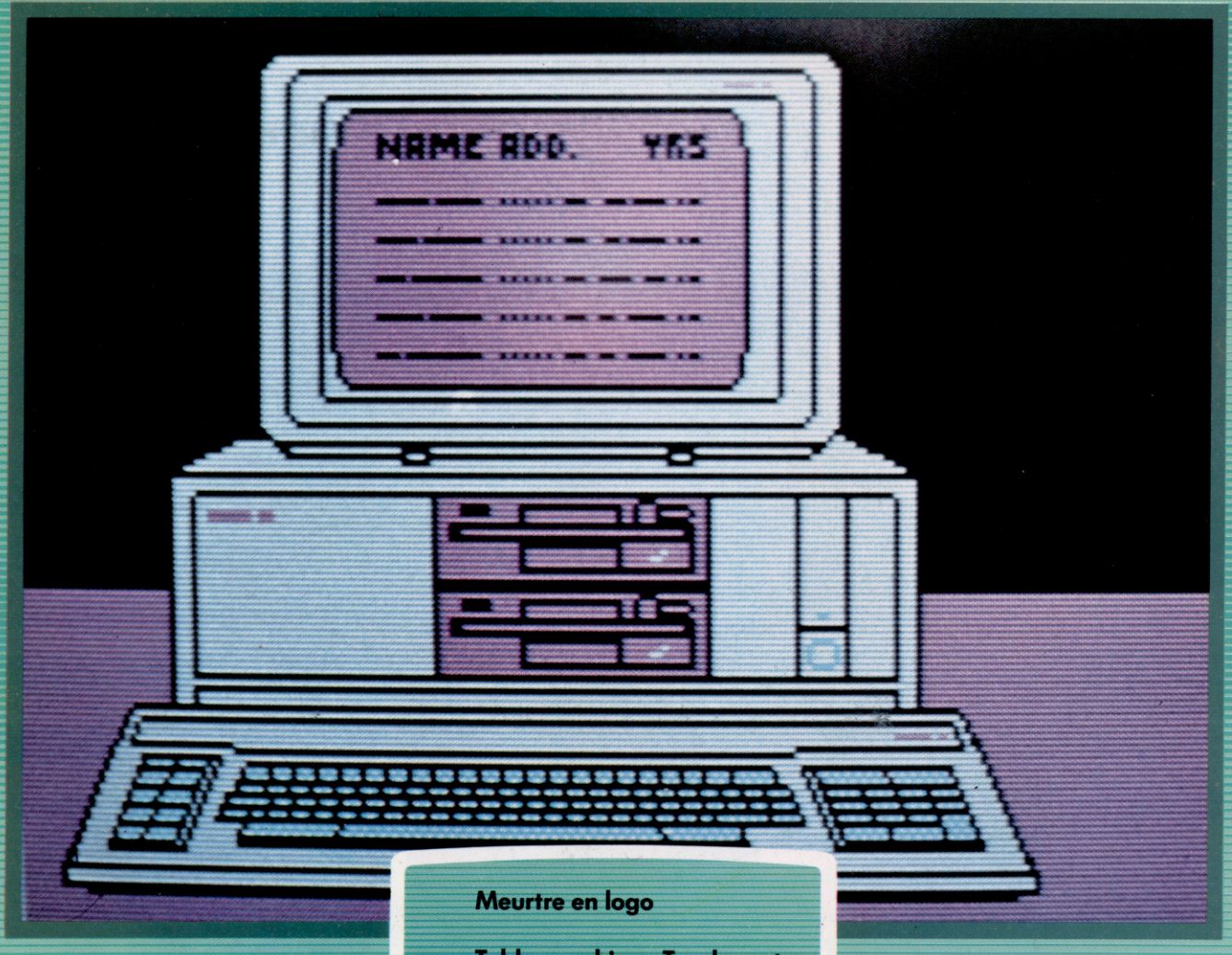
ABC INFORMATIQUE

N°66

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



- Meurtre en logo
- Table graphique Touchmaster
- Modèles de robots
- Langage d'assemblage (fin)

EDITIONS ATLAS

M 6062-66-12,50 F 90 FB - 3,80 FS - $ 1.95

Robots modèles?

Nous allons examiner certains des produits mis en vente sous le nom de robots. Nous verrons s'ils remplissent bien le rôle d'un robot et s'ils satisfont à la définition très précise que nous en avons donnée.

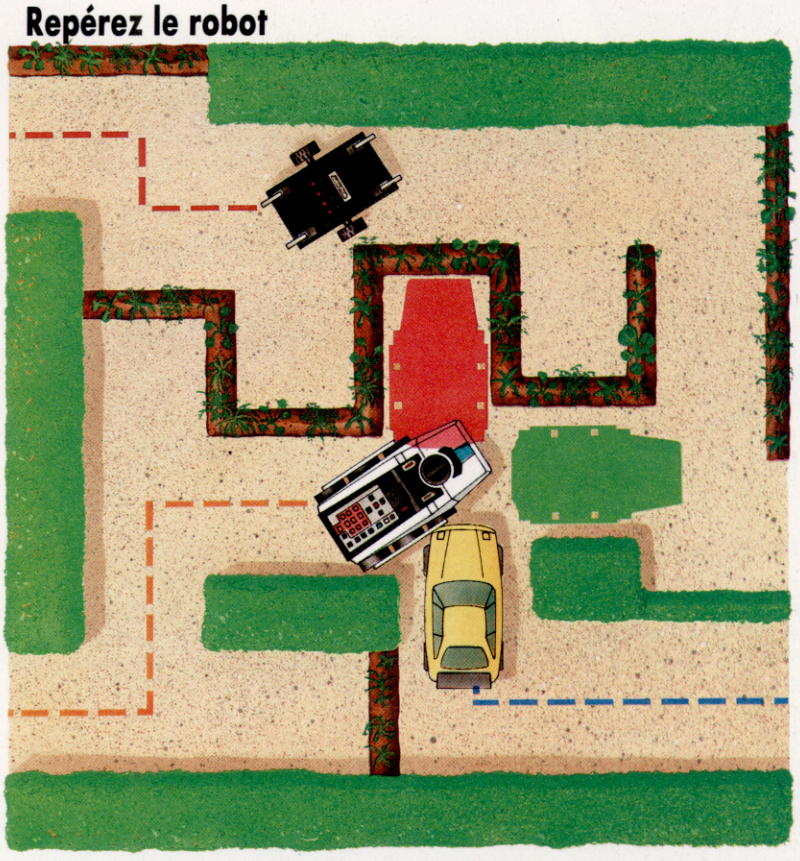
Après la théorie, la pratique. Dans la réalité, les nombreuses caractéristiques propres au robot ne sont jamais mises en œuvre ou se trouvent limitées pour des raisons de coût, de complexité des pièces mécaniques. En outre, la présence d'un logiciel performant les rend souvent superflues. Les robots réels, destinés à un usage industriel ou domestique, ont tendance à ne plus correspondre à l'image que nous nous sommes fait d'un robot. Un robot a des capteurs pour voir, entendre et sentir. Mais les « sensations » qu'il éprouve n'ont jusqu'à présent aucune signification pour lui. Elles ne peuvent du reste être synthétisées dans le but de susciter au robot un comportement propre, non programmé. Robbie le robot et ses homologues imaginaires sont encore très éloignés de la réalité.

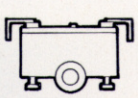
De nombreux produits sont néanmoins vendus comme « robots », depuis les petits jouets bon marché, jusqu'aux très coûteux robots industriels. Que trouve-t-on sur le marché?

Le premier critère élimine de nombreux « robots » de prix très bas. Il s'agit de la capacité à se déplacer de manière autonome. Il n'est pas question de demander au robot de s'autoprogrammer ou de décider d'une suite d'actions sans être guidé. Mais nous pouvons attendre d'un robot que, une fois mis en route, il soit capable de fonctionner indépendamment d'un contrôle humain permanent. Sans cette liberté de mouvements, un objet ne peut être considéré comme un robot.

Ayant passé avec succès le test du mouvement, notre candidat robot doit maintenant être jugé sur la manière dont le mouvement a lieu. Une petite voiture peut être dotée d'un moteur et de piles qui la font avancer toujours en ligne droite. Si vous ajoutez des pare-chocs, le jouet peut rebondir lorsqu'il rencontre un obstacle comme un mur ou une table. Si en outre vous le dotez d'un centre de gravité inhabituel et de grosses roues de caoutchouc, il pourra même monter aux

Repérez le robot





Notre robot
Alimenté et contrôlé par son ordinateur-maître, le robot est équipé de capteurs tactiles et photosensibles.

Big Track
Il est possible de programmer cet engin piloté par un microprocesseur incorporé, en lui donnant des instructions par l'intermédiaire du clavier.

Auto tamponneuse
Ce petit jouet alimenté par piles va toujours tout droit jusqu'à un obstacle; il tourne alors de 90° et reprend sa course.

Curieuses traces
Les trois véhicules essaient de venir à bout du labyrinthe : la petite voiture ne fait que se heurter aux murs; Big Track suit les instructions programmées par son pilote; le robot prend connaissance du labyrinthe par ses capteurs et son logiciel. Nous pouvons être sûrs qu'en fin de compte le robot aura résolu l'énigme du dédale, quelles que soient les vicissitudes rencontrées. Big Track suivra scrupuleusement son programme et pourra réussir si ses instructions sont bonnes. La petite voiture pourrait réussir si les labyrinthes tournaient toujours à droite, et encore, par hasard. Lorsque la petite voiture entre en collision avec Big Track, elle n'est pas perturbée dans sa course puisque sa route est aléatoire, mais Big Track est détourné de sa trajectoire de 90° (selon la trajectoire verte). Il continue alors son chemin comme s'il était toujours sur la bonne voie (trajectoire rouge). Les deux engins réagissent de manière inintelligente, alors que le robot le considère simplement comme un aspect supplémentaire de son environnement imprévisible.
(Cl. Steve Cross.)

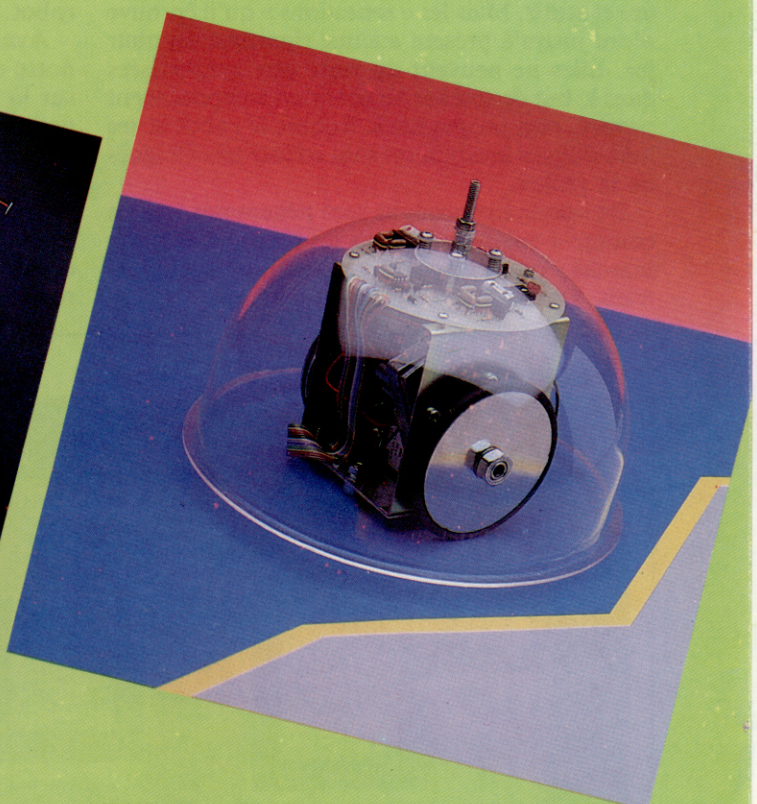
murs et se retourner pour poursuivre sa route. La petite voiture qui se déplace d'elle-même sans le contrôle de l'homme peut-elle être appelée robot? La réponse est évidemment non, mais l'explication est cruciale pour la compréhension des robots.

Comme nous l'avons vu, il existe deux catégories de mouvements pour un robot : le mouvement simple sous le contrôle d'un programme, et le mouvement « intelligent ». Dans les deux cas, la clé est la programmation. Notre petite voiture n'est pas un robot, puisqu'elle ne peut être programmée pour changer de direction. Elle ne dispose que d'un jeu d'instructions qui lui est incorporé et dont elle ne peut varier. Elle ne répond pas à des commandes humaines et ne dispose pas d'un moyen par lequel on pourrait lui demander d'effectuer des mouvements différents. Une variante intéressante, à cet égard, est le jouet à moteur, voiture ou camion, ou autre objet susceptible d'être programmé en lui indiquant une démarche à suivre par l'intermédiaire d'une carte perforée. La carte est lue mécaniquement et le véhicule suit ses indications, tournant à droite ou à gauche, ou encore continuant tout droit. Selon la définition donnée précédemment, il s'agit d'un robot, puisque cet objet peut être programmé par cartes. Le critère de mouvement programmable élimine bon nombre de produits bon marché qui se réclament de l'appellation « robot ».

Plusieurs autres critères peuvent rentrer en ligne de compte : l'objet dispose-t-il de capteurs pour lui fournir des données sur le monde extérieur? Peut-il répondre à son environnement et créer un modèle interne variable? Sait-il bien jouer aux échecs? Tous ces éléments d'appréciation peuvent être appliqués à notre robot mais, en fin de compte, seul le critère de mouvement programmable est essentiel. Les produits suivants sont disponibles dans le commerce.

NOM : Beasty
TYPE : Bras de robot
PROGRAMMABLE : Oui
LOGICIEL FOURNI : Oui
CAPTEURS-RETOUR D'INFORMATION : De position
DISTRIBUTEUR : Commotion

Beasty reçoit ses instructions du port utilisateur du BBC Micro, et est alimenté par l'alimentation auxiliaire du BBC. Il est fourni en kit et comprend trois servomoteurs. Il est livré avec deux manuels, une notice de montage et un manuel utilisateur.
Beasty dispose de son propre système d'exploitation — Robol est son nom —, ce qui permet l'utilisation indépendante de chacun des servomoteurs. Nous avons déjà fourni diverses informations à ce sujet.



NOM : Hebot II
TYPE : Robot au sol
PROGRAMMABLE : Oui
AVEC LOGICIEL : Oui
CAPTEURS-RETOUR D'INFORMATION : Tactile
DISTRIBUTEUR : Powertan Cybernetics

Hebot II est une tortue-robot pilotée par deux moteurs en continu connectés à deux roues. La tortue est reliée au connecteur de côté du ZX81 Sinclair; le fabriquant affirme que, avec une légère redistribution de la câblerie, il est possible de le faire fonctionner sur n'importe quel micro. Il est livré en kit et accompagné d'une notice de montage. La tortue a un crayon rétractable et quatre détecteurs de collision. L'alimentation est fournie directement par l'ordinateur.

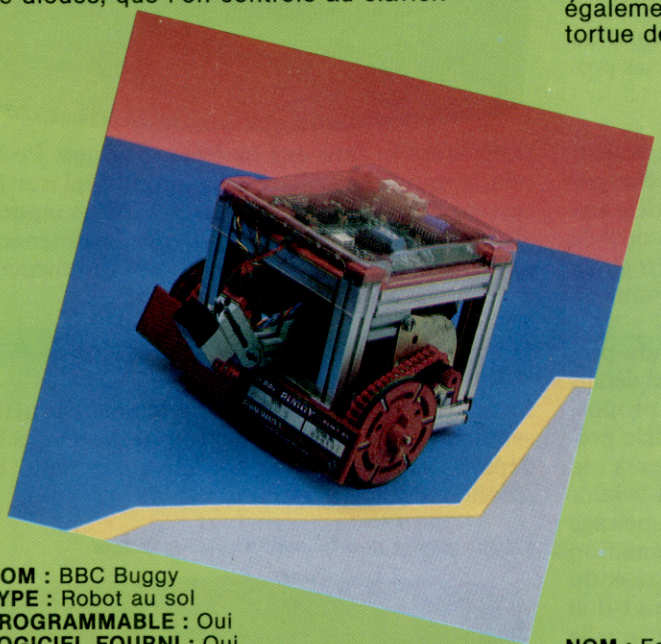

Chris Stevens

NOM : Memocon Crawler
TYPE : Robot au sol
PROGRAMMABLE : Oui
LOGICIEL FOURNI : Oui
CAPTEURS-RETOUR D'INFORMATION : Non
DISTRIBUTEUR : Prims Products

C'est le robot le plus sophistiqué de la gamme Prist Movits. La machine utilise des piles de 1,5 V pour alimenter deux moteurs électriques en continu. Le pilotage se fait par l'intermédiaire d'une boîte dotée de cinq touches, une par commande disponible. Cette boîte est reliée à Crawler par un câble-nappe. Cet engin dispose aussi d'un signal sonore et de diodes, que l'on contrôle au clavier.

NOM : Valiant Turtle
TYPE : Robot au sol
PROGRAMMABLE : Oui
LOGICIEL FOURNI : Oui
CAPTEURS-RETOUR D'INFORMATION : De position
DISTRIBUTEUR : Valiant designs

Valiant Turtle a la forme d'une tortue. Elle est pilotée par deux moteurs, un par roue. Le dispositif est contrôlé depuis l'ordinateur par un rayon infrarouge. Le logiciel fourni est conçu pour fonctionner avec LOGO bien qu'il puisse être exploité sans le langage de commande. L'alimentation est fournie par une batterie intégrée rechargeable. Il existe également un porte-crayon qui permet à la tortue de dessiner en avançant.



NOM : BBC Buggy
TYPE : Robot au sol
PROGRAMMABLE : Oui
LOGICIEL FOURNI : Oui
CAPTEURS-RETOUR D'INFORMATION : Capteur photosensible
DISTRIBUTEUR : Economatics Education

Le BBC Buggy est une tortue contrôlée par logiciel qui est déjà familière aux écoliers anglais. Il est livré en kit et branché au port utilisateur du BBC Micro, l'alimentation étant fournie par l'alimentation auxiliaire du micro. Le Buggy utilise un couple de moteurs à impulsions qui permet des déplacements très précis. Il peut réagir à la lumière et rechercher une source lumineuse. Il peut également recevoir un crayon et un lecteur de codes à barres.


Chris Stevens

NOM : Edinburgh Turtle
TYPE : Robot au sol
PROGRAMMABLE : Oui
LOGICIEL FOURNI : Oui
CAPTEURS-RETOUR D'INFORMATION : Oui
DISTRIBUTEUR : Jessop Acoustics

Tortue portant le nom de la ville où elle a vu le jour. Elle est reliée à l'ordinateur par un câble-nappe dont elle tire également son alimentation. Elle est pilotée par deux moteurs électriques en continu, à raison d'un par roue. Elle est dotée d'un crayon rétractable et d'un haut-parleur de bord. Jessop a récemment mis au point une version à télécommande.

Deviner à coup sûr

Nous terminons notre exposé sur TK! Solver — programme de traitement d'équations pour l'Apple II, l'IBM PC et ses compatibles, l'Apricot d'ACT.

Comme nous l'avons déjà vu, TK! Solver est un progiciel d'une nouvelle génération qui introduit le concept de tableur dans le domaine des mathématiques poussées et de l'ingénierie. Nous allons traiter ici de cette propriété exclusive de TK!, l'*itération*. Il s'agit d'une méthode par laquelle le programme peut trouver une variable par estimation. On détermine normalement les valeurs des variables lorsque l'on possède suffisamment d'informations sur leur contexte. Le programme réduit le problème à une suite de calculs. Ainsi :

$A^2 + B^2 = 2COS\ Y$

peut être facilement résolu pour chacune des variables, pourvu que dans chaque cas les deux autres valeurs soient connues. Avec des valeurs pour A et B, Direct Solver de TK!. exécuterait les calculs voulus et donnerait un résultat pour Y.

Il y a pourtant des circonstances où la détermination d'une valeur ne peut s'effectuer directement, comme, par exemple, une équation redondante, qui définit une variable dans ses propres termes. Par exemple :

D = (A + B) / (2* D)

dans un modèle dont A est la seule valeur connue. D'autres difficultés peuvent survenir par suite d'un modèle incomplet ou d'un modèle aux nombreuses variables interdépendantes et doté de peu de données. Le concept d'itération est particulièrement complexe, aussi prendrons-nous un exemple plus simple.

Reprenons le cas du trajet en voiture du premier article sur TK! Ajoutons-lui quelques détails afin qu'il s'applique mieux à notre propos. Comme vous vous en souvenez, le modèle précédent était construit autour de cinq valeurs : la distance, le temps, la vitesse, l'essence et le kilométrage. Le modèle devait calculer le kilométrage à partir de la vitesse et de la consommation d'essence; la distance à partir de la vitesse et du temps; et d'autres variantes. Qu'en sera-t-il si nous voulons maintenant déterminer quelle vitesse observer pour effectuer un trajet dans les limites d'un budget déterminé?

Il nous faut d'abord ajouter plusieurs facteurs au modèle. Ce dernier doit, par exemple, prendre en considération la puissance développée par la voiture, les frottements internes au moteur et la résistance de l'air. Tous ces facteurs ont un effet sur le kilométrage et sur la vitesse (nous supposons que les frottements internes sont constants). Il faut également ajouter la limite supérieure pour le budget et le coût de l'essence.

Commençons à construire le modèle en entrant les équations dans le Tableau des règles, à raison d'une équation par ligne. Elles sont automatiquement lues dans le Tableau des variables. Parce qu'il a plusieurs variantes, l'écran est trop petit pour contenir toutes les informations. Pour les afficher toutes, nous pouvons faire apparaître le

Construire les équations

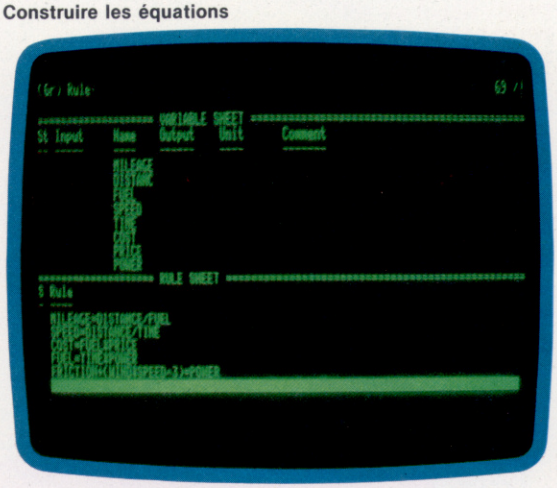


Tableau des variables dans une fenêtre spécifique. Pour cela, appuyons sur la touche ; pour déplacer le curseur dans la fenêtre à variables. Nous tapons alors F1 (Fenêtre 1). Nous voyons maintenant toutes les variables et nous pouvons commencer à saisir des valeurs à leur intention.

Résolution directe

Le modèle peut être résolu directement, si suffisamment d'informations lui sont données dès le départ. Entrez les valeurs suivantes dans la colonne Saisie :

Valeurs entrées pour le mode résolution directe

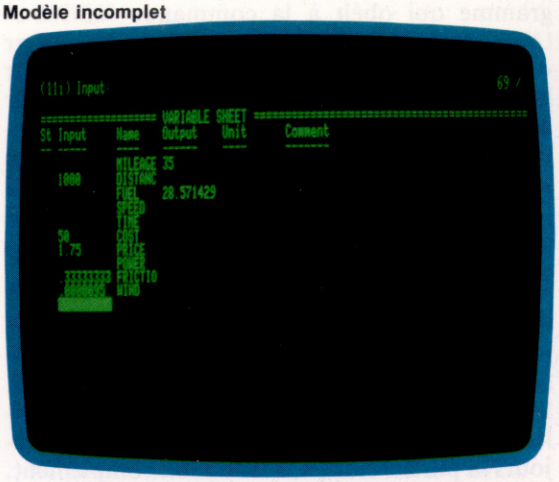
Ian McKinnell

Faites ensuite ! pour déclencher le calcul. TK! affiche le message Résolution Directe et les valeurs recherchées apparaissent dans la colonne Résultats de la sorte :

Résultats pour le mode résolution directe

Cela nous donne bien les informations recherchées. Mais que se passerait-il si nous voulions utiliser ce même modèle pour résoudre un problème avec moins d'informations ? Soit le calcul suivant : avec un budget maximum de 50 livres pour l'essence destinée à un voyage de 1 000 miles. Nous connaissons le prix de l'essence (par exemple 1,75 livre par gallon); nous pouvons facilement calculer combien dépenser au mile. En revanche, il est plus difficile de déterminer à quelle vitesse conduire pour parcourir le kilométrage voulu dans les limites du budget. Commençons par effacer les valeurs déjà présentes. Tapons RVO (Réinitialiser Variables Oui). Entrons ensuite l'information dont nous disposons : 100 pour la distance, 50 pour le coût et 1,75 pour le prix. Nous utilisons une valeur de 1/3 pour les frottements internes et 0,0000095 pour la résistance de l'air. Faisons ! pour demander le calcul; les valeurs s'affichent :

Modèle incomplet



Vous remarquez qu'il n'y a pas de valeurs pour la vitesse, le temps et la puissance — la vitesse est la valeur recherchée. Si nous faisons passer l'affichage du Tableau des variables au Tableau des règles, trois des quatre équations restent incomplètes (ce que nous indiquons par * dans la colonne Statut) :

Équations incomplètes

Résolution par itération

Puisque nous ne pouvons résoudre le problème en Résolution Directe, essayons la Résolution par Itération. Elle prend une valeur de départ pour l'équation. Si elle s'avère erronée, TK!Solver effectue des approximations successives.

Nous commençons par prendre la valeur du kilométrage obtenue précédemment, pour la mettre dans la colonne des entrées afin de donner à TK! une valeur supplémentaire de départ. Faisons ! dans la colonne « Statut » à côté du kilométrage dans le « tableau des variables ». Effectuons ensuite une estimation sur la vitesse, par exemple 50, tapons ce nombre dans la colonne des entrées, puis D pour « Deviner » dans la colonne « Statut », et ! pour « Calculer ». TK! affiche dans le haut de l'écran Résolution par Itération, ainsi que le compte des itérations. TK! a trouvé les valeurs exactes pour la vitesse, le temps et la puissance, au quatrième essai :

Valeurs successives d'itération

Selon TK!, une vitesse moyenne d'exactement 47 miles à l'heure est nécessaire pour effectuer le trajet sans excéder les limites budgétaires fixées. Plus l'estimation d'origine est proche de la réalité, plus vite TK! trouvera la solution.

TK!Solver est disponible pour l'Apple II, l'IBM PC et ses compatibles, et l'Apricot de ACT. Software Arts publie en outre un livre de modèles tout prêts pour TK!

Aux ordres

Dans notre projet de jeu d'aventures, nous regardons ici comment le programme analyse et obéit aux instructions qui lui sont données par le joueur.

Les jeux d'aventures sont généralement construits de telle sorte que le joueur puisse se déplacer d'une position à l'autre, prenant et déposant des objets sur sa route. Un ensemble de commandes lui permet de mener à bien ces tâches simples. Les commandes que nous avons utilisées sont :

ALLER (direction)	Se déplacer entre les positions
PRENDRE (objet)	Ramasser un objet
POSER (objet)	Déposer un objet
LISTER	Recenser les objets emportés
EXAMINER	Afficher à nouveau la description de la position courante
FIN	Mettre fin au jeu

Il peut y avoir des variantes telles que SE DÉPLACER au lieu de ALLER ; RAMASSER au lieu de PRENDRE... Cela fait partie du jeu. Ainsi, un joueur peut essayer la commande NAGER alors qu'il se trouve sur la terre ferme. Si le programme réagit en lui disant qu'il ne peut pas nager « là où il se trouve », le joueur pourra en déduire qu'il existe bien des endroits où cela est possible (à moins qu'il s'agisse d'un piège du programmeur).

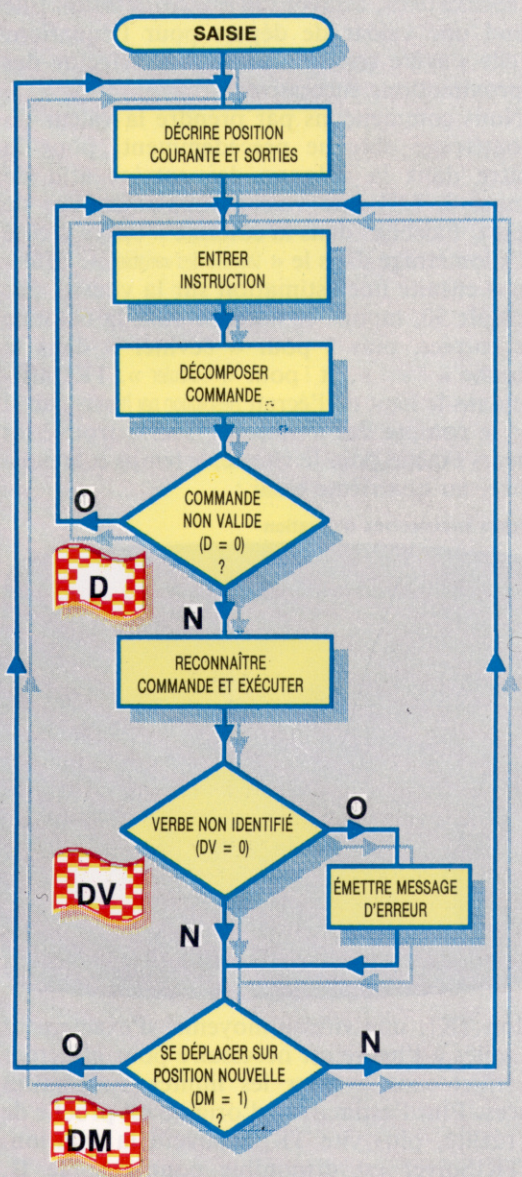
Le nombre de commandes d'un jeu dépend de la complexité et du soin apportés par le programmeur à couvrir toutes les éventualités. La chose la plus importante est de faire en sorte que le programme n'avorte pas lorsqu'une commande non prévue est entrée. Une routine de sécurité qui affiche « Je ne comprends pas » pourra éventuellement suffire si l'on a pris soin de programmer les commandes avec souplesse afin que les joueurs puissent les entrer de plusieurs manières. Il serait fastidieux, par exemple, qu'un programme qui obéit à la commande PRENDRE LA LAMPE, réponde à la variante PRENDRE LAMPE par « Je ne comprends pas ». Avant d'approfondir le problème de la souplesse, nous allons définir le type d'instructions que l'on peut donner pendant le jeu, et mettre au point une routine qui transformera la commande en une forme facilement interprétable.

Subdivision des commandes

Quelle que soit l'instruction, elle sera probablement exprimée sous forme d'un ordre tel que ALLER AU SUD VERS LA RIVIÈRE ou TUER L'ÉTRANGER. L'avantage de cette structure est qu'elle est facilement décomposable. Le verbe commence toujours la phrase et il est suivi de son complément. A la fin, il pourra y avoir une forme de qualification de l'action. La première étape de l'analyse d'une commande est de séparer le verbe du reste. Cette tâche est facilement réalisable en passant en revue tous les caractères de la phrase avec MID$, jusqu'à rencontrer un espace. La partie de

Drapeaux à damiers

Les drapeaux sont très largement utilisés pour les programmes de construction modulaire. Les expressions conditionnelles qui supposent des embranchements peuvent être testées dans un module, mais l'embranchement sur le résultat ne se fera que lors d'un retour à la partie principale du programme. En affectant une variable à une valeur prédéterminée lors du test, on peut vérifier ultérieurement à l'intérieur du programme principal que cette valeur préalable a été modifiée. Les variables utilisées pour signaler l'état d'une expression conditionnelle sont appelées « drapeaux ». Cet organigramme montre la boucle principale du programme pour la Forêt hantée, dans son état de développement. Le drapeau D indique si une commande est au bon format, et est attribué au sous-programme commande de décomposition. Le sous-programme utilisé pour identifier et exécuter les commandes normales, utilise deux drapeaux. DV signale si la partie verbe de la commande a été reconnue. Le déplacement du joueur sur une nouvelle position lors de l'exécution d'une commande est signalé par DM. Lorsque ce dernier est lu dans la boucle principale du programme, la valeur 1 indique qu'il convient de terminer la boucle afin de décrire la nouvelle position. (Cl. Ian McKinnell.)



la phrase qui est à gauche de l'espace constitue le verbe et peut être assignée à la variable VB$. La partie de la phrase venant à sa droite est assignée à une seconde variable, NM$.

Ce sous-programme est utilisé dans la Forêt hantée pour décomposer l'instruction assignée à la variable IS$:

```
2500 REM **** COMMANDE DE DECOMPOSITION S/R ****
2510 IF IS$ ="LISTER" OR IS$="FIN" THEN VB$=IS$:IF=1:RETURN
2515 IF IS$ ="EXAMINER" THEN VB$=IS$:IF=1:RETURN
2520 F=0
2530 LS=LEN(IS$)
2540 FOR C=1 TO LS
2550 A$=MID$(IS$,C,1)
2560 IF A$<>" " THEN 2590
2570 VB$=LEFT$(IS$,C-1):F=1
2580 NN$=RIGHT$(IS$,LS-C):C=LS
2590 NEXT C
2600 :
2610 IF F=1 THEN RETURN
2620 PRINT:PRINT"IL ME FAUT AU MOINS DEUX MOTS"
2630 RETURN
```

Avant de diviser la phrase, la routine s'assure d'abord que la commande n'est pas l'une des trois instructions portant sur un seul mot, c'est-à-dire LISTER, EXAMINER et FIN. S'il s'agit de l'une de ces commandes, l'instruction en entier est assignée à VB$ et la routine s'achève. Dans le cas contraire, la routine passe par une boucle FOR...NEXT et entreprend de rechercher le premier espace. Deux techniques utilisées à l'intérieur de cette boucle méritent un intérêt tout particulier. Elles partent du principe qu'effectuer un branchement conditionnel à l'extérieur d'une boucle FOR...NEXT sans passer par l'instruction NEXT relève d'un mauvais style de programmation. Aussi est-il préférable pour signaler le fait qu'une condition a été remplie de mettre un drapeau D à 1. La deuxième technique consiste à arrêter le balayage de la phrase lorsque l'espace a été trouvé.

La boucle peut alors être arrêtée de manière satisfaisante en mettant le compteur d'itération C à sa limite supérieure, LC. Le programme passe ainsi à l'instruction suivante lorsqu'il rencontre à nouveau l'instruction NEXT au lieu de boucler sur FOR. Une fois la boucle finie, l'état du drapeau peut être consulté. Une valeur de 1 signifie que la phrase comporte plus d'un mot, il ne reste alors plus qu'à retourner à la boucle principale. Si le drapeau n'est pas à 1, la commande ne contient qu'un seul mot et n'est pas une des commandes dont la présence a été testée auparavant. Un message, indiquant que la syntaxe demande au moins d'écrire deux mots, est alors affiché avant de revenir à une nouvelle commande.

Commandes normales

Selon la partie principale du programme, le joueur se déplace simplement d'une position à l'autre pour prendre ou poser des objets. Pour la majorité des positions, les commandes ALLER, PRENDRE, POSER, LISTER, EXAMINER et FIN (et leurs variantes) sont suffisantes. Ce n'est que dans des circonstances exceptionnelles que le joueur devra avoir recours à des commandes plus spécialisées. Il est de peu d'intérêt, par exemple, d'utiliser la commande TUER lorsqu'il n'y a rien à tuer. Nous pouvons réduire à six le nombre des commandes de référence pour la majorité des cas (déplacements et manipulations d'objets). Lorsqu'un joueur passe sur une nouvelle position, le programme la teste afin de déterminer s'il s'agit d'une des positions spéciales. Dans l'affirmative, toute nouvelle spécification de commande sera traitée par un sous-programme de commande particulier à cette position. Aussi, la boucle principale d'appel de notre programme devra remplir les fonctions suivantes :

1. Décrire la position et lister les sorties.
2. Déterminer s'il s'agit d'une position spéciale.
3. Demander une commande, et, si la position n'est pas « spéciale », passer en revue la liste des commandes normales.

Il doit également y avoir dans la boucle principale une caractéristique spécifique pour reconnaître une commande qui suscite un déplacement sur une nouvelle position. Dans ce cas, la boucle doit revenir à son début afin de décrire la nouvelle position et décider s'il s'agit d'une position spéciale. Sinon, il suffit de finir la boucle et d'attendre une nouvelle commande. La mise en œuvre la plus simple pour cette caractéristique est d'utiliser un drapeau de mouvement, DM, normalement mis à zéro. Lorsqu'une commande suppose un déplacement, le drapeau prend la valeur 1. L'état de DM est lu à la fin de la boucle principale, le branchement approprié étant fait. Ajoutez les lignes suivantes à la Forêt hantée :

```
270 GOSUB2500:REM DECOMPOSER L'INSTRUCTION
275 IF F=0 THEN 260:REM INSTRUCTION NON VALIDE
280 GOSUB3000:REM COMMANDES NORMALES
290 IF DV=0 THENPRINT:PRINT"JE NE COMPRENDS PAS"
300 IF DM=1 THEN 240:REM NOUVELLE POSITION
310 IF DM=0 THEN 260:REM NOUVELLE INSTRUCTION

3000 REM **** COMMANDES NORMALES S/P ****
3010 DV=0:REM DRAPEAU VERBE
3020 IF VB$=ALLER OR VB$="SE DEPLACER" THENDV=1:GOSUB3500
3030 IF VB$="PRENDRE" OR VB$="RAMASSER"THENDV=1:GOSUB3700
3040 IF VB$="DEPOSER" OR VB$="POSER"THENDV=1:GOSUB3900
3050 IF VB$="LISTER" OR VB$="INVENTAIRE"THENDV=1:GOS
     UB4100
3055 IF VB$="EXAMINER" THEN DV=1:MF=1:RETURN
3060 IF VB$="FIN" OR VB$="TERMINE" THEN DV=1:GOSUB4
     170
3070 RETURN
```

Un autre drapeau, DV, est utilisé dans la première routine pour indiquer si le verbe a été compris et exécuté. DV ne prend la valeur 1 que lorsque le verbe a été isolé. Nous pouvons ajouter le message de sécurité « Je ne comprends pas » dans la boucle principale en testant le statut de VF. S'il reste à zéro, cela signifie que le verbe de la commande n'a pas été reconnu par la routine d'analyse, et le message est affiché.

Nous verrons ultérieurement des sous-programmes pour prendre, lâcher et lister les objets. Pour l'instant, ajoutons un court sous-programme de commande FIN à notre ensemble de commandes normales :

```
4170 REM **** METTRE FIN AU JEU S/P ****
4180 PRINT:PRINT"ETES VOUS SUR(O/N ?"
4190 GET A$: IF A$  "O" AND A$  "N" THEN 4190
4200 IF A$="N" THEN RETURN
4210 END
```

La commande EXAMINER est également directe. Pour décrire à nouveau la position courante, il suffit de mettre le drapeau mouvement, DM, à 1 et revenir à la boucle principale. Pour le

programme principal, l'effet sera de revenir au début, appelant ainsi les routines qui décrivent une position et ses sorties. Comme la valeur de la variable position, P, n'est pas changée par la commande EXAMINER, la même position est décrite. Elle est utile lorsque la description de la position courante a disparu après que le joueur a effectué plusieurs actions.

Ajouter de la souplesse

Lors de la frappe des commandes de mouvements, le joueur peut entrer une même instruction sous plusieurs formes. Par exemple, ALLER AU NORD, SE DÉPLACER AU NORD et ALLER VERS LE NORD disent la même chose. Bien qu'il ne soit pas vital pour un tel programme de reconnaître toutes ces formes, cela rend le jeu plus intéressant lorsque plusieurs formats sont légaux pour une même instruction. Les trois commandes de mouvements que nous avons données ont une structure commune : elles commencent par un verbe, la description de la direction étant un mot propre au joueur. Il est possible alors d'écrire une routine qui cherche la partie de la phrase venant après le verbe et indique la direction. La routine passe en revue ces lettres, à la recherche d'espaces, isolant tout à tour chaque mot pour le comparer aux quatre mots de direction.

```
3630 REM **** RECHERCHE D'UNE DIRECTION S/P ****
3640 NN$=NN$+" ":LN=LEN(NN$):C=1
3645 FOR I=1 TO LN
3650 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
3655 O$=MID$(NN$,C,I-C):C=I+1
3660 IF O$="NORD" OR O$="EST" THEN NN$=OS:1=LN
3665 IF O$="SUD" OR O$="OUEST" THEN NN$=O$:1=LN
3670 NEXT I
3675 RETURN
```

Dans le dernier article, nous avons écrit une routine de mouvement. Pour lui ajouter la précédente routine, ajoutons la ligne suivante :

```
3505 GOSUB3630:REM RECHERCHE D'UNE DIRECTION
```

Il convient de noter que cette routine n'obéit pas à des commandes du genre ALLER VERS LA DIRECTION NORDIQUE, par exemple. Il serait possible de concevoir une routine qui examine des groupes successifs de quatre ou cinq lettres, les comparant aux quatre mots de direction. Elle supposerait néanmoins un temps d'exécution très long. Cependant, la place du mot de direction peut résoudre le problème : ALLER NORDIQUE sera ainsi pris en compte puisque la routine de mouvement utilise la première lettre de la deuxième partie de la phrase, NN$. Dans cet exemple, N de NORDIQUE est accepté comme le N de NORD.

Variantes de basic

Spectrum :
Utilisez I$ pour IS$, B$ pour VB$, et R$ pour NN$, et ce, d'un bout à l'autre. Remplacez les lignes suivantes de Digitaya :

```
1790 LET A$=I$(C TO C)
1800 IF A$="" THEN LET
     B$=I$(TO C-1):LET R$=I$
     (LEN(I$)-LS+C+1 TO):LET
     F=1:LET C=LS
2630 LET A$=INKEY$:IF A$<>"O"
     AND A$ <>"N" THEN 2630
8630 IF R$(1 TO 1)<>" " THEN
     NEXT I:RETURN
8640 LET W$=R$(C TO I-1):LET
     C=I+1
```

Remplacez les lignes de Forêt hantée

```
2550 LET A$=I$(C TO C)
2570 LET B$=I$(TO C-1):LET F=1
2580 LET R$=I$(LEN(IS)-LS+C+1
     TO):LET C=LS
3650 IF R$(1 TO 1)<>" " THEN
     NEXT I:RETURN
3655 LET W$=R$(C TO I-1):LET
     C=I+1
4190 LET A$=INKEY$:IF A$<>O AND
     A$<>"N" THEN GOTO 4190
```

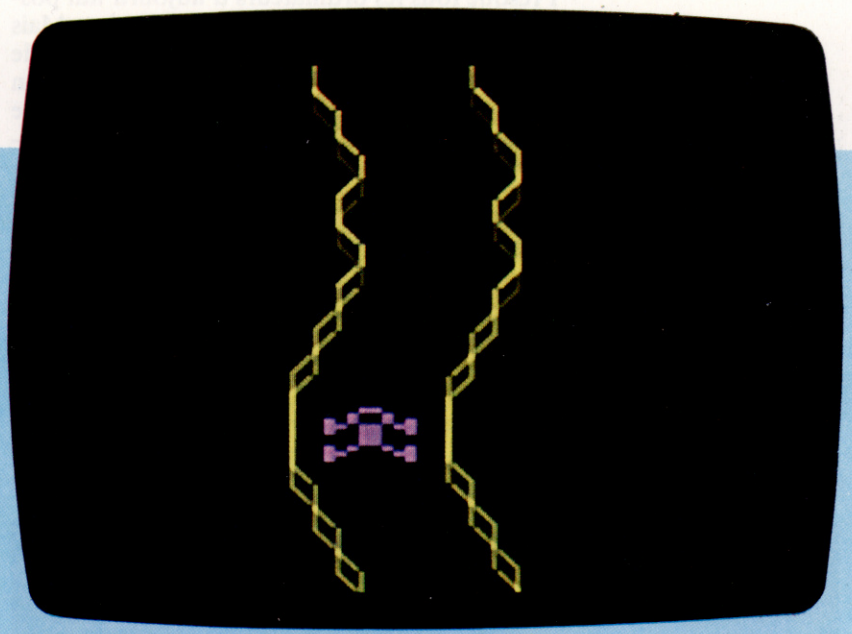
Listages Digitaya

```
1220 GOSUB1700:REM ANALYSER INSTRUCTIONS
1125 IF D=0 THEN 1210:REM INSTRUCTION NON VALIDE
1230 GOSUB 1900:REM INSTRUCTIONS NORMALES
1240 IF DV=0 THENPRINT"JE NE COMPRENDS PAS"
1250 IF DM=1 THEN 1160:REM NOUVELLE POSITION
1260 IF DV=0 THEN 1210:REM NOUVELLE INSTRUCTION


1700 REM **** ANALYSER L'INSTRUCTION S/P ****
1705 D=0:REM DRAPEAU A ZERO
1710 IFIS$="FIN OR IS$="LISTER THEN VB$=IS$:D=1:
     RETURN
1720 IF IS$="EXAMINER" THEN VB$=IS$:D=1:RETURN
1730 :
1740 REM ** DECOMPOSER L'INSTRUCTION **
1750 VB$="":NN$="":REM PAS DE VERBE NI DE NOM
1770 LS=LEN(IS$)
1780 FOR C=1TO LS
1790 A$=MID$(IS$,C,1)
1800 IF A$=" " THEN VB$=LEFT$(IS$,C-1):NN$=DROITE$(I
     S$,LS-C):D=1:C=LS
1810 NEXT
1830 IF D=0 THEN PRINT:PRINT "IL ME FAUT AU MOINS DEUX
     MOTS"
1840 RETURN
1850 :
1900 REM **** ACTIONS NORMALES S/P ****
1910 DS=0
1920 PRINT
1930 IF VB$="GO"ORVB$=DEPLACER"THENDV=1:GOSUB2000
1940 IF VB$="TAKE"ORVB$="PRENDRE"THEN DV=1:GOSUB2140
1950 IF VB$="DROP"ORVB$="POSERTHENDV=1:GOSUB2360
1960 IF VB$="LIST"ORVB$="INVENTAIRE"THENDV=1:
     GOSUB2540
1965 IF VB$="EXAMINER" THEN DV=1:DM=1:RETURN
1970 IF VB$="FIN"ORVB$="FINISH"THENDV=1:GOSUB2610
1980 RETURN

2015 GOSUB8600:REM RECHERCHE DE LA DIRECTION

2610 REM **** FIN DU JEU S/P ****
2620 PRINT:PRINT"ETES VOUS SUR" (O/N) ?"
2630 GETA$:IFA$<>"D"AND A$<>"N"THEN2630
2640 IFA$="N" THEN RETURN
2650 END

8600 REM **** RECHERCHE DE LA DIRECTION S/P ****
8610 NN$=NN$+" ":LN=LEN(NN$):C=1
8620 FORI=1 TO LN
8630 IF MID$(NN$,I,1,<>" " THEN NEXT 1:RETURN
8640 O$=MID$(NN$,C,I-C):C=I+1
8650 IF O$="NORD" OR O$="EST" THEN NN$=O$:I=LN
8660 IF O$="SUD" OR O$="OUEST" THEN NN$=O$:I=LN
8670 NEXT I
8680 RETURN
```

Grand prix 2

Voici la deuxième partie d'un jeu dont nous avons déjà donné un programme pour votre ordinateur Atari. N'oubliez pas que les manettes de jeu sont obligatoires.

Pour conduire votre bolide, utilisez le manche à balai. Si vous réussissez à atteindre l'arrivée, vous aurez un bonus et la valeur de la distance parcourue sera doublée.



```
10 REM ** GRAND PRIX 2 - P.BUNN **
15 PI=5:GOSUB 3000:ACCIDENT=1000
20 X2=14:X=116:S=PEEK(106)-8:Q=S*256:FOR
 N=Q+512 TO Q+640:POKE N,0:NEXT N:POKE 5
4279,S
30 POKE 559,46:POKE 53248,X:POKE 704,216
:POKE 704,70:POKE 53256,1
40 FOR N=Q+552 TO Q+561:READ A:POKE N,A:
NEXT N
50 DATA 129,195,165,24,24,153,219,165,36
,24
60 GRAPHICS 0:SETCOLOR 2,0,0:POKE 53277,
3:POKE 559,46
65 POKE 752,1:POKE 53278,A
66 FOR P=0 TO 23:POKE 201,X2:? ,S2$:NEXT
 P
70 S=STICK(0):X=X+(S=7)*2:X=X-(S=11)*2:P
OKE 53248,X
72 SOUND 0,40,10,15:SC=SC+PI:SOUND 0,0,0
,0
75 IF O THEN O=0:Z=2:GOTO 110
80 A=PEEK(53770)
90 IF A>85 AND A<170 THEN Z=2
100 IF A>170 AND X2<15 THEN Z=3:O=1
105 IF A<85 AND X2>2 THEN X2=X2-1:Z=1:O=
1
110 POKE 201,X2
115 IF A=192 AND  NOT I THEN ? ,F$:I=1:G
OTO 145
120 IF Z=1 THEN ? ,S1$
130 IF Z=2 THEN ? ,S2$
140 IF Z=3 THEN ? ,S3$:X2=X2+1
145 Y=PEEK(53252):IF Y<>0 AND I THEN GOT
O 2000
150 IF Y<>0 THEN GOTO ACCIDENT
160 GOTO 70
1000 REM ** ACCIDENT!!!! **
1010 N=INT(RND(0)*10):POKE Q+552+N,PEEK(
53770):SOUND 0,RND(0)*20+20,80,15:POKE 7
04,PEEK(53770)
1020 IF PEEK(53770)<240 THEN 1010
1030 POKE 53248,0:? :? "VOTRE SCORE:";SC
:? :? :? "TAPEZ UNE TOUCHE.":POKE 764,25
5
1035 SOUND 0,0,0,0
1040 IF PEEK(764)=255 THEN 1040
1050 RUN
2000 FOR Y=0 TO 255:POKE 710,Y:NEXT Y
2005 ? CHR$(125);"      SCORE :";SC:POKE
710,0
2010 ? "     BONUS  :1000":B=1000
2020 FOR G=1 TO 1000 STEP 10:B=B-10:SC=S
C+10:POSITION 14,0:? SC:POSITION 14,1:?
B;"  ":SOUND 0,G/4,10,10:NEXT G
2030 SOUND 0,0,0,0
2040 ? :PI=PI*2:? "1 KM VAUT ";PI;" POIN
TS"
2045 RESTORE :I=0
2050 GOTO 20
3000 DIM F$(8),S1$(8),S2$(8),S3$(8)
3010 RESTORE 3000:FOR P=1 TO 8
3020 READ A,B,C,D
3030 F$(P,P)=CHR$(A)
3040 S1$(P,P)=CHR$(B)
3050 S2$(P,P)=CHR$(C)
3060 S3$(P,P)=CHR$(D)
3070 NEXT P
3080 RESTORE :RETURN
3090 DATA 160,6,22,7,198,32,32,32
3100 DATA 201,32,32,32,206,32,32,32
3110 DATA 201,32,32,32,211,32,32,32
3120 DATA 200,32,2,32,160,6,32,7
```

Toucher du doigt

Bien des périphériques ont été inventés pour faciliter la création graphique. Le double avantage de la table graphique Touchmaster est qu'elle fonctionne avec la plupart des ordinateurs les plus répandus.

Presque tous les ordinateurs d'aujourd'hui possèdent un mode graphique haute résolution. Mais en faire réellement usage exige beaucoup de temps et de patience, sauf peut-être si l'on dispose d'un logiciel spécialisé. Un simple programme de dessin n'est d'ailleurs pas suffisant : l'usager, en effet, cherche moins à griffonner sur l'écran qu'à recopier une image déjà existante.

C'est même dans ce but que plusieurs dispositifs de type numérique ont été lancés sur le marché, mais ils sont généralement conçus pour un appareil spécifique. L'intérêt de Touchmaster est que cette tablette fonctionne avec de nombreux ordinateurs (bien qu'il faille parfois disposer d'une interface appropriée). Elle est même censée tenir lieu de clavier, mais sa simplicité de construction limite dans les faits ce type d'emploi à un choix entre différents menus, ou au contrôle de jeux simples.

Pour entrer des données, comme pour charger le logiciel qui accompagne Touchmaster, on aura besoin de toutes les façons d'un vrai clavier d'ordinateur qu'elle ne peut remplacer.

Touchmaster
Il faut d'abord presser l'une des commandes (à droite sur la tablette), puis se mettre à dessiner, à l'aide d'un doigt, d'un crayon ou d'un stylet. Le résultat est automatiquement affiché à l'écran. (Cl. Ian McKinnell.)

Deux interfaces

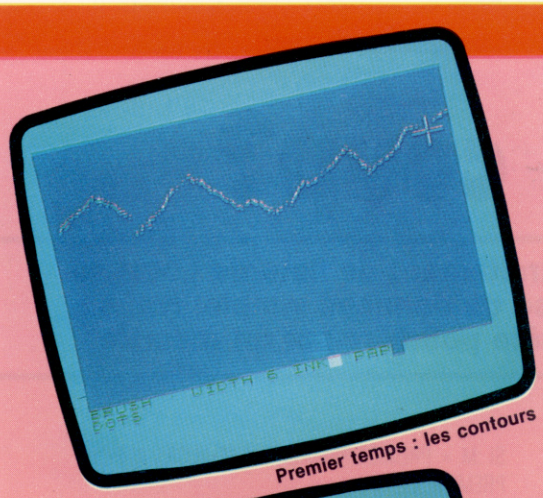
Ce périphérique est présenté dans un boîtier gris très réussi (350 × 330 × 35 mm). La partie arrière est légèrement surélevée pour faciliter le dessin. Un transformateur est fourni, et une diode rouge indique si l'appareil est, ou non, sous tension ; mais il n'y a pas de commutateur pour l'allumer ou l'éteindre. Deux interfaces installées à l'arrière (une série, une parallèle) permettent le raccord sur de nombreux ordinateurs. On remarque également une prise sur laquelle on peut installer un commutateur actionné par le pied. Mais elle n'est pas mentionnée dans la documentation, qui reste très insuffisante : si le manuel explique comment procéder aux connexions et fournit plusieurs programmes en BASIC pour lire les coordonnées, il manque de précision.

Touchmaster possède un clavier à membrane assez proche de ceux du ZX81 et du Spectrum. La résolution maximale est de 256 × 256 pixels. Un film électrosensible est séparé de la surface de la tablette par un isolant. Un microprocesseur placé dans la tablette guette tout contact sur l'une des deux couches (dans une direction particulière pour chacune d'elles). Toute pression les met en contact par l'intermédiaire de l'isolant ; l'appareil peut donc lire les coordonnées du point de jonction, et les transmettre à l'ordinateur.

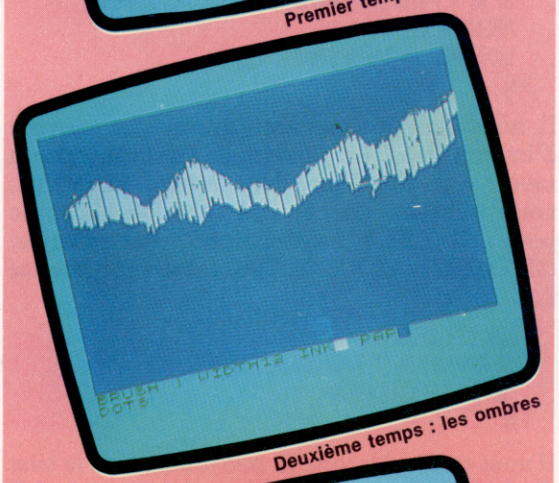
Il faudra utiliser le port parallèle pour le Spectrum, le Vic-20, le Commodore 64 et le Dragon. N'oublions pas que la résolution est souvent plus réduite que celle qu'on obtient sur les écrans munis pour la plupart d'un mode d'affichage haute résolution.

Multipaint

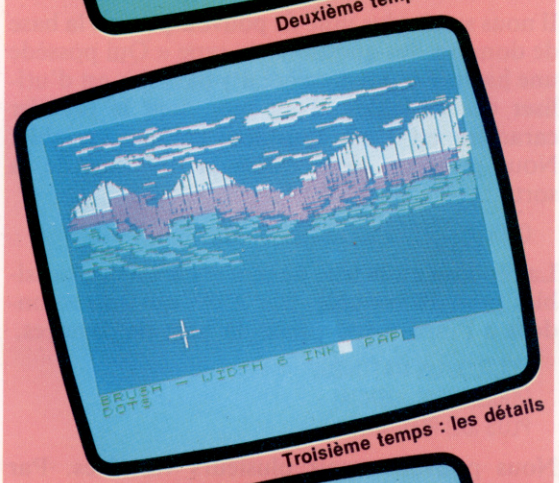
C'est un logiciel graphique fourni avec la tablette. Il donne une bonne idée des possibilités de l'ensemble, mais il est difficile d'y voir une véritable aide à la création. Une feuille de plastique donne un menu des différentes options : celle qui est choisie est affichée en bas de l'écran. On peut faire usage de cinq « pinceaux » différents, dont la largeur va de 2 à 32 pixels.



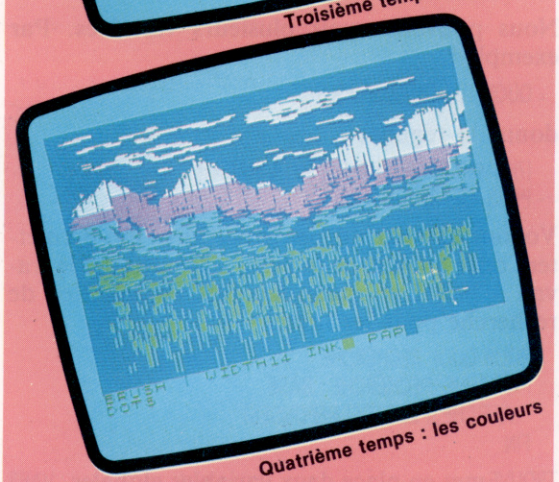
Premier temps : les contours



Deuxième temps : les ombres



Troisième temps : les détails



Quatrième temps : les couleurs

Une scène touchante

Touchmaster peut être utilisé comme tablette graphique à l'aide du logiciel Multipaint et d'une feuille en plastique posée sur la « table à dessin ». Voici les différentes étapes de création d'une image en couleurs.

Il est possible de dessiner par points, par taches, ou au gré de sa fantaisie. Les couleurs du « papier » et de l'« encre » peuvent être définies au départ. Tout cela est, là encore, affiché en bas de l'écran.

Une fois les premiers choix accomplis, l'usager pourra créer des rectangles, des cercles, des polygones ou des lignes. Un stylet est fourni avec l'appareil, mais il est préférable de se servir d'un doigt. Cette tablette graphique est assez large et a sur le Koala Pad l'avantage de la précision : toute pression est réellement convertie en coordonnées spatiales correctes, et non plus approximatives !

Malheureusement, Multipaint demeure assez rudimentaire. L'option FILL existe bel et bien, mais (en tout cas sur le Spectrum) ne fonctionne pas comme on s'y attendrait. Il n'y a pas moyen d'agrandir et d'éditer telle ou telle partie du dessin, et les couleurs, une fois définies, ne peuvent plus être modifiées. Sur le Spectrum, où il est plus facile de dessiner en noir et blanc et de colorier ensuite, c'est un sérieux handicap.

Connectable à divers ordinateurs

Du point de vue du matériel, la tablette Touchmaster offre bien des avantages par rapport à Grafpad ou Koala Pad. Elle est solide, assez grande (21 × 29,7 cm pour la zone de dessin), et peut être connectée à divers modèles d'ordinateurs. Vous pourrez donc vous en servir encore si d'aventure vous changez de machine ; il vous faudra simplement une interface et le logiciel d'accompagnement.

Il est très regrettable que la documentation et le programme d'assistance graphique ne soient pas à la hauteur. Le constructeur propose déjà différents logiciels, mais il est encore trop tôt pour dire si Touchmaster connaîtra vraiment le succès.

TOUCHMASTER

PRIX

★ ★ ★

DIMENSIONS

350 × 330 × 35 mm.

INTERFACES

Série ou parallèle, selon le type d'ordinateur utilisé ; on peut aussi recourir au clavier propre à Touchmaster.

DOCUMENTATION

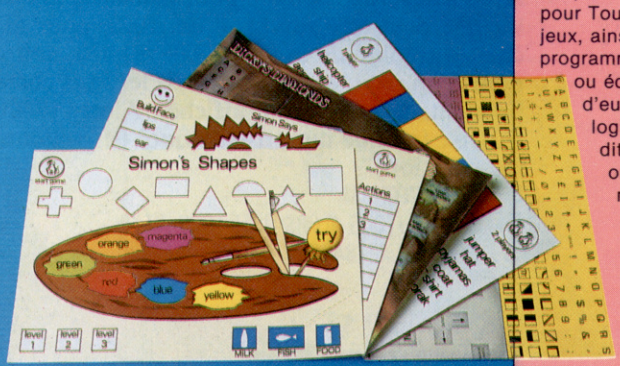
Le manuel est correct et donne les renseignements de base, mais une information plus détaillée aurait été la bienvenue.

FORCES

Touchmaster est un périphérique aux grandes capacités d'adaptation, grâce à ses interfaces et aux programmes avec lesquels elle tourne.

FAIBLESSES

Médiocrité de certains programmes, manuel insuffisant, prix trop élevé.



Interrogations écrites

On peut acheter pour Touchmaster divers jeux, ainsi que des programmes utilitaires ou éducatifs. Chacun d'eux comporte le logiciel proprement dit pour différents ordinateurs), un mode d'emploi et une feuille de plastique très colorée. En voici quelques-unes.

Qui a fait le coup?

L'utilisation des caractéristiques de traitement de liste de LOGO se poursuit par la constitution d'une base de données simple. Nous avons pris comme exemple une enquête policière sur un meurtre.

Un meurtre horrible a été commis dans une petite communauté des montagnes Ozark. Zachariah a été traîtreusement attaqué avec une hache et tué. Nous savons que Matthieu et Joshua ont tous les deux une hache, Jacques et Ebenezer ont un fusil, et que la cousine Jane a un couteau. Matthieu et Jacques avaient tous les deux du sang sur les mains lorsqu'ils furent interrogés par le shérif du coin. Notre base de données LOGO relative à ce crime va consister en une liste de faits, chacun d'eux étant un *récit* accompagné d'un ou plusieurs noms. Sous forme LOGO, un fait comme « Matthieu possède une hache », s'écrit : « POSSÈDE MATTHIEU HACHE ». Dire que Jacques a du sang sur les mains se traduit par [SANGLANT JACQUES].

Nous commençons notre enquête par une base vide :

```
POUR CONSTITUE
  FAIS « BASE[]
FIN
```

Nous ajoutons ensuite les faits au fur et à mesure que nous les découvrons (pour autant qu'ils ne soient pas déjà dans la base). Nous entrons ainsi dans la base AJOUTE [POSSÈDE JANE COUTEAU], par la procédure AJOUTE :

```
POUR AJOUTE :FAIT
  SI NON APPARTIENT? :FAIT :BASE ALORS
  FAIS « BASE FMETS :FAIT :BASE
FIN
```

La base est bientôt complète :

```
[[SANGLANT MATTHIEU] [SANGLANT JACQUES]
  [TUE ZACHARIAH HACHE] [POSSÈDE MATTHIEU HACHE]
  [POSSÈDE JOSHUA HACHE] [POSSÈDE JACQUES FUSIL]
  [POSSÈDE EBENEZER FUSIL] [POSSÈDE JANE COUTEAU]]
```

Nous utiliserons MONTRES pour consulter la base. MONTRES peut être suivi soit de «TOUS, pour afficher la totalité de la base (TOUS les FAITs), soit du nom d'un récit, auquel cas seuls les faits correspondants à ce récit sont affichés. Ainsi : MONTRES «POSSÈDE indique qui possède quoi (M = Montre, REL, RELation).

```
POUR MONTRE :M
  SI :M = «TOUS ALORS LISTE.TOUS :BASE
  LISTE.REL :M :BASE
FIN

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

POUR LISTE.REL :LISTE
  SI VIDE? :LISTE ALORS STOP
  SI :M = PREMIER PREMIER :LISTE ALORS AFFICHE
  PREMIER :LISTE
  LISTE.REL :M SAUFPREMIER :LISTE
FIN
```

Il nous faut maintenant trouver des moyens d'investigation. Le plus simple est de chercher à savoir si un fait figure dans la base. Pour cela, nous utilisons une procédure EST-CE-QUE, mise pour « EST.CE.QUE le fait est dans la base? ». Par exemple, EST.CE.QUE [POSSÈDE JANE COUTEAU] donnera le résultat OUI.

```
POUR EST.CE.QUE :FAIT
  SI APPARTIENT? :FAIT :BASE AFFICHE «OUI
  SINON AFFICHE «NON
FIN
```

Il nous serait plus utile de pouvoir poser à la base de données des questions du type « Qui possède une hache? ». La manière d'y parvenir est d'utiliser des variables. Tout mot dont le premier caractère est ? est considéré comme variable. Nous pouvons alors écrire la question de la sorte :

```
QUOI [POSSÈDE ?QUELQU'UN HACHE]
```

La réponse est la liste de toutes les valeurs possibles pour la variable ?QUELQU'UN, qui correspondent à l'information dans la base de données.

```
[?QUELQU'UN MATTHIEU]
[?QUELQU'UN JOSHUA]
PLUS (DE) RÉPONSES
```

Nous pouvons avoir plusieurs variables. Par exemple :

```
QUOI [TUE ?HOMME ?OUTIL]
```

donne la réponse :

```
[?HOMME ZACHARIAH] [?OUTIL HACHE]
PLUS (DE) RÉPONSES
```

Voyons une par une les procédures qui permettent cette analyse de la base. QUOI transmet le travail à TROUVE en indiquant BASE comme lieu de recherche pour les faits.

```
POUR QUOI :RECHERCHE
  TROUVE :RECHERCHE : BASE
  AFFICHE [PLUS (DE) RÉPONSES]
FIN
```

TROUVE met en place deux variables globales, VARS et REPS (VARiableS et RÉPonseS). VARS est utilisée pour contenir tous les jeux possibles de valeur pour les variables en question réunies dans la liste REPS.

```
POUR TROUVE :RECHERCHE :DONNÉES
```

```
    FAIS «VARS []
    FAIS «REPS[]
    COMPARE :RECHERCHE :DONNÉES
    AFFICHE :REPS
  FIN
```

COMPARE examine tour à tour chaque fait de la base de données. Lorsque cette procédure rencontre des correspondants, le nouveau jeu de valeurs de VARS est ajouté à REPS avant d'affecter à nouveau VARS à une liste vide. COMPARE poursuit alors sa consultation de la base.

```
  POUR COMPARE :RECHERCHE :DONNÉES
    SI VIDE? :DONNÉES ALORS STOP
    SI CORRESPONDANCE? :RECHERCHE PREMIER :DONNÉES
    ALORS FAIS «REP FMETS :VARS :REPS
    FAIS «VARS []
    COMPARE : RECHERCHE SAUFPREMIER :DONNÉES
  FIN
```

Pour comprendre l'effet de la CORRESPONDANCE?, voyons le cas où les entrées sont [POSSÈDE ?QUELQU'UN HACHE] et [POSSÈDE JOSHUA HACHE]. CORRESPONDANCE? donne le résultat VRAI pour ces faits et attribue [?QUELQU'UN JOSHUA] à VARS. Lorsque les données en entrée sont POSSÈDE ?QUELQU'UN HACHE] et [TUE ZACHARIAH HACHE], on obtient par CORRESPONDANCE? : FAUX.

Les vraies difficultés commencent avec plusieurs variables. VALEUR? est utilisé pour vérifier si la variable a déjà pris dans la base de données cette valeur pour ce fait.

Nous utilisons ici une autre notation pour les expressions conditionnelles. TESTE évalue une expression conditionnelle. Si le résultat est vrai, les actions suivant SIVRAI sont effectuées, sinon, ce sont celles suivant SIFAUX qui le seront.

```
  POUR CORRESPONDANCE? :RECHERCHE :FAIT
    SI TOUT VIDE? : RECHERCHE VIDE? :FAIT ALORS RÉSULTAT
    «VRAI TESTE PREMIER PREMIER :RECHERCHE = ?»
    SI VRAI SI NON VALEUR? PREMIER :RECHERCHE PREMIER
    :FAIT :VARS ALORS RÉSULTAT «FAUX
    SI FAUX SI NON (PREMIER :RECHERCHE = PREMIER :FAIT)
    ALORS RÉSULTAT «FAUX
    RÉSULTAT CORRESPONDANCE? SAUFPREMIER :RECHERCHE
    SAUFPREMIER :FAIT
  FIN
```

Pour comprendre le fonctionnement de VALEUR?, voyons d'abord le cas où les entrées sont ?OUTIL, HACHE et [?HOMME ZACHARIAH]. VALEUR? est destiné à vérifier que la variable ?OUTIL peut prendre la valeur HACHE. Il y a trois possibilités : ?OUTIL a déjà une valeur, qui n'est pas HACHE, et VALEUR? donne en résultat FAUX; ?OUTIL a déjà la valeur HACHE et VALEUR? donne le résultat VRAI; ?OUTIL n'a pas de valeur et reçoit la valeur HACHE, et cette information est ajoutée à VARS, VRAI est affiché.

```
  POUR VALEUR? :NOM :VALEUR :VLISTE
    SI VIDE?:VLISTE ALORS FAIS « VARS LMETS LISTES :NOM
    :VALEUR : VARS RÉSULTAT «VRAI
    TESTE :NOM = PREMIER :VLISTE
    SIVRAI SI :VALEUR =DERNIER PREMIER :VLIST ALORS
    RÉSULTAT «VRAI SINON RÉSULTAT «FAUX
    RÉSULTAT VALEUR? :NOM :VALEUR SAUFPREMIER :VLISTE
  FIN
```

AFFICHEL affiche seulement les éléments constitutifs de ANS les uns sous les autres.

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

Recherches plus complexes

Notre enquête pourtant ne progressera plus, à moins qu'on puisse poser des questions plus complexes telles que « Quel outil a tué Zachariah, et qui en possède? ». Ce qui s'écrit avec LOGO :

```
  LEQUEL [[TUÉ ZACHARIAH ?OUTIL]
    [POSSÈDE ?SUSPECT ?OUTIL]]
```

LEQUEL accepte maintenant une liste de questions en entrée, et les valeurs obtenues seront celles qui répondent à toutes les questions. Si vous voulez poser une seule question avec cette nouvelle syntaxe pour LEQUEL, la construction à utiliser est :

```
  LEQUEL [[POSSÈDE ?TOUTEFORME COUTEAU]
```

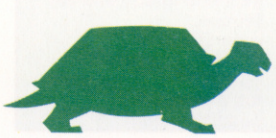
Il suffit d'apporter de légères modifications aux procédures :

```
  POUR LEQUEL :RECHERCHES
    TROUVE :RECHERCHES :BASE
    AFFICHE [PLUS (DE) RÉPONSES]
  FIN
  POUR TROUVE :RECHERCHES :DONNÉES
    FAIS «VARS []
    FAIS «REPS []
    COMPARE :RECHERCHES :DONNÉES
    AFFICHEL :REPS
  FIN
```

La tâche pour COMPARE est maintenant assez ardue. Prenons comme exemple d'entrée [[TUE ZACHARIAH ?OUTIL] [POSSÈDE ?SUSPECT ?OUTIL]]. COMPARE parcourt la base de données, à raison d'un fait à la fois, pour trouver une réponse à la première recherche, et finit par faire correspondre ?OUTIL avec HACHE. La routine passe alors à la deuxième recherche ([POSSÈDE ?SUSPECT ?OUTIL]), à nouveau à partir du début de la base. Une réponse est trouvée pour la deuxième expression conditionnelle avec HACHE comme valeur pour ?OUTIL, et MATTHIEU pour ?SUSPECT. Il n'y a pas d'autres recherches, et c'est bien une solution possible.

Mais nous n'avons pas encore fini. Il se peut qu'il y ait d'autres valeurs qui puissent satisfaire à la deuxième recherche en maintenant HACHE comme valeur pour ?OUTIL. COMPARE interroge donc maintenant la base à partir de l'endroit où elle s'était arrêtée, et trouve bien sûr une deuxième solution pour ?SUSPECT avec JOSHUA. La procédure ne s'arrête pas là et continue de passer la base de données au crible. Cette fois-ci, elle en atteint la fin sans rien trouver d'autre.

La première recherche pouvait cependant avoir d'autres solutions que ?OUTIL et HACHE. Il nous faut donc revenir en arrière jusqu'au point où nous avons trouvé cette première solution, et reprendre la procédure sur le reste de la base. Cette démarche est appelée « remonter en arrière ». Dans le cas présent, il n'existait pas d'autres solutions.



David Higham

Diagramme du meurtre

Le vieux M. Harcourt a été découvert mort, percé de trente coups de ciseaux, à l'arrière d'une camionnette. La police a identifié quatre suspects. L'un a été vu au détour d'une rue, l'autre dans un abri de jardin, le troisième prétend être alors au lit. La personne ayant un rapport avec la camionnette est l'assassin. Les enquêteurs de la police ont mis à jour les faits suivants : 1) Aucun des suspects n'était en possession de son outil professionnel la nuit du meurtre. 2) Le boucher a été vu dans l'abri en train d'ouvrir des lettres avec le scalpel. 3) Un témoin oculaire a confirmé avoir vu l'architecte à un coin de rue, là où le jardinier a retrouvé ses cisailles disparues. 4) La jardinière était au lit et se servait d'un couteau de boucher pour se préparer des sandwichs.

Variantes de logo

Certaines versions LOGO MIT ne comportent pas VIDE? ou APPARTIENT?. Pour toutes les versions LCSI, utilisez VIDEP pour VIDE?, et APPARTIENTP pour APPARTIENT?. ÉGALEP est une primitive qui teste ses deux entrées afin de savoir si ce sont les mêmes. Utilisez-la pour comparer des listes et des mots, à la place du signe égal qui ne fonctionne sur les listes que pour certaines versions LCSI.
La syntaxe de SI pour LOGO LCSI est la suivante :

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

La première liste qui suit l'expression conditionnelle est exécutée lorsque la condition est vraie, la seconde lorsqu'elle est fausse.
LOGO LCSI comporte également la syntaxe TESTE, SIVRAI et SIFAUX pour les expressions conditionnelles.

Afin de savoir où elle en est dans l'affectation de variables, COMPARE empile la valeur courante avant d'utiliser CORRESPONDANCE? (cette dernière pouvant modifier les affectations de variables). Ces valeurs sont ensuite effacées. Voici la procédure dans son entier :

```
POUR COMPARE :RECHERCHES :DONNÉES
  SI VIDE? :RECHERCHES ALORS FAIRE « REPS FMETS :VARS
  :REPS STOP SI VIDE? :DONNÉES ALORS STOP EMPILE
  :VARS
  TEST CORRESPONDANCE? PREMIER :RECHERCHES PREMIER
  :DONNÉES SIVRAI COMPARE SAUFPREMIER :RECHERCHES
  :BASE
  DÉPILE «VARS
  COMPARE :RECHERCHES SAUFPREMIER :DONNÉES
FIN
```

Nous utilisons pour COMPARE une pile qui garde trace des valeurs de VARS, et non une variable temporaire. COMPARE pourrait s'appeler elle-même entre le moment où nous voulons sauvegarder les valeurs et le moment où nous voulons les redistribuer. Aussi, une variable temporaire serait-elle recouverte en écriture par le prochain appel, les valeurs initiales étant alors perdues. La pile sert alors à prévenir cette éventualité. EMPILE met une valeur sur le dessus de la pile, en créant tout d'abord la variable PILE si elle n'existait pas encore.

```
POUR EMPILE :DONNÉES
  SI NON QUELQUECHOSE? ALORS FAIS « PILE []
  FAIS « PILE FMETS :DONNÉES :PILE
FIN
```

DEPILE retire un élément de la pile, et l'assigne en tant que valeur à une variable.

```
POUR DÉPILE :NOM
  FAIS :NOM PREMIER :PILE
  FAIS «PILE SAUFPREMIER :PILE
FIN
```

Nous avons donc les rudiments d'un langage de programmation logique. C'est-à-dire un langage dans lequel nous ajoutons simplement des faits et des règles à une base de données, que nous interrogeons ensuite au moyen de descriptions logiques des données recherchées. Le meilleur exemple actuel d'un langage de programmation logique est PROLOG, mais c'est encore une autre histoire !

Moteur!

Notre robot utilise des moteurs électriques appelés moteurs « pas à pas ». Faisant usage de signaux logiques, ils sont donc adaptés à un contrôle de type numérique.

La construction d'un moteur pas à pas ne ressemble en rien — ou presque — à celle d'un moteur électrique normal. Son mode de fonctionnement est très différent de celui d'un moteur classique, et nous chercherons d'abord à voir comment ils marchent, en partant d'un modèle simplifié.

Dans notre exemple (voir l'encadré « Un pas après l'autre »), on remarque deux bobinages différents (« a » et « b ») installés sur le stator, et quatre pôles électromagnétiques (répartis par paires) sur le rotor. Le moteur dont nous nous servirons pour notre robot est bien sûr plus complexe.

Le gros inconvénient d'un moteur pas à pas, qui est par ailleurs très maniable, est qu'il consommme autant de courant électrique au repos qu'en mouvement. De surcroît, il n'est pas possible de le faire fonctionner à grande vitesse — les bobines qui le composent ne sont pas assez rapides pour cela. Quoiqu'il en soit, aucun de ces problèmes n'a une réelle importance dans notre recherche pour mettre en œuvre par nos propres moyens un robot.

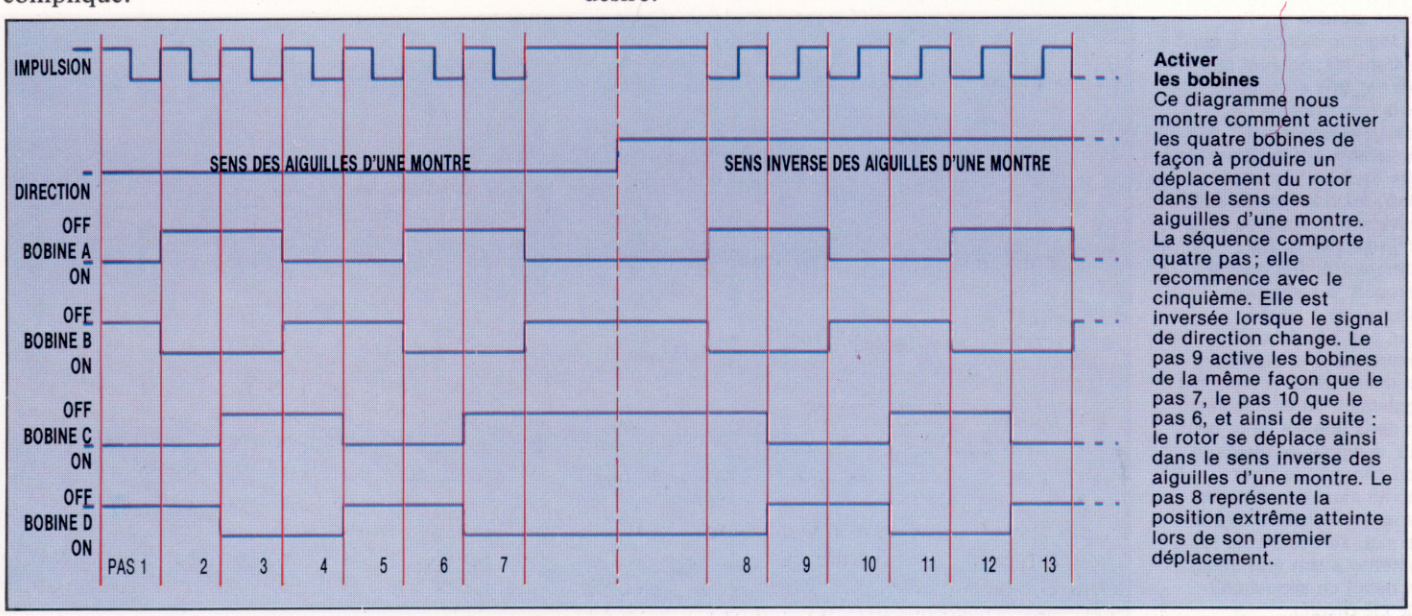
Le modèle simplifié présenté plus loin ne peut tourner que par pas de 45°, et on ne peut contrôler le sens de la rotation. Toutefois, avec le moteur que nous emploierons, un nombre de bobines plus élevé permettra de donner à chaque pas une valeur de 7,5°. Il faudra, bien entendu, que les quatre bobines dont dispose notre moteur soient activées dans un ordre bien défini et assez compliqué.

Tableau d'activation des bobines du stator

Pas	Bob A	Bob B	Bob C	Bob D	
1	ON	OFF	ON	OFF	
2	OFF	ON	ON	OFF	
3	OFF	ON	OFF	ON	
4	ON	OFF	OFF	ON	
5	ON	OFF	ON	OFF	etc.

Un logiciel pourrait se charger de cette tâche en utilisant quatre bits du port utilisateur pour contrôler les quatre bobines; mais le programme serait assez compliqué, et ne pourrait être écrit en BASIC, beaucoup trop lent pour ce type d'application. Il est bien plus simple d'employer une puce spécialisée, la SAA 1027, qui a d'ailleurs pour fonction de guider les moteurs pas à pas. Elle contient les circuits logiques nécessaires pour transmettre des instructions à l'appareil.

Pour que le moteur avance d'un seul pas, il suffit d'une impulsion en provenance du port utilisateur, et d'un signal supplémentaire pour déterminer la direction de la rotation. La puce est capable de détecter tout changement sur trois entrées différentes : une impulsion pour faire tourner le moteur d'un pas, une impulsion de remise à zéro, et un signal de direction qui inverse la séquence d'activation des bobines du stator. Elles sont dirigées vers un compteur bidirectionnel qui les fait parvenir à ces bobines dans l'ordre désiré.



Activer les bobines
Ce diagramme nous montre comment activer les quatre bobines de façon à produire un déplacement du rotor dans le sens des aiguilles d'une montre. La séquence comporte quatre pas; elle recommence avec le cinquième. Elle est inversée lorsque le signal de direction change. Le pas 9 active les bobines de la même façon que le pas 7, le pas 10 que le pas 6, et ainsi de suite : le rotor se déplace ainsi dans le sens inverse des aiguilles d'une montre. Le pas 8 représente la position extrême atteinte lors de son premier déplacement.

La puce est également dotée d'une sortie courant qui peut supporter jusqu'à 500 mW, ce qui permet de la connecter directement au moteur, sans qu'il y ait besoin d'installer des transistors supplémentaires.

La puce elle-même est très complexe, ce qui signifie que le reste du circuit sera très simple; en revanche, il en faudra deux (une par moteur). Le problème est que leur tension est de 12 volts, contre 5 pour le port utilisateur de votre ordinateur. Comprenez par là qu'en logique binaire, l'état zéro correspond pour lui à 0 volt, et l'état 1 à 5 volts. Pour la puce, ces chiffres sont respectivement de 0 et de 7,5 à 12 volts. Il nous faudra donc une interface, sous la forme d'une puce tampon à deux tensions (un pour les entrées, un autre pour les sorties).

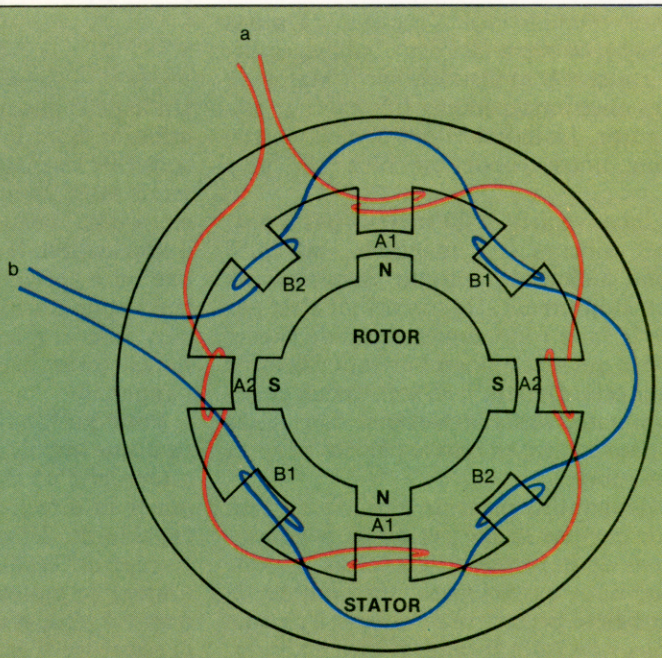
Accessoires

Nombre de pièces
1 puce tampon 40 109
3 fiches DIL 16 broches
2 résistances 100 Ω
2 résistance 0,5 watts 270 Ω
2 capacités 0,1 µF
1 plaquette d'essai (24 bandes × 50 trous)
1 rouleau fil électrique

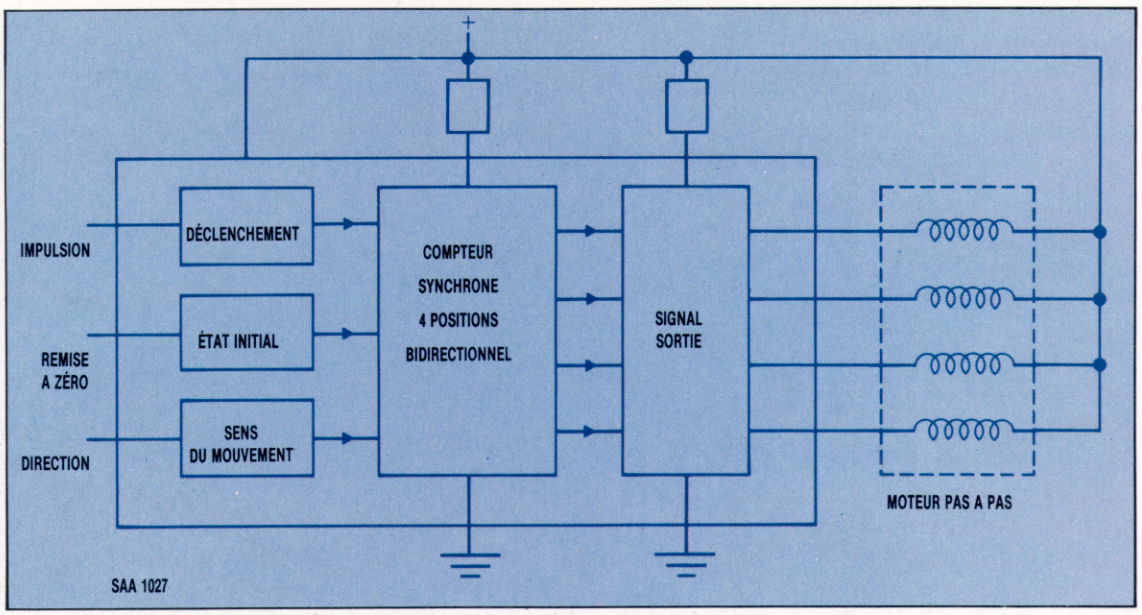
Pièces de rechange
2 puces SAA 1027 de commande de moteurs pas à pas

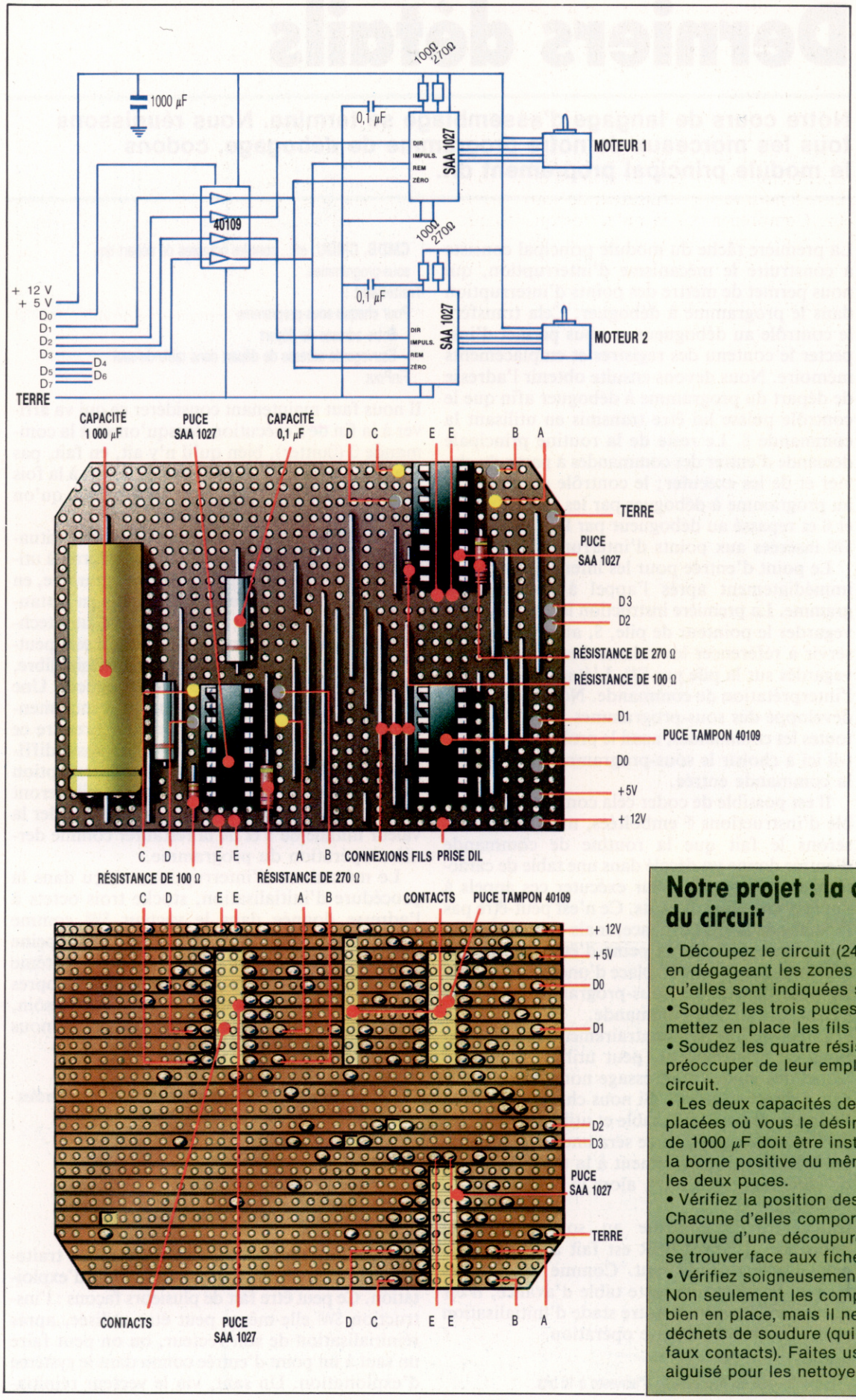
Un pas après l'autre

Ce moteur pas à pas simplifié comporte deux circuits de bobinage sur le stator (« a » et « b »), et un rotor, qui tourne dans le sens des aiguilles d'une montre lorsque les deux circuits sont activés. Notez que A1 et A2 sont bobinés dans des directions opposées. Lorsque « a » est activé, il crée deux pôles sud en A1 et donc deux pôles nord en A2. B1 et B2 sont également bobinés en sens inverse. Notre modèle ne peut tourner que selon un angle de 45°. Mais les moteurs dont nous équiperons notre robot sont plus complexes (davantage de bobines sur le stator, et donc de pôles sur le rotor), ce qui permettra de donner à chaque pas une valeur de 7,5°. (Cl. Kevin Jones.)



Force motrice
La logique des puces qui guident les moteurs pas à pas est très complexe, mais leur principe de fonctionnement est facilement compréhensible. Afin de faire tourner le rotor, les bobines du stator doivent être activées dans un ordre précis. Un compteur bidirectionnel parcourt la séquence nécessaire à raison d'un pas à la fois, en réponse à une impulsion. La même séquence permet également un déplacement en sens opposé, pour peu que le signal de direction soit changé : le rotor tourne alors dans l'autre sens. Un troisième signal permet, au besoin, de le remettre à son état initial en début de séquence. (Cl. Liz Dixon.)





Le diagramme du circuit
Le circuit d'ensemble reste assez simple. Deux puces SAA 1027 permettent en effet de gérer les séquences d'activation indispensables à chaque moteur pas à pas. Mais comme elles opèrent à 12 volts, contre 5 pour le port utilisateur de votre ordinateur, il faudra recourir à une puce tampon supplémentaire, capable de traduire simultanément les deux tensions pour rendre possible la communication. Prochainement, nous verrons comment effectuer les connections nécessaires avec les moteurs, la fiche à 15 broches et le port utilisateur.
(Cl. Kevin Jones.)

Notre projet : la construction du circuit

- Découpez le circuit (24 bandes de 50 trous) en dégageant les zones d'insertion telles qu'elles sont indiquées sur le diagramme.
- Soudez les trois puces sur le circuit, puis mettez en place les fils de connection.
- Soudez les quatre résistances, sans trop vous préoccuper de leur emplacement exact sur le circuit.
- Les deux capacités de 0,1 µF peuvent être placées où vous le désirez, mais celle de 1000 µF doit être installée avec la borne positive du même côté du circuit que les deux puces.
- Vérifiez la position des puces sur le circuit. Chacune d'elles comporte une extrémité pourvue d'une découpure : c'est ce côté qui doit se trouver face aux fiches de raccordement.
- Vérifiez soigneusement l'ensemble du circuit. Non seulement les composants doivent être bien en place, mais il ne doit pas y avoir de déchets de soudure (qui peuvent engendrer des faux contacts). Faites usage d'un couteau bien aiguisé pour les nettoyer, si vous en trouvez.

Derniers détails

Notre cours de langage d'assemblage se termine. Nous réunissons tous les morceaux de notre programme de débogage, codons le module principal proprement dit.

La première tâche du module principal consiste à construire le mécanisme d'interruption, qui nous permet de mettre des points d'interruption dans le programme à déboguer. Cela transfère le contrôle au débogueur et nous permet d'inspecter le contenu des registres et emplacements mémoire. Nous devons ensuite obtenir l'adresse de départ du programme à déboguer afin que le contrôle puisse lui être transmis en utilisant la commande S. Le reste de la routine principale demande d'entrer des commandes à partir du clavier et de les exécuter; le contrôle est transféré au programme à déboguer par les commandes S et G et repassé au débogueur par les instructions SWI insérées aux points d'interruption.

Le point d'entrée pour les interruptions vient immédiatement après l'appel à ce sous-programme. La première instruction ici est pour sauvegarder le pointeur de pile, S, afin qu'il puisse servir à référencer les valeurs des registres sauvegardés sur la pile par SWI. L'étape suivante est l'interprétation de commande. Nous avons déjà développé des sous-programmes pour effectuer toutes les commandes, aussi le problème consiste-t-il ici à choisir le sous-programme approprié à la commande entrée.

Il est possible de coder cela comme un ensemble d'instructions IF emboîtées, mais nous utiliserons le fait que la routine de commande d'entrée donne un décalé dans une table de caractères de commande pour exécuter ces appels à l'aide d'une table de sauts. Ce n'est peut-être pas la méthode la plus efficace, mais c'est une technique utile qui vaut la peine d'être considérée. Elle implique la mise en place d'une table d'adresses pour chacun des sous-programmes qui exécutent en fait une commande.

L'instruction JMP, contrairement aux instructions de branchement, peut utiliser n'importe lequel des modes d'adressage normaux, y compris indirect et indexé. Si nous chargeons X avec l'adresse de base de la table et utilisons le décalé en B (doublé parce que ce sera une table d'adresses à 16 bits, contrairement à la table de lettres de commande à 8 bits), alors la commande :

```
JMP [B,X]
```

transférera le contrôle au sous-programme approprié. L'appel BSR est fait à l'adresse de cette instruction de saut. Comme nous avons besoin de construire cette table d'avance, il est nécessaire d'avoir un autre stade d'initialisation pour mener à bien cette opération.

Données :
Table-de-saut est une table de 8 adresses à 16 bits.
CMDB, CMDU, etc., sont les adresses de départ des sous-programmes.
Traitement :

```
Pour chaque sous-programme
  Entre adresse de départ
  Sauvegarde adresse de départ dans table-de-saut
FinPour
```

Il nous faut maintenant considérer ce qui va arriver à la fin de l'exécution, lorsqu'on va à la commande Q (Quitter), bien qu'il n'y ait, en fait, pas grand-chose à faire. Il convient de laisser à la fois le débogueur et le programme intacts, afin qu'on puisse les rentrer si nécessaire.

A la sortie, la pile doit être dans la même situation qu'au départ. Une solution consisterait à utiliser une pile distincte pour notre programme, en mettant à S une nouvelle valeur, puis en restaurant l'ancienne valeur. C'est souvent une technique utile, mais dans notre situation, il sera peut-être difficile de trouver de l'espace mémoire libre, avec à la fois le programme et le débogueur. Une autre solution consiste simplement à incrémenter S de la quantité appropriée pour prendre ce que nous y avons laissé; mais c'est aussi difficile, car nous ne savons pas si une interruption a eu lieu ou non, et les quantités sur la pile seront différentes. Le plus simple est de sauvegarder la valeur initiale de S et de la restaurer comme dernière opération du programme.

Le mécanisme d'interruption, conçu dans la procédure d'initialisation, stocke trois octets à l'adresse donnée dans le vecteur SWI comme $FFFA; nous devons le restaurer sous peine d'obtenir des résultats bizarres si le système d'exploitation utilise SWI pour ses propres besoins. Ce dont nous avons évidemment besoin, c'est d'une autre étape d'initialisation où nous sauvegarderons ces valeurs à restaurer.

Données :
Sauvegardé est les 5 octets pour stocker les valeurs sauvegardées.
Pointeur-pile est la valeur actuelle de S, plus 2.
Vecteur-SWI se trouve en $FFFA.
Traitement :

```
Sauvegarde Pointeur-pile en Sauvegardé
Entre Vecteur-SWI
Sauvegarde 3 octets en Vecteur-SWI dans Sauvegardé
```

La routine Q doit simplement inverser ce traitement et redonner le contrôle au système d'exploitation. Ce peut être fait de plusieurs façons : l'instruction SWI elle-même peut être utilisée, après réinitialisation de son vecteur, ou on peut faire un saut à un point d'entrée connu dans le système d'exploitation. Un saut, via le vecteur réinitia-

lisé qui réside en $FFFE, est assuré de redonner le contrôle au système d'exploitation, bien que cela puisse causer un démarrage à froid.

Traitement quitte

Données :

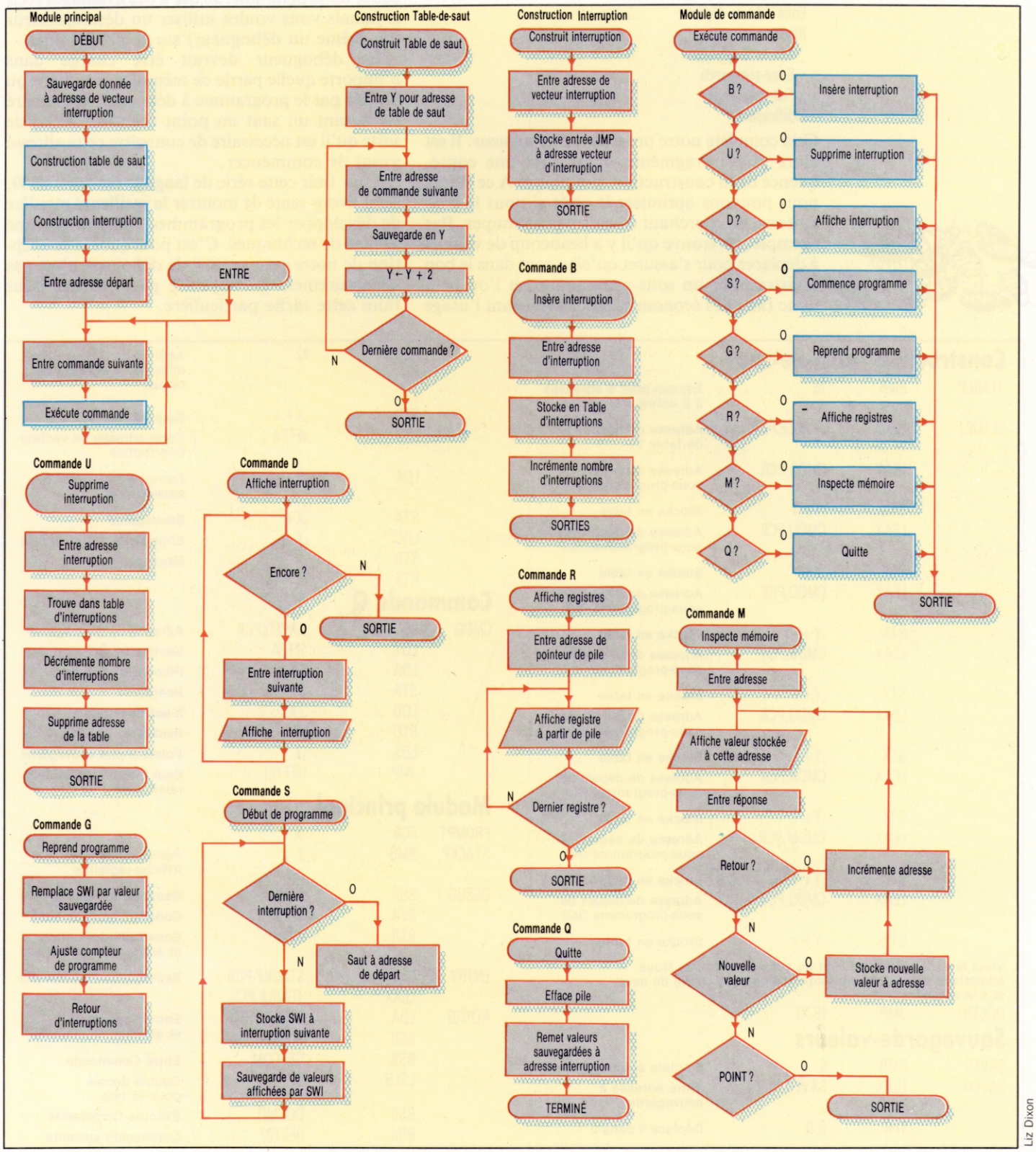
Sauvegardé est les 5 octets pour stocker les valeurs sauvegardées
Pointeur-pile est la valeur actuelle de S, plus 2
Vecteur-SWI se trouve en $FFFA
Vecteur-réinitialisé se trouve en $FFFE

Traitement :

Restaure 3 octets de Sauvegardé en Vecteur-SWI
Restaure Pointeur-pile
Saute à système d'exploitation

Nous sommes maintenant prêts à coder le module principal. La conception s'est légèrement modifiée depuis le début, mais elle reste essentiellement la même.

Déroulement du programme
Ces diagrammes correspondent aux modules du programme de débogage. Ils sont placés dans l'ordre dans lequel ils seront appelés. Les pavés en bleu indiquent des routines à appeler.



Liz Dixon

Le module principal

Données :
- **Suggestion** pour entrée de commande : c'est le caractère ASCII « > »
- **Décalé-Commande** dans table des caractères de commande et table de saut

Traitement :
- Sauvegarde-valeurs
- Construction-de-table-de-saut
- Consruction-d'interruption
- Entre Adresse-départ
- Répète
  - Affiche Suggestion
  - Entre-commande
  - Fait-commande
- Indéfiniment

Cela complète notre programme débogueur. Il est alors plutôt fragmenté, mais c'est une conséquence de la construction modulaire. A ce stade, nous pouvons optimiser le code si nous le souhaitons, en cherchant ce qu'on peut couper. Par exemple, on trouve qu'il y a beaucoup de valeurs à déplacer pour s'assurer qu'elles sont dans le bon registre pour un sous-programme, et l'on peut donc faire des économies en redéfinissant l'usage des registres. Ce n'est pas vraiment conseillé, à moins que l'espace mémoire soit très restreint. Nous avons défini les mêmes zones de données dans différents endroits, lorsqu'elles étaient requises. Il y a deux manières de manipuler les zones de données dans le programme complet : soit laisser la donnée avec le module qui l'utilise, ce qui est théoriquement la meilleure solution ; soit définir toutes les données en une fois au début du programme, ce qui a des avantages réels si jamais vous voulez utiliser un désassembleur (ou même un débogueur) sur le programme.

Le débogueur devrait être chargé dans n'importe quelle partie de mémoire inoccupée ou utilisée par le programme à déboguer. Il est entré en faisant un saut au point d'entrée DEBUG, de sorte qu'il est nécessaire de connaître cette adresse avant de commencer.

Pour finir cette série de langage machine 6809, nous avons tenté de montrer la meilleure manière de développer les programmes, illustrée par une variété de techniques. C'est pourquoi la conception de notre programme de débogage n'est pas nécessairement le moyen le plus efficace pour faire cette tâche particulière.

Construction-Table-de-saut

```
JTABLE  RMB   16          Espace pour 8 adresses à 2 octets
SETUPJ  LEAY  JTABLE,PCR  Adresse de base de table Y
        LEAX  CMDB,PCR    Adresse de départ de sous-programme CMDB
        STX   ,Y++        Stocke en table
        LEAX  CMDU,PCR    Adresse de départ de sous-programme CMDU
        STX   ,Y++        Stocke en table
        LEAX  CMDD,PCR    Adresse de départ de sous-programme CMDD
        STX   ,Y++        Stocke en table
        LEAX  CMDS,PCR    Adresse de départ de sous-programme CMDS
        STX   ,Y++        Stocke en table
        LEAX  CMDG,PCR    Adresse de départ de sous-programme CMDG
        STX   ,Y++        Stocke en table
        LEAX  CMDR,PCR    Adresse de départ de sous-programme CMDR
        STX   ,Y++        Stocke en table
        LEAX  CMDM,PCR    Adresse de départ de sous-programme CMDM
        STX   ,Y++        Stocke en table
        LEAX  CMDQ,PCR    Adresse de départ de sous-programme CMDQ
        STX   ,Y++        Stocke en table
```

Voici le saut proprement dit au sous-programme. Nous admettons que X contient l'adresse de JTABLE (table de saut) et B le décalé.

```
DOCMD   JMP   [B,X]
```

Sauvegarde-valeurs

```
SAVED   RMB   5           5 octets à sauvegarder
SAVEIT  LEAX  SAVED,PCR   Entre adresse à sauvegarder
        TFR   S,D         Déplace S dans D
        ADDD  #2          Additionne 2 pour faire attention à adresse de retour
        STD   ,X++        Sauvegarde
        LDY   $FFFA       Entre adresse de vecteur interruption
        LDA   ,Y+         Entre premier octet à sauvegarder
        STA   ,X+         Sauvegarde
        LDD   ,Y          Entre deux autres octets
        STD   ,X          Sauvegarde
        RTS
```

Commande Q

```
CMDQ    LEAX  SAVED,PCR   Adresse de Sauvegardé
        LDY   $FFFA       Vecteur-SWI
        LDA   2,X         Premier de 3 octets
        STA   ,Y+         Restauré
        LDD   3,X         2 autres octets
        STD   ,Y          Restauré
        LDS   ,X          Pointeur-pile sauvegardé
        JMP   [$FFFE]     Saut interdit via vecteur réinitialisé
```

Module principal

```
PROMPT  FCB   '>
STACKP  RMB   2           Pointeur-pile pour Affiche-registres
DEBUG   BSR   SAVEIT      Sauvegarde-valeurs
        BSR   SETUPJ      Construit-table-de-saut
        BSR   INIT        Construit-interruption et entre Adresse-départ
ENTRY   STS   STACKP,PCR  Sauvegarde Pointeur-pile
        LEAX  JTABLE,PCR
REPT02  LDA   PROMPT,PCR  Entre Suggestion et affiche
        BSR   OUTCH
        BSR   GETCOM      Entre Commande
        LSLB              Double décalé pour 16 bits
        BSR   DOCMD       Exécute Commande
        BRA   REPT02      Commande suivante
```