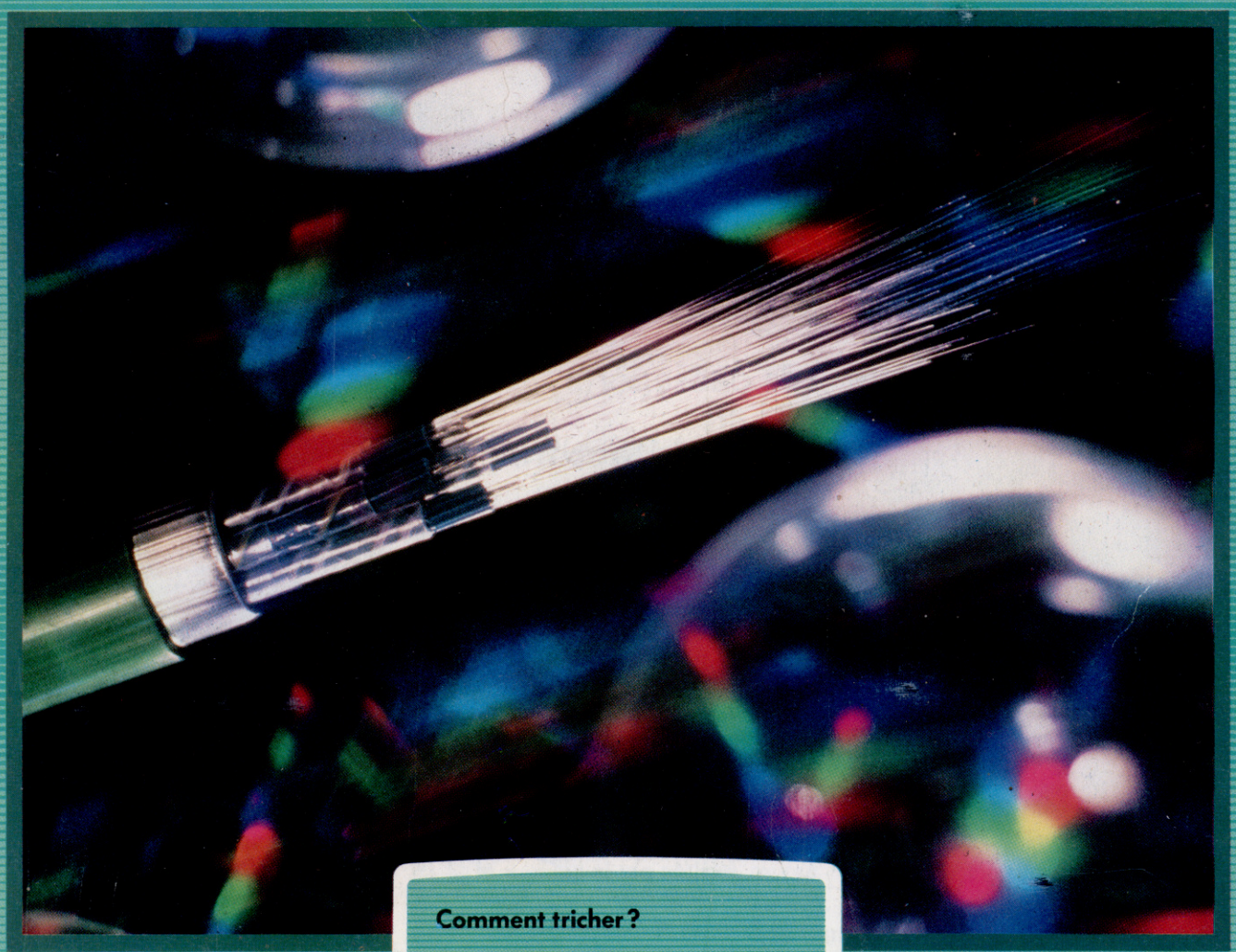
ABC INFORMATIQUE

N° 19

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Comment tricher ?

Logiciels de simulation

Imprimante couleur

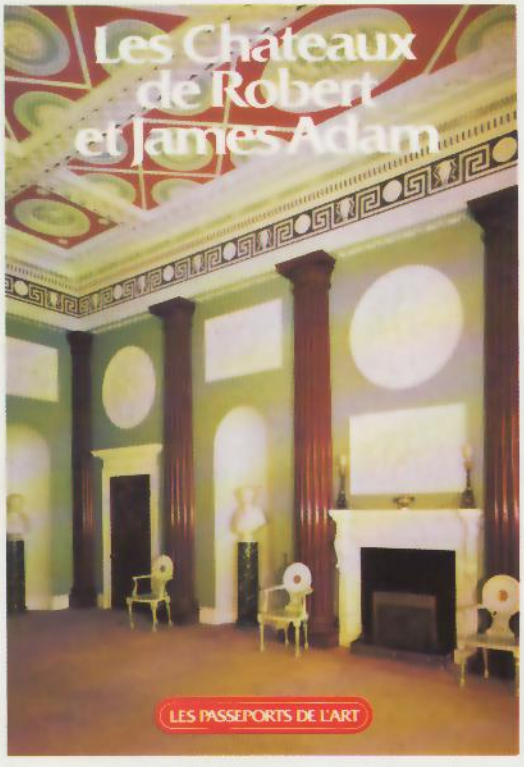
Electron de Acorn

EDITIONS ATLAS

M 6062-19-12 F 85 FB - 3,80 FS - $ 1.95

EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS

Dans toutes les librairies



Les passeports de l'art

Les châteaux de Robert et James Adam

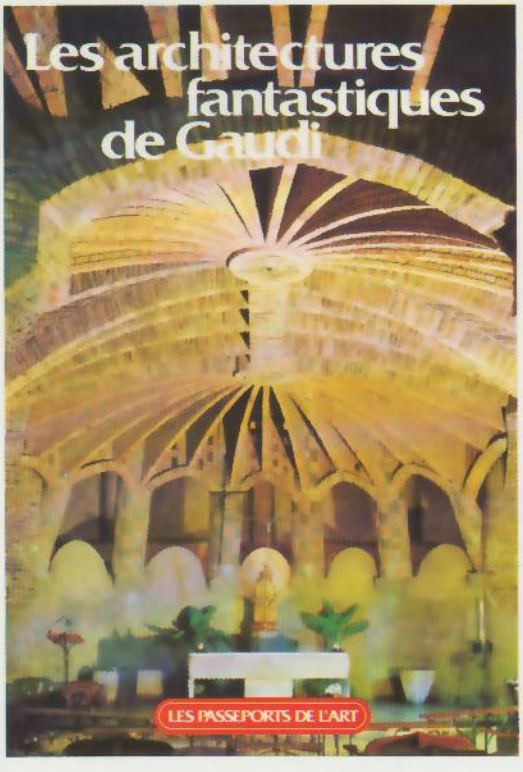
Architectes de l'aristocratie et de la grande bourgeoisie britanniques, Robert et James Adam exercèrent leur activité au XVIIIe siècle.
Ils se rendient célèbres par une interprétation toute personnelle du style néoclassique (influencé par les découvertes archéologiques gréco-romaines), qui fit en Angleterre l'objet d'un vif engouement.

Un volume relié.
Couverture cartonnée. 76 pages.
52 photos en couleurs.
Format : 17 × 24 cm.

EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS

EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS

Dans toutes les librairies



Les passeports de l'art

Les architectures fantastiques de Gaudi

Les réalisations d'Antoine Gaudí, cet architecte catalan de la fin du XIXe siècle, sont toutes guidées par la fantaisie.
A Barcelone, on peut admirer certaines de ses constructions fantastiques, comme la Casa Mila, la Casa Vicens et, surtout, la cathédrale de la Sagrada Familia.
Ce livre invite à découvrir une ornementation inspirée de la nature, exaltée par la couleur et mise en valeur par l'imagination d'un architecte visionnaire.

Un volume relié.
Couverture cartonnée. 76 pages.
59 photos en couleurs.
Format : 17 × 24 cm.

EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS EDITIONS ATLAS

aBc INFORMATIQUE
cours d'informatique pratique et familiale
EDITIONS ATLAS

Édité par ÉDITIONS ATLAS s.a., tour Maine-Montparnasse, 33, avenue du Maine, 75755 Paris *Cedex 15*. Tél. : 538-52-70.

Belgique : ÉDITIONS ATLEN s.a., Bruxelles.

Canada : ÉDITIONS ATLAS CANADA Ltée, Montréal Nord.

Suisse : FINABUCH s.a., ÉDITIONS TRANSALPINES, Mezzovico.

Réalisé par EDENA s.a., 29, boulevard Edgar-Quinet, 75014 Paris. Tél. : 320-15-01.

Direction éditoriale : J.-Fr. Gautier. *Service technique et artistique* : F. Givone et J.-Cl. Bernar. *Iconographie* : J. Pierre. *Correction* : B. Noël.
Publicité : Anne Cayla. Tél. : 202-09-80.

VENTE

Les numéros parus peuvent être obtenus chez les marchands de journaux ou, à défaut, chez les éditeurs, au prix en vigueur au moment de la commande. Ils resteront en principe disponibles pendant six mois après la parution du dernier fascicule de la série. (Pour toute commande par lettre, joindre à votre courrier le règlement, majoré de 10 % de frais de port.)

Pour la France, s'adresser à ÉDITIONS ATLAS, tour Maine-Montparnasse, 33, avenue du Maine, 75755 Paris *Cedex 15*. Tél. : 538-52-70.

Pour les autres pays, s'adresser aux éditeurs indiqués ci-dessous.

SOUSCRIPTION

Les lecteurs désirant souscrire à l'ensemble de cet ouvrage peuvent s'adresser à :

France : DIFFUSION ATLAS, 3, rue de la Taye, 28110 Lucé. Tél. : (37) 35-40-23.

Belgique : ÉDITIONS ATLEN s.a., 55, avenue Huart-Hamoir, 1030 Bruxelles. Tél. : (02) 242-39-00. Banque Bruxelles-Lambert, compte n° 310-0018465-24 Bruxelles.

Canada : ÉDITIONS ATLAS CANADA Ltée, 11450 boulevard Albert-Hudon, Montréal Nord, H 1G 3J9.

Suisse : FINABUCH s.a., ÉDITIONS TRANSALPINES, zona industriale 6849 Mezzovico-Lugano. Tél. : (091) 95-27-44.

RELIEZ VOS FASCICULES

Des reliures mobiles, permettant de relier 12 fascicules, seront en vente en permanence chez votre marchand de journaux.

ATTENTION : ces reliures, présentées sans numérotation, sont valables indifféremment pour tous les volumes de votre collection. Vous les numéroterez vous-même à l'aide du décalque qui est fourni (avec les instructions nécessaires) dans chaque reliure.

En vente tous les vendredis. Volume II, n° 19.

ABC INFORMATIQUE est réalisé avec la collaboration de Trystan Mordrel *(secrétariat de rédaction)*, J.-P. Bourcier *(coordination)*, S.I.-André Larochelle *(traduction)*, Ghislaine Goullier *(fabrication)*, Marie-Claire Jacquet *(iconographie)*, Patrick Boman *(correction)*.
Crédit photographique, couverture : Photo CNET-Lannion.

Directeur de la publication : Paul Bernabeu. Imprimé en Italie par I.G.D.A., Officine Grafiche, Novara. Distribution en France : N.M.P.P. Tax. Dépôt légal : mai 1984. 11845. Dépôt légal en Belgique : D/84/2783/27.

© Orbis Publishing Ltd., London.
© Éditions Atlas, Paris, 1984.

A NOS LECTEURS

En achetant chaque semaine votre fascicule chez le même marchand de journaux, vous serez certain d'être immédiatement servi, en nous facilitant la précision de la distribution. Nous vous en remercions d'avance.

Les Éditions Atlas

Comment tricher?

Les programmes de jeux d'échecs sont difficiles à écrire. Mais il est possible pour les débutants de construire un jeu simple et « intelligent ».

Main invisible
Les machines préprogrammées de jeu d'échecs contiennent les mêmes composants que les ordinateurs personnels : une unité centrale, une RAM et un programme en ROM. Seules les entrées/sorties diffèrent. La machine ci-contre utilise un servomécanisme et un système magnétique pour permettre à l'ordinateur de bouger les pions. Quand, par exemple, un cavalier saute une autre pièce, un algorithme compliqué est employé pour enlever l'obstacle et le remplacer par la suite.
(Cl. Ian McKinnell/ Milton Bradley.)

Quand les amateurs commencent à écrire leurs propres programmes, beaucoup rêvent du jour où ils seront capables de programmer un jeu d'échecs. Cela ne manque pas, au contraire. Les programmes pour ordinateurs domestiques ou les machines préprogrammées pour le jeu d'échecs abondent. Mais l'écriture d'un jeu d'échecs tourne parfois à l'obsession, même parmi les programmeurs peu familiarisés avec les échecs. Pour expliquer ce phénomène, il faut peut-être considérer l'image très intellectuelle qu'a ce jeu, ou encore l'idée selon laquelle un ordinateur qui joue aux échecs est une machine intelligente. Cela dit, il est très difficile de vouloir expliquer comment écrire un programme entier de jeu d'échecs. Toutefois, nous pouvons donner quelques principes sur lesquels des jeux dits « intelligents » mis sur ordinateur sont construits, et jusqu'à quel niveau il vous est possible d'écrire un programme intéressant en BASIC.

Il convient de ne pas perdre de vue que ces jeux n'ont que peu de chose à voir avec les jeux d'arcades, d'aventures ou de simulations, qui tous demandent des techniques différentes de programmation et de l'imagination dans les procédures d'adresse. Nous commencerons cette analyse des jeux « intelligents » avec un exemple que d'aucuns trouveront un peu simpliste, mais qui a l'avantage de recouvrir la plupart des principes d'écriture de ces jeux.

Beaucoup de petits (et de grands) enfants connaissent le jeu « ciseaux-papier-pierre ». Les règles en sont simples : tous les joueurs pensent à l'un de ces trois objets, puis doivent présenter simultanément la main dans la forme de l'un de ces trois objets. Le gagnant est déterminé ainsi : les ciseaux gagnent sur le papier (ils coupent), le papier sur la pierre (il couvre), la pierre sur les ciseaux (ils sont cassés).

Pour tous ceux qui ont suivi le cours de programmation BASIC, cela peut paraître facile d'écrire un programme pour faire jouer ainsi l'ordinateur. La fonction RND est utilisée pour choisir un des éléments du tableau contenant « CISEAUX », « PAPIER » et « PIERRE ». L'élément choisi est imprimé quand on appuie sur la barre d'espace. Le joueur tape selon son choix (le programme dépend de son honnêteté), et le programme détermine le gagnant en comptant les

points marqués par l'ordinateur et son adversaire. Si la fonction RND est vraiment aléatoire, les totaux des points marqués de part et d'autre devraient s'équilibrer dans l'ensemble, quelle que soit la stratégie adoptée par le joueur. Or, il faut maintenant déterminer comment améliorer la stratégie de l'ordinateur pour assurer sa victoire.

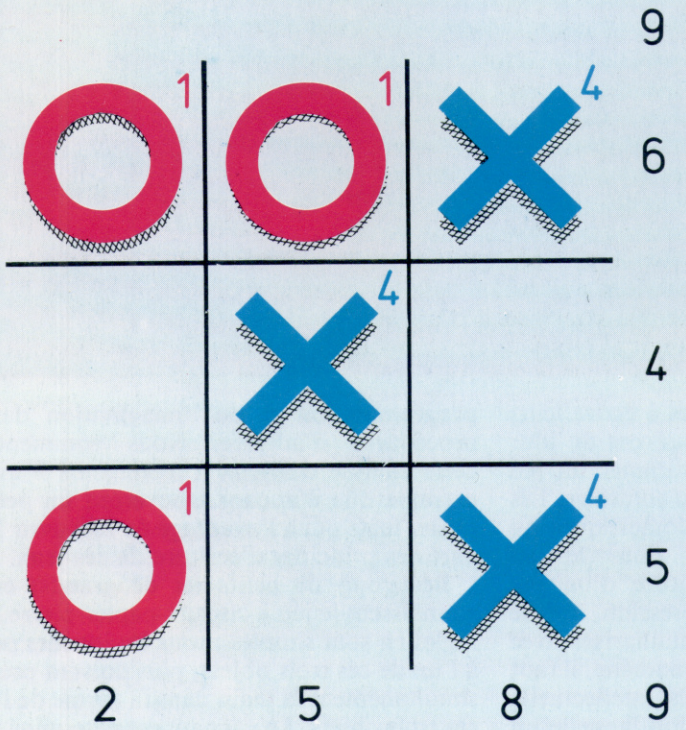
Lorsque nous avons étudié les fonctions aléatoires, nous avons appris que générer une séquence réellement aléatoire est une tâche impossible pour les êtres humains et pour les ordinateurs, bien que ces derniers arrivent à faire une bien meilleure approximation. Dans une longue série de coups, le joueur humain choisit invariablement un objet plus souvent que les autres. On peut donc inscrire dans le programme un sous-programme qui garde en mémoire les choix du joueur, en utilisant un tableau à trois éléments appelés CHOIX(1) CHOIX(2) et CHOIX(3). Chaque fois que le joueur joue un coup, on augmente d'un point le total dans la colonne correspondant à son choix. L'ordinateur peut alors calculer quel objet est le plus souvent joué, et ensuite jouer l'objet qui l'emporte sur celui-ci.

Trois problèmes se posent avec cette méthode. Premièrement, si l'ordinateur joue régulièrement le même objet, le joueur va vite s'en apercevoir et en profiter. Donc il faut faire en sorte que l'ordinateur choisisse entre les trois objets en utilisant la fonction RND, mais avec l'adjonction d'un sous-programme pour lui faire choisir un peu plus souvent l'objet qui l'emportera sur le choix préféré du joueur.

Le deuxième problème est qu'au cours de la partie le joueur tend à changer son objet préféré. Aussi, plutôt que de garder en mémoire tous les choix de l'adversaire depuis le début de la partie, il vaut mieux que le programme n'enregistre que les vingt derniers coups par exemple. Cela nécessitera un tableau CHOIX de 20 × 3 éléments et un sous-programme plus sophistiqué pour faire les totaux et préciser le meilleur choix pour le prochain coup.

Le plus grave inconvénient de cet algorithme apparaît au moment où le joueur réussit à percer la stratégie de l'ordinateur. Il lui est alors facile de jouer de telle façon que l'ordinateur perdra fatalement plus de la moitié des coups joués. Le joueur pourrait, par exemple, jouer régulièrement le même objet, puis subitement se mettre à en jouer un autre, et ainsi de suite. Nous avons donc besoin d'un autre algorithme qui évite ces problèmes. Néanmoins, cela vaudrait la peine de développer des programmes capables d'utiliser à la fois la méthode totalement aléatoire et la méthode aléatoire pondérée, puis d'observer les résultats.

Les êtres humains sont incapables de prendre une décision totalement irrationnelle ou aléatoire. Il s'ensuit que chaque choix dépend des choix précédents. Si l'ordinateur réussit à élaborer une approximation sur ces choix, alors il devrait pouvoir gagner assez régulièrement. Étant donné que chaque joueur possède une formule personnelle inconsciente, et qu'elle change sans doute au cours d'une longue partie, le programme doit être écrit de telle façon qu'il puisse interpréter la formule pendant le déroulement de la partie. Les programmes qui sont capables d'apprendre de cette façon s'appellent des programmes « heuristiques ».



Position gagnante

Une évaluation correcte des positions est fondamentale à tout programme de jeu même si ce jeu est aussi simple que le morpion. Sur ce tableau 3 × 3 les zéros du joueur sont représentés par le chiffre 1, les croix de l'ordinateur par le chiffre 4. A l'aide de ces chiffres, on peut évaluer n'importe quelle situation de jeu; il suffit de faire les totaux pour chaque rangée, colonne et diagonale. Un total de 12 indique que l'ordinateur peut donc gagner, et ainsi de suite. Les valeurs 1 et 4 sont utilisées parce qu'elles fournissent des totaux différents pour chaque combinaison de coups. (Cl. Liz Dixon.)

Un programme heuristique permet à l'ordinateur de détecter des changements dans la stratégie de son adversaire et de modifier son algorithme en conséquence. Un tel programme est doté de la possibilité de stocker dans sa mémoire, disons les cinquante derniers coups joués par les deux adversaires. Il balaie sans cesse cet enregistrement des coups précédents, appliquant une technique statistique appelée « corrélation ».

L'ordinateur doit faire des centaines de comparaisons entre un choix du joueur et les choix précédents. L'ordinateur effectue la même opération sur ses propres choix. Considérons maintenant la corrélation entre le coup du joueur et son coup précédent. D'abord nous devons composer un tableau de 3 × 3 que nous appelons CORR1, parce qu'il constitue notre premier test de corrélation. Ensuite nous devons examiner l'historique de la partie, sur les cinquante derniers coups, en donnant le numéro 1 à « ciseaux », 2 à « papier » et 3 à « pierre ». Chaque fois que le joueur a choisi pierre (3) après ciseaux (1), nous ajoutons un point à l'élément CORR1(1,3) ; lorsque pierre (3) est suivi de papier (2), on ajoute un point à CORR1(3,2) et ainsi de suite.

Si le joueur fait des choix réellement aléatoires, alors chaque élément de CORR1 devrait contenir à peu près le même nombre de points — mais il est très peu probable que ce soit le cas. Donc, si le dernier choix du joueur était papier, alors l'élément de la rangée 2 (papier) de CORR1 qui a le total le plus fort nous indiquera la meilleure probabilité pour son prochain choix. Plus la différence est grande entre les éléments d'une rangée, meilleure est la corrélation et plus sûre la prédiction. Il est possible cependant que l'on trouve peu de corrélation entre le choix du joueur et son choix précédent, auquel cas il faudra effectuer des calculs de corrélation avant l'avant-dernier choix, ou entre le choix du joueur et le choix précédent de l'ordinateur.

Il se peut que les divers programmes de corrélation prédisent tous des probabilités différentes pour le prochain coup du joueur. Le programme doit alors décider quel est le conseil le plus sûr. Pour ce jeu simple, il lui suffit de voir lequel des tests donne la corrélation la plus forte. Par exemple, le tableau CORR1 pourrait donner les possibilités suivantes : ciseaux 51 %, papier 29 %, pierre 20 %, tandis que CORR2 (qui compare le choix du joueur avec le dernier coup de l'ordinateur) pourrait donner : ciseaux 24 %, papier 60 %, pierre 16 %. Il est évident que CORR2 a une meilleure corrélation; sa prédiction doit être retenue. Un programme de jeu intelligent comportera souvent plusieurs sous-programmes, chacun ayant une stratégie différente pour analyser le jeu et conseiller le programme principal sur le meilleur coup à jouer. Le programme de jeu peut considérer ces sous-programmes comme une sorte de « commission » et suivre les conseils de la majorité. Mais, au cours de la partie, il peut décerner des points aux sous-programmes qui donnent de « bons » conseils pour identifier les plus fiables.

```
 5 CLS
10 DIM C1(3,3), C2(3,3), C3(3,3)
20 CR = 0
30 FOR I = 1 TO 3
40 IF C1(PL,I) >CR THEN BG = I:CR = C1(PL,I)
50 IF C2(PP,I) >CR THEN BG = I:CR = C2(P2,I)
60 IF C3(P3,I) >CR THEN BG = I:CR = C3(P3,I)
70 NEXT I
80 CT = BG-1
90 IF BG = 1 THEN CT = 3
100 GET PT:IF PT = 0 THEN 100
110 REM LINE 100 WAITS FOR A DIGIT TO
120 REM BE PRESSED
130 IF CT = PT-1 THEN CS = CS + 1
140 IF CT = PT-2 THEN PS = PS + 1
150 IF CT = PT + 1 THEN PS = PS + 1
160 IF CT = PT + 2 THEN CS = CS + 1
170 CLS
180 PRINT « YOUR CHOICE : »; PT
190 PRINT « MY CHOICE : »; CT
200 PRINT « YOUR SCORE IS »; PS
210 PRINT « MY SCORE IS »; CS
220 C1(PL,PT) = C1(PL,PT) + 1
230 C2(PP,PT) = C2(PP,PT) + 1
240 C3(P3,PT) = C3(P3,PT) + 1
250 P3 = PP
260 PP = PL
270 PL = PT
280 GOTO 20
```

Apprenti sorcier
Ce programme conçu pour le jeu ciseaux-papier-pierre montre comment on peut « apprendre » au cours d'une partie. L'ordinateur sélectionne parmi les chiffres 1, 2 et 3, compare son choix avec celui que vous venez de taper et compte un point au gagnant. L'instruction GET est utilisée pour vous permettre de frapper les trois touches numériques très rapidement. Si vous jouez vraiment au hasard, vous devez vous apercevoir qu'au bout de quelque 200 coups l'ordinateur commence à vous distancer. Il est possible de tromper ce programme et de continuer à gagner, mais des sous-programmes plus sophistiqués peuvent être ajoutés pour vous empêcher de réussir.

S'il s'avère qu'il y a effectivement une corrélation entre les coups ou choix du joueur et les coups précédents de l'ordinateur, il devient possible de programmer un facteur de « bluff » pour tromper le joueur, comme dans les jeux d'argent.

Le journal *Scientific American* a rendu compte d'une expérience menée à l'université d'État de New York à Buffalo. Plusieurs programmes spécialisés dans le poker (tous dotés d'une capacité d'apprentissage) furent opposés les uns aux autres dans plusieurs milliers de parties. Le gagnant était un programme appelé Adaptive Evaluator of Opponents (Évaluation adaptée de l'adversaire), qui effectuait une première évaluation de la force de jeu de ses adversaires, puis modifiait cette évaluation au cours de la partie. Le programme Sells and Buys Imags (Vend et achète des images) n'a obtenu que des résultats médiocres : sa technique était de « bluffer » pour « vendre » une image fausse à ses adversaires, ou bien d'« acheter » en quelque sorte le style de jeu des autres.

Le Bayesian Player (Joueur bayésien) utilisait des méthodes inductives pour améliorer son jeu en comparant les conséquences réelles. Enfin, le programme appelé Adaptative Aspiration Level (Niveau d'aspiration évolutif) tentait d'imiter un trait que l'on croit avoir observé dans le jeu humain : l'adaptation du niveau d'aspiration (c'est-à-dire le degré de risque que l'on accepte de courir) selon les résultats passés et la situation actuelle.

Topographie

Les langages évolués, tel le BASIC, gèrent automatiquement la mémoire. Quand ce n'est pas le cas, il nous faut un relevé détaillé de la mémoire afin de nous y retrouver.

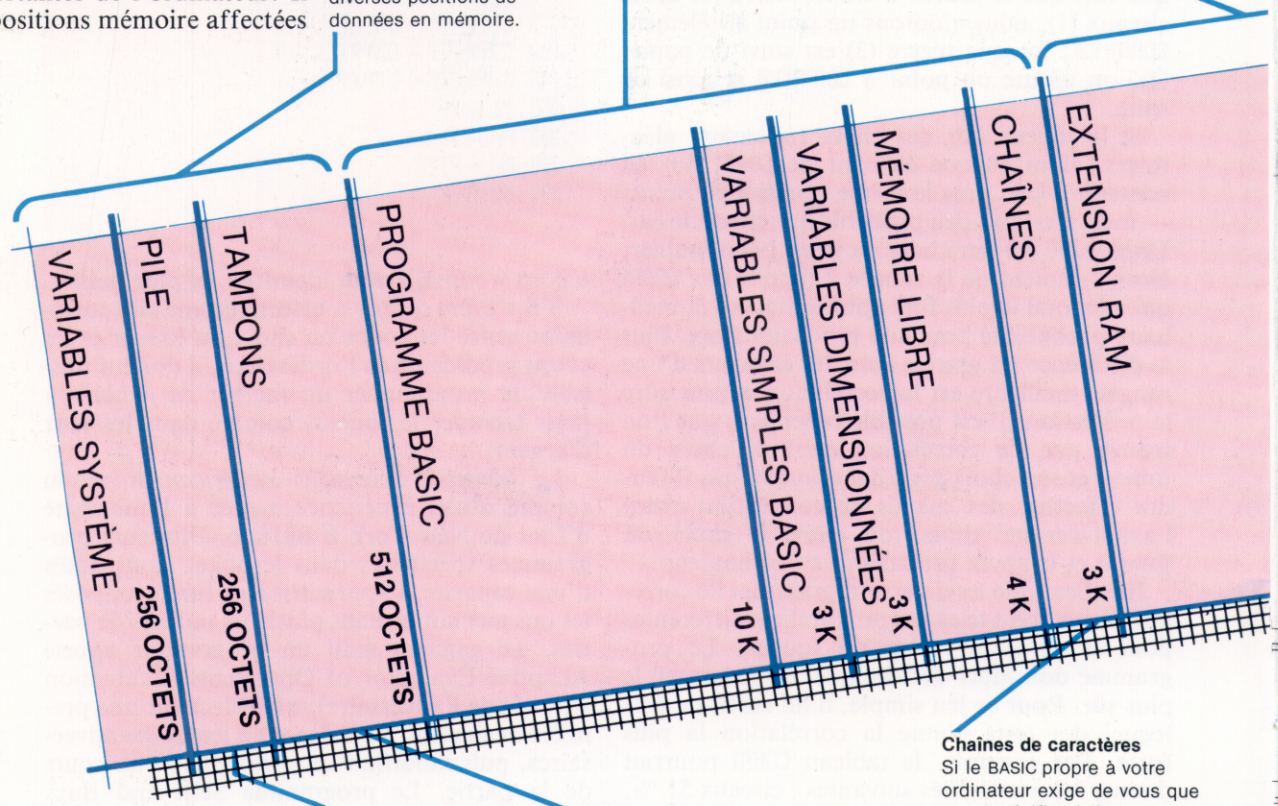
L'unité centrale de l'ordinateur possède une capacité d'adressage dont la taille détermine le nombre de positions mémoire possibles. Pour la plupart des micro-ordinateurs, cette capacité mémoire est de 64 K. La mémoire doit comporter la mémoire vive et la mémoire morte propres à la machine (RAM et ROM), leurs extensions éventuelles ainsi que l'adressage des composants et des ports d'interface. Ces derniers étant considérés également comme positions mémoire. La table mémoire constitue une des parties les plus importantes de l'ordinateur. Il s'agit de la liste des positions mémoire affectées aux différentes fonctions et représentées par un diagramme. Si vous programmez seulement en BASIC, cela ne vous intéresse pas. En revanche, si vous souhaitez travailler en langage machine ou si vous désirez construire vous-même des extensions supplémentaires, cela est essentiel.

Nous exposons ici le contenu caractéristique d'une table mémoire. Notre exemple se rapproche davantage d'un système fondé sur un 6502 que d'un Z80, mais les grandes lignes sont les mêmes. Les constructeurs ne communiquent pas tous la conception de leur table mémoire, mais des utilisateurs le font souvent à leur place.

Mémoire système
Un micro-ordinateur de 4 K de capacité RAM ne comportera le plus souvent que 3 K de mémoire utilisateur (programme). La différence ou réserve machine sert au système d'exploitation lorsque la machine est allumée. Elle est utilisée pour stocker des variables système (telles que les valeurs provisoires lors des calculs d'expression complexes) et aux pointeurs ou chapeaux indiquant les diverses positions de données en mémoire.

RAM utilisateur
Sa taille détermine le niveau de complexité possible des programmes que vous pouvez utiliser. C'est un des critères fondamentaux pour le choix d'un micro-ordinateur.

Mémoire disponible
La table mémoire doit comporter de la place pour les extensions mémoire vive. Certains systèmes autorisent 64 K d'ajout, mais il s'agit dans ce cas d'un dispositif d'acquisition de données qui intègre à volonté la partie désirée de la mémoire vive, à la table mémoire.



Pile
Cette portion de la mémoire est réservée à l'unité centrale et est organisée selon le principe LIFO : dernière information entrée, première sortie. Un octet peut être soit empilé soit dépilé. Lors de l'exécution d'une routine GOSUB en BASIC par exemple, l'UC empile l'adresse mémoire de destination finale (RETURN). La pile est très utilisée pour l'évaluation d'expressions arithmétiques, et pour des boucles de type FOR...NEXT.

Tampons
Un emplacement mémoire doit être prévu pour préserver les caractères frappés au clavier que le programme ne peut traiter immédiatement. Un tampon cassette est également nécessaire car la plupart des systèmes d'exploitation écrivent les données par blocs entiers avant de les transmettre.

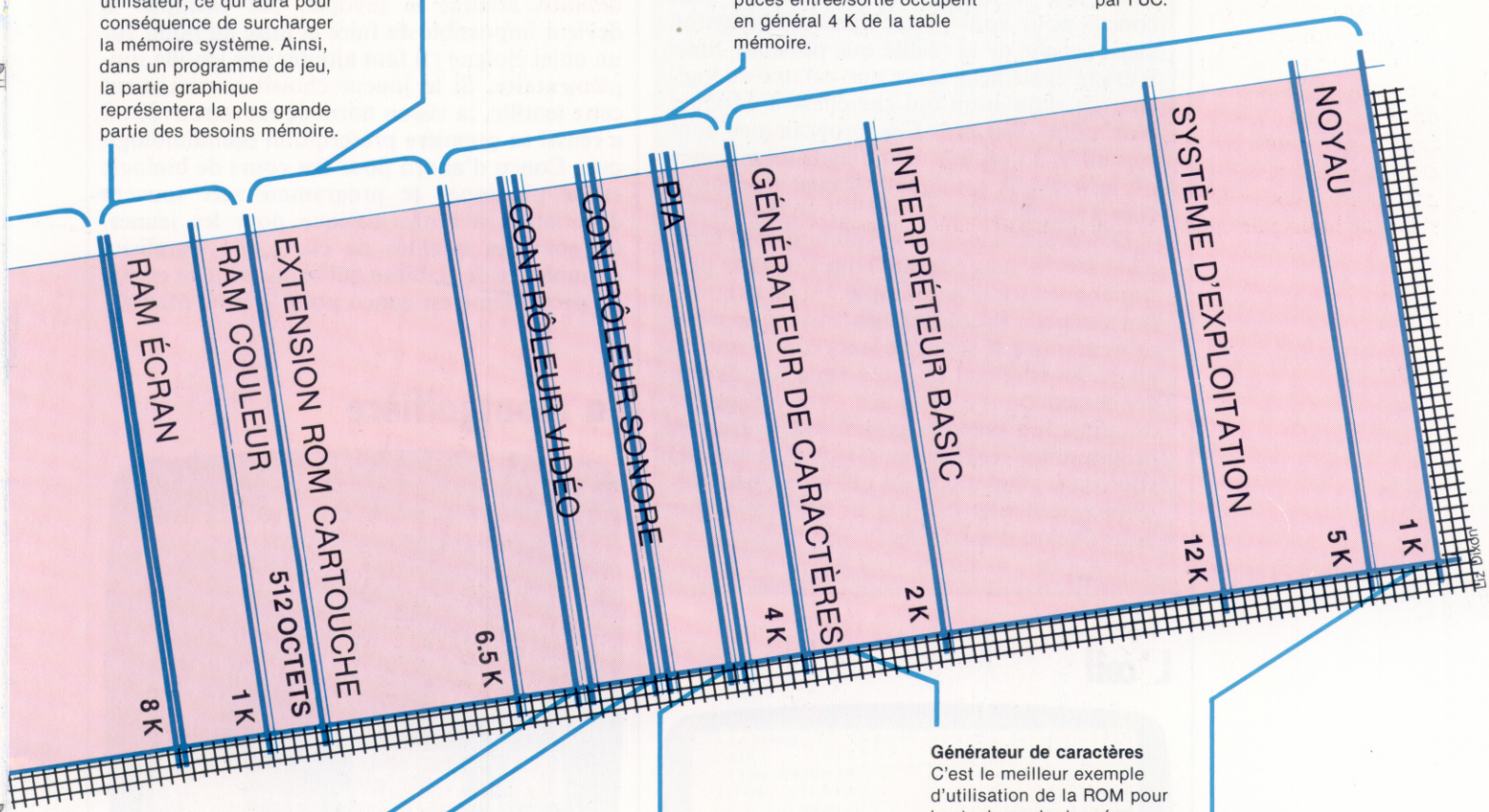
Chaînes de caractères
Si le BASIC propre à votre ordinateur exige de vous que vous spécifiez à l'avance la longueur de toutes les chaînes, elles seront stockées dans une table mémoire comme des variables dimensionnées. Si au contraire il accepte des chaînes de type dynamique, susceptibles de varier en longueur, alors les données seront stockées séparément en une zone mémoire changeant constamment de taille. Régulièrement, le système d'exploitation « nettoiera » la zone mémoire de la pile en supprimant les données devenues inutiles.

RAM système
Certains ordinateurs comportent une RAM système distincte de la RAM utilisateur. Elle est généralement destinée à la RAM écran (pour laquelle chaque octet correspond à une position sur l'écran) et à la RAM couleur (pour laquelle un octet donne la couleur d'écriture et la couleur de fond). Les ordinateurs dotés de capacités graphiques multiples et à haute résolution nécessiteront beaucoup de RAM utilisateur, ce qui aura pour conséquence de surcharger la mémoire système. Ainsi, dans un programme de jeu, la partie graphique représentera la plus grande partie des besoins mémoire.

Mémoire disponible
Lorsque vous utilisez un programme en cartouche, il vient figurer dans la table mémoire sous la forme d'une extension ROM. Certaines machines disposent d'emplacements sur leurs cartes électroniques destinés à recevoir en mémoire morte des langages supplémentaires. Ces derniers seront également chargés dans la table mémoire.

Puces entrée/sortie
L'UC ne peut communiquer qu'avec des organes apparaissant dans la table mémoire. Ainsi, les ports d'interface et les autres composants doivent-ils y figurer. On trouvera ainsi les interfaces pour le clavier, le lecteur de cassettes, le contrôleur vidéo, et les interfaces externes telles que l'imprimante. L'UC adresse généralement la mémoire sous forme de blocs (par exemple de 4 K chacun). C'est pourquoi les puces entrée/sortie occupent en général 4 K de la table mémoire.

ROM système
Dans un ordinateur individuel, la ROM sert à stocker l'information de base. C'est-à-dire, avant tout, le système d'exploitation constitué de l'ensemble des programmes en code machine. Ces programmes effectuent des fonctions telles que le balayage du clavier et le stockage ou la restitution des données sur cassette. Il y a également l'interpréteur BASIC, qui traduit les programmes BASIC en instructions de base compréhensibles par l'UC.



Contrôleur vidéo
Les modes graphiques les plus sophistiqués tels que les jeux faisant intervenir la résolution graphique multimodes sont de plus en plus gérés sur le plan matériel et non logiciel. La table mémoire comportera ainsi une douzaine de registres environ à un octet chacun, qui régiront tous les éléments visuels, depuis la couleur du fond jusqu'à la position exacte de chaque plan-objet.

Contrôleur son
Le logiciel peut permettre d'obtenir des sons rudimentaires, mais les sons complexes et le contrôle du son nécessitent un contrôleur son spécifique, dont la sortie attaque un petit amplificateur.

PIA (adaptateurs d'interface périphérique)
Ils servent à gérer la plupart des interfaçages simples avec le clavier, la cassette, la manette de jeux et l'imprimante. Les puces les plus évoluées (tel que l'adaptateur d'interface multifonctions 6522) sont capables de faire des conversions entre données en parallèle et en série. Ils comportent en outre des horloges internes utilisables pour la programmation ou pour le contrôle de la vitesse de transmission des données.

Générateur de caractères
C'est le meilleur exemple d'utilisation de la ROM pour le stockage de données plutôt que pour les programmes. Dans ce cas, les structures de bits définissent la façon dont les caractères apparaîtront à l'écran. Certains ordinateurs permettent de copier tout ou partie des données en RAM, ce qui laisse la possibilité de définir de nouveaux caractères.

Programme noyau
Le programme noyau est le cœur du système d'exploitation. C'est le programme systématiquement exécuté lors du démarrage de l'ordinateur. Il comptabilise la RAM disponible et vérifie si la cartouche programme est présente. Le programme noyau gère également les formes les plus simples d'entrée/sortie.

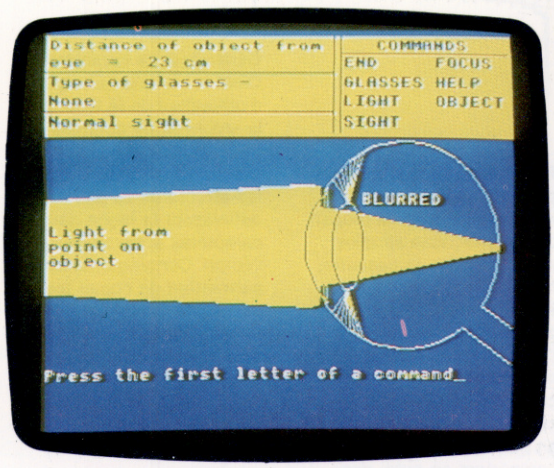
Faire semblant

Les logiciels de simulation permettent de faire des expériences scientifiques sans appareils, sans spécimens et sans matériaux, chez soi ou à l'école.

Les programmes de simulation, tels les jeux qui vous permettent de prendre les commandes d'une formule 1 ou d'un avion, sont conçus pour vous placer dans une situation aussi proche de la réalité que possible. Mais il existe également toute une gamme de logiciels de simulation qui cherchent à éduquer plus qu'à distraire. Ces programmes sont très utiles à l'école, surtout dans les matières où les travaux pratiques seraient trop difficiles à organiser.

Des programmes de simulation sont aussi à la base des jeux éducatifs que l'on peut pratiquer chez soi. Par exemple, un programme qui s'appelle Promenade en voiture apprend aux enfants à se servir de leurs connaissances en arithmétique et de leurs capacités de raisonnement pour « conduire » une voiture sur un itinéraire entre plusieurs villes. De tels programmes représentent le type de logiciel éducatif le plus stimulant qui se fait actuellement; malheureusement, ils n'existent que pour quelques machines.

L'œil



Ce programme explique le fonctionnement de l'œil, en montrant que, pour qu'on voie clair, tous les éléments de l'organe doivent être bien réglés. Il simule le passage d'un rayon de lumière à partir d'un objet jusqu'à la rétine (le fond de l'œil, où se forment les images). Le joueur tient le rôle du cerveau et agit sur des paramètres tels que l'éloignement de l'objet, l'ouverture de la pupille et la longueur focale de la lentille, afin d'obtenir une image nette sur la rétine. Sur l'écran l'œil est représenté en éclaté, les éléments fonctionnels étant repérés par des étiquettes. Au moyen de la commande LUMIÈRE, la trajectoire d'un rayon de lumière peut être matérialisée, de l'objet jusqu'à l'œil. Si toutes les variables sont bien réglées, on voit apparaître le message « AU POINT ». Sinon, l'ordinateur signale « BROUILLÉ ».

Une fois maîtrisé le fonctionnement normal de l'œil, on peut passer à la simulation de défauts, comme la myopie, par exemple. Il devient impossible de faire la mise au point sur un objet éloigné : il faut ajouter une lentille supplémentaire. Si le joueur choisit correctement cette lentille, la vision normale est retrouvée : il a réussi sa première prescription ophtalmologique. Conçu d'abord pour des cours de biologie et de physique, ce programme sert souvent d'initiation à l'informatique pour les jeunes. Ce sont ses qualités de clarté, de simplicité d'emploi et de fiabilité qui expliquent ce choix. Le programme est conçu pour le BBC Micro.

La montgolfière

La montgolfière est un programme éducatif pour les enfants âgés de huit à douze ans, produit pour le Spectrum. Le joueur doit piloter une montgolfière. Sur l'écran il voit le terrain en coupe, avec l'image du ballon qui, au départ, repose sur le sol. Des renseignements utiles sont donnés par quatre cadrans qui s'affichent sur l'écran : vitesse de montée ou de descente, température de l'air, altitude, carburant. On dispose simplement de deux commandes : celle du brûleur à gaz qui chauffe l'air pour faire monter le ballon et celle de la soupape qui laisse échapper l'air pour entraîner le ballon vers le bas.

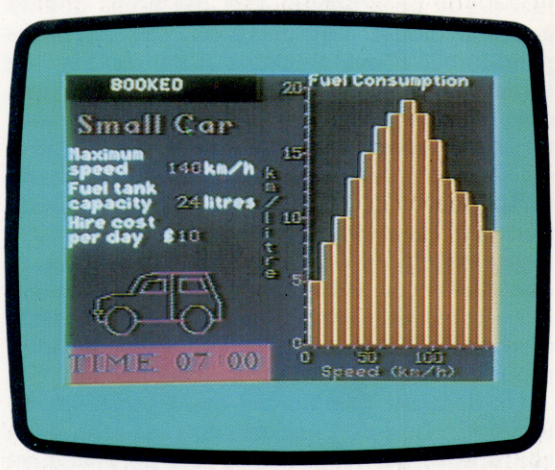
Une « leçon de vol » très simple initie le joueur à l'emploi des commandes et des jauges pour décoller, voler et atterrir. Une fois maîtrisées ces techniques de base, vous êtes prêt à partir pour votre « mission ». Pour cela, il faut

atterrir à une série de points sélectionnés (marqués d'un X) pour recevoir chaque fois des instructions. Par exemple, il s'agira d'aider un berger à retrouver des moutons égarés, en allant jusqu'au champ (marqué sur l'écran) pour les chercher. Si le ballon manque de carburant, il faut atterrir pour charger de nouveaux cylindres de gaz.

Après quelques petits incidents au départ, vous apprenez vite à contrôler le ballon avec précision, au moyen de petits coups de brûleur. Il faut aussi surveiller les cadrans pour savoir à l'avance s'il faut agir sur la soupape ou le brûleur. On apprend surtout à contrôler un système qui ne réagit qu'avec un certain délai.

Ce programme très fidèle à la réalité amuse bien son public, et c'est probablement un des rares sujets de « jeux » qui intéressent autant les filles que les garçons. Peut-être parce que ce programme conjugue réalisme et poésie!

Promenade en voiture

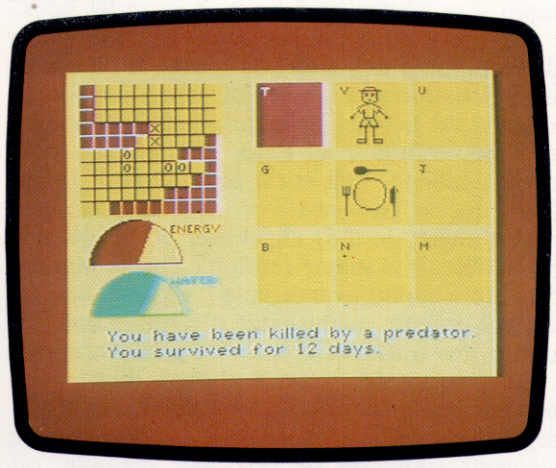


Conçu pour le Spectrum, ce programme éducatif met le joueur dans la situation de l'exploitant d'une petite entreprise de messageries. Il lui faut prendre des décisions concernant le choix des contrats à accepter, la vitesse de croisière à adopter, le type de véhicule à utiliser... Le joueur est amené à faire des calculs sur l'argent, les distances, le temps et même la consommation d'essence. Une carte du pays est visualisée, avec quinze villes et les principales autoroutes. Le compteur de vitesse, le totalisateur kilométrique, la jauge d'essence et une montre figurent également sur l'écran.

La première tâche du joueur est de fixer sa ville de départ, puis de choisir, parmi une douzaine de possibilités offertes, le contrat qui lui semble réalisable et rentable. Par exemple, il peut lui être demandé de chercher un paquet de diamants à Orléans à midi et de le livrer à Calais avant 18 heures le même jour. Pour cela, il faut louer une voiture, la conduire jusqu'à Orléans, aller chercher le paquet, et rouler jusqu'à Calais. S'il réussit, il gagne 4 000 F, plus une prime de 100 F pour une livraison en avance sur l'horaire. Il peut ensuite choisir un autre contrat. Il faut dépenser de l'argent pour les nuits d'hôtel, les réparations, l'essence et les amendes pour excès de vitesse. Si jamais il ne remplit pas correctement le contrat, le forfait est de 1 000 F. S'il accepte un chargement lourd, il lui faut échanger la voiture contre une camionnette qui coûte cher, consomme davantage et roule moins vite.

Ce jeu met en pratique des connaissances en géographie, mais aide surtout à développer la capacité de réflexion logique et de prise de décision. Il enseigne même des rudiments d'économie car en pesant le pour et le contre d'un contrat, le joueur se livre, en fait, à une analyse de rentabilité.

Survie



Ian McKinnell

Vous êtes-vous jamais posé la question de savoir comment vit un lion (ou une souris)? Alors, Survie est un jeu pour vous. Il vous permet de jouer le rôle d'un animal (épervier, rouge-gorge, lion, souris, mouche, papillon), d'être confronté aux problèmes qu'il rencontre dans sa vie de tous les jours, et de comprendre les décisions qu'il doit prendre pour survivre.

Le monde est représenté par une grille sur l'écran. Vous vous déplacez d'une case à l'autre en tapant les coordonnées sur le clavier. Vos soucis principaux sont de rechercher la nourriture (les cases marquées d'un 0), et d'éviter les prédateurs (les cases marquées d'un X). A mesure que vous vous approchez d'une case marquée, un agrandissement à droite de l'écran vous indique la nature exacte de l'ennemi ou de la nourriture que vous allez rencontrer. Également visibles, deux jauges vous montrent vos réserves en énergie et en eau. Si votre niveau d'énergie baisse trop, il vous faut chercher de la nourriture sans tarder; si vous manquez d'eau, il faut vous approcher d'une case bleue (mais sans vous y poser, car vous seriez « noyé »).

Certains animaux ont une vie plus difficile que d'autres : le papillon doit chercher sa nourriture sur des fleurs qui sont parfois difficiles à trouver. L'épervier se nourrit d'escargots, de mouches et de souris, mais doit craindre l'homme. A travers Survie, vous apprendrez la place de chaque espèce dans la chaîne alimentaire, et vous évaluerez les problèmes qui se posent aux animaux dans leur milieu naturel.

Le bon choix

Trouver la meilleure solution d'un problème est parfois chose facile, mais requiert souvent un haut niveau mathématique. Pour l'ordinateur c'est un jeu d'enfant.

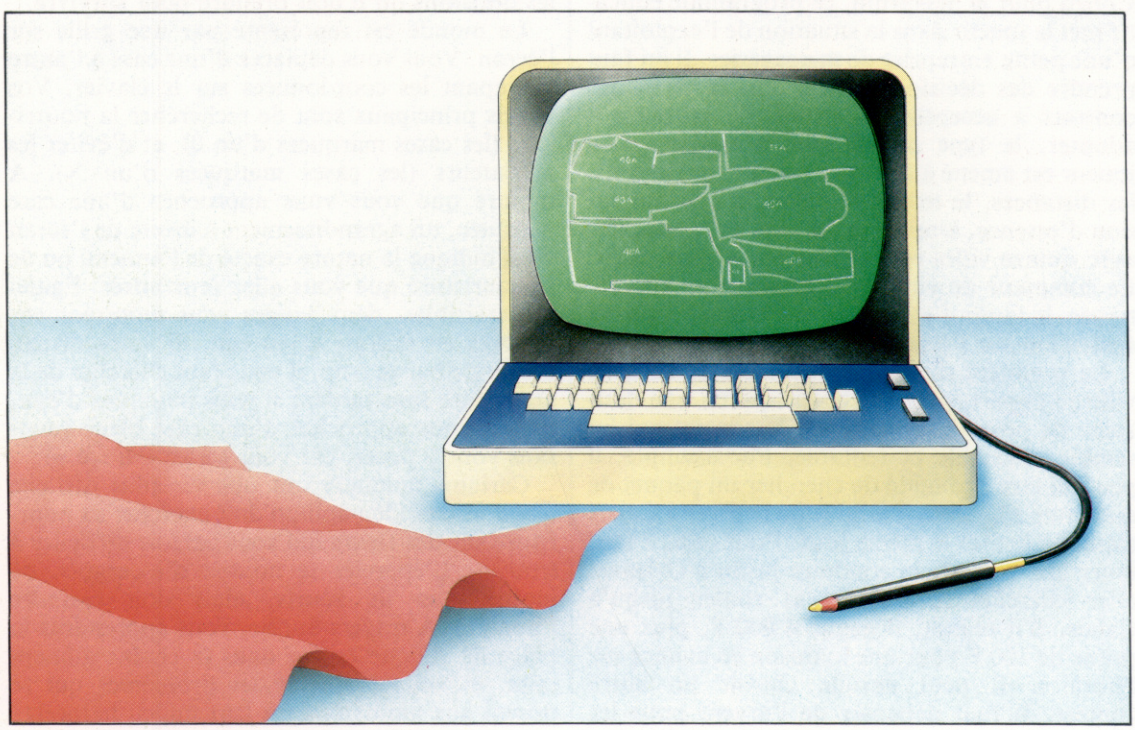
Dans toute décision que nous prenons, il existe inévitablement un compromis — entre prix et qualité, ou temps et argent, par exemple. Il est peu probable d'obtenir dans l'absolu une efficacité maximale pour un coût minimal. L'« optimum » se situera quelque part entre les deux.

Si, par exemple, nous avons le choix entre deux marques de poudre de lavage, le raisonnement sous-jacent à la décision sera à peu près le suivant : « Si j'achète cette poudre de lavage, elle me coûtera 7,50 F les 500 g, mais si je prends celle-là, j'en aurai pour 13,80 F le kg. Mais que se passera-t-il si, avec la poudre la moins chère, je dois en utiliser 20 % de plus pour obtenir le même résultat? Quelle marque sera alors la plus avantageuse? » La réponse est facile à prévoir, même avant d'entamer le calcul qui revient à des différences de pourcentage.

Le concept de « pondération » d'un calcul par une valeur constante est bien connu. Cette méthode marche bien lorsque les écarts entre éléments similaires (le prix, par exemple, ou bien le poids) sont eux-mêmes constants. Mais quand ces écarts eux-mêmes varient, alors les mathématiques se compliquent, et nous devons avoir recours à une forme de calcul (où il faut résoudre un système d'équations comportant les mêmes termes) pour parvenir à la réponse exacte. Si le nombre de termes est petit, nous pouvons les regrouper dans un tableau, afin de les manipuler. On peut aussi essayer de deviner une réponse, puis la modifier successivement jusqu'à ce qu'elle vérifie les conditions. Naturellement, meilleure sera l'estimation moins le temps de calcul durera.

Ces techniques d'optimisation sont essentielles dans le commerce et l'industrie, et sont universellement appliquées, surtout dans la fabrication et la construction. La *programmation linéaire,* l'*analyse du chemin critique* et la *méthode PERT* (technique de recherche d'évaluation de programme) sont des noms donnés à des méthodes d'optimisation. Ces méthodes précédèrent d'une trentaine d'années l'ère de l'informatique. Elles exigeaient d'énormes efforts pour obtenir une réponse correcte dans un délai acceptable. De telles applications conviennent tout à fait aux ordinateurs domestiques, mais il ne faut pas oublier que les opérations sur les matrices (tableaux à deux dimensions) exigent un espace mémoire important, et que le calcul est assez complexe. Heureusement, il existe de nombreux logiciels pour les petits systèmes informatiques, de sorte que cette technique est aisément utilisable.

Un domaine commercial qui a considérablement bénéficié de l'optimisation est celui de la confection. Les vêtements se font normalement



L'ajustement parfait
L'arrangement de patrons sur une feuille de métal ou une pièce de tissu pour minimiser les chutes est un bon exemple d'optimisation par ordinateur. Dans un cas, il s'agit de découper de la tôle, dans l'autre de façonner des vêtements. L'ordinateur affiche sur l'écran la disposition qu'il propose, et un opérateur expérimenté peut ensuite effectuer des ajustements mineurs à l'aide d'un crayon lumineux. (Cl. Kevin Jones/ Burtons Tailoring.)

	PRIX EN FRANCS	PROTÉINES mg	HYDRATE DE CARBONE mg	GRAISSE mg	CALORIES	QUANTITÉS NÉCESSAIRES
400 g PATATES	2	10	90	0	400	8,5 kg
200 g SARDINES	5	80	5	5	500	—
400 g VIANDE	25	150	10	80	1300	—
1/2 LITRE DE LAIT	3	15	25	20	200	3,75 l
MINIMUM NÉCESSAIRE PAR SEMAINE		250	1000	150	10000	

Optimisation d'un régime
Dans cet exemple, il s'agit de trouver la combinaison optimale de quatre aliments, qui satisfasse un besoin diététique spécifique minimal et pour un coût le plus bas possible. Pour cela, nous devons entrer dans l'ordinateur la composition (en rose) et les prix unitaires (en bleu) de chaque aliment, ainsi que les besoins minimaux pour chaque composant alimentaire par semaine (en jaune). Dans cet exemple, les besoins ont été satisfaits rien qu'avec les pommes de terre et le lait, pour un prix minimal de 48 F par semaine.
(N.B. — ce régime n'est pas recommandé.)
(Cl. Kevin Jones.)

dans des tailles standard. Le problème du fabricant consiste à minimiser les chutes de tissu, tout en restant attentif à des facteurs tels que la direction des fils de l'étoffe.

Chez l'un des tailleurs les plus modernes d'Europe, la disposition des pièces du patron sur une longueur donnée de tissu pour la production de prêt-à-porter est effectuée grâce à des techniques d'optimisation, et le résultat proposé est montré sur une console de visualisation. A ce stade, utilisant des méthodes de programmation orientées par l'objet, l'opérateur doit exercer son jugement et son expérience afin de tenter d'améliorer le calcul sur ordinateur. L'opérateur réalise, en moyenne, une amélioration une fois sur cinq.

Les exigences de chaque travail, ou de chaque vêtement, diffèrent. C'est un excellent exemple de l'usage intelligent d'une optimisation sur ordinateur combinée avec l'expérience de l'opérateur. Dans l'industrie, on utilise des méthodes à plus grande échelle pour couper de façon répétitive des objets identiques dans de grandes pièces de métal; tout le processus d'optimisation peut alors suivre son cours. Étant donné que les opérations de coupe et d'impression constituent une partie d'une chaîne de production, elles sont répétées des milliers de fois. Dans ce cas, le coût du processus d'optimisation divisé par le nombre de pièces fabriquées est largement couvert par les économies réalisées sur les chutes.

Comme son nom l'indique, l'analyse du chemin critique sert à déterminer la partie du travail ayant la plus grande probabilité d'entraver le reste, s'il n'est rien prévu de particulier. C'est une méthode fondée sur le temps : la période exigée pour l'exécution d'une tâche devenant élément sur le diagramme d'analyse de chemin critique. Elle est couramment utilisée pendant la phase de planification de projets de construction, de sorte que les entrepreneurs peuvent assigner des hommes et des matériaux aux différentes phases du projet dans le bon ordre — la plomberie avant les parquets, les peintures après les plâtres. Là encore, des logiciels sont disponibles pour une large variété de micro-ordinateurs.

Tandis que les mathématiques des processus d'optimisation peuvent être assez décourageantes pour les non-initiés, on ne peut nier le succès et l'efficacité de la technique elle-même. C'est l'une des rares tâches numériques accomplies communément sur des mini-ordinateurs. Elle constitue une composante importante des systèmes d'intelligence artificielle qui reproduisent le raisonnement commun.

Cauchemar d'autoroutes
Le tracé des autoroutes, en agglomération comme à la campagne, dépend beaucoup des techniques d'optimisation. L'architecte sera surtout concerné par les dénivellations et les courbes. Mais le fermier, qui a été dépossédé de sa terre, n'aura pas les mêmes critères. Lorsqu'une nouvelle route est prévue, on rassemble une grande quantité de données qui servent à établir un modèle d'ensemble de la situation. Ce modèle est ensuite utilisé pour des objectifs variés, qui vont de la représentation graphique jusqu'à l'optimisation du tracé.
(Cl. Ministère des Transports.)

Electron de Acorn

Au cours des deux années qui séparent chez Acorn le BBC Model B de l'Electron, la technologie micro-informatique s'est développée de manière spectaculaire.

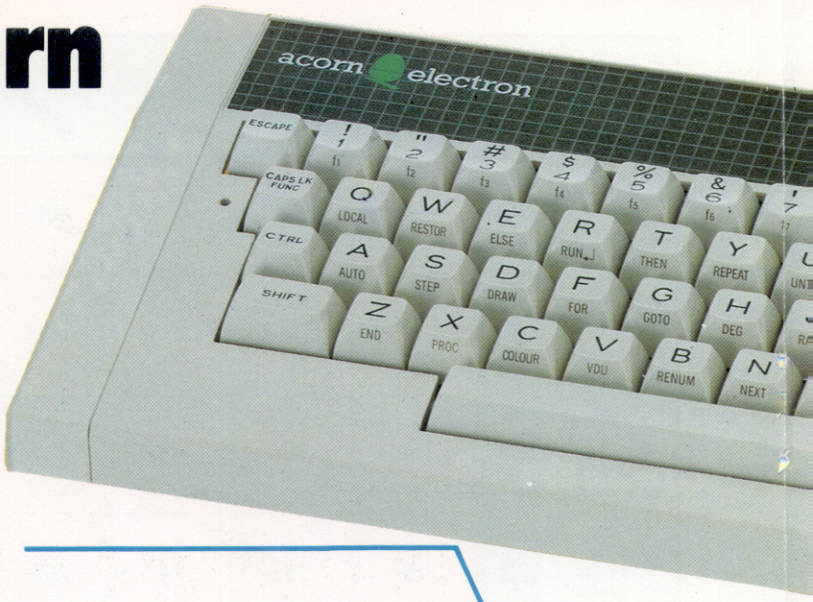
Acorn Electron est un micro d'aspect agréable dont l'usage confirme l'impression initiale de robustesse et de fiabilité. Il s'agit d'un modèle issu du BBC Micro, mais moins performant. En revanche, son utilisation se révèle être plus souple. La plupart des caractéristiques du BBC Micro ont été reprises dans l'Electron. Ainsi la commande SOUND est utilisée conjointement avec ENVELOPPE pour générer le timbre de divers instruments, comme sur le BBC Micro.

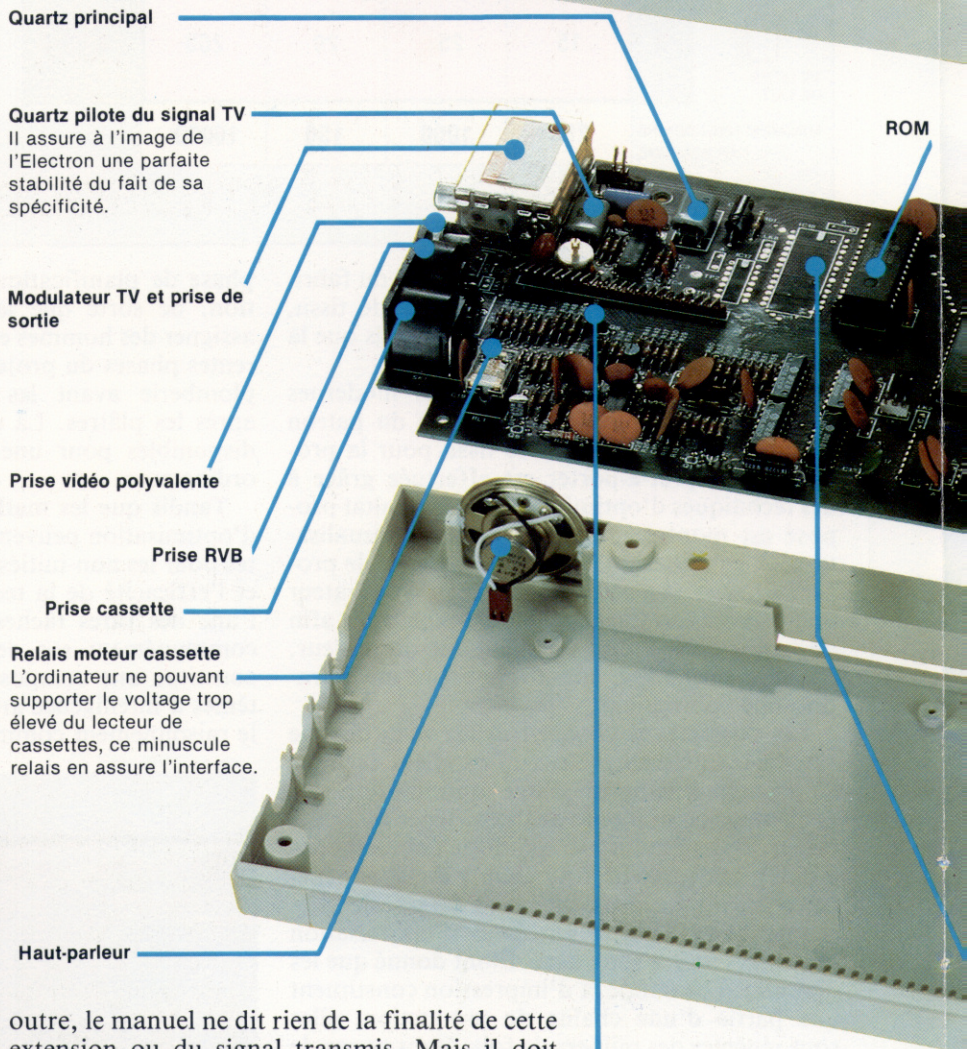
Tous les modes graphiques du BBC existent sur l'Electron, à l'exception de Teletext (MODE 7), généré par un composant propre au BBC. L'affichage de type Teletext sur l'Electron devra se faire en redéfinissant la plupart des caractères et en imitant le mode Teletext par l'intermédiaire du MODE 6 (limité à deux couleurs). Cette lacune est un sévère handicap, car Teletext sur BBC Micro est un mode d'affichage puissant, économe de place en mémoire.

Les entrées-sorties sont également moins performantes. La sortie image se fait *via* le canal 36 de la télévision ainsi que par l'intermédiaire du support vidéo polyvalent et du support RVB (rouge, vert, bleu), vers des moniteurs monochromes ou couleur. En dehors du port cassette, il n'existe pas sur l'Electron d'interface directement utilisable.

Une connexion à l'arrière de la machine permet d'envisager une extension. Malheureusement, cette dernière est mal située — en saillie sous un renfoncement — et mal protégée. En outre, le manuel ne dit rien de la finalité de cette extension ou du signal transmis. Mais il doit s'agir d'un boîtier intermédiaire puisqu'il y a un dispositif de verrouillage mécanique.

Le BASIC résident est le fameux langage BBC, sensiblement étendu et dont de nombreuses caractéristiques rendent la machine agréable à utiliser. La routine OSCLI se révèle particulièrement utile puisqu'elle permet à un programme BASIC d'envoyer des commandes directement au système d'exploitation, libérant le programmeur confirmé de certaines contraintes du BASIC. Le programme assembleur (une des

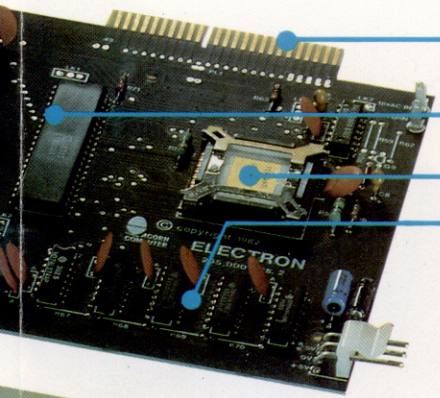




Un duo dynamique
Les créateurs de l'Electron sont Chris Curry (à gauche) et Hermann Hauser (à droite). Ils sont également à l'origine du BBC Micro. Curry était ingénieur du développement chez Clive Sinclair lorsque ce dernier engagea Hauser. (Cl. Judy Goldhill.)

Clavier
Le clavier est parmi les meilleurs possibles, de type machine à écrire, et d'excellente qualité. Il se révèle très semblable à celui du BBC. Il n'existe pas de touches de fonctions programmables, mais la touche "Caps Lock Key" offre des ressources équivalentes. En effet la frappe de cette touche, simultanément à celle d'une touche numérique revient à solliciter une touche fonction.

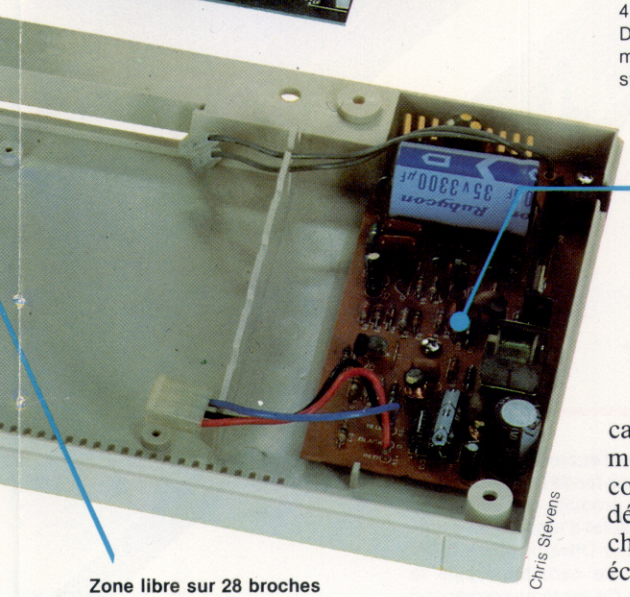
Connecteur d'extension
Il n'est pas fourni de détails sur la valeur des broches ou sur la synchronisation du signal ; cependant, la connection autorisera certainement la plupart des fonctions du système bus, et des lignes transistor-transistor line (TTL) et alimentation électrique.



Unité centrale
Le microprocesseur est un 6502A standard régulé par une horloge à 1,79 MHz. C'est le lieu des décisions, ce que ne peut faire l'unité arithmétique et logique.

Réseau de portes programmables
Il s'agit de la plus grande ULA jamais construite. En dehors de l'ULA, l'unité centrale 6502A, la RAM et la ROM, il n'y a que neuf composants sur la carte fond de panier, tous en logique TTL, chacun ne fournissant que quelques portes logiques.

RAM
Les octets sont chargés de la RAM dans l'UC en deux temps. Les 4 bits de poids faible sont d'abord accessibles (1 bit de chaque puce) puis les 4 autres de poids fort. Dans la plupart des machines, les 8 bits sont stockés dans la même puce.



Circuits de régulation d'alimentation
L'alimentation de Acorn Electron est originale : 1 volt en alternatif. La stabilité du courant en est accrue, mais les circuits de régulation de l'alimentation pour l'ordinateur s'en trouvent compliqués.

Zone libre sur 28 broches
Cet emplacement, disposé pour recevoir un composant et ayant à proximité un lien libre, semble suggérer l'adjonction possible soit d'un bloc de mémoire morte supplémentaire et commutable, soit d'un autre composant.

Chris Stevens

caractéristiques uniques du BASIC BBC) a également été étendu par rapport au BBC Micro. Il comporte des mots clés supplémentaires pour définir le stockage des variables et pour l'affichage des chaînes, supprimant la corvée de leur écriture en langage assembleur.

Le micro-ordinateur Acorn Electron peut être considéré comme au-dessus de la moyenne pour ses performances. L'image transmise est stable et de bonne définition, les couleurs sont bien distinctes. Avec l'adjonction d'extensions appropriées, telles que les lecteurs de disquettes, il pourra prétendre à une grande diffusion auprès du public.

ACORN ELECTRON

PRIX
★ ★

DIMENSIONS
340 × 160 × 65 mm.

UC
6502.

HORLOGE
1,79 MHz.

MÉMOIRE
ROM 64 K.
RAM 32 K (sans compter les extensions possibles).

AFFICHAGE VIDÉO
32 lignes de 80 caractères.
8 couleurs avec les couleurs de fond et d'affichage positionnées indépendamment.
127 caractères prédéfinis, et 255 caractères à définir par l'utilisateur.

INTERFACES
TV, canal 36, vidéo polyvalente, sortie à niveau TTL/RVB, cassette, fond de panier système (BUS sans documentation).

LANGAGE FOURNI
BASIC BBC avec assembleur.

AUTRES LANGAGES DISPONIBLES
Utilise les langages AcornSoft, tels que le FORTH, pourvu qu'ils figurent en RAM. Les langages résidant en ROM tels que BCPL et PASCAL sont incompatibles avec une machine sans extension.

ACCESSOIRES FOURNIS
Manuel d'installation et manuel BASIC, câble TV, transformateur, cassette d'explications.

CLAVIER
56 touches style machine à écrire, une seule clé d'accès au BASIC.
10 fonctions paramétrables par l'utilisateur.

DOCUMENTATION
Simplement excellente. L'ensemble des éléments d'information se révèle très utile aussi bien pour le débutant que pour le programmeur avancé. Chaque mot BASIC est exposé séparément. Le chapitre sur l'assembleur est précieux (assembleur incorporé). Les fonctions du système d'exploitation sont bien décrites. La plupart des tâches de cette machine devraient s'en trouver facilitées.

Jet d'encre

L'impression couleur est devenue possible à un prix raisonnable avec l'apparition d'une imprimante à jet d'encre de couleur sur papier point par point.

Les divers types d'imprimantes disponibles pour un ordinateur individuel sont d'une qualité d'impression très inégale. Les imprimantes à impact de caractère entier donnent les meilleurs résultats (les imprimantes dites à « marguerite » en sont un bon exemple). Les résultats les plus médiocres sont obtenus avec des imprimantes thermiques et électrostatiques. Cependant l'imprimante à matrice de points, bien que de qualité moyenne et assez bruyante, est la plus répandue.

Avec l'apparition de systèmes nouveaux tels que l'imprimante-traceur GP 115 de Tandy, les limites des imprimantes à matrice de points sont devenues plus évidentes. L'imprimante-traceur utilise de minuscules aiguilles pour dessiner des caractères complets ou pour tracer des graphiques, quelquefois en quatre couleurs. Mais le système le plus intéressant est celui de l'imprimante à jets d'encre : de microscopiques jets d'encre sont ordonnés pour former un graphisme particulier.

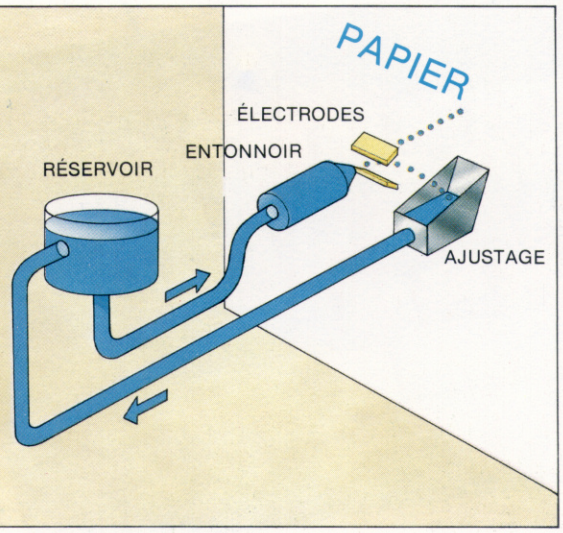
Après l'industrie et le secteur tertiaire (où elles cohabitent avec les imprimantes à laser également très évoluées), ces imprimantes à encre abordent la micro-informatique. L'encre est transmise depuis un réservoir jusqu'à l'extrémité d'un gicleur extrêmement fin. De minuscules particules d'encre sont alors électrisées à haut voltage avant d'être éjectées. La valve est de type piézo-électrique. Elle permet de donner une forme à la projection d'encre par l'intermédiaire de vibrations à haute fréquence.

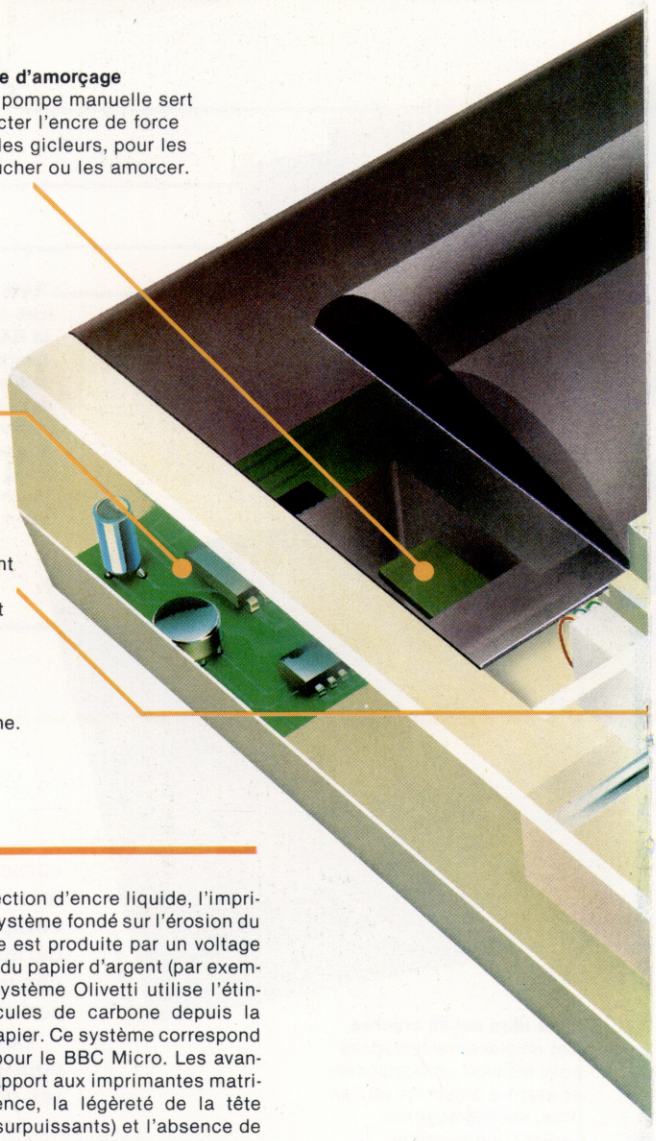
La giclée d'encre est prise dans un champ électrique et se trouve attirée par une charge électrique inverse à la sienne qui la conduit sur le papier. Comment cette rencontre est-elle possible? Grâce à la présence d'une plaque métallique (sur laquelle est tendue la feuille de papier) dont la charge électrique est du signe opposé à celui des gouttelettes. Le principe est simple mais surprenant pour une imprimante. La fiabilité est correcte. Dans le pire des cas, l'ajustage se bouche ou les gouttes d'encre deviennent trop grosses.

Une imprimante à jet d'encre fonctionne selon le même principe qu'une imprimante à matrice de points à un seul marteau. La chaîne des caractères ASCII parvenant à l'imprimante est stockée dans une mémoire tampon jusqu'à ce qu'elle se remplisse ou jusqu'au prochain retour du chariot. L'imprimante examine alors un par un les caractères et recherche leurs équivalents en ROM. Généralement, chaque carac-

Missiles guidés

Les premières imprimantes à jet d'encre étaient très compliquées et onéreuses. Un flux constant d'encre chargée électriquement se déplaçait verticalement, guidé par deux électrodes. La tête d'impression balayait la feuille. En l'absence de caractère à imprimer, l'encre revenait dans un entonnoir avant d'être renvoyée dans le réservoir principal.





Pompe d'amorçage
Cette pompe manuelle sert à injecter l'encre de force dans les gicleurs, pour les déboucher ou les amorcer.

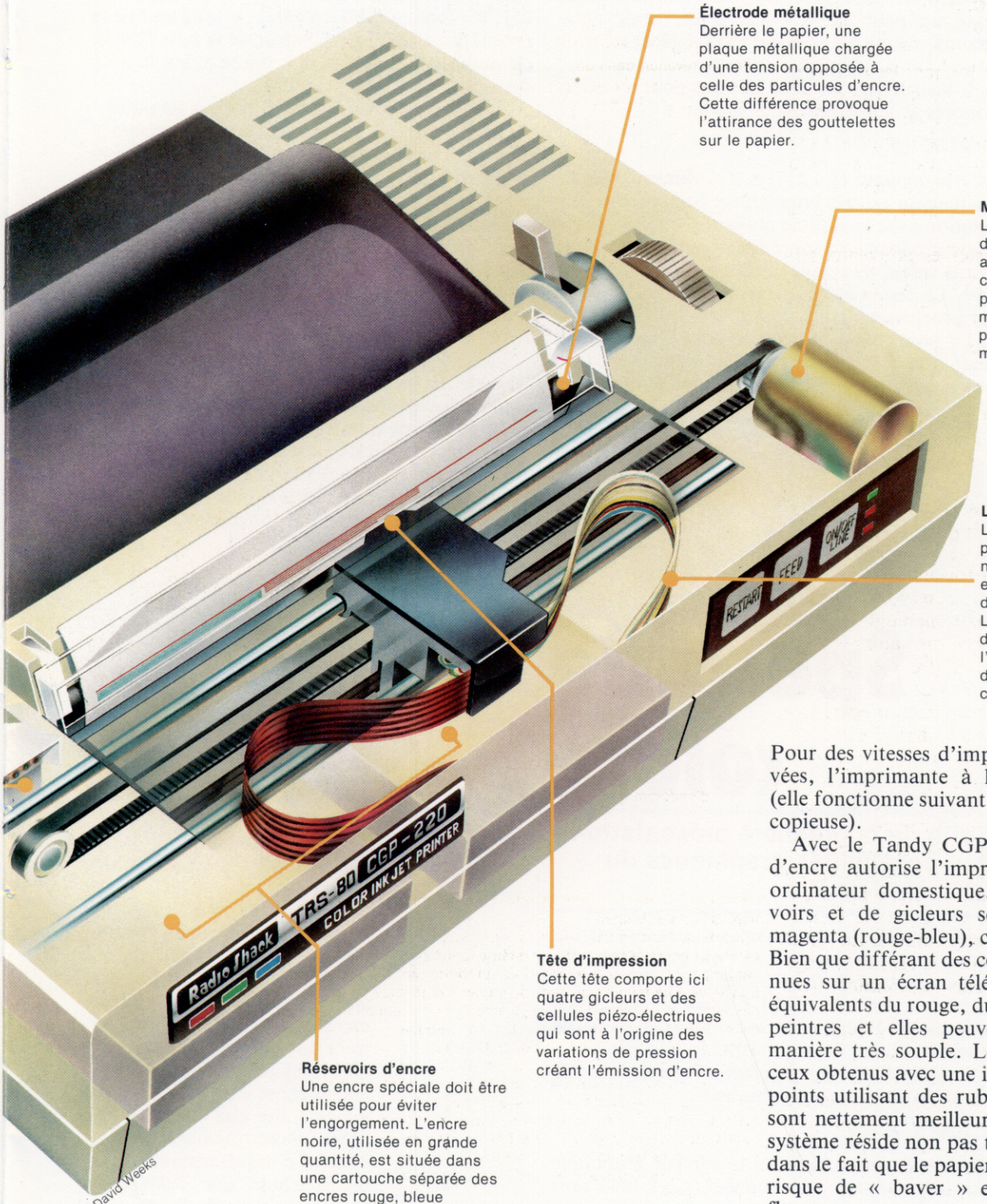
Circuit imprimé
L'imprimante possède son propre microprocesseur 6809, ROM et RAM. Les données reçues sont stockées dans une mémoire tampon (une seule ligne de points étant imprimée par passage).

Clapet de sécurité
La tête d'impression étant très fragile, un clapet l'obstrue lorsqu'elle n'est pas en fonctionnement. Il convient de respecter les procédures de mise en route sous peine d'endommager la machine.

Impression par brûlure
Cousine de l'imprimante à projection d'encre liquide, l'imprimante à « encre sèche » est un système fondé sur l'érosion du papier par brûlure. Une étincelle est produite par un voltage très élevé. Elle fait un trou dans du papier d'argent (par exemple avec l'imprimante ZX). Le système Olivetti utilise l'étincelle pour déplacer des particules de carbone depuis la pointe d'un bâtonnet jusqu'au papier. Ce système correspond à celui de l'imprimante Acorn pour le BBC Micro. Les avantages de ces imprimantes par rapport aux imprimantes matricielles classiques sont le silence, la légèreté de la tête d'impression (plus de moteurs surpuissants) et l'absence de contraintes concernant le papier. Les véritables inconvénients sont la lenteur de l'impression (une seule ligne de points par passage) et le fait que l'encre a trop tendance à diffuser, à baver.

tère est constitué d'un certain nombre de points disposés sur une grille de 8 × 8 que l'imprimante reproduit sur le papier. Pour être nette, une ligne de caractères demande huit passages de la tête d'impression, mais la vitesse est augmentée en permettant à l'imprimante de fonctionner dans les deux sens. Au cours d'une opération d'impression a lieu le chargement du lot de caractères suivants, de sorte que dès l'impression du premier lot achevée commence celle du suivant. La seule différence entre l'imprimante à jet d'encre et l'imprimante à matrice de caractères est que la première projette des gouttelettes d'encre chargées électriquement, alors que la seconde percute la page à travers un ruban d'encre.

Dans leur version commerciale, les imprimantes à jet d'encre peuvent imprimer une page en quelques secondes seulement. La qualité d'impression est bien meilleure si le papier ne « boit » pas. Pour des applications de grand volume, de niveau professionnel, c'est l'idéal.



David Weeks

Électrode métallique
Derrière le papier, une plaque métallique chargée d'une tension opposée à celle des particules d'encre. Cette différence provoque l'attirance des gouttelettes sur le papier.

Moteurs d'entraînement
Le glissement transversal de la tête d'impression est assuré par un moteur conventionnel. L'avance du papier est assurée par un moteur pas à pas, comme pour une imprimante à matrice de points.

Liaisons flexibles
La plupart des imprimantes possèdent des câbles-nappes flexibles de liaison entre la carte et la tête d'impression.
La vitesse du dispositif d'impression rend l'alimentation en encre délicate (quatre encres de couleurs différentes).

Tête d'impression
Cette tête comporte ici quatre gicleurs et des cellules piézo-électriques qui sont à l'origine des variations de pression créant l'émission d'encre.

Réservoirs d'encre
Une encre spéciale doit être utilisée pour éviter l'engorgement. L'encre noire, utilisée en grande quantité, est située dans une cartouche séparée des encres rouge, bleue et jaune.

Pour des vitesses d'impression encore plus élevées, l'imprimante à laser conviendra mieux (elle fonctionne suivant le principe d'une photocopieuse).

Avec le Tandy CGP220, l'imprimante à jet d'encre autorise l'impression couleur pour un ordinateur domestique. Elle dispose de réservoirs et de gicleurs séparés pour les encres magenta (rouge-bleu), cyan (bleu-vert) et jaune. Bien que différant des couleurs habituelles obtenues sur un écran télé, ces couleurs sont les équivalents du rouge, du vert et du bleu pour les peintres et elles peuvent être mélangées de manière très souple. Les résultats comparés à ceux obtenus avec une imprimante à matrice de points utilisant des rubans de diverses couleur sont nettement meilleurs. L'inconvénient de ce système réside non pas tant dans son prix, mais dans le fait que le papier doit être absorbant, au risque de « baver » et d'obtenir une image floue.

Test musical

Synthèse du son : le Dragon 32 à l'épreuve.

Le Dragon 32 est muni d'un seul oscillateur à onde carrée pour programmer le son. Mais les instructions sonores remarquablement simples du BASIC Extended Color de Microsoft permettent de construire une chaîne musicale qui joue un petit air à l'aide d'une seule commande, même s'il n'existe pas de générateur de bruit.

L'instruction SOUND ne sert que pour les effets sonores et le format est le suivant :

```
SOUND H,D
```

où H = hauteur (1 à 255) et D = durée (1 à 255). La hauteur est très imprécise et comporte peu de relation avec la gamme musicale, quoique le *do* majeur puisse être approximativement remplacé par la valeur 89 et le *la* de référence à 440 Hz par 159. La durée est également inexacte, mais 16 est proche de 1 s, 32 de 2 s, etc.

Le programme suivant montre comment on peut utiliser SOUND pour obtenir un effet spécial; voici, avec un peu d'imagination, le décollage d'un ovni :

```
10 FOR H = 10 TO 170 STEP 10
20 FOR D = 16 TO 1 STEP - 1
30 SOUND H,D
40 NEXT D
50 NEXT H
```

PLAY fixe exactement la hauteur, la durée et le volume d'une note. Elle peut aussi spécifier qu'une série de notes soit jouée avec des silences et un tempo variable. Cela permet de construire des mélodies comprenant des notes et des silences de longueurs différentes, à l'aide de la seule instruction :

```
PLAY « T;O;V;L;N;P »
```

où T = tempo (T1 à T255); O = octave (01 à 05); V = volume (V0 à V15); L = longueur de la note (L1 à L255); N = valeur de la note (1 à 12 ou nom de la note); et P = pause avant la note suivante (P1 à P255).

Il n'est pas indispensable d'utiliser les points-virgules entre les paramètres, mais la clarté y gagne. Cet exemple n'est qu'une représentation arbitraire, puisque les paramètres peuvent être placés dans n'importe quel ordre. T, O, V et L gardent leurs valeurs jusqu'à une nouvelle instruction. En fait, T, O, V, L et P prennent les valeurs respectives de T2, O2, V15, L4 et P0, à moins d'une autre spécification, de sorte qu'il n'est pas toujours nécessaire de les inclure dans l'instruction PLAY.

Là où le temps intervient, comme dans L et P, les valeurs spécifiées peuvent être considérées comme des « notes » et des fractions de « note » : L1 ou P1 est une note entière, L2 ou P2 une demi-note, etc. Leur rythme est choisi à l'aide du paramètre « T », où T1 est lent (une note dure longtemps) et T255 est très rapide (une note est très brève). En outre, les longueurs des notes

Spectacle coloré

Suite de la première présentation des possibilités graphiques du BBC Modèle B.

Le BASIC BBC ne fournit pas toutes les instructions haute résolution que l'on trouve sur certains micro-ordinateurs. Par exemple, les instructions CIRCLE ou PAINT n'existent pas. Il est néanmoins possible de les simuler à l'aide de quelques lignes de BASIC BBC.

Les instructions suivantes permettent le contrôle de l'écran graphique :

```
MOVE x, y
```

Cette instruction déplace le curseur graphique jusqu'au point de coordonnées (x, y), mais sans tracer de ligne.

```
DRAW x, y
```

DRAW (= dessiner) trace une ligne à partir de la position du curseur graphique jusqu'au point de l'écran de coordonnées (x, y).

```
PLOT k, x, y
```

PLOT est une instruction à usages multiples; sa fonction est gouvernée par la valeur donnée à la variable K :

Valeur de K	Fonction
0	déplace par rapport au dernier point;
1	trace une ligne à partir de l'origine dans la couleur de l'avant-plan;
2	trace une ligne à partir de l'origine dans la couleur complémentaire;
3	trace une ligne à partir de l'origine dans la couleur du fond;
4	même chose que MOVE;
5	même chose que DRAW;
6	même chose que DRAW mais en couleur complémentaire;
7	même chose que DRAW mais dans la couleur du fond.

peuvent être définies avec plus de souplesse en ajoutant des points tels que L1... ou L5., chaque point augmentant la longueur de la note de la moitié de sa valeur normale. Ainsi

L1... = $1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} = 2\frac{1}{2}$ notes et

L5. = $\frac{1}{5} + \frac{1}{10} = \frac{3}{10}$ de note.

Il n'existe pas, dans l'absolu, de moyen de représenter la relation entre note et tempo. Les valeurs peuvent varier pour chaque air et seront choisies à la suite d'essais. Cela peut prendre du temps, mais l'instruction y gagne en souplesse.

Le paramètre O spécifie l'octave dans laquelle la prochaine note doit être jouée. O1 commence par le *do* à 131 Hz et O5 termine par le *si* à 2 039 Hz. Le *do* médian commence l'octave O2. A l'intérieur d'une octave, les notes peuvent être spécifiées de deux manières. La première utilise un nombre qui correspond à une note musicale de la façon suivante :

1	2	3	4	5	6
do	*do ♯*	ré	ré ♯	mi	fa
7	8	9	10	11	12
fa ♯	sol	sol ♯	la	la ♯	si

Il est ainsi possible de spécifier une note comme une variable à l'intérieur d'une octave donnée. La seconde désigne directement la note par son nom pour rendre l'instruction plus compréhensible dans un listage.

Un exemple illustrera mieux ces explications. L'instruction suivante joue *fa* (6) de l'octave O2, pendant la moitié de durée d'une note (L2) au volume (V15). Puis il y a un silence durant un quart de note (L1) au volume V20. Le tempo est fixé à T3 :

```
PLAY « T3;L2;6;P4;O3;V20;L1; LA ♯ »
      < FA>< LA ♯ >
            silence
```

En outre, on peut faire varier les paramètres T, O, V et L de quantités fixées d'avance par l'instruction, à l'aide d'un suffixe :

Suffixe	*Effet*
+	ajoute 1 à la valeur en cours ;
–	retranche 1 de la valeur en cours ;
<	multiplie la valeur en cours par 2 ;
>	divise la valeur en cours par 2.

Le format est : T+, T–, T> ou T< pour chaque paramètre.

La meilleure possibilité du Dragon est de jouer des airs de musique en utilisant des « sous-chaînes ». Celles-ci sont définies avant d'être jouées dans n'importe quel ordre ou répétées :

```
10 A$ = «FA;LA ♯;SOL»
20 B$ = «DO;RÉ ♯;FA;P4;XA$;»
30 PLAY B$
```

Ayant ainsi défini A$, on l'inclut dans B$ sous la forme XA$. Voici l'air qui en résulte : *do - ré ♯ - fa - P4 - fa - la ♯ - sol.*

Cette technique peut être poursuivie si nécessaire lorsque des séquences de notes sont répétées un certain nombre de fois dans un morceau de musique. Dans tous les cas, il faut mettre un point-virgule après une sous-chaîne, comme en XA$ ci-dessus.

Les nombres supérieurs répètent ces huit fonctions, mais avec des effets supplémentaires tels que des pointillés au lieu de traits continus. Les valeurs de K comprises entre 80 et 87 accomplissent une fonction particulièrement utile. PLOT 80,x,y joint le point (x,y) aux deux derniers points tracés pour former un triangle. L'intérieur du triangle se remplit de la couleur d'avant-plan. C'est là la seule manière simple de « peindre » des formes graphiques.

VDU x est l'équivalent de l'instruction BASIC PRINT CHR$ (x). Nous avons vu, dans l'introduction aux graphiques sur le Micro BBC, que VDU peut être suivi d'une série de nombres. VDU v,w,x,y,z est équivalent à :

```
PRINT CHR$(v);CHR$(w);CHR$(x);CHR$(y);CHR$(z)
```

Les instructions VDU permettent à l'utilisateur d'accéder à la partie du système d'exploitation qui contrôle l'affichage. Quoique l'instruction VDU puisse être utilisée dans les programmes en BASIC, elle fonctionne indépendamment du langage employé. Ainsi, les mêmes instructions VDU pourraient servir à un affichage graphique en PASCAL ou en tout autre langage offert sur le BBC. Toutes les possibilités graphiques en BASIC discutées jusqu'ici peuvent aussi être exécutées par l'instruction VDU appropriée.

Il est très facile de définir des caractères sur le Micro BBC. VDU contrôle cette fonction. Dans la section sur les graphiques définis par l'utilisateur, nous avons appris que les codes ASCII normaux sont construits à partir d'un bloc de 8 × 8 pixels. Les pixels qui sont visibles peuvent être représentés par un 1 en binaire et les invisibles par un 0. Chaque rangée est alors convertie en son équivalent décimal ; soit huit nombres décimaux pour définir un caractère. VDU 23 permet à l'utilisateur de redéfinir ce caractère avec un code ASCII compris entre 224 et 255. Par exemple :

```
10 REM DÉFINIR UN CARACTÈRE
20 MODE 2
30 VDU 23,240,16,56,124,146,16,16,16,0
40 PRINT CHR$(240)
50 END
```

Ce bref programme redéfinit le caractère de code ASCII 240 pour créer une forme de flèche. Les huit derniers nombres définissent cette nouvelle forme, et la ligne 40 affiche le caractère sur l'écran. En utilisant ces fonctions, les sorties graphiques et textes sur l'écran peuvent être limités à certaines zones définies. Cela est très utile dans le cas de programmes interactifs où il est souhaitable de partager l'écran.

MODE 1

Ce petit programme trace une fleur en spirales coloriées sur l'écran en utilisant la résolution en MODE 1. Noter l'utilisation de triangles colorés pour le dessin des pétales.

```
10 REM FLEUR
20 CLS
30 MODE 1
40 FOR D = 1 TO 3
50 A = 600:B = 500
60 MOVE A,B
70 FOR C = 1 TO 550 STEP 3
80 GCOL0,RND(3)
90 S = (C/RND(5)+10)
100 X = S*5*SIN(C/16)+A
110 Y = S*5*COS(C/16)+B
120 PLOT 85,x,y
130 NEXT C
140 NEXT D
150 END
```

Le motif spiralé est produit par la combinaison de sinus et cosinus dans les lignes 100 et 110. Normalement, cette relation entre les coordonnées x et y produit un cercle, mais la boucle FOR...NEXT augmente progressivement le rayon C, et crée une spirale. Les coordonnées du centre de la spirale, A et B, peuvent être modifiées pour changer la position de la fleur.

Passage d'essai

Pour utiliser des fichiers de données, il faut d'abord les créer sous forme de canevas avant de les remplir avec des informations.

A la fin du dernier cours, nous avions laissé le lecteur face à cette impossibilité apparente : comment faire lire à un programme tournant pour la première fois un fichier qui n'existe pas (sur cassette ou sur disque)? La première fonction que nous sommes en droit d'attendre du programme est que celui-ci lise le fichier de données et attribue ces données à des tableaux ou à des variables. Toutefois, si nous insistons d'abord sur l'écriture du fichier, nous devrons veiller, en programmant, à ne pas perdre toutes les données dans le fichier. Comme nous l'avons déjà constaté, si on tente d'ouvrir un fichier qui n'existe pas, soit cela ne fonctionnera absolument pas, soit le programme se « plantera » (arrêtera de fonctionner).

Heureusement, il existe une solution simple. De nombreux logiciels commerciaux comportent un programme de « mise en route » que l'on doit faire marcher avant d'utiliser le programme principal; nous adopterons cette approche. Ces programmes permettent, en principe, de faire un certain nombre de modifications (par exemple de choisir d'utiliser une imprimante Epson ou une Brother, parallèle ou série, etc.).

Pour résoudre notre problème et permettre la procédure *LECFCR* (le programme qui lit dans le fichier et attribue les données aux tableaux), nous pouvons écrire un programme de « mise en route » simple qui ne fait qu'ouvrir un fichier et y écrire une valeur factice. Nous choisirons une valeur qui puisse être reconnue ensuite par le programme principal comme un enregistrement dont l'adresse n'est pas valide. Une valeur convenable serait la chaîne de caractères @PREMIER parce qu'aucun nom ni adresse, si obscure que soit son origine, n'a de chance de commencer par cette chaîne particulière. *LECFCR* devra être légèrement modifié afin que, lorsqu'il ouvre et lit le fichier, il vérifie sa valeur avant d'aller plus loin. Si votre ordinateur n'a pas le symbole @, vous pouvez le remplacer par ! ou par tout autre caractère qu'on ne rencontre normalement pas dans le registre d'adresses. Voici le programme de mise en route :

```
10 REM CE PROGRAMME CRÉE UN FICHIER DE DONNÉES
20 REM A UTILISER PAR LE PROGRAMME DE REGISTRE
30 REM D'ADRESSES
40 REM IL ÉCRIT UN ENREGISTREMENT FACTICE
50 REM QUI PEUT ÊTRE UTILISÉ PAR *LECFCR*
60 REM
70 OPEN « O », #1, « ADBK.DAT »
80 PRINT #1, « @PREMIER »
90 CLOSE #1
100 END
```

Comme nous l'avons mentionné précédemment dans ce cours, les détails de lecture et d'écriture de fichiers diffèrent considérablement d'une version de BASIC à une autre, mais le principe est presque toujours le même. D'abord il faut ouvrir le fichier à l'aide de l'instruction OPEN avant de pouvoir l'utiliser pour des entrées ou des sorties. Ensuite, on indique le sens du flux de données (IN ou OUT). Puis on attribue un numéro de « canal » au fichier. Cela permet d'ouvrir et d'utiliser plusieurs fichiers à la fois (pour le moment, nous n'utiliserons qu'un seul fichier). Enfin, il faut déclarer le nom du fichier que nous voulons utiliser.

La ligne 70 du programme est en BASIC Microsoft et elle est en principe semblable aux instructions OPEN utilisées par la plupart des BASIC. Bien entendu, OPEN déclare qu'un fichier doit être ouvert et O (« output ») indique que des données seront sorties. #1 est le numéro attribué au fichier pour cette opération; un autre numéro de fichier pourra être utilisé plus tard s'il le faut. ADBK.DAT est le nom que nous avons donné au fichier.

La ligne 80 inscrit simplement un enregistrement dans le fichier. La syntaxe d'écriture de données dans un fichier est généralement (dans la plupart des BASIC) exactement la même que celle qui est utilisée pour PRINT, sinon que celle-ci doit être suivie du numéro de fichier — #1 dans le cas présent.

La ligne 90 renferme le fichier. On peut laisser les fichiers ouverts aussi longtemps que l'exige le programme, mais ils sont très vulnérables et il faut les refermer dès que possible pour protéger les données qu'ils contiennent.

Il règne une certaine confusion sur la manière dont les termes « enregistrement » et « fichier » sont utilisés en informatique. Cette confusion s'aggrave lorsqu'il est question de « bases » de données, d'une part, et de « fichiers » de données, d'autre part. Dans une base de données, le fichier est un ensemble d'informations reliées entre elles. Par analogie, considérons un classeur de bureau : le fichier pourrait être un tiroir marqué « PERSONNEL ». Ce fichier comprend un enregistrement (une fiche dans une chemise) sur chaque personne de la société. Chaque enregistrement (fiche) contient un certain nombre de zones, identiques pour chaque fiche, contenant des informations telles que « NOM », « SEXE », « AGE », « SALAIRE », « ANNÉES DE SERVICE », etc.

Pour un fichier séquentiel sur disque ou sur cassette, la façon dont l'information est utilisée ou organisée par le programme n'a pas d'importance. Les fichiers de données contien-

nent simplement une série d'articles, dont chacun, pris individuellement, constitue un enregistrement. Un seul enregistrement dans un fichier de données ne correspondrait donc pas à un enregistrement dans le sens de base de données.

Le programme doit lire les enregistrements du fichier de données et les attribuer à des variables, ou tableaux. Ceux-ci doivent être organisés pour former un enregistrement « conceptuel » contenant un ensemble limité d'informations. Il n'y a pas de relation biunivoque entre les enregistrements dans un fichier et ceux qui comprennent une base de données.

Une fois le programme de mise en route passé, on n'en aura plus jamais besoin. Si on le faisait repasser, il détruirait toutes les données « légitimes » qu'on aurait pu entrer dans la base de données du registre d'adresses. Nous allons voir pourquoi, en considérant le programme *LECFCR* modifié.

Lorsque le programme tourne, il ne « sait » pas s'il y a des données légitimes ou non dans le fichier de données. La première chose que fasse *LECFCR* est d'ouvrir le fichier ADBK.DAT et d'y lire le premier enregistrement (ou article). Il ne s'agit pas de lire dans un élément de tableau, comme on pourrait le croire, mais dans une variable de chaîne particulière que nous avons appelée TEST$. Avant de lire tout autre enregistrement, TEST$ est vérifié pour voir s'il contient la chaîne @PREMIER. Si TEST$ contient @PREMIER, le programme « sait » qu'il n'y a pas de données valables dans le fichier et qu'il n'y a donc pas lieu de continuer à lire. Par conséquent, on peut fermer le fichier et le reste du programme peut se poursuivre. Puisqu'il n'y a pas de données valables dans le fichier, l'utilisateur ne peut rien faire de valable avant qu'au moins un enregistrement n'ait été entré; la valeur de TEST$ peut donc également servir à obliger le programme à aller au sous-programme *AJOUTENR*, afin d'ajouter au moins un enregistrement valable.

Si, en revanche, la valeur de TEST$ n'est pas @ PREMIER, le programme admet qu'il y a des données valables dans le fichier et il peut commencer à attribuer les données aux tableaux appropriés.

Voici le sous-programme *LECFCR* modifié :

```
1400 REM *LECFCR* SOUS-PROGRAMME
1410 OPEN « I », #1, « ADBK.DAT »
1420 INPUT #1, TEST$
1430 IF TEST$ = «@PREMIER » THEN GOTO 1530: REM
     CLOSE AND RETURN
1440 LET NOMCHP$(1) = TEST$
1450 INPUT #1, MODCHP$(1), RUECHP$(1), VILCHP$(1),
     CPOCHP$(1), TELCHP$(1)
1460 INPUT #1, NDXCHP$
1470 LET TAILLE = 2
1480 FOR L = 2 TO 50
1490 INPUT #1, NOMCHP$(L), MODCHP$(L), RUECHP$(L),
     VILCHP$(L), CPOCHP$(L)
1500 INPUT #1, TELCHP$(L), NDXCHP$(L)
1510 REM ESPACE POUR APPELER SOUS-PROGRAMME
     « TAILLE »
1520 NEXT L
1530 CLOSE #1
1540 RETURN
```

La ligne 1420 assigne un seul enregistrement du fichier ADBK.DAT à la variable TEST$. La ligne suivante vérifie alors si cette valeur est @PREMIER. Si oui, l'instruction GOTO renvoie à la ligne 1530 qui ferme le fichier, puis le sous-programme revient (RETURN) au programme principal. Plus aucune tentative ne sera faite pour lire les données. En supposant qu'il n'y a pas de données valables dans le fichier, la commande de programme sera ramenée à *INITIL* qui appelle alors *DEFDRA*. Tout ce que ce programme fait pour le moment est de fixer la grandeur à 1 si TEST$ = @PREMIER. Le code pour *DEFDRA* est donné ci-dessous.

```
1600 REM *DEFDRA*
1610 REM DÉFINIR DRAPEAUX APRÈS *LECFCR*
1620 REM
1630 REM
1640 IF TEST$ = «@PREMIER » THEN LET TAILLE = 0
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
```

Puis *DEFDRA* retourne à *INITIL* qui, à son tour, retourne au programme principal. *PROPRINC* appelle alors *ACCUEIL* qui affiche le message d'accueil. *ACCUEIL* ne nécessite aucune modification de la version précédemment publiée.

Voici le programme *CHOIX* appelé par le programme principal. Une toute petite modification au sous-programme *CHOIX* établira un moyen d'obliger l'utilisateur à ajouter un enregistrement si le programme tourne pour la première fois.

```
3500 REM *CHOIX* SOUS-PROGRAMME
3510 REM
3520 IF TEST$ = «@PREMIER » THEN GOSUB 3860
3530 IF TEST$ = «@PREMIER » THEN RETURN
3540 REM « CHEMENU »
3550 PRINT CHR$(12)
3560 PRINT « SELECT UN DE SUITE »
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT « 1. TROUVER ENREGISTREMENT (DE NOM) »
3610 PRINT « 2. TROUVER NOMS (DE NOM INCOMPLET) »
3620 PRINT « 3. TROUVER ENREGISTREMENT (DE VILLE) »
3630 PRINT « 4. TROUVER ENREGISTREMENT (D'INITIALE) »
3640 PRINT « 5. LIST TOUS ENREGISTREMENTS »
3650 PRINT « 6. AJOUTER NOUVEAUX ENREGISTREMENTS »
3660 PRINT « 7. CHANGER ENREGISTREMENT »
3670 PRINT « 8. ENLEVER ENREGISTREMENT »
3680 PRINT « 9. EXIT & SAVE »
3690 PRINT
3700 PRINT
3710 REM « ENTRER CHOIX »
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 0 TO 1
3760 PRINT « ENTRER CHOIX (1 - 9) »
3770 FOR I = 1 TO 1
3780 LET A$ = INKEY$
3790 IF A$ = « » THEN I = 0
3800 NEXT I
3810 LET CHOI = VAL(A$)
3820 IF CHOI < 1 THEN L = 0 ELSE L = 1
3830 IF CHOI > 9 THEN L = 0
3840 NEXT L
3850 RETURN
```

Deux lignes ont été rajoutées. La première teste TEST$. Cette variable contient encore la valeur lue au cours du programme *LECFCR*. Si c'est @PREMIER, nous savons qu'il n'y a pas de données valables dans le fichier, et la seule option appropriée est donc AJOUTENR, dont le numéro est 6. Si le test est passé, la commande passe à *PREM*, sous-programme qui affiche un message approprié et fixe la variable CHOI à 6. Lorsque le sous-programme retourne à la ligne 3530, TEST$

est à nouveau testé (il est obligé de passer) et le sous-programme revient au programme principal, en sautant le reste du sous-programme *CHOIX* puisqu'il ne convient pas.

Vous vous êtes peut-être demandé pourquoi TEST$ est testé deux fois. C'est pour éviter que le sous-programme ne retourne au mauvais endroit du programme. Sans la ligne 3530, le programme continuerait jusqu'au bout de *CHOIX*, en présentant un menu même s'il n'est pas nécessaire. Cela évite aussi d'utiliser GOTO, bien que IF TEST$ = "@PREM" THEN GOTO 3850 marche tout aussi bien.

Avant de poursuivre avec *PREM*, le lecteur se reportera à *LECFCR* et au GOTO de la ligne 1430. Puisque nous avons contesté l'usage de GOTO, pourquoi l'avoir utilisé ici? Il aurait été très facile de fermer (CLOSE) le fichier et de revenir au programme principal (RETURN) en testant simplement la valeur de TEST$ en deux lignes distinctes. Mais nous avons utilisé ici GOTO pour illustrer un des quelques exemples où son usage est excusable : à l'intérieur d'un segment de programme très court et identifiable, et sa fonction est évidente (d'autant plus qu'elle est commentée par REM). GOTO ne doit jamais être utilisé pour sortir d'une boucle (cela peut laisser la valeur des variables dans un état indéterminé), ni d'un sous-programme (sous peine d'embrouiller l'instruction RETURN, à moins de rajouter une instruction pour retourner dans le sous-programme), ni pour aller vers des parties trop éloignées du programme (ce qui rend le programme impossible à suivre).

Le sous-programme *PREM* est simple et direct : l'écran est effacé et un message s'affiche pour informer l'utilisateur qu'un enregistrement doit être entré. La ligne 3870 donne à CHOI la valeur 6 afin que le programme *AJOUTENR* soit exécuté automatiquement lorsqu'on repasse le contrôle à *EXECUT*.

Le code pour *PREM* est le suivant :

```
3860 REM *PREM* SOUS-PROGRAMME (AFFICHE
     MESSAGE)
3870 LET CHOI = 6
3880 PRINT CHR$(12): REM EFFACER ÉCRAN
3890 PRINT
3900 PRINT TAB(8): « PAS D'ENREGISTREMENTS DANS »
3910 PRINT TAB(8): « LE FICHIER. VOUS AUREZ A
3920 PRINT TAB(6): « COMMENCER PAR AJOUTER
     ENREGISTREMENT »
3930 PRINT
3940 PRINT TAB(5): « (APPPUYER BARRE-ESPACE POUR
     CONTINUER)
3950 FOR B = 1 TO 1
3960 IF INKEY$ < >« » THEN B = 0
3970 NEXT B
3980 PRINT CHR$(12): REM EFFACER ÉCRAN
3990 RETURN
```

Le sous-programme *AJOUTENR* comporte deux modifications légères mais importantes par rapport à la version rencontrée précédemment. Après avoir entré les zones en tant qu'éléments dans les différents tableaux, on incrémente la variable TAILLE et on donne à TEST$ une valeur nulle (voir lignes 10090 et 10100). TAILLE est une variable importante utilisée dans différentes parties du programme, de sorte qu'elle « sait » sur quels enregistrements on est en train de travailler. A l'origine, on avait posé TAILLE = 0 en tant que partie du sous-programme *CRETAB*. Ensuite, dans *DEFDRA*, il est fixé à 1 si TEST$ = « @PREMIER ». Ainsi, la première fois que l'on exécute *AJOUTENR*, les instructions d'entrée mettront les données dans le premier élément de chaque tableau. En d'autres termes, INPUT « ENTRER NOM »; NOMCHP$ (TAILLE) est équivalent à INPUT « ENTRER NOM »; NOMCHP$(1).

La ligne 10090 incrémente TAILLE qui est maintenant égal à 2. Si *AJOUTENR* est à nouveau exécuté, les données seront entrées dans le second élément de chaque tableau. Finalement *AJOUTENR* fixe TEST$ à « » à la ligne 10100. Cela a lieu parce qu'un enregistrement est entré (quoique non encore stocké dans le fichier de données sur bande ou disque). Si l'on exécute à nouveau *CHOIX*, comme il faut le faire pour sauvegarder les données et sortir du programme, nous ne voulons pas être forcés d'ajouter un nouvel enregistrement. Si TEST$ n'était pas remis à zéro, le programme serait pris dans une boucle sans fin. Le seul moyen d'en sortir serait de faire RESET ou de débrancher l'ordinateur. Les données seraient perdues.

En posant TEST$ égal à une chaîne nulle, les tests des lignes 3520 et 3530 de *CHOIX* ne réussissent pas et permettent l'affichage du menu. Ce qui arrive alors à TAILLE dépend du programme exécuté. Jusqu'à présent nous avons posé que TAILLE = 1 s'il n'y a pas de données valables dans le fichier, et que sa valeur est incrémentée de 1 chaque fois qu'on ajoute un enregistrement. Mais que se passerait-il s'il y avait un certain nombre d'enregistrements valables dans le fichier? Pour le savoir, reconsidérons *LECFCR*.

La ligne 1420 lit la première donnée dans TEST$. Si ce n'est pas @PREMIER, il est admis qu'elle est valable. Les enregistrements dans le fichier suivent toujours le même ordre : NOMCHP, MODCHP, RUECHP, VILCHP, CPOCHP, TELCHP, NDXCHP, NOMCHP, MODCHP, et ainsi de suite. Si le premier enregistrement lu est une donnée valable, elle doit faire partie du premier élément du tableau NOMCHP$, donc la ligne 1440 transfère cette donnée de TEST$ à NOMCHP$(1). Les deux lignes suivantes remplissent les cinq autres tableaux. Nous savons que nous avons au moins un enregistrement complet (base de donnée), donc on pose TAILLE = 2. Cette valeur doit être supérieure au nombre d'enregistrements valables lus dans les tableaux, sinon *AJOUTENR* écrirait de nouvelles données dans des éléments qui contiennent déjà des données valables.

Ensuite, une boucle de 2 à 50 lit les enregistrements dans les six tableaux, incrémentant l'indice L à chaque passage. Nous avons décidé de restreindre notre programme à des fichiers de cinquante noms et adresses, et les instructions DIM dans le sous-programme *CRETAB* ont prévu cet espace. A la première utilisation du programme, il est peu probable que l'on dispose d'un fichier complet de cinquante articles. Il faudra donc prévoir un sous-programme qui puisse détecter cela le cas échéant, fixer la valeur de TAILLE en conséquence et sortir de la boucle de lecture.

Nous avons donc inclus la ligne 1510 pour appeler un sous-programme 'TAILLE' que nous expliciterons ultérieurement. Il y a trois façons d'aborder ce problème. Premièrement, lorsque nous écrivons la donnée sur la bande, nous pourrions faire en sorte que le premier enregistrement écrit soit la variable TAILLE. Le sous-programme *LECFCR* peut alors être modifié pour lire d'abord TAILLE avant d'entrer dans une boucle de la forme FOR L = 1 TO TAILLE pour lire les enregistrements. Deuxièmement, on ne commence la procédure que lorsque tous les enregistrements ont été écrits et terminés par un indicateur spécial (peut-être de la forme @END). Cette méthode est préférable car elle ne se heurte pas à notre précédent test @PREMIER à la ligne 1430. On peut ensuite faire un test dans *LECFCR* pour sortir de la boucle lorsqu'on rencontre @END. Troisièmement, on peut faire usage de la fonction EOF (fin de fichier) existant sur certains ordinateurs, qui consiste en une version automatisée de la deuxième méthode. Ces ordinateurs ont un indicateur d'EOF qui a normalement la valeur 0, c'est-à-dire « faux », et parfois une autre valeur, généralement 1 (« vrai »), lorsque la fin du fichier est atteinte. Certains BASIC permettent de tester l'indicateur EOF comme s'il s'agissait d'une variable BASIC; dans ce cas, une construction de la forme :

```
TANT QUE EOF(N) N'EST PAS VRAI (N est le numéro de
fichier)
  DO ...
    INPUT #N, données à lire
FIN
```

résoudra le problème. Sur d'autres machines, EOF est représenté comme un simple bit auquel on accède par l'instruction PEEK. Pour trouver si votre machine possède une fonction EOF, vous devez consulter le manuel d'instructions.

```
4000 REM SOUS-PROGRAMME *EXECUTE*
4010 REM
4019 IF CHOI = 6 THEN GOSUB 10000: REM VOIR BAS
     DE PAGE
4020 REM NORMALEMENT « ON CHOI GOSUB etc »
4030 REM
4040 REM 1 EST *TROUVENR*
4050 REM 2 EST *TRNOMS*
4060 REM 3 EST *TRVILLE*
4070 REM 4 EST *TRINIT*
4080 REM 5 EST *MODENR*
4090 REM 6 EST *AJOUTENR*
4100 REM 7 EST *MODENR*
4110 REM 8 EST *EFFENR*
4120 REM 9 EST *QUITTE*
4130 REM
4140 RETURN
```

Le sous-programme *EXECUT* ne devrait normalement pas avoir la ligne 4019 (d'où ce numéro de ligne particulier), la ligne 4020 devrait être :

```
ON CHOI GOSUB numéro, numéro, numéro, etc.
```

ou une série de :

```
IF CHOI = 1 THEN GOSUB numéro
IF CHOI = 2 THEN GOSUB numéro, etc.
```

La ligne 4019 permet de faire tourner le programme même si les autres sous-programmes EXECUT n'ont pas encore été codés.

```
10 REM « PROG PRINCIPAL »
20 REM *INITIL*
30 GOSUB 1000
40 REM *ACCUEIL*
50 GOSUB 3000
60 REM *CHOIX*
70 GOSUB 3500
80 REM *EXECUTE*
90 GOSUB 4000
100 END

1000 REM *INITIL* SOUS-PROGRAMME
1010 GOSUB 1100: REM *CRETAB* SOUS-PROGRAMME (CRÉER
     TABLEAUX)
1020 GOSUB 1400: REM *LECFCR* SOUS-PROGRAMME (LECTURE
     FICHIER)
1030 GOSUB 1600: REM *DEFDRA* SOUS-PROGRAMME (VALIDATION
     DRAPEAUX)
1040 REM
1050 REM
1060 REM
1070 REM
1080 REM
1090 RETURN

1100 REM *CRETAB* SOUS-PROGRAMME (CRÉER TABLEAUX)
1110 DIM NOMCHP$
1120 DIM MODCHP$
1130 DIM VILCHP$
1140 DIM CPOCHP$
1150 DIM TELCHP$
1160 DIM NDXCHP$
1170 REM
1180 REM
1190 REM
1200 REM
1210 LET TAILLE = 0
1220 LET RMOD = 0
1230 LET SVED = 0
1240 LET ACT = 0
1250 REM
1260 REM
1270 REM
1280 REM
1290 REM
1300 RETURN

10000 REM SOUS-PROGRAMME *AJOUTENR*
10010 PRINT CHR$(12): REM EFFACER ÉCRAN
10020 INPUT « ENTRER NOM »; NOMCHP$(TAILLE)
10030 INPUT « ENTRER RUE »; RUECHP$(TAILLE)
10040 INPUT « ENTRER VILLE »; VILCHP$(TAILLE)
10050 INPUT « ENTRER CODE POSTAL »; CPOCHP$(TAILLE)
10060 INPUT « ENTRER NUMÉRO TÉLÉPHONE »; TELCHP$(TAILLE)
10070 LET RMOD = 1: REM INDICATEUR ARTICLE MODIFIÉ
10080 LET NDXCHP$(TAILLE) = RUECHP$(TAILLE)
10090 LET TAILLE = TAILLE + 1
10100 LET TEST$ = « »
10110 REM INSÉRER ICI APPEL DE *MODNOM*
10120 REM
10130 REM
10140 REM
10150 RETURN
```

Variantes de basic

SPECTRUM

Étant donné que le Spectrum a la possibilité de sauvegarder ou de charger des tableaux entiers à l'aide de l'instruction SAVE-DATA, le sous-programme LECFCR sera complètement différent. Lorsque nous commencerons à écrire les données, nous publierons une version complète des sous-programmes correspondant à cette machine. En attendant, à titre d'exercice, ceux qui possèdent un Spectrum peuvent s'attaquer au problème qui consiste à créer le fichier factice contenant @PREMIER ; il faut également déterminer combien il y a d'articles valables dans le tableau, en lisant le fichier.

INKEY$

Voir « Variantes de basic ».

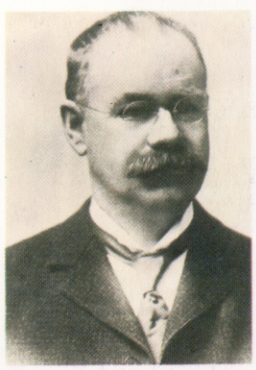
OPEN CLOSE

Voir « Variantes de basic ».

Jeux de cartes

Hermann Hollerith et James Powers ont tous les deux développé des machines à cartes perforées. Leur rivalité dura six décennies.

James Powers
Les machines de Powers étaient destinées à une seule application. Il s'est néanmoins révélé un rival sérieux d'Hollerith.



Hermann Hollerith
Hollerith inventa la « liseuse » de cartes électromécanique qui devint plus tard la machine à cartes ou tabulatrice. (Cl. IBM.)

Les machines que Hermann Hollerith inventa pour dépouiller les résultats du recensement de 1890 aux États-Unis évoluèrent vers une gamme d'appareils de traitement d'informations à utilisations variées, connus sous le nom de « machines à cartes perforées » ou « tabulatrices ». Jusqu'à l'apparition des premiers ordinateurs commerciaux dans les années cinquante, les machines à cartes perforées favorisèrent le développement de l'industrie et des affaires. A Pittsburgh, dans les années trente, un grand magasin expérimenta un système de comptes clients pour lequel deux cent cinquante terminaux disposés partout dans le magasin étaient reliés par lignes téléphoniques à des tabulatrices regroupées dans une même pièce. Le prix des marchandises était écrit sur des étiquettes perforées, et cette information était envoyée automatiquement aux tabulatrices qui ensuite enregistraient la vente et préparaient une facture pour le client, après diverses vérifications.

En fait, la concurrence dans les affaires a favorisé le développement des tabulatrices. L'exclusivité dont jouissait Hollerith pour la fourniture des équipements électromécaniques destinés au service de recensement poussa James Powers à fournir d'autres machines. Powers proposa un système de tabulatrices entièrement mécaniques. La rivalité entre les deux hommes, puis entre les deux sociétés qu'ils créèrent par la suite, stimula la croissance de machines de traitement de l'information.

En 1902, Hollerith dessina un tableau de connexions (ressemblant aux premiers standards téléphoniques) permettant de choisir les colonnes de la carte perforée à additionner et à sortir. Ainsi, la machine de Hollerith avait une possibilité de programmation qui faisait défaut aux machines inventées par son rival. En 1924, Powers fit breveter une méthode de représentation des données alphanumériques sur une carte perforée utilisant une seule perforation dans chaque colonne pour représenter un nombre et une combinaison de perforations pour représenter une lettre. Hollerith réagit rapidement en créant son propre système, la carte standard à quatre-vingts colonnes. Chaque colonne de cette carte comprenait douze lignes de perforations « lues » par des balais métalliques dont le contact avec une plaque de métal située sous la carte fermait un circuit électrique. Certains systèmes plus modernes utilisaient une cellule photoélectrique au lieu d'un balai.

Les premières tabulatrices servaient uniquement à compter ou à additionner, mais plus tard elles furent dotées de fonctions mathématiques plus avancées permettant de traiter les données. D'autres applications de ces nouvelles machines sont vite apparues. Des tabulatrices spéciales furent créées pour l'utilisation des tables de calcul, pour l'analyse des ondes et pour l'astronomie — ce sont des tabulatrices qui ont permis d'identifier Pluton en 1930. Les tabulatrices devinrent finalement suffisamment sophistiquées pour traiter en chaîne de grandes masses de données — IBM en a fait breveter une qui pouvait suivre les mouvements de dix mille comptes en banque.

Les machines à cartes

A leur apogée, dans les années cinquante, ces machines comprenaient huit unités différentes.
Les données étaient mises sur carte à l'aide d'une « perforatrice » qui pouvait traiter deux cents cartes à l'heure. Une « vérificatrice » contrôlait les fautes de frappe, et lorsque les cartes vieillissaient, une reproductrice en perforait de nouvelles.
La tabulatrice, elle, accumulait les totaux des données chiffrées dans les colonnes et sortait les résultats à une cadence de neuf mille cartes à l'heure. Cette tabulatrice était souvent reliée à une calculatrice mécanique qui fournissait des fonctions mathématiques plus complexes. L'« interclasseuse » avait pour tâche de comparer les informations contenues dans deux paquets de cartes perforées ou de regrouper les paquets. Enfin, la « trieuse » pouvait ventiler le paquet de cartes en treize piles différentes — une pile pour chacune des douze perforations et une pile pour les colonnes vierges.
Le fonctionnement de la tabulatrice était commandé par des caractères de contrôle dans les positions 11 et 12. Les cartes de contrôle étaient de couleur vive pour être facilement repérables dans une pile. Lorsque la tabulatrice rencontrait une carte de contrôle, elle commençait une nouvelle opération : par exemple, elle se mettait à compter dans une zone différente.
(Cl. BBC Hulton.)

PROGRAMME N° 8

JEU DE DÉS

Nous avons déjà abordé le rôle des fonctions RND et INT.
Utilisons à nouveau ces instructions et mêlons-les au graphique. Pour cela, nous allons simuler le jet de deux dés : un bleu et un vert.
Tapez le programme suivant :

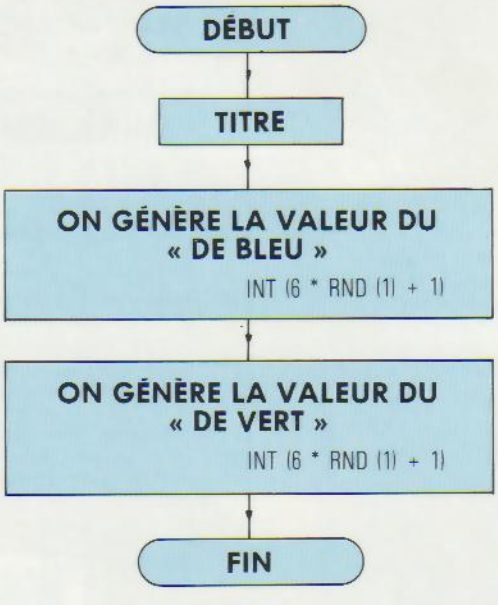
```
§LIST

 5 HOME
10 PRINT "SIMULATION D'UNE PAIRE DE DES"
20 PRINT : PRINT : PRINT "DE BLEU :",
30 PRINT INT (6* RND (1) + 1)
40 PRINT : PRINT : PRINT "DE VERT :",
50 PRINT INT (6* RND (1) + 1)
60 END
```

```
§RUN
SIMULATION D'UNE PAIRE DE DES

DE BLEU :                1

DE VERT :                5

§RUN
SIMULATION D'UNE PAIRE DE DES

DE BLEU :                4

DE VERT :                4

§RUN
SIMULATION D'UNE PAIRE DE DES

DE BLEU :                6

DE VERT :                4

§RUN
SIMULATION D'UNE PAIRE DE DES

DE BLEU :                5

DE VERT :                5
```

En 10, le titre.
En 20, deux sauts de ligne : ? : ? : et on fait écrire à la machine « DE BLEU ».
En 30 ? INT (6 * RND (1) + 1).

RND génère un nombre décimal aléatoire compris entre 0 et 1. Je multiplie ce nombre par 6 et j'y ajoute 1 (pour un dé, le 0 est une valeur qui ne convient pas). Enfin, je prends la partie entière (INT) du nombre généré.
Lignes 40 et 50, mêmes choses pour le dé vert.

ORGANIGRAMME



Supprimons la ligne 60 et ajoutons une petite conclusion à ce programme grâce à un test IF ... THEN.

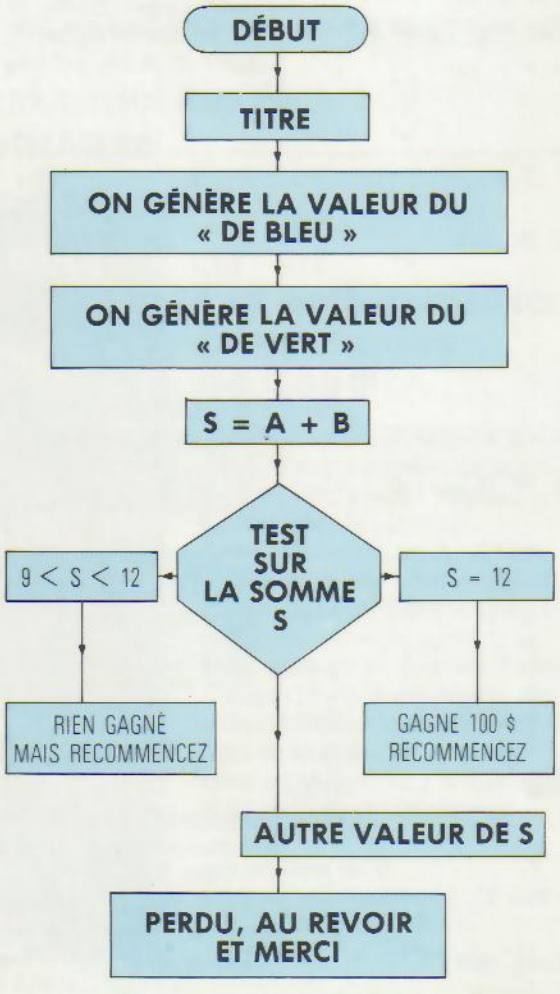
Tout d'abord, affectons un nom A et B aux deux valeurs affichées par les dés.

Tapez alors :

Programmes : D. Le Goff - S.I.I.

```
§LIST

 5  HOME
10  PRINT "SIMULATION D'UNE PAIRE DE DES"
20  PRINT : PRINT : PRINT "DE BLEU :",
25  A = INT (6* RND (1) + 1)
30  ? A
40  PRINT : PRINT : PRINT "DE VERT :",
45  B = INT (6* RND (1) + 1)
50  ? B
60  S = A + B : ? : ?
70  IF S = 12 THEN FLASH :
    ? "VOUS AVEZ GAGNE 100 $
    RECOMMENCEZ !" :
    NORMAL : GOTO 10
80  IF S>9 AND S<12 THEN FLASH :
    ? "RIEN GAGNE, MAIS RECOMMENCEZ !"
    NORMAL : GOTO 10
90  FLASH : ? "PERDU...PERDU...PERDU..."
100 ? "AU REVOIR ET MERCI" : NORMAL
110 END
```

L'ORGANIGRAMME EST LE SUIVANT :



KALÉIDOSCOPE

Lions tout ceci au graphique et créons des points de couleur d'une manière aléatoire.

```
§LIST

10  HOME
20  GR
30  REM CREATION D'UNE COULEUR ALEATOIRE
40  COLOR = INT (16)* RND (1)
50  REM CREATION D'UN POINT ALEATOIRE
60  X = INT (40* RND (1))
70  Y = INT (40* RND (1))
80  REM TRACAGE DU POINT
90  PLOT X,Y
100 REM RENCORE
110 GOTO 40
```

40 génère une couleur d'une manière aléatoire.

INT (16) * RND (1) + 1

génère un nombre aléatoire compris entre 0 et 1 qui est multiplié par 16 dont on prend la partie entière.
60 et 70 génèrent les coordonnées d'un point (X et Y) d'une manière aléatoire.

Rappel : l'écran GR (40 col., 40 lignes)
90 trace le point
110 on recommence
Stoppez l'exécution par CTRL - C RETURN simultanément.

ORGANIGRAMME