A MARSHALL CAVENDISH 3 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol 1 No 3

MACHINE CODE 3

GET DOWN TO LOW-LEVEL LANGUAGES 65

Learn why BASIC is limited for some jobs, and get to know machine code and assembly language

GAMES PROGRAMMING 3

FUN-TO-PROGRAM MAZE GAMES 68

Create your own maze on the screen, complete with travelling 'eater'

APPLICATIONS 2

STREAMLINE YOUR HOBBIES FILE-2 75

Now you've set up your datafile, here's how to search it, amend or delete your records

MACHINE CODE 4

DRAW A FIRE-BREATHING DRAGON 80

Another useful character to add to your stock

BASIC PROGRAMMING 5

HOW TO PLOT, DRAW, LINE AND PAINT 84

Get started on computer art

BASIC PROGRAMMING 6

VARIABLES: THE MYSTERY EXPLAINED 92

Learn to understand all those Xs and Ys

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover: Dave King. Pages 65-67, Malcolm Harrison. Pages 68-74, illustrations, Peter Bentley; photos, John Darling. Pages 75-79, Dave King. Pages 80-83, Julek Heller. Pages 84-91, illustrations, Colin Hadley; photos, John Darling. Pages 92-96, Dave King.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.45) for each binder to the address below:

Marshall Cavendish Services Ltd,
Department 980, Newtown Road,
Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Gordon and Gotch Ltd, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Copies of any part of INPUT can be obtained from the following addresses at the regular cover price, with no extra charge for postage and packing:

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services,
Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent

COPIES BY POST

Our Subscription Department can supply your copies direct to you regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. These rates apply anywhere in the world. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd,
Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64. .

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +

COMMODORE 64 and 128

ACORN ELECTRON, BBC B and B+

DRAGON 32 and 64

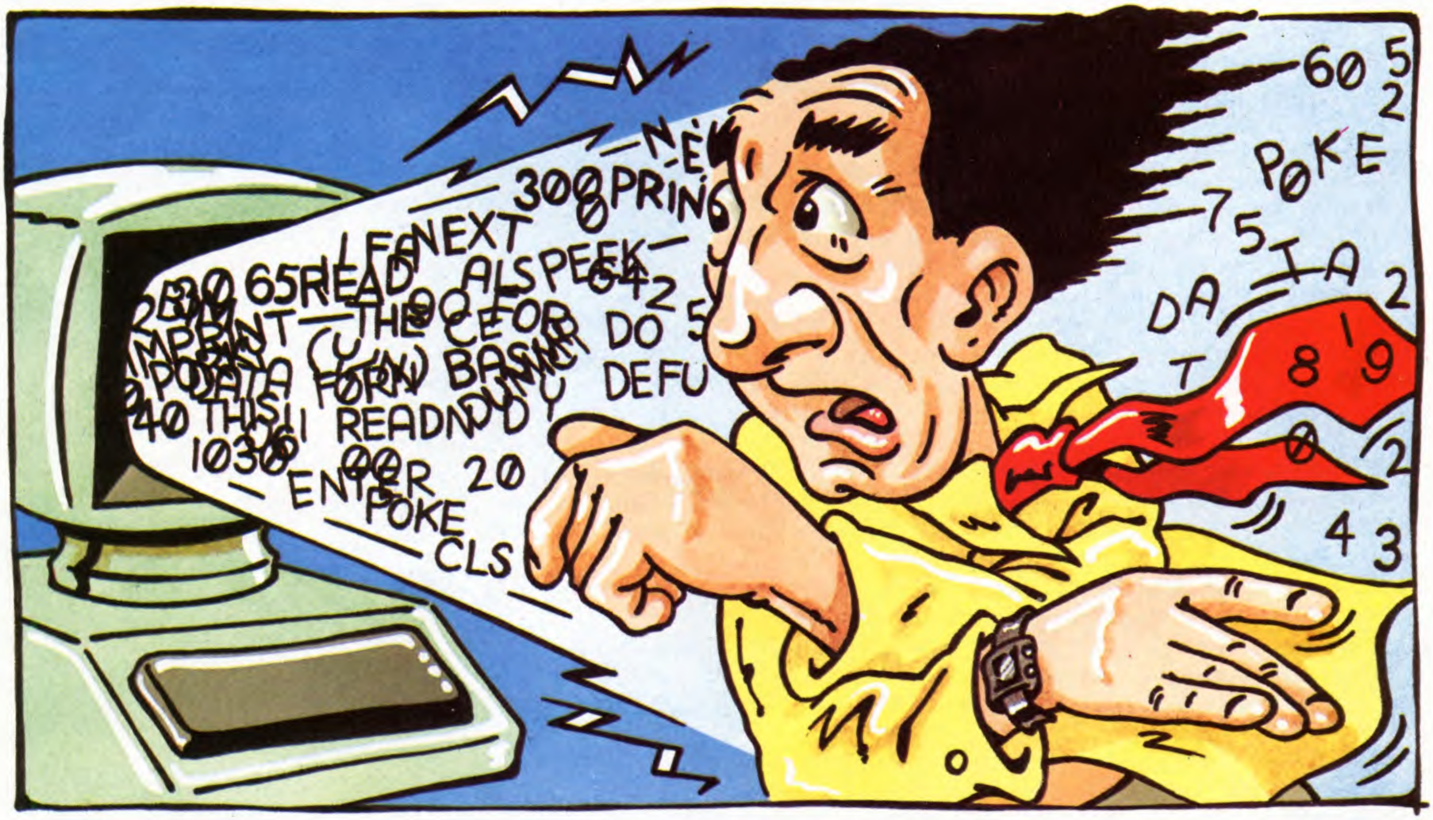
ZX81

VIC 20

TANDY TRS80 COLOUR COMPUTER

GET DOWN TO LOW-LEVEL LANGUAGES

- WHAT IS MACHINE CODE?
- ADVANTAGES OF MACHINE CODE OVER BASIC
- UNDERSTANDING OPCODES AND ASSEMBLY LANGUAGE



Faster action and smoother movement across the screen are just two of the advantages you get from machine code in games programming. First, though, you need to know how machine code actually affects your computer

Most home computer users find BASIC more than adequate for all their programming needs. It can be used to write simple games, business and household programs. What's more, it is fairly easy to learn. It is also flexible, simple to use and easy to adapt for different machines.

The problem is that it is slow and cumbersome. BASIC programs eat up a lot of memory. The computer finds these great wodges of programming unwieldy to handle.

Once you get into games programming seriously, you will find that using BASIC makes movements jerky and allows only one thing to happen at a time. If a gun fires, for example, the rest of the action has to stop, if only for a split second. And if your program is burdened down with IF... THEN statements, the pace will slow to an excruciating crawl.

This is where the serious games programmer leaves BASIC behind and moves onto what the arcade-game programmers use, machine code, which is—as near as makes no difference—the language of the computer itself.

WHAT IS MACHINE CODE?

The reason BASIC is slow is that it is a *high level language*. That means it uses English words, plus operations that are familiar from ordinary arithmetic, which are easy for human beings to learn and understand.

But your computer does not think in English, nor does it understand standard human arithmetic symbols. In fact, it does not think or understand anything. It operates on electrical pulses which represent numbers.

Machine code is a computer language that is composed entirely of numbers which are the direct equivalent of those the computer uses. So when you use machine code you are no longer expecting the computer to speak your language. You are, in effect, communicating with it in its own language.

As one example (in this case from the Spectrum), machine code looks like this:

```
B9 28 Ø8
```

In BASIC this is equivalent to:

```
1ØØ IF A=C THEN GOTO 19Ø
```

The machine code, as you'll see, is a series of two-digit numbers. The letter B in this line of machine code is, in fact, a figure. It represents 11 in *hexadecimal* arithmetic.

These hexadecimal numbers are fed directly into the computer's memory. Operating instructions, data, numbers, letters, words and memory addresses are all represented in machine code by these two digit numbers.

And the computer can tell the difference between instructions, addresses and data only by the order in which they occur. For example, the first number in any program must represent an instruction. If, by mistake, you keyed in something as your first entry which contained a number, such as an address or some data, the computer would try to read it as an instruction when the program was executed.

Absolute accuracy is needed when you key in these numbers, otherwise the program will not work and it may even be wiped from the memory when you try to RUN it.

ADVANTAGES OF MACHINE CODE

When you key in a line of BASIC the computer has to interpret it into its own operating language before it can execute it. This is a cumbersome process. It takes up a lot of time and as BASIC statements rarely translate directly into a single machine code procedure, it often results in much more involved operating procedures than are really necessary.

When a whole program written in BASIC is RUN, each line has to be interpreted into the machine's operating numbers, then performed, before the computer can go onto the next line. If it goes round and round a loop in the BASIC program, it will have to interpret each line again each time it gets to it.

Everytime the computer comes across an instruction in BASIC it will have to perform the following procedure:

- Look at the instruction
- Translate the instruction into a series of machine code instructions
- Perform the machine code instructions one by one
- Move onto the next instruction

It is not difficult to see how slow this process can be if it has to be performed over and over again.

It is possible to *compile* BASIC—that is translate the whole program into machine code in one go before it is RUN—by the use of commercially available compiling programs. But this still results in unnecessarily long, unwieldy programs. The most convenient analogy for the difficulties this interpretation process causes is the manual for a car or camera that has been written in Japanese and translated into English.

In comparison, the step from machine code into the computer's own operating process hardly takes any time at all. Each number in machine code translates directly into a certain configuration of the electrical pulses the computer operates on. There is no laborious interpretation routine, no substitution of a whole set of machine level instructions for one written command—just a simple fast, number-for-number conversion.

This means that you can write programs so that they will be short and efficient to run.

As an example of the comparative speeds of the two systems, try the following program which does the same thing using both BASIC and machine code:

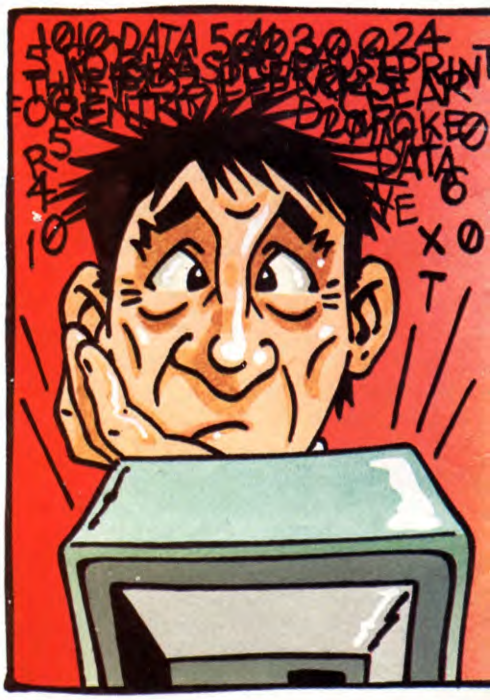
```
1 MODE 6
1Ø FOR T=&1ØØØ TO &1Ø1C
2Ø READ A:?T=A
3Ø NEXT
4Ø PRINT"THIS IS SLOW BASIC"
6Ø FOR T=&61ØØ TO &7FFF
7Ø ?T=?(T+&2ØØØ)
8Ø NEXT
9Ø CLS
1ØØ PRINT"NOW THIS IS FAST MACHINE
   CODE"
1Ø5 DUMMY=INKEY(1ØØ)
11Ø CALL &1ØØØ
12Ø END
2ØØ DATA 162,96,16Ø,Ø,185,Ø,128,153,Ø,
   96,2ØØ,2Ø8,247,232,138,24,1Ø5,32,141,6,
   16,142,9,16,224,128,2Ø8,23Ø,96
```

```
1Ø CLEAR 29999
2Ø FOR n=3ØØØØ TO 3ØØ11
3Ø READ a
4Ø POKE n,a
5Ø NEXT n
6Ø PRINT "This is BASIC"
7Ø FOR n=16384 TO 22527
8Ø POKE n,PEEK (n-16384)
9Ø NEXT n
1ØØ CLS
11Ø PRINT "Now—MACHINE CODE"
12Ø PAUSE 1ØØ
13Ø RANDOMIZE USR 3ØØØØ
14Ø STOP
15Ø DATA 33,Ø,Ø,17,Ø,64,1,Ø,24,237,
   176, 2Ø1
```

```
1Ø CLEAR 2ØØ,31ØØØ
2Ø DEFUSRØ=31ØØØ
3Ø FOR N=31ØØØ TO 31Ø15
4Ø READ A
5Ø POKE N,A
6Ø NEXT
7Ø CLSØ
8Ø PRINT @Ø,"THIS IS BASIC!"
9Ø FOR N=1 TO 5ØØ:NEXT
1ØØ FOR N=1Ø56 TO 1535
11Ø POKE N,PEEK(N+34ØØØ)
12Ø NEXT
13Ø FOR N=1 TO 1ØØØ:NEXT
14Ø CLSØ
15Ø PRINT @Ø,"THIS IS MACHINE
   CODE!"
16Ø FOR N=1 TO 1ØØØ:NEXT
17Ø N=USRØ(Ø)
18Ø FOR N=1 TO 2ØØØ:NEXT
19Ø DATA 2Ø6, 136, 24Ø, 142, 4,
   32,166,192,167,128,14Ø,6,Ø,
   38,247,57
```

```
1Ø PRINT "♡■THIS IS SLOW BASIC":
   POKE 5328Ø,7:POKE 53281,7
2Ø FOR Z=Ø TO 959:POKE 55336+Z,1:
   POKE 1Ø64+Z,46:NEXT Z
3Ø FOR Z=1 TO 1ØØØ:NEXT Z
4Ø PRINT "♡"
5Ø FOR Z=1 TO 1ØØØ:NEXT Z
1ØØ PRINT"▤NOW THIS IS FAST MACHINE
   CODE":POKE 5328Ø,1Ø:POKE
   53281,1Ø
1Ø5 FOR Z=1 TO 2ØØØ:NEXT Z
11Ø FOR Z=Ø TO 19:READ X:POKE 832+Z,
   X:NEXT Z
12Ø SYS 832
13Ø DATA 162,Ø,169,46,157,4Ø,4,157,
   Ø,5,157,Ø,6
135 DATA 157, 232,6,232,2Ø8,241,96
```

```
1Ø PRINT"♡☰▒THIS IS SLOW
   BASIC":POKE 36879,8
2Ø FOR Z=Ø TO 483:POKE
   77Ø2+Z,46:NEXTZ
3Ø FOR Z=1 TO 1ØØØ:NEXT Z
4Ø PRINT "♡"
5Ø FOR Z=1 TO 1ØØØ:NEXT Z
```



```
100 PRINT"■NOW FAST MACHINE CODE"
105 FOR Z=1 TO 2000:NEXT Z
110 FOR Z=0 TO 13:READX:POKE
   832+Z,X:NEXT Z
120 SYS 832
130 DATA 162,0,169,46,157,22,30,
   157,0,31,232,208,245,96
```

As you will see, all the program does is to fill the screen with rubbish. But you will see how much quicker the machine code routine performs the task.

ASSEMBLY LANGUAGE

One disadvantage of machine code is that it is extremely difficult to write and debug. Few people can remember the codes for the various instructions. Even if you can, the operation codes—or *opcodes*—are indistinguishable from the numbers that are fed in as data or addresses. So you can't understand any section of a machine code program unless you read the whole program from the beginning, which is no help when you are trying to pick out bugs.

The opcode numbers differ considerably between different microprocessors, so that translating machine code programs from machine to machine can be very difficult.

One way round some of these problems is to step back and write your program in a language one level higher than machine code. This is known as *assembly language*, explained in detail in later articles.

In assembly language, machine code operations are represented by abbreviations. For example, LD means load and J means jump (though the syntax of these instructions varies between microprocessors).

With some practice it is as easy to read assembly language as it is to read BASIC, though understanding the structure of the language is a bit more complicated than the equivalent BASIC.

The disadvantage of assembly language is that the computer cannot read it directly. It has to be *assembled* by an assembler—which is a program in either BASIC or machine code—before it will work.

But this is a simple process compared with interpreting BASIC. Assembly language is equivalent to machine code—that is, every assembly language instruction corresponds directly with a machine code operation. It can be translated word for word, number by number, instruction by instruction into the computer's operating code.

The assembler also translates the whole program in one go before the computer executes it, rather than interpreting it line by line while the program is RUNning. This gives it further speed advantages.

The BBC and Electron computers come with an assembler program built in. For the Spectrum, Dragon, Tandy and Commodore, assemblers are available commercially; or you can use those provided in a later issue of *Input*. But for the moment, if you do not have an assembler, you can hand-assemble programs. Even on the simplest programs you will find it easier to write them in assembly language, then translate them by hand from the machine code tables given in any machine code book for your machine.

All this might seem like a lot of bother. But before you decide this is so, try out these machine code programs. Each is fed in as DATA statements in BASIC programs. The DATA is listed at the end.

```
10 CLEAR 29999
20 FOR n=30000 TO 30020
30 READ a
40 POKE n,a
50 NEXT n
60 RANDOMIZE USR 30000
70 GOTO 60
80 DATA 17,0,88,46,0,237,95,71,58,140,
   92,128,230,63,103,1,0,3,237,176,201
```

```
10 FOR T=4047 TO 4123
20 READ A:?T=A
30 NEXT T:CALL 4047
40 DATA 160,0,185,0,128,41,7,9,144,153,
   0,124,200,169,255,153,0,124,200,192,
   0,208,235
50 DATA 174,211,15,232,224,255,208,2,162,
   128,142,211,15,174,218,15,232,224,
   128,240,8,142,218,15
60 DATA 142,224,15,208,206,162,124,142,
   218,15,142,224,15,24,173,217,15,105,1,
   41,1,141,217,15,141,223,15,76,209,15
```

```
10 CLEAR 200,31000
20 DEFUSR0=31000
30 FOR N=31000 TO 31020
40 READ A
50 POKE N,A
60 NEXT
70 N=USR0(0)
80 POKE 32767,RND(256)-1
90 FOR N=1 TO 100:NEXT
100 GOTO 70
110 DATA 142,4,0,206,136,184,166,192,
   184,127,255,138,129,167,128,140,6,0,
   38,242,57
```

```
20 FOR Z=1 TO 26
30 PRINT "■□□□□□□□□□□
   □□□□□□□□□□□□□□
   □□□□□□□□□□□□□□";
40 NEXT Z
100 FOR Z=0 TO 25:READ X:POKE 832+
   Z,X:NEXT Z
110 SYS 832:FOR Z=1 TO 50:NEXT Z:
   GOTO 110
120 DATA 162,0,200,192,7,208,2,160,
    0,152,157,0,216,157,0,217,157,0,218
130 DATA 157,232,218,232,208,233,96
```

```
20 FOR Z=1 TO 25
30 PRINT "■□□□□□□□□□□
   □□□□□□□□□□□□";
40 NEXT Z
100 FOR Z=0 TO 19:READ X:POKE
   832+Z,X:NEXT Z
110 SYS 832:FOR Z=1 TO 50:NEXT Z:GOTO
   110
120 DATA 162,0,200,192,7,208,2,
   160,0,152,157,0,150
130 DATA 157,0,151,232,208,239,96
```

Now—just for comparison—see if you can write a similar program in BASIC. We think you will see our point. Later on, we shall look at what is happening here.

FUN-TO-PROGRAM MAZE GAMES

Sophisticated maze games require long programs. But you can design simple ones—and learn important principles—with not much more than a loop and a DATA statement

Maze games have a perennial fascination for computer owners, and variations on that well-known gobbling man continue to pour from the software houses.

This article shows you how to jump on the bandwagon in a small way by writing a maze game of your own.

The maze in this first attempt includes no 'enemies' or obstacles, which would require a big lump of program. But it does show how to program so that your main character cannot walk through the walls—the basis of all maze games. And it includes scoring and timing, and a 'best score' routine, to add competitive interest.

The easiest way to understand how the Spectrum maze game works is to enter it in stages. So start by building the maze itself:

```
100 FOR n=3 TO 17
110 READ a$
120 FOR m=7 TO 21
130 PRINT AT n,m;"."
140 IF a$(m-6)="p" THEN PRINT
   PAPER 1; INK 1;AT n,m;"□"
150 NEXT m
160 NEXT n
9000 DATA "ppppppppppppppp"
9010 DATA "p . . . . . . . . . . . . . p"
9020 DATA "p . pp . pp . pp . pp . p"
9030 DATA "p . p . . . . . . . . . p . p"
9040 DATA "p . . . p . p . p . p . . . p"
9050 DATA "p . ppp . p . p . ppp . p"
9060 DATA "p . . . . . p . p . . . . . p"
9070 DATA "pppp . pp . pp . pppp"
9080 DATA "p . . . . . p . p . . . . . p"
9090 DATA "p . ppp . p . p . ppp . p"
9100 DATA "p . . . p . p . p . p . . . p"
9110 DATA "p . p . . . . . . . . . p . p"
9120 DATA "p . pp . pp . pp . pp . p"
9130 DATA "p . . . . . . . . . . . . . p"
9140 DATA "ppppppppppppppp"
```

Lines 100, 120, 150 and 160, using a pair of the now-familiar FOR ... NEXT loops, set out the boundaries of the maze. Line 130 PRINTs a full stop in each square.

Then Lines 110 and 140 take over, READing the DATA in Lines 9000 to 9140 and replacing the full stop with a space—but in blue ink on blue paper—everywhere that the DATA pattern shows the letter p. They can do this because the Spectrum will not normally accommodate two characters at the same location, so the second one replaces the first.

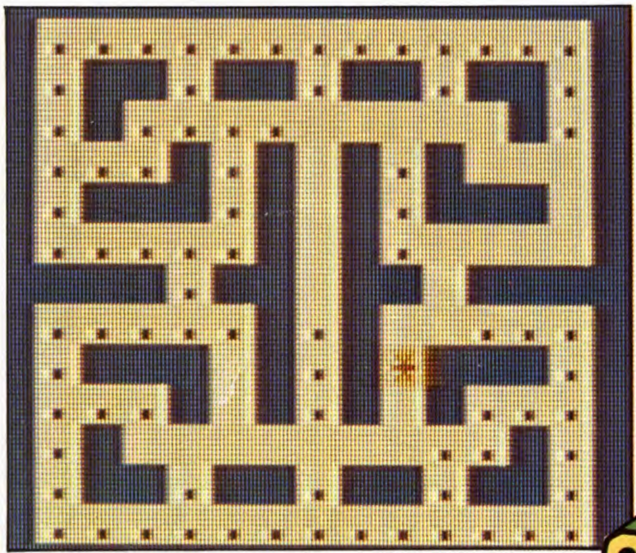
CREATING THE 'EATER'

A maze is pretty useless, however, without someone or something to move around in it. So RUN the program and then add:

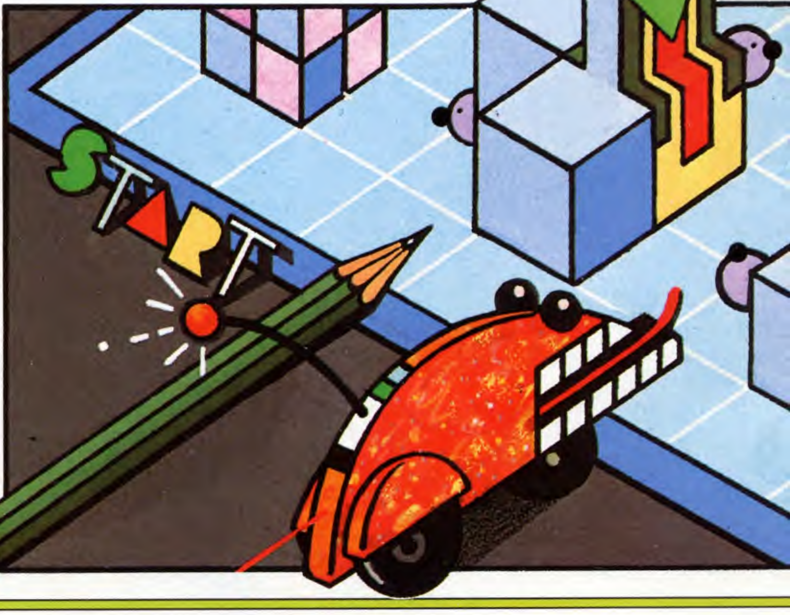
```
50 LET x=10
60 LET y=14
1000 PRINT PAPER 6; INK 2;AT x,y;"*"
1010 LET xx=x
1020 LET yy=y
1030 IF INKEY$="" THEN GOTO 1030
1040 IF INKEY$="p" AND ATTR (x-1,y)
   <>9 THEN LET x=x-1
1050 IF INKEY$="l" AND ATTR (x+1,y)
   <>9 THEN LET x=x+1
1060 IF INKEY$="z" AND ATTR (x,y-1)
   <>9 THEN LET y=y-1
1070 IF INKEY$="x" AND ATTR (x,y+1)
   <>9 THEN LET y=y+1
1080 PRINT INK 7;AT xx,yy;"□"
1090 GOTO 1000
```

If you've already tried your hand at keyboard control (pages 54 to 59), most of these lines will already be familiar. Like all famous games characters, our asterisk friend is hatched at x,y and is moved around—with your help—by a series of INKEY$ lines.

Here, however, there is a vital difference: he can move onto a particular square only if it *isn't* blue. That is, if ATTR does *not* equal 9—9 being the numerical value of the colours in which you have craftily PRINTed the lines of your maze.



1. A Spectrum maze needs only a short program



- THE PRINCIPLES OF CREATING A MAZE GAME
- SETTING OUT A MAZE USING LOOPS AND **DATA** STATEMENTS
- SCORING AND TIMING
- HOW TO MAKE A CHARACTER MOVE THROUGH THE MAZE
- MAKING YOUR MAZE WALLS 'CHARACTER PROOF'
- DESIGNING YOUR OWN MAZE

In case you feel like creating a maze of a different colour, here is how you work out the value of ATTR:

1 Take the code for the INK colour, as printed on the computer keyboard—in this case, 1. Result, 1.

2 Take the code for the PAPER colour—in this case, 1—and multiply by 8. Result, 8.

3 Add 64 if the area concerned is BRIGHT. Result, in this case, Ø.

4 Add another 128 if the area concerned is in FLASH. Result, in this case, Ø.

Then you add all the numbers together. The result is the ATTR number.

ATTR on the Spectrum has other uses than preventing a character from entering a prohibited area. In a bigger maze, for example, you could use the same idea to make a character explode or burn up if it *did* enter the forbidden zone. For a red INK, red PAPER maze you would need a line starting

```
IF ATTR (x,y)=18 THEN...
```

In the meantime Lines 1Ø1Ø, 1Ø2Ø and 1Ø8Ø define the position which your character has just left, and obliterate, by PRINTing a space, the full stops he has 'eaten'. These are useful lines in any game, but note where they come—the first lines just before the INKEY$ 'action', the final line straight after it.

Note that Line 1Ø9Ø is only temporary and will be overwritten later. Its purpose is to allow you to RUN the program and test it—without the loop that this sets up, the character would only move one square.

SCORING AND TIMING

Now you have the makings of a simple game. To make it playable—in the absence of 'enemies', which would require a long lump of program—the best thing is a scoring and timing routine. So add these extra lines:

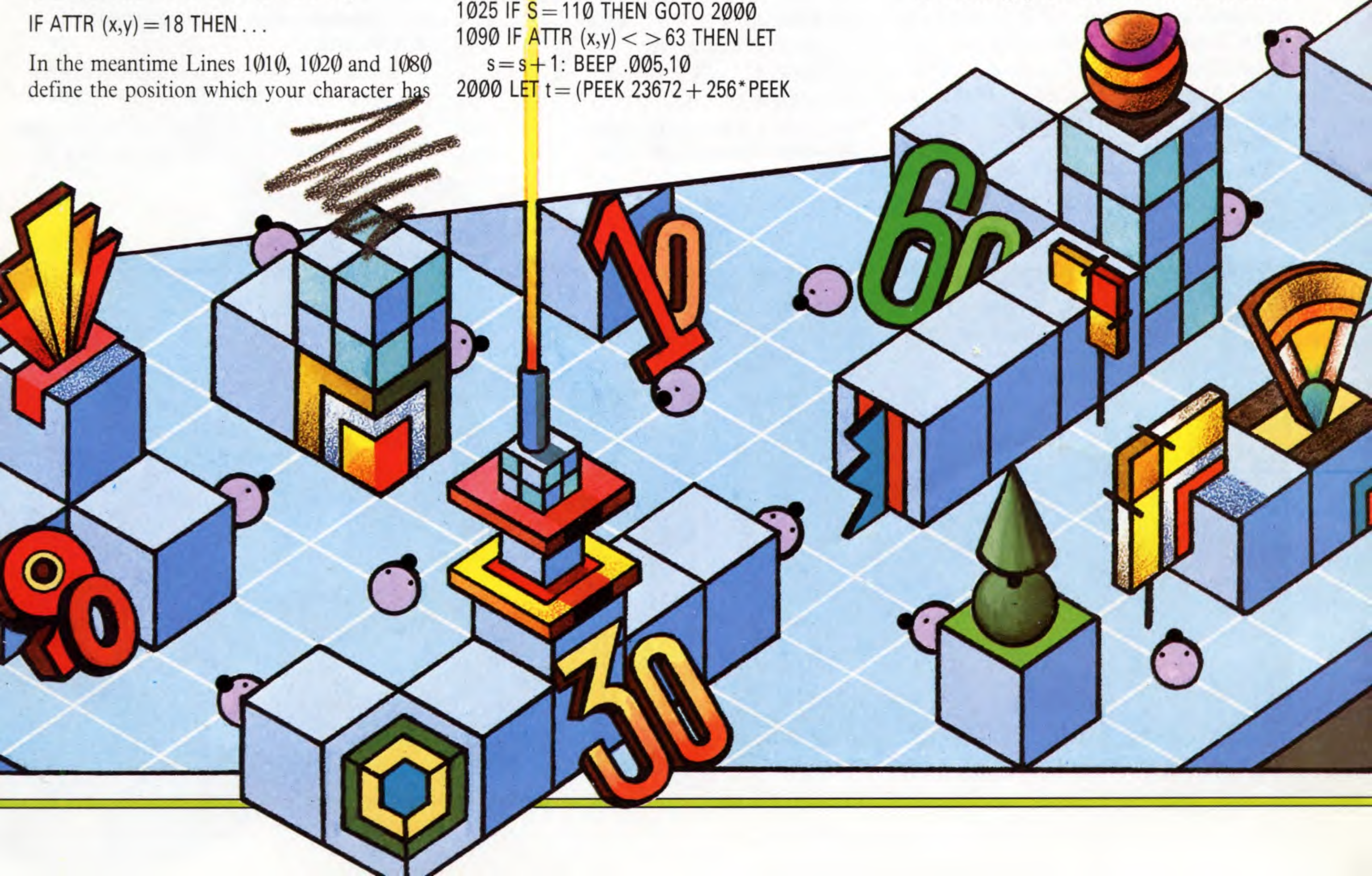
```
1Ø LET bt=1ØØØØØ
4Ø LET s=Ø
99Ø POKE 23672, Ø: POKE 23673, Ø
1Ø25 IF S=11Ø THEN GOTO 2ØØØ
1Ø9Ø IF ATTR (x,y)< >63 THEN LET
   s=s+1: BEEP .ØØ5,1Ø
2ØØØ LET t=(PEEK 23672+256*PEEK
   23673)/5Ø
2Ø1Ø PRINT AT 1,6;t;"□SECONDS□"
2Ø2Ø IF t<bt THEN LET bt=t
2Ø3Ø PRINT AT 19,4;"BEST TIME□";
   bt;"□SECONDS"
```

To test this you need to add a temporary line. Remember to delete it later:

```
11ØØ GOTO 1ØØØ
```

The scoring bit is quite straightforward. Line 4Ø sets the initial score at zero. Line 1Ø9Ø adds 1 to the score each time your character eats a full stop, or rather, each time it crosses a square whose INK and PAPER are not both white—ATTR, again. And Line 1Ø25, which comes into action when all have been eaten, tells the computer to check out (at Line 2ØØØ) how long the player has taken.

The timing part is a little more difficult



until you understand PEEKs and POKEs, explained fully in a later article. But, in brief, Line 990 sets the Spectrum 'clock' at zero by POKEing Ø into the relevant memory locations. Line 2ØØØ counts the number of frames displayed on your TV screen since the game started, then divides by 5Ø—frames per second on a British TV screen.

At the same time, Line 1Ø sets the original 'best time' at 1ØØ,ØØØ—far longer than anyone is likely to take, so that it *must* be beaten—while Line 2Ø2Ø compares the newest time with 'best time'.

ANOTHER GO?

To give the player the chance to try again, you will need these extra lines. First enter CAPS SHIFT and BREAK, then ENTER.

```
2Ø4Ø PRINT AT 21,2;"PRESS ANY KEY
   TO PLAY AGAIN"
2Ø5Ø IF INKEY$<>"" THEN GOTO 2Ø5Ø
2Ø6Ø IF INKEY$="" THEN GOTO 2Ø6Ø
2Ø7Ø RESTORE
2Ø8Ø GOTO 4Ø
```

RESTORE resets to the beginning of the DATA list.

OTHER MAZES

If you want to try other mazes within the same dimensions as this one, you can do so simply by changing the pattern of letter p's in the DATA statements, Line 9ØØØ onwards. For a bigger or smaller maze, you would also have to redefine the boundaries by altering the numbers in Lines 1ØØ and 12Ø.

If you do this, make sure you have enough DATA to fill the whole maze, or you will get an error report before the maze even begins to appear on your screen.

The Acorn maze program is built up in three parts. The first part draws the maze on the screen, the second part lets you move your 'man' about, eating up all the dots, and the third times the game and PRINTs the score.

Here is the first part. Type it in, then RUN it, and you should see a green maze on the left of the screen with yellow dots marking the pathways through the maze.

```
1Ø MODE 1
3Ø VDU 23,224,255,255,255,255,255,
   255,255,255
4Ø VDU 19,1,4;Ø;19,3,5;Ø;
5Ø CLS
8Ø VDU 5
85 □
11Ø FOR row=8 TO 23
12Ø READ A$
13Ø FOR column=1 TO 19
14Ø MOVE column*32, row*32
15Ø IF MID$(A$,column,1)="A" THEN
   GCOL Ø,1:PRINT CHR$224 ELSE GCOL
   Ø,2: PRINT "."
16Ø NEXT column
17Ø NEXT row
46Ø □
5ØØ DATA AAAAAAAAAAAAAAAAAAA
51Ø DATA A . . . . . . . AAA . . . . . . . A
52Ø DATA A . AA . AA . . . . . AA . AA . A
53Ø DATA A . A . . . . . AAA . . . . . A . A
54Ø DATA A . . . A . A . . . . . A . A . . . A
55Ø DATA A . AAA . AA . A . AA . AAA . A
56Ø DATA A . . . A . . . . . . . . . A . . . A
57Ø DATA AAA . . . A . AAA . A . . . AAA
58Ø DATA AAA . . . A . AAA . A . . . AAA
59Ø DATA A . . . A . . . . . . . . . A . . . A
6ØØ DATA A . AAA . AA . A . AA . AAA . A
61Ø DATA A . . . A . A . . . . . A . A . . . A
62Ø DATA A . A . . . . . AAA . . . . . A . A
63Ø DATA A . AA . AA . . . . . AA . AA . A
64Ø DATA A . . . . . . . AAA . . . . . . . A
65Ø DATA AAAAAAAAAAAAAAAAAAA
```

The main part of this program is the list of DATA statements. These define the shape of the maze, with As representing the walls and each line of DATA representing a single line of the maze. This is a very convenient way of writing out the DATA, as you can see at a glance the shape of the maze and you can also see mistakes more easily.

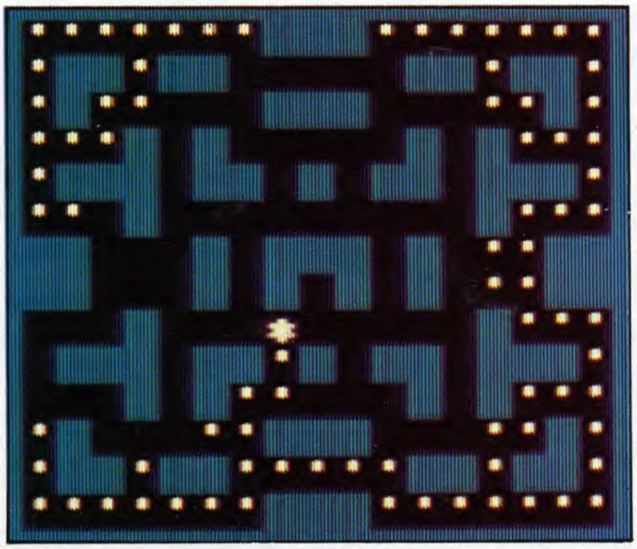
All the rest of the program does is convert each A into a blue block and PRINT it in the right position on the screen. It sounds easy, but there are one or two complications.

HOW IT WORKS

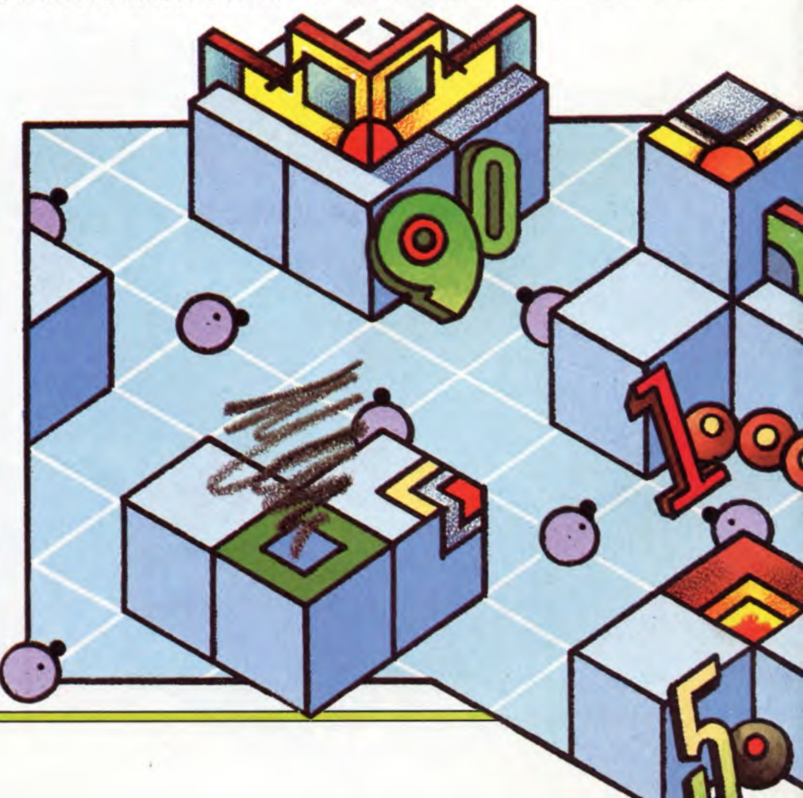
First of all there is no block character on the keyboard so you have to make your own. The simplest way is to define a solid UDG character using VDU 23, as Line 3Ø does.

The next problem is that the colours available in Mode 1 are black, red, yellow and white, whereas we want a green maze. Line 4Ø uses VDU 19 to redefine the colours so that red appears as blue, but don't worry if you don't understand how this works at the moment. Working with colours is the subject of a whole separate article.

The last complication is the VDU 5 in Line 5Ø. This prints the text at the graphics cursor so that all text is positioned using the graphics coordinates. This means that, in-



2. The Dragon version also displays your time



stead of using PRINT TAB(column,row), you have to use MOVE column*32,row*32 and then PRINT. (The number 32 converts the text coordinates into graphics coordinates. For example, there are 4Ø columns in Mode 1. So if you multiply 4Ø by 32 you get 128Ø—the width of the graphics screen.)

Now that the preliminaries are over, the program can get down to drawing out the maze. It is drawn half-way down on the left-hand side of the screen, between rows 8 and 23 and columns 1 to 19.

Lines 11Ø to 17Ø work like this. Line 11Ø takes each row of the maze, then Line 12Ø reads the DATA for this row and calls it A$. Line 13Ø takes each column in the row and Line 14Ø moves the cursor to that position. Line 15Ø does all the work; if the character is an A it PRINTs a blue block; otherwise it PRINTs a yellow dot. MID$ is used to step along the line of DATA one character at a time.

MOVING THROUGH THE MAZE

So far all the program does is draw the maze. Now enter these next few lines to move the asterisk and eat up the dots.

```
6Ø *FX11,15
7Ø *FX12,15
19Ø LET X=1Ø*32 : LET Y=16*32
2ØØ MOVE X,Y: GCOL Ø,Ø: VDU 224,8:
   GCOL Ø,3: PRINT "*"
22Ø LET LX=X: LET LY=Y
23Ø LET K$=GET$
24Ø IF K$="Z" AND POINT(X-32,Y)<>1
   THEN X=X-32
25Ø IF K$="X" AND POINT(X+32,Y)<>1
   THEN X=X+32
26Ø IF K$="L" AND POINT(X,Y-32)<>1
   THEN Y=Y-32
27Ø IF K$="P" AND POINT(X,Y+32)<>1
   THEN Y=Y+32
29Ø MOVE LX,LY :GCOLØ,Ø :VDU224
3ØØ GOTO 2ØØ
```

Now RUN the program. Z, X, P and L move the asterisk as usual.

These lines are very similar to the routine for moving a character given on pages 11 and 12. But this time the program has to prevent the asterisk moving through the walls of the maze. This is what the POINT function is there for.

POINT(X,Y) checks the *graphics* colour at the point X,Y. (It cannot check the text colour—the computer has no function that does this. And that is why the text was printed at the graphics cursor: so that the text characters—the block and the dot—could be PRINTed in graphics colours and their numbers checked.)

In this program, POINT works like this. When a key is pressed, it looks at the colour of the square you want to move to and lets you move as long as the colour is *not* blue—that is, its number is *not* 1. If the square is blue, the line is ignored and the asterisk stays where it is.

Note the 32s creeping in again. Remember that a jump of 32 is equivalent to moving by one character position.

As for the other lines, Line 19Ø sets the start position of the asterisk. Line 2ØØ moves there and immediately PRINTs a black block, using VDU 224 in order to wipe out any dot that may be there. (VDU 224 means the same as PRINT CHR$ 224 but in this case is more convenient.) The other number, 8, moves the cursor back again so it is in the right position to PRINT the asterisk. Line 29Ø blanks out the asterisk's previous position.

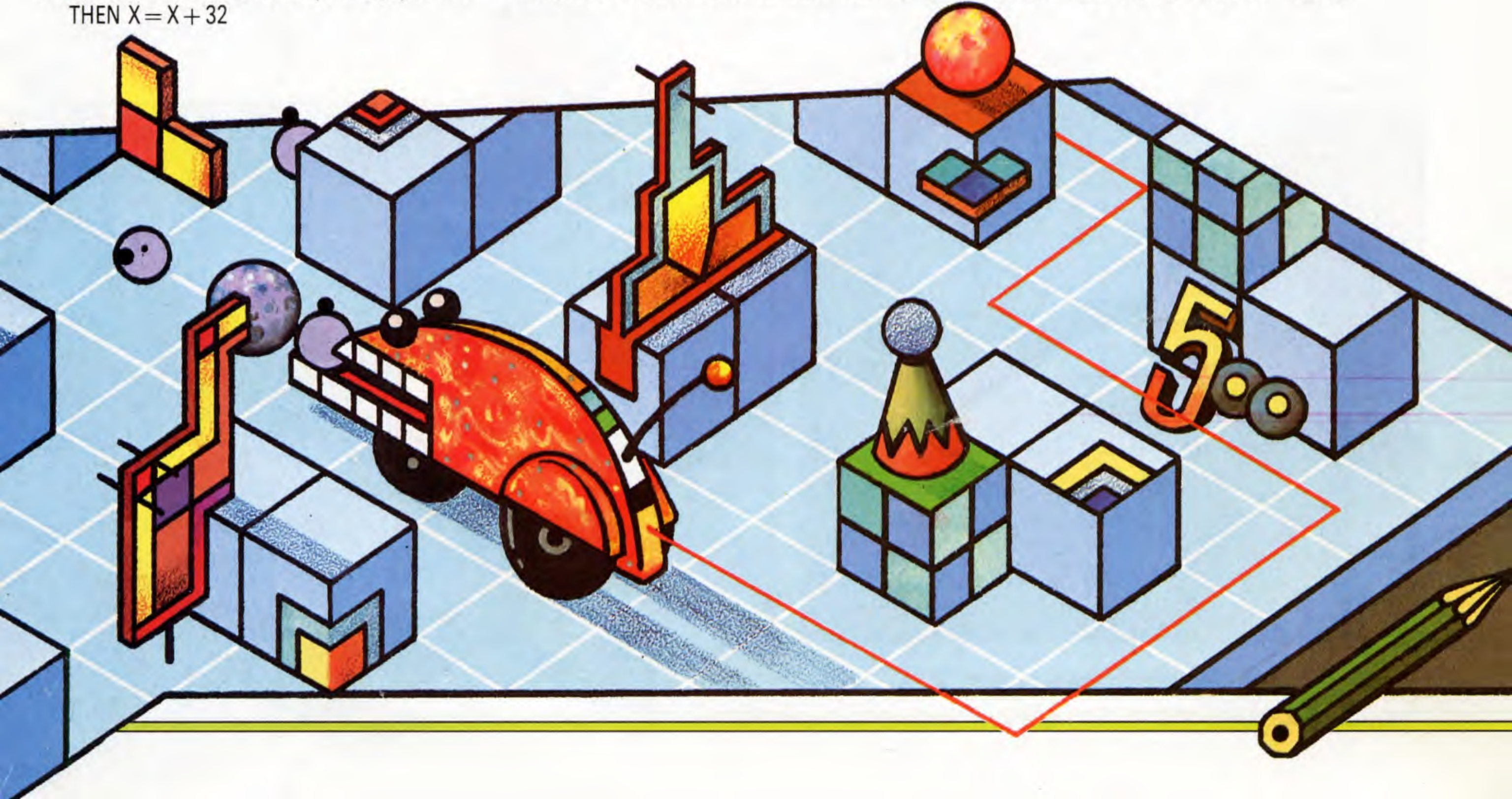
If you RUN the program now you'll find the asterisk moves rather slowly and hesitates each time it changes direction. The two other extra lines, Lines 6Ø and 7Ø, speed up the auto-repeat on the keys so the asterisk moves more quickly. *FX 11,15 sets the delay before auto-repeat starts to 15 hundredths of a second, while *FX 12,15 sets the rate of repeat to be the same.

There is one problem, though. When you finally press ESCAPE you'll find the *FX commands are still in operation, making it more difficult to type because the keys repeat too fast. LIST, for example, can come out as LLIISSTT! The thing to do is press BREAK, then type OLD and LIST. BREAK resets the FX to normal.

TIMING THE GAME

The program is not complete yet as there's no timing and nothing happens when you've eaten all the dots. But the next few lines remedy this:

```
2Ø LET besttime=999999
9Ø LET dots=Ø
1ØØ RESTORE
18Ø TIME=Ø
21Ø IF dots=154 THEN GOTO 31Ø
28Ø IF POINT(X+12,Y-2Ø)=2 THEN
   dots=dots+1 : SOUND Ø,-15,1,1
```



```
310 LET time=TIME/100
320 IF besttime>time THEN besttime=
   time
330 MOVE X,Y :GCOL0,0 :VDU 224
340 *FX12,0
350 VDU 4
360 VDU 23;8202;0;0;0;
370 FOR delay=1 TO 1000: NEXT
380 COLOUR3
390 PRINT TAB(24,10) "TIME□ ="
400 PRINT TAB(24,11);time;"□secs"
410 PRINT TAB(24,13) "BEST TIME"
420 PRINT TAB(24,14);besttime;"□secs"
430 PRINT TAB(24,20) "PRESS RETURN"
440 PRINT TAB(24,21) "TO PLAY AGAIN"
450 IF GET=13 THEN GOTO 50 ELSE
   GOTO 450
```

First of all the best time is set to 999,999 seconds and the number of dots eaten is set to zero. Line 100 is necessary to let the program reREAD the DATA each time the game is played. Without it you would get an OUT OF DATA error after the first game.

Line 280 is the tricky one. This looks to see if you are moving over a dot. It uses the POINT function again and this time checks the colour of the square where the full stop should be. If it is colour 2 (yellow in this case) then the player's score is increased by 1 and a munching sound is made. If the asterisk is running over a blank square then this line is ignored. Fig 3 explains how to work out the position of the dot.

As soon as all the dots have been eaten, Line 210 makes the program jump to Line 310 which stops the timer and divides TIME by 100 to convert it to seconds. This time is then compared with the best time, in Line 320, and if it is faster it calls this new time the best time. Line 330 then blanks out the asterisk.

The next two lines are just 'housekeeping' lines. *FX12,0 resets both the previous *FX commands to normal and VDU 4 undoes the effects of VDU 5.

Finally, your time for this game and the best time are PRINTed out to the right of the maze and you are invited to press RETURN to play again. Line 450 waits for a key and if it is RETURN—code 13—then the game restarts. If any other key is pressed the program just keeps waiting and you'll have to press BREAK when you've really finished.

DESIGNING YOUR OWN MAZE

To design your own maze, draw out the shape on graph paper first. Then convert all the sections of wall to As and all the spaces to dots and you've got your DATA statements for your own game.

If your new maze is larger or smaller than the original one, you'll have to alter some lines in the program as well. These are the lines which set up the FOR ... NEXT loop to READ the DATA—Lines 110 and 130.

The number of dots in the new maze will almost certainly be different, so change Line 210 as well.

That should be all, but if the new maze is a lot wider you may need to alter the position of the text that PRINTs out your time. Play around with the screen display until you get something that looks right.

These first few lines set up the maze itself:

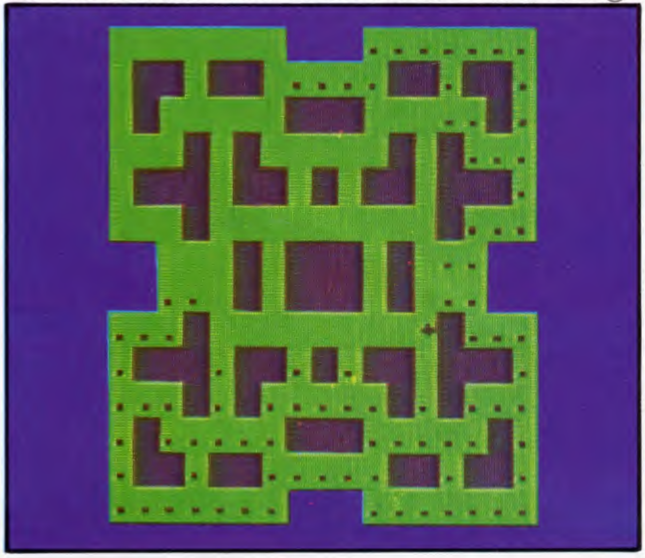
```
30 CLS4
40 LET P=297
1000 FOR N=0 TO 15
1010 READ A$
1020 PRINT@ N*32,A$;
1030 FOR M=0 TO 18
1040 IF PEEK(1024+N*32+M)=65
   THEN POKE(1024+N*32+M),175
1050 NEXT M
1060 NEXT N
2000 DATA A A A A A A A A A A A A A A A A A A A
2010 DATA A . . . . . . . A A A . . . . . . . A
2020 DATA A . A A . A A . . . . . A A . A A . A
2030 DATA A . A . . . . . A A A . . . . . A . A
2040 DATA A . . . A . A . . . . . A . A . . . A
2050 DATA A . A A A . A A . A . A A . A A A . A
2060 DATA A . . . A . . . . . . . . . A . . . A
2070 DATA A A A . . . A . A A A . A . . . A A A
2080 DATA A A A . . . A . A A A . A . . . A A A
2090 DATA A . . . A . . . . . . . . . A . . . A
2100 DATA A . A A A . A A . A . A A . A A A . A
2110 DATA A . . . A . A . . . . . A . A . . . A
2120 DATA A . A . . . . . A A A . . . . . A . A
2130 DATA A . A A . A A . . . . . A A . A A . A
2140 DATA A . . . . . . . A A A . . . . . . . A
2150 DATA A A A A A A A A A A A A A A A A A A A
```

The shape of the maze is contained in the DATA lines—Lines 2000 to 2150. At this stage the boundaries are represented by As.

Before the DATA is read, Line 30 clears the screen and colours it red, whose code number is 4. Line 40 sets the asterisk's start position.

Lines 1000 to 1060 are the section which actually plots the maze on the screen. Each time the FOR ... NEXT loop in Lines 1000 and 1060 is executed the program takes the next line of DATA. Line 1010 READs the DATA and calls it A$, and Line 1020 displays it.

But a collection of dots and As on the screen doesn't look much like a proper maze. So Line 1040 examines the part of the machine's memory which corresponds to each screen location. If the memory location contains the



3. An easy maze for the Electron or BBC B

number 65, it means that there is an A on the screen. If there is, it is changed to a blue block (code 175). Lines 1Ø3Ø and 1Ø5Ø, meanwhile, provide the loop which allows Line 1Ø4Ø to scan each location in turn.

Now add these lines and you'll be able to move a man around the maze. On the Tandy, change the 223s in Lines 11ØØ and 111Ø to 247. Change the 239 in Line 112Ø to 251, and change the 247 in Line 113Ø to 253.

```
1Ø8Ø PRINT @ P,“*”;
1Ø9Ø LET LP=P
11ØØ IF PEEK(34Ø)=223 AND PEEK
   (1Ø23+P)<>175 THEN LET P=P−1
111Ø IF PEEK(338)=223 AND PEEK
   (1Ø25+P)<>175 THEN LET P=P+1
112Ø IF PEEK(338)=239 AND PEEK
   (992+P)<>175 THEN LET P=P−32
113Ø IF PEEK(342)=247 AND PEEK
   (1Ø56+P)<>175 THEN LET P=P+32
115Ø PRINT @ LP,“□”;
117Ø GOTO 1Ø8Ø
```

This section of program is an adaptation of the 'moving around the screen' program on page 13. There are a few differences, though. The man will be moved only if the screen location he is trying to move into doesn't contain a blue block. For example, Line 11ØØ checks two memory locations—the one which corresponds to the Z key and the one which corresponds to the screen location the man is about to enter. If the Z key (code 223) is being pressed *and* the screen location doesn't contain a blue block (code 175) he can move.

The movement control keys are exactly the same as those used previously—Z for left, X for right, P for up, and L for down. Try guiding the asterisk round the maze so that the dots are obliterated.

HOW FAST ARE YOU?

Add these lines and you'll be able to time how long it takes you to squash all the dots:

```
2Ø LET FA=999999
5Ø RESTORE
6Ø LET D=Ø
1Ø7Ø TIMER=Ø
114Ø IF PEEK(1Ø24+P)=11Ø THEN
   PLAY “T8ØB”:LET D=D+1
116Ø IF D=153 THEN GOTO 118Ø
118Ø PRINT@ P,“□”;
119Ø TI=TIMER/5Ø
12ØØ PRINT@52,“TIME=”;
121Ø PRINT@84,TI;“SEC□”;
122Ø IF FA>TI THEN FA=TI
123Ø PRINT@212,“BEST TIME”;
124Ø PRINT@244,FA;“SEC□”;
125Ø PRINT@34Ø,“PRESS”;
126Ø PRINT@372,“ENTER”;
127Ø PRINT@4Ø4,“TO START”;
128Ø PRINT@436,“AGAIN”;
129Ø LET IN$=INKEY$:IF IN$=“”
   THEN GOTO 129Ø
13ØØ IF IN$=CHR$(13) THEN GOTO 4Ø
131Ø GOTO 129Ø
```

When you RUN the program you'll see the time taken to remove the dots shown to the right of the maze, and the best time.

The fastest or best time is set to 999,999 seconds initially by Line 2Ø. Line 6Ø sets the score to zero, and Line 1Ø7Ø resets the timer.

Line 114Ø examines the screen location that the asterisk is in to see if it contains a dot. If it does, the machine's PLAY command is used to make a 'blip' as the dot is run over. At the same time it adds 1 to D, the number of dots that have been wiped out.

Line 116Ø checks if no more dots remain on the screen. If none remains, the program jumps to Line 118Ø which deletes the asterisk.

TI=TIMER/5Ø in Line 119Ø stops the timer and divides the reading by 5Ø to convert it into seconds. Lines 12ØØ and 121Ø display the time on the screen.

Line 122Ø checks if the fastest time (FA) is longer than the time that has just been achieved (TI). If it is, the best time is made equal to the last time. Lines 123Ø and 124Ø display the best time, before the player is told to press the ENTER key to start.

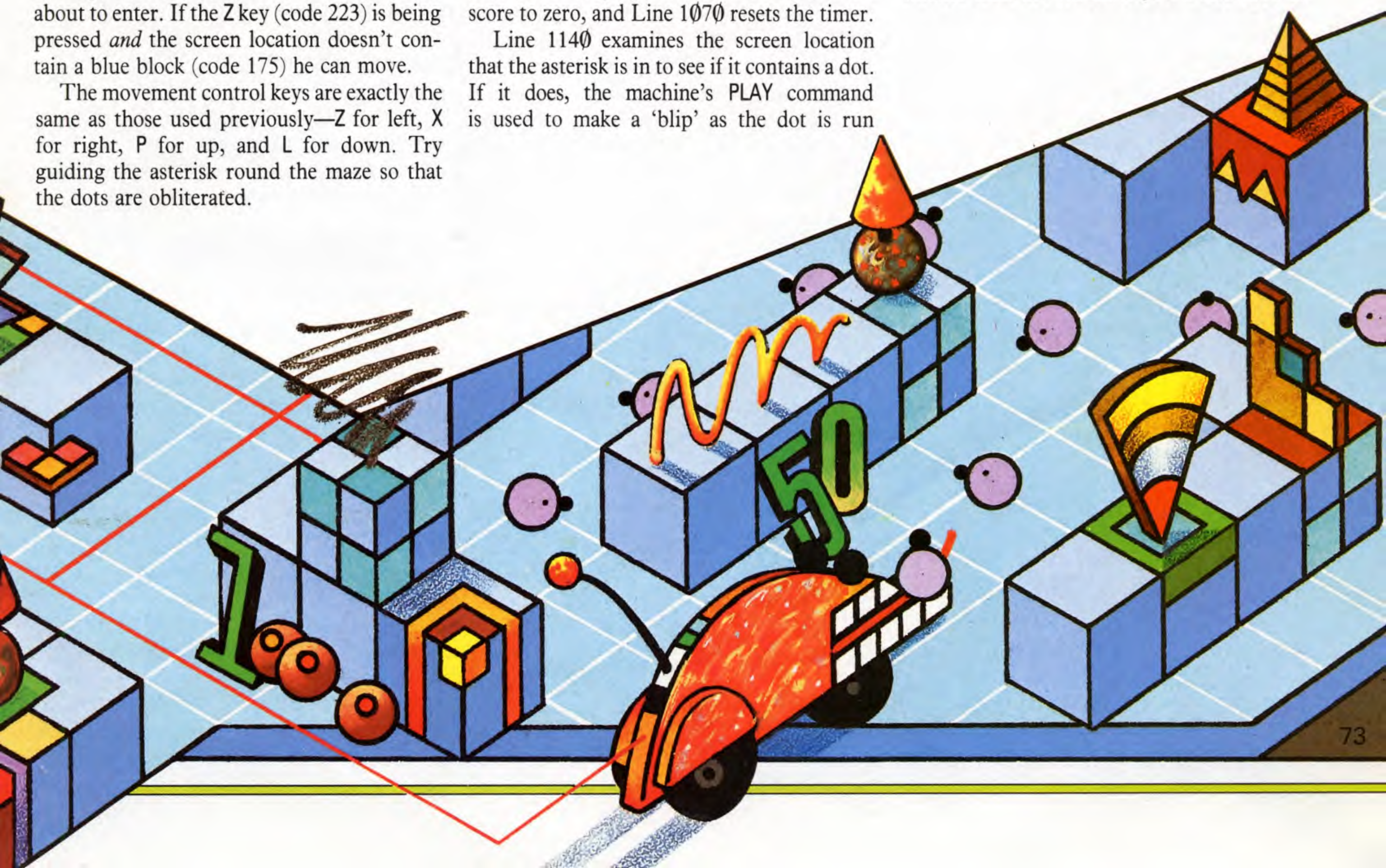
Line 129Ø waits for a key to be pressed, and Line 13ØØ checks if ENTER has been pressed. If it has, the game restarts, the RESTORE in Line 5Ø restoring all the DATA so that the maze can be PRINTed again. If the keypress was anything other than ENTER the program goes back to waiting for another keypress.

Pressing the BREAK key will end the game.

PLANNING YOUR OWN MAZES

Use graph paper to design your own mazes. Bear in mind the size of the text screen on the computer—32 by 16 characters.

Once you've designed the new maze, just alter the contents of the DATA lines at the end of the program. If the new maze is larger or smaller than the old maze you'll also have to



change the FOR lines (Lines 1ØØØ and 1Ø3Ø). Count how many screen lines the new maze occupies, then subtract 1. Now put the result at the end of Line 1ØØØ. Similarly, count the number of screen locations along the top of the new maze, and subtract 1. Put this number, instead of 18, at the end of Line 1Ø3Ø.

Finally, count how many dots there are in the new maze. If the count is different from 153 you'll have to change Line 116Ø too. Just substitute the new number for 153.

This Commodore maze is constructed from PRINT statements and ROM graphics.

RUNning this program displays a predefined maze towards the middle of the screen. Use the Z and X keys to move the π character left or right, and the P and L keys to move it up and down. The dots—'coins' in this game—are erased as π overruns them, giving a count as it does so.

Looking at the program, you can see that the early part, Lines 1Ø to 4Ø, is used for setting the variables. P1 is the screen memory location at the point where the maze 'coins' begin. BD is the border address and BG is the screen (background) colour, changed in Lines 45 and 5Ø to blue and black, respectively. The character colour is set in Line 3Ø: by altering the value in brackets to an acceptable code (see your manual appendix for ASCII and CHR$ codes) you can change the colour of the maze.

Line 35 shows another way you can program a CLR/HOME. Previously, the reverse heart symbol has been used as part of a PRINT statement, to clear the screen and position the cursor at the top left, ready for any following instruction. Well, you can use CHR$(147) in exactly the same way—but for a long program, where memory saving is important, this would be defined as a string variable early in the program RUN, as in Line 35.

Line 55 sets up the repeat key facility which is essential in a maze game such as this.

Then comes the maze PRINTing sequence, followed in Line 2ØØ by a POKE which places π at location 1117, the top left corner of the maze. And then the program displays a keypress request message for the game to begin (or continue). Line 225 has two embedded cursor up-controls and forty blank spaces within the PRINT statement. As soon as SPACE is pressed, Line 225 is PRINTed—and overwrites the keypress request message, a very simple blanking out routine. Another way of programming this is:

```
225 PRINT "◙◙◙":FOR T=Ø TO 39:
    PRINT "□";:NEXT T
```

As you can see, a simple FOR ... NEXT loop can do the same a little more elegantly!

Next, in Lines 3ØØ to 32Ø, there is the keypress detection routine using GET and a succession of IF ... THEN statements to set the value of P2, the new position of the π.

Line 325 PEEKs the location address of π, GOTOing the coin count display if the location does not contain either a space or a coin. If it contains a coin, the coin count register is increased by 1 (Line 33Ø), the previous π position, P1, is blanked off (Line 335) and a new value established for it (Line 34Ø). In effect, Line 325 prevents you overwriting the maze characters.

The program continues with a display of the coin count and the GET routine is repeated until CO equals 1Ø3 in Line 365, whereupon the program is reRUN automatically.

You'll need to make a number of changes to get this to RUN on the Vic. Omit Lines 15 and 45 completely. In Line 1Ø, use 7735 instead of 1117. In Line 2Ø, use 36879 instead of 53281. In Line 25, use 22 instead of 4Ø. In Line 5Ø, use 14 instead of Ø. In Lines 1ØØ to 17Ø, use TAB (3), instead of TAB (12). In Line 21Ø, insert four spaces after PRESS; also insert three extra spaces in Line 225. In Line 36Ø, change TAB (14) to TAB (4).

```
10 LET P1=1117: LET P2=P1
15 LET BD=53280
20 LET BG=53281
25 LET LL=40
30 LET CC$=CHR$(5)
35 LET CS$=CHR$(147)
40 LET CH$=CHR$(19)
45 POKE BD,6
50 POKE BG,0
55 POKE 650,128
60 PRINT CS$,CC$
100 PRINT TAB(12) "┌─────────────┐"
105 PRINT TAB(12) "│.............│"
110 PRINT TAB(12) "│.□□□.□□□.□□□.│"
115 PRINT TAB(12) "│......□......│"
120 PRINT TAB(12) "│□□□.□...□.□□□│"
125 PRINT TAB(12) "│....□...□....│"
130 PRINT TAB(12) "│.□□...■...□□.│"
135 PRINT TAB(12) "│□■■□.■□■.□■■□│"
140 PRINT TAB(12) "│.□□...■...□□.│"
145 PRINT TAB(12) "│.............│"
150 PRINT TAB(12) "│.□□□□□.□□□□□.│"
155 PRINT TAB(12) "│..□.......□..│"
160 PRINT TAB(12) "│□□□□.□□□.□□□□│"
165 PRINT TAB(12) "│......□......│"
170 PRINT TAB(12) "└─────────────┘"
200 POKE P1,94
210 PRINT "▣END OF GAME. PRESS
    (SPACE BAR) TO START."
215 GET K$
220 IF K$<>"□" THEN 215
225 PRINT "◙◙□□□□□□
    □□□□□□□□□□□□
    □□□□□□□□□□□
    □□□□□□□□□□□"
300 GET A$
305 IF A$="Z" THEN P2=P1-1
310 IF A$="X" THEN P2=P1+1
315 IF A$="P" THEN P2=P1-LL
320 IF A$="L" THEN P2=P1+LL
325 IF PEEK(P2)<>32 AND
    PEEK(P2)<>46 THEN GOTO 350
330 IF PEEK(P2)=46 THEN
    CO=CO+1
335 POKE P1,32
340 LET P1=P2: POKE P1,94
350 PRINT CH$;
360 PRINT TAB(14);"COINS :";CO
365 IF CO<103 THEN 300
400 POKE BG,6: CO=0
415 FOR T=1 TO 2000: NEXT T: GOTO 10
```

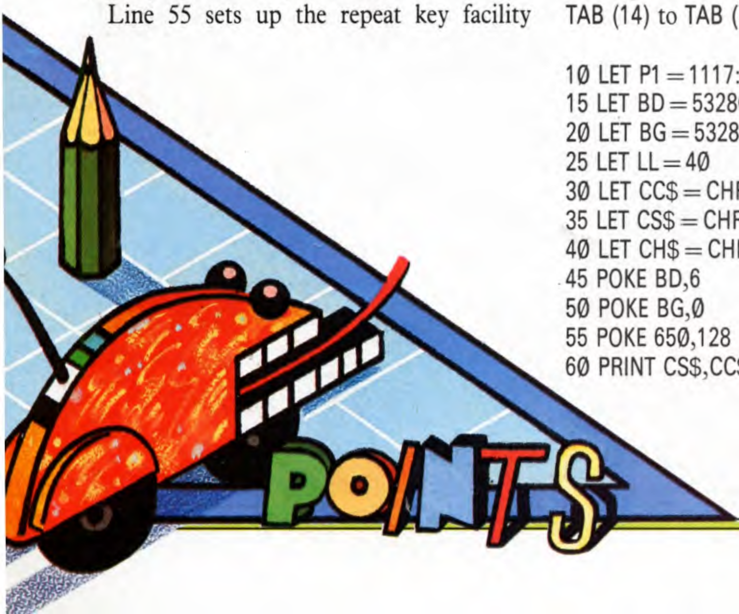
A BIT OF COMPETITION

Now add the timing lines:

```
205 PRINT TAB(12);"▣BEST
    TIME:□";BT
230 LET TI$="000000"
355 PRINT TAB(2);"TIME: ";TI$;
400 IF G=0 THEN BT=VAL(TI$) :
    G=1
405 IF VAL(TI$) <BT THEN BT=
    VAL(TI$)
410 LET CO=0: POKE BG,6
```

CHANGING THE MAZE

If you wish to change the maze, adjust the contents of the PRINT statements in Lines 1Ø5 to 165. You can use any of the ROM symbols and even change the size of the maze—but remember to adjust the value of CO in Line 365 for the number of dots in your maze.



STREAMLINE YOUR HOBBIES FILES-2

- SEARCHING A RECORD
- AMENDING A RECORD
- DELETING A RECORD
- SAVEING ONTO A PRINTER
- USING YOUR HOBBIES FILE

Extracting information from a filing system is easy when you have it on computer. You can find a person's name and address, for example, even if you know only his middle name, or only that he lives in London.

One of the great advantages of computer filing compared with old-fashioned methods is the ease with which you can search through your files. This applies even if you can remember only, for example, the middle name of the person whose name and address you need.

The datafile program in the last article in this series allowed you to set up a highly flexible computer filing system. This article shows how you can search the files for whatever you need; how to amend a record which is out of date or inaccurate; and how to delete an obsolete file.

SEARCHING A RECORD

The search option is number 4 on the main menu. This allows you to search through any of the fields for records containing a particular piece of information.

If you press the 4 key, the computer will ask you which field you want to search. You have to key in the number of the field—1 for the first, 2 for the second, 3 for the third and so on, counting from the top of the screen. Then it will ask you what you want to search that field for. So you key in the word or number it should look for in that field—for example, JOHN, or CASSEROLE, or GAVIIDAE, or DMU—then press RETURN or ENTER.

The word or number you key in must be exactly what is written in the field. If a word is stored in the records in capital letters and you are searching for it in small letters, the computer will not find it. Even the spaces left between words must correspond exactly. If, by mistake, you have left a space before the entry in the record, or accidentally hit the space bar after the entry, the computer will probably not find it. This applies even if the word in the record and the word you are searching for look exactly the same when displayed on the screen.

So, as a precaution, always feed in your records in capital letters and omit any unnecessary spaces, then do the same when searching.

If the computer cannot find any records with the word you have asked for in the field you have asked it to search, it will tell you and return you to the main menu. If it does find some, it will list them in alphabetical order.

Every record has two lines of options at the bottom of the screen. The first reads:

F(ORWARD) B(ACK) M(ENU)

These work in roughly the same way as before, with the F key advancing through the records the search has selected one by one, the B key taking you one by one backwards through the records and M taking you direct to the main menu. But, if you press the F key when you're on the last of the records selected, nothing will happen. B on the first in the search mode gives a 'no records found' message.

Perhaps the most useful application of the search option is to find one particular record. If you can remember anything about that record you should be able to locate it easily enough. Say you were using this file to store the names, addresses and telephone numbers of club members and had the following fields set up: surname, forename, middle name, street, city, county, postcode, phone number. In this case, you would be able to search for a particular record even if you could only remember the person's middle name!

And if the piece of information you remembered was not exclusive to this record—if you knew only that the person whose record you wanted lived in London, or his christian name was John—you would easily be able to call up all the Londoners or all the Johns and flick through them until you found the right one. Even if there were a lot of them that would be a great deal easier than going through the whole file. Cross-referencing like this is virtually impossible without using a computer.

AMENDING A RECORD

The second line of options displayed at the bottom of each record reads:

A(MEND) D(ELETE) P(RINTER)

If you want to change the record displayed on the screen you hit the A key. The computer will then ask you WHICH FIELD NUMBER you want to amend and you must key in the number, counting from the top as before.

Then you'll be told to ENTER MODIFICATION. You have to key in the whole of the new field you want to enter, even if there is only one letter you want to amend. When you hit the RETURN or ENTER key, the computer will write your amendment to the record in the appropriate place. You can then press A again and amend another field.

If you want to amend a record that is not on the screen, you have to press M to get back to the main menu, then choose the search option 4, to locate the record you want to modify, as above, then amend it. Alternatively, you could choose option 3, VIEW RECORDS, find it that way and amend it.

DELETING A RECORD

Again, you first have to locate the record you want to delete by using the search or view options. Then you will be presented with the six standard options—F(ORWARD), B(ACK), M(ENU), A(MEND), D(ELETE) and P(RINTER)—at the bottom of the screen. To delete a record you simply press D. The computer then asks you ARE YOU SURE? This precaution prevents your deleting a record by accident.

If you are sure that you want to delete it you should press Y. The computer will then delete that record and display the next one—either the next one picked alphabetically if you are in the view mode, or the next one that has the same field that you have been searching if you are in the search mode.

If you have deleted the last record found by the search, the computer will tell you that there are no more records with that item in that field and take you back to the main menu.

PRINTING OUT RECORDS

The last option given at the bottom of the screen when a record is shown is P(RINTER). If you press the P key on the Commodore, Dragon, Tandy or Acorn, the computer will ask you to CHECK PRINTER. At this point you must check to see that the printer is connected and switched on, otherwise if you continue with this option and the printer is not working you will have to come out of the program.

On the Spectrum this CHECK PRINTER line is not necessary as the Spectrum will not respond to P if no printer is attached.

Once you have checked that the printer is connected and switched on, you press C to continue. Press any other letters, the CHECK PRINTER line disappears and you can use any of the six standard options at the bottom of the screen again. If you should press C when you had no printer attached, or the printer did not work for whatever reason, you would then have to press the ESCAPE key (Acorn), RUN/STOP (Commodore) or RESET button (Dragon/Tandy) and come out of the program. To get back in again, key in:

```
GOTO 3Ø
```

On the Commodore, use instead:

```
GOTO 1ØØ
```

USING YOUR FILING SYSTEM

You now have a highly flexible filing system which can be used to store the details of just about everything. Of course, the same program can be used to store a whole cabinet worth of files on one long tape.

But it is probably best to store different files on different tapes to make them easier to find. In that case, on computers other than the Spectrum, you will need to SAVE the program itself on a separate tape. This way you can LOAD it before you select option 6 and LOAD the DATA.

This may seem a rather wasteful approach. After all, you will have to find shelf space for a number of cassette tapes, rather than just one or two. But then, think how much space the old-fashioned card-index files would have taken up.

ENTERING THE PROGRAM

The second section of the datafile program—much shorter than the first, you'll be pleased to see—follows.

When you add it to the existing program you will notice that a few line numbers are duplicated. Do not worry about this. The numbers in the first section were there merely to keep the program intact and working so you could test it. The new numbers will replace the earlier ones as you key them in.

```
4ØØØ FOR N=V TO A: PRINT INVERSE V;AT
   N*2,9;N; INVERSE U; TAB 11;":—□";
   N$(N): NEXT N
4Ø1Ø PRINT ''"SEARCH WHICH FIELD
   (1 TO□";A;") ?"
4Ø2Ø IF INKEY$="" THEN GOTO 4Ø2Ø
4Ø3Ø LET Y$=INKEY$: IF CODE Y$<49 OR
   CODE Y$>48+A THEN GOTO 4Ø3Ø
4Ø4Ø BEEP .1,1Ø: LET Z=VAL Y$: PRINT
   '"Search field□";Z;"□for what ?":
   DIM Z$(V, A(Z)): INPUT LINE Z$(V)
4Ø44 CLS : LET K=V
4Ø45 IF A$(K,B(Z)+V TO B(Z+V))=Z$(V)
   THEN GOTO 4Ø5Ø
4Ø46 IF K=R OR A$(K,V)="□" THEN
   CLS : PRINT AT 9,3;"□NO RECORDS
   WITH□";Z$(V),'TAB 1Ø;"IN FIELD□";
   Z: PAUSE 15Ø: RETURN
4Ø47 LET K=K+V: GOTO 4Ø45
4Ø5Ø LET D=V: LET PM=V: LET MO=V
4Ø6Ø IF D>R THEN LET D=PM
4Ø7Ø IF D=U THEN LET D=PM
4Ø8Ø IF A$(D,V)="□" THEN LET D=PM
4Ø9Ø IF A$(D,B(Z)+V TO B(Z+V))
   <>Z$(V) THEN LET D=D+MO:
   GOTO 4Ø6Ø
41ØØ GOSUB 95ØØ
411Ø LET PM=D
412Ø IF OP=V THEN LET MO=V: LET
   D=D+MO: GOTO 4Ø6Ø
413Ø IF OP=2 THEN LET MO=−V: LET
   D=D+MO: GOTO 4Ø6Ø
414Ø IF OP=3 THEN RETURN
415Ø IF OP=4 THEN GOSUB 8ØØØ
416Ø IF OP=5 THEN LET DF=U: LET
   MD=2: GOSUB 9ØØØ: IF DF=V OR
   A$(V,V)="□" THEN RETURN
417Ø GOTO 41ØØ
8ØØØ INPUT AT U,U;"AMEND which field (1
   TO□";(A);") ? ";J: IF J>A THEN
   GOTO 8ØØØ
8Ø1Ø PRINT FLASH V;AT V+2*J,12;A$
   (D,B(J)+V TO B(J+V)): INPUT AT
   U,U;"Enter modified field now", LINE
   A$(D,B(J)+V TO B(J+V)): PRINT AT
   V+2*J,12;A$(D,B(J)+V TO B(J+V))
8Ø2Ø IF D=R THEN LET J=−V:
   GOTO 8Ø7Ø
8Ø3Ø IF D=V THEN LET J=V: GOTO 8Ø6Ø
8Ø4Ø IF A$(D)>A$(D+V) THEN LET J=V
8Ø5Ø IF A$(D)<A$(D−V) THEN LET
   J=−V
8Ø6Ø IF A$(D+V,V)="□" AND J=V THEN
   GOTO 85ØØ
8Ø7Ø IF J=V THEN GOTO 81ØØ
8Ø8Ø IF A$(D)>=A$(D−V) THEN GOTO
   85ØØ
8Ø9Ø LET X$=A$(D): LET A$(D)=
   A$(D−V): LET A$(D−V)=X$: LET
   D=D−V: GOTO 8Ø2Ø
81ØØ IF A$(D)<=A$(D+V) THEN GOTO
   85ØØ
811Ø LET X$=A$(D): LET A$(D)=
   A$(D+V): LET A$(D+V)=X$: LET
   D=D+V: GOTO 8Ø2Ø
85ØØ BEEP .1,1Ø: PRINT AT U,U;"Record
   number□";D;"□□□": RETURN
```

Q+A

Why do long programs that I type in sometimes fail to RUN properly?

Typing in long programs that someone else has written can be a laborious, even daunting, task. To make it easier—and more accurate—type in, then check, only short sections at a time. Many programs are structured into self-contained sections, such as subroutines, loops or procedures. The easiest way to check each section is to type GOTO, followed by the line number at which the section begins. If the section requires a value from a variable from another part of the program, you may be able to identify it and type it in. If you cannot see, or cannot isolate, such a structure then type in only short sections of, say, 2Ø to 3Ø lines at a time, checking each block carefully on the TV screen as you go. Fortunately, the job becomes easier with time—the more you know about your programs, the easier other people's are.

If you have typed in the whole program and it still does not work, more drastic methods of diagnosis need to be employed. Identify the variables and try PRINTing them out at various points during the program by adding temporary PRINT lines at appropriate points. You may be able to spot when a variable takes on an inappropriate value while the program is being RUN.

Program alterations for the Electron

A few alterations must be made to get the program to run on the Electron. Change Lines 5040 onwards to:

```
5038 Q=-1
5040 PROCFIND
5041 IF C< >1 THEN PROCVDU:
   PROCKEY:$B%=A$
5042 UNTIL Q=1 OR C=0
5043 IF Q=1 THEN 5045
5044 CLS:PRINTTAB(12,5)“NO
   RECORDS WITH”TAB(20—LEN(A$)/
   2,7)A$TAB(15,9)“IN FIELD”;
   F:G=INKEY(300)
5045 ENDPROC
5047 DEF PROCFIND
5050 T=0
5060 D%=D%+C
5065 IF D%<1 OR D%>N%-1 THEN
   C=-C:T=T+1:D%=D%+C
5070 IF $B%=$(B%+D%*R%+TRL
   (F)) THEN 5100
5080 IF T<2 THEN 5060
5090 C=0
```

```
9000 PRINT AT 19,U;“ARE YOU SURE YOU
   WISH TO DELETE?”: PAUSE U
9010 IF INKEY$=“” THEN
   GOTO 9010
9020 IF INKEY$< >“Y” THEN PRINT AT
   19,U;“□□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□”: BEEP .1,10: RETURN
9030 PRINT AT 19,U;“□DELETING
   □□□□□□□□□□□□□□□
   □□□□□□□□”
9035 LET DD=D
9040 IF D=R THEN LET DD=DD-V:
   GOTO 9060
9050 IF A$(D+V,V)< >“□” THEN LET
   A$(D)=A$(D+V): LET D=D+V:
   GOTO 9040
9060 LET A$(D)=“”: FOR F=V TO 100:
   NEXT F: PRINT AT 19,U;“□□□
   □□□□□□□”
9070 LET D=DD: IF A$(V,V)=“□” THEN
   LET D=U: RETURN
9072 IF D=U THEN LET D=V
9075 IF A$(D,V)=“□” THEN LET D=D-V
9080 IF MD=V THEN RETURN
9090 LET K=V
9100 IF A$(K,B(Z)+V TO B(Z+V))=Z$(V)
   THEN GOTO 9130
9110 IF K=R OR A$(K,V)=“□” THEN LET
   DF=V: GOTO 4046
9120 LET K=K+V: GOTO 9100
9130 LET DD=D: LET PA=V
9140 IF A$(DD,V)=“□” OR DD=U THEN
   LET PA=2: LET DD=D: LET MO=
   MO-V
9150 IF A$(DD,B(Z)+V TO B(Z+V))=Z$(V)
   THEN LET D=DD: RETURN
9160 LET DD=DD+MO: GOTO 9140
```

```
3020PRINT'“Which field number ?”
3035 R=GET-48:IF R<1 OR R>A□
   THEN 3035
3060VDU11:PRINT“Enter modification□□:”;:
   PROCINPUT(A(R)):P.'
3065IF LEN($B%)<1 THEN 3060
3070$(B%+D%*R%+TRL(R))=$B%
3080IF D%=1 THEN3140
3090X=B%+D%*R%:Y=B%+(D%-1)*R%
3100IF $X>=$Y□THEN3130
3110FOR T=1 TOA:$B%=$(X+TRL(T)):
   $(X+TRL(T))=$(Y+TRL(T)):
   $(Y+TRL(T))=$B%:NEXT
3120D%=D%-1:IF D%=1 THEN 3180 ELSE
   3090
3130IF D%=N%-1 THEN 3180
3140X=B%+D%*R%:Y=B%+(D%+1)*R%
3150IF $X<=$Y□THEN 3180
3160FOR T=1 TOA:$B%=$(X+TRL(T)):
   $(X+TRL(T))=$(Y+TRL(T)):
   $(Y+TRL(T))=$B%:NEXT
3170D%=D%+1:GOTO3130
3180D%=D%-C
4010VDU11,11:PRINT“Are you sure you want to
   delete this□?□□”;STRING$(40,“□”)
4020Q=GET AND &5F:IF Q< >89 THEN
   ENDPROC
4030IF D%=N%-1 THEN 4080
4050FOR T=D%+1 TO N%
4060FOR Q=1 TO A:$(B%+R%*(T-1)+
   TRL(Q))=$(B%+R%*(T)+TRL(Q)):NEXT
4070NEXT
4080N%=N%-1:D%=D%-C
5001IF N%=1 THEN ENDPROC
5005CLS:FOR N=1 TO A:PRINT';N,N$(N):NEXT
5007D%=0:C=1:Q=0
5010PRINT'“Search which field ?”;
5020F=GET-48:IF F<1 OR F>A□THEN
   5020
```

From Lines 5030 to 5090 BBC users can use the lines shown in the panel, top-left.

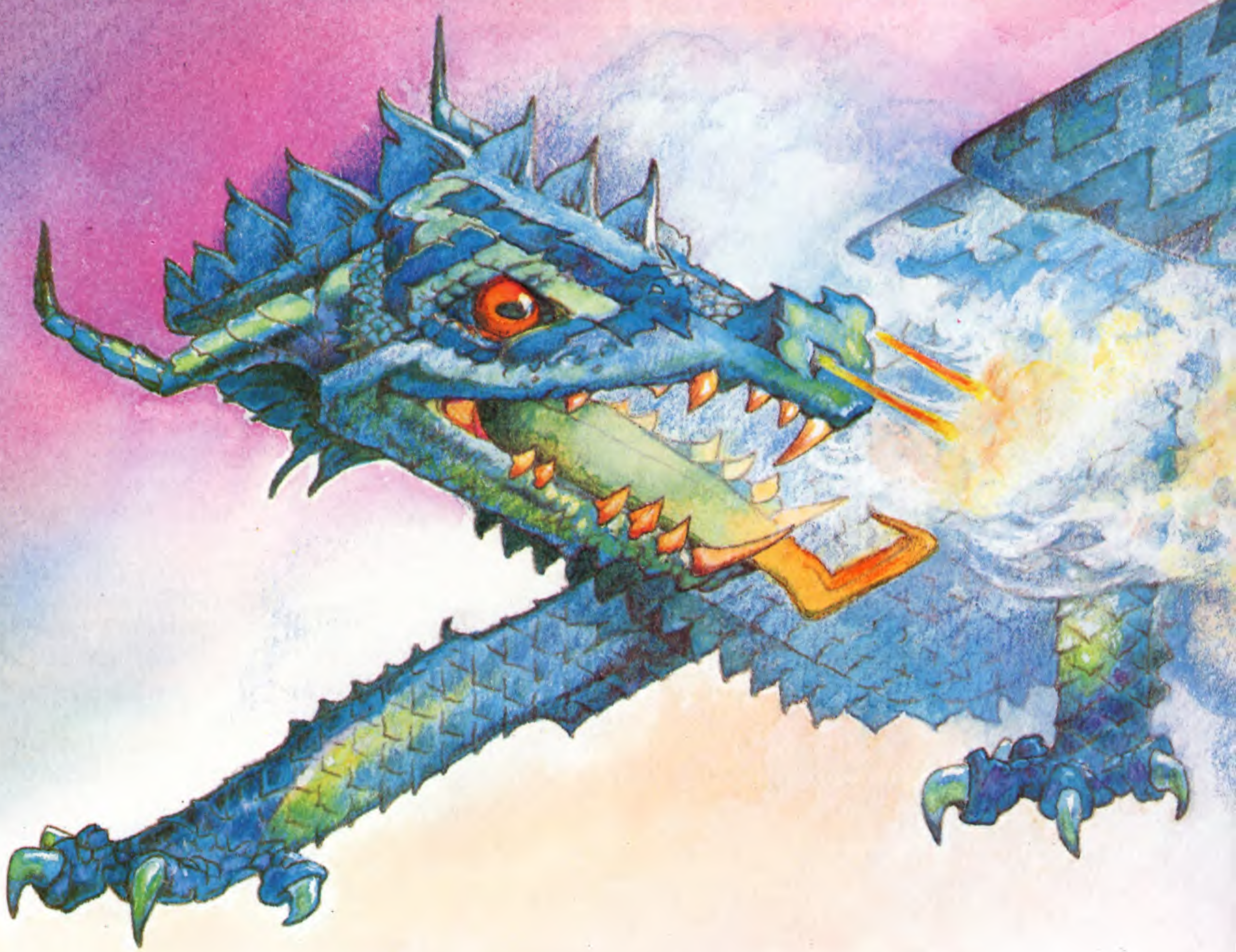
```
5030PRINT'“Search field□”;F;“□for what
   ?□”;
5035PROCINPUT(A(F)):A$=$B%:P.
5037REPEAT
5040PROCFIND
5041IF C< >0 THEN PROCVDU:PROCKEY:Q=0
5042D%=D%+C
5043UNTIL Q=2:Q=1
5044CLS:PRINTTAB(12,5)“No records with”
   TAB(20-LEN(A$)/2,7)A$TAB(15,9)
   “in field□”;F:G=INKEY(300)
5045ENDPROC
5047DEF PROCFIND
```

```
5050T=0:REPEAT
5070IF $B%=$(B%+D%*R%+TRL(F))
   THEN T=2:GOTO 5080
5073D%=D%+C
5075IF D%>N%-2 OR D%<2 THEN
   T=T+1:C=-C:Q=Q+1
5080UNTIL T>1
5090IF Q=2 THEN C=0
```

```
3000 GOSUB8500
3010 PRINT@417,“MODIFY WHICH FIELD
   (1 TO”;A;“)□?”;
3020 IN$=INKEY$:IFIN$=“” THEN3020
3030 J=VAL(IN$)
3040 IFJ<1 ORJ>A THEN3020
3050 PRINT@448,“”
3060 PRINT@45+32*J,“”:PRINT@417,
   “ENTER MODIFIED FIELD NOW—”:
   LINE INPUTA$(D,J)
3070 A$(D,J)=LEFT$(A$(D,J),A(J))
3080 IFD=R THENJ=-1:GOTO3130
3090 IFD=1 THENJ=1:GOTO3120
3100 IFA$(D,1)>A$(D+1,1) THENJ=1
```

```
3110 IFA$(D,1)<A$(D-1,1) THENJ=-1
3120 IFA$(D+1,1)="" AND J=1
   THEN318Ø
313Ø IFJ=1 THEN316Ø
314Ø IFA$(D,1)>=A$(D-1,1) THEN318Ø
315Ø FORN=1TOA:X$=A$(D,N):
   A$(D,N)=A$(D-1,N):A$(D-1,N)=X$:
   NEXT:D=D-1:GOTO3Ø8Ø
316Ø IFA$(D,1)<=A$(D+1,1) THEN318Ø
317Ø FORN=1TOA:X$=A$(D,N):
   A$(D,N)=A$(D+1,N):A$(D+1,N)=X$:
   NEXT:D=D+1:GOTO3Ø8Ø
318Ø FORF=1TO3ØØ:NEXT:RETURN
4ØØØ G=D
4Ø1Ø GOSUB85ØØ
4Ø2Ø PRINT@449,"ARE YOU SURE YOU
   WISH TO □□□□□□□□
   DELETE ?";
4Ø3Ø IN$=INKEY$:IFIN$="" THEN4Ø3Ø
4Ø4Ø IFIN$<>"Y" THENRETURN
4Ø5Ø IFG=NR THENG=G-1
4Ø6Ø CLS7:PRINT@236,"DELETING";:
   SOUND3Ø,1
4Ø7Ø IFD=NR THEN4Ø9Ø
4Ø8Ø FORN=1TOA:A$(D,N)=A$(D+1,N):
   NEXT:D=D+1:GOTO4Ø7Ø
4Ø9Ø FORN=1TOA:A$(D,N)="":NEXT:
   NR=NR-1:D=G
41ØØ FORF=1TO4ØØ:NEXT:RETURN
5ØØØ FORN=1TOA:PRINT@33+32*N,N,
   N$(N):NEXT
5Ø1Ø PRINT@417,"SEARCH WHICH FIELD
   (1 TO";A;")□?";
5Ø2Ø IN$=INKEY$:IFIN$="" THEN5Ø2Ø
5Ø3Ø IFVAL(IN$)<1
   OR VAL(IN$)>A THEN5Ø2Ø
5Ø4Ø SOUND3Ø,1:Z=VAL(IN$):
   PRINT@417,"SEARCH FIELD";Z;"FOR
   WHAT□?"
5Ø5Ø LINEINPUTZ$
5Ø6Ø D=1:G=Ø:CH=1
5Ø7Ø IFD>NR ANDG=1 THENG=Ø:
   CH=-1 ELSEIFD>NR THEN523Ø
5Ø8Ø IFD<1 ANDG=1 THENG=Ø:CH=1
   ELSEIFD<1 THEN523Ø
5Ø9Ø IFA$(D,Z)<>Z$ THEND=D+CH:
   GOTO5Ø7Ø
51ØØ LD=D:G=1:GOSUB85ØØ
511Ø PRINT@451,"fORWARDS□□□□
   bACKWARDS□□□mENU□□aMEND
   □□□□□dELETE□□□□□
   pRINT";
512Ø IN$=INKEY$:IFIN$="" THEN512Ø
513Ø IN=INSTR(1,RS$,IN$)
514Ø ON IN GOTO 516Ø,516Ø,517Ø,
   518Ø,519Ø,52ØØ,521Ø
515Ø GOTO512Ø
516Ø CH=1:D=D+1:GOTO5Ø7Ø
517Ø CH=-1:D=D-1:GOTO5Ø7Ø
518Ø RETURN
519Ø GOSUB3ØØØ:GOTO5Ø9Ø
52ØØ GOSUB4ØØØ:GOTO5Ø9Ø
521Ø GOSUB1ØØØØ:GOTO5Ø7Ø
522Ø GOTO511Ø
523Ø CLS2:PRINT@2ØØ," NO RECORD
   WITH□";:PRINT@271-LEN(Z$)/2,"□";
   Z$;"□";
524Ø PRINT@33Ø,"□IN FIELD";Z;
525Ø FORG=1TO5:SCREENØ,1:
   FORF=1TO5ØØ:NEXT:SCREENØ,Ø:
   FORF=1TO5ØØ:NEXTF,G:RETURN
```

```
34ØØ ix$=au$(ix):gosub145ØØ
342Ø ifaa$="n"thengosub372Ø:goto31ØØ
343Ø u=u-1
344Ø ifu=Øthenreturn
345Ø ru=r(up):ifup=uthen347Ø
346Ø fordn=uptou-1:r(dn)=r(dn+1):next
347Ø ifru=uthen351Ø
348Ø fordn=Øtou-1:ifr(dn)<>uthen35ØØ
349Ø r(dn)=ru:forn=1tonf:t$(ru,n-1)=
   t$(u,n-1):next:dn=u
35ØØ next
351Ø ifup=uthen398Ø
352Ø goto3Ø4Ø
37ØØ gosub115ØØ:printx2$ro$"□□□□
   □□□□□□□□□□□□□
   □□□□□□□□□□□□□
   □□□□□□□□□□";
37Ø5 print"□□□□PLEASE INPUT FIELD
   NUMBER□"x3$;
371Ø ok$=left$("12345678",nf):
   gosub1Ø6ØØ
3712 gosub372Ø:goto38ØØ
4ØØØ printcs$x1$gr$"□WHICH FIELD IS TO
   BE SEARCHED?:"cc$"□"cl$;
4Ø1Ø ok$=left$("12345678",nf):
   gosub1Ø6ØØ
4Ø3Ø printchr$(ix+48):fx=ix
4Ø4Ø printx1$gr$"□WHAT ARE YOU
   LOOKING FOR?:"cc$
4Ø5Ø y=8:x=Ø:gosub115ØØ:printro$hd$(ix)
   "□:?";:x=len(hd$(ix))+4
4Ø6Ø z=lx(ix):gosub13ØØØ:ifright$(ix$,1)=
   "□"thenix$=left$(ix$,len(ix$)-1)
4Ø7Ø ifix$=""thengosub1ØØØØ:goto4Ø5Ø
4Ø8Ø fx$=ix$:ff=len(fx$)
41ØØ goto398Ø
```



DRAW A FIRE-BREATHING DRAGON

The 3 by 3 frame that you learned how to create earlier can be used for a wide variety of graphics. Here's one—a fearsome dragon that walks and breathes fire

A fire-breathing dragon is a useful character to add some thrills to the adventure games you're going to write. In fact, no tale of sword and sorcery would be complete without one.

Far from being a difficult character to create, move around and ignite, you can make a dragon in exactly the same way as you created the tank and the frog on pages 8 to 15. The movement routines are also identical and the fire-breathing mechanism is a good deal simpler than the one which fired the shell from the tank.

To create a dragon, you first have to set up a frame or grid in the computer's memory. Exactly the same grid as the one you set up for the tank and the frog will be suitable.

If you SAVEd that frame-construction program on tape when you keyed it in to build the tank, you can simply LOAD it back into your computer off tape and then key in the dragon-making program for your machine listed below. If not, you will have to key in the frame-construction program on page 8 for the Spectrum, page 11 for the Electron or the BBC micro, or page 13 for the Dragon/Tandy.

If you don't want to SAVE this useful program on tape this time either, leave out the SAVE lines—that is, Line 5Ø in the Spectrum program, Line 4Ø in the BBC or Electron program and Lines 6Ø to 12Ø in the Dragon and Tandy program.

Once you've keyed in the program, with or without the omissions, you must RUN it. This stores the machine code in the machine's memory. When the program has finished RUNning and you get an OK message on the Spectrum, Dragon and Tandy, or the cursor returns on the BBC and Electron, you type in NEW and hit the ENTER or RETURN key.

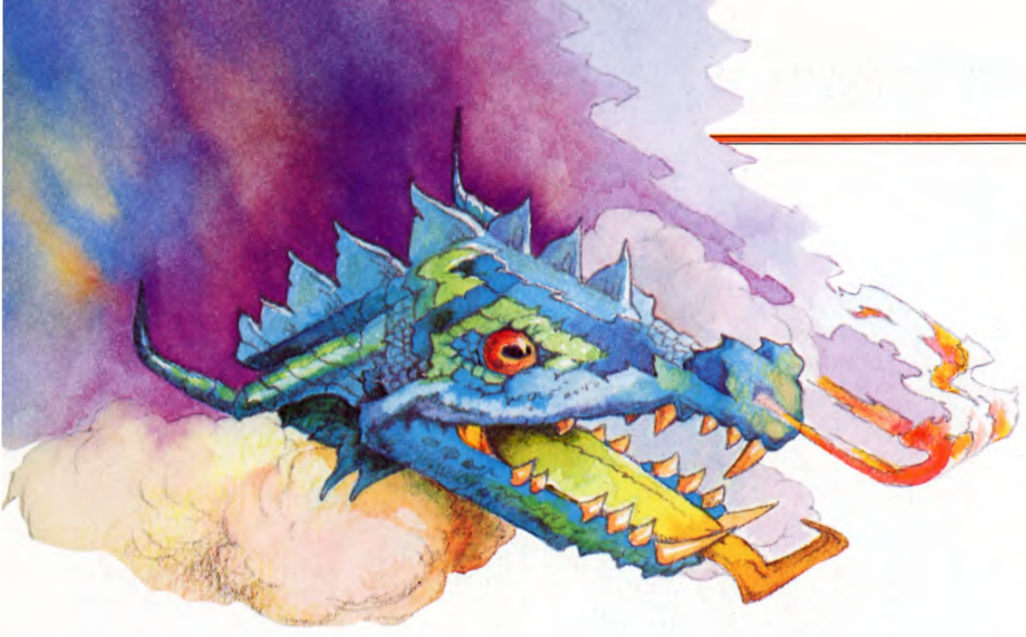
This clears the computer's BASIC memory area and minimizes the chance of your dragon program being corrupted or spoiled by leftover program lines.

Then type in the following program lines for your machine, being careful to check the numbers in each line beginning DATA before you press RETURN to go on to the next line. When you RUN the program you will get a dragon on the screen. Press Z and it will walk

- FITTING A NEW CHARACTER INTO A GRAPHICS FRAME
- MOVING AROUND THE SCREEN
- HOW TO MAKE YOUR DRAGON BREATHE FIRE

to the left, press X and it will walk to the right, P and L move it up and down and hitting the space bar makes it breathe fire.

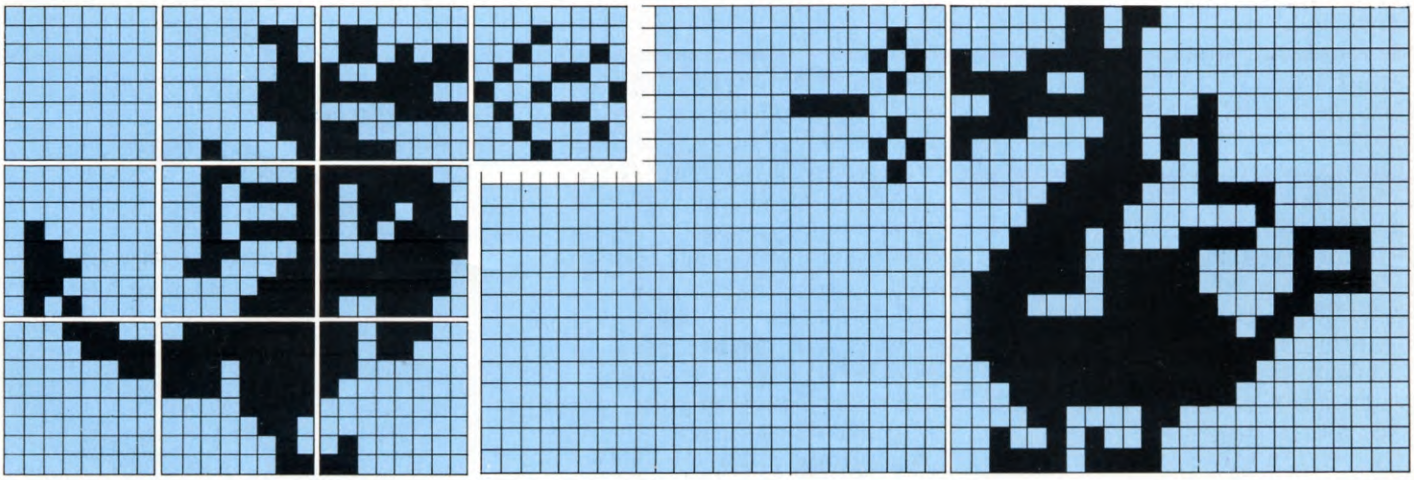
```
10 FOR n=USR "a" TO USR "t"+7: READ
   a: POKE n,a: NEXT n
20 LET print=32400: LET b=32402: IF
   PEEK 23733=255 THEN LET print=65200:
   LET b=65202
90 BORDER 0: PAPER 0: INK 4: CLS:
   PRINT AT 8,15;: RANDOMIZE USR print
100 LET y=8: LET x=15: LET y1=8:
   LET x1=15: LET z=1
110 LET a$=INKEY$: IF a$=""
   THEN GOTO 110
115 IF a$="□" THEN GOTO 200
120 IF a$="z" AND x>1 THEN LET
   x1=x-1: LET z=1
130 IF a$="x" AND x<28 THEN LET
   x1=x+1: LET z=2
140 IF a$="p" AND y>0 THEN LET
   y1=y-1
150 IF a$="l" AND y<19 THEN LET
   y1=y+1
160 PRINT AT y,x;: POKE b,0:
   RANDOMIZE USR print
170 LET x=x1: LET y=y1
180 PRINT AT y,x;: POKE b,z:
   RANDOMIZE USR print
190 GOTO 110
200 IF z=2 THEN GOTO 300
220 PRINT INK 6;AT y,x-1;CHR$ 162
230 PAUSE 1
240 PRINT AT y,x-1;"□"
260 GOTO 110
300 PRINT INK 6;AT y,x+3;CHR$ 163
310 PAUSE 1
320 PRINT AT y,x+3;"□"
330 GOTO 110
1000 DATA 6,214,249,63,240,0,3,15,96,
   64,192,224,224,192,196,204,0,0,0,0,
   0,0,0,0
1010 DATA 63,125,107,245,249,127,127,
   51,244,196,194,255,128,192,224,240,
   0,0,0,2,6,14,6,10
1020 DATA 55,23,7,3,1,0,4,7,254,255,
   239,239,224,192,128,128,56,240,192,
   0,0,0,0,0
1030 DATA 0,0,0,0,0,0,0,0,6,2,3,7,7,
   3,35,51,96,107,159,252,15,0,192,240
```



```
1040 DATA Ø,Ø,Ø,64,96,112,96,8Ø,47,
   33,65,255,1,3,7,15,252,19Ø,182,
   175,159,254,254,2Ø4
1Ø5Ø DATA 28,15,3,Ø,Ø,Ø,Ø,Ø,127,255,
   247,247,7,3,1,1,236,232,224,192,
   128,Ø,32,224
1Ø6Ø DATA 8,68,5Ø,139,5Ø,68,8,Ø,16,
   34,76,145,76,34,16,Ø
```

On the Tandy, you will need to change the 239 in Line 3ØØ to 251. Also change the 247 in Line 31Ø to 253, and the 223s in Line 32Ø and 33Ø to 247:

```
2Ø PCLEAR5
3Ø CLEAR2ØØ,32ØØØ
4Ø FORI=323ØØ TO 32587
5Ø READ N
6Ø POKE I,N
7Ø NEXT
8Ø DIMA(2),B(2),C(2)
9Ø PMODE4,1
1ØØ PCLS
11Ø FORI=1536 TO 176Ø STEP32
12Ø READ N
13Ø POKEI,N
14Ø NEXT
15Ø GET(Ø,Ø)-(7,7),A
16Ø FORI=1536 TO 176Ø STEP32
17Ø READ N
18Ø POKE I,N
19Ø NEXT
2ØØ GET(Ø,Ø)-(7,7),B
21Ø PCLS
22Ø SCREEN1,Ø
23Ø T=1
24Ø DP=35ØØ
25Ø POKE327ØØ,INT(DP/256)
26Ø POKE 327Ø1,DP-256*INT(DP/256)
27Ø POKE3225Ø,T+DT*2
28Ø EXEC32ØØØ
29Ø LP=DP
3ØØ IFPEEK(338)=239 THENDP=DP-32:
   GOTO36Ø
31Ø IFPEEK(342)=247 THENDP=DP+32:
   GOTO36Ø
32Ø IFPEEK(34Ø)=223 THENDP=DP-1:
   T=T+1:DT=1:GOTO36Ø
33Ø IFPEEK(338)=223 THENDP=DP+1:
   T=T+1:DT=Ø:GOTO36Ø
34Ø IFINKEY$="□" AND T=2 GOSUB41Ø
35Ø GOTO3ØØ
36Ø IFDP<1536 OR DP>6941
   THENDP=LP
37Ø IFT>2 THENT=1
38Ø POKE3225Ø,Ø
39Ø EXEC32ØØØ
4ØØ GOTO25Ø
41Ø IF DT=1 GOTO47Ø
42Ø YP=INT((DP-1536)/32)
43Ø XP=8*(DP-1533-YP*32)
44Ø IFXP>255 THEN53Ø
45Ø PUT(XP,YP)-(XP+7,YP+7),A
46Ø GOTO51Ø
47Ø YP=INT((DP-1536)/32)
48Ø XP=8*(DP-1537-YP*32)
49Ø IFXP<Ø THEN53Ø
5ØØ PUT(XP,YP)-(XP+7,YP+7),B
51Ø FORI=1TO2ØØ:NEXT
52Ø PUT(XP,YP)-(XP+7,YP+7),C
53Ø RETURN
54Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,6,2,3,7,7,3,
   35,51,96,1Ø7,159,252,15,Ø,192,24Ø
55Ø DATA Ø,Ø,Ø,Ø,64,96,112,96,47,33,
   65,255,3,7,15,31,248,188,182,175,
   159,255,254,254
56Ø DATA 8Ø,28,15,7,1,Ø,Ø,Ø,63,255,
   255,255,255,15,2,3,252,248,252,252,
   248,48,82,222
57Ø DATA Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,6,2,3,7,7,
   3,33,Ø,96,1Ø7,159,252,15,192,24Ø
58Ø DATA Ø,Ø,Ø,64,96,112,96,8Ø,49,47,
   33,47,49,99,7,15,252,19Ø,182,175,
   159,254,254,156
59Ø DATA 28,15,3,Ø,Ø,Ø,Ø,Ø,127,255,
   239,239,15,6,2,3,22Ø,216,192,128,Ø,
```

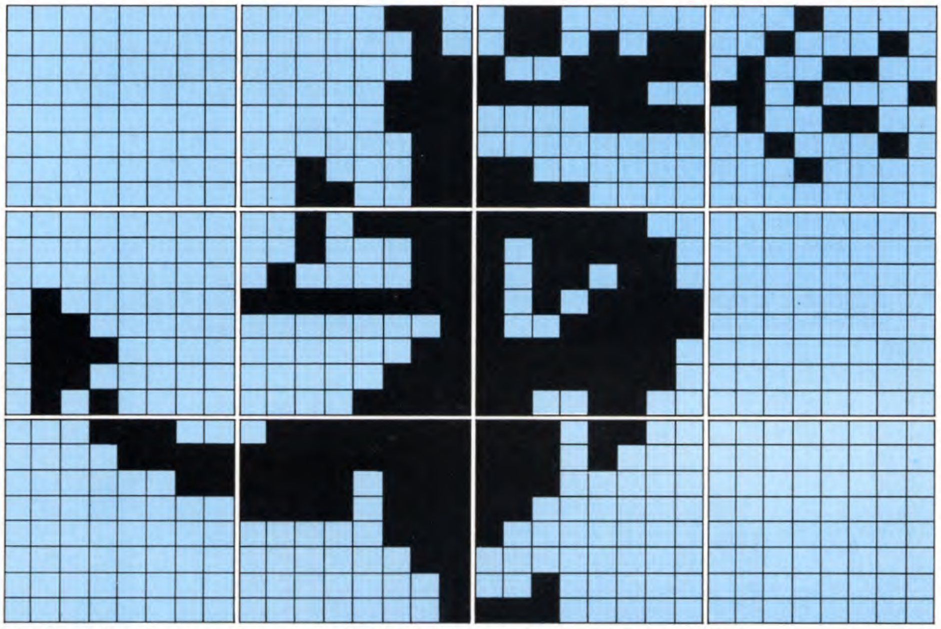


1. The Dragon (and Tandy) dragon is made up from standard UDGs

2. The Commodore version uses this machine's 'sprites'

```
    0,64,192
600 DATA 6,214,249,63,240,0,3,15,96,
    64,192,224,224,192,196,204,0,0,0,0,
    0,0,0,0
610 DATA 31,61,109,117,249,255,127,
    127,244,196,194,255,192,224,240,248,
    0,0,0,0,2,6,14,6
620 DATA 63,31,63,63,31,12,74,123,252,
    255,255,255,255,240,64,192,10,56,
    240,224,128,0,0,0
630 DATA 0,6,214,249,63,240,3,15,0,
    96,64,192,224,224,192,132,0,0,0,0,
    0,0,0,0
640 DATA 63,125,109,245,249,127,127,
    57,140,244,132,244,140,198,224,
    240,0,0,0,2,6,14,6,10
650 DATA 59,27,3,1,0,0,2,3,254,255,247,
    247,240,96,64,192,56,240,192,0,0,
    0,0,0
660 DATA 0,16,34,76,145,76,34,16
670 DATA 0,8,68,50,137,50,68,8
```

```
10 MODE1
20 VDU23;8202;0;0;0;
30 FOR T=224 TO 243:VDU23,T
40 FOR P=1 TO 8
50 READ A
60 VDUA:NEXT
70 NEXT:VDU 19,3,2,0,0,0:CALL&D00
80 X=0:Y=0:X2=0:Y2=0:Z=1
90 A$=GET$
100 IF A$="Z" AND X>0 THEN X2=X-1:
    Z=1
110 IF A$="X" AND X<37 THEN
    X2=X+1: Z=2
120 IF A$="L" AND Y<29 THEN
    Y2=Y+1
130 IF A$="P" AND Y>0 THEN Y2=Y-1
140 IF A$="□" THEN 200
150 VDU31,X,Y:X%=0:CALL &D00
160 X=X2:Y=Y2
170 VDU31,X,Y
180 X%=Z:CALL &D00
190 GOTO 90
200 IF Z=2 THEN 240
210 VDU31,X-1,Y,242,8
220 A$=INKEY$(5):VDU32
230 GOTO 90
240 VDU31,X+3,Y,243,8
250 A$=INKEY$(5):VDU32
260 GOTO 90
270 DATA 6,214,249,63,240,0,3,15,96,
    64,192,224,224,192,196,204,0,0,0,0,
    0,0,0,0
280 DATA 63,125,109,245,249,127,127,
    51,244,196,194,255,128,192,224,240,
    0,0,0,2,6,14,6,10
290 DATA 55,23,7,3,1,0,4,7,254,255,
    239,239,224,192,128,128,56,240,192,0,
    0,0,0,0
```



3. How the Spectrum and Acorn dragons are built up from UDGs

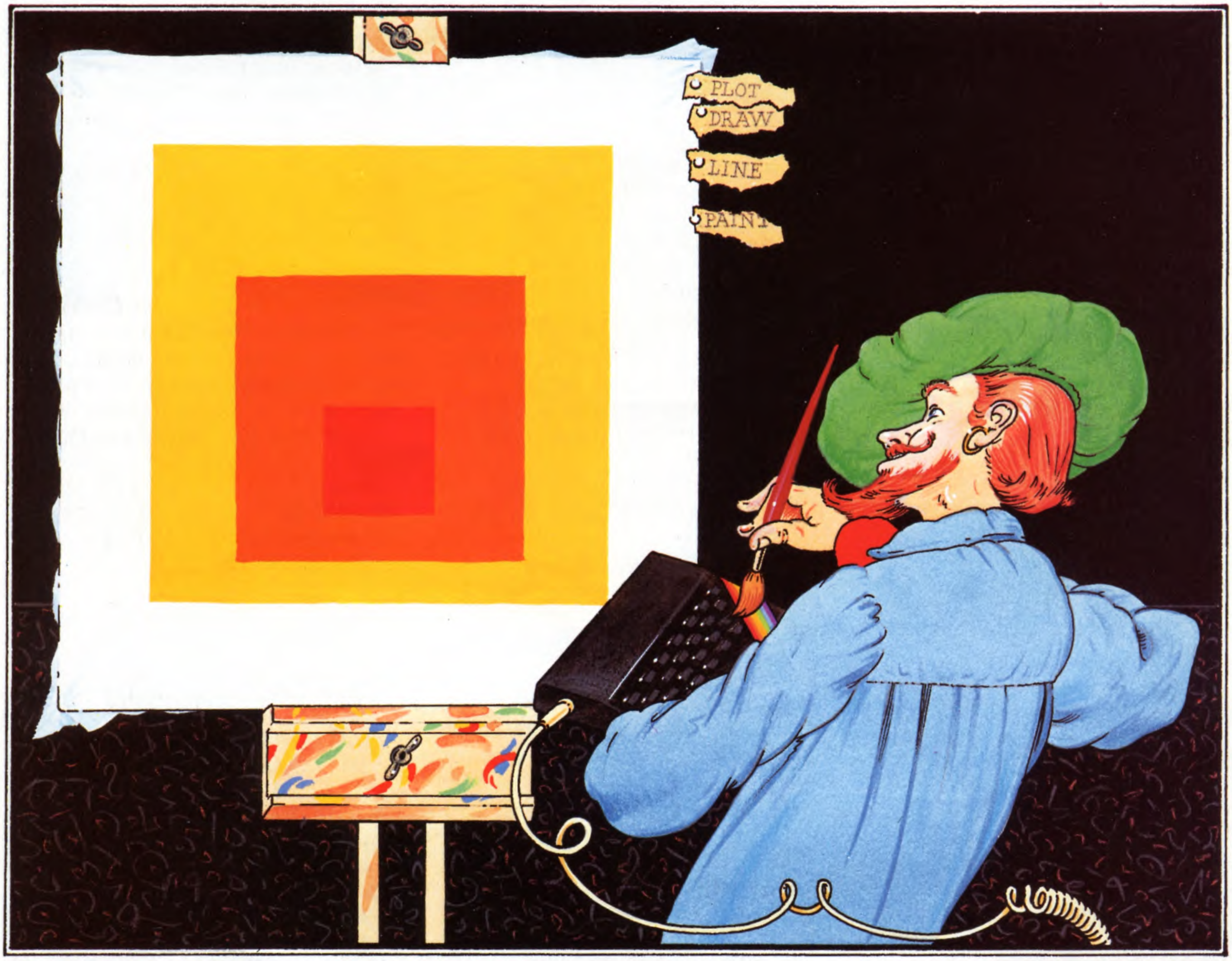
```
300 DATA 0,0,0,0,0,0,0,0,6,2,3,7,7,3,35,
    51,96,107,159,252,15,0,192,240
310 DATA 0,0,0,64,96,112,96,80,47,33,
    65,255,1,3,7,15,252,190,182,175,159,
    254,254,204
320 DATA 28,15,3,0,0,0,0,0,127,255,
    247,247,7,3,1,1,236,232,224,192, 128,
    0,32,224
330 DATA 8,68,50,139,50,68,8,0,16,34,
    76,145,76,34,16,0
```

Again, user defined graphics are a little different on the Commodore 64 as the graphics frames—or sprites, as they are known—already exist in the computer's memory. Consequently you do not need to RUN a frame-construction program before you enter the dragon-creation program below. Just key in and RUN it.

```
10 SC=53248:X=150:Y=157:ZZ=+45:
   POKE 650,255:PRINT "♥":POKE
   53281,0:POKE SC+39,5
20 FOR I=16000 TO 16254:READ A:
   POKE I,A:NEXT I
30 POKE 2040,250:POKE SC+29,255:
   POKE SC+23,255:POKE 2041,252
40 GET A$:POKE SC+21,253:IF A$=
   "□" THEN POKE SC+2,X+ZZ:POKE
   SC+3,Y:POKESC+21,255
50 IF A$="Z" AND X>60 THEN
   X=X-2:POKE 2040,251:POKE 2041,253:
   ZZ=-45
60 IF A$="X" AND X<200 THEN
   X=X+2:POKE 2040,250:POKE 2041,252:
   ZZ=+45
70 IF A$="P" AND Y>70 THEN Y=Y-2
80 IF A$="L" AND Y<200 THEN Y=Y+2
90 POKE SC,X:POKE SC+1,Y
100 IF A$="□" THEN POKE SC+2,
    X+ZZ:POKE SC+3,Y:POKE SC+21,255
110 IF A$="□" THEN POKESC+40,
    RND(1)*8
120 GOTO 40
130 DATA0,6,192,0,2,192,0,3,241,
    0,3,159,0,35,252,0,59,15,0,43,129,
    0,39,192
140 DATA1,227,224,1,1,240,60,241,
    120,36,31,124,60,31,126,6,63,142,3,
    127,254
150 DATA1,255,254,0,255,252,0,127,
    248,0,12,48,0,4,164,0,7,188,0
160 DATA3,96,0,3,64,0,143,192,0,
    249,192,0,63,196,0,240,220,0,129,
    212,0,3,228,0
170 DATA7,199,128,15,128,128,30,
    143,60,62,248,36,126,248,60,112,
    252,96,127
180 DATA254,192,127,255,128,63,255,
    0,31,254,0,12,48,0,37,32,0,61,224,
    0,0
190 DATA0,0,0,32,0,0,80,0,0,32,0,0,
    15,0,0,32,0,0,80,0,0,32,0,0,0,0,0,
    0,0,0,0,0
200 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
210 DATA0,0,0,0,0,4,0,0,10,0,0,4,0,0,240,
    0,0,4,0,0,10,0,0,4,0,0,0,0,0,0
220 DATA0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,0,0,0,0
```

HOW TO PLOT, DRAW, LINE AND PAINT

A few simple commands, plus your own imagination—that's all you need to produce computer pictures on your TV screen. Here we show you how to begin



Connect a TV set to your computer and the screen is at your disposal. You can draw on it and colour it as you wish and with a little time you can make quite complicated pictures.

Computer graphics has come a long way in the last few years and is now used for all sorts of things. Graphs and charts are an obvious example but you may also notice computer-generated pictures in TV adverts, magazines and even feature films. In fact many of the pictures in *Input* were produced or modified by a computer.

Of course, the computers used for pictures of this sort are huge, much larger than the humble micro you have at home. But there's an awful lot you *can* do on your home micro.

Most computers understand simple commands such as PLOT, DRAW, PAINT and INK. The TV screen is divided up into a number of individual picture elements called 'pixels' and each of these can be lit up and coloured as instructed by the different commands. On some computers you can choose between high resolution drawings with small pixels and fine lines, or low resolution pictures with larger pixels and coarse lines.

So drawing lines is just a matter of telling the computer which pixels to light up. And after a few minutes experimenting with the commands and drawing simple pictures you'll soon know your way around the screen.

The ZX81 has relatively limited graphics capabilities—so its programs explore the use of its single command, PLOT. And neither of the Commodore computers' standard BASIC lets you use the machines' high resolution graphic capability directly, but both can be fitted with an accessory cartridge to give them this facility.

- USING YOUR COMPUTER'S **BASIC** GRAPHICS COMMANDS
- DRAWING SQUARES, TRIANGLES AND CIRCLES
- COLOURING AND SHADING
- USING **FLASH** TO ENLIVEN SPECTRUM PROGRAMS
- SIMON'S BASIC AS AN AID TO COMMODORE GRAPHICS
- VARYING THE MODE ON ACORN

The image which the Spectrum produces on your TV screen is 256 pixels, or dots, in width and 176 pixels high.

They are numbered in what at first sight seems a rather odd way. If you are used to PRINT AT statements, you will know that

```
PRINT AT 1Ø, 15. . . .
```

. . . means that what follows will be PRINTed 1Ø lines from the top of the screen and 15 'columns' from the left.

When you PLOT a single pixel, however, the first number gives the distance from the *left*, and the second number gives the distance up from the *bottom* of the screen. So

```
PLOT 1Ø, 15
```

. . . puts a spot 1Ø pixels in from the left and 15 rows up. (Try it—you will just see the dot near the bottom left-hand corner.)

The horizontal positions are numbered from Ø to 255, and the vertical ones from Ø to 175. To get the 'feel' of these positions, try these lines one at a time:

```
PLOT Ø, Ø
PLOT Ø, 175
PLOT 255, Ø
PLOT 255, 175
```

The four dots you now see mark the limits of the area in which you can produce a picture. Everything outside them is just BORDER (see below). You can PLOT in any colour, as explained below.

DRAWING LINES

DRAW on the Spectrum does exactly what you would expect it to: it draws a line. But before it can do so, the computer has to know where to start drawing the line, how long it is to be, and in what direction you want it.

To start the line off, you can use a PLOT position. Try these lines in order, without clearing the screen between them:

```
PLOT 1ØØ, 75
DRAW 5Ø, Ø
```

If you do not give a new PLOT position, the computer will just carry on from where it is already—that is, the end of the line it has just drawn. To show this, add these lines one at a time to the two you have just drawn:

```
DRAW Ø, 5Ø
DRAW −5Ø, Ø
DRAW Ø, −5Ø
```

As you see, you use *positive* numbers to draw up or to the right. You use *negative* numbers to drawn down or to the left.

If you want an angled line, you just follow the same rule: first state how many places to the right (or left) it has to move, then how many up (or down). For example, add this extra line to those above:

```
DRAW 5Ø, 5Ø
```

BUILDING A PROGRAM

Here is a more interesting program using DRAW. So you can see what's happening, first enter and RUN these lines:

```
1Ø INK 2
2Ø PLOT Ø, 175
3Ø LET t=255: LET r=−175: LET b=−255:
   LET l=169
4Ø DRAW t, Ø: DRAW Ø, r: DRAW b, Ø:
   DRAW Ø, l
```

If you have entered this correctly (and not mistaken small L (l) for the number 1, which *cannot* be a variable!) you should have a box with a corner missing. Next, add these lines:

```
5Ø LET t=t−6
6Ø LET r=r+12: LET b=b+12: LET
   l=l−12
7Ø DRAW t, Ø: DRAW Ø, r: DRAW b, Ø:
   DRAW Ø, l
```

This—RUN it to check—will give you another box inside the first.

Now add the rest of the program:

```
8Ø LET t=t−6
9Ø GOTO 5Ø
```

When you RUN this you will probably be surprised by what has happened in the middle of the screen.

The reason is simple. Every time your program executes the loop between Lines 5Ø and 9Ø, it deducts 12 from each of the positive numbers. The variable t (for top), for instance, reduces from 255 to 243 . . . to 231 . . . to 219 and so on. Eventually it passes zero and becomes a negative number—so the line that it is controlling reverses direction. In the same way, your negative variables (such as b for bottom) eventually become positive and also change direction.

To watch the process more closely, reRUN the program with this addition:

```
85 PAUSE 1ØØ
```

LIGHT AND SHADE

The Spectrum has no COLOUR statement to give you a quick, easy means of shading or colouring in an area of background.

You can 'shade in' using rows of parallel lines. But as each new line must have its own PLOT statement, programs doing this can become cumbersome.

One way round this is to use a FOR . . . NEXT loop, as in this program, which draws the right-hand wall of the house in fig 2:

```
2Ø FOR h=56 TO 1ØØ STEP 3
3Ø PLOT 16Ø,h
4Ø DRAW 23,12
5Ø NEXT h
6Ø PLOT 183,11Ø: DRAW Ø,−97:
   DRAW −23,−13: DRAW Ø,1ØØ
```

How does it work? Each time the loop is executed, the program draws a line 23 pixels long, and 'sloping' by 12 pixels from left to right. Each time, it starts 16Ø pixels from the left.

But on each 'trip' round the loop, Line 2Ø adds 3 to the variable h, representing the height of the PLOT position. So the beginning of each line is 3 pixels higher than before.

Add the next few lines and your house will have most of its roof:

```
7Ø LET r=1ØØ
8Ø FOR p=14Ø TO 15Ø
9Ø PLOT r,p
1ØØ DRAW 69,−44
11Ø LET r=r+2
12Ø NEXT p
```

Here there are two variables, p and r. This is because the PLOT positions have to move not just upwards, but to the right as well.

Note, incidentally, the use of negative numbers in Line 1ØØ to make the roof slope downwards. The next few lines complete the outline of the house:

```
13Ø PLOT 1ØØ,14Ø
14Ø DRAW −6Ø,−4Ø
15Ø PLOT 46,1Ø3
16Ø DRAW Ø,−96
17Ø DRAW 113,−7
```

Now you can complete the house with a door, windows and chimney of your own design, using PLOT and DRAW statements. If you decide to copy it as it stands, here are some hints:

- The triangle of 'cladding' below the roof requires *three* variables. One controls the height at which each line starts. One controls how far to the right each line starts. And one controls the length of the line.
- The upstairs windows are 'knocked through' the cladding after it is installed. To do this, you will need to PRINT blocks of spaces, □, AT suitable points, using the method given in Games Programming 1. Then you 'line' the openings, using PLOT and DRAW in the usual way.

DRAWING CIRCLES

The Spectrum's CIRCLE statement also behaves quite predictably: it draws a circle.

All you have to remember about it is that the first two numbers after a CIRCLE statement locate the centre of the circle—which is not printed—while the third one gives its radius in pixels. Thus the line:

```
CIRCLE 127, 87, 5Ø
```

. . . draws a circle 1ØØ pixels in diameter (5Ø radius) in the middle of the screen.

In the house program above, for instance, you could use CIRCLE to 'install' a round window in the front door.

If you follow a CIRCLE with a DRAW statement, the line you draw will start from the point where the circle finished.

SPECTRUM COLOURS

The Spectrum has a range of eight colours, labelled immediately above the keys—from Ø to 7—which you use to call them up.

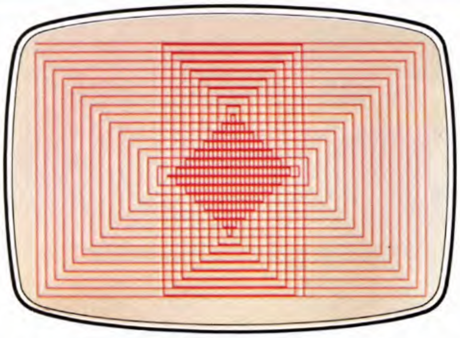
BORDER controls the area around the outside, in which you cannot DRAW or PRINT.

The other colour commands can be used in two quite different ways. If you RUN a program with, for example

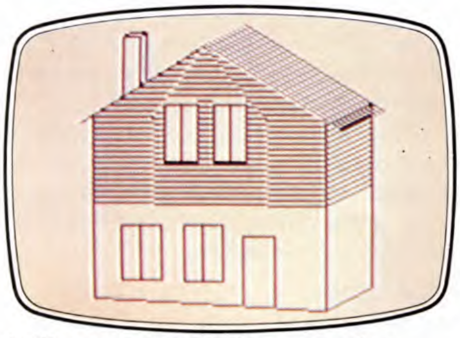
```
1Ø BORDER 2: PAPER 2: INK 7
```

everything that follows will be PRINTed white on red screen until you replace this line with another one which changes the colour commands. That includes your program listings, which can become very hard to read.

If, on the other hand, you begin with



1. An easy Spectrum pattern.



2. You can also mimic perspective

```
1Ø PRINT PAPER 2; INK 7; AT 15,1Ø; "*"
```

. . . then only the small area of screen where the character itself appears will be red.

Notice that in the first example each colour statement is followed by a colon, whereas in the second one each is separated from the next by a semi-colon.

You can incorporate these colour statements in any program lines using PLOT, DRAW or CIRCLE. Try out these lines, for example, clearing the screen (CLS) between each:

```
PLOT PAPER 1; INK 7; 125, 87
```

(Can you see the dot?)

```
PLOT 125, 87: DRAW INK 2; 5Ø, 5Ø
PLOT 125, 87: DRAW PAPER 1; INK 7;
  5Ø,5Ø
CIRCLE INK 4; 127, 87, 5Ø
```

. . . and even this:

```
CIRCLE PAPER 2; INK 6; 125, 87, 5Ø:
   DRAW PAPER 3; INK 7; 75,Ø
```

This last combination can be used to produce some fairly spectacular (if useless?) effects.

Equally spectacular, and much used in games programming, is the last of the Spectrum statements dealt with in this article: FLASH. What it does is to reverse the INK and PAPER colours rapidly.

FLASH must be followed by a number—either 1 to switch it on, or Ø to turn it off. Here is an example of its use:

```
1Ø PAUSE Ø
2Ø IF INKEY$="" THEN GOTO 3Ø
3Ø FOR n=1Ø TO 23: BEEP .Ø15,n:
   NEXT n
4Ø BORDER 2: PAPER 2: INK 6:
   FLASH 1
5Ø PRINT "DO NOT TOUCH!"
6Ø PRINT "THIS COMPUTER IS BOOBY
   TRAPPED"
7Ø PRINT
8Ø GOTO 3Ø
```

When you have entered this program, RUN it. Then touch any key—and stand back!

The ZX81 uses a PLOT command in a similar way to the Spectrum: PLOT, followed by two numbers which control the position of the pixel. But there is one difference, in that the ZX81's pixels are sixteen times the size of the Spectrum's—so there are only four pixels in each character space, while the Spectrum has 64 pixels in a character space.

As a result, you cannot draw such detailed pictures or patterns—but at least it makes each pixel easier to see! This program PLOTs a pattern on the screen. Since it restarts every time it gets to Line 7Ø, you will have to press the BREAK key when you want to stop:

```
1Ø LET X=INT (RND*32)
2Ø LET Y=INT (RND*24)
3Ø PLOT X,Y
4Ø PLOT 63−X,Y
5Ø PLOT X,43−Y
6Ø PLOT 63−X, 43−Y
7Ø GOTO 1Ø
```

The ZX81 has no DRAW command, but you can use the PLOT command to PLOT a series of dots which make up a line. You can see how this works with the following program, which 'draws' a number of squares using two FOR . . . NEXT loops:

```
1Ø FOR N=Ø TO 2Ø STEP 2
2Ø FOR M=N TO 43−N
3Ø PLOT M,N
4Ø PLOT M, 43−N
5Ø PLOT N,M
6Ø PLOT 43−N,M
7Ø NEXT M
8Ø NEXT N
```

And in a similar way you can mimic the CIRCLE command using the following short routine:

```
1Ø FOR N=Ø TO 2*PI STEP .1
2Ø PLOT 32+2Ø*SIN N, 22+2Ø*COS N
3Ø NEXT N
```

This uses the two maths functions SIN and COS to work out the coordinates of each pixel that makes up the circle. A later article will explain these in detail, but for the moment you can experiment with the size of the circle by changing the two number 2Øs in Line 2Ø. The maximum number you can have is 22—larger will not fit on the screen. Or change the position of the circle. At the moment its centre is at 32,22—you can see these coordinates in Line 2Ø. Don't forget to make the circle smaller if you move it near the edge of the screen.

Although the Commodore 64 has diverse and potentially very sophisticated graphics facilities, there's no provision for accessing the more spectacular of these directly from BASIC.

In fact, you can only access the low resolution block graphics, letters and symbols which form part of the ROM character sets.

You may find it easier to begin with a Commodore 64 accessory, the Simons' BASIC cartridge. This is a ROM device which clips into the rear of the machine. It provides many of the programming facilities and keywords necessary to use the sound and graphics of the Commodore.

Eighteen specific keywords (or 'commands') are provided for the graphics. All but one of these—COLOUR—have to be used as part of a program. This means you can't 'draw' pictures on the screen in direct mode. And the cartridge has to be fitted if the programs written with it are to RUN at all.

BEGINNING WITH SIMONS' BASIC

Powering up with the cartridge in place gives a white screen, blue border and black upper case (capitals) text—and confirmation that about 8K of memory has been used up or allocated to the new BASIC.

You can try out the only direct command straight away:

```
COLOUR 5,5
```

The code numbers after the keyword set the screen background colour and then the screen border colour. Value 5, in each case, is the colour value for green (check your manual for others)—black lettering is maintained.

HIRES is the keyword to use to initialize high resolution graphics. The code following sets the plotting colour (again using the standard colour values listed in your manual), and then the background colour. (Note the space which *must* follow this instruction.)

```
1Ø HIRES□ 1,Ø
```

This prepares the computer for hi-res plotting of white on black—the screen will immediately turn black in response.

The first of the 'drawing' commands is REC, whose purpose is to draw rectangles. The code following this keyword starts with the X,Y 'start' position.

In the X,Y notation, location Ø,Ø represents the top left corner of the screen. The X value increases as you go to the right horizontally; the Y value increases as you go downwards.

The grid size varies according to the graphic mode in use. In hi-res mode, the resolution is normally 32Ø pixels horizontally, 2ØØ vertically. The horizontal resolution is halved in multicolour mode, to 16Ø by 2ØØ pixels.

The REC keyword—which must be preceded by the HIRES instruction—requires further coding:

```
1Ø HIRES□ 1,Ø
2Ø REC 1Ø,5Ø,2ØØ,5Ø,1
3Ø GOTO 1Ø
```

In this example, the first pair of numbers after the REC keyword are the X,Y coordinates, which place the corner of the rectangle at the pixel 1Ø along from the left and 5Ø down. The next value (2ØØ) is the horizontal length of the rectangle, and this is followed by the vertical (5Ø) drop, all measurements being in pixels.

The final value in Line 2Ø is the 'plot type' value, used in Simons' BASIC to indicate the function to be performed. A Ø here clears a dot, 1 plots a dot, and 2 inverses a dot (so turning it 'on' if it is 'off', or 'off' if it is 'on').

MULTI is used in support of the HIRES keyword to initialize the multicolour graphics mode and allows you to choose three plotting colours. These are predefined within the MULTI instruction and selected by changing the plot type value in other instructions. So:

```
1Ø HIRES□ 1,Ø
2Ø MULTI 7,8,14
3Ø REC Ø,75,1ØØ,5Ø,1
4Ø REC 7Ø,5Ø,5Ø,5Ø,2
5Ø REC 1ØØ,25,5Ø,1ØØ,3
1ØØ GOTO 1Ø
```

Note how the last item in each REC instruction—the plot type value—selects, by number, a value in the MULTI instruction: this is how a predefined colour is 'called up'. Try changing around the last values in Lines 3Ø, 4Ø and 5Ø. Remember that the first MULTI colour is chosen if the plot type value is 1, the second if it is 2, the third if it is 3. A plot value of 4, in MULTI mode, inverses the plot.

You can change the colours even after they have been defined by MULTI. LOW COL does this by specifying three new colours as:

```
99 LOW COL 3,5,6
```

These values are accessed by subsequent commands in the same way as the REC program lines call on the MULTI instruction.

The program can be made to revert to the original plotting colours by using the HI COL instruction, which requires no further coding. Thus it is possible to switch to a new set of colours and back again many times within a program should you choose to make use of this instruction.

DRAWING

There are five 'drawing' keywords—PLOT, LINE, CIRCLE, ARC, ANGL and DRAW—and all may be used in combination to produce screen graphics. In a moment you'll see how to make a picture of a flying bird. All except DRAW are followed by coding to indicate the relevant X,Y values required. Let's look at PLOT, LINE and CIRCLE first.

PLOT is used to plot individual dots where required: (NEW your computer)

```
1Ø HIRES□ 1,Ø
2Ø FOR X=Ø TO 1ØØ : FOR Y=Ø TO 1ØØ
3Ø PLOT X,Y,1
4Ø FOR D=1 TO 1Ø: NEXT D
5Ø NEXT Y,X
6Ø GOTO 1Ø
```

This plots a line from the top left of the screen, with the X,Y values dictated by the FOR . . . NEXT loop starting in Line 2Ø. The last value in Line 3Ø is the plot type value.

Simons' BASIC has a useful keyword which provides an alternative to the now familiar FOR . . . NEXT delay loop used in Line 4Ø. This is the keyword PAUSE which creates delays in multiples of one second.

```
4Ø PAUSE 1
```

This gives a one second delay before each of the dots is plotted. Try inserting other values after the keyword.

PAUSE may also be used to 'hold' the program at its completion, so:

```
6Ø PAUSE 15
```

will hold the program for 15 seconds before the program ends and the 'ready' prompt shows on the screen.

LINE is probably the most useful of the drawing commands. It is used to draw a line between two points. All you have to do is specify the X,Y start and finish locations. Press NEW on your computer and follow the line numbering suggested.

```
1Ø COLOUR 3,Ø: HIRES 1,Ø: MULTI 8,3,6
4Ø LINE 55,1ØØ,4Ø,8Ø,3
16Ø PAUSE 2Ø
```

This draws a line in the third colour specified by the MULTI keyword, and will form part of a graphic when further lines have been added.

The CIRCLE command is followed by X,Y coding which locates its centre, then the horizontal radius, then the vertical radius, and finally the plot type value:

```
20 CIRCLE 65,100,10,16,2
```

The circle is plotted in the second colour specified by MULTI. Note that the radius values differ: this counteracts the effect of a rectangular screen on 'square' plotting values. In HIRES mode, the Y-radius (the second to last value in the instruction sequence after the CIRCLE command) has to be 1.4 times the X-radius, the value preceding it in the line. For MULTI mode, the Y-radius has to be 1.6 times the X-radius if a true circle is required. By adjusting the relative X and Y values circles and ellipses of various types and sizes can be programmed.

PAINTING

PAINT is a useful command which enables a shape to be filled with a colour specified in a MULTI statement:

```
30 PAINT 65,110,3
```

This colours the circle in cyan, as specified by Line 1Ø keyed in before. The area to be coloured—or PAINTed—must be completely enclosed for the command to be executed, and the X,Y values which follow the keyword must be any point *within* a previously defined shape. Try playing around with the values in Line 4Ø: you can change colour simply by adjusting the plot type value, but note the effect of making deliberate mistakes on the X,Y values that precede this.

To get the program you've keyed in to produce a picture, type in the following extra lines:

```
50 LINE 40,80,0,120,3
60 LINE 0,120,40,100,3
70 LINE 40,100,55,105,3: PAINT
   50,100,2
80 LINE 70,100,110,70,3
90 LINE 110,70,150,90,3
100 LINE 150,90,110,90,3
110 LINE 110,90,75,100,3: PAINT
   90,95,2
120 LINE 60,110,50,130,3
130 LINE 70,105,80,130,3
140 LINE 50,130,80,130,3: PAINT
   60,120,2
150 FOR Z=63TO67:LINE Z,110,65,
   140,1:NEXT:PAUSE 10
```

Notice, in particular, how the plot type value in the MULTI statement is accessed many times during the program. RUNning the program displays a flying bird.

The program first sets the screen colour, hi-res mode, and the colour of the plot in Line 1Ø. Line 2Ø draws the circle of the bird's head, subsequently coloured by Line 3Ø. The draw and colour routine for the bird's left wing takes place in Lines 4Ø to 7Ø, that for the right wing in Lines 8Ø to 11Ø. The tail is drawn and coloured by Lines 12Ø to 14Ø, and the beak by an interesting use of FOR ... NEXT in Line 15Ø. Finally, there is a PAUSE before the screen returns to normal.

As with the Commodore 64, the Vic's standard BASIC has no provision for direct access of the machine's more sophisticated graphics. Once again, the best way to begin is with a Commodore accessory, the Vic Super Expander cartridge. This clips onto the rear of the machine to give it the extended BASIC you need. With this in place, power up, and try this:

```
10 GRAPHIC 2: SCNCLR
20 FOR X=0 TO 100 STEP6:FOR Y=0 TO
   100 STEP6
30 POINT 1,X,Y
40 FOR D=1 TO 10:NEXT D
50 NEXT Y,X
60 GOTO 10
```

This is the Vic version of the Commodore 64 program after the heading DRAWING on the previous page—and the explanation is the same. You can also produce a Vic version of the 64's bird, with the following program:

```
10 GRAPHIC 2:COLOR 0,0,3,6
20 CIRCLE1,460,400,40,64
25 CIRCLE1,460,300,30,34
30 PAINT1,460,440
40 DRAW1,420,400TO360,320
50 DRAW1,360,320TO200,480
60 DRAW1,200,480TO360,400
70 DRAW1,360,400TO420,420:
   PAINT1,400,400
80 DRAW1,480,400TO640,280
90 DRAW1,640,280TO800,360
100 DRAW1,800,360TO640,360
110 DRAW1,640,360TO500,400:
   PAINT1,560,350:COLOR0,0,5,5
120 DRAW1,440,440TO400,520
130 DRAW1,480,420TO560,520
140 DRAW1,400,520TO560,520:
   PAINT1,440,480
150 DRAW1,480,300TO520,300
160 POKE 36865,39+RND(1)*2:
   GOTO 160
```

Once again, the explanation of this is similar to the 64's, although the BASIC keywords are different.

Acorn computers always start off in a text mode when you first turn them on, so the first thing to do before you can start drawing is to turn to one of the graphics modes. Five graphics modes are available—modes Ø, 1, 2, 4 and 5, and each has a different set of colours and a different resolution.

For the moment just use Mode 1. So type MODE 1 and press RETURN. Now you can start.

All drawing is done with the graphics cursor. You can move it about using MOVE X,Y, draw lines using DRAW X,Y and do all sorts of other things using PLOT.

Don't worry that you can't see the graphics cursor—it happens to be invisible!

Type DRAW 64Ø,512 and press RETURN. You should see a line going from the bottom left to the centre of the screen.

Now enter DRAW 1279,Ø and another line is drawn from the last point to the bottom right. You just need one more command to complete the triangle—DRAW Ø,Ø.

Well, that was fairly easy, but you may have been confused by the numbers. These are the X and Y coordinates of the point you wanted to go to. The screen is divided up into lots of tiny squares. There are 128Ø squares across the screen along the X-axis numbered from Ø to 1279, and 1Ø24 squares up the screen along the Y-axis numbered from Ø to 1Ø23. That is why the centre of the screen has the coordinates 64Ø,512.

DRAWING A PICTURE

When you are planning out a drawing it is best to draw it out first on a sheet of graph paper so you can work out the coordinates of each point. In fact there is a handy graphics planning sheet in the back of the User Guide that you can use.

When you were drawing the triangle you may have noticed that the first line started at the origin (position Ø,Ø on the screen), and the other lines started at the last point visited. What if you wanted your line to start at some other point? Here you must use the MOVE X,Y command. This simply takes the cursor to the point X,Y without drawing a line. Then when you use DRAW, the line starts from this point.

Try typing in the following program:

```
1Ø MODE 1
2Ø MOVE 3ØØ,2ØØ
3Ø DRAW 9ØØ,2ØØ
```

Type RUN and you'll see a line drawn near the bottom of the screen. This is the base line of a house, and these next lines complete the drawing:

```
4Ø DRAW 9ØØ,6ØØ
5Ø DRAW 3ØØ,6ØØ
6Ø DRAW 3ØØ,2ØØ
7Ø MOVE 3ØØ,6ØØ
8Ø DRAW 6ØØ,9ØØ
9Ø DRAW 9ØØ,6ØØ
```

Try typing RUN after you enter each line of the program to see how the drawing is built up. Then try adding a few more lines to give the house a door and a couple of windows.

USING OTHER MODES

So far you have been working entirely in Mode 1. Try changing Line 1Ø of the last program to MODE 5. You'll find that the pixels are larger and the lines of the drawing are rather coarse—especially the sloping lines that make up the roof. In other words, Mode 5 has a lower resolution than Mode 1. Try it again in Mode Ø. This has the smallest pixels and gives the finest lines (or the highest resolution).

Mode 2 has the same resolution as Mode 5, and Modes 1 and 4 have the same resolution.

The number of colours available in each mode also varies. Modes 1 and 5 have four colours, Modes Ø and 4 have two colours and Mode 2 has the full range of 16 colours.

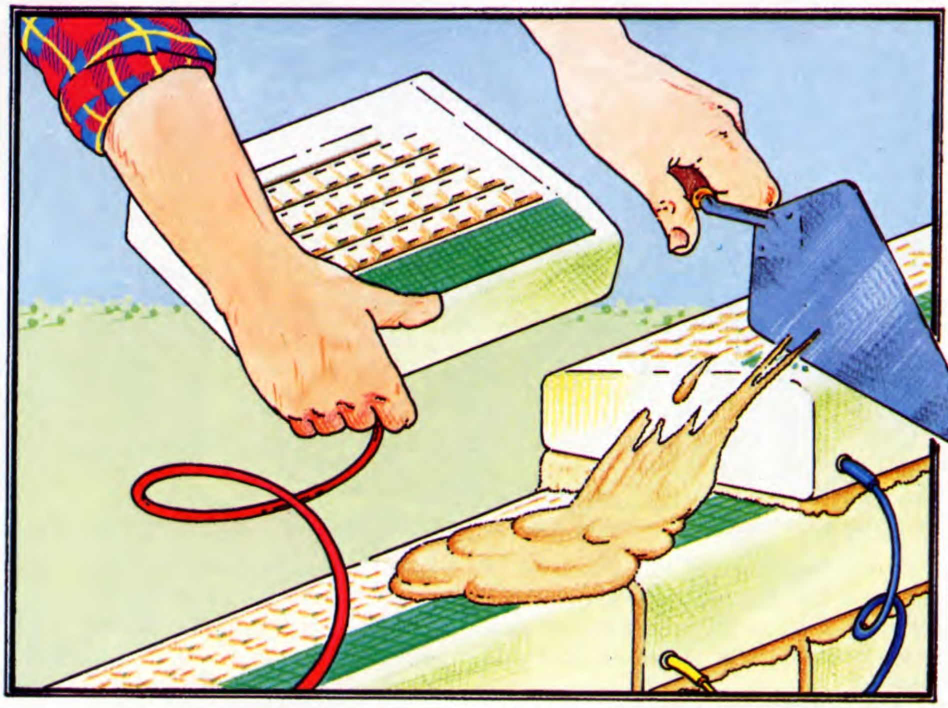
DRAWING IN COLOUR

If you don't specify what colour you want, the lines are always drawn in white on a black background. These are known as the *default* colours.

To choose another colour you must use the command GCOL Ø. This stands for Graphics COLour; (don't worry about the Ø).

Change Line 1Ø back to MODE 1. Mode 1 has four colours—black, red, yellow and white—and these are numbered from Ø to 3. If you type GCOL Ø,2 and then draw a line it will be drawn in yellow. GCOL Ø,1 will draw a line in red. Try adding these two commands to the last program so that the walls of the house are drawn in yellow and the roof is drawn in red. The first one will have to go at Line 15, and the second one at Line 75.

You can change the colour of the background as well as the foreground. Use the same command—GCOL Ø—but this time add 128 to the colour code number. For example, GCOL Ø,129 followed by CLG to CLear the Graphics screen will give a red background.



Type in and RUN this next program. It draws random-sized rectangles on the screen in red, yellow and white on a black background.

```
1Ø MODE 1
15 FOR boxes=1 TO 3Ø
2Ø X=RND(1ØØØ) : Y=RND(8ØØ)
3Ø base=RND(4)*1ØØ : side=RND(4)*1ØØ
4Ø GCOL Ø,RND(3)
5Ø MOVE X,Y
6Ø DRAW X+base,Y
7Ø DRAW X+base,Y+side
8Ø DRAW X,Y+side
9Ø DRAW X,Y
1ØØ NEXT
```

Line 15 starts the loop to draw 3Ø rectangles, and Line 2Ø selects the coordinates for the bottom left corner. Line 3Ø chooses the length of the base and side of each box. Line 4Ø selects the colours, then the next few lines draw them out.

Perhaps you would prefer a different colour scheme, say black and white boxes on a red background? You just need to make two changes. Add one line to change the background to red:

```
13 GCOL Ø,129: CLG
```

Then alter Line 4Ø to:

```
4Ø GCOL Ø,(RND(2)-1)*3
```

This makes sure that the foreground colour codes are either Ø for black or 3 for white.

USING THE PLOT COMMAND

So far you can move the graphics cursor about the screen and draw lines, and you can also use colour. But there are a lot more commands which use the keyword PLOT. For example, PLOT 69,X,Y plots a dot at position X,Y. Try adding a doorknob to the house!

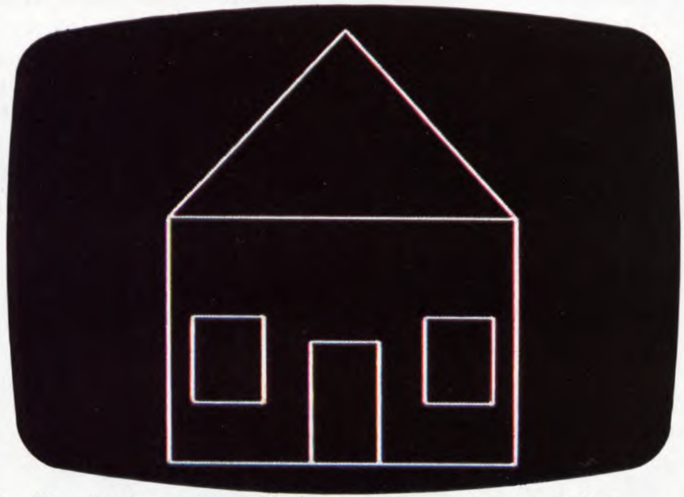
You can also draw dotted lines with PLOT 21,X,Y as this next program shows:

```
1Ø MODE 1
2Ø GCOL Ø,13Ø : CLG
3Ø PRINT TAB(Ø,3)"Please sign on "'
   "the dotted line"
35 GCOL Ø,Ø
4Ø MOVE 5ØØ,19Ø
5Ø PLOT 21,11ØØ,19Ø
6Ø INPUT TAB(17,25),signature
```

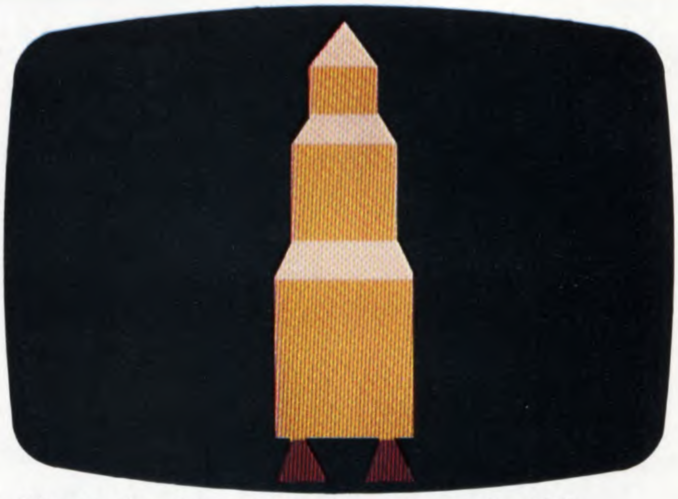
As well as demonstrating the PLOT 21 command it also shows how you can mix text and graphics on the same screen. Note that the text is still positioned using TAB so you will need to use the text and graphics planning sheets in the back of the User Guide to make sure the text matches up with the graphics.

One of the most useful PLOT commands is PLOT 85 which plots and fills a triangle. It is written out as PLOT 85,X,Y and this draws a solid triangle between the point, X,Y and the last two points visited. Try this:

```
1Ø MODE 1
2Ø GCOL Ø,2
3Ø MOVE 2ØØ,2ØØ
4Ø MOVE 1Ø8Ø,2ØØ
5Ø PLOT 85,64Ø,9ØØ
```



3. **Simple shapes like this house will get you started**



4. **Fifteen triangles make up this rocket on the BBC**

Note that you don't actually have to DRAW anything. You just have to visit two of the points, in this case at Lines 3Ø and 4Ø, then use PLOT 85 to the third point to make a solid triangle.

With a bit of thought almost any shape can be divided into a number of triangles and plotted as a solid shape right from the start. For example, a rectangle can also be made up out of two triangles, and the program to draw this is:

```
1Ø MODE 1
2Ø GCOL Ø,2
3Ø MOVE 3ØØ,3ØØ
4Ø MOVE 3ØØ,7ØØ
5Ø PLOT 85,1ØØØ,3ØØ
6Ø PLOT 85,1ØØØ,7ØØ
```

Finally, here is a program that makes good use of the triangular fill facility. It draws a large, three-stage rocket in three colours in the centre of the screen (fig 4). It is actually built up out of 15 triangles.

```
1Ø MODE 1
2Ø MOVE 6ØØ,1ØØØ
3Ø MOVE 52Ø,9ØØ
4Ø PLOT 85,68Ø,9ØØ
5Ø GCOL Ø,2
6Ø PLOT 85,52Ø,8ØØ
7Ø PLOT 85,68Ø,8ØØ
8Ø GCOL Ø,3
9Ø PLOT 85,48Ø,74Ø
1ØØ PLOT 85,72Ø,74Ø
11Ø GCOL Ø,2
12Ø PLOT 85,48Ø,54Ø
13Ø PLOT 85,72Ø,54Ø
14Ø GCOL Ø,3
15Ø PLOT 85,44Ø,46Ø
16Ø PLOT 85,76Ø,46Ø
17Ø GCOL Ø,2
18Ø PLOT 85,44Ø,12Ø
19Ø PLOT 85,76Ø,12Ø
2ØØ GCOL Ø,1
21Ø MOVE 72Ø,12Ø
22Ø MOVE 68Ø,12Ø
23Ø PLOT 85,76Ø,2Ø
24Ø PLOT 85,64Ø,2Ø
25Ø MOVE 52Ø,12Ø
26Ø MOVE 48Ø,12Ø
27Ø PLOT 85,56Ø,2Ø
28Ø PLOT 85,44Ø,2Ø
```

Now try drawing your own pictures. If you want more colours you can try Mode 2—this gives you the whole range of 16 colours.

When you first switch on the computer it displays its text screen. On this you can display all the characters on the keyboard, plus the ROM graphics you have used in previous programs.

If you want to draw anything more subtle than these chunky graphics allow, you must use the machine's high resolution graphics. These are displayed on a completely different screen from the text screen—it isn't possible to write text on the high resolution screen, nor draw high resolution graphics on the text screen.

DRAWING LINES

Here is a program which draws two lines across the screen using graphics mode 3:

```
2Ø PMODE 3,1
3Ø PCLS
4Ø SCREEN 1,Ø
5Ø LINE (Ø,191) – (255,Ø),PSET
6Ø LINE (Ø,Ø) – (255,191),PSET
7Ø GOTO 7Ø
```

The program draws lines by first specifying the mode and page (part of the computer's memory) you wish to put the graphics information in. Line 2Ø says PMODE 3,1. This tells the machine to draw in graphics mode three and to store whatever you draw on the screen starting on page 1 of graphics memory. If you want to draw just one screenful of graphics it's easiest to specify page 1.

Next, Line 3Ø clears the high resolution screen. PCLS is the high resolution equivalent of CLS.

To switch on the high resolution screen you must use the SCREEN command. So Line 4Ø says SCREEN 1,Ø. The colour set chosen by putting Ø as the second number consists of green, yellow, blue and red—if you'd put in 1 instead, the colour set would have been buff, cyan, magenta and orange.

This program draws lines in green and red, which are known as its *default* colours because they appear if you specify no others. To use any other combination of colours from the colour set you would have to use the COLOR command (note the spelling); this will be covered in detail later.

The first line is drawn by Line 5Ø. The numbers in the brackets tell the computer where you want the line to start and finish. The high resolution screen is divided into 256 pixels by 192—along the width of the screen they are numbered from Ø to 255, and along the height of the screen they are numbered from Ø to 191, starting from the top.

Finally, PSET tells the machine to draw in the highest numbered colour in the colour set you are using—in this case red.

Line 6Ø draws the other line. It works in exactly the same way as Line 5Ø, but the start and end points have been redefined.

To prevent the machine automatically switching back to the text screen once it has drawn on the high resolution screen you have to set up a loop so that it never reaches the end of the program. So Line 7Ø says GOTO 7Ø. If the computer *does* switch back to the text screen you'll not be able to see what has been drawn on the high resolution screen.

DRAWING CIRCLES

It's very easy to draw circles with the Dragon or Tandy because their BASIC has a CIRCLE command.

This program will draw three different sized circles, each in a different colour.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 CIRCLE (70,95),10,2
60 CIRCLE (118,95),20,3
70 CIRCLE (184,95),30,4
80 GOTO 80
```

The mode is chosen as before, as is the colour set for the circles.

The CIRCLE command in Lines 5Ø to 7Ø has four elements. The first and second numbers plot the centre of the circle, the third one is the radius in pixels, and the fourth is the colour. So the circle drawn by Line 5Ø has its centre at (7Ø,95), a radius of 1Ø pixels, and is drawn in yellow. The circle drawn by Line 6Ø is a blue one, of radius 2Ø, and with its centre at (118,95). The last circle, drawn by Line 7Ø, is a red one, of radius 3Ø, with its centre at (184,95).

Finally, Line 8Ø is a loop which prevents the program stopping and switching off the high resolution screen.

DRAWING A JEEP

You've now got the ingredients for drawing many different pictures. Type in and RUN this program and you'll see the outline of a Jeep without the details shown on the right.

```
20 PMODE 3,1
30 PCLS
40 SCREEN 1,0
50 LINE (108,64)—(188,64),PSET
60 LINE —(188,116),PSET
70 LINE —(104,116),PSET
80 LINE —(96,96),PSET
90 LINE —(70,96),PSET
110 LINE —(74,96),PSET
120 LINE —(74,86),PSET
130 LINE —(114,86),PSET
140 LINE —(132,104),PSET
150 LINE —(140,104),PSET
160 LINE —(140,64),PSET
170 LINE(76,96)—(76,108),PSET
180 LINE —(100,108),PSET
320 GOTO 320
```

As before, the mode chosen is 3. Line 5Ø starts drawing at (108,64), and draws a red line to (188,64). In order to continue drawing from the same point, you don't have to tell the computer where to start drawing from—it just carries on from where it is now. Lines 6Ø to 16Ø, therefore, only define the ends of successive lines.

Line 17Ø, though, draws a line from a new starting point, so both the start and end have to be defined.

Now add a coach line, windscreen, tyres and hubcaps:

```
200 LINE(140,88)—(188,88),PSET
220 LINE(118,64)—(104,86),PSET
230 CIRCLE(82,116),14,3
250 CIRCLE(82,116),6,2
270 CIRCLE(168,116),14,3
290 CIRCLE(168,116),6,2
```

Line 2ØØ draws the coach line, while the windscreen is drawn by Line 22Ø. CIRCLE is used by Lines 23Ø and 27Ø to draw the outline of the tyres, and by Lines 25Ø and 29Ø to draw the hubcaps.

PAINTING AN AREA

It's easy to spray your Jeep—you don't even need a spraygun because the machine does it for you with its PAINT command.

The PAINT command fills any shape you've drawn on the screen with a specified colour. It too has four components. The first two, as you'll have guessed, mark the spot where painting has to start. The third is the paint code. And the fourth is the colour code of the border, or line, where the painting has to stop.

Add these lines to the program and you'll see how the command looks in practice:

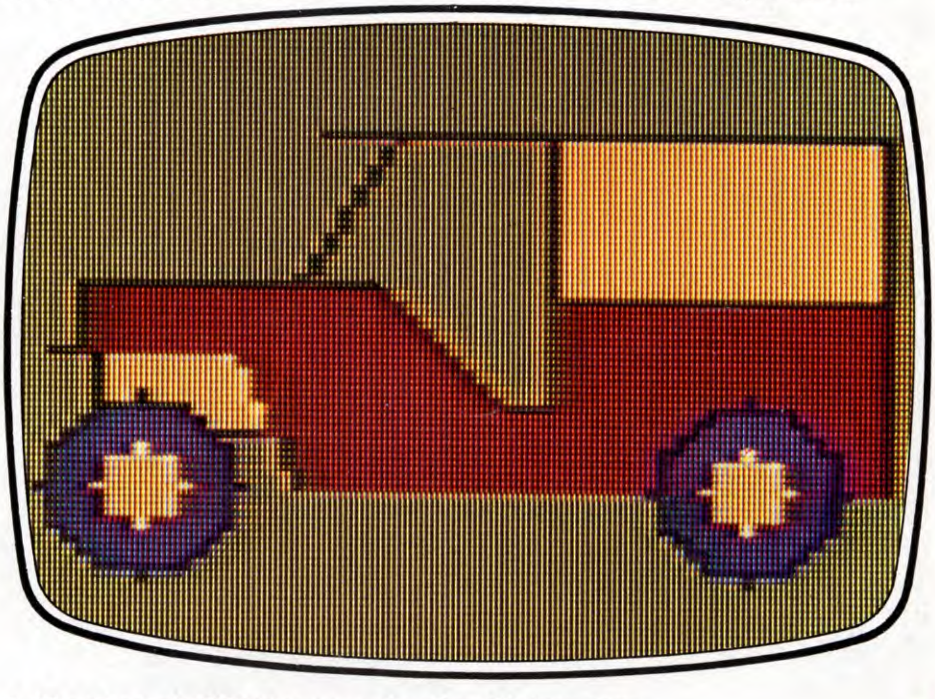
```
190 PAINT(90,100),2,4
210 PAINT(120,110),4,4
240 PAINT(82,116),3,3
260 PAINT(82,116),2,2
280 PAINT(168,116),3,3
300 PAINT(168,116),2,2
310 PAINT(180,80),2,4
```

RUN the program and you'll see each section of the Jeep being filled with colour.

It is not possible to have all the painting lines at the end of the program for one important reason—the area that you want to paint must be fully enclosed by the same colour (the border colour), or the edge of the screen. Any lines of a different colour will be painted over.

The colours available in PMODE 3,1 are green (1), yellow (2), blue (3) and red (4). You may like to try painting the Jeep differently. All you have to do is change the paint colour number in the appropriate PAINT lines.

You may like to try adding a few more parts to the Jeep using the LINE and CIRCLE commands. For example, it probably needs a steering wheel—see what you can do.



5. The Dragon's PAINT command makes this Jeep easy

VARIABLES: THE MYSTERIES EXPLAINED

All those baffling Xs and Ys in programs are really quite simple when you understand what they're doing. Here's how you can use variables in your own programs

When you want to store some information in your computer, you have to tell the computer how to identify the information. Otherwise the computer will not know how to perform this simple task, nor will it be able to find the information again when you want it.

One of the pieces of information that you will want to store most often is a number which increases and decreases as the program progresses. This number might represent, say, your score in a game. Or it might represent the number of places that a character has moved across, or up or down, the screen.

Quite often it will simply be a number whose value changes each time the computer 'loops' through a section of program. Here is one simple example:

```
1Ø LET X=Ø
2Ø LET X=X+1
3Ø PRINT X; "□";
4Ø FOR T=Ø TO 1Ø: NEXT T
5Ø GOTO 2Ø
```

Here it is Line 5Ø which is creating the loop, making the program repeat itself indefinitely—or until you pull out the plug. Each time it reaches Line 2Ø, the number represented by X (which was originally set at zero by Line 1Ø) is increased by one.

The X in this program is called a *numeric variable*, because it controls a number.

One way of representing how a numeric variable works is to imagine a series of boxes or compartments, like those used for sorting letters. Each box is given a name. In a simple computer, the names would be the letters of the alphabet. When you wanted to store a number you would simply assign that value to one of the boxes. For example:

```
LET C=25
```

This stores the value 25 in the variable C. Not all computers require you to use the word LET (which is known as an *assignment statement*), but until you are experienced in computing it is a good idea to include it. It simply makes your life a little easier when reading through programs.

Once stored, this value can be used within other statements and calculations. For example you could type:

```
PRINT C*4
```

or

```
PRINT C/5
```

These statements would give the answers 1ØØ and 5 respectively. But if you type in:

```
PRINT C
```

you will see that the value of C has remained unchanged. To change the value of C, to increase it to 3Ø for example, either do it directly by typing

```
LET C=3Ø
```

or by typing

```
LET C=C+5
```

This is, of course, precisely what your original mini-program did.

NAMES OF VARIABLES

The combinations of letters and so on that you are allowed to use as numeric variables varies from computer to computer. The main rules are given in the box (page 95).

But one thing no computer will allow is a variable beginning with a number. This is because it knows that

```
LET 7=14
```

... must be nonsense. Nor are you allowed to use key command or statement words like GOTO, THEN or AND.

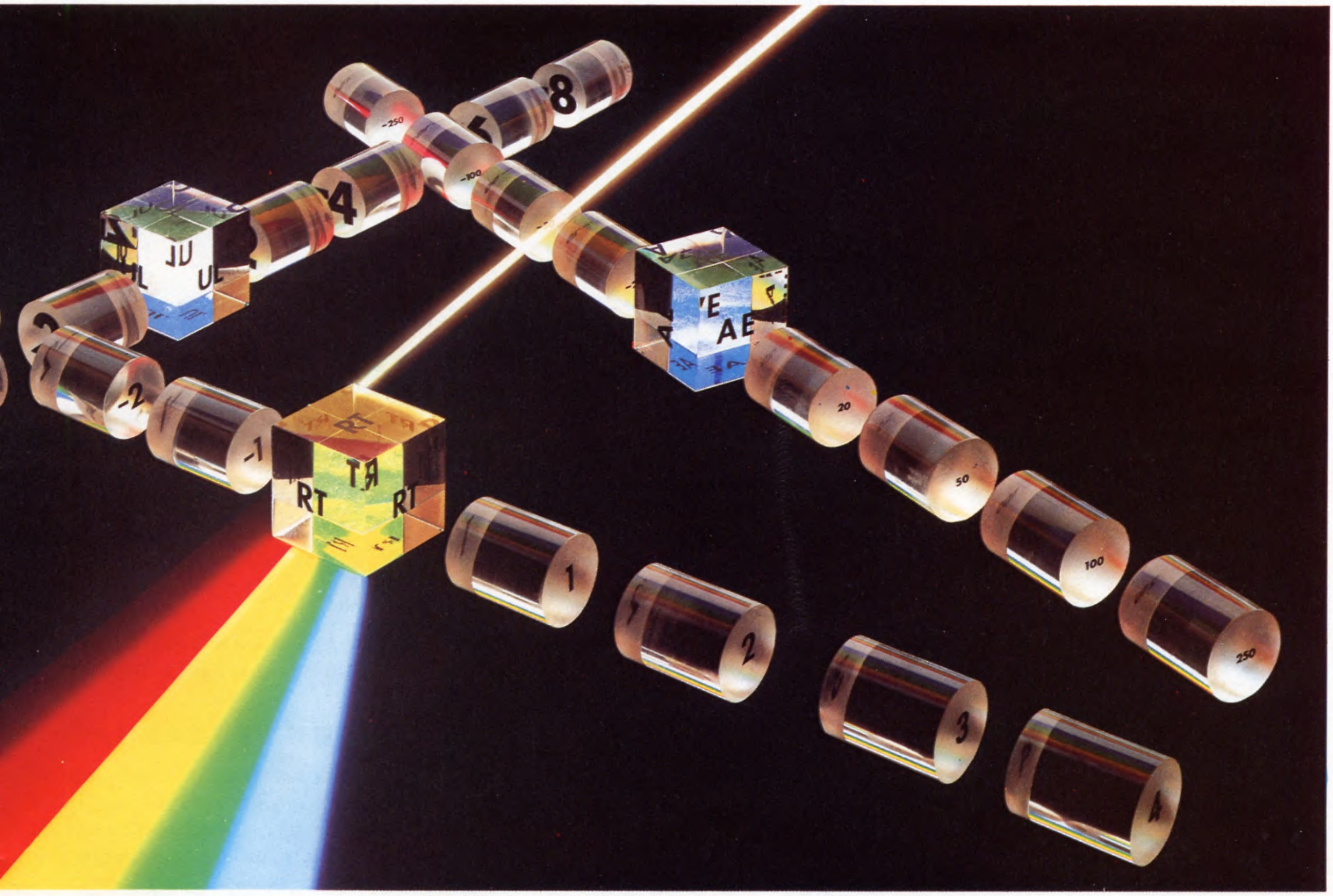
VARIABLES IN ACTION

Most of the programs you write will use several variables, often interacting on one another. Here, for example, is a short program simulating the action of a petrol pump:

```
1Ø LET litres=Ø
2Ø LET pence=Ø
4Ø PRINT "Enter 2, 3 or 4 (star)"
5Ø INPUT a
6Ø IF a=2 THEN LET qty=1/38
7Ø IF a=3 THEN LET qty=1/4Ø
8Ø IF a=4 THEN LET qty=1/42
9Ø CLS
1ØØ IF INKEY$="n" THEN LET pence=
    pence+1: LET litres=litres+qty
1Ø5 PRINT AT 7, 12; a; "□STAR"
11Ø PRINT AT 9,12; "LITRES"
12Ø PRINT AT 1Ø, 12; litres; "□□□□
    □□□□"
13Ø PRINT AT 12, 12; "THIS SALE"
14Ø PRINT AT 13, 1Ø; "£□";
    pence/1ØØ;"□□□"
15Ø IF INKEYS="f" THEN CLS: GOTO 1Ø
2ØØ GOTO 1ØØ
```

```
1Ø LET LITRES=Ø
2Ø LET PENCE=Ø
4Ø PRINT "ENTER 2, 3 or 4 (STAR)"
```

- WHAT IS A VARIABLE?
- HOW VARIABLES ARE USED
- CONTROL VARIABLES FOR **FOR . . . NEXT** LOOPS
- WHAT IS A STRING?
- NUMERIC AND STRING VARIABLES AND THEIR USES
- WHAT IS A NULL STRING?
- HOW NULL STRINGS ARE USED IN GAMES PROGRAMMING



```
50 INPUT A
60 IF A=2 THEN LET QTY=1/38
70 IF A=3 THEN LET QTY=1/40
80 IF A=4 THEN LET QTY=1/42
90 CLS
100 IF INKEY$<>"N" THEN GOTO 105
102 LET PENCE=PENCE+1
103 LET LITRES=LITRES+QTY
105 PRINT AT 7, 12; A; "□STAR"
110 PRINT AT 9,12; "LITRES"
120 PRINT AT 10, 12; LITRES;
   "□□□□□□□□"
130 PRINT AT 12, 12; "THIS SCALE"
140 PRINT AT 13, 10; "£□";
   PENCE/100;"□□□"
150 IF INKEY$="F" THEN RUN
200 GOTO 100
```

```
10 LET LITRES=0
20 LET PENCE=0
30 PRINT "♥▨":POKE650,128
40 PRINT "ENTER 2,3 OR 4 (STAR)"
50 INPUT A
60 IF A=2 THEN LET QTY=1/38
70 IF A=3 THEN LET QTY=1/40
80 IF A=4 THEN LET QTY=1/42
90 PRINT "♥"
100 PRINT "▤▨▨▨▨▨
   ▟▟▟▟▟";A;"STAR"
110 PRINT "▤▨▨▨▨▨▨▨
   ▟▟▟▟▟▟LITRES"
120 PRINT "▤▨▨▨▨▨▨▨▨▨
   ▨▨▟▟▟▟▟▟THIS SALE"
130 GET K$
140 IF K$="F" THEN PRINT"♥":
   GOTO10
150 IF K$="N" THEN LET PENCE=
   PENCE+1:LET LITRES=LITRES+QTY
160 PRINT "▤▨▨▨▨▨▨▨▨▨
   ▟▟▟▟▟▟";LITRES
170 PRINT "▤▨▨▨▨▨▨▨▨▨
   ▨▨▨▨▟▟▟▟▟▟£
   "PENCE/100
200 GOTO 130
```

```
10 LET litres=0
20 LET pence=0
25 @%=&90A
30 CLS: PRINT
40 PRINT "Enter 2, 3 or 4 (star)"
```

```
50 INPUT a
60 IF a=2 THEN LET qty=1/38
70 IF a=3 THEN LET qty=1/40
80 IF a=4 THEN LET qty=1/42
90 CLS
100 PRINT TAB(16,7);a;"□STAR"
110 PRINT TAB(16,9);"LITRES"
120 PRINT TAB(16,12);"THIS SALE"
125 @%=&2020A
130 K$=GET$
140 IF K$="F" THEN CLS: GOTO 10
150 IF K$="N" THEN LET pence=
    pence+1: LET litres=litres+qty
160 PRINT TAB(16,10);litres
170 PRINT TAB(14,13);"£";TAB(2);
    pence/100
200 GOTO 130
```

```
10 LET LITRES=0
20 LET PENCE=0
30 CLS:PRINT
40 PRINT "ENTER 2,3 OR 4 (STAR)"
50 INPUT A
60 IF A=2 THEN LET QTY=1/38
70 IF A=3 THEN LET QTY=1/40
80 IF A=4 THEN LET QTY=1/42
90 CLS
100 PRINT@ 140,A;"STAR"
110 PRINT@ 200,"LITRES"
120 PRINT@ 267,"THIS SALE"
130 LET K=PEEK(344)
140 IF K=251 THEN CLS:GOTO 10
150 IF K=247 THEN LET PENCE=
    PENCE+1:LET LITRES=LITRES+QTY
160 PRINT@ 206,LITRES
170 PRINT@ 330,PENCE;"PENCE"
200 GOTO 130
```

It is worth experimenting with this program until you understand it thoroughly because it has a whole set of variables.

What some of them are doing may be obvious just from looking. The variable a, for example, adjusts a pennyworth of petrol from 1/38th of a litre down to 1/42nd, depending on which grade you want.

Then, as soon as you press N (for 'nozzle') the 'pump' starts clocking over. Pence, originally set at zero by Line 2Ø, clocks up in units of one (there's not much point in having fractions of a penny). And litres, the amount delivered, increases by a pennyworth, qty.

Every time the program reaches Line 2ØØ, so long as you keep pressing the N key, the program goes back to Line 13Ø (1ØØ on the Spectrum) to keep the clock turning over. Not to Line 1Ø, you'll notice—that would just set everything to zero again.

When you RUN this program you'll notice one oddity: some of the decimal fractions you get are very long indeed. This does not mean that there's a fault in your machine. It is just a weakness in computers' ability to convert binary, the computer's arithmetic, into everyday decimal.

CONTROL VARIABLES

As well as being used in LET statements, variables are also used in FOR . . . NEXT loops as you have already discovered (pages 16 to 21). Here, they are known as *control variables*.

On some computers, the characters you can use as control variables differ from those permitted for numeric variables (see box). On the Spectrum, for example, you can say

```
LET score=0
LET best=0
```

. . . but you *cannot* make your program clearer by using a similar set of characters in place of a numeric variable—as in

```
FOR time=99999
```

You must use a single letter, such as T, instead.

STRINGS AND STRING VARIABLES

A string in BASIC is rather like a shopping basket. You can fill it with a variety of items, then handle it as one unit—without having to move each item separately.

Strings can contain almost anything—letters, numbers, spaces, even graphics characters. They must always be 'tied up' with double quotation marks, " and ", at each end.

Here are a few examples:

"CUSTOMER'S NAME AND ADDRESS"
"PLEASE ENTER YOUR ANSWER NOW"
"January 24th"
"Ø1-734 671Ø"
"SW1A 1AA"

The following is also a string, although only on the Spectrum can you key it in directly in this form:

"■■■■■■■■■■■■■■"

The computer can measure a string—that is, how many letters and so on it contains. It can also join two or more strings together end-to-end, or slice one into two or more bits if it doesn't fit your program.

But the *contents* of the string do not interest the computer at all. It can't add them, except in the sense of joining them end-to-end, and it won't multiply or divide them.

To prove this—and also to bewilder your non-computer-owner friends—try this short program:

```
10 PRINT "2+2=□";
20 FOR N=1 TO 200: NEXT N
30 PRINT 1+2*2
```

```
10 PRINT "2+2=□";
20 FOR N=1 TO 40
25 NEXT N
30 PRINT 1+2*2
```

Variables: what you can and cannot use

Type of variable			
Numeric variable	Length: no limit On the Spectrum, capital or lower case letters are allowed but computer does not distinguish between them Punctuation: not allowed Spaces: allowed, but do not count	Length: Maximum 255 characters Capital or lower case letters: either are allowed, but if capitals, must not start with a keyword – eg, TO as in TOTAL Punctuation: only underline allowed Spaces: not allowed	Length: Maximum 255 characters, but only first two are recognised Capital letters only are allowed, but must not start with keyword – eg, TO as in TOTAL Punctuation: not allowed Spaces: allowed, but do not count
Control variable of FOR. . .NEXT loop	Single letter only – eg, A	As numeric variable	Single letter or single letter followed by one character – eg, A, A3
String variable	Single letter followed by $ eg, A$	As numeric variable, but with $ added eg, ADDRESS$	As numeric variable, but with $ added – eg, AD $

As you'll see, the contents of the string are PRINTed exactly as you entered them. Only Line 3Ø is actually calculated and its result PRINTed out. (Line 2Ø, of course, merely makes the computer 'scratch its head' before delivering its answer. On the Acorn change the 2ØØ to 2ØØØ for a better delay.)

If you want to use the same string only once in a program, that's all there is to know.

But if you want to use it more than once, you can save typing time—and memory space—by giving the string a label. This label is called a *string variable*. The length it can be varies from one computer to another (see box), but must be followed by a dollar sign, $.

Here, for example, is the 'bones' of an automatic ordering system:

```
1Ø LET A$ = "PLEASE TYPE IN HOW
   MANY□"
2Ø LET B$ = "□YOU WANT"
3Ø PRINT A$; "BURGERS"; B$:
   INPUT B
4Ø PRINT A$; "FRIES"; B$:
   INPUT F
5Ø PRINT A$; "SHAKES"; B$:
   INPUT S
6Ø PRINT "THANK YOU. THAT WILL
   BE "; (B*55 + F*4Ø + S*35)/1ØØ;
   "□PLEASE"
```

```
1Ø LET A$ = "PLEASE TYPE IN HOW
   MANY□"
2Ø LET B$ = "□ YOU WANT"
3Ø PRINT A$; "BURGERS";B$
35 INPUT B
4Ø PRINT A$; "FRIES";B$
45 INPUT F
5Ø PRINT A$; "SHAKES";B$
55 INPUT S
6Ø PRINT "THANK YOU. THAT WILL BE";
   (B*55 + F*4Ø + S*35)/1ØØ;
   "□PLEASE"
```

It isn't a particularly *good* ordering system; a more visual one would be better. But it does show how strings can shorten a program (though less so on the ZX81).

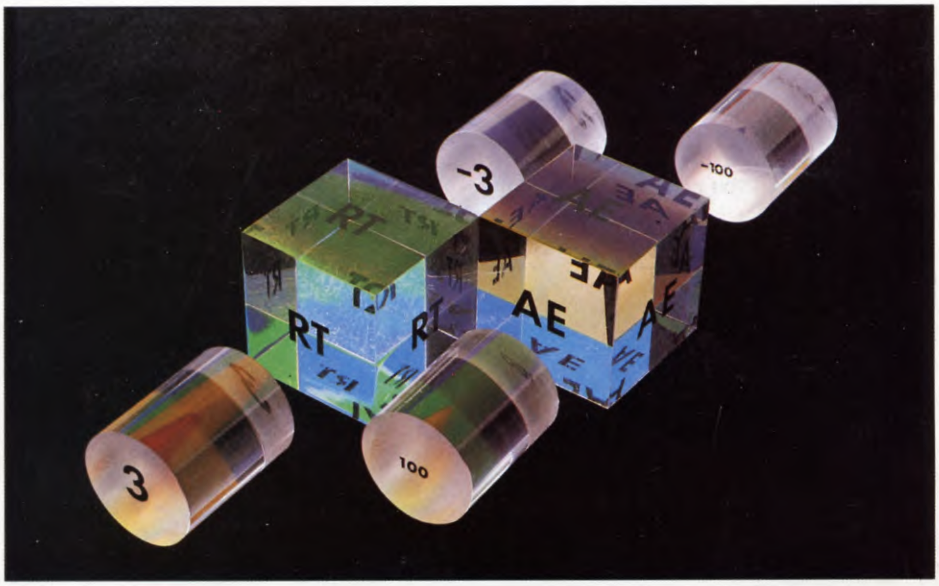
This is why word- and data-processing systems—with things like "Customer account number" and "Price per thousand" and "plus VAT at 15%" occurring over and over again—use string variables extensively.

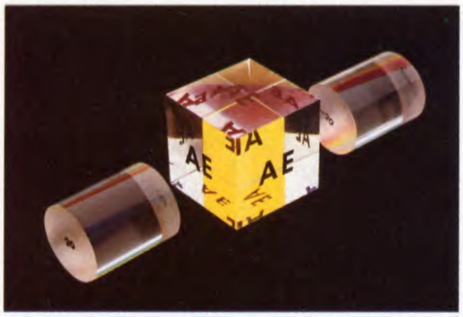
But you can use them in games programming too. Suppose you have a long line of graphics symbols—the wall of a dungeon for an adventure game, say. All you have to do is type it in once, 'label' it and you can use it as often as you like.

You can move such a string around the screen, too, as this short program shows:

```
1Ø LET b$ = "□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□"
2Ø LET a$ = b$ + "□□□□A HAPPY
   EASTER TO YOU ALL□□□" + b$
3Ø FOR n = 1 TO 65
4Ø PRINT INK 2; AT 8, Ø; a$
   (n TO n + 31)
5Ø PRINT INK 2; AT 12, Ø; a$
   (66 – n TO 97 – n)
6Ø BEEP .Ø2, n/2
7Ø NEXT n
8Ø GOTO 3Ø
```

```
4 MODE4
5 VDU23;82Ø2;Ø;Ø;Ø;
```





```
6 VDU19,Ø,4;Ø;
7 VDU19,1,3;Ø;
8 CLS
10 LET B$=“□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□”
20 LET A$=B$+“A HAPPY EASTER
   TO YOU ALL”+B$
30 FOR N=1 TO 65
40 PRINT TAB(Ø,12);MID$(A$,
   N,4Ø)
50 PRINT TAB(Ø,16);MID$(A$,65−N,4Ø)
60 SOUND1,−1Ø,N*2,1
65 D=INKEY(1Ø)
70 NEXT N
80 GOTO 3Ø
```

```
1Ø LET B$=“□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□”
2Ø LET A$=B$+“□□□□A HAPPY
   EASTER TO YOU ALL□□□”+B$
3Ø FOR N=1 TO 65
4Ø PRINT AT 8, Ø; A$ (N TO N+31)
5Ø PRINT AT 12, Ø; A$ (66−N TO 97−N)
7Ø NEXT N
8Ø GOTO 3Ø
```

```
5 CLS
1Ø LET B$=“□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□”
2Ø LET A$=B$+“□□□A HAPPY EASTER
   TO YOU ALL□□□”+B$
3Ø FOR N=1 TO 65
4Ø PRINT@ 16Ø,MID$(A$,N,32)
5Ø PRINT@ 32Ø,MID$(A$,66−N,32)
6Ø SOUND N*3,1: NEXT N
8Ø GOTO 3Ø
```

```
1Ø PRINT “♥”
15 POKE 54277,33:POKE 54278,255:
   POKE 54273+23,15:POKE 54276,33
2Ø LET A$=“□□□□□HAPPY EASTER
   TO YOU ALL□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□”:B$=A$
4Ø LET A$=RIGHT$(A$,61)+LEFT$(A$,1)
45 PRINT “▯⊟▣▣▣▣▣▣▣▣”
   LEFT$(A$,4Ø)
5Ø LET B$=RIGHT$(B$,1)+
   LEFT$(B$,61)
55 PRINT “π▣▣▣▣▣▣”LEFT$
   (B$,4Ø)
6Ø LET S=S+5:IFS>255THENS=Ø
7Ø POKE 54273,S
8Ø GOTO4Ø
```

```
1Ø PRINT“♥”
15 POKE 36878,15:S=127
2Ø A$=“□□□□□HAPPY EASTER
   TO YOU ALL□□□□□□□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□”:B$=A$
4Ø A$=RIGHT$(A$,61)+LEFT$(A$,1)
45 PRINT“▒⊟▣▣▣▣▣▣▣▣”
   LEFT$(A$,22)
5Ø B$=RIGHT$(B$,1)+LEFT$(B$,61)
55 PRINT“π▣▣▣▣▣▣”LEFT$
   (B$,22)
6Ø LET S=S+5:IF S>255THENS=128
7Ø POKE 36876,S
8Ø GOTO4Ø
```

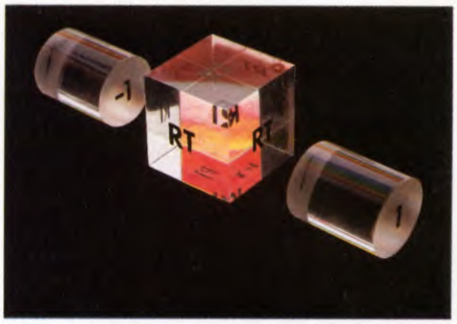
Here, as Line 2Ø shows, you have an enormous string—all the spaces from Line 1Ø (2Ø in the Commodore programs), plus the message, plus all the spaces again—joined together into one. How this is done, and how the string is 'sliced' again to fit the screen, is in a later article. But if in the meantime you want to change the message, make sure your new one (including spaces) is the same length as the original. Otherwise you will have to figure out the implications of the numbers in Lines 4Ø to 55 and change them instead.

An even more common use of the string variable is when the computer is expecting information which will constantly change. For example:

```
1Ø PRINT “WHAT IS YOUR NAME?”
2Ø INPUT N$
3Ø PRINT “HELLO,□”; N$
```

In this program, Line 1Ø PRINTs out the string WHAT IS YOUR NAME on the screen, followed by a question mark. The computer then waits for you to enter a name—that is, a series of letters which form another string—and calls it N$.

On Line 3Ø, it PRINTs out the string “HELLO”, then recalls the N$ it has stored in the line above. So the screen display reads:



HELLO, TOM (or DICK or HARRY)

or whatever name you INPUT.

Note that here the strings WHAT IS YOUR NAME? and HELLO are not given names like A$ or WHAT$ or HELLO$, because they appear only once and the computer does not need to remember what HELLO is—whenever it gets to Line 2Ø it just reads it and PRINTs it out.

The name that you enter does have to be given a label, though. It can be different each time the program is RUN. It also has to be stored on Line 2Ø so that it can be recalled by Line 3Ø.

There is, incidentally, a shorter version of this last program, which is 'stretched' a bit for the sake of clarity. The shorter form is:

```
1Ø INPUT “WHAT IS YOUR NAME”; N$
2Ø PRINT “HELLO,□”; N$
```

(On the Acorn computers change the ; in Line 1Ø to a ,).

NULL STRINGS

Yet another use for the string variable is one you may have already encountered in games programming (pages 54 to 59). It varies slightly from computer to computer, but goes something like this:

```
2Ø IF A$=“” THEN GOTO 1Ø
```

The two lots of quotation marks with nothing between them are known as a *null string*. They mean, 'If the input equals nothing'—in other words, if no key is being pressed—' then go back to Line 1Ø and wait until one is pressed.'

This prevents the computer from skipping through the program so quickly that there simply isn't time for the player to press one of the keys.

Two sets of quotation marks with a space between them, however, are quite a different thing. If you change the line to read:

```
2Ø IF A$=“□” THEN GOTO 1Ø
```

then the program will jump back to Line 1Ø only if the player's input equals a space—in other words, if he is pressing the space bar or space key.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT.*

A
AND 35, 36
Animation 26-32
Applications
hobbies' files 46-53

B
Basic programming
decision making 33-37
programmer's road signs 60-64
random numbers 2-7
the FOR...NEXT loop 16-21
Binary 38, 41, 44, 45
Brackets, use of 35
Breaking out of a program 4

C
Cassette, choice of 25
Christmas program,
Acorn 64
CHR$, *Dragon, Tandy* 26-27
CLEAR
Dragon, Tandy 14, 27
Spectrum 10
CLS, explanation of 27
Craps program 63

D
DATA
best systems to use 45
statements
Acorn 11
Commodore 64 14, 43-44
Dragon, Tandy 13, 40-41
Spectrum 8-9, 44-45
Vic 20 14
Decimal, conversions
from binary 38, 42
Decision making 33-37
DEFPROC
Acorn, see procedures
Delaying action 17
Dice-throwing program 64

E
ENDPROC, *Acorn* 64
Error, causes of 36

F
Filing system program
Acorn 52-53
Commodore 64 48-50
Dragon, Tandy 51-52
Spectrum 50-51
FOR...NEXT loop 16-21
Frog, controlling and
creating a 10-15

G
Games
animation 26-32
controlling
movement 54-55, 57-59
firing missiles 55-58
fruit machine 36
guessing 3-5
moving characters 54-59
routines 8-15
GET 55
GET$, *Acorn, Commodore 64,*
Vic 20 55, 57, 58
GOSUB 62-64
GOTO 18-21, 60-62
Graphics
characters 38-45
creating and moving
UDGs 8-15
frog UDG 10-15
instant embroidery 21
low-resolution 26-32
painting by numbers 19
sunset pattern 20
tank UDG 10-15
also see animation;
movement; ROM graphics;
teletext; UDG
Grid, of UDG
see UDG
Gunshell, firing a
Acorn 12
Commodore 64 15
Dragon, Tandy 14
Spectrum 10

H
Helicopter, building a
Commodore 64 31
Hexadecimal, conversion
from binary 38, 42, 45
Hobbies file 46-53

I
IF...THEN 3, 33-37
IF...THEN...ELSE 37
IF...THEN...GOTO 36
INKEY
Acorn 28-29
INKEY$ 54-55
INPUT statement 3, 4-5
INT 2-3

K
Keypress, detection of 54-55
Keywords, spelling of 19

L
Line numbers, in programs 7
LOAD command 22-25
Logical operators 35-37
Low-resolution graphics
see graphics

M
Machine Code programming
games graphics 38-45
speeding up games routines 8-15
Menu 46-47
MODE, *Acorn* 28
Movement
Acorn 28-29, 58
*Commodore 64, Vic 20*30-31, 59
Dragon, Tandy 26-27, 57
Spectrum, ZX81 31-32, 57

N
Nested loop, definition
and use of 19

O
ON...GOSUB 64
ON...GOTO 62
Opcodes 67
Operators 35
OR 35-36

P
Parameters 64
Peripherals, cassette 22-25
PMODE, *Dragon, Tandy* 12
POKE
Commodore 64 15
Dragon, Tandy 13, 40
PRINT 26-32
Procedures, *Acorn* 64
Program
BASIC 8
BREAKing into 4, 7, 11
line numbers 7
punctuation of 4
slowing down 17
PSET, *Dragon, Tandy* 13

R
RAM 25
RANDOMIZE 2
Random numbers 2-7
ranges of 7
also see games, guessing;
RND function;
RANDOMIZE; tables,
multiplication
READ 40-44
REPEAT...UNTIL, *Acorn* 36
RND function 2-7
Acorn 2, 3, 4-5
Commodore 64 2, 3, 4, 5, 6, 7
Dragon, Tandy 2, 3, 4, 6, 7
Spectrum, ZX81 2, 3, 4, 6, 7
ROM graphics
Acorn 28, 29
Commodore 64 31, 37
Dragon, Tandy 26, 27
Spectrum, ZX81 31, 32
Vic 20 31
Running man, building a,
Acorn 28-29

S
Satellite, building a
Dragon 26-27
SAVE
command 22-25
setting controls 22
verifying 24-25
Shell, firing a 10-15
Sprite definition and use of,
Commodore 64 14, 15
STEP 17, 21
STOP, *Spectrum, ZX81* 4, 64
String variables 4-5
Subroutines 62-63
Symbols, arithmetic 6

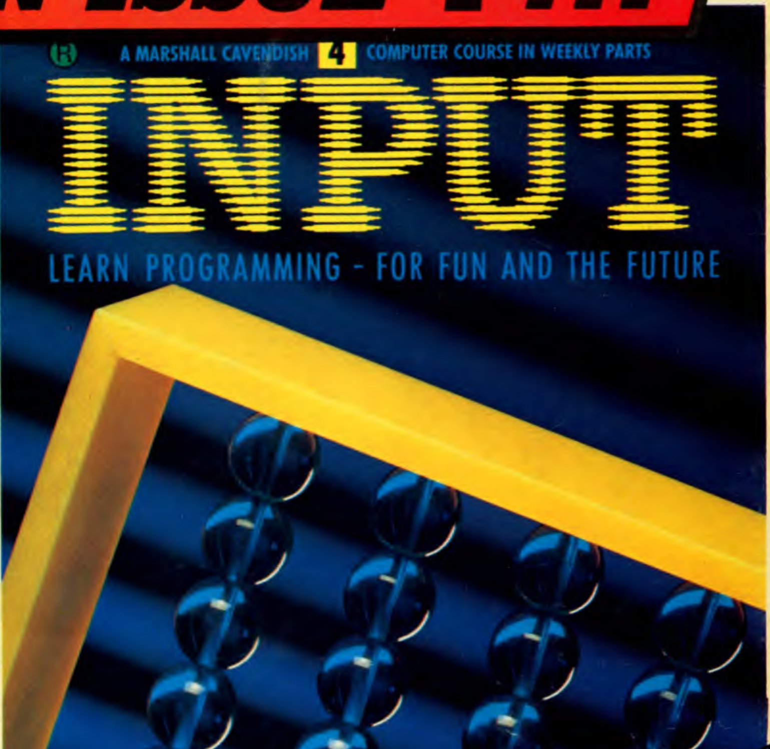
T
Tables, multiplication 5-7
Tank, controlling and
creating a 11-15
Teletext graphics, *BBC* 28-29
Terminating numbers 34

U
UDG, definition of 8-15, 40-44
grids for 8-11
Acorn 11
Dragon, Tandy 13
Spectrum 8-9
DATA for 45
creating your own 38-45

V
Variables,
names for 17
string 4-5
use of 3
VDU command, use of
Acorn 28-29

The publishers accept no responsibility for unsolicited material sent for publication in INPUT. All tapes and written material should be accompanied by a stamped, self-addressed envelope.

COMING IN ISSUE 4 ...


A MARSHALL CAVENDISH 4 COMPUTER COURSE IN WEEKLY PARTS
INPUT
LEARN PROGRAMMING - FOR FUN AND THE FUTURE

- □ *Get your pulse racing by adding* **SCORING AND TIMING** *routines to your games programs. There's lots of information on the routines you need, plus new games to try them on*

- □ *Make use of your computer's ability to handle large amounts of information with the* **BASIC** *commands* **READ** *and* **DATA** *– for anything from directories to screen graphics*

- □ *If you need to write lots of letters and ensure that they all look neat and tidy, there's a simple* **TEXT EDITING PROGRAM** *to help you do just that*

- □ *Understanding how computers count is essential to machine code programs. To start with, learn about different number systems, and what makes* **BINARY** *so important*

- □ *Tidy up your screen displays by learning how to make the best use of the formatting commands when you ask the computer to* **PRINT** *or* **INPUT**

ASK YOUR NEWSAGENT FOR INPUT