(32+MZ(X,Y)*134)
4300 NEXT Y,X
4400 :
4500 X=XS:Y=YS:GOSUB4900:PRINT"D"
4600 X=XF:Y=YF:GOSUB4900:PRINT"A"
4700 RETURN
4800 :
4900 REM **** POSITION CURSEUR EN X,Y ****
5000 PRINT CHR$(19);:PRINTTAB(X)LEFT$(CD$,Y);:RETURN
5300 :
5400 REM **** CONSTRUCTION ARBRE ****
5500 :
5600 REM ** INITIALISE ****
5700 FOR P=0 TO SX*SY-1:FORI=1TO4:TR(P,I)=-1:NEXT I,P
6100 :
6110 REM ** CALCULATE START &  FINISH ****
6120 X=XS:Y=YS:GOSUB9200:S=N
6130 X=XF:Y=YF:GOSUB9200:F=N
6140 :
6200 REM ** CONSTRUCT INIT **
6300 LC=1:CC=0:REM INIT POINTEURS DE PILE
6400 LN(LC)=S:REM POINT DEPART
6450 CN=LN(LC):REM RETIRE NOEUD DE LA PILE
6500 DF=0
6600 FOR D=1 TO 4
6700 N=CN+DR(D):GOSUB 8900:REM CONVERTIR EN X,Y
6800 IF(X<0 OR X>SX-1 OR Y<0 OR Y>SY-1)THEN 7300
6900 IF MZ(X,Y)=1 THEN 7300
7000 IF (D=2 AND N/SX=INT(N/SX))THEN7300
7100 IF (D=4 AND CN/SX=INT(CN/SX))THEN7300
7200 IF TR(N,ID(D))=CN THEN 7300
7210 TR(CN,D)=N:DF=1
7220 CC=CC+1:CN(CC)=N:REM METTRE SUR PILE EN COURS
7300 NEXT D
7310 IF(DF=0 AND LC-1) THEN RETURN:REM NOEUD FINAL
7320 :
7330 LC=LC-1:REM DERNIER POINTEUR PILE
7340 IF LC>0 THEN 6450:REM NOEUD SUIVANT
7345 :
7350 REM ** COPIE PILE ACTUELLE SUR DERNIERE PILE **
7360 FOR I=1 TO CC:LN(I)=CN(I):NEXT I
7390 LC=CC:CC=0:GOTO6450:REM NOEUD SUIVANT
7600 :
7700 REM **** ANALYSE ARBRE ****
7720 C=0:RN=S:CN=RN:EF=0
7730 C=C+1:RT(C)=CN
7740 IF CN=F THEN GOSUB8100:GOSUB8000:IF EF=0 THEN 7730
7745 IF EF=1 THEN RETURN
7750 DF=0
7760 FOR D=1 TO 4
7770 IF TR(CN,D)>-1 THEN CN=TR(CN,D):DF=1:DR=D:D=4
7780 NEXT D
7790 IF DF=0 THEN GOSUB8000
7800 IF EF=0THEN 7730
7810 IF EF=1 THEN RETURN
7820 :
8000 REM **** RETOUR A LA RACINE ****
8010 TR(RT(C-1),DR)=-1
8020 CN=RN:C=0
8030 IF(TR(CN,1)ANDTR(CN,2)ANDTR(CN,3)ANDTR(CN,4)>-1
     THEN EF=1
8040 RETURN
8050 :
8100 REM **** SAVE ARRAY ****
8110 IF C>SR(0) THEN RETURN:REM NOUVEAU PARCOURS PLUS COURT?
8120 SR(0)=C
8130 FOR I=1 TO C:SR(I)=RT(I):NEXT I:RETURN
8140 :
8200 REM **** DIRIGE VEHICULE ****
8205 PD=1:REM SUPPOSE DIRECTION INITIALE NORD
8210 FOR C=1 TO SR(0)-1
8220 DF=SR(C+1)-SR(C)
8222 :
8225 REM ** TROUVE DIRECTION REQUISE **
8230 FOR I=1 TO 4
8240 IF DF=DR(I) THEN D=I:I=4
8250 NEXT I
8260 :
8265 DR=D-PD:PD=D
8270 H=INT((4+DR/4):R=(4+DR)-4*H
8275 :
8277 REM ** DO TURN **
8280 FOR I=1 TO R
8290 POKE DATREG=9:REM VIRAGE SENS AIGUILLES MONTRE
8300 T=TI
8310 IF (TI-T)<AF THEN 8310:REM ATTEND
8320 POKE DATREG=0:REM OFF
8330 NEXT I
8340 :
8350 REM ** EN AVANT **
8360 POKE DATREG=5
8370 T=TI
8380 IF (TI-T)<FF THEN 8380
8390 POKE DATREG=0
8400 NEXT C
8410 :
8420 RETURN
8430 :
8900 REM **** CONVERTIT EN X,Y ****
9000 Y=INT(N/SX):X=N-SX*Y:RETURN
9200 :
9210 REM **** CONVERTIT X,Y en N ****
9220 N=Y*SX+X:RETURN
```

Jeu de mots

Le traitement de liste permet à LOGO de s'appliquer aux jeux. Nous montrons ici comment utiliser ce langage pour le développement d'un jeu d'aventures fondé sur le texte.

Nous nous limitons dans cet article aux aspects les plus généreux de la programmation d'un jeu d'aventures avec LOGO. Nous verrons ultérieurement comment traiter les détails propres à chaque jeu.

Dans tout jeu d'aventures, le joueur est confronté à cinq types d'activité de base : déplacer des objets graphiques et les poser, recenser les objets déplacés, observer les alentours et se déplacer d'une position à l'autre. Il s'agit donc des commandes à programmer en premier. Pour des raisons de simplicité, nous aurons des commandes de deux types : des verbes comme REGARDE, et des verbes avec complément comme POSE.ANNEAU. Le programme gérera deux listes : l'une, appelée INVENTAIRE, qui recense les objets que le joueur déplace couramment, et l'autre, appelée simplement CONTENU, qui recense les objets présents dans la pièce.

La première commande définie est INVENTAIRE :

```
POUR INV
   AFFICHE [VOUS DÉPLACEZ:]
   SI VIDE? :INVENTAIRE ALORS AFFICHE [RIEN]
   SINON AFFICHE :INVENTAIRE
FIN
```

Vous remarquerez que cette commande utilise la forme complète de l'instruction SI : SI < expression conditionnelle > ALORS < action 1 > SINON < action 2 >. La commande pour prendre un objet est PRENDRE :

```
POUR PRENDRE : ÉLÉMENT
   SI APPARTIENT? :ÉLÉMENT :CONTENU ALORS PRENEZLE :
   ÉLÉMENT SINON AFFICHE [IMPOSSIBLE IL N'Y A RIEN]
FIN
```

APPARTIENT? est une primitive qui vérifie si un élément appartient à la liste. Pour « prendre » un élément, il faut faire deux choses : l'ajouter à l'inventaire des objets déplacés, et le retirer de la liste du contenu de la pièce. Voici les procédures qui s'en chargent :

```
POUR PRENEZLE : ÉLÉMENT
   AJOUTE.A.INV :ÉLÉMENT
   RETIRE.DE.LA.PIÈCE :ÉLÉMENT
FIN

POUR AJOUTE.A.INV :ÉLÉMENT
   FAIRE «SUPPRIME CONTENU :ÉLÉMENT :CONTENU
FIN

RETIRE DE LA PIÈCE :ÉLÉMENT
FAIRE «SUPPRIME CONTENU :ÉLÉMENT CONTENU
FIN
```

La dernière de ces procédures suppose de supprimer un élément d'une liste (c'était l'un des exercices du dernier numéro) :

```
POUR SUPPRIMER :ÉLÉMENT :LISTE
   SI :ÉLÉMENT = PREMIER :LISTE ALORS RÉSULTAT
   SAUFPREMIER
   :LISTE
   RÉSULTAT PHRASE PREMIER :LISTE SUPPRIME :ÉLÉMENT
   SAUFPREMIER :LISTE
FIN
```

La commande pour poser un objet s'implémente de manière similaire :

```
POUR POSER :ÉLÉMENT
   SI APPARTIENT? :ÉLÉMENT :INVENTAIRE ALORS POSELE
   :ÉLÉMENT SINON AFFICHE [IL N'Y A RIEN A POSER!]
FIN

POUR POSELE :ÉLÉMENT
   RETIRE.DE.INV :ÉLÉMENT
   AJOUTE.A.LA.PIÈCE :ÉLÉMENT
FIN

POUR RETIRER.DE.INV :ÉLÉMENT
   FAIRE «SUPPRIME INVENTAIRE :ÉLÉMENT :INVENTAIRE
FIN

POUR AJOUTER.A.LA.PIÈCE : ÉLÉMENT
   FAIRE «MET CONTENU :ÉLÉMENT :CONTENU
FIN
```

Après avoir saisi ces procédures, il est temps de voir comment elles fonctionnent. Il nous faut en premier définir les deux variables globales INVENTAIRE et CONTENU, et mettre ensuite en œuvre les commandes suivantes :

```
FAIRE «CONTENU [ÉPÉE LANCE FLAMBEAU]
FAIRE «INVENTAIRE [LANTERNE]
PRENDRE «ÉPÉE
POSER «LANTERNE
```

Essayez CONTENU et INVENTAIRE :

```
AFFICHE :CONTENU
AFFICHE :INVENTAIRE
```

et vérifiez que tout se passe bien.

Vous remarquez que nous utilisons des guillemets avant les noms des objets pour les commandes PRENDRE et POSER. Cette utilisation des guillemets peut paraître évidente à un programmeur LOGO confirmé, mais sera sûrement déroutante

pour un novice. Afin de permettre l'utilisation de PRENDRE ÉPÉE, qui est plus naturelle, il nous faut définir ÉPÉE :

```
POUR ÉPÉE
   CRÉER «ÉPÉE
FIN
```

Il nous faudra bien sûr répéter cette définition pour tous les noms du jeu.

La commande REGARDER affiche la description de la pièce, une liste de son contenu, et les sorties possibles. Il nous faut, pour REGARDER, deux listes supplémentaires, une liste de description et une liste des sorties. Pour permettre des descriptions relativement longues sur plus d'une ligne d'écran, la liste de description est définie comme une liste de listes. Par exemple :

```
FAIRE «DESCRIPTION [[VOUS ÊTES SUR LE SEUIL] [D'UNE
CAVE]]
```

Pour mémoriser l'enchaînement des pièces, chacune d'elles porte un numéro distinctif. La liste des sorties est simplement une liste de sous-listes, chacune consistant en une direction et un numéro. Ainsi :

```
FAIRE «LISTE.SORTIES [[N 4] [E 6]]
```

Nous pouvons maintenant définir REGARDER :

```
POUR REGARDER
   AFFICHELISTE :DESCRIPTION
   AFFICHE »
   AFFICHE [VOUS VOYEZ:]
   SI VIDE? : CONTENU ALORS AFFICHE [RIEN DE PARTICU-
   LIER] SINON AFFICHE :CONTENU
   AFFICHE »
   AFFICHE [VOUS POUVEZ Y ALLER:] AFFICHE.SORTIES
   :LISTE.SORTIES
   AFFICHE »
FIN
```

Deux routines spéciales d'affichage ont été utilisées ici afin de rendre l'affichage plus lisible. AFFICHE permet d'afficher plusieurs lignes de texte.

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

AFFICHE.SORTIES permet d'afficher les sorties de la pièce sans mentionner leur numéro.

```
POUR AFFICHE.SORTIES :LISTE
   SI VIDE? :LISTE ALORS AFFICHE « STOP
   FAIRE «SORTIE PREMIER :LISTE
   AFFICHE PREMIER :SORTIE
   AFFICHE 1 »
   AFFICHE. SORTIES SAUFPREMIER :LISTE
FIN
```

Nous décrivons ce qui se trouve dans la pièce pour le jeu, en ajoutant bout à bout les trois sous-listes : la description, le contenu et les sorties. Par exemple :

```
FAIRE «PIÈCE.1 [[[VOUS ÊTES SUR LE SEUIL] [D'UNE CAVE]]
[ÉPÉE] [[N 4] [E 6]]]
```

La PIÈCE.1 étant définie de la sorte, nous pourrions éclater cette procédure en ses constituants élémentaires :

```
POUR ASSIGNER.VARIABLES
   FAIRE «PIÈCE OBJET «PIÈCE.1
   FAIRE «DESCRIPTION DESCRIPTION :PIÈCE
   FAIRE «CONTENU CONTENU :PIÈCE
   FAIRE «LISTE.SORTIES LISTE.SORTIES :PIÈCE
FIN
```

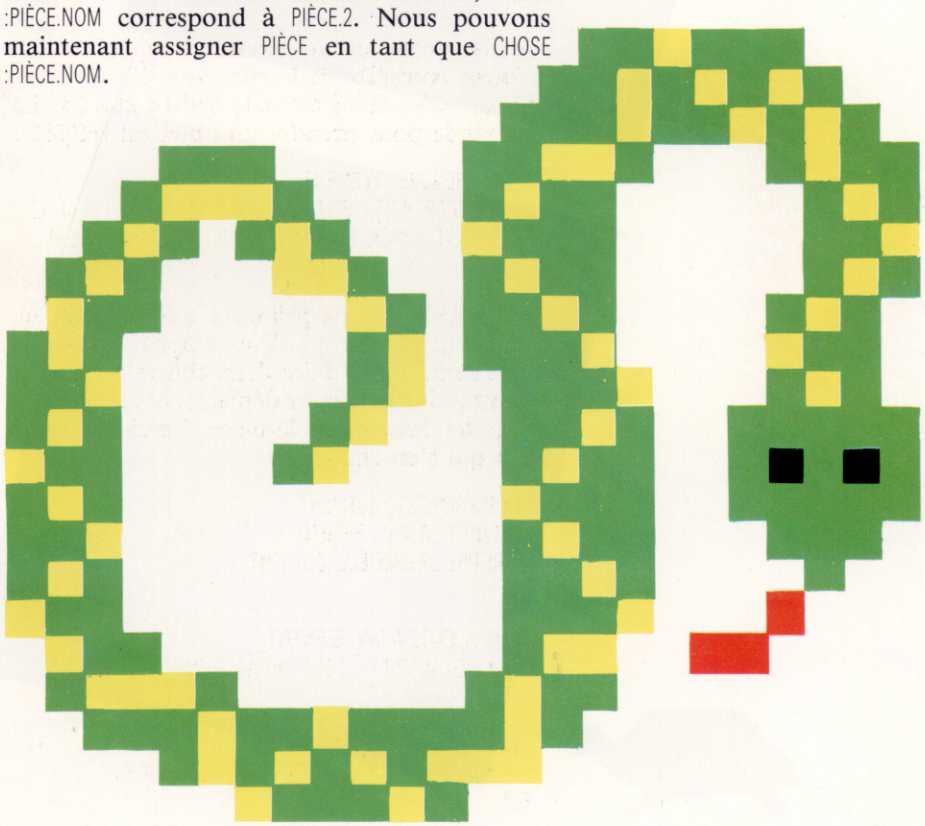
OBJET «PIÈCE.1 est une alternative à :PIÈCE.1. Elle signifie : « Le contenu de la variable PIÈCE.1. » Nous verrons l'utilité de cette syntaxe plus loin. Les sous-procédures se définissent comme suit :

```
POUR DESCRIPTION :PIÈCE
   RÉSULTAT ÉLÉMENT 1 :PIÈCE
FIN

POUR CONTENU :PIÈCE
   RÉSULTAT ÉLÉMENT 2 :PIÈCE
FIN

POUR SORTIE.LISTE :PIÈCE
   RÉSULTAT ÉLÉMENT 3 :PIÈCE
FIN
```

Telle qu'elle est, cette procédure n'est valable que pour PIÈCE1. Il nous faut la rendre plus générale si on veut qu'elle s'applique à toutes les pièces. Nous utilisons à cette fin une variable locale, ICI, qui contient le numéro de la pièce. Supposons qu'il s'agisse de la 2. La primitive MOT donne un mot se présentant comme la combinaison de ses deux entrées. Par exemple, MOT « PIÈCE. :ICI donnera PIÈCE.1. Nous assignons ensuite à ce nom la variable PIÈCE.NOM, ainsi :PIÈCE.NOM correspond à PIÈCE.2. Nous pouvons maintenant assigner PIÈCE en tant que CHOSE :PIÈCE.NOM.



```
PIÈCE COMME CHOSE :PIÈCE.NOM
POUR ASSIGNER.VARIABLES
   FAIRE «NOM.PIÈCE MOT «PIÈCE. :ICI
   FAIRE «PIÈCE CHOSE :PIÈCE.NOM
   FAIRE «DESCRIPTION DESCRIPTION :PIÈCE
   FAIRE «CONTENU CONTENU :PIÈCE
   FAIRE «LISTE.SORTIE LISTE.SORTIE :PIÈCE
FIN
```

Il vous est maintenant possible de faire la carte de votre jeu d'aventures, et de lister les positions (avec contenu et sorties). Nous achèverons plus loin ces généralités en étudiant les déplacements entre positions ainsi que les moyens de mettre en œuvre les « périls ».

Variantes de logo

Certaines versions de MIT LOGO ne comportent pas VIDE?, ÉLÉMENT, COMPTER et APPARTIENT?. Nous avons donné dans le précédent numéro des définitions pour les trois premiers. Voici la définition de APPARTIENT? :

```
POUR APPARTIENT? :ÉLÉMENT :LISTE
   SI :LISTE = [] ALORS RÉSULTAT « FAUX
   SI :ÉLÉMENT = PREMIER :LISTE ALORS RÉSULTAT «VRAI
   RÉSULTAT APPARTIENT? :ÉLÉMENT SAUFPREMIER :LISTE
FIN
```

Pour toutes les versions LCSI, utilisez :

```
VIDEP pour VIDE?
LISTEP pour LIST?
APPARTIENTP pour APPARTIENT?
TAPER pour AFFICHE1
```

EGALP est une primitive qui teste l'équivalence de deux entrées. Utilisez-la pour comparer des listes ou des mots, à la place de =. Le signe « égal » est accepté par certaines versions LCSI, mais pas par toutes. La syntaxe complète pour SI selon LOGO LCSI est donnée dans l'exemple suivant :

```
SI VIDEP : CONTENU [AFFICHE [RIEN DE SPÉCIAL]] [AFFICHE :CONTENU]
```

La première liste après l'expression conditionnelle est exécutée lorsque la condition est vraie; la deuxième, lorsque l'expression est fausse.

Réponses

1. Afficher en ordre inverse :

```
POUR AFFICHE INV :LISTE
   SI VIDE? :LISTE ALORS AFFICHE «STOP
   AFFICHE 1 DERNIER :LISTE
   AFFICHE1 »
   AFFICHE INV SAUFDERNIER :LISTE
FIN
```

La procédure suivante affiche la liste inversée plutôt que de l'imprimer :

```
POUR INVERSER :LISTE
   SI VIDE? :LISTE ALORS RÉSULTAT []
   RÉSULTAT PHRASE DERNIER :LISTE INVERSE SAUFDERNIER :LISTE
FIN
```

2. Nous donnons dans le texte plus haut une procédure pour détruire un élément d'une liste.

Poésie logo

Nous avions posé précédemment le problème de l'amélioration de l'écriture poétique avec le LOGO. Une des méthodes possibles est de créer un modèle type de phrase (par exemple, « nom verbe complément ou nom »), et de prendre des mots dans des listes successives représentant ces catégories. Par exemple, avec ce schéma :

```
POUR POÈME2 [NOM VERBE NOM]
```

Nous pourrions avoir :

```
POUR POÈME 2 :SCHÉMA
   SI VIDE? :SCHÉMA AFFICHE «STOP
   POÈME2.1 D'ABORD :SCHÉMA
   POÈME 2 SAUFPREMIER :SCHÉMA
FIN

POUR POÈME2.1 :MOT
   SI :MOT = «NOM [AFFICHE1» PRENDREHASARD :NOM]
   SI :MOT = «VERBE [AFFICHE2» PRENDRE HASARD :VERBE]
FIN
```

Cette méthode deviendra vite très lourde si nous mettons en œuvre beaucoup d'autres catégories. Revoyons à nouveau les variables pour d'éventuelles améliorations. Commencez par taper :

```
FAIRE «ROSE «DOUCE
FAIRE «AUTRENOM «ROSE
```

Vous découvrez ceci :

```
AFFICHE :ROSE affiche DOUCE
AFFICHE :AUTRENOM affiche ROSE
AFFICHE CHOSE :AUTRENOM affiche DOUCE
```

CHOSE nous donne la valeur associée à un nom. Dans le dernier cas, le nom suivant CHOSE est la valeur de la variable AUTRENOM, c'est-à-dire ROSE. Nous pouvons maintenant réécrire la procédure pour notre poème en nous fondant sur ce principe :

```
POUR POÈME2.1
   [AFFICHER1» PRENDREHASARD CHOSE :MOT]
FIN
```

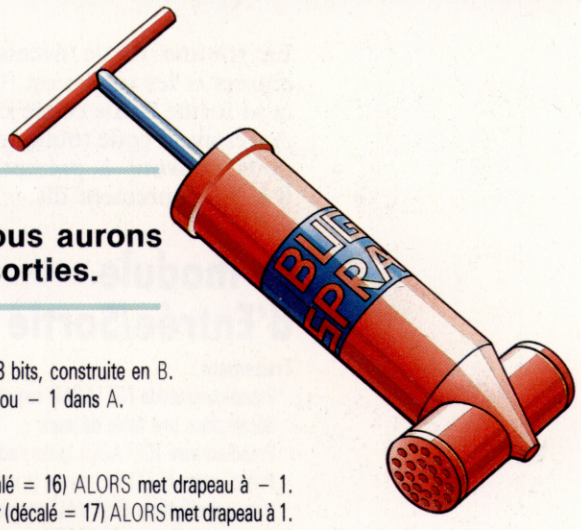
L'appel de procédure POÈME2.1«NOM assigne la valeur NOM à la variable MOT. CHOSE:MOT est alors la valeur associée à NOM (la liste des noms). En utilisant le schéma suivant :

```
POÈME2 [ART ADJ NOM VERBE ADV PREP ART NOM ADJ]
```

avec un choix approprié de vocabulaire, nous obtenons le résultat suivant :

```
UNE VERTE PLANÈTE TOURNA BRUYAMMENT SOUS UN DOME PARANOÏDE
UN VAISSEAU DÉLABRÉ PARTIT LENTEMENT POUR LA PLANÈTE
SPLENDIDE
```

Insecticide



Complétons le développement de notre programme de « débogage » par l'ensemble des routines dont nous aurons besoin pour le module qui manie les entrées et les sorties.

Quatre routines restent encore à développer pour le module E/S : GETHX2, GETHX4, PUTHEX et PUTCR. Les deux premières servent à entrer des chiffres hex au clavier : GETHX2 entre un nombre hex à deux chiffres et GETHX4 à quatre chiffres. La première chose à faire lorsqu'on conçoit ces routines est de décider si nous voulons toujours entrer deux ou quatre chiffres (ce qui est plus facile à programmer) ou moins de caractères suivis d'un retour chariot.

Nous utiliserons la méthode la plus simple pour la routine GETHX4 : il faut entrer 4 chiffres, et l'espace arrière n'est pas admis. La valeur 16 bits (correspondant à une adresse) peut être restituée dans le registre D.

GETHX2 pose plus de problèmes si nous considérons les circonstances de son utilisation. Il faudra entrer des quantités à 8 bits pour l'inspection et la modification de mémoire (commande M), qui implique l'accès à une adresse. Le contenu de cette adresse est affiché, et l'utilisateur peut alors entrer un retour chariot (pour se déplacer vers l'espace suivant) ou un nombre hex à deux chiffres (qui sera stocké à cet emplacement) ou encore un autre caractère (un point, par exemple, pour sortir du niveau commande). Nous pouvons ajouter les deux caractères valides supplémentaires à la fin de la chaîne de chiffres hex valides. GETHX2 doit alors accepter deux chiffres hex, soit un retour (RETURN), soit un point (DOT). La valeur 8 bits peut être mise en B, et il faut utiliser A pour indiquer quelle est la situation. A prend la valeur 0 si on a entré un nombre à deux chiffres, 1 si c'est un retour chariot ou -1 si c'est un point. Ces valeurs nous permettent de tester la valeur en A sans avoir à la comparer à une autre valeur.

Supposons que les déclarations suivantes ont été faites pour ce module :

```
HEXCHS  FCC'0123456789ABCDEF'
  DOT   FCB'.
  RETURN FCB 13 (Code ASCII pour retour chariot.)
```

Nous pouvons faire passer 16 comme la longueur de la chaîne pour GETHX4, où il ne faut que des chiffres hex, et 18 comme la longueur pour GETHX2, mais où il nous faut également les deux autres caractères.

La routine GETHX2

Données :
Le **caractère-suivant** est le code ASCII en A.
Décalé dans caractère-valide en B.
La **valeur-hex** est une valeur 8 bits, construite en B.
Le **drapeau** a pour valeur 0,1 ou – 1 dans A.

Traitement :
Prend caractère-suivant.
SI caractère est un point (décalé = 16) ALORS met drapeau à – 1.
SINON si caractère est un retour (décalé = 17) ALORS met drapeau à 1.
SINON Sauvegarde temporairement décalé. Prend caractère-suivant (chiffres hex seulement valides en ce point).
Construit valeur-hex.
FINSI

Le codage de la routine GETHX4 est maintenant légèrement facilité en utilisant des parties de cette routine. En faisant de HX4 un point d'entrée possible pour la routine GETHX2, nous pouvons appeler cette routine et nous assurer que seuls des chiffres hex valides sont acceptés — pourvu que nous chargions B avec 16 avant l'appel. Ainsi, le traitement pour entrer quatre chiffres hex devient beaucoup moins complexe.

La routine GETHX4

Données :
Le **nombre-hex** est une valeur 16 bits qui doit retourner en D.
Les **octets le-plus-significatif** et **le-moins-significatif** sont tous deux des valeurs 8 bits qui doivent retourner en B.

Traitement :
Prend l'octet le-plus-significatif.
Le sauvegarde temporairement.
Prend l'octet le-moins-significatif.
Construit un nombre-hex.

La routine est donnée, dans sa forme finale, après celle de GETHX2.

Les routines pour afficher ces caractères sont moins compliquées à concevoir. Pour la routine PUTHEX, nous supposerons que le nombre 8 bits requis doit se trouver en B.

La routine PUTHEX

Données :
Le nombre est une valeur 8 bits trouvée en B.
Le décalé est la valeur à 4 bits entrée dans HEXCHS.

Traitement :
Extrait les 4 bits les plus significatifs du nombre comme décalé
Affiche HEXCHS (décalé).
Extrait les 4 bits les moins significatifs du nombre.
Affiche HEXCHS (décalé).

La routine finale nécessaire pour manier les entrées et les sorties est PUTCR. Elle est évidente, et sa forme finale codée s'explique d'elle-même. Après avoir codé toutes les routines nécessaires, nous pouvons à présent concevoir le module d'E/S proprement dit.

Le module d'Entrée/Sortie

Traitement :

Prend-commande (GETCOM) remet décalé en B, qui peut servir de décalé dans une table de saut.
Prend-adresse (GETADD) laisse l'adresse de retour en D.
Prend-valeur (GETVAL) remet la valeur en B, le drapeau en A.
Valeur-d'affichage (DSPVAL) est passée en B.
Adresse-d'affichage (DSPADD) est passée en D.

La forme finale codée du module E/S est donnée à la page suivante. Nous pouvons revenir au module point-d'interruption que nous avions commencé dans le dernier cours. Nous avons déjà donné le code pour le second traitement de ce module, qui met des points d'interruption. Nous avons laissé de côté le problème du codage des premiers traitements (insertion de points d'interruption) parce qu'ils nécessitaient une adresse. Maintenant que nous avons résolu cela avec les routines données ici, nous pouvons donner la version codée du traitement — qui incorpore un branchement au sous-programme GETADD.

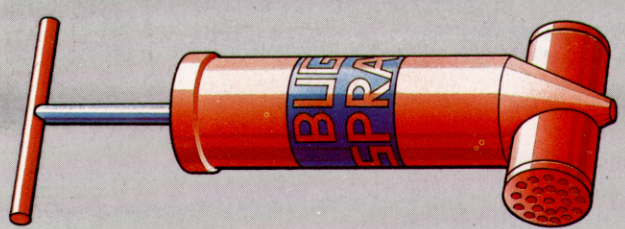
Notez que, dans le code, la commande INC NUMBP, PCR ajoute 1 au nombre-d'interruptions. A est alors inférieur d'une unité au nombre-d'interruptions, qui est le décalé correct dans la table d'interruption. Cependant l'adresse est

Routine GETHX2

```
GETHX2   LDB    #18          Nombre-de-caractères-valides
HX4      PSHS   X            Sauvegarde registre utilisé
         LEAX   HEXCHS,PCR   Prend adresse de car.-valide dans X
         BSR    GETCH        Prend caractère-suivant
IFOO     CMPB   #16          Si Décalé = 16
         LDA    #$FF         Met drapeau à –1 (en complément à 2)
         BRA    ENDF00
         CMPB   #17          Si Décalé = 17
         LDA    #1           Met drapeau à 1
         BRA    ENDF00
         LSLB                Décale B de 4 positions pour former le chiffre le plus significatif; B contient le décalé en HEXCHS et donc la valeur binaire
         LSLB
         LSLB
         LSLB
         PSHS   B
         LDB    #16          Sauvegarde temporairement B
         BSR    GETCH        Seuls les chiffres hex maintenant valides
         ADDB   1,S+         Caractère-suivant
         PULS   X,PC         Construit nombre 8 bits et laisse B temporairement
```

Routine PUTCR

```
PUTCR    PSHS   A            Sauvegarde A
         LDA    #13          Code ASCII pour Retour
         BSR    OUTCH        Affiche ce nombre
         PULS   A,PC
```



Routine GETHX4

```
GETHX4   LDB    #16
         BSR    HX4          Prend l'octet le plus significatif
         PSHS   B            Le sauvegarde temporairement
         LDB    #16
         BSR    HX4          Prend l'octet le moins significatif dans B
         PULS   A            Reprend l'octet le plus significatif dans A
         RTS                 La valeur requise est dans D
```

Routine d'affichage d'interruptions

```
BPLABS   FCC    '12345678910111213141516'
SPACE    FCB    32           Code ASCII pour un espace
DISPBP   PSHS   A,B,X,Y
         LEAX   BPTAB,PCR    Adresse de la table-d'interruptions
         LEAY   BPLABS,PCR   Adresse des labels
         CLRB                Met numéro-d'interruption à décalé zéro
WHIL01   CMBP   NUMBP,PCR    Tant que numéro d'interruption < = nombre d'interruptions
         BGT    ENDW01
         LDA    ,Y+
         BSR    OUTCH
         LDA    ,Y+          Affiche label
         BSR    OUTCH
         LDA    SPACE,PCR    Affiche un espace
         BSR    OUTCH
         PSHS   B            Sauvegarde temporairement B
         LDD    ,X++         Sauvegarde adresse
         BSR    DSPADD
         PULS   B            Restaure B
         BRA    WHIL01
ENDW01   PULS   A,B,X,Y      Restaure et retour
```

placée en D, et cela va détruire la valeur en A parce que le registre D comprend les registres A et B. C'est pourquoi nous utilisons Y, pour y mettre l'adresse en cours.

Après avoir codé le premier traitement d'interruption, il reste encore trois traitements. Deux d'entre eux inversent les deux traitements que nous avons codés jusqu'à présent : enlever-interruption supprimera un point d'interruption de la table, et Réinitialiser-interruption enlève l'opc SWI et remet la valeur initiale. Nous verrons ces deux routines. La troisième routine, pour afficher toutes les interruptions, est la dernière qui sera codée ici.

L'apprentissage de ces routines peut sembler une tâche fastidieuse et sans utilité immédiate. Il n'en est rien. Il suffit d'un seul « bug » pour que le meilleur programme ne fonctionne pas.

Affichage d'interruptions

Données :

Le **numéro-d'interruption** est un compteur 8 bits qui parcourt la table d'interruption en B.

L'**interruption-actuelle** est l'adresse à afficher.

Les **labels-d'interruption** sont à deux chiffres (décimaux) pour étiqueter les adresses lorsqu'elles sont affichées.

Espace est le caractère d'espacement qui sépare un label d'une adresse.

Traitement 3 : Affichage d'interruptions :

Met le numéro d'interruption à 1 (un décalé de zéro).
TANT QUE Numéro-d'interruption < = Nombre-d'interruptions.
  Afficher Labels d'interruption (numéro-d'interruption).
  Afficher table d'interruption (numéro-d'interruption).
    Incrémenter numéro-d'interruption.
FIN (TANT QUE)

Fin du traitement 3

Routine PUTHEX

```
PUTHEX   PSHS   A,B,X          Sauvegarde registres utilisés
         PSHS   B              Sauvegarde temporairement B
         LEAX   HEXCHS,PCR     Adresse de HEXCHS dans X
         LSRB                  4 décalages à droite pour les 4 bits les plus significatifs
         LSRB
         LSRB
         LSRB
         LDA    B,X            Prend caractère approprié en A
         BSR    OUTCH          L'affiche
         PULS   B              Reprend B
         ANDB   #%00001111     Masque les 4 bits les plus significatifs
         LDA    B,X            Second caractère
         BDR    OUTCH
         PULS   A,B,X,PC       Restaure et retour
```

Routine Insertion d'interruption

```
BP01     PSHS   A,B,X,Y        Sauvegarde registres utilisés
         LDX    BPTAB          Adresse de table-d'interruptions
         LDA    NUMBP,PCR
IF01     CMPA   MAXBP,PCR      Si Nombre-d'interruptions < Max
         BGE    ENDF01
         INC    NUMBP,PCR      Additionner 1 à Nombre-d'interruptions
         LSLA                  Multiplié Décalé par 2 pour la table 16 bits
         LEAY   A,X
         BSR    GETADD         Prend l'adresse
         STD    ,Y             Stocke l'adresse dans la table-d'interruptions
ENDF01   PULS A,B,X,Y          Restaure et retour
```

Module d'Entrée/Sortie

```
HEXCHS   FCC      '0123456789ABCEDF'
DOT      FCB      '.
RETURN   FCB      13             Code ASCII pour Retour
COMNDS   FCC      'BUDSGRMQ'
GETCOM   PSHS     A,X            Sauvegarde contenus de A et X
         LEAX     COMNDS,PCR     Adresse de caractères en commande en X
         LDB      #8             Nombre de caractères-valides
         BSR      GETCH          Prend caractère
         PULS     A,X,PC         Retour
GETADD   BSR      GETHX4
         BSR      PUTCR
         RTS
GETVAL   BSR      GETHX2
         BSR      PUTCR
         RTS
DSPVAL   BSR      PUTHEX
         BSR      PUTCR
         RTS
DSPADD   PSHS     B              Sauvegarde temporairement B
         TFR      A,B            Octet le plus significatif en B
         BSR      PUTHEX
         PULS     B              Récupère B
         BSR      PUTHEX
         PULS     B
         BSR      PUTHEX
         BSR      PUTCR
         RTS
```

Jeu de truands

Mugsy est un jeu de stratégie produit par Melbourne House. Ici, le joueur prend la place de Mugsy, chef de bande menant un racket de « protecteur », dans l'Amérique des années trente.

Mugsy doit prendre diverses décisions sur la conduite du gang et sur le sort de ses « clients » : quels sont ceux qui doivent subir des « pressions », combien dépenser pour les munitions et reverser à la police en pots-de-vin ! Si le gang manque d'armes, il sera éliminé. En outre, si la police ne touche pas assez, elle harcèlera le gangster et lui raflera tout.

Le personnage principal à l'écran est Louey, le bras droit de Mugsy. Il donne, au début de la partie, un aperçu des règles du jeu. Le joueur prend ses décisions stratégiques en réponse aux messages de Louey, et attend les résultats. Deux ou trois écrans d'animation apparaissent alors. Sur le premier se déclenche une fusillade dans un bar clandestin, au cours de laquelle un gangster élimine un rival. Le deuxième montre une rue et un gangster qui, penché à la portière d'une voiture, lâche une rafale de mitraillette.

Après la « pause », Louey revient avec les résultats des décisions de l'année précédente dont

Au pays des gangs
Voici deux scènes tirées de Mugsy. La première montre un épisode de la phase de questions/réponses du jeu, au cours duquel Mugsy vous informe du prix courant pratiqué pour les « clients ». La deuxième appartient à l'une des séquences d'animation. Remarquez le halo blanc autour du voyou sur l'escalier : c'est un exemple de masquage d'attribut pour les positions d'un caractère.
(Cl. Ian McKinnell.)

l'exercice financier doit se conclure par des profits. Un autre exercice recommence avec la répétition de la routine de questions et de réponses.

Le score du joueur est donné à la fin de la partie sous la forme d'un pourcentage. Ce dernier dépend du « fric fait » pendant l'année, du nombre de clients « capitalisés » et du temps qu'a pu tenir Mugsy avant de se faire descendre à son tour...

A part les instructions qui apparaissent sur une petite fenêtre dans la partie inférieure de l'écran et qui informent sur le nombre de « voyous » et la quantité d'« oseille bien au chaud », les écrans sont constitués exclusivement d'images graphiques. Comme l'écran graphique haute-résolution du Spectrum occupe environ 6 K, il doit nécessairement y avoir une technique de compression des données pour loger le code dans l'espace mémoire disponible.

Un examen attentif de Mugsy révèle que les programmeurs ont eu recours à diverses techniques pour économiser de la place mémoire. Dans la plupart des cas, les images sont constituées d'une suite de traits, et la couleur est utilisée parcimonieusement. Des commandes de logiciel graphique Melbourne Draw comme REMPLIR et TRACER permettent de constituer des images avec un minimum de code. TRACER, par exemple, ne demande que deux coordonnées pour sauvegarder un trait. La topographie de bits, par exemple, suppose de tracer tout un ensemble de points pour obtenir le même effet. La manière dont le Spectrum sauvegarde les informations sur les couleurs impose certaines des méthodes utilisées. Dans la scène de la rue animée, par exemple, une voiture passe devant une fenêtre d'où quelqu'un regarde. Comme le Spectrum ne permet pas d'afficher plus de deux couleurs pour le même carré d'un caractère, il faut avoir recours à un « masque » pour changer très rapidement de couleur : faute de quoi, le visage à la fenêtre prendrait la couleur de la voiture. Les attributs pour la couleur (CLIGNOTEMENT, BRILLANCE, GOUACHE et PAPIER) figurent dans un seul octet. Pour générer un masque de premier plan, il faut modifier l'attribut GOUACHE qui réside sur les 3 bits les moins significatifs de l'octet. L'octet est d'abord AJOUTE (AND) à 248 afin de mettre la couleur de INK-GOUACHE à 0. Après quoi, il est à nouveau ajouté (AND) avec la couleur de gouache désirée, pour produire le nouveau masque. Le visage change de couleur pour prendre celle de la voiture, mais reprend très vite la sienne.

Le graphisme de Mugsy est certainement impressionnant, mais le jeu lui-même devient vite lassant. L'action est répétitive et le joueur apprend vite à apprécier les facteurs nécessaires à sa « survie ». Néanmoins, Mugsy donne aux programmeurs aspirants un bon exemple de la manière de tasser du graphisme haute-résolution dans peu d'espace mémoire.

Mugsy : pour le Spectrum 48 K.
Éditeurs : Melbourne House.
Auteurs : Philip Mitchell, Crag Cull, Clive Barrett, Russel Comte.
Manettes de jeu : non nécessaires.
Format : cassette.