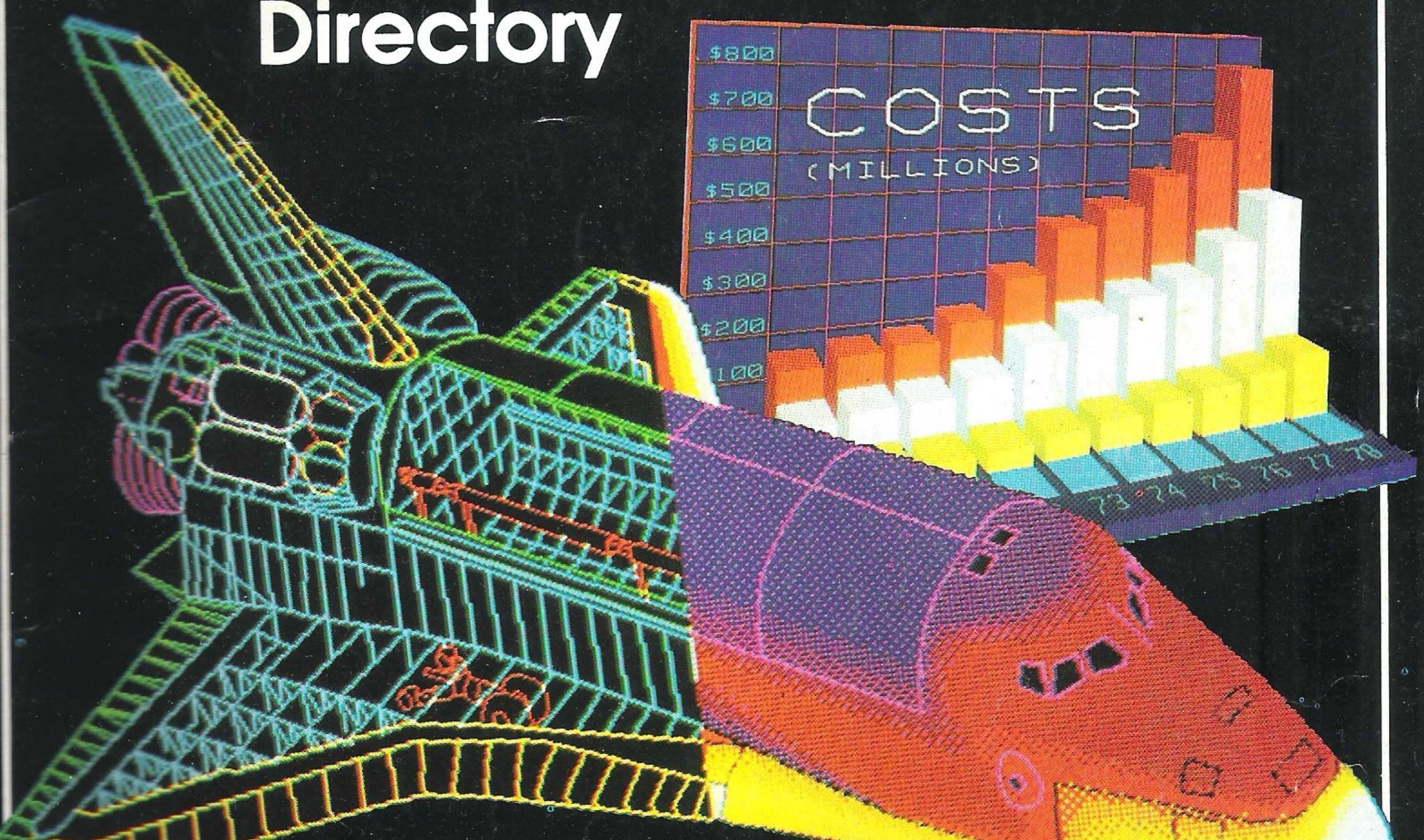
Personal SOFTWARE

ISSN. 0263-0605
AUTUMN 1982 £1.95

The Microcomputer Graphics Directory



- Utilities
- Facts
- Features
- Games

All the essential graphics information a programmer needs in one volume.

Acorn ATOM:Apple II:Atari 400, 800:BBC Model A, Model B:
Commodore 2001,3000,4000, VIC-20:Compukit UK101:DAI:
Exidy Sorcerer:NASCOM 1,2,3:NEC PC-8001:Ohio Scientific
Superboard II:Research Machines 380Z:Sharp MZ-80A,
MZ-80B,MZ-80K, PC-1500:Sinclair Spectrum,ZX81:Tandy
TRS-80 Model 1,Model 3,Color Computer:Tangerine
Microtan 65,Micron:Texas Instruments TI-99/4A:
Transam Triton,Tuscan:Video Genie 1,2

AF

ATOM & BBC

A.T.O.M

NUMBER

ADVENTURE II COMPETITION

FIRST PRIZE £100 CASH

ZODIAC

Following the success of our first Adventure Competition A & F are launching "Adventure Competition II".
The £100 First Prize will be in the winners hands in time for Christmas.

Find your way through the Houses of the Zodiac, beware the demons and monsters, survival is not easy but the answers can be found! Can you gain immortality?

Requires 12K RAM
Price £6.00
Closing date 30th November 1982

ATOM GAMES AT £4.95

CYLON ATTACK
A FAST MOVING 3D SPACE BATTLE
As you sit in your starfighter looking out into the void of space remember the CYLON race only want Mankind for food!! You glance up at your long range scanner the CYLON fleet is in range. Quickly you select a target and turn to meet it ready to defend Earth to the end!!
5K Text 6K Graphics
(An excellent screen, nice sound and a fast moving game. Y.C. July 82)

MISSILE COMMAND A fast moving version of the popular arcade game. You have three bases from which to fire your defence missiles, protecting your cities and bases from the missiles and aircraft attacking you!!!
SCORE/HI SCORE/MULTIPLE LEVELS/SOUND
5K Text 6K Graphics

MINEFIELD A fast moving all action game that requires skill and quick reaction! You must shoot the supermines with your killer tank before their time fuse runs out, or risk being destroyed when they explode!!!
SCORE/HI SCORE – FUEL/AMMO COUNTERS – SOUND
5K Text 1/2K Graphics

EARLY WARNING Using radar controlled defence systems you must destroy the waves of ICBMs attacking your city, if one gets through, you're DEAD!!
HI RES DISPLAY – 48 LEVELS – SOUND
5K Text 6K Graphics

POLECAT Avoiding the hungry POLECAT can you find the way through the rabbit warren onto the surface and return to the safety of the burrow with the food you have collected? Millions of mazes just as you get used to the maze it changes!!
5K Text 6K Graphics

ATOMIC CUBE Our version of the popular RUBIC CUBE. A 3D front and rear view of the cube is displayed in high resolution colour* graphics. Mixed and ready for you to solve. THIS IS NOT AN AID TO CUBE SOLVING BUT A PUZZLE WHICH YOU HAVE TO SOLVE!!! If you think RUBIC is easy try our challenge!!!
*LOOKS GREAT IN MONOCHROME AS WELL!
5K Text 6K Graphics + FP, ROM

DEATH SATELLITE
AN ADVENTURE PROGRAMME
Your Time Capsule has materialized on a deserted satellite. The only way to escape is to find a fuel source to recharge your capsules energy cells. Can you solve the mystery and avoid the traps on the "DEATH SATELLITE" or will you fail?
Requires 12K RAM

A ★ A & F SOFTWARE ★ F

ATOM GAMES AT £3.95

ROBOT NIM
An animated version of this popular puzzle. Watch the antics of the control robot as you struggle with the ATOM for control of the game. The writer has incorporated a fine touch to the animation and matched with the sound effects makes it a challenging puzzle with a touch of humour making ROBOT NIM fun to play or watch.
5K Text 1/2K Graphics
(Even if you find NIM boring you will be entranced by this gorgeous version Your Computer July 1982)

POLARIS
Your submarine is ordered to sink an enemy convoy. By using skill, cunning and strategy can you avoid the escorting warships and dangerous shallows to succeed with your mission?
5K Text 6K Graphics

ATOM UTILITY PROGRAMME
ATOM SUPERCOS

Tired of waiting for your programs to load? SUPERCOS is a low cost HIGH SPEED (1200 baud) COS you will be able to save/load 5 times faster than normal. In addition SUPERCOS provides visible load, program test verifying, plus 8 other commands. Req's 1.25RAM **Only £5.95**

BBC GAMES

MODEL A & B
BBC A popular game from our Atom range converted and enhanced to run on the BBC Micro

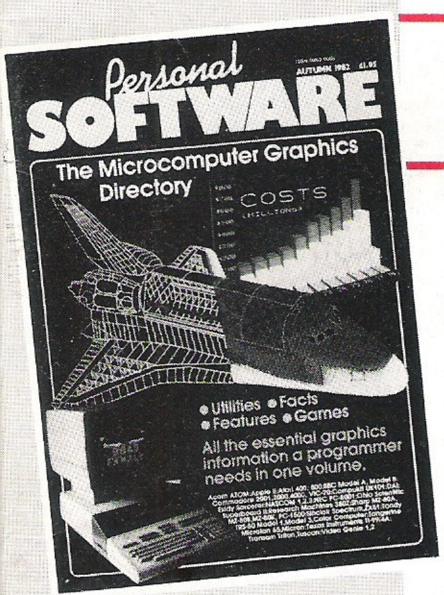
POLECAT Find your way through the warren onto the surface and return to your burrow with winter supplies. Easy!! – Beware the Polecats. **£6.00**

MODEL B ONLY
EARLY WARNING Destroy the attacking waves of ICBMs using a radar tracking system and intercept missiles. High resolution colour graphics. **£6.00**

ROADRUNNER The opposition will stop to get you. In this arcade style car chase: dodge through the Sunday traffic, weave to avoid their wire as the black cars try to shoot you off the road! Beware the hells angels who assist them! Can you survive?
Excellent sound and graphics **Introductory Offer £6.00**

MANCHESTER SHOWROOM NOW OPEN
830 HYDE ROAD, GORTON, MANCHESTER

Orders to:
A & F SOFTWARE (Dept. AC1)
10 Wilpshire Ave., Longsight, Manchester M12 5TL
Access/Visa orders on (061) 320 5482.
All prices as stated, no hidden extras.
Send sae for catalogue


Personal SOFTWARE
AUTUMN 1982
The Microcomputer Graphics Directory
COSTS
• Utilities • Facts
• Features • Games
All the essential graphics information a programmer needs in one volume.

Volume 1
No 2
Autumn 1982

Editor: Henry Budgett
Assistant Editor:
Roger Munford
Advertisement Manager:
Neil Johnson
Advertisement Copy Control: Sonia Hunt
Managing Editor:
Ron Harris BSc
Managing Director: T J Connell

Origination and design by MM Design & Print

Personal Software is a quarterly magazine appearing on the third Friday in May, August, November and February.

Distribution by: Argus Press Sales & Distribution Ltd, 12-18 Paul Street, London EC2A 4JS Tel: 01-247 8233. Printing by: Alabaster Passmore & Sons Ltd, Tovil, Maidstone, Kent.

Subscription Rates : UK £7.80 for four issues including postage. Airmail and other rates upon application to Personal Software Subscriptions, 513 London Road, Thornton Heath, Surrey CR4 6AR.

CONTENTS

Editorial & Advertisement Office
145 Charing Cross Road, London WC2H 0EE
Telephone: 01-437 1002 Telex: 8811896

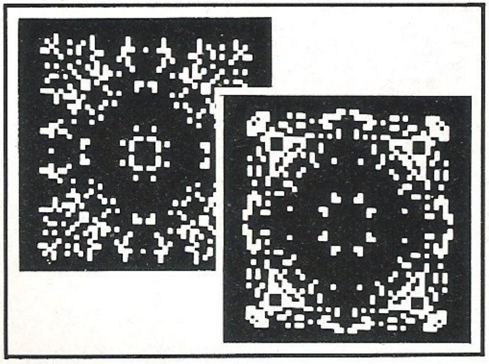
Preface 4

FEATURE

Interactive Graphics 6
The beginner's guide

GAMES

NASCOM Life 18
That's life!
Moonbase Alert 22
A lunar listing
Snakes 24
Wriggle out of this one
Holocaust 26
Wear your mushroom with pride
Ambush 30
They'll get you in the end
Ski Run 32
Watch out for the trees!
NAS Wars 34
At war with your NASCOM?
Labyrinth 36
An a-maze-ing game
Pinball 42
Be a Pinball Wizard
Stomper 44
Stamp them out!
Kirk vs The Cursor 46
A boldly ongoing game
Patterns 49
Pretty good graphics!



UTILITIES

'Scope Simulation 52
Just like the real thing
Screen Print 56
Printing made easy
Line Plotter 58
A new line in plotting?
PET Lister 60
Standard listings for all
Graph Plotter 62
The ups and downs of plotting
Cross Hatcher 64
Don't get cross with your TV
Double Density 65
Packing your display
Fast Plotter 66
A quick worker!

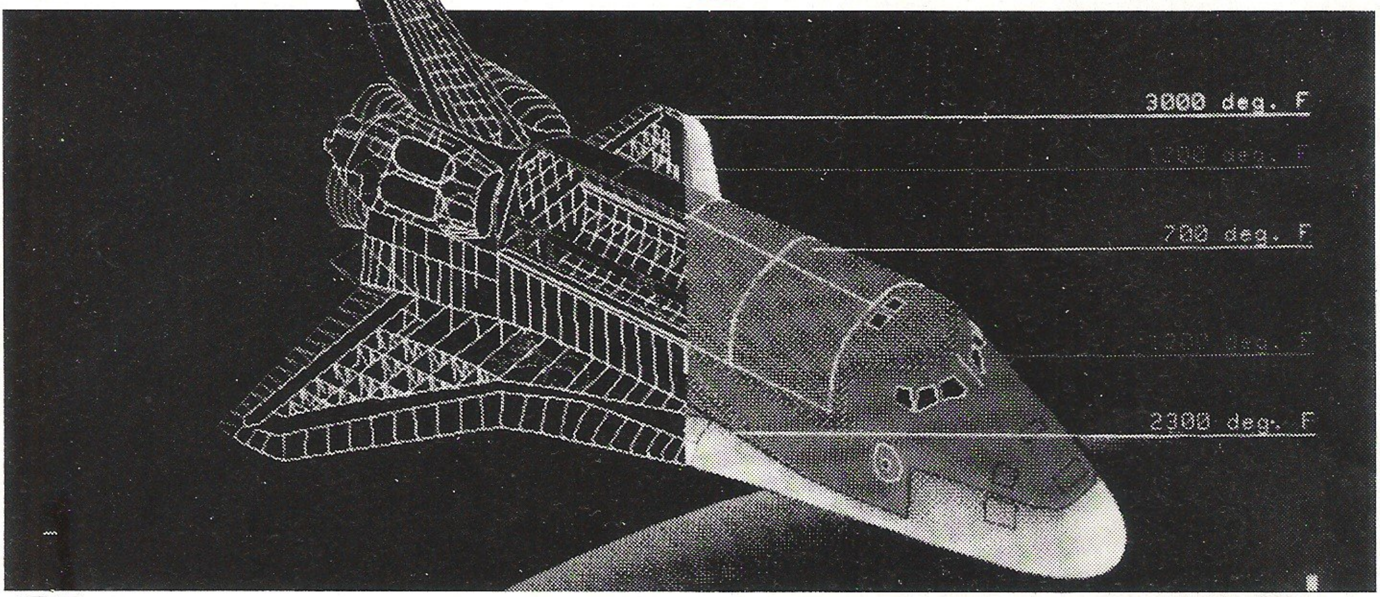
INFORMATION

Graphic Details 70
All you ever wanted to know . . .
Graphics Directory 85
The essential guide to all the popular micros and their graphics capabilities
Corrections 96
BBC program corrections
Bibliography 98
Read all about it

Computing Today 16
Book your monthly copy here
ZX Computing 21
The magazine for the Sinclair user
The Valley 45
Now available for Atari, VIC and Sharp MZ-80A users
Personal Computing Today .. 50
Make it monthly
Personal Software 97
Copies by post, post free!

Credits: Our grateful thanks to the following individuals and organisations for their assistance in the production of this issue. Hewlett-Packard's Redhill office for providing the photographs, Hemdale International Films Ltd, Leisure Investments Co, UIP (Paramount), 20th Century Fox, Columbia Pictures Industries Inc, Kieran, Walter, Lesley, Chris, John, Enzo and Junior.

PREFACE



Photograph courtesy of Hewlett-Packard

Graphics. What are they and what can you do with them? Those two questions are not quite as silly as they may at first sound, because there are many types of graphics and even more ways of using them. The first use that springs to most people's minds is games such as Space Invaders, Pacman, arcade games and the like. Certainly, these are valid and attractive uses of the graphics facilities of small computers but they are not the only use.

In this, the second issue of Personal Software, I have tried to assemble a complete, across-the-board package of features, information and programs to do with things graphical. All the features and program material has been previously published in Computing Today over the last three years and has been thoroughly tested by you, the readership. The information section at the back of the magazine is intended to provide the maximum amount of assistance for people converting programs which incorporate graphics from one system to another. The Graphics Details charts have been added to over the years and this is the first time the whole collection has been published as one . . .we've even added in a couple of new ones too! The Graphics Directory, however, is a totally new compilation of information and sets out to provide a quick reference to the sort of graphics facilities available on virtually every common micro under £1,000.

The main area of interest for many readers will, of course, be the collections of programs. These have been split into two sections; Games and Utilities. The first section is really fairly self-explanatory and provides a reasonable cross-section of the sort of games which can be brightened up by the clever use of graphics. The second, and more serious, section provides a wide range of ready-made utility programs which can be used either on their own or as part of an existing program.

All the programs have been reformatted into the CT standard format and all known errors found in the originally printed versions have been corrected. The range of machines covered is wide and there should be something for almost everybody, even if it is an idea rather than an piece of code.

After the problems we encountered with the first issue of Personal Software over the sizes of programs for the BBC Micro, we have slightly altered our way of indicating the size of memory required for the programs to operate. Unless specified to the contrary, ***all*** programs presented here should run in the standard size of machine; a Tandy Level II BASIC program will, for example, assume a 16K machine. This may seem a somewhat negative method of approaching the problem but it should prove more reliable.

Having covered the contents of the book from the back to the middle, what about the front? The first main feature is Interactive Graphics and this was first seen as a three-part series in Computing Today. The reason for its inclusion is simple, it is virtually the only complete introduction to the use of simple graphics techniques which has ever been published! If you have just bought a machine which has graphics facilities or you are looking for new and different ideas on how to get more out of your graphics system, then this is the logical starting point. If you already feel that you know all there is to know about computer graphics, then it is still probably a good place to refer to as it is unlikely that you know ***everything***. After all, the answer to Life, The Universe and . . .well, Everything is only an asterisk!

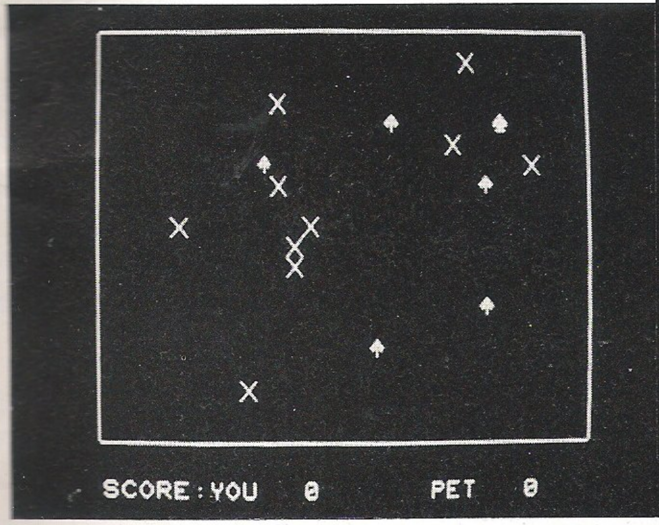
FEATURE

Interactive Graphics

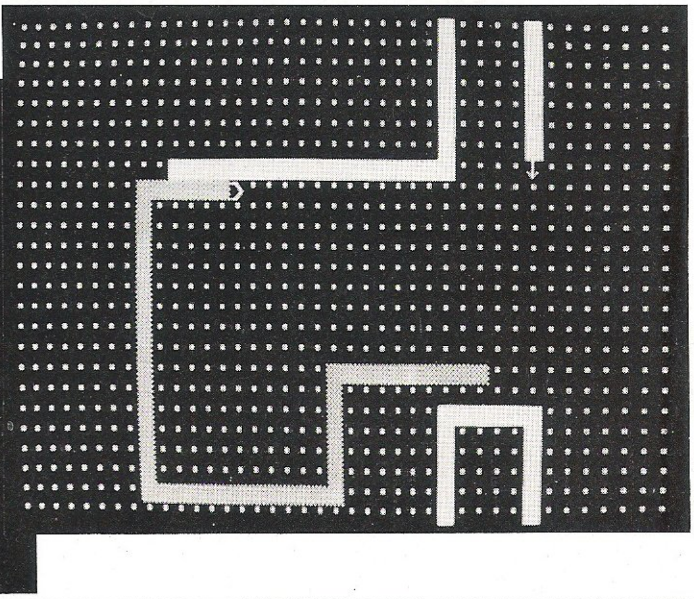
Originally published in Computing Today (November, December 1980 and January 1981) as a three-part series, Interactive Graphics introduces the concept of creating moving graphics on your micro and how to utilise them within existing programs.

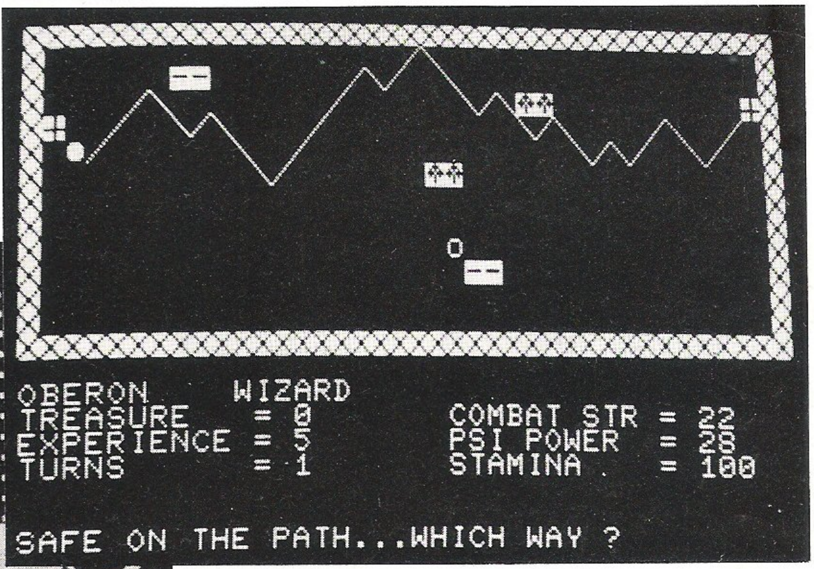
Complete with a full explanation of the use of POKEs and PEEKs, the feature continues with details on how to make cursor control work for you. The last part of the article looks at the various characters a micro can display and how to use them to your best advantage.

Programs contained within the feature include simple movement routines, demonstrations of PEEK and POKE in use, examples of how cursor controls can be utilised and ways that machine code can be made to speed everything up.

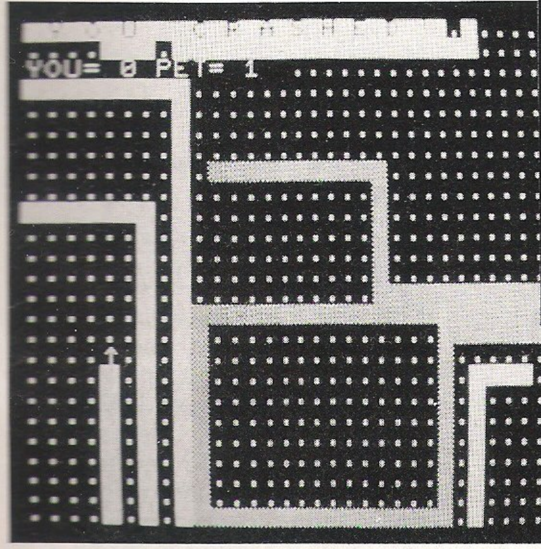


```
SCORE:YOU   0        PET   0
```





```
OBERON      WIZARD
TREASURE    = 0          COMBAT STR = 22
EXPERIENCE  = 5          PSI POWER  = 28
TURNS       = 1          STAMINA    = 100

SAFE ON THE PATH...WHICH WAY ?
```

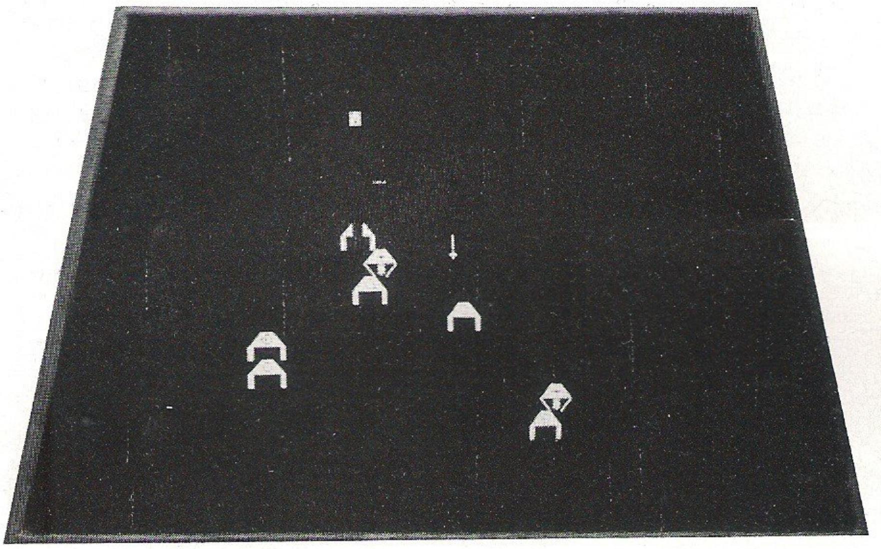


```
YOU= 0 PET= 1
```

Trevor Lusty

INTERACTIVE GRAPHICS

A programmed course in making things move around your screens.



Getting things to move on a video display is not so very difficult when you think about it. It's surprising that we ever manage to keep anything stationary. After all, you're actually watching a moving spot which crosses the screen 625 times every 1/25 of a second. Now, if your display is 12" wide, that's 625x25 ft/sec — which is ove 21,000 mph!

Memory mapped video is a display system in which each character position on the screen is actually a memory location. A unique function of this block of memory is that it is accessed by both the processor and the video circuitry. The memory assigned to video is constantly scanned to refresh and update the screen, and consequently, any change in the location value is immediately visible. We shall be using the PEEK function to look at the values in these memory locations and the POKE function to change them.

Any video display which uses a memory mapped technique is capable of producing simple graphics, and the programs contained in this article are examples of various possibilities. They should work with any computer which has PEEK and POKE statements and uses this type of display.

A Random Start

Before we actually begin, it is important to realise some of the dangers. Indiscriminate POKEing is likely to spell instant disaster! Find out the value, base 10, of the memory address of the top left-hand corner of your screen, the number of characters per line and the number of lines per screen. I have assigned the following variables to these values:

SP = Screen Pointer
(PET = 32768, TRS-80 = 15360, etc)
LL = Line Length
(PET = 40, RML 380Z = 64, etc)
PL = Page Length
(PET = 25, TRS-80 = 16, etc)

Set up these values according to your system and you should have few problems. The value of the address for a position X spaces across and Y lines down may be calculated as SP + Y*LL + X. Okay — enter and run the following program. (Remember, my values are for a PET.)

```
10  SP=32768:LL=40:PL=25
20  FOR J=0 TO 255
30  POKE SP,J
40  NEXT J
```

All the characters available should have appeared in quick succession in the top left-hand corner of the screen. If this does not happen check on the value for SP. Now, that wasn't very helpful, so let's try to space the characters out a bit. Change line 30 to POKE SP + J,J and re-run the program. The characters should appear on the top few lines of the screen, but the RML 380Z will not display all the characters using this method as there is some addressing conflict at the right-hand edge of the screen.

We are now ready for our first 'mind-blower'. The following program POKEs random characters to random positions on the screen — try it and see:

```
10  SP=32768:LL=40:PL=25
20  FOR I=1 TO 1000
30  RL=INT(PL*RND(1))
40  RP=INT(LL*RND(1))
50  POKE SP+RL*LL+RP,INT(256*
    RND(1))
60  NEXT I
```

You should now have characters splattered all over the screen. What we need now is to bring some order to this apparent chaos.

A New Kind Of Art

The first thing to do is to choose a character from those available. I have found the reverse of the Space key to be the most suitable, this has a POKE code of 160 on the PET. (Your value may be different.) Enter the following program with a POKE code number which works for you:

```
1ØØØ   SP=32768:LL=4Ø:PL=25
1Ø2Ø   PRINT CHR$(147)
11ØØ   POKE SP+4*LL+4,16Ø
```

The CHR$(147) is to clear the screen, and when you run this program the result should be a single white blob in the top left-hand quadrant. Add the following lines of code and re-run the program:

```
112Ø   POKE SP+4*LL+(LL-5),16Ø
114Ø   POKE SP+(PL-5)*LL+4,16Ø
116Ø   POKE SP+(PL-5)*LL+(LL-5),16Ø
```

The result this time should be four blobs symmetrically placed on the screen. Figure 1 shows how the addresses are calculated in order to achieve this effect. Remember that the top line of the screen is line zero and the left-hand column is column zero. Adding four blank lines at the top of the screen therefore puts us on the fifth line down. This explains why five must be subtracted from the page and line lengths in order to obtain a symmetrical result.

You are now ready for 'BLOTCH' which is listed below. This program POKEs a symmetrical pattern to the screen and then, after a short pause, continues with a new display. Lines 1200 to 1260 show how to POKE a string of characters to the screen. The + 64 in line 1240 produces reverse video on the PET — you might have to leave it out.

```
1ØØØ   SP=32768:LL=4Ø:PL=25
1Ø2Ø   HL=INT(LL/2):HP=INT(PL/2):PRINT
       CHR$(147)
1Ø4Ø   FOR I=1 TO 15Ø
1Ø6Ø   X=INT(RND(1)*(HL+2)):
       Y=INT(RND(1)*(HP+2))
1Ø8Ø   X=INT(RND(1)*X):Y=INT(RND(1)*Y)
11ØØ   POKE SP+Y*LL+X,16Ø
112Ø   POKE SP+Y*LL+(LL-X-1),16Ø
114Ø   POKE SP+(2*HP-Y)*LL+X,16Ø
116Ø   POKE SP+(2*HP-Y)*LL+(LL-
       X-1),16Ø
118Ø   NEXT I
12ØØ   W$="BLOTCH"
122Ø   FOR X=1 TO LEN(W$)
124Ø   POKE SP+HP*LL+HL-1+X-LEN(W$)/2,
       ASC(MID$(W$,X,1))+64
126Ø   NEXT X
128Ø   FOR I=1 TO 5ØØØ:NEXT I
13ØØ   RUN
```

'BLOTCH' weights the address numbers towards the corners using lines 1060 and 1080. However, if the result is still too random for you, try the following program. This has a much more mathematical flavour and produces a real piece of modern art. Line 1050 may be omitted by non-PET users, it merely demonstrates another way of getting the title onto the screen.

```
1ØØØ   SP=32768:LL=4Ø:PL=25
1Ø1Ø   FOR I=SP TO SP+PL*LL:POKE
       I,16Ø:NEXT I
1Ø2Ø   FOR L1=Ø TO PL-1
1Ø3Ø   POKE SP+L1*LL+(INT(6*RND(1)+1)*
       INT(6*RND(1)+1)),32
1Ø4Ø   NEXT L1
1Ø5Ø   PRINT "[HOM][REV]CUBISM[CD]
       [6 CR]U[CD][CL]B[CD][CL]I[CD]
       [CL]S[CD][CL]M[OFF]"
1Ø6Ø   FOR P1=Ø TO LL-1
1Ø7Ø   POKE SP+LL*(INT(5*RND(1)+1)*
       INT(4*RND(1)+1))+P1,32
1Ø8Ø   NEXT P1
1Ø9Ø   GOTO 1Ø2Ø
```

Getting Things Moving

So far, so good — we can now make patterns appear before our eyes, but as yet, we have no illusion of movement. Think of the Blackpool illuminations, and the way they make a static line of bulbs appear to move. Let's try to emulate that effect. For each bulb we will use an asterisk (*) for which the POKE code is 42. Enter and run the following program:

```
15Ø   SP=32768:LL=4Ø:PL=25:PRINT
      CHR$(147)
16Ø   FOR J=1Ø TO LL-1Ø
17Ø   POKE SP+J,42
18Ø   NEXT J
```

Well, there's our line of bulbs — not very exciting yet. Add the following code to your program and re-run it.

```
19Ø   FOR J=1Ø TO LL-1Ø STEP 2
2ØØ   POKE SP+J,32
21Ø   NEXT J
```

The last piece of code has switched some of the lights off because 32 is the POKE code for a space. Now we will try to be clever. We will work our way along the line of bulbs, moving the pattern one position to the right. Add this coding and re-run the program:

```
22Ø   T2=PEEK(SP+LL-1Ø)
23Ø   FOR J=1Ø TO LL-1Ø
24Ø   T1=PEEK(SP+J)
25Ø   POKE SP+J,T2
26Ø   T2=T1
27Ø   NEXT J
```

Can you see the way variable T2 is used to make the pattern loop back on itself? All we have to do now is add:

```
28Ø   GOTO 23Ø
```

and the trick is complete — try it and see.

The following program incorporates the above techniques to illustrate how they might be used to bring the

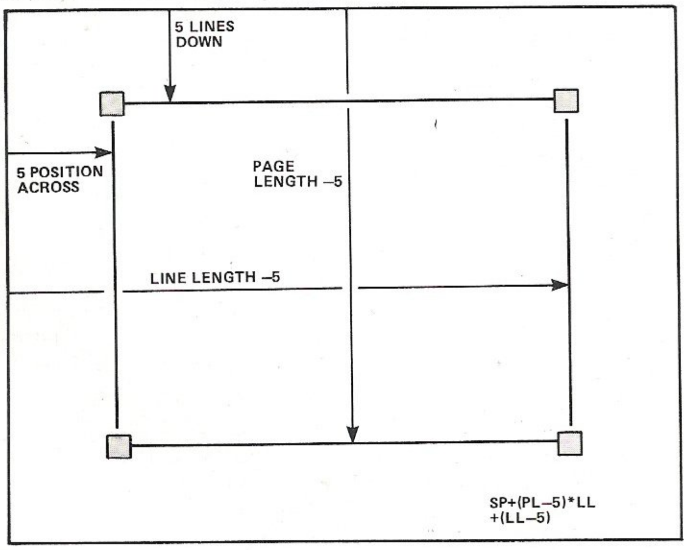
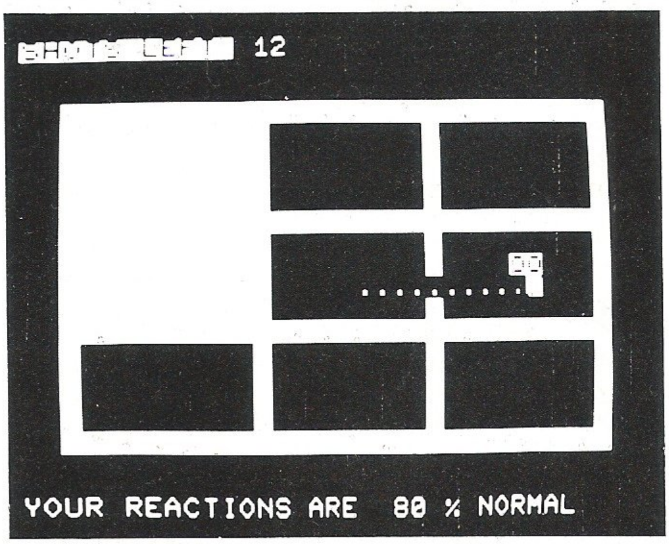


Fig. 1. How to calculate character positions on the screen.



12

YOUR REACTIONS ARE 80 % NORMAL

A symmetrically placed grid is used as the play area in the game of Wumpus.

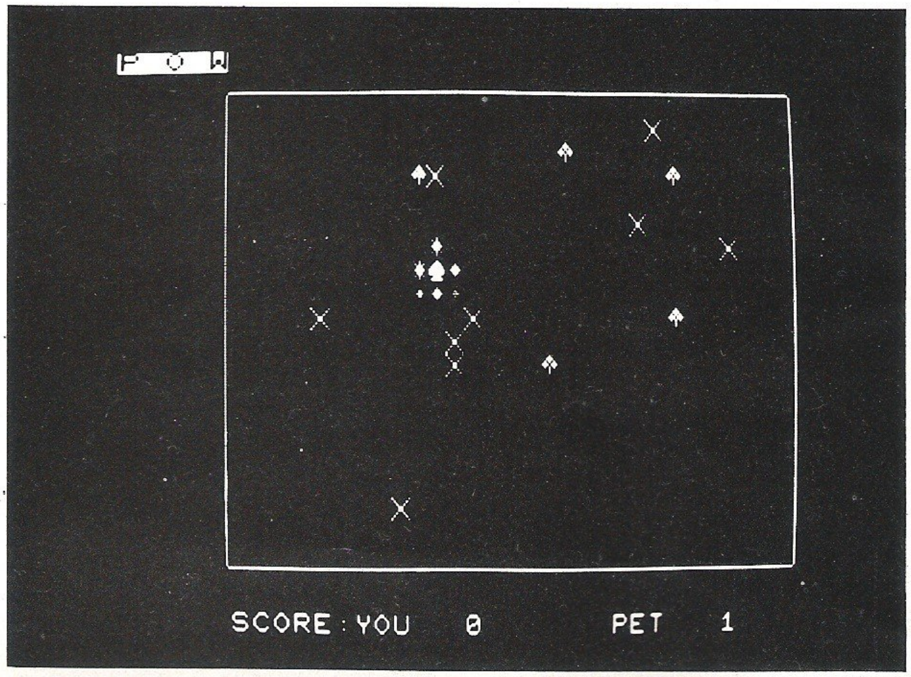
▶

screen to life. I thought of the name 'ENTOMB' as I was writing it — I think you'll agree that it's quite apt. Here is the complete listing, and the explanation follows:

```
200   SP=32768:LL=4Ø:PL=25:PRINT
      CHR$(147)
21Ø   FOR J=Ø TO PL-1
22Ø   POKE SP+J*LL,16Ø
23Ø   POKE SP+J*LL+LL-1,16Ø
24Ø   NEXT J
25Ø   FOR J=Ø TO LL=1
26Ø   POKE SP+J,16Ø
27Ø   POKE SP+24*LL+J,16Ø
28Ø   NEXT J
29Ø   REM ** SET STARTING POSITION
3ØØ   REM ** AND DIRECTION VECTORS
31Ø   X=INT(LL/2):Y=INT(PL/2)
32Ø   P=SP+Y*LL+X;X1=1:Y1=LL
33Ø   REM ** CHANGE DIRECTION IF
34Ø   REM ** WE ARE BLOCKED
35Ø   IF PEEK(P+Y1)=16Ø THEN Y1=-Y1
36Ø   IF PEEK(P+X1)=16Ø THEN X1=-X1
37Ø   P=P+X1+Y1:POKE P,42
38Ø   IF (P=P6) AND (P=P2) THEN END
39Ø   REM ** ADJUST LENGTH AND
4ØØ   REM ** POKE THE OBSTACLES
41Ø   P7=P6:P6=P5:P5=P4:P4=P3:P3=P2:
      P2=P1:P1=P
42Ø   POKE P7,32
43Ø   POKE SP+1ØØØ*RND(1),16Ø
44Ø   POKE SP+LL*(1+INT((PL-2)*
      RND(1)))+(LL*RND(1)),32
45Ø   GOTO 35Ø
```

'ENTOMB' may be entered and run in stages. Let's start with lines 200 to 280. These lines clear the screen and set up the border. Type in the program up to this point and run it to ensure that it works correctly. This is most important as we will be using the border to keep our asterisk snake confined. . .and if it gets loose you might be POKEing in some unfortunate places.

Make sure you understand the following piece of arithmetic gymnastics before you proceed further. In order to give the illusion of movement we must be able to move vertically, diagonally or horizontally from any position on the screen, but our only reference to our present position will be a POKE number P. What values must be added or subtracted to this number for the required movement? The solution is fairly easy: to move right one position add one and to move left subtract one. To move vertically down,we add a whole line length and to move up, we subtract this value. Diagonal movement requires a combination of these two movements, as Fig. 2 shows. The variables, X1 and Y1, are used to store the required increments, and direction may be changed by changing the sign of one or both of these. Lines 350 and 360 show how we can look ahead in the direction of motion and change direction if we hit the border. Type in the program up to line 370, add line 450 and run the program again.



Another old favourite, Chase, uses a fixed play area in which you move around avoiding unpleasant death.

The result should be an ever-increasing string of asterisks which appear to bounce off the border surrounding the screen. Use the Break key to stop the program as it is in an infinite loop. You would be well advised to store temporary copies of your program as you go along, just in case the ravenous, expanding monster which you have just created escapes and plonks asterisks through main memory!

Some limit on the length of our snake is required, and this is achieved by lines 410 and 420. These lines store previous positions for the head and POKE a space when the head has moved seven steps. Insert these lines into your program and you'll see what I mean.

The program is completed by line 430 which adds extra obstacles to the screen, line 440 which gives the snake a chance to escape and line 380 which stops the program when the snake is in danger of disappearing up its own posterior. The program continues until the poor snake is entombed in a mass of white blobs.

Making It Faster

As you add more coding and try to make more things move at the same time, things gradually get slower. This can be overcome in two ways, either by improving your BASIC coding or by using machine code. The following program is an example of how a little thought will speed up your BASIC.

```
1ØØ   SP=3282Ø:LL=4Ø
11Ø   FOR K=1 TO 5
12Ø   FOR L=1 TO 2Ø
13Ø   POKE SP+K*LL+L-2,32
14Ø   POKE SP+K*LL+L-1,46
15Ø   POKE SP+K*LL+L,42
16Ø   NEXT L
17Ø   POKE SP+K*LL+L-2,32
18Ø   POKE SP+K*LL+L-1,32
19Ø   NEXT K
2ØØ   REM ** THE SAME BUT
21Ø   REM ** MUCH FASTER
22Ø   FOR K=1 TO 5:Y=SP+K*LL:FOR L=1
      TO 2Ø:X=Y+L
23Ø   POKE X-2,32:POKE X-1,46:
      POKE X,42
24Ø   NEXT:POKE X-1,32:POKE X,32:NEXT
```

The POKEing techniques should now be familiar to you, but note how the second section is condensed and how unnecessary calculations are removed. When you run this program you will see just how much faster the second version is. The coding for the second section is much more difficult to understand, so I suggest that

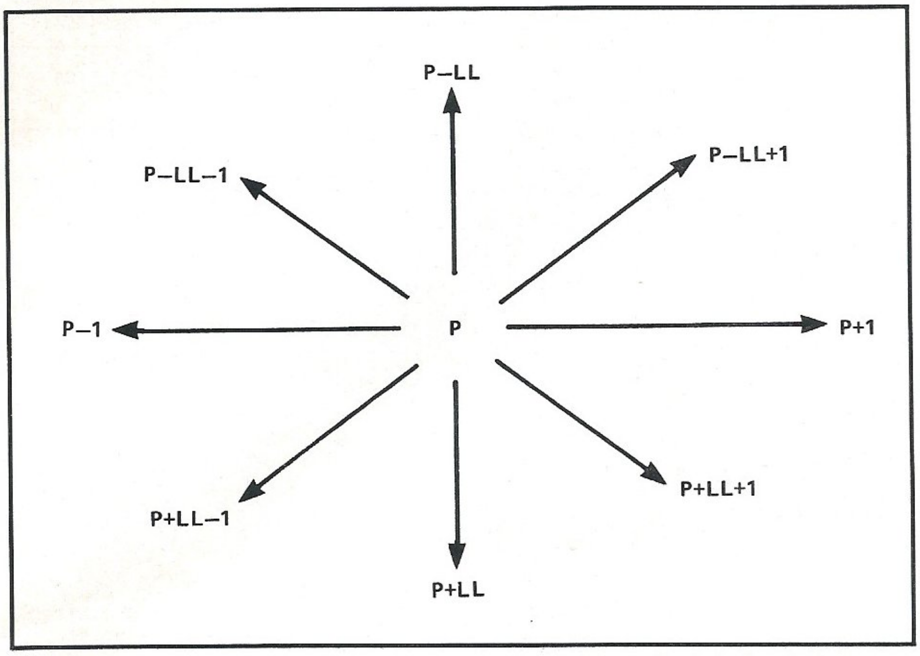


Fig. 2. Movement direction from any given point can be calculated quite simply.

you only use this technique to speed up an already working program.

There are times, however, when BASIC just isn't fast enough. Imagine that you are on the Enterprise when the Klingons attack, POKEing a few blobs on the screen is just not dramatic enough. What we would like to do is to reverse the whole screen a couple of times to simulate an explosion. This requires a machine code routine. My problem here is that not all machines use the same processor or have the same memory map, but the following routine shows how this effect can be achieved on a PET.

```
Ø33A           REVERSE SCREEN ROUTINE
Ø33A
Ø33A           65Ø2 ASSEMBLER LISTING
Ø33A           FOR THE PET
Ø33A
ØØØ1           SCREEN=1
Ø33A
Ø33A A2 8Ø                LDX   #$8Ø
Ø33C 86 Ø2                STX   SCREEN+1
Ø33E A9 ØØ                LDA   #$ØØ
Ø34Ø 85 Ø1                STA   SCREEN
Ø342 CA        LOOPA      DEX
Ø343 AØ ØØ                LDY   #ØØ
Ø345 B1 Ø1     LOOPB      LDA   (SCREEN),Y
Ø347 49 8Ø                EOR   #$8Ø
Ø349 91 Ø1                STA   (SCREEN),Y
Ø34B C8                   INY
Ø34C DØ F7                BNE   LOOPB
Ø34E E6 Ø2                INC   Ø2
Ø35Ø EØ 7C                CPX   #$7C
Ø352 DØ EE                BNE   LOOPA
Ø354 6Ø                   RTS
Ø355                      .END
```

The code resides in the PET's second cassette buffer, but it is relocatable. The routine may be called using a SYS(826) statement, and may be easily incorporated into a BASIC program using the following loader.

```
1ØØ   FOR I=826 TO 852:READ J:POKE
      I,J:NEXT I
11Ø   DATA 162,128,134,2,169,Ø,133,1,
      2Ø2,16Ø,Ø,177,1,73,128
12Ø   DATA 145,1,2ØØ,2Ø8,247,23Ø,2,
      224,124,2Ø8,238,96
```

Having POKEd about inside the video RAM and moved things around the screen, it might seem that all has been covered. This is not so, for, if your computer has cursor control, you have an alternative method for creating the illusion of movement. Indeed, some VDUs only have this method available. 'If you have a choice, however, you might well be asking why you need to bother with a second method. The answer to that question is in two parts:

i) Cursor control can be quicker than POKEing — and this is important when a large number of characters have to be moved.

ii) It's easier to assemble a cursor controlled PRINT statement because you may be able to use the keyboard graphics symbols directly. There's no need to calculate all the correct ASCII or POKE numbers.

A Cursor String

Cursor control characters need not always be contained in quotes in PRINT statements. It's easy to build up a string which contains the necessary 'ups', 'downs' or 'sideways'. Perhaps the simplest example is as follows:

```
1Ø   A$="*[CL][SPC]"
2Ø   FOR I=1 TO 6
3Ø   A$=A$+A$
4Ø   NEXT I
5Ø   PRINT A$
```

Note that line 30 doubles the length of the string every time it is executed and the final string is 192 characters long. When A$ is PRINTed the sequence is as follows:

i) Print an asterisk.

ii) Move the cursor one space to the left so that the next character will be printed over the asterisk.

iii) Print a space, thus removing the asterisk.

▶

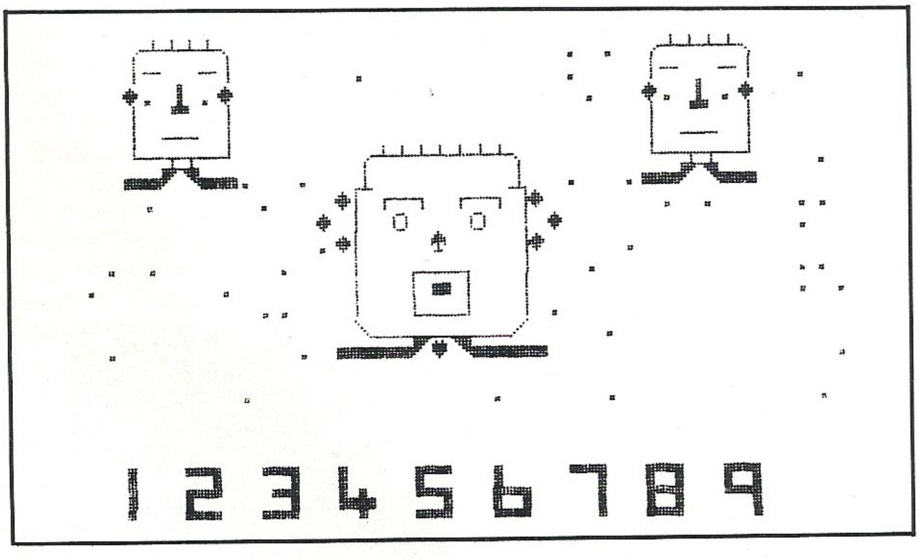


Fig. 3. An open and shut case.

```
1100  REM ** CLEAR SCREEN, SET UP
1120  REM ** CURSOR STRINGS AND
1140  REM ** PRINT THE BACKGROUND
1160  PRINT"[CLS]":LZ=20
1180  FOR I=1 TO 50:POKE 32768+INT(700*RND(1)),46:
      NEXT I
1200  P$="[HOM][5 CD][13 CR]"
1220  C$="[CD][11 CL]":D$="[CD][6 CL]":GOSUB 1640
1240  PRINT"[HOM][2 CR]"+E$:PRINT"[HOM][29 CR]"+E$
1260  REM ** GET A CHARACTER
1280  PRINT P$+A$
1300  GET Q$:IF Q$="" THEN 1280
1320  REM ** CHECK FOR A DIGIT
1340  IF ASC(Q$)<48 OR ASC(Q$)>57 THEN 1280
1360  REM ** WORK THROUGH DIGITS
1380  FOR XZ=0 TO VAL(Q$)
1400  POKE 0,120:POKE 1,255-11*VAL(Q$)
1420  IF XZ<1 AND VAL(Q$)=0 THEN GOSUB 1580:GOTO 1500
1440  IF XZ<1 AND VAL(Q$)>0 THEN 1500
1460  REM ** OPEN MOUTH AND MAKE A NOISE
1480  PRINT P$+B$:GOSUB 1580:SYS 826:PRINT P$+A$
1500  NEXT XZ:PRINT P$+A$:FOR I=1 TO 500:NEXT I
1520  REM ** CLEAR THE NUMBERS
1540  PRINT "[3 CD]":FOR I=1 TO 20:PRINT "[6 SPC]";:
      NEXT I:POKE 158,0:GOTO 1300
1560  REM ** PLOTTING ROUTINE
1580  PZ=18:IF VAL(Q$)>1 THEN PZ=4*XZ+15-2*VAL(Q$)
1600  PRINT LEFT$(LZ$,LZ+1);TAB(PZ);NZ$(XZ):RETURN
1620  REM ** A$=FACE WITH SHUT MOUTH
1640  A$=A$+"[SPC][^U][7^1][^I][SPC]"+C$
1660  A$=A$+"[SPC][^]][7 SPC][^]][SPC]"+C$
1680  A$=A$+"[^]][^O][^0][[^@][^.][SPC][^0][^@][^.][^P]
      [^]]"+C$+"[CL]"
1700  A$=A$+"[^C][^Z][^%][SPC][^Q][3 SPC][^Q][SPC][^']
      [^Z][^C]"+C$+"[CL]"
1720  A$=A$+"[^]][^%][3 SPC][^]][3 SPC][^'][^]]"+C$
1740  A$=A$+"[SPC][^%][3 SPC][^A][3 SPC][^'][SPC]"+C$
1760  A$=A$+"[SPC][^%][2 SPC][3^"][2 SPC][^'][SPC]"+C$
1780  A$=A$+"[SPC][^L][7^$][^:][SPC]"+C$
1800  A$=A$+"[4 SPC][^]][SPC][^]][4 SPC]"+C$
1820  A$=A$+"[4^"][^)][^S][^←][4^"]"+C$
1840  REM ** B$=FACE WITH OPEN MOUTH
1860  B$=B$+"[SPC][^U][7^1][^I][SPC]"+C$
1880  B$=B$+"[SPC][^]][7 SPC][^]][SPC]"+C$
1900  B$=B$+"[^Z][^O][^0][[^@][^.][SPC][^0][^@][^.][^P]
      [^Z]"+C$+"[CL]"
1920  B$=B$+"[^Z][SPC][^%][SPC][^W][3 SPC][^W][SPC][^']
      [SPC][^Z]"+C$+"[CL]"
1940  B$=B$+"[^Z][^%][3 SPC][^A][3 SPC][^'][^Z]"+C$
1960  B$=B$+"[SPC][^%][2 SPC][3^$][2 SPC][^'][SPC]"+C$
1980  B$=B$+"[SPC][^%][2 SPC][^%][^"][^'][2 SPC][^']
      [SPC]"+C$
2000  B$=B$+"[SPC][^%][2 SPC][^L][^$][^:][2 SPC][^']
      [SPC]"+C$
2020  B$=B$+"[SPC][^M][7^$][^N][SPC]"+C$
2040  B$=B$+"[4^"][^)][^S][^←][4^"]"+C$
2060  REM ** E$=BACKGROUND FACE
2080  E$=E$+"[^U][4^1][^I]"+D$
2100  E$=E$+"[^]][^@][2 SPC][^@][^]]"+D$
2120  E$=E$+"[^Z].[^'][^%].[^Z]"+D$
2140  E$=E$+"[^]][SPC][^<][^>][SPC][^]]"+D$
2160  E$=E$+"[^]][SPC][2^@][SPC][^]]"+D$
2180  E$=E$+"[^-][^@][2^2][^@][^=]"+D$
2200  E$=E$+"[2^"][^)][^←][2^"]"+D$
2220  REM ** MACHINE CODE ROUTINE
2240  REM ** FOR THE SOUNDBOX
2260  POKE 59459,255
2280  FOR HB=826 TO 870
2300  READ B:POKE HB,B:NEXT HB
2320  DATA 165,1,162,215,142,64,232,170
2340  DATA 202,208,253,240,0,240,0,240,0
2360  DATA 240,0,240,0,162,223,142,64,232
2380  DATA 170,202,208,253,198,0,208,5
2400  DATA 234,234,234,234,96,240,0
2420  DATA 240,0,208,213
2440  REM ** SET NUMBERS
2460  LZ$="[HOM][23 CD]"
2480  NZ$(0)="[REV][^,][^;][OFF][CD][2 CL][^!][REV][^!]
      [OFF][CD][2 CL][REV][2^"][OFF]":NZ$(1)="[SPC][^!]
      [CD][2 CL][SPC][^!][CD][2 CL][SPC][^>]"
2500  NZ$(2)="[REV][^"][^;][OFF][CD][2 CL][REV][^,][^"]
      [OFF][CD][2 CL][REV][2^"][OFF]":NZ$(3)="[REV][^"]
      [^;][OFF][CD][2 CL][^<][REV][^;][OFF][CD][2 CL]
      [REV][2^"][OFF]"
2520  NZ$(4)="[^!][SPC][CD][2 CL][REV][2^<][OFF][CD]
      [2 CL][SPC][^>]":NZ$(5)="[REV][^,][^"][OFF][CD]
      [2 CL][REV][^"][^;][OFF][CD][2 CL][REV][2^"][OFF]"
2540  NZ$(6)="[^!][SPC][CD][2 CL][REV][^,][^;][OFF][CD]
      [2 CL][REV][2^"][OFF]":NZ$(7)="[REV][^"][^;][OFF]
      [CD][2 CL][SPC][REV][^!][OFF][CD][2 CL][OFF][SPC]
      [^<]"
```

The Robot listing.

iv) Repeat the above steps until the end of the string.

Animation

So far we have always restricted ourselves to moving the odd one or two characters on the screen. This often leads to an impression of movement, but animation requires that we move large blocks of characters simultaneously. After all, solid objects move as a whole, not one piece at a time.

If some of my examples make you wonder whether I'm in my second childhood, perhaps I should explain that I have a three year old son who considers Daddy's 'blooter' the best toy he's ever seen. He'll only leave me to work (play?) in peace if he gets his fair share of button pushing. I found I could either curse or cursor!

Figure 3 shows two screen prints from one of his programs. He pushes a number, the robot opens its mouth, burps the required number of times (the program uses a Petsoft soundbox) and for every burp,displays the correct digit. The reason for its inclusion in this article is that both the robot animation and the digits are produced under cursor control. The only POKEing required is for the starry background on which the robots appear. Apart from being enjoyable to watch, it has also been very effective in teaching him the digits 0 to 9.

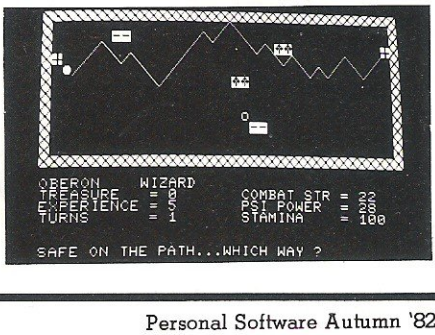
In the program, A$ holds a string of characters which print the robot with its mouth shut and B$ holds the string for the mouth open picture. By printing these alternately in quick succession it gives the appearance of a talking robot. The place at which the robot is printed is also governed by cursor control characters, the string variable P$, which when printed, positions the cursor at the correct point. Similarly, line 1600 uses a chunk of LZ$ to find the correct line on which to print the numbers. This number printing routine may be extracted and used as a subroutine in other programs.

Putting It Together

So far we have looked mainly at techniques and how they may be used. You will not normally be writing a program to illustrate a technique, you are much more likely to be interested in how you might best implement some bright idea. Let's look at a simple game and how it might be programmed.

Consider a railway yard with three sidings. What is the best way to sort a set of goods trucks into order using those sidings? As most of us don't have a goods yard, can we write a program to simulate the problem?

Obviously, we're not going to be satisfied with just the printout of an answer — where's the fun in that? What we want is to see the engine chugging backwards and



```
OBERON      WIZARD
TREASURE    = 0         COMBAT STR = 22
EXPERIENCE  = 5         PSI POWER  = 28
TURNS       = 1         STAMINA    = 100

SAFE ON THE PATH...WHICH WAY ?
```

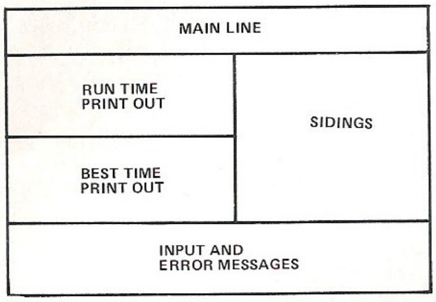


Fig. 4. The general screen layout.

forwards,depositing and collecting trucks. If we want to find the best way, we must have some method of making comparisons of different solutions. Where do we start?

The first thing to consider is the general screen layout. Figure 4 shows a possible solution.

Now, how are we going to simulate train movement? Cursor control is difficult because the pattern to be printed will vary with every shunting action. This leaves POKEing — but how do we know where and what to POKE?

I'll work through the program a bit at a time to show how the two methods may be mixed to produce a final working program.

```
1200  DIM P1(55),P2(55),P3(55)
1220  SP=32768:LL=40
1240  GOSUB 3340:IN$="[3 SPC].[3 CL]:
      BT$="999999"
1260  REM ** SET UP CURSOR
1280  REM ** CONTROL STRINGS
1300  CD$="[HOM]":FOR J=1 TO 40
1320  CD$=CD$+"[CD]":CR$=CR$+"[CR]"
1340  CU$=CU$+"[CU]":CL$=CL$+"[CL]"
1360  BL$=BL$+"[2 SPC]"
1380  NEXT J
1400  BL$=BL$+CL$+CL$
1420  PRINT "[7 SPC][REV]PRESS A KEY
      TO CONTINUE[OFF]"
1440  GET A$:IF A$="" THEN 1440
1460  PRINT "[CLS]";:POKE 59468,12
```

The main function of this portion of the program is to set up the cursor control strings. The four C?$ strings are filled with sets of cursor control characters for each of the four directions. We can then position the cursor using these strings and the string functions. To move the cursor to the 20th position along the 10th line down, for example, would require that we PRINT LEFT$(CD$,10);LEFT$(CR$,19). IN$ is a string used to position a dot under the cursor when using INPUT and BL$ is a string containing 80 blanks and 80 cursor lefts which is used to clear garbage from two lines of the screen without altering the cursor position.

The rest of the coding is fairly standard. GOSUB 3340 calls the instructions and lines 1420 and 1440 halve the program while we read. BT$ is used in the clock routine to hold the 'Best Time' and is set initially to a false value.

```
1480  REM ** SET UP THE MAIN LINE
1500  FOR J=0 TO 39
1520  P1(J)=SP+J:P2(J)=SP+J:
      P3(J)=SP+J
1540  POKE P1(J),61
1560  NEXT J
1580  REM ** SET UP AND POKE SIDINGS
1600  FOR J=1 TO 16
1620  P1(J+19)=SP+19+LL*J
1640  POKE P1(J+19),34
1660  P2(J+29)=SP+29+LL*J
1680  POKE P2(J+29),34
1700  P3(J+39)=SP+39+LL*J
1720  POKE P3(J+39),34
1740  NEXT J
1760  REM ** POKE THE SIDINGS
1780  POKE P1(35)+2*LL,177
1800  POKE P2(45)+2*LL,178
1820  POKE P3(55)+2*LL,179
```

The screen POKE numbers of the main line and sidings are held in arrays, one for each siding. This section of the program sets up those arrays and POKEs the lines onto the screen. Note that if your line length is less than 40 characters you will have to change line 1500, and the numbers in lines 1780 to 1820 are peculiar to the PET.

```
1840  REM ** SET UP INITIAL
      CONDITIONS
1860  T1=35:S1=0:T2=45:S2=0:T3=55:
      S3=0:LT=15
1880  TI$="000000":PRINT LEFT$(CD$,8)
      ;"[REV]RUN TIME[OFF]"
1900  PRINT "[HOM]=[REV][^)][^;][OFF]
      [^:]-[REV]FEHACDBKJGI"
1920  REM ** INPUT ROUTINE
1940  PRINT LEFT$(CD$,21);BL$+"SIDING
      "+IN$;:INPUT S
1960  IF S<1 OR S>3 THEN PRINT "[CU]
      [REV]";:GOTO 1940
1980  PRINT LEFT$(CD$,22);BL$+"NUMBER
      "+IN$;:INPUT SX
2000  IF LT-SX<4 THEN PRINT "[CU]
      [REV]";:GOTO 1980
2020  ON S GOSUB 2340,2680,3020
2040  REM ** RUN-TIME ROUTINE
2060  ST$=TI$
2080  PRINT LEFT$(CD$,10);MID$(ST$,3,
      2);" MINS ";MID$(ST$,5,2);
      " SECS"
2100  FOR I=1 TO 11:IF PEEK(32772+I)
      <>128+I THEN I=12:NEXT I:
      GOTO 1940
2120  NEXT I
```

Here we have the guts of the program. Moves are input and checked, the appropriate subroutine is selected and the running time is updated at the end of each move. The run time could be continuously updated but this slows down the train movement too much.

```
2140  REM ** BEST TIME ROUTINE
2160  IF ST$<BT$ THEN BT$=ST$
2180  PRINT LEFT$(CD$,14);"[REV]BEST
      TIME[CD]"
2200  PRINT MID$(BT$,3,2);" MINS ";
      MID$(BT$,5,2);" SECS"
2220  PRINT LEFT$(CD$,21);BL$
2240  PRINT "[CU][REV]ANOTHER GO?
      [OFF]":FOR I=1 TO 100:NEXT I
2260  GET A$:IF A$="" THEN PRINT
      "[CU]ANOTHER GO?":FOR I=1 TO
      100:NEXT I:GOTO 2240
2280  IF A$="Y" THEN TI$="000000":
      GOTO 1900
2300  STOP
```

Once the train has been properly sorted, this routine checks whether or not the previous best time has been beaten. The 'Another Go' routine shows how cursor control may be used to flash the question on and off using reverse video. The FOR . . .NEXT loops in this part of the program are for timing purposes.

```
2320  REM ** PUT 1
2340  IF S1+SX<0 THEN PRINT "[CU]
      [REV]";:GOTO 1980
2360  FOR J=1 TO T1-LT-S1
2380  FOR K=J+LT TO J STEP-1
2400  POKE P1(K),PEEK(P1(K-1))
2420  NEXT K
2440  NEXT J
2460  S1=S1+SX:LT=LT-SX
2480  REM ** TAKE 1
2500  FOR J=T1-LT-S1 TO 1 STEP-1
2520  FOR K=J TO J+LT-1
2540  POKE P1(K),PEEK(P1(K+1))
2560  NEXT K
2580  IF P1(K)>32808 THEN POKE P1(K),
      34:GOTO 2620
2600  POKE P1(K),61
2620  NEXT J
2640  RETURN
```

This subroutine moves the train to and from siding one. Line 2340 is testing for a legitimate move as trying to remove non-existent trucks could result in one of the sidings disappearing completely! The movement is produced by the caterpillar method described earlier.

```
2660  REM ** PUT 2
2680  IF S2+SX<0 THEN PRINT "[CU]
      [REV]";:GOTO 1980
2700  FOR J=1 TO T2-LT-S2
2720  FOR K=J+LT TO J STEP-1
2740  POKE P2(K),PEEK(P2(K-1))
2760  NEXT K
2780  NEXT J
2800  S2=S2+SX:LT=LT-SX
2820  REM ** TAKE 2
2840  FOR J=T2-LT-S2 TO 1 STEP-1
2860  FOR K=J TO J+LT-1
2880  POKE P2(K),PEEK(P2(K+1))
2900  NEXT K
2920  IF P2(K)>32808 THEN POKE P2(K)
      34:GOTO 2960
2940  POKE P2(K),61
2960  NEXT J
2980  RETURN
3000  REM ** PUT 3
3020  IF S3+SX<0 THEN PRINT "[CU]
      [REV]";:GOTO 1980
3040  FOR J=1 TO T3-LT-S3
3060  FOR K=J+LT TO J STEP-1
3080  POKE P3(K),PEEK(P3(K-1))
3100  NEXT K
3120  NEXT J
3140  S3=S3+SX:LT=LT-SX
3160  REM ** TAKE 3
3180  FOR J=T3-LT-S3 TO 1 STEP-1
3200  FOR K=J TO J+LT-1
3220  POKE P3(K),PEEK(P3(K+1))
```

▶

```
3240  NEXT K
3260  IF P3(K)>32808 THEN POKE P3(K),
      34:GOTO 3300
3280  POKE P3(K),61
3300  NEXT J
3320  RETURN
```

This section is similar to the one above but is for the other two sidings.

```
3340  REM ** INSTRUCTIONS
3360  POKE 59468,14:PRINT "[CLS]";TAB
      (15);"[REV]SHUNTING[OFF][3 CD]"
3380  PRINT "[2 SPC]SHUNTING IS A
      RAILWAY SIMULATION GAME"
3400  PRINT "WHERE YOU HAVE TO SHUNT
      A SET OF GOODS"
3420  PRINT "WAGONS INTO ORDER[3 CD]"
3440  PRINT "YOU MUST SPECIFY A
      SIDING (1-3) AND"
3460  PRINT "THE NUMBER OF WAGONS TO
      BE MOVED. IF YOU";
3480  PRINT "TYPE A POSITIVE NUMBER
      WAGONS WILL BE"
3500  PRINT "ADDED TO THE SIDING, A
      NEGATIVE NUMBER"
3520  PRINT "REMOVES THEM[3 CD]"
3540  PRINT "[2 SPC]THE AIM IS TO
      SORT THE TRAIN IN THE"
3560  PRINT "SHORTEST POSSIBLE TIME
      [3 CD]":RETURN
```

Here are the instructions. Although they appear every time the program is run, they do serve the purpose of having something on the screen while the setting up is taking place.

As we progress deeper into the graphics jungle so we move further away from any pretence at common standards. To write a general article on PEEK and POKE is relatively easy because most modern micros have a memory mapped display and their BASICs support these statements. Cursor control is more difficult because not all machines have it, and those that do have different methods of implementing it. We are now going to look at the actual characters which a micro may display and this depends not only on the hardware and software, but also on the manufacturer's philosophy towards graphics.

Shades Of Definition

Let's start by considering each character position on the screen as a rectangle which may be either on (white) or off (black). On the RML 380Z this would give us a basic resolution of 40 across by 24 down, on the TRS-80 it would be 64 by 16 and on the PET it would be 40 by 25. If we only had this definition to work with, all pictures would be very crude and difficult to decipher. However, each character position is itself made up of a matrix of dots. The size of this matrix varies from machine to machine but let's take the RML 380Z standard of six dots wide by nine dots high as an example. If we could switch each of these dots on and off individually, our resolution would leap from 40 by 24 to 240 by 216 and we would have what is know as high resolution graphics. The snag is that you require more memory and additional hardware.

Manufacturers have solved this problem in a variety of ways, but most use the fact that normal characters (ABC . . ., abc . . .,/*+ − . . .,etc) need only half of the 256 combinations available in a single eight-bit byte. They use the remaining codes to define new characters which may be specailly designed à la PET and Sharp MZ-80K, or chunky for use on the TRS-80 and RML 380Z.

Pixel Characters

The chunky graphics referred to above are known as Pixel characters and this type of graphics is similar to that used in Teletext transmissions on BBC and ITV. Each character is about three times as high as it is wide and includes six blocks, each of which may be thought of as having a specific value. Each character has an ASCII code and these are all allocated as if the six positions had values 1,2,4,8,16 and 32 as shown in Fig. 5. Using this method we can consider the TRS-80 screen as an 128 by 48 grid, and the RML 380Z screen as an 80 by 72 grid; both machines have statements which allow you to switch individual pixels 'on' or 'off'. However, these statements differ from machine to machine, and each of the manufacturers has numbered the screen in a different way. The TRS-80 uses SET and RESET with the grid numbered across and down, RML 380Z uses PLOT with the grid numbered across and up. By way of an explanation here are two programs, one for each machine, which produce an ever-changing pattern over the complete screen.

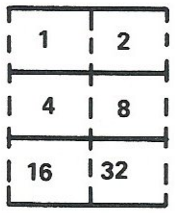


Fig. 5. How the pixel character can be encoded.

```
10  REM ** TRS-80
15  CLS
20  X=RND(128)-1
25  Y=RND(48)-1
30  SET(X,Y)
35  X=RND(128)-1
40  Y=RND(48)-1
45  RESET(X,Y)
50  GOTO 20
```

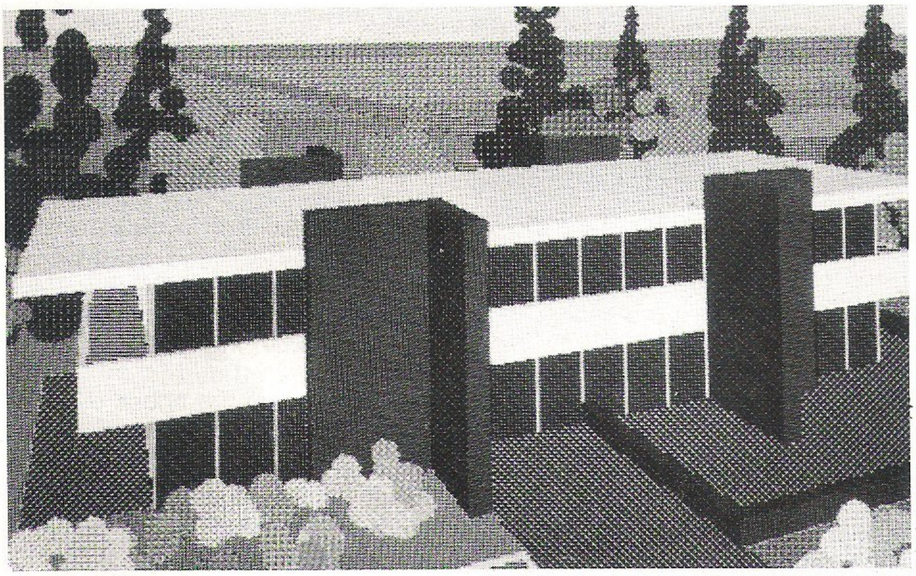
The X and Y coordinates are selected randomly using the TRS-80's random number generator, which is able to select integers within a given range. SET (X,Y) switches the required pixel 'on' and RESET (X,Y) switches the pixel 'off'.

```
10  REM ** RML 380Z
15  GRAPH1:PRINT CHR$(12)
20  X=80*RND(1)
25  Y=60*RND(1)
30  PLOT X,Y,2
35  X=80*RND(1)
40  Y=60*RND(1)
45  PLOT X,Y,0
50  GOTO 20
```

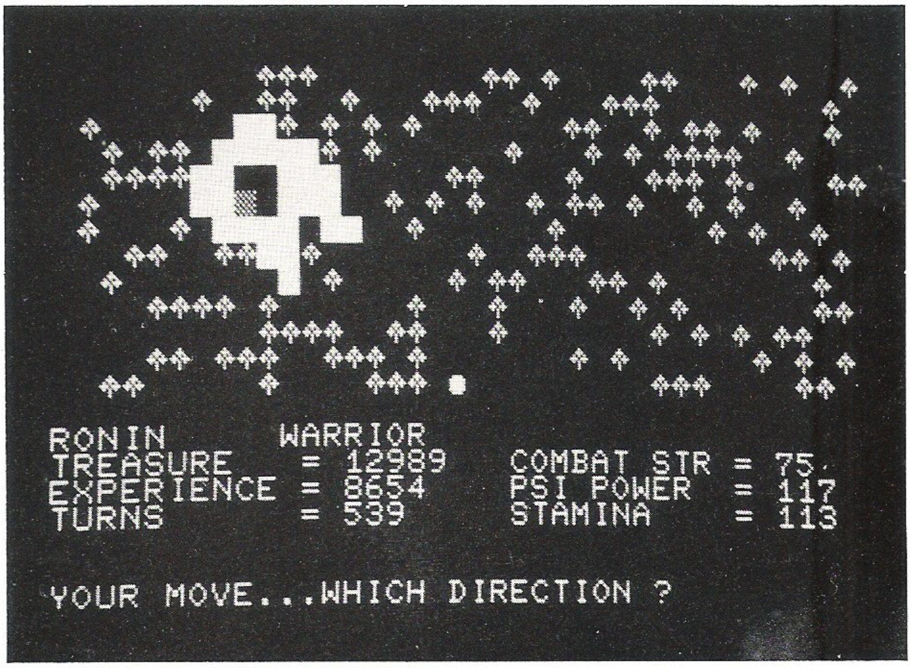
The GRAPH 1 statement switches on the graphics 'window' of the RML 380Z, which does not cover the complete area of the screen. This is why 60, rather than 72, is required in lines 20 and 40. The machine also has the capability of plotting both grey and white pixels; all that is required is a change from 2 to 1 in line 30 (ie 0 for off, 1 for grey and 2 for white).

Shape Reduction

The SET or PLOT statements are fine for producing graphs, but the method becomes tedious if large shapes are required on the screen. However, it is possible to save time and energy by printing the ASCII character which corresponds to a given 3 by 2 shape. Let's imagine that we wish to print a reduced version of the domino shown in Fig. 6. You will see that the grid has a 3 by 2 pattern marked over it, and the top left-hand portion of the domino has the shape shown in Fig 7. The total of the 'on' squares is 23 and the pixel

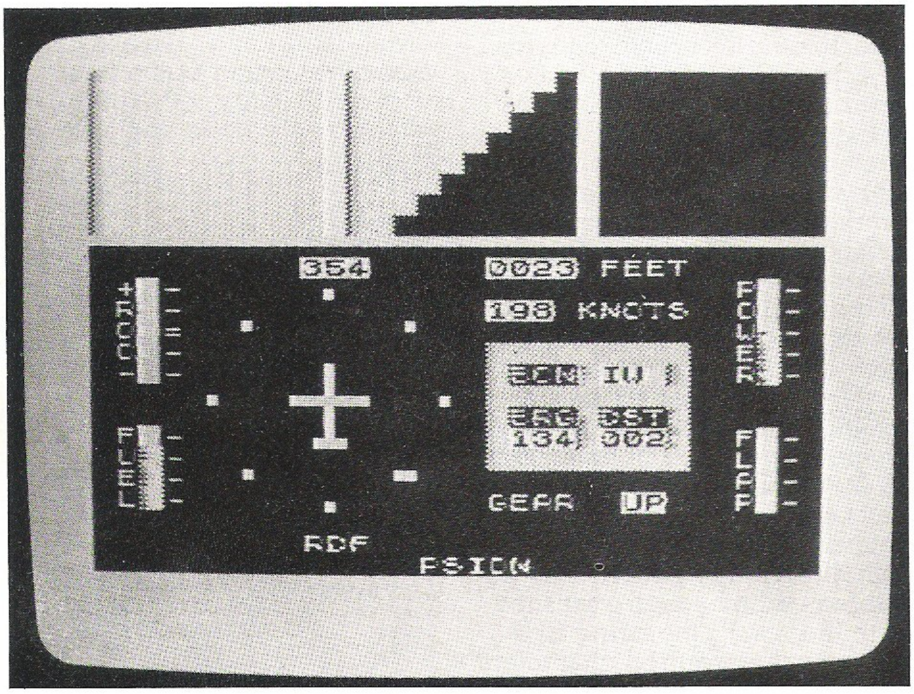


Left: This graphic display was produced by the Hewlett-Packard 9845 system shown on the front cover of this magazine. The computer first displays the bare surface and background, then builds the walls and places a roof on the building. The three-dimensional simulation can then be rotated as if you were actually walking around the building.
Photographic facilities courtesy of Hewlett-Packard.



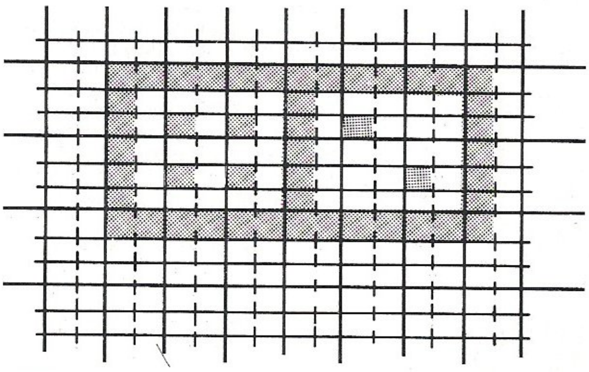
```
RONIN        WARRIOR
TREASURE    = 12989      COMBAT STR = 75
EXPERIENCE  = 8654       PSI POWER  = 117
TURNS       = 539        STAMINA    = 113

YOUR MOVE...WHICH DIRECTION ?
```

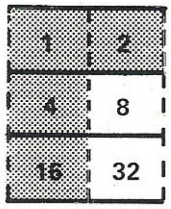
Right: Here we have Ronin the Warrior about to enter the woods surrounding the dread Vounim's Lair. Will he manage to find the Helm of Evanna before being zapped by a Ring Wraith or eaten by the Kraken? The Valley is a program originally published in our sister magazine, Computing Today, April 1982, and is now sold on cassette for many popular micros by CT Software.



Left: This is a screen display of the Sinclair ZX81 Flight Simulation program from Psion. This real time simulation provides the parameters of flight displaying a mock instrument panel with navigation instructions as well as a view of where your 'plane is at any time. Taking off is relatively easy, but landing . . .

▶





▲ **Fig. 7. One segment showing the pixel value.**

◀ **Fig. 6. A double domino generated from pixel characters.**

graphics have ASCII codes starting at 128. The ASCII code for our character is 128 + 23 = 151, and therefore, the statement PRINT CHR$ (151) will print it on the screen at the current cursor position.

Pseudo-Chunkies

As stated earlier, not all machines have graphics of this type, but it is often possible to write a routine to accomplish the same function. Providing the machine has a complete set of quarter square graphics, it is possible to PEEK the screen to see what is already there and then POKE back the updated character. This is possible with the PET and the technique is usually referred to as double density graphics.

Being, by nature, a lazy person, I searched for an easy way to incorporate double density shapes into my programs. The following program allows me to design a shape using full size blocks and then, when I press Return, it automatically produces a string (SH$) which represents the half-size picture.

```
100  REM ** SHAPE REDUCER
110  REM ** FOR HALF SIZE PICTURES
120  DIM SH(9,11),SY$(15)
130  CD$="[HOM][15 CD]":CR$=
     "[25 CR]"
140  FOR I=0 TO 15:READ SY$(I):
     NEXT I
150  DATA "[SPC]",[^>]","[^<]",
     "[REV][^"][OFF]","[^;]","[^!]",
     "[REV][^?][OFF]","[REV][^,]
     [OFF]"
160  DATA "[^,]","[^?]","[REV][^!]
     [OFF]","[REV][^;][OFF]","[^"]",
     "[REV][^<][OFF]","[REV][^>]
     [OFF]","[REV][SPC][OFF]"
170  L=0:M=0
180  PRINT "[CLS]";RT$;"[20^&]"
190  FOR I=1 TO 10
200  PRINT RT$;"[4^&][12 SPC][4^&]"
210  NEXT I
220  PRINT RT$;"[20^&]"
230  GOTO 360
240  PRINT "[SPC][CL]";:FOR I= 1 TO
     50:GET A$:IF A$<>"" THEN 270
250  NEXT I:PRINT "[REV][SPC][OFF]
     [CL]";:FOR I=1 TO 50:GET A$:IF
     A$<>"" THEN 270
260  NEXT I:GOTO 240
270  IF SH(L,M)=0 THEN PRINT "[SPC]
     [CL]";
280  IF SH(L,M)=1 THEN PRINT "[REV]
     [SPC][OFF][CL]";
290  IF A$=CHR$(13) THEN 480
300  IF A$="[SPC]" OR A$="[REV]"
     THEN 380
310  IF A$="[CR]" THEN M=M+1
320  IF A$="[CL]" THEN M=M-1
330  IF A$="[CU]" THEN L=L-1
340  IF A$="[CD]" THEN L=L+1
350  GOSUB 430
360  PRINT LEFT$(CD$,L+2);LEFT$(CR$,
     M+4);
370  GOTO 240
380  IF A$="[SPC]" THEN PRINT
     "[SPC]";:SH(L,M)=0:M=M+1
390  IF A$="[REV]" THEN PRINT
     "[REV][SPC][OFF]";:SH(L,M)=1:
     M=M+1
400  GOSUB 430:PRINT LEFT$(CD$,L+2);
     LEFT$(CR$,M+4);:GOTO 240
410  REM ** ADJUST POSITION
420  REM ** OF IMAGE
430  IF M<0 THEN M=11:L=L-1:IF L<0
     THEN L=9
440  IF M>11 THEN M=0:L=L+1:IF L>9
     THEN L=0
450  IF L<0 THEN L=9:M=M-1:IF M<0
     THEN M=11
460  IF L>9 THEN L=0:M=M+1:IF M>11
     THEN M=0
470  RETURN
480  SH$="":FOR L1=0 TO 8 STEP 2:
     FOR M1=0 TO 10 STEP 2
490  VX=SH(L1,M1)+2*SH(L1,M1+1)+4*
     SH(L1+1,M1)+8*SH(L1+1,M1+1):
     SH$=SH$+SY$(VX)
500  NEXT M1:SH$=SH$+"[CD][6 CL]"
510  NEXT L1:SH$=SH$+"[2 CU]"
520  PRINT "[HOM]";TAB(25);SH$;
     "[11 CD]"
530  GOTO 360
```

The 16 quarter square patterns are stored in SY$ and READ from DATA statements in lines 150 and 160. Lines 240 to 260 are an INPUT routine which shows the position of the cursor on the screen, and the cursor position may be altered using the usual cursor control buttons. The RVS button will produce a white square and the Space bar provides a black square.

The conversion routine, which reduces the size of the shape, takes place in lines 480 to 510. Once the reduced shape has been printed, control returns to the main program so that the original pattern may be altered. When you are satisfied with the result, the string SH$ contains the required characters and may be inserted in another program.

A Final Breakthrough

Well, if you've managed to get this far with this article, you are more than likely ready for a bit of relaxation. So the final program is designed to show how all we have covered so far may be put together to form a complete working program, in this case the game of Breakthrough. For those of you who are unfamiliar with it, the game consists of bouncing a ball off a bat so that it rebounds to knock pieces out of a barrier. Your score increases with each piece removed, and if you obtain enough points within the time limit, you win a replay.

When I started to experiment with the component subroutines for the program, it soon became clear that a version written entirely in BASIC would be far too slow. So I looked for a frequently used routine which could be easily translated into machine code. I wanted this section to be self-contained, as access to variables used in the BASIC part of the program would be difficult. I finally chose the bat moving routine, for it is called more often than any other and is almost independent from the rest of the coding. It also had the advantage that it could be tested without the BASIC program, thus speeding up the usual debugging. Here is 6502 assembler listing of the final version:

```
033A               BAT MOVE ROUTINE
033A
033A A5 97                   LDA  151
033C C9 29                   CMP  #41
033E F0 07                   BEQ  VAL1
0340 C9 2A                   CMP  #42
0342 F0 10                   BEQ  VAL2
0344 4C 5E 03                JMP  PLOT
0347 AD 7B 03      VAL1      LDA  POSIT
034A C9 23                   CMP  #35
034C B0 10                   BCS  PLOT
034E EE 7B 03                INC  POSIT
0351 4C 5E 03                JMP  PLOT
0354 AD 7B 03      VAL2      LDA  POSIT
0357 C9 02                   CMP  #2
0359 90 03                   BCC  PLOT
035B CE 7B 03                DEC  POSIT
035E 20 70 03      PLOT      JSR  BLANK
0361 AE 7B 03                LDX  POSIT
0364 A0 04                   LDY  #4
0366 A9 E2                   LDA  #226
0368 9D 98 83      BAT       STA  SCREEN,X
036B E8                      INX
036C 88                      DEY
036D D0 F9                   BNE  BAT
036F 60                      RTS
0370               BLANK A BLOCK
0370 A2 26         BLANK     LDX  #38
0372 A9 20                   LDA  #32
```

```
Ø374 9D 98 83     NEXT1 STA  33688,X
Ø377 CA                 DEX
Ø378 DØ FA              BNE  NEXT1
Ø37A 6Ø                 RTS
Ø37B                    POSIT=*
8398                    SCREEN=33688
Ø37B                    .END
```

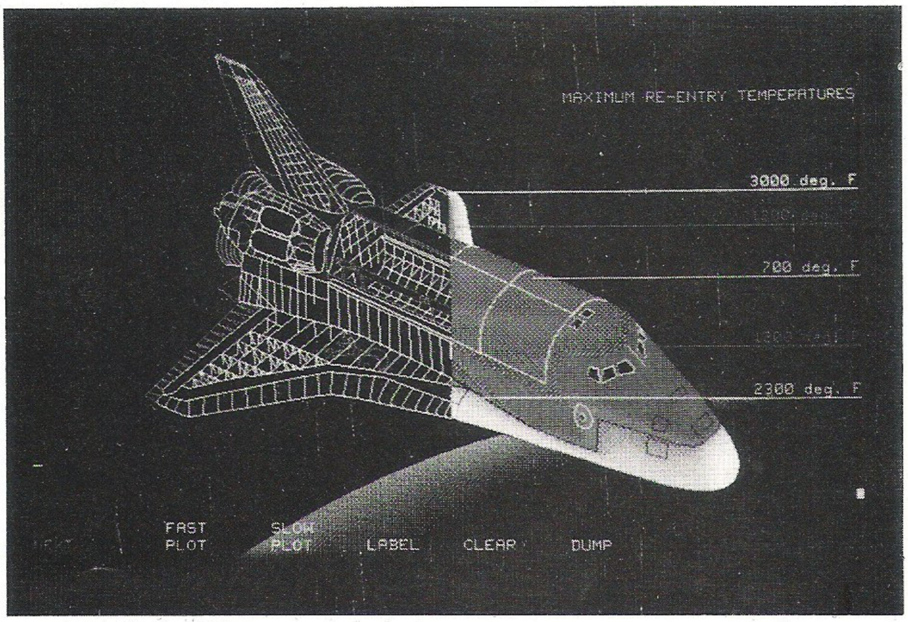
The Hex coding was then changed into decimal and incorporated into the BASIC program as DATA statements. When the program is run, it loads the routine into the PET's second cassette buffer and calls it with the SYS(826) statement. Below is a complete listing of the final program with the machine code routine starting in line 850.

I hope that the REMark statements will enable you to follow the program, but here is a general description. The ball is moved under POKE control and variable S holds the screen address position it will move to. The move is make by POKEing a ball symbol (POKE code = 81) to location S and a space (POKE code = 32) to the current position.

The information about the current state of play is found by PEEKing the screen location S. The values obtained are tested in lines 510 to 540, and a jump is executed to the appropriate position.

The time elapsed, score and ball number are all printed onto the screen under cursor control. The instructions, the results routine and other messages also use this this method of display.

The program is fairly fast, with most of the time being spent in the loop from line 450 to 490. If you want to speed it up still further, change the last statement in line 490 to GOTO 480. The only adverse effect of this is that the clock will not be updated continuously.



Photographic facilities courtesy of Hewlett-Packard.

```
150  POKE 59468,14:PRINT "[CLS][7 SPC][REV]THIS IS THE
     BREAKTHROUGH GAME"
160  PRINT "[2 CD]THE OBJECT OF THE GAME IS TO KNOCK AS"
170  PRINT "MANY BRICKS FROM THE WALL AS POSSIBLE."
180  PRINT "[2 CD]TO DO THIS YOU MUST BOUNCE THE BALL
     OFF";
190  PRINT " THE BAT AT THE BOTTOM OF THE SCREEN."
200  PRINT "[2 CD]THERE IS A TIME LIMIT OF SEVEN
     MINUTES"
210  PRINT "FOR EACH GAME BUT YOU EARN A REPLAY IF"
220  PRINT "YOU SCORE MORE THAN 75Ø POINTS."
230  PRINT "[2 CD]TO MOVE THE BAT TO THE LEFT PRESS THE"
240  PRINT "4 KEY."
250  PRINT "[CD]TO MOVE THE BAT TO THE RIGHT PRESS THE"
260  PRINT "6 KEY."
270  GOSUB 87Ø:PRINT "[3 CD][8 SPC][REV]PRESS ANY KEY TO
     BEGIN.[OFF]";
280  GET A$:IF A$="" THEN 28Ø
290  REM ** SET UP SCREEN
300  PRINT "[CLS]";:S=33Ø5Ø+INT(RND(1)*37):TI$="ØØØØØØ":
     J=1:PO=Ø
310  POKE 59468,12:PRINT "[HOM][REV][4Ø^#][OFF]"
320  PRINT "[CD][39^&]"
330  PRINT "[REV][39^Z]"
340  PRINT "[REV][39^V]"
350  FOR M=328Ø8 TO 33728 STEP 4Ø:POKE M,229:POKE M+39,
     231:NEXT M
360  PRINT "[HOM][CD][29 CR]BALL # ";J
370  PRINT "[HOM][2 CD][15 CR]SCORE ";PO
380  M=INT(RND(1)*2):B=39:IF M=1 THEN B=41
390  POKE S,81:S=S+B:IF S>3281Ø THEN 44Ø
400  REM ** CHECK THE CORNERS
410  IF S=32768 THEN S=328Ø9:B=41:GOTO 39Ø
420  IF S=328Ø7 THEN S=32846:B=39:GOTO 39Ø
430  REM ** TIME ROUTINE
440  IF TI$>"ØØØ7ØØ" THEN 7ØØ
450  PRINT "[HOM][CD][CR]TIME ";MID$(TI$,4,1);":";
     RIGHT$(TI$,2)
460  REM ** MOVE THE BAT AND BALL
470  REM ** WHEN THE PATH IS CLEAR
480  SYS 826:IF S>33768 THEN 59Ø
490  IF PEEK(S)=32 THEN POKE S,81:POKE S-B,32:
     S=S+B:SYS 826:GOTO 45Ø
500  REM ** WHAT HAVE WE BUMPED INTO?
510  IF PEEK(S)=229 THEN 56Ø
520  IF PEEK(S)=231 THEN 57Ø
530  IF PEEK(S)=226 THEN 62Ø
540  IF PEEK(S)<>227 THEN 65Ø
550  S=S-B:POKE S,32:B=8Ø-ABS(B):S=S+B:GOTO 44Ø
560  S=S-B:POKE S,32:B=B+2:S=S+B:GOTO 44Ø
570  S=S-B:POKE S,32:B=B-2:S=S+B:GOTO 44Ø
580  REM ** BALL LOST ROUTINE
590  POKE (S-B),32:FOR Z=1 TO 5Ø:FOR Z1=1 TO 1Ø:
     NEXT Z1:SYS 826:NEXT Z
600  J=J+1:S=33Ø75+INT(RND(1)*5):GOTO 36Ø
610  REM ** BOUNCE BALL OFF BAT
620  S=S-B:POKE S,32:B=B-8Ø:S=S+B:GOTO 44Ø
630  REM ** UPDATE SCORE AND
640  REM ** DELETE TARGET
650  POKE (S-B),32:IF PEEK(S)=1Ø2 THEN PO=PO+5:
     IF B>Ø THEN B=B-8Ø:GOTO 67Ø
660  IF B<Ø THEN B=8Ø+B
670  PO=PO+5:IF PO>=75Ø THEN 7ØØ
680  POKE S,81:PRINT "[HOM][2 CD][15 CR]SCORE ";PO:
     S=S+B:GOTO 44Ø
690  REM ** RESULTS ROUTINE
700  TM=6Ø*VAL(LEFT$(TI$,4))+VAL(RIGHT$(TI$,2))
710  FOR M=32768 TO 33767:POKE M,16Ø:NEXT M
720  POKE 59468,14:PRINT "[CLS][CD]BALLS USED ";J
730  PRINT "[CD]TIME TAKEN ";TM;" SECONDS"
740  PRINT "[CD]SCORE IS ";PO
750  BF=INT(((PO+1ØØ)/J)*1Ø)/1Ø
760  PRINT "[CD]YOUR BREAKTHROUGH FACTOR IS ";BF
770  IF PO>=75Ø OR BF>2Ø THEN 83Ø
780  REM ** REPLAY ROUTINE
790  POKE 158,Ø:INPUT "[2 CD][REV]DO YOU WANT A REPLAY
     [OFF] ";A$
800  IF LEFT$(A$,1)="Y" THEN 3ØØ
810  IF LEFT$(A$,1)<>"N" THEN PRINT "[CD][REV]ANSWER 'Y'
     OR 'N'[5 CU]":GOTO 79Ø
820  POKE 58468,12:PRINT "[CLS][3 CD][3 SPC]THANKS FOR
     PLAYING":END
830  PRINT "[HOM][14 CD][11 CR][REV]YOU WIN A REPLAY"
840  FOR RR=Ø TO 3ØØØ:NEXT RR:GOTO 3ØØ
850  REM ** MACHINE CODE ROUTINE
860  REM ** TO MOVE THE BAT
870  FOR IT=Ø TO 65:READ DA:POKE 826+IT,DA:NEXT IT:
     RETURN
880  DATA 165,151,2Ø1,41,24Ø,7,2Ø1,42,24Ø,16,76,94
890  DATA 3,173,123,3,2Ø1,35,176,16,238,123,3,76
900  DATA 94,3,173,123,3,2Ø1,2,144,3,2Ø6,123,3
910  DATA 32,112,3,174,123,3,16Ø,4,169,226,157,152
920  DATA 131,232,136,2Ø8,249,96,162,38,169,32,157,152
930  DATA 131,2Ø2,2Ø8,25Ø,96,2Ø
```

The Breakthrough listing.

Subscriptions

Are you looking for a more personal approach to computing? You are . . . then Computing Today is the magazine for you! Packed full of feature articles, projects, general topics, news and reviews, Computing Today is aimed at readers who want to get more out of their microcomputer.

The latest ABC circulation figures show Computing Today has increased its readership by 85% over the previous year — great news for us at CT. However, the ever increasing demand for Computing Today has meant that, despite our printing more each month, some readers seem to be missing out on their regular copy.

If you would like to ensure a regular supply for the next twelve months, each issue lovingly wrapped and posted to you, nothing could be simpler. Just fill in the form below, cut it out and send it with your cheque or Postal Order (made payable to ASP Ltd) to:

Computing Today Subscriptions,
513 London Road,
Thornton Heath,
Surrey CR4 6AR.

Alternatively you can pay by Access or Barclaycard in which case simply fill in your card number, sign the form and send it off. Do NOT send your card.

Do yourself a favour, make 1982 the year you start to take Computing Today every month and we'll give you a truly

Personal Approach To Microcomputing.

SUBSCRIPTION ORDER FORM

Cut out and SEND TO :
Computing Today Subscriptions
513, LONDON ROAD,
THORNTON HEATH,
SURREY,
ENGLAND.

Please commence my personal subscription to Computing Today with the .... issue.

SUBSCRIPTION RATES
(tick □ as appropriate)

£12.10 for 12 issues
UK □
£15.75 for 12 issues
Overseas Surface □
£35.35 for 12 issues
Overseas Air Mail □

Autumn '82

I am enclosing my (delete as necessary)
Cheque/Postal Order/International Money
Order for £..........
(made payable to ASP Ltd)
OR
*Debit my Access/Barclaycard**
*(*delete as necessary)*

Access
We welcome Access

BARCLAYCARD
VISA

□□□□□□□□□□□□□□□□□□

Please use BLOCK CAPITALS and include post codes.

Name (Mr/Mrs/Miss) ....
delete accordingly

Address ....

....

....

Signature ....

Date ....

GAMES

NASCOM Life — The game of Life, as played by John Horton Conway . . .and other intellectuals!
Originally published in Computing Today, December 1979.

Moonbase Alert — PEEKing and POKEing can be fun as you try to stop the spaceship land on your planet.
Originally published in Computing Today, March 1980.

Snakes — Try to steer your snake home before its tail gets too long and trips you up.
Originally published in Computing Today, April 1981.

Holocaust — A tactical thermonuclear wargame you can fight out in your own living room.
Originally published in Computing Today, July 1981.

Ambush — First published as an electronic game project in our sister magazine, Electronics Today International, here is the software version.
Originally published in Computing Today, April 1979.

Ski Run — No chance of breaking a leg with this program!
Originally published in Computing Today, July 1980.

NAS Wars — An enjoyable space game utilising high speed graphics.
Originally published in Computing Today, March 1981.

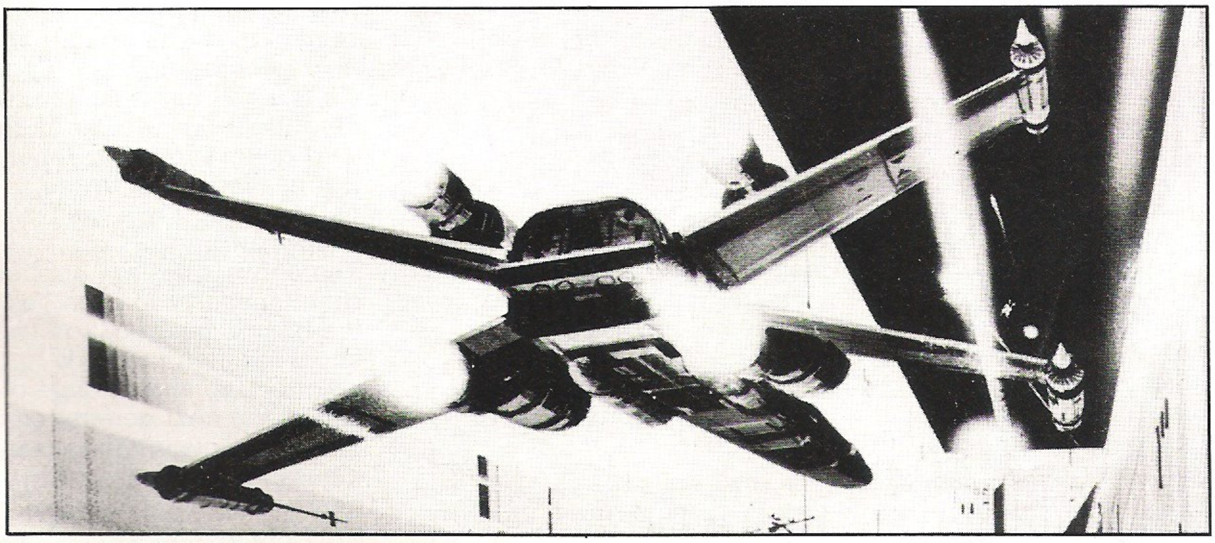
Labyrinth — The program first constructs a two-dimensional maze and then allows you to wander around inside. Not one for those of you who are claustrophobic.
Originally published in Computing Today, January 1981.

Pinball — Ever fancied being a Pinball Wizard? Now's your chance.
Originally published in Computing Today, April 1980.

Stomper — The object of this sensitive little game is to chase an insect around the screen . . .and then stamp on it!
Originally published in Computing Today, March 1979.

Kirk vs The Cursor — An enterprising game for all you Commodores out there.
Originally published in Computing Today, June 1980.

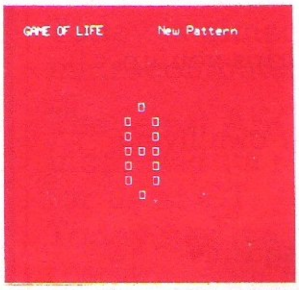
Patterns — Here's a program to show off the graphics capabilities of your machine.
Originally published in Computing Today, December 1980.



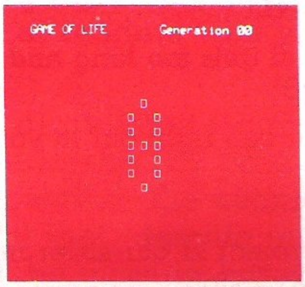
Marek Kuczynski

NASCOM LIFE

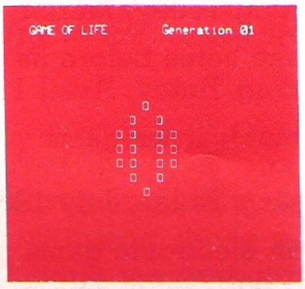
A famous game on a popular system, what more could you ask?



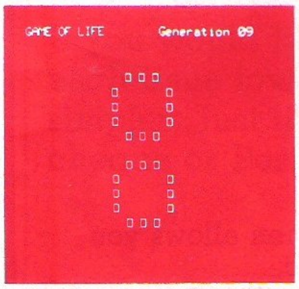
```
GAME OF LIFE        New Pattern
```



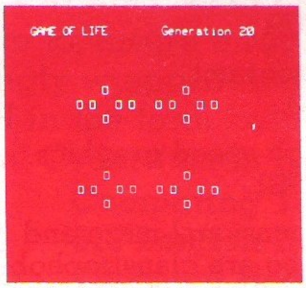
```
GAME OF LIFE        Generation 00
```



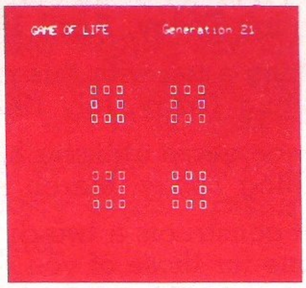
```
GAME OF LIFE        Generation 01
```



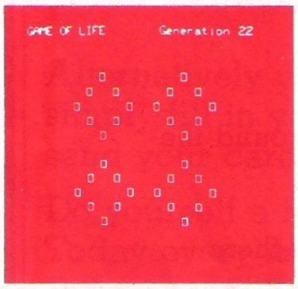
```
GAME OF LIFE        Generation 09
```



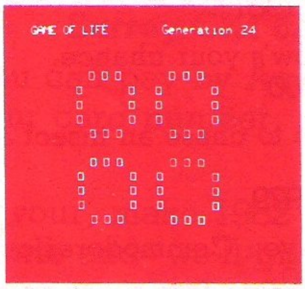
```
GAME OF LIFE        Generation 20
```



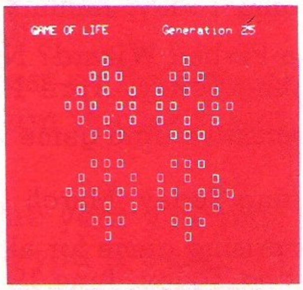
```
GAME OF LIFE        Generation 21
```



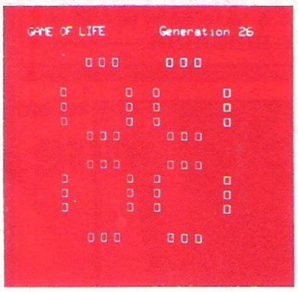
```
GAME OF LIFE        Generation 22
```



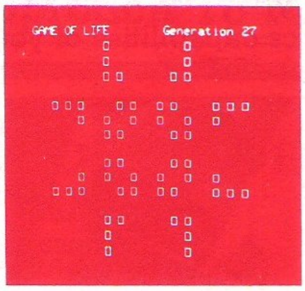
```
GAME OF LIFE        Generation 24
```



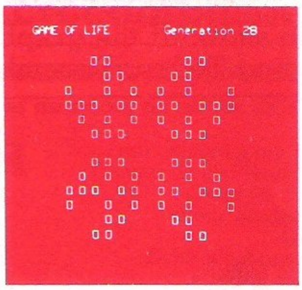
```
GAME OF LIFE        Generation 25
```



```
GAME OF LIFE        Generation 26
```



```
GAME OF LIFE        Generation 27
```



```
GAME OF LIFE        Generation 28
```

Many of you owning a NASCOM 1 will, no doubt, be tired of playing Mastermind or Hangman, and would welcome a more interesting game to play. This article is about the game of Life, and includes a fully assembled program listing to run on a minimum NASCOM 1.

Life was developed by John Horton Conway, a mathematician from Cambridge University. Its name comes from its resemblance to changing societies of living organisms which alter in position and number from generation to generation.

What Is Life?

Life can be described as a mathematical game, played on a two-dimensional board, similar to a chess board with all white squares. An initial pattern is entered on the board, and then the computer applies the basic laws of life to the pattern: Birth, Survival or Death.

Rules Of The Game

Imagine a large board containing a gridwork of squares. Each square is called a cell and all cells are identical. Ideally, the board is infinite, but for practical purposes, it is bounded on all four sides by a border, which confines the pattern to a particular area.

Each cell can be either dead or alive, and can sense the state of its eight neighbouring cells.

After each generation, the board changes. Some cells die, some are born and some remain the same, according to these simple laws of life:

(1) Any cell (dead or alive) which has exactly three live neighbouring cells will be alive in the next generation.

(2) Any cell with exactly two live neighbouring cells will remain in the same state in the next generation.

(3) If a cell has less than two neighbours, it will die in the next generation from loneliness; if it has more than three neighbours, it will die from over-crowding.

The above rules are applied simultaneously to each cell on the board. Every cell is checked, along with its neighbours, and the fate of that cell in the next generation is decided. This involves many thousands of checks per generation — a task ideally suited to a computer, which can compute a new generation in a fraction of a second. It would take an average person with a pen and paper at least an hour to check through the entire board once.

TECHNICAL DETAILS

The program was written to run on a minimum NASCOM 1, utilising almost all of the available user RAM from 0C50 to 0FFF Hex. The program can be split into various parts.

Routines INIT to BOTTOM generate a playing board or buffer starting at location 0E3D Hex. A grid of cells, 24 x 15, is produced surrounded by a border, which limits the growth of any pattern to a fixed area.

Routines SCAN to QUIT enable a pattern to be written onto the screen via the keyboard.

Routines CPYVDU to BLANK transfer the screen contents to the buffer in RAM. For each live cell on the screen, a '1' is written into the corresponding cell in the buffer. A '0' is written for each dead cell.

The heart of the program begins at LOOP. Each cell is selected in turn and routine CHECK is called for each of its eight neighbours. CHECK tests bit 0 of each cell in the buffer. Bit 0 is the 'current generation' bit in each cell. Border cells and dead cells are ignored, and the 'neighbour counter' in register B is incremented for each live neighbouring cell found.

After testing the cell, the number of neighbours is determined. If three neighbours are found, bit 1 is set in the buffer cell, this being the 'next generation' bit. If two neighbours are found, bit 0 is tested and copied into bit 1; if neither three nor two live neighbouring cells are found, bit 1 is reset. These operations satisfy the laws of life.

After the complete buffer has been tested, its entire contents are shifted, cell by cell, one bit to the right by routine REGEN. The 'next generation' now becomes the 'current generation'.

After regenerating each cell, bit 0 is tested, and if it is a '1', a □ is written into the corresponding screen position, by routines DEAD to LDSCRN. A blank is written if bit 0 is zero.

GEN and COUNTR deal with the generation counter. The method employed here may seem rather odd, ie incrementing the ASCII code for the appropriate numbers — but it works, and also takes less bytes than the conventional means of using the DECIMAL ADJUST ACCUMULATOR (DAA) instruction.

Routines MANUAL and AUTO deal with the two selectable run modes of the program, while CHOICE decides on action to be taken when the run has been terminated. Routine KEY is used extensively throughout the program, scanning the keyboard until a key is pressed.

At the end of the program are strings of ASCII text which are written into the top line of the screen. This line is non-scrolling, so it is ideal for headings and comments.

The memory from 0E3D to 0FE6 Hex is reserved for the buffer, while the remaining RAM is used by the program stack. Extensive use had been made of labels in the program; this not only makes the listing more understandable, but allows straightforward re-assembly for those fortunate enough to have an assembler to run on their NASCOM and who would like to relocate the program. All labels external to the program, ie monitor routines for the loader, are listed at the end of the program.

The program has been written using standard Z80 mnemonics on a Z80 assembler. All hexadecimal constants are prefixed by a '0', if the first digit is A to F, and suffixed by an 'H'. All other constants are decimal, and are converted to the appropriate hexadecimal object code by the assembler.

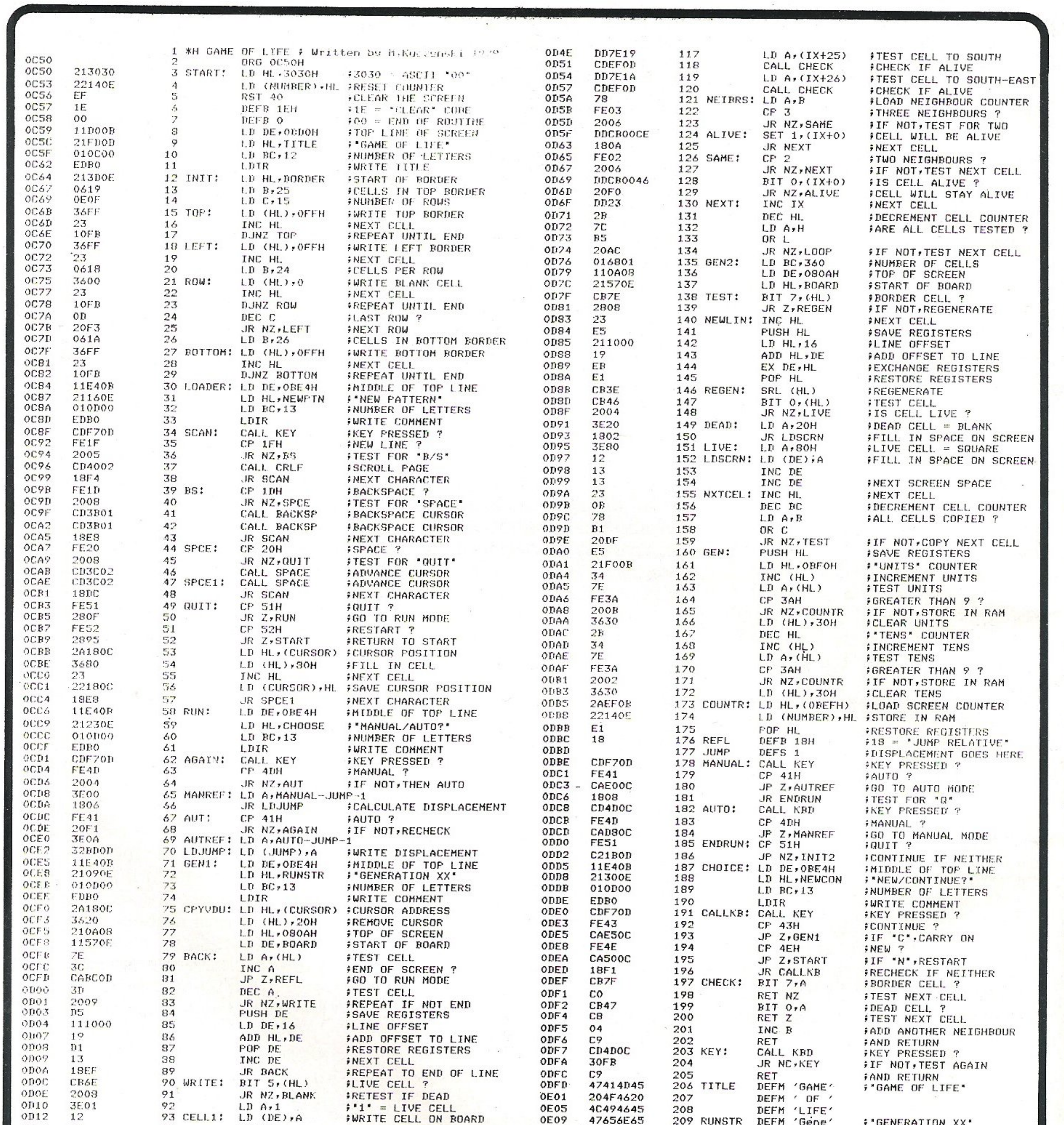
DEFB 18H assigns the value 18 (Hex) to that particular byte.
DEFS 1 reserves one byte in RAM for a variable.
DEFM 'GAME' stores the ASCII equivalent of a string of text in quotes in RAM.
EQU assigns a numeric value to a label.

HOW TO PLAY

Once the program has been loaded into RAM (and saved on cassette), enter EC50 'NEWLINE' to start the program. The screen should clear and the text 'GAME OF LIFE New Pattern' should appear on the top line; the cursor will be positioned at the bottom left of the screen. A pattern can now be entered via the keyboard, using any key with the exception of 'Space', 'B/S' and 'Newline' which provide the normal functions, 'R' which restarts the loader if a mess has been made of the pattern, or 'Q' which quits input mode and awaits further instructions.

Having entered the pattern (and making sure that it is as near to the centre of the screen as possible), press 'Q' and the text on the top line should change to 'Manual/Auto', to which the reply can either be 'A' which immediately enters auto mode, or 'M' for manual mode. In auto mode, the generations are computed one after another. Due to the large amount of cell testing involved in the program, it takes approximately 0.2 seconds to compute and display a new generation so in auto mode, the pattern changes five times a second. This fortunately happens to be fast enough to give a continuously changing display, yet slow enough to be able to observe the individual steps without having to introduce any extra delay loops into the program. This mode is ideal for tried and tested patterns, however, for new patterns it is always best to use manual mode. In this mode, pressing any key (except 'A' or 'Q') will single-step the program a generation at a time, enabling easy analysis of users' patterns.

During auto mode, if 'M' is pressed, manual mode is selected; likewise during manual mode, control can be transferred to auto mode by pressing 'A'. During both ▶



```
                  *H GAME OF LIFE ; Written by H.Kuczenski 1979
0C50                         ORG 0C50H
0C50  213030      START:     LD HL,3030H      ;3030 = ASCII "00"
0C53  22140E                 LD (NUMBER),HL   ;RESET COUNTER
0C56  EF                     RST 40           ;CLEAR THE SCREEN
0C57  1E                     DEFB 1EH         ;1E = "CLEAR" CODE
0C58  00                     DEFB 0           ;00 = END OF ROUTINE
0C59  11D00B                 LD DE,0BD0H      ;TOP LINE OF SCREEN
0C5C  21FD0D                 LD HL,TITLE      ;"GAME OF LIFE"
0C5F  010C00                 LD BC,12         ;NUMBER OF LETTERS
0C62  EDB0                   LDIR             ;WRITE TITLE
0C64  213D0E      INIT:      LD HL,BORDER     ;START OF BORDER
0C67  0619                   LD B,25          ;CELLS IN TOP BORDER
0C69  0E0F                   LD C,15          ;NUMBER OF ROWS
0C6B  36FF        TOP:       LD (HL),0FFH     ;WRITE TOP BORDER
0C6D  23                     INC HL           ;NEXT CELL
0C6E  10FB                   DJNZ TOP         ;REPEAT UNTIL END
0C70  36FF        LEFT:      LD (HL),0FFH     ;WRITE LEFT BORDER
0C72  23                     INC HL           ;NEXT CELL
0C73  0618                   LD B,24          ;CELLS PER ROW
0C75  3600        ROW:       LD (HL),0        ;WRITE BLANK CELL
0C77  23                     INC HL           ;NEXT CELL
0C78  10FB                   DJNZ ROW         ;REPEAT UNTIL END
0C7A  0D                     DEC C            ;LAST ROW ?
0C7B  20F3                   JR NZ,LEFT       ;NEXT ROW
0C7D  061A                   LD B,26          ;CELLS IN BOTTOM BORDER
0C7F  36FF        BOTTOM:    LD (HL),0FFH     ;WRITE BOTTOM BORDER
0C81  23                     INC HL           ;NEXT CELL
0C82  10FB                   DJNZ BOTTOM      ;REPEAT UNTIL END
0C84  11E40B      LOADER:    LD DE,0BE4H      ;MIDDLE OF TOP LINE
0C87  21160E                 LD HL,NEWPTN     ;"NEW PATTERN"
0C8A  010D00                 LD BC,13         ;NUMBER OF LETTERS
0C8D  EDB0                   LDIR             ;WRITE COMMENT
0C8F  CDF70D      SCAN:      CALL KEY         ;KEY PRESSED ?
0C92  FE1F                   CP 1FH           ;NEW LINE ?
0C94  2005                   JR NZ,BS         ;TEST FOR "B/S"
0C96  CD4002                 CALL CRLF        ;SCROLL PAGE
0C99  18F4                   JR SCAN          ;NEXT CHARACTER
0C9B  FE1D        BS:        CP 1DH           ;BACKSPACE ?
0C9D  2008                   JR NZ,SPCE       ;TEST FOR "SPACE"
0C9F  CD3B01                 CALL BACKSP      ;BACKSPACE CURSOR
0CA2  CD3B01                 CALL BACKSP      ;BACKSPACE CURSOR
0CA5  18E8                   JR SCAN          ;NEXT CHARACTER
0CA7  FE20        SPCE:      CP 20H           ;SPACE ?
0CA9  2008                   JR NZ,QUIT       ;TEST FOR "QUIT"
0CAB  CD3C02                 CALL SPACE       ;ADVANCE CURSOR
0CAE  CD3C02      SPCE1:     CALL SPACE       ;ADVANCE CURSOR
0CB1  18DC                   JR SCAN          ;NEXT CHARACTER
0CB3  FE51        QUIT:      CP 51H           ;QUIT ?
0CB5  280F                   JR Z,RUN         ;GO TO RUN MODE
0CB7  FE52                   CP 52H           ;RESTART ?
0CB9  2895                   JR Z,START       ;RETURN TO START
0CBB  2A180C                 LD HL,(CURSOR)   ;CURSOR POSITION
0CBE  3680                   LD (HL),80H      ;FILL IN CELL
0CC0  23                     INC HL           ;NEXT CELL
0CC1  22180C                 LD (CURSOR),HL   ;SAVE CURSOR POSITION
0CC4  18E8                   JR SPCE1         ;NEXT CHARACTER
0CC6  11E40B      RUN:       LD DE,0BE4H      ;MIDDLE OF TOP LINE
0CC9  21230E                 LD HL,CHOOSE     ;"MANUAL/AUTO?"
0CCC  010D00                 LD BC,13         ;NUMBER OF LETTERS
0CCF  EDB0                   LDIR             ;WRITE COMMENT
0CD1  CDF70D      AGAIN:     CALL KEY         ;KEY PRESSED ?
0CD4  FE4D                   CP 4DH           ;MANUAL ?
0CD6  2004                   JR NZ,AUT        ;IF NOT,THEN AUTO
0CD8  3E00        MANREF:    LD A,MANUAL-JUMP-1
0CDA  1806                   JR LDJUMP        ;CALCULATE DISPLACEMENT
0CDC  FE41        AUT:       CP 41H           ;AUTO ?
0CDE  20F1                   JR NZ,AGAIN      ;IF NOT,RECHECK
0CE0  3E0A        AUTREF:    LD A,AUTO-JUMP-1
0CE2  32BD0D      LDJUMP:    LD (JUMP),A      ;WRITE DISPLACEMENT
0CE5  11E40B      GEN1:      LD DE,0BE4H      ;MIDDLE OF TOP LINE
0CE8  21090E                 LD HL,RUNSTR     ;"GENERATION XX"
0CEB  010D00                 LD BC,13         ;NUMBER OF LETTERS
0CEE  EDB0                   LDIR             ;WRITE COMMENT
0CF0  2A180C      CPYVDU:    LD HL,(CURSOR)   ;CURSOR ADDRESS
0CF3  3620                   LD (HL),20H      ;REMOVE CURSOR
0CF5  210A08                 LD HL,080AH      ;TOP OF SCREEN
0CF8  11570E                 LD DE,BOARD      ;START OF BOARD
0CFB  7E          BACK:      LD A,(HL)        ;TEST CELL
0CFC  3C                     INC A            ;END OF SCREEN ?
0CFD  CABC0D                 JP Z,REFL        ;GO TO RUN MODE
0D00  3D                     DEC A            ;TEST CELL
0D01  2009                   JR NZ,WRITE      ;REPEAT IF NOT END
0D03  D5                     PUSH DE          ;SAVE REGISTERS
0D04  111000                 LD DE,16         ;LINE OFFSET
0D07  19                     ADD HL,DE        ;ADD OFFSET TO LINE
0D08  D1                     POP DE           ;RESTORE REGISTERS
0D09  13                     INC DE           ;NEXT CELL
0D0A  18EF                   JR BACK          ;REPEAT TO END OF LINE
0D0C  CB6E        WRITE:     BIT 5,(HL)       ;LIVE CELL ?
0D0E  2008                   JR NZ,BLANK      ;RETEST IF DEAD
0D10  3E01                   LD A,1           ;"1" = LIVE CELL
0D12  12          CELL1:     LD (DE),A        ;WRITE CELL ON BOARD
0D13  23                     INC HL
0D14  23                     INC HL           ;NEXT SCREEN POSITION
0D15  13                     INC DE           ;NEXT BOARD POSITION
0D16  18E3                   JR BACK          ;NEXT CELL
0D18  AF          BLANK:     XOR A            ;"0" = DEAD CELL
0D19  18F7                   JR CELL1         ;WRITE CELL ON BOARD
0D1B  DD21570E    INIT2:     LD IX,BOARD      ;START OF BOARD
0D1F  217601                 LD HL,374        ;NUMBER OF CELLS
0D22  0600        LOOP:      LD B,0           ;CLEAR NEIGHBOUR COUNTER
0D24  DDCB007E               BIT 7,(IX40)     ;BORDER CELL ?
0D28  2045                   JR NZ,NEXT       ;IGNORE IT
0D2A  DD7EE6                 LD A,(IX-26)     ;TEST CELL TO NORTH-WEST
0D2D  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D30  DD7EE7                 LD A,(IX-25)     ;TEST CELL TO NORTH
0D33  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D36  DD7EE8                 LD A,(IX-24)     ;TEST CELL TO NORTH-EAST
0D39  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D3C  DD7EFF                 LD A,(IX-1)      ;TEST CELL TO WEST
0D3F  CDEF0D                 CALL CHECK       ;CHFCK IF ALIVE
0D42  DD7E01                 LD A,(IX+1)      ;TEST CELL TO EAST
0D45  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D48  DD7E18                 LD A,(IX+24)     ;TEST CELL TO SOUTH-WEST
0D4B  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D4E  DD7E19                 LD A,(IX+25)     ;TEST CELL TO SOUTH
0D51  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D54  DD7E1A                 LD A,(IX+26)     ;TEST CELL TO SOUTH-EAST
0D57  CDEF0D                 CALL CHECK       ;CHECK IF ALIVE
0D5A  78          NEIBRS:    LD A,B           ;LOAD NEIGHBOUR COUNTER
0D5B  FE03                   CP 3             ;THREE NEIGHBOURS ?
0D5D  2006                   JR NZ,SAME       ;IF NOT,TEST FOR TWO
0D5F  DDCB00CE    ALIVE:     SET 1,(IX+0)     ;CELL WILL BE ALIVE
0D63  180A                   JR NEXT          ;NEXT CELL
0D65  FE02        SAME:      CP 2             ;TWO NEIGHBOURS ?
0D67  2006                   JR NZ,NEXT       ;IF NOT,TEST NEXT CELL
0D69  DDCB0046               BIT 0,(IX+0)     ;IS CELL ALIVE ?
0D6D  20F0                   JR NZ,ALIVE      ;CELL WILL STAY ALIVE
0D6F  DD23        NEXT:      INC IX           ;NEXT CELL
0D71  2B                     DEC HL           ;DECREMENT CELL COUNTER
0D72  7C                     LD A,H           ;ARE ALL CELLS TESTED ?
0D73  B5                     OR L
0D74  20AC                   JR NZ,LOOP       ;IF NOT,TEST NEXT CELL
0D76  016801      GEN2:      LD BC,360        ;NUMBER OF CELLS
0D79  110A08                 LD DE,080AH      ;TOP OF SCREEN
0D7C  21570E                 LD HL,BOARD      ;START OF BOARD
0D7F  CB7E        TEST:      BIT 7,(HL)       ;BORDER CELL ?
0D81  2808                   JR Z,REGEN       ;IF NOT,REGENERATE
0D83  23          NEWLIN:    INC HL           ;NEXT CELL
0D84  E5                     PUSH HL          ;SAVE REGISTERS
0D85  211000                 LD HL,16         ;LINE OFFSET
0D88  19                     ADD HL,DE        ;ADD OFFSET TO LINE
0D89  EB                     EX DE,HL         ;EXCHANGE REGISTERS
0D8A  E1                     POP HL           ;RESTORE REGISTERS
0D8B  CB3E        REGEN:     SRL (HL)         ;REGENERATE
0D8D  CB46                   BIT 0,(HL)       ;TEST CELL
0D8F  2004                   JR NZ,LIVE       ;IS CELL LIVE ?
0D91  3E20        DEAD:      LD A,20H         ;DEAD CELL = BLANK
0D93  1802                   JR LDSCRN        ;FILL IN SPACE ON SCREEN
0D95  3E80        LIVE:      LD A,80H         ;LIVE CELL = SQUARE
0D97  12          LDSCRN:    LD (DE);A        ;FILL IN SPACE ON SCREEN
0D98  13                     INC DE
0D99  13                     INC DE           ;NEXT SCREEN SPACE
0D9A  23          NXTCEL:    INC HL           ;NEXT CELL
0D9B  0B                     DEC BC           ;DECREMENT CELL COUNTER
0D9C  78                     LD A,B           ;ALL CELLS COPIED ?
0D9D  B1                     OR C
0D9E  20DF                   JR NZ,TEST       ;IF NOT,COPY NEXT CELL
0DA0  E5          GEN:       PUSH HL          ;SAVE REGISTERS
0DA1  21F00B                 LD HL,0BF0H      ;"UNITS" COUNTER
0DA4  34                     INC (HL)         ;INCREMENT UNITS
0DA5  7E                     LD A,(HL)        ;TEST UNITS
0DA6  FE3A                   CP 3AH           ;GREATER THAN 9 ?
0DA8  200B                   JR NZ,COUNTR     ;IF NOT,STORE IN RAM
0DAA  3630                   LD (HL),30H      ;CLEAR UNITS
0DAC  2B                     DEC HL           ;"TENS" COUNTER
0DAD  34                     INC (HL)         ;INCREMENT TENS
0DAE  7E                     LD A,(HL)        ;TEST TENS
0DAF  FE3A                   CP 3AH           ;GREATER THAN 9 ?
0DB1  2002                   JR NZ,COUNTR     ;IF NOT,STORE IN RAM
0DB3  3630                   LD (HL),30H      ;CLEAR TENS
0DB5  2AEF0B      COUNTR:    LD HL,(0BEFH)    ;LOAD SCREEN COUNTER
0DB8  22140E                 LD (NUMBER),HL   ;STORE IN RAM
0DBB  E1                     POP HL           ;RESTORE REGISTERS
0DBC  18          REFL       DEFB 18H         ;18 = "JUMP RELATIVE"
0DBD              JUMP       DEFS 1           ;DISPLACEMENT GOES HERE
0DBE  CDF70D      MANUAL:    CALL KEY         ;KEY PRESSED ?
0DC1  FE41                   CP 41H           ;AUTO ?
0DC3  CAE00C                 JP Z,AUTREF      ;GO TO AUTO MODE
0DC6  1808                   JR ENDRUN        ;TEST FOR "Q"
0DC8  CD4D0C      AUTO:      CALL KBD         ;KEY PRESSED ?
0DCB  FE4D                   CP 4DH           ;MANUAL ?
0DCD  CAD80C                 JP Z,MANREF      ;GO TO MANUAL MODE
0DD0  FE51        ENDRUN:    CP 51H           ;QUIT ?
0DD2  C21B0D                 JP NZ,INIT2      ;CONTINUE IF NEITHER
0DD5  11E40B      CHOICE:    LD DE,0BE4H      ;MIDDLE OF TOP LINE
0DD8  21300E                 LD HL,NEWCON     ;"NEW/CONTINUE?"
0DDB  010D00                 LD BC,13         ;NUMBER OF LETTERS
0DDE  EDB0                   LDIR             ;WRITE COMMENT
0DE0  CDF70D      CALLKB:    CALL KEY         ;KEY PRESSED ?
0DE3  FE43                   CP 43H           ;CONTINUE ?
0DE5  CAE50C                 JP Z,GEN1        ;IF "C",CARRY ON
0DE8  FE4E                   CP 4EH           ;NEW ?
0DEA  CA500C                 JP Z,START       ;IF "N",RESTART
0DED  18F1                   JR CALLKB        ;RECHECK IF NEITHER
0DEF  CB7F        CHECK:     BIT 7,A          ;BORDER CELL ?
0DF1  C0                     RET NZ           ;TEST NEXT CELL
0DF2  CB47                   BIT 0,A          ;DEAD CELL ?
0DF4  C8                     RET Z            ;TEST NEXT CELL
0DF5  04                     INC B            ;ADD ANOTHER NEIGHBOUR
0DF6  C9                     RET              ;AND RETURN
0DF7  CD4D0C      KEY:       CALL KBD         ;KEY PRESSED ?
0DFA  30FB                   JR NC,KEY        ;IF NOT,TEST AGAIN
0DFC  C9                     RET              ;AND RETURN
0DFD  47414D45    TITLE      DEFM 'GAME'      ;"GAME OF LIFE"
0E01  204F4620               DEFM ' OF '
0E05  4C494645               DEFM 'LIFE'
0E09  47656E65    RUNSTR     DEFM 'Gene'      ;"GENERATION XX"
0E0D  72617469               DEFM 'rati'
0E11  6F6E20                 DEFM 'on '
0E14              NUMBER     DEFS 2           ;COUNTER GOES HERE
0E16  4E657720    NEWPTN     DEFM 'New '      ;"NEW PATTERN  "
0E1A  50617474               DEFM 'Patt'
0E1E  65726E20               DEFM 'ern '
0E22  20                     DEFM ' '
0E23  4D616E75    CHOOSE     DEFM 'Manu'      ;"MANUAL/AUTO? "
0E27  616C2F41               DEFM 'al/A'
0E2B  75746F3F               DEFM 'uto?'
0E2F  20                     DEFM ' '
0E30  4E65772F    NEWCON     DEFM 'New/'      ;"NEW/CONTINUE?"
0E34  436F6E74               DEFM 'Cont'
0E38  696E7565               DEFM 'inue'
0E3C  3F                     DEFM '?'
0E3D              BORDER     DEFS 25+1        ;26 BYTES FOR TOP BORDER
0E57              BOARD      DEFS 400         ;400 BYTES FOR BOARD
                  BACKSP     EQU 013BH
                  CRLF       EQU 0240H
                  CURSOR     EQU 0C18H
                  KBD        EQU 0C4DH
                  SPACE      EQU 023CH
```

modes, the generation number is updated at the top of the screen, which is a very useful feature for keeping track of new patterns. Pressing 'Q' quits run mode, and the text 'New/Continue?' is written on the top line. The user then has the choice of 'C' to continue the run, or 'N' to clear the screen and enter a new pattern.

When entering large patterns, make sure that they are as centrally positioned on the screen as possible. If you do not take this precaution, one edge of the growing pattern may hit the border before the other edge, and either symmetry will be lost or the pattern will take a totally different course and maybe even die out completely.

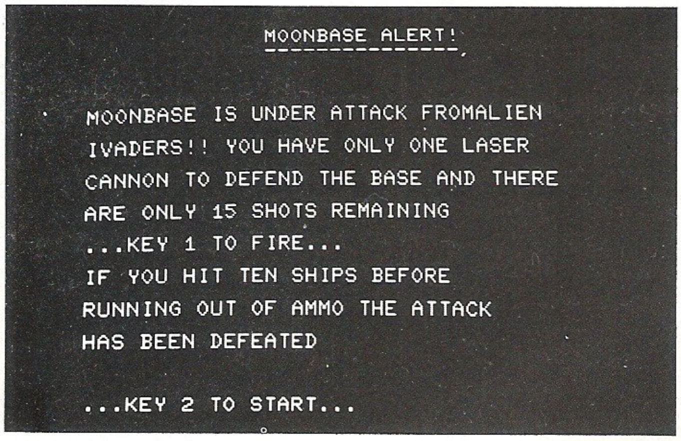
I would be interested to hear from readers who have tried this program and have discovered some new and interesting patterns, or ideas on improving the program.

Life is a very addictive game. Once hooked, it is all too easy to stay up to the early hours of the morning trying to invent the 'ultimate pattern'.

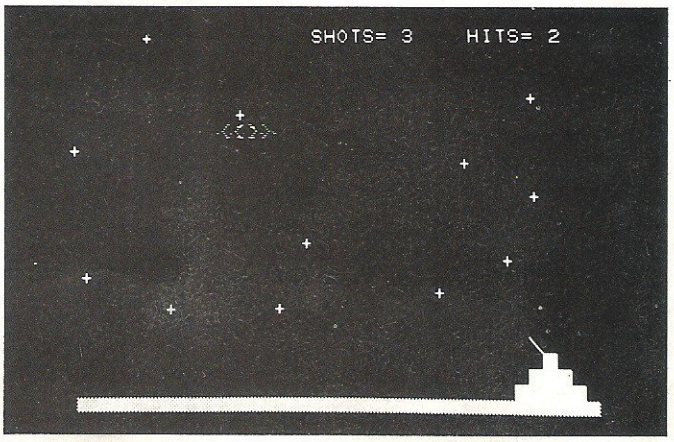
J Consadine

MOONBASE ALERT

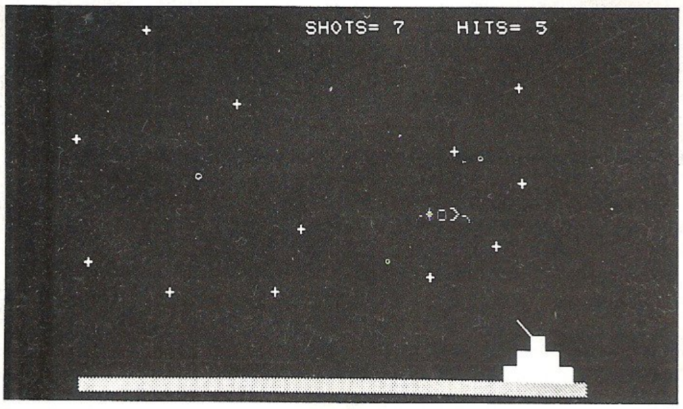
The aliens are coming . . .



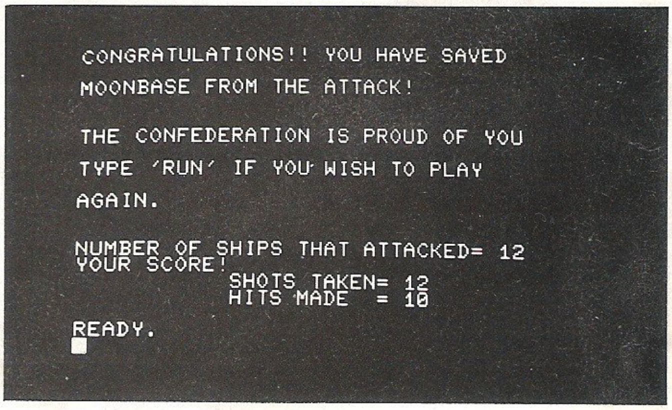
```
                MOONBASE ALERT!
                ---------------

 MOONBASE IS UNDER ATTACK FROMALIEN
 IVADERS!! YOU HAVE ONLY ONE LASER
 CANNON TO DEFEND THE BASE AND THERE
 ARE ONLY 15 SHOTS REMAINING
 ...KEY 1 TO FIRE...
 IF YOU HIT TEN SHIPS BEFORE
 RUNNING OUT OF AMMO THE ATTACK
 HAS BEEN DEFEATED

 ...KEY 2 TO START...
```



```
SHOTS= 3    HITS= 2
```



```
SHOTS= 7    HITS= 5
```



```
CONGRATULATIONS!! YOU HAVE SAVED
MOONBASE FROM THE ATTACK!

THE CONFEDERATION IS PROUD OF YOU
TYPE 'RUN' IF YOU WISH TO PLAY
AGAIN.

NUMBER OF SHIPS THAT ATTACKED= 12
YOUR SCORE!
           SHOTS TAKEN= 12
           HITS MADE   = 10

READY.
```

PROGRAM STRUCTURE

Statement	Function	Action
Lines 280-400	Set Up	Generate Moon surface and base.
Lines 410-470	Background	Generate star background.
Lines 520-570	Flight Path	Generate flight path of the ship.
Lines 580-630	Ship Position	POKE ship on the screen.
Line 680	Shell Position	PEEKs next position of shell so the following lines print a shell or, if a ship is hit, print an explosion.
Line 760	Speed	This line can be altered to slow the present rate of POKEing on and off the screen if desired.
Lines 770-820	Off Screen	POKE the ship and shell of the screen.
Line 850	Ship Hit	Strikes the ship off the screen if hit.

This program was originally written as part of a trilogy published in March 1980. The game is a variant on the old theme of judging where your target will be when the shell arrives . . . not always as easy as it looks.

Written for the Commodore PET and easily transferred to any system using a memory mapped screen supporting PEEK and POKE statements, the program can be altered to suit the player.

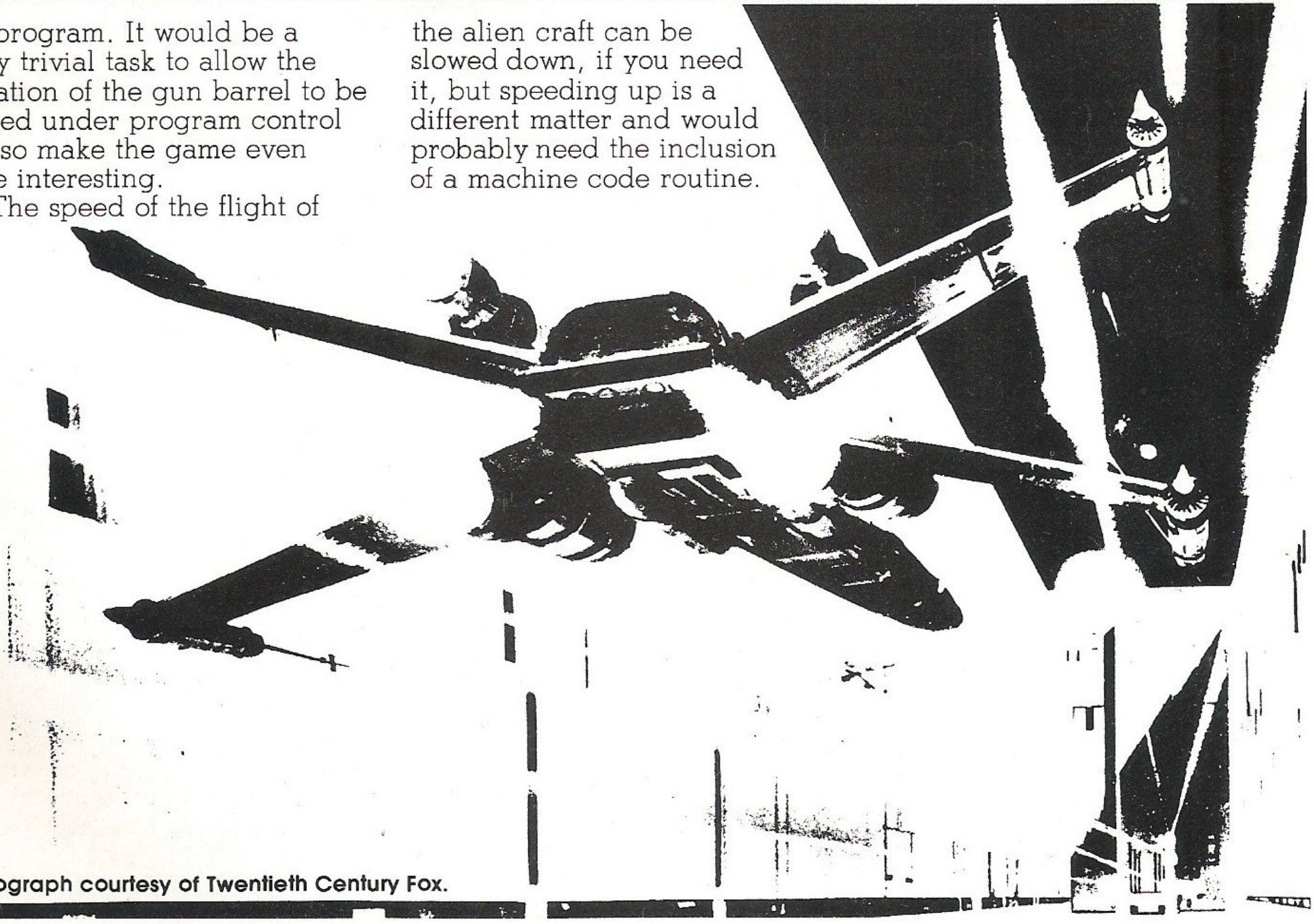
HOW TO PLAY

Actually using the program is simplicity itself, all you have to do is judge when to press the fire button! The alien ships can cross at a number of different heights above your gun; these are determined randomly within

```
100  PRINT "[CLS]"
110  PRINT "[13 SPC]---------------"
120  PRINT "[13 SPC]MOONBASE ALERT!"
130  PRINT "[13 SPC]---------------"
140  PRINT:PRINT:PRINT
150  PRINT "MOONBASE IS UNDER ATTACK FROM ALIEN":PRINT
160  PRINT "INVADERS!! YOU HAVE JUST ONE LASER":PRINT
170  PRINT "CANNON TO DEFEND THE BASE AND THERE":PRINT
180  PRINT "ARE JUST 15 SHOTS LEFT":PRINT
190  PRINT "...KEY 1 TO FIRE...":PRINT
200  PRINT "IF YOU HIT TEN SHIPS BEFORE":PRINT
210  PRINT "THE AMMO RUNS OUT THE ATTACK":PRINT
220  PRINT "HAS BEEN DEFEATED.":PRINT
230  PRINT:PRINT
240  PRINT "...KEY 2 TO START..."
250  GET A:IF A=Ø THEN 25Ø
260  ON A GOTO 27Ø,29Ø
270  PRINT "[CLS]":GOTO 24Ø
280  REM ** MOON SURFACE AND BASE
290  PRINT "[CLS]"
300  POKE 32768+(4Ø*2Ø)+34,77
310  POKE 32768+(4Ø*21)+35,16Ø
320  FOR X=1 TO 3
330  POKE 32768+(4Ø*22)+33+X,16Ø
340  NEXT X
350  FOR Y=1 TO 5
360  POKE 32768+(4Ø*23)+32+Y,16Ø
370  NEXT Y
380  FOR Z=1 TO 38
390  POKE 32768+(4Ø*24)+Z,1Ø2
400  NEXT Z
410  REM ** BACKGROUND STARS
420  DATA 45,234,252,32Ø,389,474,577,632,641,7Ø7,727,735
430  FOR X=Ø TO 11
440  READ A
450  POKE 32768+A,43
460  NEXT X
470  FOR A=1 TO 1ØØØ:NEXT A
480  T=Ø:M=Ø:REM ** TOTALS, SHOTS AND HITS
490  N=Ø:S=Ø:REM ** SHELL HEIGHT AND SHIPS
500  S$="SHOTS="
510  H$="HITS="
520  REM ** HEIGHT OF SHIP
530  D=INT(1Ø*RND(1)+Ø.5):N=Ø:W=Ø
540  S=S+1
550  IF D>=7 THEN C=4:GOTO 58Ø
560  IF D<=3 THEN C=7:GOTO 58Ø
570  IF 3<D<7 THEN C=13
580  FOR X=Ø TO 35
590  POKE 32768+(4Ø*C)+X,85
600  POKE 32768+(4Ø*C)+X+1,6Ø
610  POKE 32768+(4Ø*C)+X+2,87
620  POKE 32768+(4Ø*C)+X+3,62
630  POKE 32768+(4Ø*C)+X+4,73
640  IF N<>Ø THEN 67Ø
650  GET B:IF B=Ø THEN 76Ø
660  IF B>Ø THEN T=T+1:PRINT "[18 SPC]";S$;T:
     PRINT "[2 CU]"
670  N=N+1
680  W=PEEK(32768+(4Ø*(19-N))+33-N)
690  IF W=85 OR W=6Ø OR W=87 OR W=62 OR W=73 THEN 71Ø
700  GOTO 75Ø
710  POKE 32768+(4Ø*(19-N))+33-N,42
720  M=M+1:PRINT "[3Ø SPC]";H$;M:PRINT "[2 CU]"
730  IF M=1Ø THEN 87Ø
740  GOTO 76Ø
750  POKE 32768+(4Ø*(19-N))+33-N,46
760  POKE 32768+(4Ø*C)+X,32
770  POKE 32768+(4Ø*C)+X+1,32
780  POKE 32768+(4Ø*C)+X+2,32
790  POKE 32768+(4Ø*C)+X+3,32
800  POKE 32768+(4Ø*C)+X+4,32
810  POKE 32768+(4Ø*(19-N))+33-N,32
820  IF N=21 THEN N=Ø:GOTO 85Ø
830  IF T=15 THEN 98Ø
840  IF W=85 OR W=6Ø OR W=87 OR W=62 OR W=73 THEN 53Ø
850  NEXT X
860  GOTO 53Ø
870  PRINT "[CLS]":PRINT:PRINT
880  PRINT "CONGRATULATIONS!! YOU HAVE SAVED":PRINT
890  PRINT "MOONBASE FROM THE ATTACK!":PRINT:PRINT
900  PRINT "THE CONFEDERATION IS PROUD OF YOU":PRINT
910  PRINT "TYPE 'RUN' IF YOU WANT TO PLAY AGAIN"
920  PRINT:PRINT
930  PRINT "NUMBER OF SHIPS THAT ATTACKED=";S
940  PRINT "YOUR SCORE!"
950  PRINT "[11 SPC]SHOTS TAKEN=";T
960  PRINT "[11 SPC]HITS MADE  =";M
970  END
980  PRINT "[CLS]":PRINT:PRINT
990  PRINT "YOU HAVE JUST RUN OUT OF AMMO!":PRINT
1000 PRINT "THE MOONBASE HAS BEEN DESTROYED"
1010 PRINT:PRINT:PRINT "TYPE 'RUN' IF YOU WANT TO TRY
     AGAIN"
1020 PRINT
1030 PRINT "[11 SPC]SHIPS ATTACKED":PRINT
1040 PRINT "YOU HIT ";M;" WITH ";T;" SHOTS"
1050 END
```

the program. It would be a fairly trivial task to allow the elevation of the gun barrel to be altered under program control and so make the game even more interesting.

The speed of the flight of the alien craft can be slowed down, if you need it, but speeding up is a different matter and would probably need the inclusion of a machine code routine.

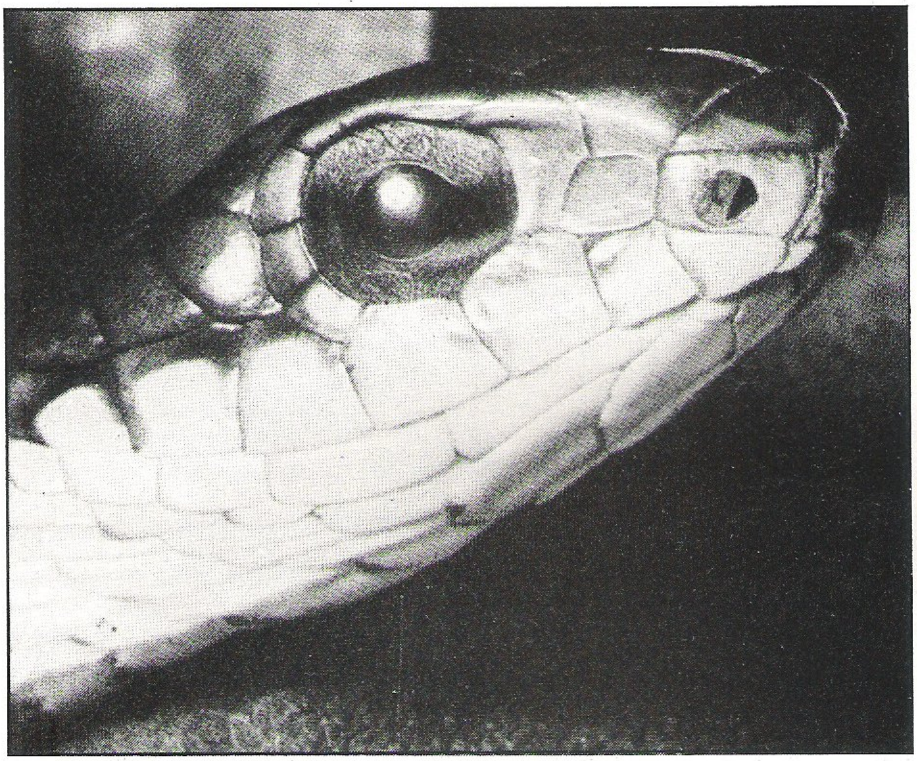


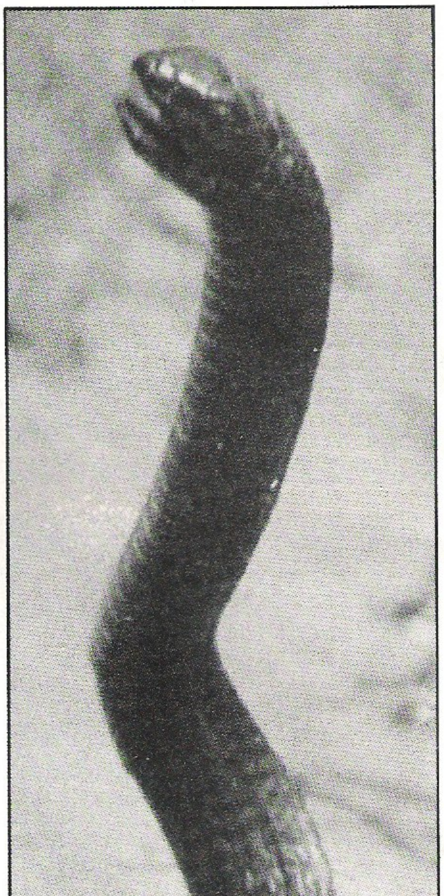
Photograph courtesy of Twentieth Century Fox.

T G Royle

SNAKES

Here's a game to tempt you — wriggle out of this one if you can!





This program was written on a Tangerine Micron and plays a slightly unusual graphics game. The object is to steer your 'snake', represented by ***>, around the screen. At random time intervals and in random locations, blocks will appear and the object is to get the head of your snake into the block. If you do this before the block disappears then you are awarded a score. This score is added to your total and is then 'counted-down'. When it reaches zero you can roam off in search of another block. As time progresses your snake gets longer and the risk of crossing your previous path increases. If this happens, or if you hit the outer wall, you will lose one of your three lives.

Game Alterations

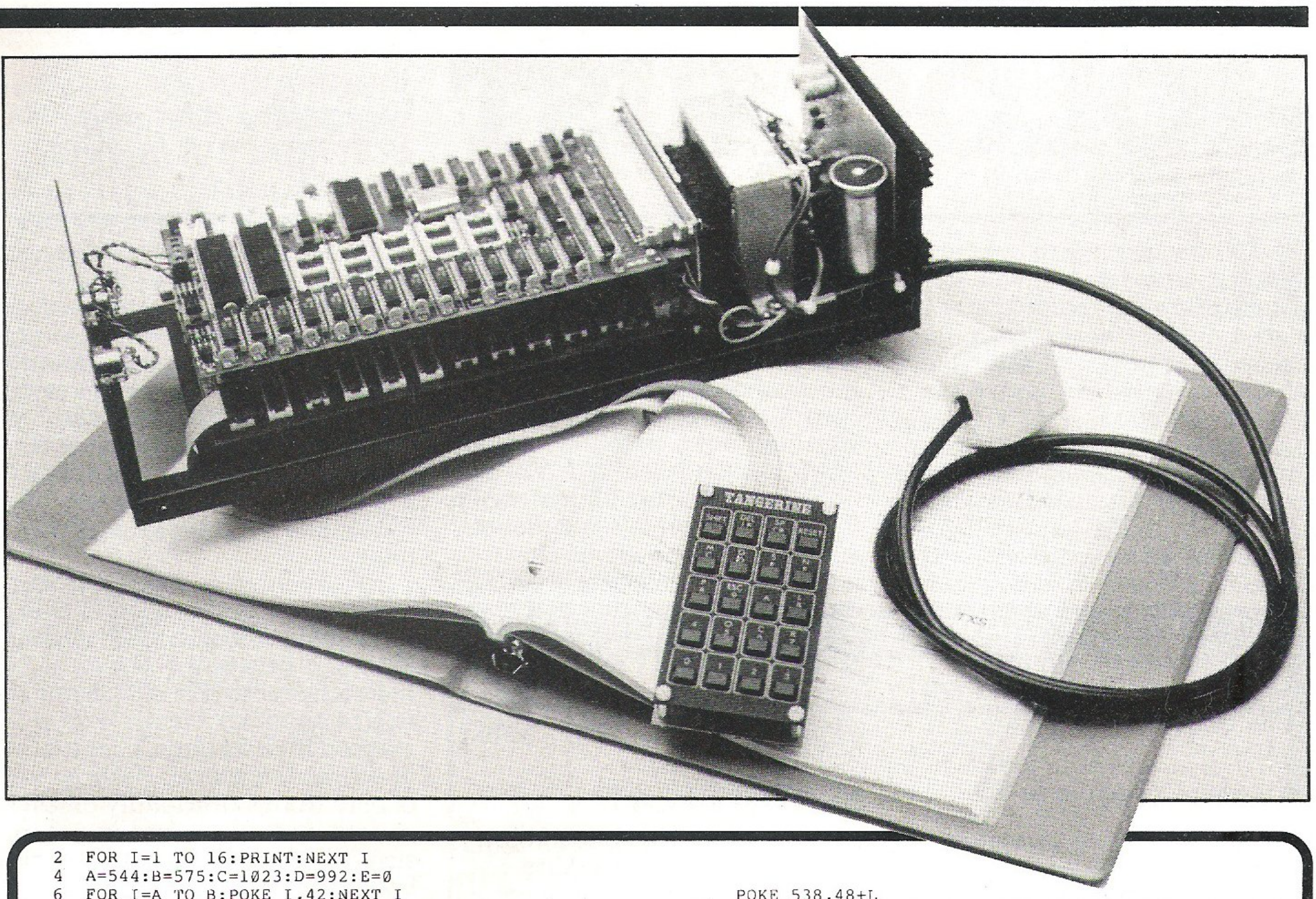
Changes can be made by adjusting the value of R in lines 82-86, a smaller value making the snake move faster. Reducing the value of W in line 253 increases the speed at which the snake gets longer.

The random number seed in line 800 controls the time between each block being generated. The seed in line 710 controls the time for which each block is displayed.

Some of the other lines are of note for non-Micron owners. Line 2 performs a 'Clear Screen' function and line 26 deletes the character on the screen after a GET; this is needed owing to a fault in the original Microsoft BASIC. The PEEK in line 255 returns the Hex value of the last key pressed.

TECHNICAL DETAILS

The Micron screen is based on a 32 character line with 16 lines on the screen at any one time. Memory locations between 512 and 1023 are used for PEEKing and POKEing to the display. In general, the character set for graphics is the same as that used by the NASCOM (see our 'Graphic Details' article).



```
   2  FOR I=1 TO 16:PRINT:NEXT I
   4  A=544:B=575:C=1Ø23:D=992:E=Ø
   6  FOR I=A TO B:POKE I,42:NEXT I
   8  FOR I=B TO C STEP 32:POKE I,42:NEXT I
  1Ø  FOR I=C TO D STEP-1:POKE I,42:NEXT I
  12  FOR I=D TO A+32 STEP-32:POKE I,42:NEXT I
  14  A=A+32:B=B+31:C=C-33:D=D-31:E=E+1
  16  IF E=6 GOTO 2Ø
  18  GOTO 6
  2Ø  POKE 781,83:POKE 782,78:POKE 783,65:POKE 784,75:
      POKE 785,69:POKE 786,83
  24  PRINT "DO YOU WANT INSTRUCTIONS":PRINT "PRESS 'Y'
      FOR YES, 'N' FOR NO"
  26  GET A$:POKE 3,Ø:IF A$="N" THEN FOR I=1 TO 8:PRINT:
      NEXT I:GOTO 7Ø
  3Ø  PRINT:PRINT:PRINT:PRINT "A SNAKE OF '<**' WILL
      MOVE"
  32  PRINT "AROUND THE SCREEN UNDER YOUR"
  34  PRINT "CONTROL. YOU CAN CHANGE ITS"
  36  PRINT "DIRECTION BY PRESSING:-"
  38  PRINT "[2 SPC]2 TO MAKE IT MOVE DOWN"
  4Ø  PRINT "[2 SPC]8 TO MAKE IT MOVE UP"
  42  PRINT "[2 SPC]4 TO MAKE IT MOVE LEFT"
  44  PRINT "[2 SPC]6 TO MAKE IT MOVE RIGHT"
  45  PRINT "PRESS 'SPACE' TO CONTINUE":GET A$
  46  FOR I=1 TO 4:PRINT:NEXT I:PRINT "YOU HAVE 3 LIVES
      (NUMBER TOP RIGHT)"
  48  PRINT "YOU WILL LOSE ONE IF YOU:-"
  5Ø  PRINT "[2 SPC]1) HIT AN OUTER WALL"
  52  PRINT "[2 SPC]2) DOUBLE BACK ON YOURSELF"
  54  PRINT "[2 SPC]3) CROSS OVER YOUR PATH":FOR I=1 TO
      6:PRINT:NEXT I
  55  PRINT "PRESS 'SPACE' TO CONTINUE":GET A$
  56  PRINT:PRINT:PRINT:PRINT "THE OBJECT IS TO RUN INTO
      THE"
  58  PRINT "BLOCKS WHICH APPEAR RANDOMLY BUT";
  6Ø  PRINT "ONLY STAY FOR A SHORT TIME SO BE";
  62  PRINT "QUICK"
  64  PRINT "THE SNAKE GETS LONGER AS THE GAME GOES ON."
  7Ø  PRINT:PRINT:PRINT:PRINT "ENTER YOUR RATING:-"
  72  PRINT "[2 SPC]BEGINNER[4 SPC]=B"
  74  PRINT "[2 SPC]NOVICE[6 SPC]=N"
  76  PRINT "[2 SPC]EXPERT[6 SPC]=E"
  78  GET A$:POKE 3,Ø
  82  IF A$="B" THEN R=12Ø:GOTO 1ØØ
  84  IF A$="N" THEN R=1ØØ:GOTO 1ØØ
  86  IF A$="E" TH3N R=6Ø:GOTO 1ØØ
  88  GOTO 7Ø
 1ØØ  FOR I=1 TO 16:PRINT:NEXT I
 12Ø  J=PEEK(49136):FOR I=545 TO 574:POKE I,192:POKE
      I+448,3:NEXT I
 13Ø  FOR I=576 TO 96Ø STEP 32:POKE I,17Ø:POKE I+31,85:
      NEXT I:POKE 49139,Ø
 135  DIM P(1ØØ)
 14Ø  X=1:Y=32:L=3:LL=3:T=Ø:T1=1+INT(9*RND(1)):SC=Ø:
      POKE 538,48+L
 145  S2=48:S3=48:S4=48:S5=48:POKE 516,S5:POKE 517,S4:
      POKE 518,S3:POKE 519,S2
 15Ø  P=INT((698-677)*RND(1)+677):B=INT(5*RND(1)):
      P=P+(B*32)
 16Ø  A=2*INT((28-25)*RND(1)+25):GOTO 26Ø
 25Ø  T1=T1=1:IF T1=T THEN 7ØØ
 252  T2=T2-1:IF T2=T THEN 8ØØ
 253  W=W+1:IF W=5Ø THEN 16ØØ
 255  A=PEEK(1)
 26Ø  IF A=5Ø THEN M=Y:D1=86:GOTO 41Ø
 27Ø  IF A=52 THEN M=-X:D1=6Ø:GOTO 41Ø
 28Ø  IF A=54 THEN M=X:D1=62:GOTO 41Ø
 29Ø  IF A=56 THEN M=-Y:D1=94
 41Ø  P=P+M:IF PEEK(P)<>32 THEN 6ØØ
 42Ø  POKE P,D1:FOR I=1 TO R:NEXT I
 43Ø  FOR LA=LL TO 1 STEP-1:P(LA)=P(LA-1):NEXT LA:
      P(1)=P:POKE P(LL),32:POKE P(1),42
 44Ø  GOTO 25Ø
 6ØØ  IF PEEK(P)=255 THEN 9ØØ
 61Ø  L=L-1:POKE 538,48+L
 62Ø  IF L=Ø THEN 1ØØØ
 625  FOR LB=1 TO LL:POKE P(LB),32:NEXT LB:POKE E,32
 63Ø  FOR I=1 TO 2ØØØ:NEXT I:GOTO 15Ø
 7ØØ  E=INT((6Ø7-577)*RND(1)+577):F=INT(13*RND(1)):
      E=E+(F*32)
 7Ø5  IF PEEK(E)<>32 THEN 7ØØ
 71Ø  J=PEEK(49136):POKE E,255:POKE 49139,Ø:
      T2=INT((6Ø-1Ø)*RND(1)+1Ø):GOTO 255
 8ØØ  POKE E,32:T1=1+INT(9*RND(1)):GOTO 255
 9ØØ  S1=INT((58-49)*RND(1)+49)
 91Ø  POKE P,S1:SC=SC+1:FOR I=1 TO 2ØØ:NEXT I
 92Ø  S2=S2+1:IF S2>57 THEN S3=S3+1:S2=48
 93Ø  IF S3>57 THEN S4=S4+1:S3=48
 94Ø  IF S4>57 THEN S5=S5+1:S4=48
 95Ø  IF S5>57 THEN 12ØØ
 96Ø  POKE 516,S5:POKE 517,S4:POKE 518,S3:POKE 519,S2
 97Ø  S1=S1-1:IF S1>47 GOTO 91Ø
 98Ø  GOTO 8ØØ
1ØØØ  FOR I=1 TO 8:PRINT:NEXT I:PRINT "[12 SPC]GAME OVER"
1Ø1Ø  PRINT "[8 SPC]YOUR SCORE IS";SC:FOR I=1 TO 6:PRINT:
      NEXT I:GOTO 15ØØ
12ØØ  FOR I=1 TO 8:PRINT:NEXT I:PRINT "[13 SPC]YOU WIN"
121Ø  PRINT "[4 SPC]YOUR SCORE IS OVER 9999":FOR I=1 TO
      6:PRINT:NEXT I
15ØØ  PRINT "DO YOU WISH TO PLAY AGAIN"
151Ø  PRINT "PRESS 'Y' FOR YES, 'N' FOR NO"
152Ø  GET A$:POKE 3,Ø
153Ø  IF A$="Y" THEN 1ØØ
154Ø  PRINT "[3 SPC]THANK YOU":END
16ØØ  LL=LL+1:W=Ø:R=R-Z
161Ø  IF R<=1Ø THEN R=1Ø
162Ø  GOTO 255
```

Simon N Goodwin

HOLOCAUST

A tactical thermonuclear wargame that you can fight out in your living room.

Holocaust is a jolly game, giving you a chance to press the red button and start a nuclear war! If that seems rather morbid, think that at least when your computer is in charge nobody gets hurt . . .

The program puts you in control of an arsenal of atomic bombs, featuring old-fashioned A bombs, bigger and better H bombs and everyone's favourite, the N or neutron bomb, which kills everything but doesn't damage the valuable factories that you will need when the war is over (to build some more missiles naturally).

The End Is Nigh?

You are faced with an invasion from the East. As the attacking tanks come rolling over the horizon, your radar scanners help you to target on them and protect your cities from capture. This is not one of the common games where you have to enter the Cartesian coordinates that you want to shoot at. In this game, your radar sights scan back and forth horizontally and vertically across the display. You select your bomb when they are pointing in the right direction and the missile comes whistling down onto the chosen target — you hope!

Of course, it is all too easy to make a slight miscalculation and blow up one of your favourite cities instead of an enemy infantry division. There again, if you'd used an H bomb you would possibly have zapped two or three cities as well as the enemy unit you were aiming at.

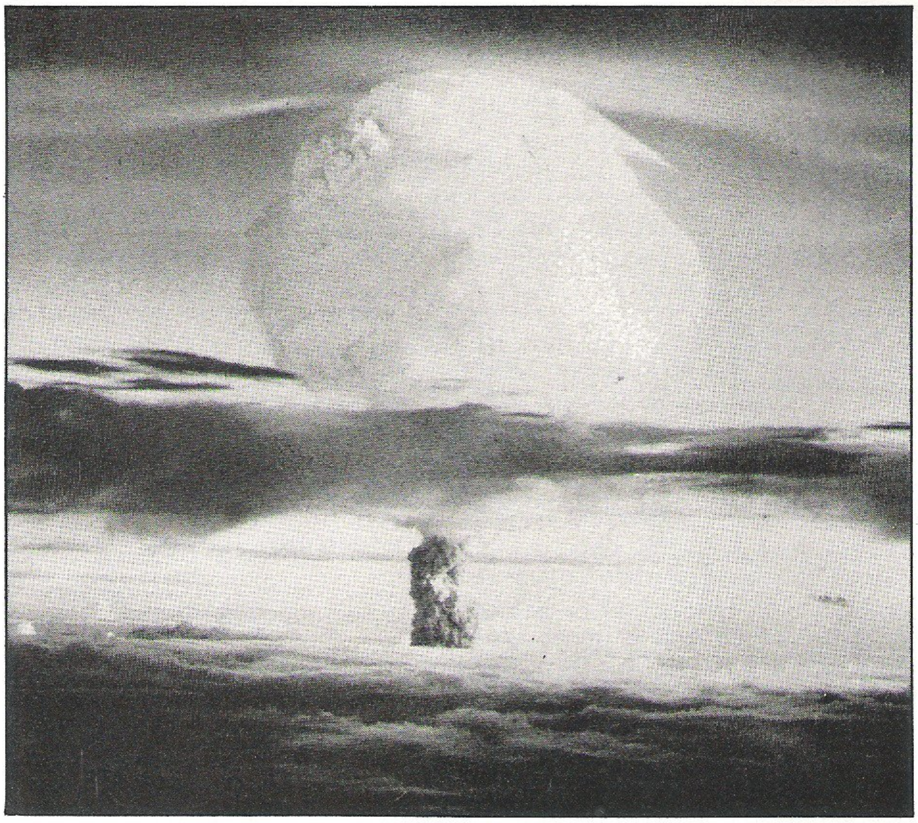
HOW TO PLAY

The aim of the game is to blow up each of the 12 attacking units before they are able to cross the display. They are continually moving in a semi-random manner, generally from east to west across the screen. Your success is measured in terms of your Devastation Rating — the 'score' printed on the left-hand edge of the screen. The lower this rating at the end of the game, the better. It ***increases*** whenever you fire a bomb, especially if it lands on one of your cities. The more damage your missile does, the greater its effect on your score. Your rating ***falls*** when you manage to hit one of the enemy units.

A continuous display of the number of missiles of each kind remaining is maintained on the left-hand side of the screen as the battle takes place. The explosion of the bombs is marked by a flashing haze on the screen. This will destroy any enemy troops or friendly cities caught beneath it, and varies in size according to the type of bomb that has been launched. H bombs and A bombs leave an area of permanent damage (the footprint) after they have exploded — if an enemy unit moves into one of these areas it is killed by radiation poisoning.

At the start of the game you are asked to enter your skill rating. This governs the number of missiles you have at the start of play — remember that the invaders will win automatically if you run out of missiles before they have all been destroyed. If you press any key other than 0-9, the machine will ignore you and wait for you to specify a valid rating. When you have done so the screen is cleared and split into two areas — a column on the left for information such as score and ammunition supplies, and a larger square area upon which the battle will take place.

The 12 arrow heads on the right of the display are the attacking forces — as soon as the screen has been fully set up they will begin to move. The 14 random asterisk '*' symbols represent your cities and industries. Two small cursors will flash along the side and top of the screen — these are your



radar scanners. As they move you can stop them by pressing either A , H or N . As soon as both vertical and horizontal target lines have been set the missile will be launched to the appropriate point on the screen — H will launch an H bomb and so forth. There is no need to press New Line or Enter when launching missiles. If you try to fire a type of missile that you have run out of, a message will appear at the bottom left of the screen and the invaders will take advantage of the chance to move unmolested.

SCORING

Rating + 15:	Neutron bomb dropped.
Rating + 50:	Hydrogen bomb dropped.
Rating + 20:	Atom bomb dropped.
Rating − 30:	Enemy unit. destroyed.
Rating + 100:	For each city blasted.
Rating + 40:	For each city captured.

TECHNICAL DETAILS

The listing may seem quite short in relation to the description of the program's facilities. This is for two main reasons — first, the requirement that it should run fairly quickly. The more variables or lines in a BASIC program, the more time it will take the interpreter to find each one. To make the program run as quickly as possible the most often used subroutine, the one that moves the attackers, has been put at the start so that BASIC can find it quickly when it searches through memory for a given line. The program has been written in a number of small subroutines. This slows it down slightly but makes it much easier to test or to modify for a different type of computer since the writing can be done piece by piece and the routines can be tested one at a time. Unfortunately, as it is a game using graphics it will not be possible to enter it straight onto other machines, except a TRS-80 level II, which should run it without changes.

Whenever possible, sensible names have been chosen for variables to make debugging the program easier — for example, temporary variables start with T , V and H contain vertical and horizontal co-ordinates, and so forth. The X coordinate of an attacker is set to zero when it is destroyed.

Particular statements which may seem odd are as follows:

CLS The command to clear the display.

DEFINT A-Z Makes all variables other than K$ be stored as integers (whole numbers only), to speed up the program.

SET(X,Y) Turns on (white) a point on the screen. X is a value between 0 and 127 and Y can be between 0-47. SET(0,0) turns on the point in the top left-hand corner of the screen.

RESET(X,Y) The reverse of SET — it turns a point on a 0-127,0-47 matrix black. If the point is black already RESET has no effect.

PRINT @ X, Moves the printing cursor to position X on the screen. As it is made up of 16 lines of 64 characters, X can be any value between 0 (top left) and 1023 (bottom right).

(PEEK(P)< >42) This expression lets you look into the screen memory. P is the place

PROGRAM STRUCTURE

Statement	Function	Action
Lines 120-280	Move Attackers	Compute and position attackers after move.
Lines 290-430	Bomb Hit	Check all attackers to see if bomb has hit.
Lines 440-470	Keyboard Scan	Look at the keyboard and see if any key has been pressed. If valid key pressed bringing it back, otherwise clear K $
Lines 480-580	Vertical Scan	Move the sights up the screen waiting for a key to be pressed.
Lines 590-660	Horizontal Scan	As above but across the screen instead of up.
Lines 670-850	A Bombing	Select weapon and drop it on the selected co-ordinates.
Lines 860-920	Out Of Ammo	You don't have any of the selected bombs left.
Lines 930-970	Bombs Dropped	Go and move the enemy.
Lines 980-1240	H Bombing	As with A bomb but make more of a mess!
Lines 1250-1410	N Bombing	Much neater, kills attackers but leaves your cities safe.
Lines 1420-1440	Conversion	Change a SET function to a POKE.
Lines 1450-1510	All Dead?	Check to see if you killed all the attackers.
Lines 1520-1620	Control	Look after the various subroutines.
Lines 1630-1690	Scores	Update and look after the scoring.
Lines 1700-1920	Instructions	Display the game rules and instructions.
Lines 1930-2200	Set Up	Create the battlefield on the screen with the cities, attackers and initial scores displayed.
Lines 2210-2290	You Lose	Too bad, you were overrun.
Lines 2300-2410	End Game	Final messages.
Lines 2420-2470	You Win	Well done message.

▶

in memory; P-15360 would tell you where to PRINT @ if you wanted to print a character there. The function returns 0 if the memory cell at P contains 42 or − 1 if it does not. (42 is the character code for '*'.)
POKE P, 191 This expression puts a character code 191 at location P in memory. This can be the same as a PRINT @ P-15360, except that it allows you to display characters that can't be typed directly on the Video Genie keyboard — code 191 is an all-white block.
K$=INKEY$ This statement reads in the current key being pressed on the keyboard — it returns a Null (empty) string if no keys are down. On a PET use the GET statement — GET on an Apple does a different thing so use:

"K$=CHR$(PEEK(−16384)) ;POKE −16368,0".

RND(N) This expression returns a random whole number between 1 and N. RANDOM at the start of the program will make sure that each game has a different sequence of random numbers. An RND statement is also used in one of the keyboard loops to vary the sequence.

VARIABLES

T,T1,T2	Temporary results.
CT	Number of cities remaining.
CA	Attacker move count.
P	Position on screen.
XL,YL	Range of bomb blast.
K$	Last character from keyboard.
AB	Atom bombs remaining.
NB	Neutron bombs remaining.
HB	Hydrogen bomb stocks.
SC	Score (Devastation Rating).
CH	Explosion character.
SK	Skill rating.
VA	The constant 15360 — address of the start of screen memory.
XP,YP	Top left X and Y coordinates hit by a missle.
AX( ),AY( )	Attackers X and Y coordinates (X positions 0-63, Y 1-15).
V,H	Vertical and Horizontal aiming coordinates (V 0-47, H 0-127).
FL	A 'flag' set to zero when only one attacker is to be moved.

TACTICS

To conclude, a few tips on how to succeed when playing the game. It is sometimes useful to lay down a barrage of missiles across the screen to act as a net to stop the advancing forces. As H bombs and A bombs leave some areas permanently 'radioactive', they can be used like landmines (!) in the hope that the attackers will walk into them. This will filter out some of them, leaving the rest to be individually blasted with the N bombs. At the higher 'Skill Ratings' you may not be able to do this as you will not have enough weapons. The main weakness of that strategy is that it increases your score since you will have to blast large areas to make a reasonable net, but it can save cities in the long run.

The targeting system generates one of the standard military problems — by the time you've taken aim, the enemy have moved somewhere else! If after you have set a horizontal line of fire the enemy move out of range, you can abort the launching of your missile by not pressing any key during the vertical radar scan. The horizontal scan will restart without any missile being launched. 'Deflection shooting' will make it easier to hit the targets — try to judge how often and how far they move and aim ahead of them accordingly.

```
100  REM ** PRINT RULES AND SET UP SCREEN
110  GOTO 1700
120  REM ** MOVE ATTACKERS
130  T=Ø
140  CA=(CA<12)*-CA
150  IF AX(CA)=Ø THEN 26Ø
160  POKE VA+AX(CA)+AY(CA)*64,32
170  AX(CA)=AX(CA)-RND(3)+1
180  AY(CA)=AY(CA)+RND(3)-2
190  IF AY(CA)<1 THEN AY(CA)=1
200  IF AY(CA)>15 THEN AY(CA)=14
210  T1=PEEK(VA+AX(CA)+AY(CA)*64)
220  IF T1=42 THEN CT=CT-1:SC=SC+4Ø
230  IF T1>127 THEN AX(CA)=Ø:GOTO 26Ø
240  IF AX(CA)>Ø THEN POKE VA+AX(CA)+AY(CA)*64,6Ø
250  IF AX(CA)<18 THEN 222Ø
260  CA=CA+1:T=T+1
270  IF T<4 AND FL THEN 14Ø
280  RETURN
290  REM ** SEE IF BOMB STRUCK ATTACKER
300  P=P-VA
310  YP=P/64
320  XP=P-YP*64
330  IF K$="H" THEN XL=2:YL=1:GOTO 35Ø
340  XL=1:YL=Ø
350  FOR Y=YP TO YP+YL
360  FOR X=XP TO XP+XL
370  FOR T=Ø TO 11
380  IF AY(T)=Y AND AX(T)=X THEN AX(T)=Ø:SC=SC-3Ø
390  NEXT T
400  NEXT X
410  NEXT Y
420  RETURN
430  REM ** SCAN THE KEYBOARD
440  K$=INKEY$
450  IF K$="" OR K$="N" OR K$="H" OR K$="A" THEN RETURN
460  K$="":RETURN
470  REM ** VERTICAL DISPLAY SCAN
480  V=4
490  SET(32,V+1):SET 32,V)
500  GOSUB 44Ø
510  FL=Ø
520  GOSUB 14Ø
530  FL=1
540  IF K$<>"" OR V>42 THEN RETURN
550  RESET(32,V+1):RESET(32,V)
560  V=V+3
570  GOTO 49Ø
580  REM ** HORIZONTAL DISPLAY
590  H=32
600  SET(H,1):SET(H+1,1)
610  GOSUB 44Ø
620  IF K$<>"" OR H>119 THEN RETURN
630  RESET(H,1):RESET(H+1,1)
640  H=H+2
650  GO   6ØØ
660  REM ** DROP AN 'A' BOMB
670  IF AB<1 THEN 86Ø
```

```
680  SC=SC+2Ø
690  AB=AB-1
700  GOSUB 142Ø
710  T=(PEEK(P)=42) OR (PEEK(P+1)=42)
720  IF T THEN SC=SC+1ØØ
730  CT=CT+T
740  FOR T=Ø TO 3
750  IF T=Ø OR T=2 THEN CH=155:GOTO 77Ø
760  CH=32
770  POKE P,CH
780  POKE P+1,CH
790  GOSUB 145Ø
800  NEXT T
810  POKE P,162
820  POKE P+1,145
830  GOSUB 3ØØ
840  RETURN
850  REM ** NONE OF THOSE BOMBS
860  PRINT @96Ø,"OUT OF ";K$;" BOMBS";
870  IF AB+HB+NB<1 THEN 226Ø
880  GOSUB 13Ø
890  GOSUB 13Ø
900  GOSUB 13Ø
910  PRINT @96Ø,"[15 SPC]";
920  REM ** BOMB DROPPED, MOVE ENEMY
930  RESET(H,1)
940  RESET(32,V+1)
950  GOSUB 26Ø
960  RETURN
970  REM ** DROP AN 'H' BOMB
980  IF HB<1 THEN 86Ø
990  SC=SC+5Ø
1000 HB=HB-1
1010 GOSUB 142Ø
1020 T=(PEEK(P)=42) OR (PEEK(P+1)=42)
1030 T=(PEEK(P+2)=42) OR (PEEK(P+64)=42) OR T
1040 T=(PEEK(P+65)=42) OR (PEEK(P+66)=42) OR T
1050 IF T THEN SC=SC-1ØØ*T
1060 CT=CT+T
1070 FOR T=Ø TO 3
1080 IF T=Ø OR T=2 THEN CH=155:GOTO 11ØØ
1090 CH=128
1100 POKE P,CH
1110 POKE P+2,CH
1120 POKE P+65,CH
1130 POKE P+1,CH
1140 POKE P+64,CH
1150 POKE P+66,CH
1160 GOSUB 145Ø
1170 NEXT T
1180 POKE P,188
1190 POKE P+66,143
1200 POKE P+2,188
1210 POKE P+64,143
1220 GOSUB 3ØØ
1230 RETURN
1240 REM ** DROP AN 'N' BOMB
1250 IF NB<1 THEN 86Ø
1260 SC=SC+15
1270 NB=NB-1
1280 GOSUB 142Ø
1290 T=(PEEK(P)=42) OR (PEEK(P+1)=42
1300 IF T THEN SC=SC+1ØØ
1310 CT=CT+1
1320 FOR T=Ø TO 3
1330 IF T=Ø OR T=2 THEN CH=191:GOTO 135Ø
1340 CH=32
1350 POKE P,CH
1360 POKE P+1,CH
1370 GOSUB 145Ø
1380 NEXT T
1390 GOSUB 3ØØ
1400 RETURN
1410 REM ** CONVERT A SET TO A POKE:
     PASS H,V - RETURN P
1420 P=VA+H/2+INT(V/3)*64
1430 RETURN
1440 REM ** SEE IF ALL ENEMIES ARE DEAD (DELAY)
1450 T1=Ø
1460 FOR T2=Ø TO 11
1470 T1=T1+AX(T2)
1480 NEXT T2
1490 IF T1=Ø THEN 243Ø
1500 RETURN
1510 REM ** MAIN CONTROL LOOP
1520 GOSUB 13Ø
1530 GOSUB 59Ø
1540 GOSUB 13Ø
1550 GOSUB 48Ø
1560 GOSUB 145Ø
1570 IF K$="N" THEN GOSUB 125Ø
1580 IF K$="A" THEN GOSUB 67Ø
1590 IF K$="H" THEN GOSUB 98Ø
1600 GOSUB 163Ø
1610 GOTO 152Ø
1620 REM ** UPDATE THE SCORES
1630 PRINT @135,HB;
1640 PRINT @199,AB;
1650 PRINT @263,NB;
1660 PRINT @391,SC;
1670 PRINT @519,CT;
1680 RETURN
1690 REM ** DISPLAY INSTRUCTIONS
1700 RANDOM
1710 DEFINT A-Z
1720 DIM AX(12),AY(12)
1730 CLS
1740 PRINT "H O L O C A U S T"
1750 PRINT:PRINT
1760 PRINT "YOU MAY DROP 'N' BOMBS, 'H' BOMBS & GOOD OLD
     FASHIONED 'A' BOMBS"
1770 PRINT "ON THE CITIES OF YOUR BELOVED COUNTRY,
     HOPING TO MISS THEM & HIT"
1780 PRINT "THE MOVING ARROWS REPRESENTING ENEMY
     INVADERS."
1790 PRINT "H BOMBS DESTROY THE LARGEST AREA AND N BOMBS
     THE LEAST"
1800 PRINT "AS THEY DO NOT LEAVE PERMANENT DAMAGE."
1810 PRINT "YOU MUST DESTROY ALL THE ENEMIES, WITHOUT
     LETTING THEM CROSS THE"
1820 PRINT "COUNTRY FROM EAST TO WEST - BLOWING UP AS
     LITTLE OF YOUR COUNTRY"
1830 PRINT "AS POSSIBLE. TO FIRE A BOMB PRESS THE
     APPROPRIATE LETTER WHEN YOUR"
1840 PRINT "HORIZONTAL AND VERTICAL SIGHTS INDICATE THE
     CORRECT CO-ORDINATES"
1850 PRINT "PRESS THE KEY WHILE THE SIGHTS ARE MOVING TO
     CHOOSE WHERE TO STOP"
1860 PRINT
1870 PRINT "ENTER YOUR SKILL RATING WHEN YOU WANT TO
     START - BETWEEN Ø AND 9"
1880 K$=INKEY$
1890 T=RND(1Ø)
1900 IF K$="" THEN 188Ø
1910 IF K$<"Ø" OR K$>"9" THEN 188Ø
1920 SK=VAL(K$)
1930 REM ** SET UP BATTLEFIELD
1940 T=Ø
1950 HB=4-SK/3:AB=14-SK:NB=18-SK
1960 SC=Ø:CT=14
1970 CLS
1980 CA=Ø:VA=1536Ø:FL=1
1990 FOR Y=Ø TO 47
2000 SET(3Ø,Y)
2010 SET(127,Y)
2020 NEXT Y
2030 PRINT @1,"HOLOCAUST !!";
2040 PRINT @128,"H BOMBS";HB;
2050 PRINT @192,"A BOMBS";AB;
2060 PRINT @256,"N BOMBS";NB;
2070 PRINT @384,"SCORE :";SC;
2080 FOR V=1 TO 14
2090 POKE VA+V*64+17+RND(4Ø),42
2100 NEXT V
2110 PRINT @512,"CITIES ";CT;
2120 FOR V=Ø TO 5
2130 AX(V)=62
2140 AY(V)=V
2150 AY(V+6)=11-V
2160 AX(V+6)=62
2170 NEXT V
2180 GOSUB 13Ø
2190 GOSUB 13Ø
2200 GOSUB 152Ø
2210 REM ** ATTACKERS WIN
2220 SC=SC+1ØØØ
2230 GOSUB 231Ø
2240 PRINT @64Ø,"COUNTRY OVERRUN";
2250 GOTO 245Ø
2260 SC=SC+1ØØØ
2270 GOSUB 231Ø
2280 PRINT @64Ø,"OUT OF MISSILES'
2290 GOTO 245Ø
2300 REM ** END OF GAME
2310 FOR T=Ø TO 5
2320 FOR T1=Ø TO 2ØØ:NEXT T1
2330 PRINT @768,"<BATTLE OVER>";
2340 PRINT @391,"[7 SPC]";
2350 FOR T1=Ø TO 2ØØ:NEXT T1
2360 K$=INKEY$
2370 PRINT @768,"[13 SPC]";
2380 PRINT @391,SC;
2390 FOR T1=Ø TO 2ØØ:NEXT T1
2400 NEXT T
2410 RETURN
2420 REM ** DEFENDERS WIN
2430 GOSUB 231Ø
2440 PRINT @64Ø,"ENEMY SURRENDER";
2450 K$=INKEY$:IF K$="" THEN 245Ø
2460 REM ** THAT'S ALL
2470 END
```

Pete Howells

AMBUSH

Keep looking over your shoulder for that surprise attack!

PROGRAM STRUCTURE

Statement	Function	Action
Lines 110-160	Set Up	The table is set up in a matrix using DATA read from line 110.
Lines 170-190	Game Speed	Control the speed at which the game is played.
Lines 200-450	Game Display	Print the game display on the screen. The semi-colon at the end of line 450 is necessary to prevent losing the top line of the display as the cursor returns to the start of a new line after having written the entire screen.
Lines 460-470	Counters	Initialise the counters, 'H' for attacks sustained and 'AA' for ammunition used.
Lines 480-490	Delay	Introduce a suitable random delay between attacks by looping until a high enough value is taken.
Lines 500-550	Direction	'AX' stores the direction of fire and is reset to zero from the previous direction.
Line 700	Attacker Test	Tests if the attacker has been destroyed.
Line 710	Player Test	Tests if the player has been destroyed.
Lines 720-740	Next Attacker	POKE a blank to the current position of the attacker on the screen, alter the current position according to the direction of travel and POKE the attacker into the next position.
Line 750	Speed Delay	Provides speed delay.
Lines 790-830	Explosion	POKE a minor explosion to the position of the attacker. The attacks sustained counter is incremented, the screen display is updated and the explosion is cleared from the screen for the next attack.
Lines 840-890	Player Hit	Display a 'KERBLAM' explosion on the screen should the attacker hit the player. The display is held on screen for a while, then the game is restarted.
Lines 900-1000	Congratulations	Print a congratulatory message on the screen and the game either restarts from scratch or is stopped according to player's response.

The original Ambush was published as a project in our sister magazine, Electronics Today International, and this game represents a computerised simulation of its features.

Because the original game used flashing lights and a number of buttons to select the direction of fire, we have tried to preserve these in this piece of software. However, the sound effects are missing; the PET doesn't have an integral sound generator so if you want to expand the program, this could be one area of interest.

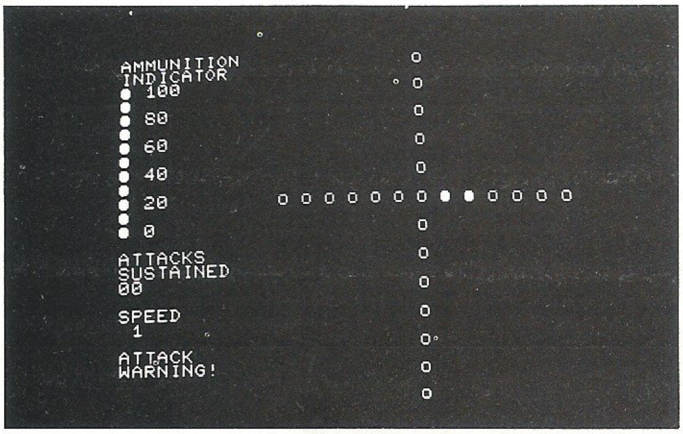
In the original version, the number of shots fired at an incoming attacker was determined by the length of time you held the button down. As the GET command will only input one character, the ammunition provided is just sufficient to beat off the 100 attacks.

HOW TO PLAY

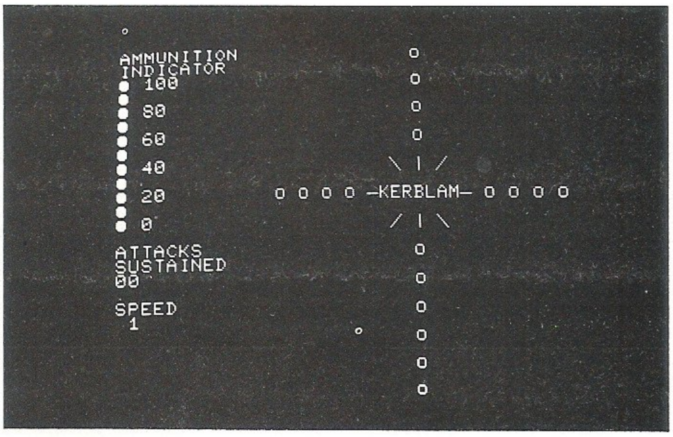
Attacks from the rear and sides are heralded by a warning message on the screen, attacks from the front are 'silent'. The missiles must be fired by pressing the key corresponding to the direction of the attack; 8 for a frontal attack, 4 for the left side, 6 for the right side and 2 for the rear.

PROGRAM CONVERSION

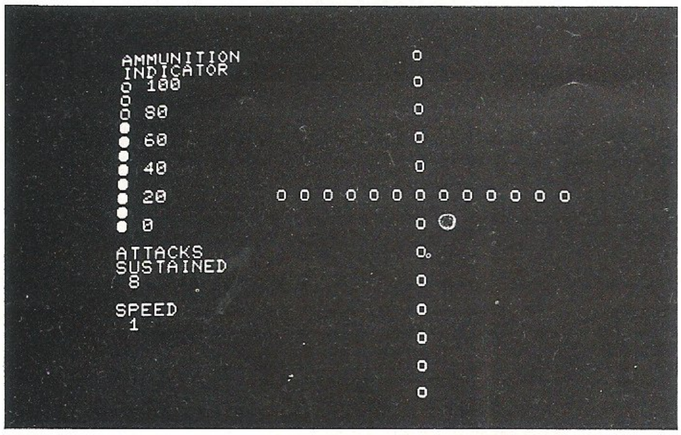
The game is written for the Commodore PET, although it uses none of the special features other than the memory mapped screen and PEEK and POKE statements. Conversion to other systems should be relatively simple provided they possess both these features. The cursor controls are used simply to provide the PET with a pseudo PRINT AT function and can be replaced on many other systems by that command (Sharp owners can use the CURSOR function).



```
AMMUNITION
INDICATOR
 100
 80
 60
 40
 20
 0
ATTACKS
SUSTAINED
00
SPEED
 1
ATTACK
WARNING!
```



```
AMMUNITION
INDICATOR
 100
 80
 60
 40
 20
 0
ATTACKS
SUSTAINED
00
SPEED
 1
```



```
AMMUNITION
INDICATOR
 100
 80
 60
 40
 20
 0
ATTACKS
SUSTAINED
 8
SPEED
 1
```

Above left: The game has just begun and already an attacker approaches from your left. The message 'ATTACK WARNING!' appears at the bottom left-hand corner of the screen — take care though, you won't get this message for frontal attacks!

Above: Too late, the attacker has reached its target and the game is lost.

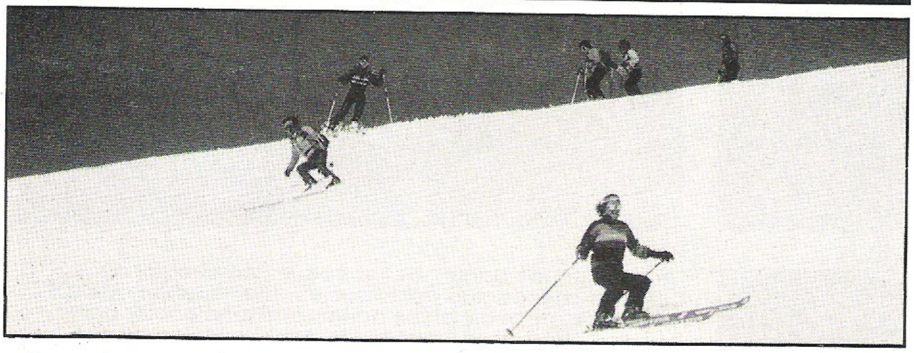
Left: Well,done, you have sustained eight attackers so far. Notice that the Ammunition Indicator at the top left of the screen has been decremented.

```
100  DIM M(4,3)
110  DATA 32793,Ø,+80,33205,Ø,-2,33753,Ø,-80,33181,Ø,+2
120  FOR I=1 TO 4
130  FOR J=1 TO 3
140  READ M(I,J)
150  NEXT J
160  NEXT I
170  INPUT "[CLS]SET SPEED (1 TO 9)";S
180  IF S<1 OR S>9 THEN 17Ø
190  SS=INT(1ØØ/S)
200  PRINT "[CLS]"
210  PRINT "AMMUNITION[15 SPC][^W]"
220  PRINT "INDICATOR"
230  PRINT "[^Q] 1ØØ[2Ø SPC][^W]"
240  PRINT "[^Q]"
250  PRINT "[^Q] 8Ø[21 SPC][^W]"
260  PRINT "[^Q]"
270  PRINT "[^Q] 6Ø[21 SPC][^W]"
280  PRINT "[^Q]"
290  PRINT "[^Q] 4Ø[21 SPC][^W]"
300  PRINT "[^Q]"
310  PRINT "[^Q] 2Ø[9 SPC][13^W][SPC]"
320  PRINT "[^Q]"
330  PRINT "[^Q] Ø[22 SPC][^W]"
340  PRINT
350  PRINT "ATTACKS[18 SPC][^W]"
360  PRINT "SUSTAINED"
370  PRINT "ØØ[23 SPC][^W]"
380  PRINT
390  PRINT "SPEED[2Ø SPC][^W]"
400  PRINT S
410  PRINT "[25 SPC][^W]"
420  PRINT
430  PRINT "[25 SPC][^W]"
440  PRINT
450  PRINT "[25 SPC][^W]";
460  H=Ø
470  AA=Ø
480  IF RND(TI)>Ø.994 THEN 5ØØ
490  GOTO 48Ø
500  AX=Ø
510  X=RND(TI)
520  IF X<=Ø.25 THEN D=1
530  IF X>Ø.25 AND X<=Ø.5 THEN D=2
540  IF X>Ø.5 AND X<=Ø.75 THEN D=3
550  IF X>=Ø.75 THEN D=4
560  POKE M(D,1),81
570  IF D>1 THEN PRINT "[HOM][21 CD]ATTACK[CD][6 CL]
     WARNING!";
580  M(D,2)=M(D,1)
590  IF AA=1ØØ THEN 71Ø
600  GET A$
610  IF A$="" THEN 71Ø
620  IF A$="8" THEN AX=1
630  IF A$="6" THEN AX=2
640  IF A$="2" THEN AX=3
650  IF A$="4" THEN AX=4
660  AA=AA+1
670  PRINT "[HOM]"
680  FOR AB=1 TO 1+INT(AA/1Ø):PRINT "[CD]";:NEXT AB
690  PRINT "[^W]";
700  IF AX=D THEN 77Ø
710  IF M(D,2)=33193 THEN 84Ø
720  POKE M(D,2),87
730  POKE M(D,2)+M(D,3),81
740  M(D,2)=M(D,2)+M(D,3)
750  FOR SA=1 TO SS:NEXT SA
760  GOTO 6ØØ
770  POKE M(D,2),42
780  H=H+1
790  PRINT "[HOM][16 CD]]";H;
800  POKE M(D,2),87
810  IF H=1ØØ THEN 9ØØ
820  PRINT "[HOM][21 CD][6 SPC][CD][6 CL][7 SPC]";
830  GOTO 48Ø
840  POKE 3319Ø,11:POKE 33191,5:POKE 33192,18:
     POKE 33193,2:POKE 33194,12
850  POKE 33195,1:POKE 33196,13
860  POKE 33111,77:POKE 33113,93:POKE 33115,78:
     POKE 33189,64
870  POKE 33197,64:POKE 33271,78:POKE 33273,93:
     POKE 33275,77
880  FOR XX=1 TO 1ØØØ:NEXT XX
890  GOTO 17Ø
900  FOR I=1 TO 5ØØ:NEXT I
910  PRINT "[CLS]CONGRATULATIONS! - YOU HAVE WIPED OUT"
920  PRINT "THE ENTIRE YAPPANIE SUICIDE SQUAD AND"
930  PRINT "SAVED THE CEETEE FROM DESTRUCTION"
940  PRINT
950  PRINT "ARE YOU FEELING FIT ENOUGH FOR"
960  PRINT "ANOTHER MISSION?"
970  GET A$
980  IF A$="" THEN 97Ø
990  IF A$="Y" THEN 2ØØ
1000 END
```

Christopher Hales.

SKI RUN

Learn to ski, the UK101 way!



Ski Run is an interactive graphics game for the UK101. However, as the program is written in BASIC,it should be easily adaptable to other machines.

The VDU screen is dotted with numerous trees and the player moves a skier from the top left to the bottom right of the screen towards his 'house'. The screen represents a snowy slope and so if the player does not press any buttons the skier will move downwards. The player has two keys, the 'Q' and 'P' keys, which will move the skier left or right — but whenever no key is pressed the skier will move down the screen. The player has to manoeuvre the skier through the gaps in the trees to the character space occupied by his house in the lower right corner.

If the skier hits a tree he has an accident, of course, so you must start again. Before the run starts the player chooses the speed the skier moves at — from 5 (very slow) to 0 (very fast), with any value in between being available (ie not just integer values). If the skier goes off the bottom of the screen he reappears the same distance across at the top of the screen and then makes his second 'run'. When the skier reaches the space occupied by the house, a flag goes up on the house and the number of runs and the speed is given.

TECHNICAL DETAILS

This version works for a portable TV screen giving a

PROGRAM STRUCTURE

Statement	Function	Action
Lines 10-40	Inputs	Instructions and skiing speed inputs.
Line 50	Clears Screen	Clears the screen.
Lines 60-90	Position Trees	Put tree characters on 125 random screen character slots.
Lines 100-120	Position Skier	Put skier in top left corner and clear the space under him, put house in lower right corner and initialise runs variable to 1.
Line 130	Delay	Slight delay before skier moves.
Line 140	Disables Control C	Disables 'Control C' — necessary for disabling polled keyboard.
Lines 150-190	Position Store	Store previous skier position, disable normal keyboard polling routine and test for P or Q keys being pressed. Change skier's screen reference.
Lines 200-210	Hit Routines	Go to routines for if skier hits a tree or reaches the house.
Line 220	Movement	Moves skier.
Line 230	Delay	Gives the delay which alters speed.
Line 240	Routine Jump	If skier goes off screen at bottom, goes to routine to put him back.
Lines 330-340	Crash	Skier hits tree: puts up a crash character and gives relevant comments.
Lines 400-460	Home Flag	Skier reaches home: puts a flag above house and gives relevant comments.
Lines 470-480	Re-Start	Ask for another game.
Line 490	Enables Control C	Enables 'Control C', end.
Lines 500-530	Off Screen Routine	If skier goes off bottom, returns him directly above on top line screen, removing a tree if this puts him on top of one.
Lines 600-750	Instructions	Game instructions.
Line 760	End	Ends.

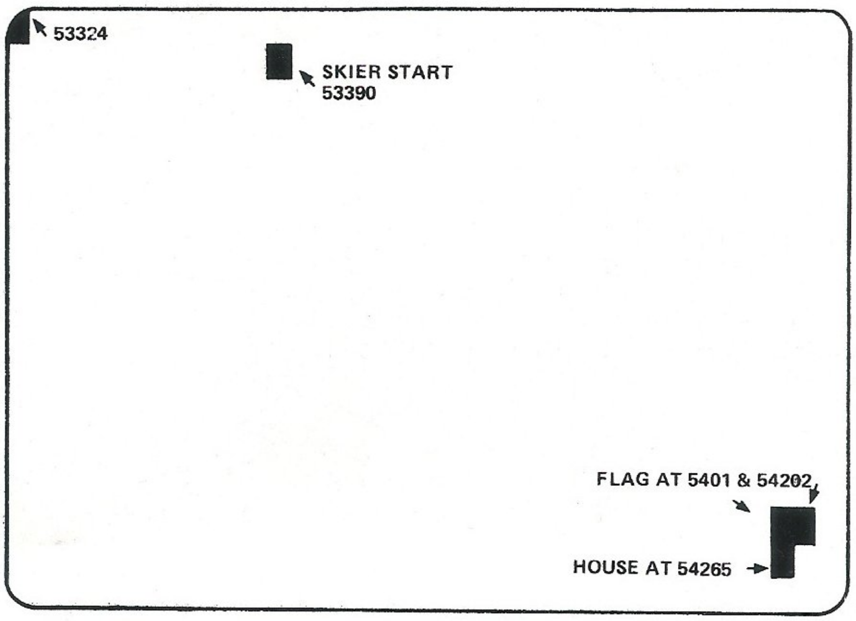
width of 47 characters and a depth of 16 lines. The RAM values given with the POKE function refer to the screen positions illustrated in the diagram on the right. (NB 54278 comes after the last line on the screen and is used to check if the skier goes off the bottom.)

The ASCII characters used are:

4	an explosion-type character
13	tree (but on my computer this was not accessible by the CHR$ function)
15	house
32	space
143, 151	a horizontal rectangle and vertical line to give a flag
240	a man

Here are some other notes on the UK 101 BASIC:

POKE 530, 1 and POKE 530,0



disable and enable the 'Control C' key so that it will not intrude on a region, enabling control of the keyboard to be obtained.

POKE 57088,RA and IF PEEK (57088)=CA THEN. . .are used to alter key functions given the row address (RA) and column address (CA) of the keys involved. The polling routine will respond to only one key being down at any time, given the same row address.

RND(X) for any argument always returns a random number between 0 and 1 — spaces are not necessary.

The best result yet seen is a success at level 0.15 in one run (after hours of trying). This is a suggested classification of the levels:

5	EASY
4	QUITE EASY
3	AVERAGE
2	QUITE HARD
1	HARD
0	ALMOST IMPOSSIBLE

But, of course, you can have any intermediate level.

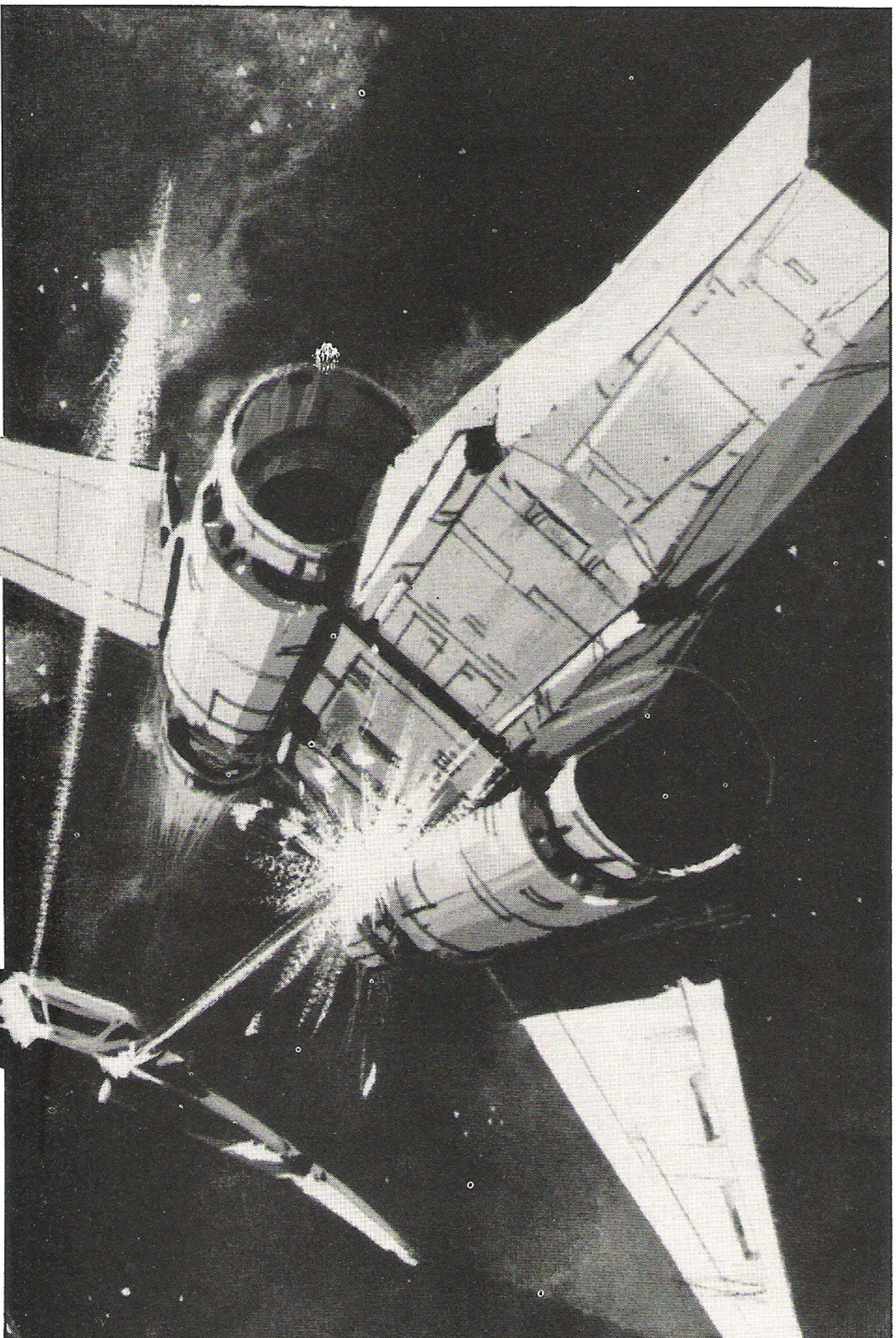
Possible modifications are to have only one key, moving right; to alter the range of speeds; to allow only one run.

```
 10  INPUT "DO YOU NEED INSTRUCTIONS";I$
 20  IF LEFT$(I$,1)="Y" THEN 610
 30  INPUT "WHAT IS YOUR SKIING SPEED (0-5)";K
 40  IF K<0 OR K>5 THEN 30
 50  FOR LINE=1 TO 16:PRINT:NEXT LINE
 60  FOR TREE=1 TO 125
 70  P=53324+INT(50*RND(1))+64*INT(17*(RND(1))
 80  POKE P,13
 90  NEXT TREE
100  R=53390:J=1
110  POKE R,240:POKE R+64,32
120  POKE 54265,15
130  FOR T=1 TO 700:NEXT T
140  POKE 530,1
150  PRE=R
160  POKE 57088,253:M=PEEK(57088)
170  IF M=127 THEN R=R-1:GOTO 200
180  IF M=253 THEN R=R+1:GOTO 200
190  R=R+64
200  IF PEEK(R)=13 THEN 310
210  IF PEEK(R)=15 THEN 410
220  POKE PRE,32:POKE R,240
230  FOR Y=1 TO K*100:NEXT Y
240  IF R>54278 THEN POKE R,32:GOTO 510
250  GOTO 150
300  REM ** CRASH ROUTINE
310  POKE PRE,32:POKE R,4
320  PRINT "YOU HAVE JUST HAD AN ACCIDENT.."
330  PRINT "WHEN YOU RECOVER WOULD YOU LIKE"
340  GOTO 470
400  REM ** WIN ROUTINE
410  POKE PRE,31
420  POKE 54201,143:POKE 54202,151
430  PRINT "WELL DONE...YOU JUST MADE IT IN"
440  PRINT "TIME FOR YOUR TEA!!"
450  PRINT "IT TOOK YOU "J" RUNS DOWN THE SLOPE"
460  PRINT "AND YOUR SPEED LEVEL WAS "K
470  INPUT "ANOTHER GAME...";A$
480  IF LEFT$(A$,1)<>"N" THEN 30
490  POKE 530,0:END
500  REM ** NEW RUN
510  R=R-960:J=J+1
520  IF PEEK(R)=13 THEN POKE R,32
530  GOTO 220
600  REM ** INSTRUCTIONS
610  PRINT "[9 SPC]** SKI RUN **":PRINT
620  PRINT "YOU ARE AT THE TOP OF A SNOWY HILL"
630  PRINT "WHICH IS DOTTED WITH TREES"
640  PRINT "YOU START AT THE TOP LEFT CORNER OF"
650  PRINT "THE SCREEN AND YOU HAVE TO GET TO"
660  PRINT "YOUR HOME AT THE BOTTOM RIGHT"
670  PRINT
680  PRINT "TO GO LEFT PRESS THE 'Q' KEY"
690  PRINT "TO GO RIGHT PRESS THE 'P' KEY"
700  PRINT
710  PRINT "IF NO KEY IS PRESSED YOU WILL MOVE"
720  PRINT "VERTICALLY DOWNWARDS..."
730  PRINT "PRESS ONLY ONE KEY AT A TIME"
740  INPUT "PRESS 'Y' AND RETURN TO CONTINUE";B$
750  IF LEFT$(B$,1)="Y" THEN 30
760  GOTO 490
```

S Battle

NAS WARS

Raid the stars with your NASCOM 1.



This program was written for an 8K NASCOM 1 using the T4 monitor with the 8K ROM BASIC and the NAS-GRA-V3 graphics ROM. With suitable modification it would be possible to run this program under NAS-SYS.

You sit at the controls of a rebel Star-fighter. Your mission is to seek out and destroy as many rogue Eti-fighters as possible within the confines of the energy reserves available.

To steer your fighter the following four keys on the NASCOM keyboard are used:

—	. . . UP
@	. . . LEFT
BACKSPACE	. . . RIGHT
NEWLINE	. . . DOWN

Only one key may be pressed at any one time until you fire the LASER which is activated by depressing all four keys at once. It must be remembered that you are steering towards the Eti-fighter and it may seem, at first, to be back to front!

A double bar at each end of the sight indicates that a target is present within the sight, or that the LASER is recharging. Angled brackets signify that the target is central within the sight.

Mission Control

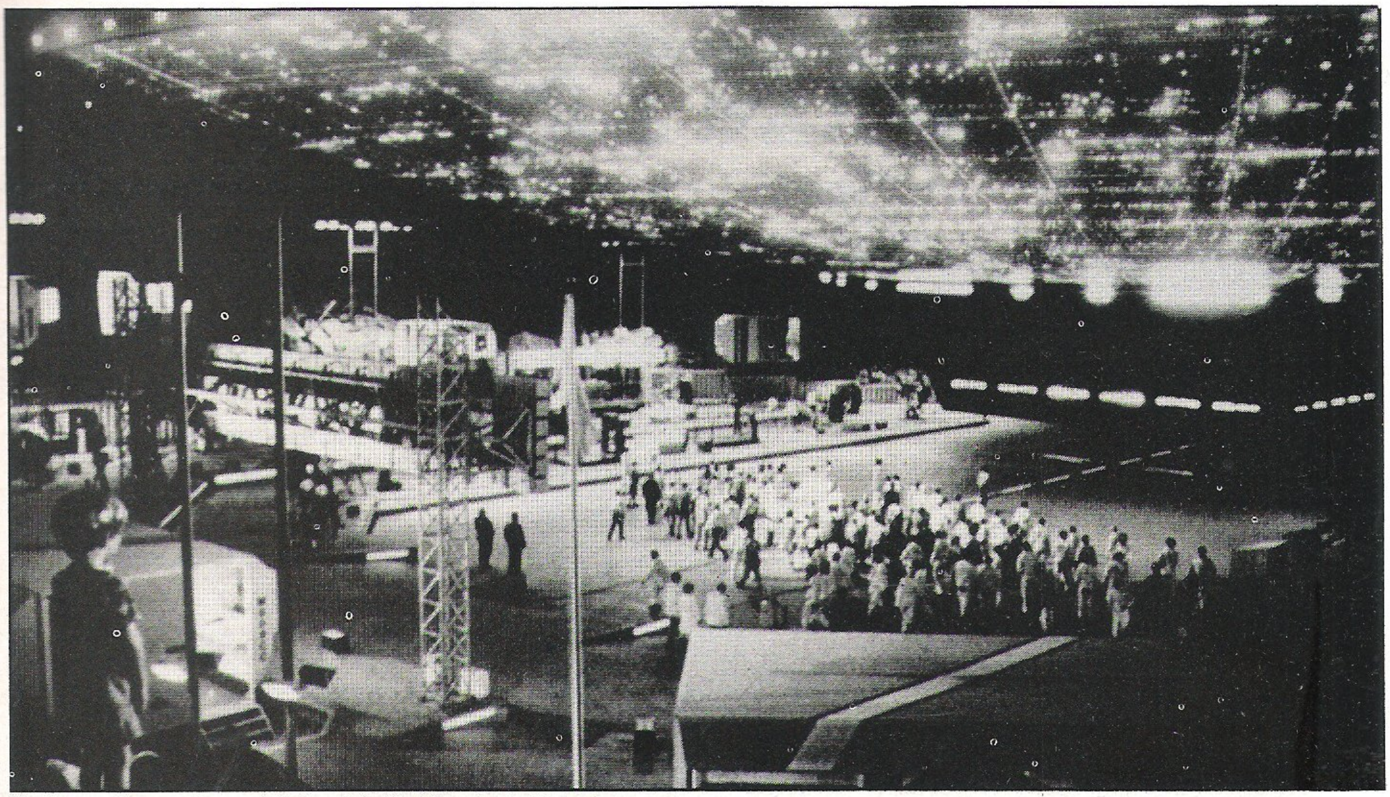
You start out on your mission with 500 units of energy, on each burst of LASER fire you consume 10 units of energy.

A score of 10 points is awarded for a hit on the main hull of the Eti-fighter, whereupon a hopefully satisfying explosion will ensue. To gain additional marks, a point is given for every stabilising fin which is shot off. There are four fins in all, two per wing, giving a possible score of 14.

When your energy has been exhausted, an appropriate comment will be printed, this being dependent on your performance. If your score has exceeded the best recorded score, the program will invite you to enter your name which will be displayed beside your score until this, in turn, is exceeded.

SCOPE FOR IMPROVEMENT

With only minor modifications to the subroutine at line 7000 onwards, it would be possible to use this program with a Joystick interface, perhaps even with a trigger to fire the LASER (Line 170).



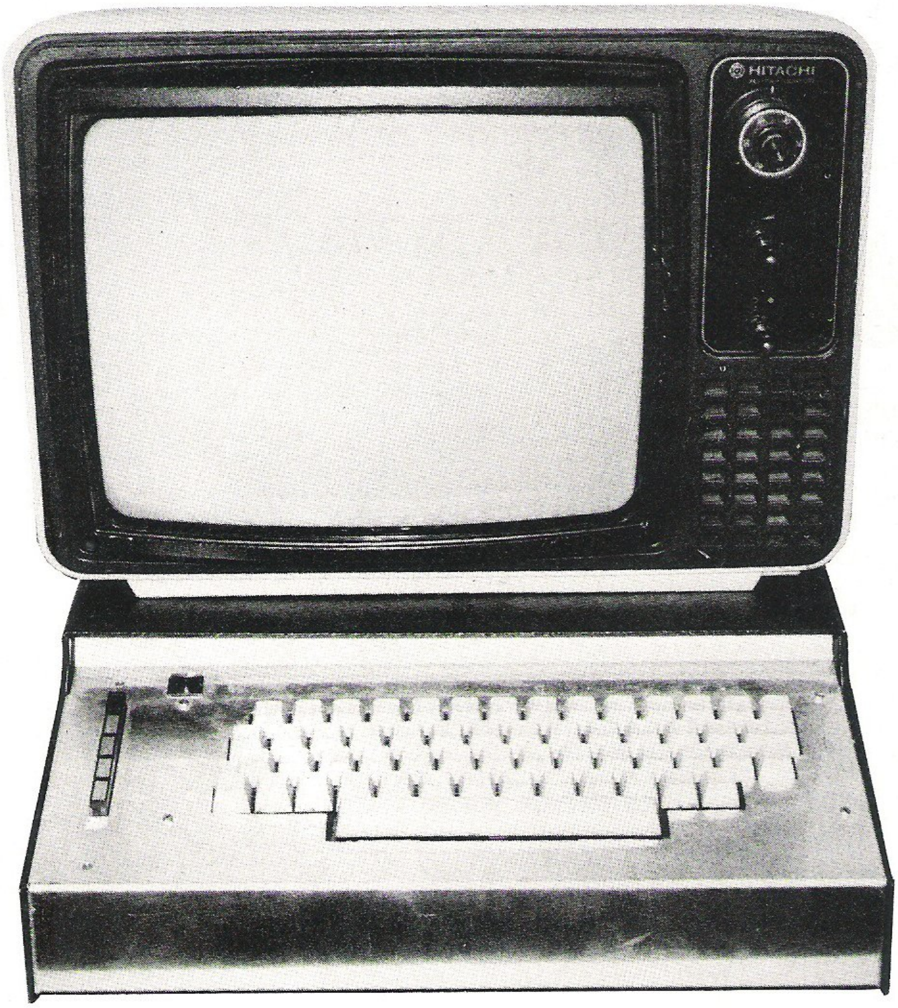
Photograph courtesy of Columbia Pictures Industries, Inc.

```
  60  WIDTH 255
  70  INPUT "Enter previous highest score ";BEST
  80  INPUT "By whom ";NAME$
  90  GOSUB 3ØØØ:REM ** CONVERT NAME TO LOWER CASE
 1ØØ  MARK=Ø:ENERGY=5ØØ:CLS
 11Ø  LEFT=149:RIGHT=151
 12Ø  GOSUB 1ØØØ:REM ** PRINT OUT BEST SCORE
 13Ø  GOSUB 2ØØØ:REM ** PRINT OUT SCORE AND ENERGY
 14Ø  X=INT(RND(1)*46+2):Y=INT(RND(1)*15+1)
 15Ø  GOSUB 4ØØØ:REM ** DRAW SIGHT
 16Ø  GOSUB 5ØØØ:REM ** DRAW FIGHTER
 17Ø  IF INP(Ø)=152 THEN GOSUB 6ØØØ:REM ** FIRE LASER
 18Ø  IF ENERGY=Ø THEN 22Ø
 19Ø  IF E=1 THEN E=Ø:GOTO 11Ø
 2ØØ  GOSUB 7ØØØ:REM ** MOVE FIGHTER
 21Ø  GOTO 15Ø
 22Ø  F=Ø:IF MARK>BEST THEN BEST=MARK:F=1
 23Ø  IF MARK>1ØØ THEN MARK=1ØØ
 24Ø  RESTORE:SCREEN 1,15
 25Ø  FOR C=1 TO MARK/2Ø:READ COMMENT$:NEXT C
 26Ø  PRINT COMMENT$
 27Ø  IF F=1 THEN INPUT "What is your name ";NAME$
 28Ø  INPUT "Another game?";COMMENT$
 29Ø  IF LEFT$(COMMENT$,1)="N" THEN END
 3ØØ  IF F=1 THEN GOSUB 3ØØØ:REM ** LOWER CASE
 31Ø  IF F=1 THEN PRINT "May the Force be with you"
 32Ø  FOR C=1 TO 2ØØØ:NEXT C:GOTO 1ØØ
1ØØØ  SCREEN 24,16
1Ø1Ø  PRINT "Best"BEST" by "NAME$;:RETURN
2ØØØ  SCREEN 1,16
2Ø1Ø  PRINT "Score"MARK" Energy"ENERGY";:RETURN
3ØØØ  IF LEN(NAME$)<2 THEN RETURN
3Ø1Ø  TEMP$=MID$(NAME$,2,1)
3Ø2Ø  IF TEMP$<"A" OR TEMP$>"Z" THEN RETURN
3Ø3Ø  TEMP$=LEFT$(NAME$,1)
3Ø4Ø  FOR L=2 TO LEN(NAME$)
3Ø5Ø  IF MID$(NAME$,L,1)="[SPC]" THEN 3Ø8Ø
3Ø6Ø  TEMP$=TEMP$+CHR$(32+ASC(MID$(NAME$,L,1)))
3Ø7Ø  NEXT L
3Ø8Ø  NAME$=TEMP$
3Ø9Ø  RETURN
4ØØØ  REM ** DRAW SIGHT
4Ø1Ø  POKE 24Ø2,154:POKE 2658,153
4Ø2Ø  POKE 2533,151:POKE 2527,149
4Ø3Ø  M=PEEK(253Ø):IF M=21Ø OR E=1 THEN 4Ø7Ø
4Ø4Ø  IF PEEK(2661)=147 THEN POKE 2661,32
4Ø5Ø  POKE 2399,32:POKE 24Ø5,32:POKE 2655,32
4Ø6Ø  GOTO 4Ø9Ø
4Ø7Ø  IF PEEK(2661)<>13Ø THEN POKE 2661,147
4Ø8Ø  POKE 2655,146:POKE 2399,144:POKE 24Ø5,145
4Ø9Ø  IF M<>32 THEN M=148
41ØØ  POKE 2526,M:POKE 2534,M:RETURN
5ØØØ  REM ** DRAW FIGHTER
5Ø1Ø  POKE Z-1,32:POKE Z+1,32:POKE Z,32
5Ø2Ø  X=X+X1:Y=Y+Y1:X1=Ø:Y1=Ø
5Ø3Ø  REM ** KEEP FIGHTER ON THE SCREEN
5Ø4Ø  IF X>47 THEN X=47
5Ø5Ø  IF X<2 THEN X=2
5Ø6Ø  IF Y<1 THEN Y=1
5Ø7Ø  IF Y>15 THEN Y=15
5Ø8Ø  Z=1993+X+64*Y
5Ø9Ø  POKE Z,21Ø:POKE Z-1,LEFT:POKE Z+1,RIGHT
51ØØ  RETURN
6ØØØ  REM ** FIRE LASER
6Ø1Ø  M=PEEK(253Ø):ENERGY=ENERGY-1Ø
6Ø2Ø  E=Ø:IF M=21Ø THEN E=1
6Ø3Ø  REM ** DAMAGE FIGHTER
6Ø4Ø  IF M=146 OR M=144 THEN LEFT=152:MARK=MARK+1
6Ø5Ø  IF M=147 OR M=145 THEN RIGHT=152:MARK=MARK+1
6Ø6Ø  IF M=149 THEN MARK=MARK+1:LEFT=146:IF RND(1)<Ø.5
      THEN LEFT=144
6Ø7Ø  IF M=151 THEN MARK=MARK+1:RIGHT=147:IF RND(1)<Ø.5
      THEN RIGHT=145
6Ø8Ø  FOR L=1 TO 2:C=2986
6Ø9Ø  FOR R=2971 TO 253Ø STEP-63
61ØØ  POKE R,32:IF L=1 THEN POKE R,131
611Ø  POKE C,32:IF L=1 THEN POKE C,13Ø
612Ø  C=C-65:NEXT R
613Ø  GOSUB 4ØØØ:REM ** DRAW SIGHT
614Ø  IF E=Ø THEN GOSUB 7ØØØ:GOSUB 5ØØØ
615Ø  NEXT L
616Ø  GOSUB 4ØØØ:REM ** DRAW SIGHT
617Ø  IF E=Ø THEN GOSUB 2ØØØ:RETURN
618Ø  MARK=MARK+1Ø
619Ø  REM ** EXPLOSION
62ØØ  GOSUB 5ØØØ:REM ** DRAW FIGHTER
621Ø  GOSUB 2ØØØ:REM ** PRINT SCORE
622Ø  FOR L=1 TO 2
623Ø  FOR R=1 TO 4
624Ø  FOR C=Ø TO 6.28 STEP Ø.78
625Ø  X=49+R*SIN(C):Y=22+R*COS(C)
626Ø  RESET(X,Y):IF L=1 THEN SET(X,Y)
627Ø  NEXT C:GOSUB 4ØØØ:REM ** DRAW SIGHT
628Ø  NEXT R:NEXT L:RETURN
7ØØØ  REM ** MOVE FIGHTER
7Ø1Ø  I=INP(Ø)
7Ø2Ø  X1=INT(RND(1)*3-1):Y1=INT(RND(1)*3-1)
7Ø3Ø  IF I=159 THEN X1=1:REM ** LEFT
7Ø4Ø  IF I=19Ø THEN X1=-1:REM ** RIGHT
7Ø5Ø  IF I=187 THEN Y1=1:REM ** TOP
7Ø6Ø  IF I=189 THEN Y1=-1:REM ** BOTTOM
7Ø7Ø  RETURN
8ØØØ  DATA You have failed you miserable dog...
8Ø1Ø  DATA Your humble attack was of little consequence.
8Ø2Ø  DATA The Empire continues its reign of terror.
8Ø3Ø  DATA Well done...You shot down an entire squadron.
8Ø4Ø  DATA Congratulations...You have defeated the
      Empire.
```

Don Scales

LABYRINTH

The original 3D maze game from Computing Today.



Labyrinth is a fairly large program written in Tiny BASIC. Each time the program is run, it will construct a different two-dimensional maze and then allow the player to explore a three-dimensional projection of this maze.

The program is divided roughly into two halves. The first half randomly builds a maze with a single route through it. A 2D plot of the maze is available at the end of this stage for those who suffer from claustrophobia. The second half of the program produces 3D projections as the player wanders along the corridors of the maze.

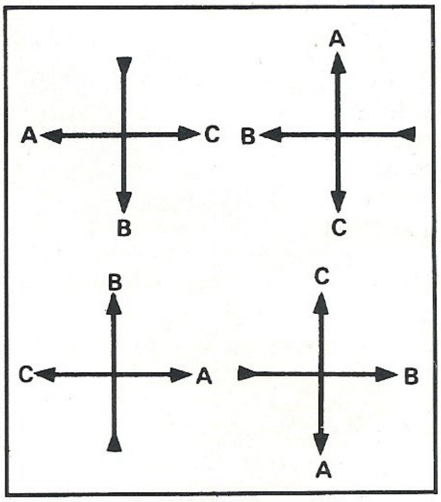
Building The Maze

The basic maze is a 'simple connected' maze (one which has no closed circuits). It is constructed using two two-dimensional arrays. The first array holds an indication of which cells of the maze have been used and the order in which they have been allocated. The second array holds the description of the topology of the maze.

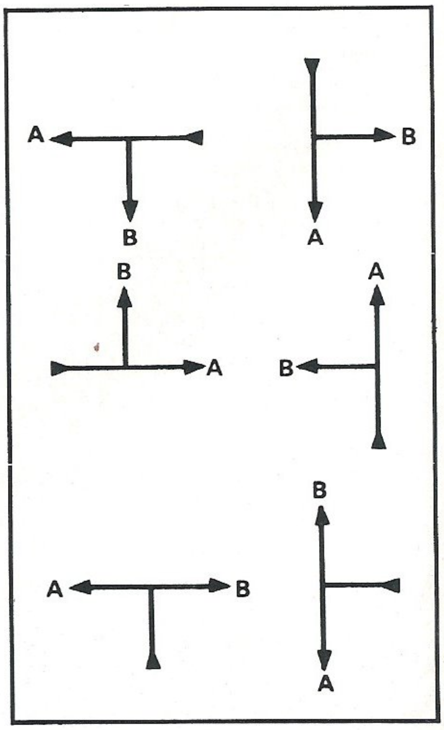
The maze construction starts by randomly selecting an entrance along the width of the maze. This location is saved in a spare element of the array.

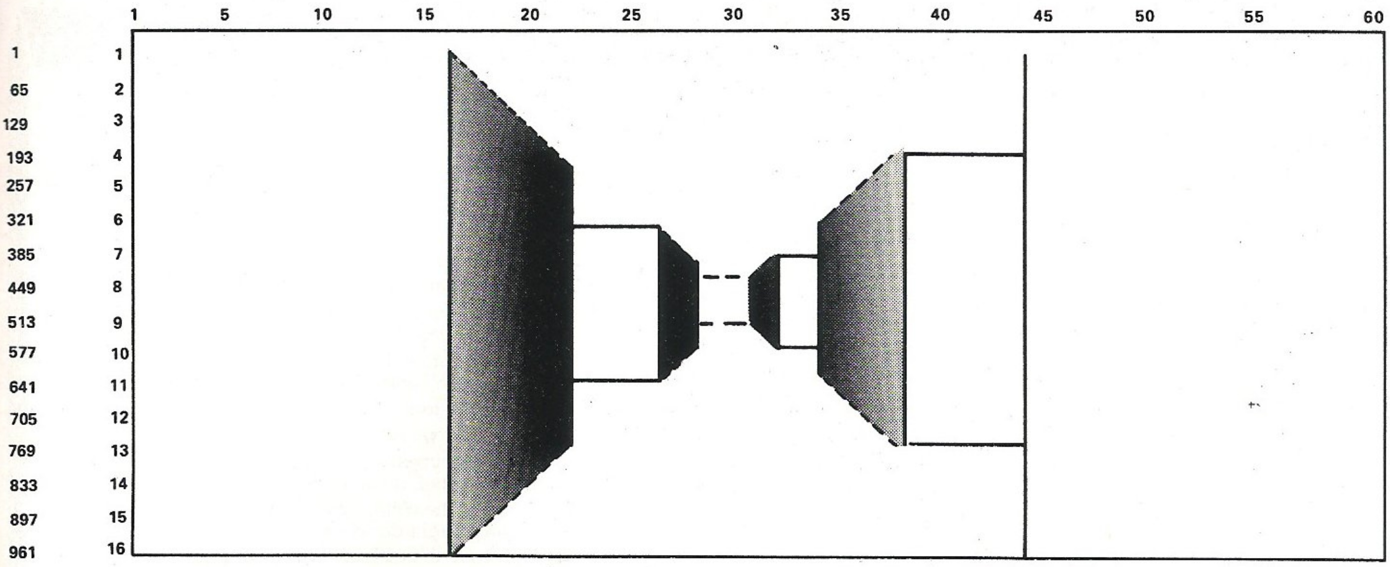
From this start location the maze is constructed. At each cell, the program scans the adjacent cells to see which are available to use. Having decided which are available, the program then selects one cell randomly.

Consider the following examples. In each of these four there are three possible choices, A, B and C:



Hence the route can be chosen from the three possibilities. Next there are six combinations of two choices:





To arrive at these choices, the program must first scan the adjacent cells. As the program knows the direction it has just come from, it only needs to check the other three directions.

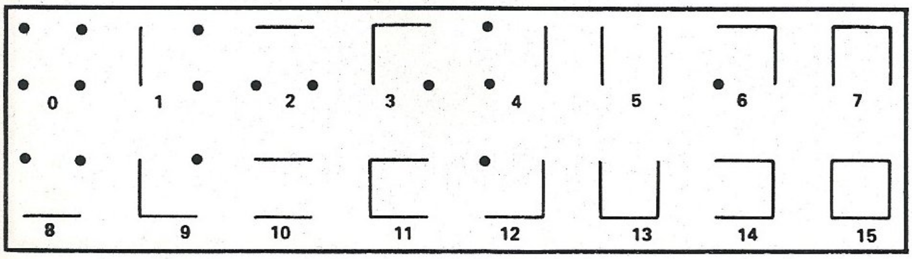
The program continues its random route through the maze until it hits a dead end. A branch is then made from the first route at this point and continued until the next dead end. This procedure is continued until the maze is complete.

At this point, the player can obtain a two-dimensional display of the maze. Each element of the second array contains information about one cell of the maze. This information is incomplete as it is only for the top and right-hand wall:

0=·˥   1=:| 2=∸   3=: :

The Third Dimension

To produce a three-dimensional picture, it is necessary to complete the cell information and organize it in such a manner that it can be rotated. The binary system fulfils both these requirements. A bit is used to indicated a wall.



To turn left, the cell information is cyclically shifted right one bit. 2 becomes 4, 3 becomes 6 and 8 becomes 1. To turn right, the cell information is cyclically shifted left one bit. 2 becomes 1, 1 becomes 8 and 10 becomes 5.

The information for the 2D maze is therefore translated and the information completed by inspecting the neighbouring cells. The 3D pictures are produced using memory mapping and the graphics available on the Triton.

The display is constructed simply with horizontal, vertical and diagonal lines. A reasonable display would be possible with | — and / \. To move in the maze, the player can turn left or right or move forward. The player's current position can also be obtained.

Giving The Picture

To produce the 3D picture, the program starts with the cell corresponding to the player's current position. This cell is then rotated, as described earlier, until it faces the same way as the player. The program then decodes the cell information and checks for the walls left, right and in front of the player. At the first depth, either a blank wall or two columns are produced. If a blank wall is produced, no further information is available. If looking out of the maze, no further information is produced and if outside the maze and looking away from it, a blank screen is all you get.

If, on the other hand, a passage exists to the next cell, the program obtains the information about the next cell by making the appropriate index and rotates and decodes this cell. At the second depth, it is possible to have walls or passages to the left, right and straight ahead.

Each depth has its own display routine which checks for and plots the three walls or passages. Each depth produces a display continuing from the previous and maintains the perspective. The display stops either with a blank wall or when depth 5 has been reached.

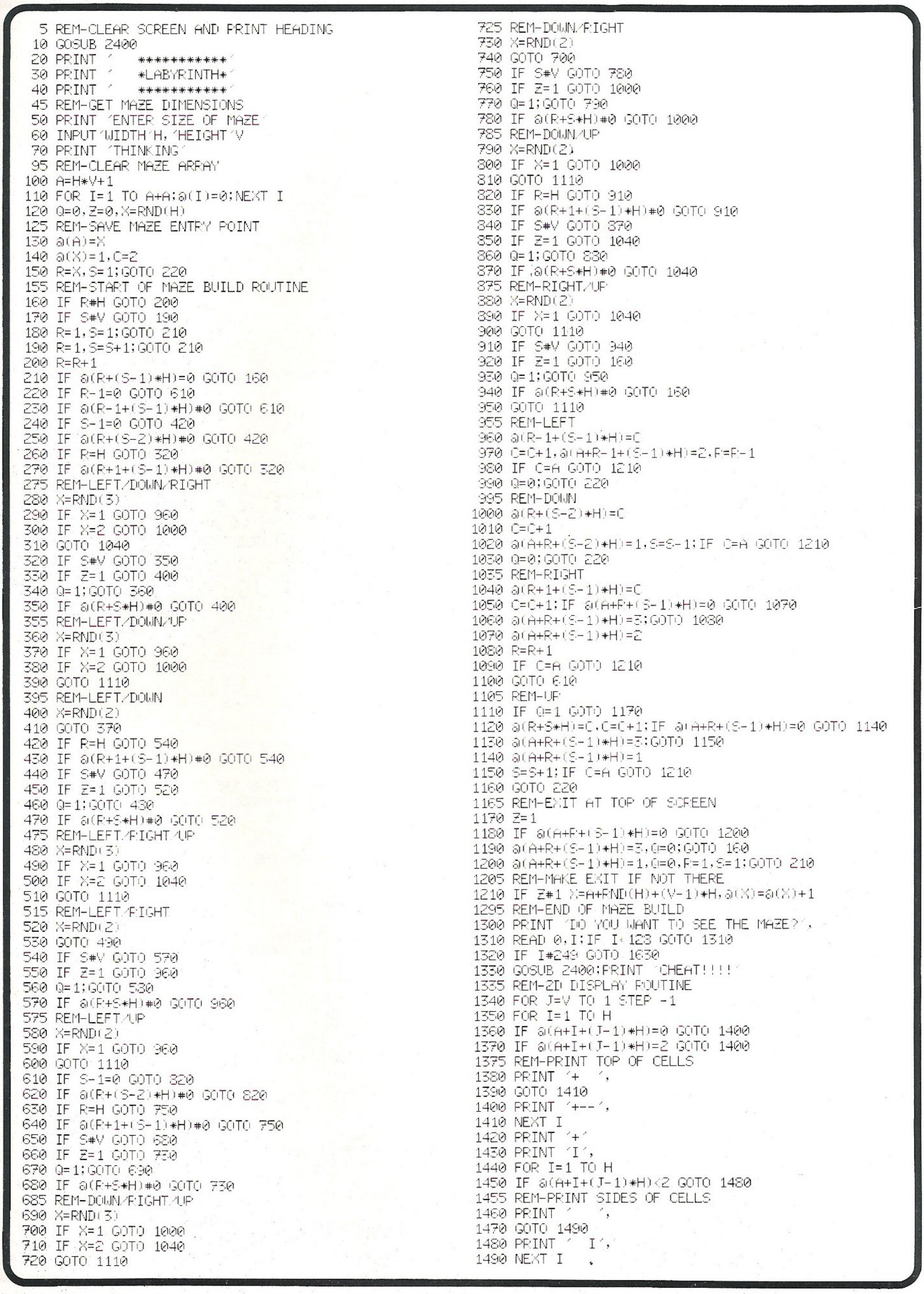
The program listing following contains the full Tiny BASIC commands and is commented to make it easier to follow and to translate. If using a floating point BASIC, take great care in the rotate and decode routines as they rely on integer rounding effects. A large number of INT commands will be required!

The program will fit on a Triton with mother board and an extra 8K of RAM but the Tiny BASIC commands should be abbreviated for size and speed reasons.

▶

PROGRAM STRUCTURE

Statement	Action
Lines 5-40	Clear Screen and print heading.
Lines 45-70	Ask for size of maze.
Lines 95-120	Clear arrays used to construct the maze and initialize variables. Obtain random entry point.
Lines 125-150	Save entry point and start the maze.
Lines 155-1295	Maze build routine.
Lines 155-200	Find the next starting point when a route comes to a dead end.
Lines 210-270	Do an initial check on the number of allowable routes from the current position in the maze.
Lines 275-310	Randomly select Left, Down or Right as the next route.
Lines 320-350	More route checking. Z= 1 when an exit exists.
Lines 355-390	Randomly select Left, Down or Up.
Lines 395-410	Use when exit already exists or no way up. Randomly select Left or Down.
Lines 420-470	Move route checking.
Lines 475-510	Randomly select Left, Right or Up.
Lines 515-530	No way up. Randomly select Left or Right.
Lines 540-570	Move route checking.
Lines 575-600	Randomly select Left or Up.
Lines 610-680	Move route checking.
Lines 685-720	Randomly select Down, Right or Up.
Lines 725-740	No way up, select Down or Right.
Lines 750-780	Yet more route checking.
Lines 785-810	Just Down or Up.
Lines 820-870	Not much more route to check.
Lines 875-900	Right or Up.
Lines 910-950	Last bit of route checking.
Lines 955-990	Set up maze for route to go Left. Check if maze finished, if not, see where it goes next.
Lines 995-1030	Route goes Down.
Lines 1035-1100	Route goes Right.
Lines 1105-1160	Route goes Up. Checks if exit made.
Lines 1165-1200	Make exit at top, loop back if maze not complete.
Lines 1205-1210	Make sure maze has an exit.
Lines 1300-1320	Keyboard scan to see if 2D print required. READ 0, I scans a byte from the keyboard on the Triton. Substitute INPUT if necessary.
Lines 1330-1570	2D print routine.
Lines 1330-1335	Clear screen and print 'CHEAT'.
Line 1340	Loop for height of maze.
Lines 1350-1420	Print the top of a line of cells checking to see if wall or gap required. To use Triton graphics change + to w and − to s.
Lines 1430-1500	Print the sides of a line of cells, checking to see if wall or gap required. To use Triton graphics change I to t.
Line 1510	End of height loop.
Lines 1520-1570	Print bottom of last row of cells, leaving an entrance.
Lines 1595-1620	Reset cursor to top of screen and loop on the keyboard until a key is pressed. Again, INPUT can be substituted.
Lines 1625-1630	Call the instruction print routine.
Lines 1635-1870	Translate the maze into binary cell information and then give each cell the information about all its walls.
Lines 1635-1670	Translate maze to convenient notation and move into other buffer.
Lines 1710-1870	Take each cell in turn and check with adjacent cells to obtain information about all the walls.
Lines 1875-1890	Set up start parameters and go display entrance to maze in 3D.
Lines 1895-1950	Print instruction for wandering in maze.
Lines 1995-2100	Print helpful information when lost. Note the Λ l and ΛJ which perform a Carriage Return without clearing the screen and a Line Feed.
Lines 2195-2270	Another keyboard scan routine. Routine loops scanning the keyboard until L, R, F or H are pressed. When pressed it jumps to the appropriate routine. No real problem to substitute INPUT.
Lines 2295-2320	Turn Left, then go display new view.
Lines 2345-2370	Turn Right, and go display new view.
Lines 2395-2440	Clear screen and wait while it is cleared. VDU 0, 12 is the Clear Screen command for a Triton.
Lines 2445-2460	Reset cursor to top of screen and wait. VDU 0, 28 is the reset cursor command.
Lines 2495-2540	Routine to space cursor and erase messages.
Lines 2595-2790	Rotate routine.
Lines 2595-2630	Check current position (A,B) and extract cell information if inside maze.
Lines 2635-2660	Rotate the cell infomation if not facing North until facing right direction.
Lines 2670-2700	Decode the cell information into C, D and E. C is Left wall, D is Right wall and E is front wall. If zero no wall and if one, a wall.
Lines 2705-2750	Set up if outside maze but facing retaining wall.
Lines 2755-2790	Set up if in no mans land.
Lines 2795-2850	Index the display to the next cell according to direction faced.
Lines 2855-2920	Position cursor for messages. ΛJ and ΛI perform Line Feed and cursor right commands on the Triton.
Lines 2930-2980	Print error messages when you hit a dead end or no mans land.
Lines 2995-3040	Routine to move the player forward to the next cell.
Lines 3045-4980	3D display routines.
Lines 3045-3060	Set up start position, rotate and look from first cell.
Lines 3065-3080	Set up loop for up to 5 depths and call display routine.
Lines 3085-3140	Check if possible to see into next cell. If so, index to and rotate next cell. Loop to a depth of five unless wall in way. Return to keyboard routine.
Lines 3195-3200	Jump to appropriate depth routine.
Lines 3205-3300	Clear screen and check if facing no mans land.If yes, nothing to display — otherwise display first depth.
Lines 3240-3270	Map vertical lines of walls. Triton screen is 64 wide by 16 high. The screen is numbered left to right, top to bottom from 1 to 1024. VDU I, 116 maps graphic 116 ▥ at the location in I.
Lines 3280-3330	Check for a wall ahead and if so, map top and bottom. Graphic 107 is □ and 108 is ▣ .
Lines 3600-3940	Display second depth.
Lines 3600-3720	Check for left wall or passage and map projection. Graphic 114 is \ , 113 is /.
Lines 3730-3840	Check for right wall or passage and map.
Lines 3850-3880	Map end walls.
Lines 3890-3940	Check for end wall and return if no wall —otherwise map top and bottom.
Lines 4000-4300	Display third depth.
Lines 4400-4620	Display fourth depth.
Lines 4800-4980	Display fifth depth. Graphic 106 is □ and 105 is □ .
Lines 4995-5030	Clear screen and display WAY OUT. End of game.



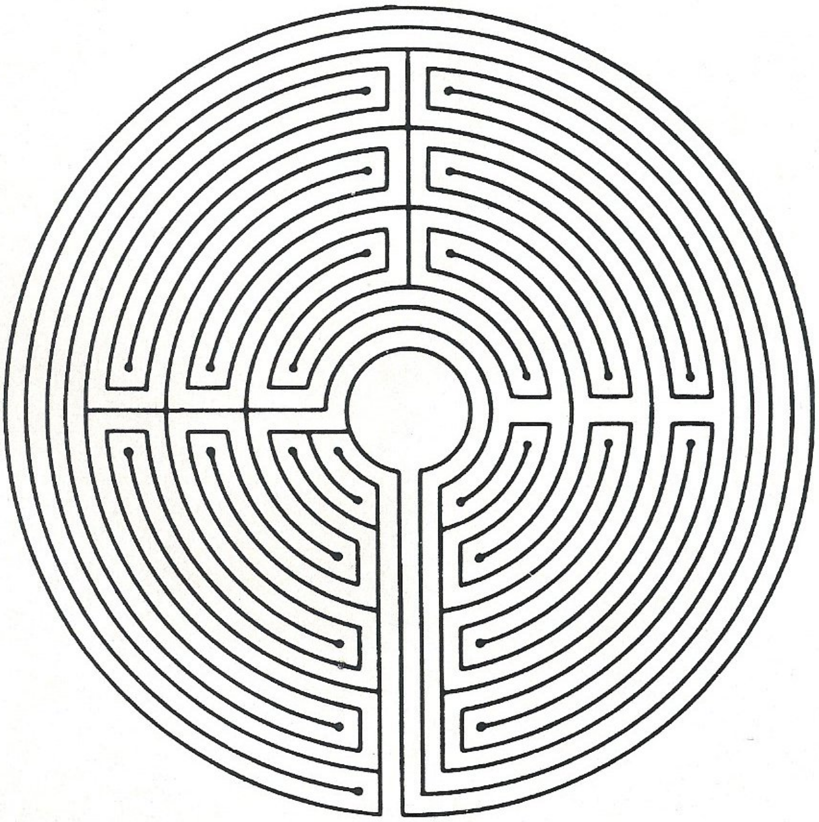
```
   5 REM-CLEAR SCREEN AND PRINT HEADING
  10 GOSUB 2400
  20 PRINT '    ***********'
  30 PRINT '    *LABYRINTH*'
  40 PRINT '    ***********'
  45 REM-GET MAZE DIMENSIONS
  50 PRINT 'ENTER SIZE OF MAZE'
  60 INPUT'WIDTH'H,'HEIGHT'V
  70 PRINT 'THINKING'
  95 REM-CLEAR MAZE ARRAY
 100 A=H*V+1
 110 FOR I=1 TO A+A;@(I)=0;NEXT I
 120 Q=0,Z=0,X=RND(H)
 125 REM-SAVE MAZE ENTRY POINT
 130 @(A)=X
 140 @(X)=1,C=2
 150 R=X,S=1;GOTO 220
 155 REM-START OF MAZE BUILD ROUTINE
 160 IF R#H GOTO 200
 170 IF S#V GOTO 190
 180 R=1,S=1;GOTO 210
 190 R=1,S=S+1;GOTO 210
 200 R=R+1
 210 IF @(R+(S-1)*H)=0 GOTO 160
 220 IF R-1=0 GOTO 610
 230 IF @(R-1+(S-1)*H)#0 GOTO 610
 240 IF S-1=0 GOTO 420
 250 IF @(R+(S-2)*H)#0 GOTO 420
 260 IF R=H GOTO 320
 270 IF @(R+1+(S-1)*H)#0 GOTO 320
 275 REM-LEFT/DOWN/RIGHT
 280 X=RND(3)
 290 IF X=1 GOTO 960
 300 IF X=2 GOTO 1000
 310 GOTO 1040
 320 IF S#V GOTO 350
 330 IF Z=1 GOTO 400
 340 Q=1;GOTO 360
 350 IF @(R+S*H)#0 GOTO 400
 355 REM-LEFT/DOWN/UP
 360 X=RND(3)
 370 IF X=1 GOTO 960
 380 IF X=2 GOTO 1000
 390 GOTO 1110
 395 REM-LEFT/DOWN
 400 X=RND(2)
 410 GOTO 370
 420 IF R=H GOTO 540
 430 IF @(R+1+(S-1)*H)#0 GOTO 540
 440 IF S#V GOTO 470
 450 IF Z=1 GOTO 520
 460 Q=1;GOTO 430
 470 IF @(R+S*H)#0 GOTO 520
 475 REM-LEFT/RIGHT/UP
 480 X=RND(3)
 490 IF X=1 GOTO 960
 500 IF X=2 GOTO 1040
 510 GOTO 1110
 515 REM-LEFT/RIGHT
 520 X=RND(2)
 530 GOTO 490
 540 IF S#V GOTO 570
 550 IF Z=1 GOTO 960
 560 Q=1;GOTO 580
 570 IF @(R+S*H)#0 GOTO 960
 575 REM-LEFT/UP
 580 X=RND(2)
 590 IF X=1 GOTO 960
 600 GOTO 1110
 610 IF S-1=0 GOTO 820
 620 IF @(R+(S-2)*H)#0 GOTO 820
 630 IF R=H GOTO 750
 640 IF @(R+1+(S-1)*H)#0 GOTO 750
 650 IF S#V GOTO 680
 660 IF Z=1 GOTO 730
 670 Q=1;GOTO 690
 680 IF @(R+S*H)#0 GOTO 730
 685 REM-DOWN/RIGHT/UP
 690 X=RND(3)
 700 IF X=1 GOTO 1000
 710 IF X=2 GOTO 1040
 720 GOTO 1110
```

```
 725 REM-DOWN/RIGHT
 730 X=RND(2)
 740 GOTO 700
 750 IF S#V GOTO 780
 760 IF Z=1 GOTO 1000
 770 Q=1;GOTO 790
 780 IF @(R+S*H)#0 GOTO 1000
 785 REM-DOWN/UP
 790 X=RND(2)
 800 IF X=1 GOTO 1000
 810 GOTO 1110
 820 IF R=H GOTO 910
 830 IF @(R+1+(S-1)*H)#0 GOTO 910
 840 IF S#V GOTO 870
 850 IF Z=1 GOTO 1040
 860 Q=1;GOTO 880
 870 IF @(R+S*H)#0 GOTO 1040
 875 REM-RIGHT/UP
 880 X=RND(2)
 890 IF X=1 GOTO 1040
 900 GOTO 1110
 910 IF S#V GOTO 940
 920 IF Z=1 GOTO 160
 930 Q=1;GOTO 950
 940 IF @(R+S*H)#0 GOTO 160
 950 GOTO 1110
 955 REM-LEFT
 960 @(R-1+(S-1)*H)=C
 970 C=C+1,@(A+R-1+(S-1)*H)=2,R=R-1
 980 IF C=A GOTO 1210
 990 Q=0;GOTO 220
 995 REM-DOWN
1000 @(R+(S-2)*H)=C
1010 C=C+1
1020 @(A+R+(S-2)*H)=1,S=S-1;IF C=A GOTO 1210
1030 Q=0;GOTO 220
1035 REM-RIGHT
1040 @(R+1+(S-1)*H)=C
1050 C=C+1;IF @(A+R+(S-1)*H)=0 GOTO 1070
1060 @(A+R+(S-1)*H)=3;GOTO 1080
1070 @(A+R+(S-1)*H)=2
1080 R=R+1
1090 IF C=A GOTO 1210
1100 GOTO 610
1105 REM-UP
1110 IF Q=1 GOTO 1170
1120 @(R+S*H)=C,C=C+1;IF @(A+R+(S-1)*H)=0 GOTO 1140
1130 @(A+R+(S-1)*H)=3;GOTO 1150
1140 @(A+R+(S-1)*H)=1
1150 S=S+1;IF C=A GOTO 1210
1160 GOTO 220
1165 REM-EXIT AT TOP OF SCREEN
1170 Z=1
1180 IF @(A+R+(S-1)*H)=0 GOTO 1200
1190 @(A+R+(S-1)*H)=3,Q=0;GOTO 160
1200 @(A+R+(S-1)*H)=1,Q=0,R=1,S=1;GOTO 210
1205 REM-MAKE EXIT IF NOT THERE
1210 IF Z#1 X=A+RND(H)+(V-1)*H,@(X)=@(X)+1
1295 REM-END OF MAZE BUILD
1300 PRINT 'DO YOU WANT TO SEE THE MAZE?',
1310 READ 0,I;IF I<128 GOTO 1310
1320 IF I#249 GOTO 1630
1330 GOSUB 2400;PRINT 'CHEAT!!!!'
1335 REM-2D DISPLAY ROUTINE
1340 FOR J=V TO 1 STEP -1
1350 FOR I=1 TO H
1360 IF @(A+I+(J-1)*H)=0 GOTO 1400
1370 IF @(A+I+(J-1)*H)=2 GOTO 1400
1375 REM-PRINT TOP OF CELLS
1380 PRINT '+  ',
1390 GOTO 1410
1400 PRINT '+--',
1410 NEXT I
1420 PRINT '+'
1430 PRINT 'I',
1440 FOR I=1 TO H
1450 IF @(A+I+(J-1)*H)<2 GOTO 1480
1455 REM-PRINT SIDES OF CELLS
1460 PRINT '   ',
1470 GOTO 1490
1480 PRINT '  I',
1490 NEXT I
```

```
1500 PRINT
1510 NEXT J
1520 FOR I=1 TO H
1530 IF I=@(A) GOTO 1550
1535 REM-PRINT BOTTOM OF MAZE
1540 PRINT '+--',;GOTO 1560
1550 PRINT '+  ',
1560 NEXT I
1570 PRINT '+'
1595 REM-PAUSE FOR VIEWING
1600 GOSUB 2450
1610 PRINT 'READY    ',
1620 READ 0,I;IF I<128 GOTO 1620
1625 REM-PRINT INSTRUCTION
1630 GOSUB 1900
1635 REM-TRANSLATE ROUTINE
1640 FOR I=1 TO A-1
1650 J=I+A
1660 @(I)=(3-@(J))*2
1670 NEXT I
1710 W=@(A)
1715 REM-COMPLETE CELL INFORMATION
1720 FOR J=1 TO V
1730 K=(J-1)*H
1740 FOR I=1 TO H
1750 L=I+K
1760 IF J#1 GOTO 1790
1770 IF I=W GOTO 1820
1780 M=1;GOTO 1810
1790 M=@(L-H)/2
1800 M=M-(M/2)*2
1810 @(L)=@(L)+M*8
1820 IF I=1 M=1;GOTO 1850
1830 M=@(L-1)/4
1840 M=M-(M/2)*2
1850 @(L)=@(L)+M
1860 NEXT I
1870 NEXT J
1875 REM-SET UP START PARMS
1880 X=W,Y=0,Z=1
1890 GOTO 3050
1895 REM-INSTRUCTION PRINTOUT
1900 GOSUB 2400
1910 PRINT 'ENTER L TO TURN LEFT'
1920 PRINT '      R TO TURN RIGHT'
1930 PRINT '      F TO GO FORWARD'
1940 PRINT '      H FOR HELP'
1950 RETURN
1995 REM-HELP ROUTINE
2000 PRINT 'YOU ARE AT',^],^J,
2010 PRINT #1,X,' EAST',^],^J,
2020 PRINT #1,Y,' NORTH',^],^J,
2030 PRINT 'YOU ARE FACING',^],^J,
2040 IF Z=1 PRINT 'NORTH',
2050 IF Z=2 PRINT 'EAST ',
2060 IF Z=3 PRINT 'SOUTH',
2070 IF Z=4 PRINT 'WEST ',
2080 PRINT ^],^J,
2090 GOSUB 2450
2100 GOTO 2200
2195 REM-KEYBOARD ROUTINE
2200 IF Y>V GOTO 5000
2210 READ 0,A
2220 IF A<128 GOTO 2210
2230 IF A=236 GOTO 2300
2240 IF A=242 GOTO 2350
2250 IF A=230 GOTO 3000
2260 IF A=232 GOTO 2000
2270 GOTO 2210
2295 REM-LEFT TURN
2300 Z=Z-1
2310 IF Z<1 Z=Z+4
2320 GOTO 3050
2345 REM-RIGHT TURN
2350 Z=Z+1
2360 IF Z>4 Z=Z-4
2370 GOTO 3050
2395 REM-CLEAR SCREEN AND WAIT
2400 I=12
2410 VDU 0,I
2420 FOR I=1 TO 600
2430 NEXT I
```

```
2440 RETURN
2445 REM-RESET CURSOR AND WAIT
2450 I=28
2460 GOTO 2410
2495 REM-ERAZE MESSAGE ROUTINE
2500 GOSUB 2860
2510 PRINT '                  ',
2520 GOSUB 2450
2530 S=0
2540 RETURN
2595 REM-ROTATE AND LOOK ROUTINE
2600 IF B=0 GOTO 2710
2610 IF B>V E=2;RETURN
2620 F=@(A+(B-1)*H)
2630 IF Z=1 GOTO 2670
2635 REM-ROTATE
2640 FOR I=2 TO Z
2650 F=F/2+(F-(F/2)*2)*8
2660 NEXT I
2670 C=F-(F/2)*2
2680 D=F/4-(F/8)*2
2690 E=F/2-(F/4)*2
2700 RETURN
2705 REM-OUTSIDE MAZE
2710 C=0,D=0,E=-1
2720 IF Z#1 GOTO 2760
2730 E=1
2740 IF A=W E=0
2750 RETURN
2755 REM-NO MANS LAND
2760 IF Z=3 E=2
2770 IF Z=2 IF A=H E=2
2780 IF Z=4 IF A=1 E=2
2790 RETURN
2795 REM-INDEX TO NEXT CELL
2800 IF E>0 GOTO 2930
2810 IF Z=1 B=B+1
2820 IF Z=2 A=A+1
2830 IF Z=3 B=B-1
2840 IF Z=4 A=A-1
2850 RETURN
2855 REM-MESSAGE ROUTINE
2860 FOR I=1 TO 8
2870 PRINT ^J,
2880 NEXT I
2890 FOR I=1 TO 23
2900 PRINT ^I,
2910 NEXT I
2920 RETURN
2930 GOSUB 2860
2940 IF E=1 PRINT '  DEAD END  ',
2950 IF E=2 PRINT 'NO MANS LAND',
2960 GOSUB 2450
2970 S=1
2980 RETURN
2995 REM-FORWARD ROUTINE
3000 A=X,B=Y
3010 GOSUB 2600
3020 GOSUB 2800
3030 X=A,Y=B
3040 IF E>0 GOTO 2200
3045 REM-3D DISPLAY ROUTINE
3050 A=X,B=Y
3060 GOSUB 2600
3065 REM-5 DEPTHS
3070 FOR T=1 TO 5
3080 GOSUB 3200
3085 REM-CHECK FOR NEXT DEPTH
3090 IF E#0 GOTO 2200
3100 GOSUB 2800
3110 GOSUB 2600
3120 IF E=2 GOTO 2200
3130 NEXT T
3140 GOTO 2200
3195 REM-JUMP TO DISPLAY DEPTH
3200 GOTO T*400+2810
3205 REM-DISPLAY DEPTH 1
3210 GOSUB 2400
3220 IF E<0 RETURN
3230 IF E>1 RETURN
3240 FOR I=80 TO 976 STEP 64
3250 VDU I,116
```

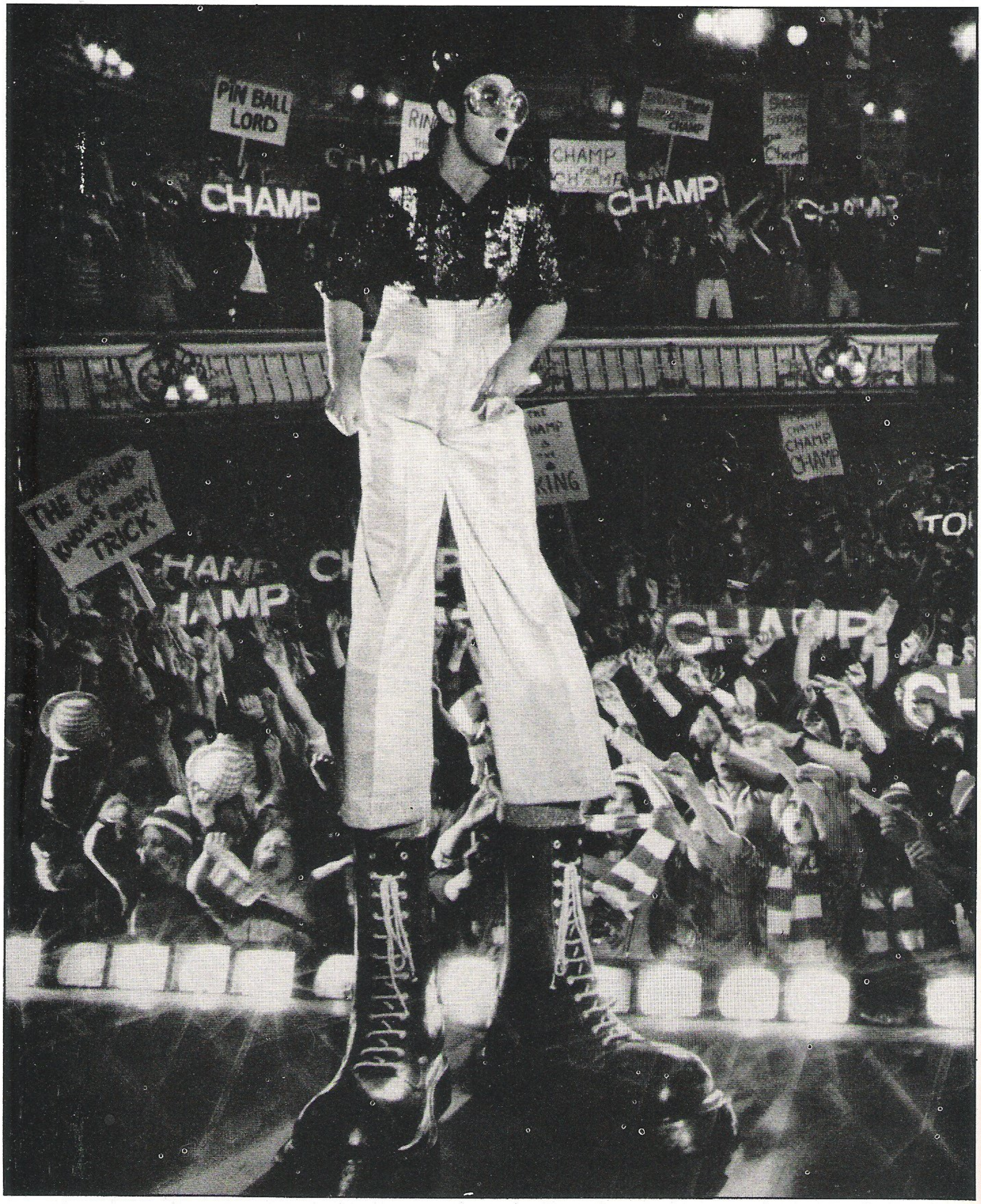
```
3260 VDU I+28,116
3270 NEXT I
3280 IF E=0 RETURN
3290 FOR I=81 TO 107
3300 VDU I,107
3310 VDU I+896,108
3320 NEXT I
3330 RETURN
3600 REM-DISPLAY DEPTH 2
3610 IF C=0 GOTO 3690
3620 VDU 81,114
3630 VDU 147,114
3640 VDU 213,114
3650 VDU 977,113
3660 VDU 915,113
3670 VDU 853,113
3680 GOTO 3730
3690 FOR I=273 TO 277
3700 VDU I,107
3710 VDU I+512,108
3720 NEXT I
3730 IF D=0 GOTO 3810
3740 VDU 107,113
3750 VDU 169,113
3760 VDU 231,113
3770 VDU 1003,114
3780 VDU 937,114
3790 VDU 871,114
3800 GOTO 3850
3810 FOR I=295 TO 299
3820 VDU I,107
3830 VDU I+512,108
3840 NEXT I
3850 FOR I=278 TO 790 STEP 64
3860 VDU I,116
3870 VDU I+16,116
3880 NEXT I
3890 IF E=0 RETURN
3900 FOR I=279 TO 293
3910 VDU I,107
3920 VDU I+512,108
3930 NEXT I
3940 RETURN
4000 REM-DISPLAY DEPTH 3
4010 IF C=0 GOTO 4070
4020 VDU 279,114
4030 VDU 345,114
4040 VDU 791,113
4050 VDU 729,113
4060 GOTO 4110
4070 FOR I=407 TO 409
4080 VDU I,107
4090 VDU I+256,108.
4100 NEXT I
4110 IF D=0 GOTO 4170
4120 VDU 293,113
4130 VDU 355,113
4140 VDU 805,114
4150 VDU 739,114
4160 GOTO 4210
4170 FOR I=419 TO 421
4180 VDU I,107
4190 VDU I+256,108
4200 NEXT I
4210 FOR I=410 TO 666 STEP 64
4220 VDU I,116
4230 VDU I+8,116
4240 NEXT I
4250 IF E=0 RETURN
4260 FOR I=411 TO 417
4270 VDU I,107
4280 VDU I+256,108
4290 NEXT I
4300 RETURN
4400 REM-DISPLAY DEPTH 4
4410 IF C=0 GOTO 4450
4420 VDU 411,114
4430 VDU 667,113
4440 GOTO 4470
4450 VDU 475,107
4460 VDU 603,108
4470 IF D=0 GOTO 4510
4480 VDU 417,113
4490 VDU 673,114
4500 GOTO 4530
4510 VDU 481,107
4520 VDU 609,108
4530 FOR I=476 TO 604 STEP 64
4540 VDU I,116
4550 VDU I+4,116
4560 NEXT I
4570 IF E=0 RETURN
4580 FOR I=477 TO 479
4590 VDU I,107
4600 VDU I+123,108
4610 NEXT I
4620 RETURN
4800 REM-DISPLAY DEPTH 5
4810 IF C=0 GOTO 4850
4820 VDU 477,114
4830 VDU 605,113
4840 GOTO 4870
4850 VDU 477,108
4860 VDU 605,107
4870 IF D=0 GOTO 4910
4880 VDU 479,113
4890 VDU 607,114
4900 GOTO 4930
4910 VDU 479,108
4920 VDU 607,107
4930 VDU 541,106
4940 VDU 543,105
4950 IF E=0 RETURN
4960 VDU 478,108
4970 VDU 606,107
4980 RETURN
4995 REM-WAY OUT FOUND
5000 GOSUB 2400
5010 GOSUB 2860
5020 PRINT ´  WAY OUT
5030 STOP
```



S Hueber

PINBALL

Sure plays a mean pinball . . .

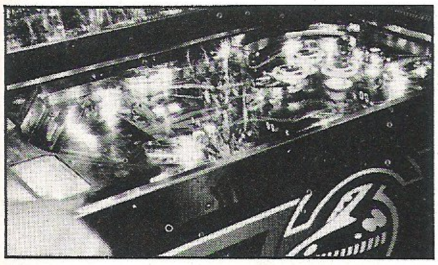


Photograph courtesy of Hemdale International Films Ltd.

Ever since I was a young boy,
I played the silver ball,
From Soho down to Brighton,
I must have played them all.

Pete Townshend
excerpt from the rock
opera, Tommy.

Remember a time when amusement arcades were filled with row upon row of pinball machines instead of the brightly flashing screens of the Space Invader kind. Well now you can bring the thrills and spills of the pinball game into your own living room with this program.



PROGRAM STRUCTURE

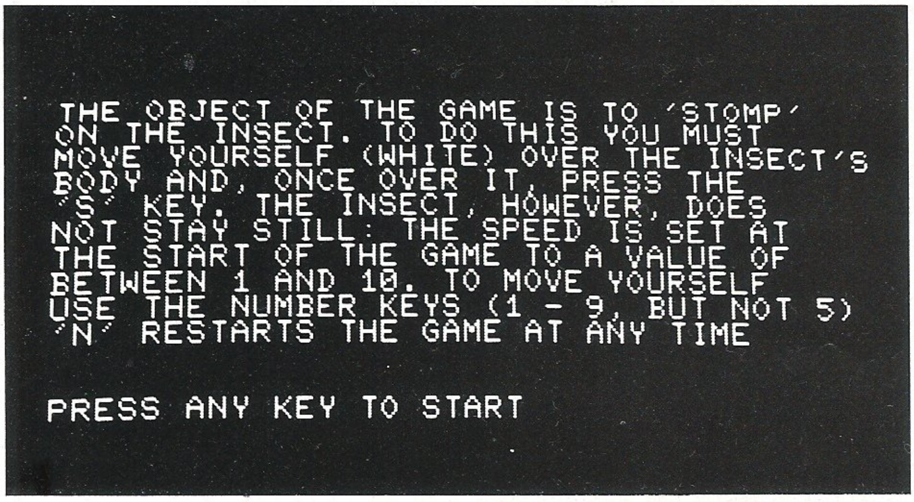
Statement	Action
Line 9	Player starts off with one game.
Lines 10-100	Instructions.
Lines 110-196	Set up pin-table.
Lines 197-205	Give ball initial position and direction. Reset drop targets.
Lines 210-215	Put ball into play.
Lines 220-300	Process selected depending upon contents of next location in ball's path.
Lines 320-330	Bat control.
Lines 340-360	Calculate next location and test for ball out of play.
Line 362	Same ball again if no points scored.
Lines 363-365	Test for final ball.
Lines 367-370	End of game messages.
Lines 449-530	Subroutines.
Lines 449-459	Limits of bat movement.
Line 500	Prevents ball standing still!
Lines 511-512	Count drop targets hit. If all hit extra ball awarded.
Lines 513-519	Print score, test for replays and print replays.

```
  9  CR=1
 1Ø  PRINT "DO YOU WANT INSTRUCTIONS (Y OR N)"
 2Ø  GOSUB 52Ø
 3Ø  IF A$="N" THEN 11Ø
 35  PRINT:PRINT
 4Ø  PRINT "3 BALLS PER GAME. PRESSING '1' MOVES"
 5Ø  PRINT "BAT 1 SPACE TO LEFT, '2' MOVES IT TO"
 6Ø  PRINT "RIGHT. BAT DETERMINES NEW DIRECTION OF"
 7Ø  PRINT "BALL ACCORDING TO WHERE ON BAT BALL"
 8Ø  PRINT "LANDS."
 81  PRINT "COMPLETING DROP TARGET SCORES"
 82  PRINT "EXTRA BALL. MAXIMUM 1 EXTRA BALL"
 83  PRINT "PER BALL IN PLAY."
 84  PRINT "1 REPLAY AWARDED WHEN 5Ø POINTS"
 85  PRINT "SCORED. 1 REPLAY FOR EACH"
 86  PRINT "ADDITIONAL SCORE OF 2Ø POINTS."
 87  PRINT "TO GET EACH BALL INTO PLAY PRESS"
 88  PRINT "ANY KEY."
1ØØ  PRINT:PRINT:PRINT "PRESS ANY KEY TO CONTINUE"
1Ø5  GOSUB 52Ø
11Ø  CR=CR-1
12Ø  PRINT "[2 SPC]";:FOR N=3281Ø TO 3282Ø:POKE N,1ØØ:
     NEXT N:POKE 32849,78:POKE 32861,77
13Ø  FOR N=32888 TO 334Ø8 STEP 4Ø:POKE N,1Ø3:NEXT N
14Ø  FOR N=329Ø2 TO 33422 STEP 4Ø:POKE N,1Ø1:NEXT N
15Ø  B=33415:POKE B-1,233:POKE B,16Ø:POKE B+1,223
16Ø  N=328Ø9:POKE N,78:POKE N+39,78:POKE N+12,77:
     POKE N+53,77
17Ø  X=33135
175  POKE X-123,15:POKE X-117,15
18Ø  POKE X-2,15:POKE X+2,15
185  POKE X+78,15:POKE X+82,15
19Ø  PRINT TAB(2Ø);"BALL IN PLAY Ø"
193  PRINT "[CD]";TAB(2Ø);"CREDIT"
194  GOSUB 518
195  S=Ø
196  N=1
197  IY=-1:IX=2:GOSUB 49Ø
198  P=32855:X1=7+IX:Y1=21
199  T=32895+IX
2ØØ  POKE 328Ø1,N+48
2Ø1  X=32852:FOR Y=X TO X+2
2Ø2  POKE Y,9Ø:NEXT Y:FOR Y=X+4 TO X+6
2Ø3  POKE Y,9Ø:NEXT Y
2Ø4  E=Ø
2Ø5  S1=S
21Ø  GOSUB 52Ø
215  POKE P,81
22Ø  Q=PEEK(T)
23Ø  IF Q=32 OR G=96 THEN POKE P,32:P=T:POKE P,81:
     X=X1:Y=Y1
235  IF Q=9Ø THEN GOSUB 511
24Ø  IF Q=1Ø3 OR Q=1Ø1 THEN IX=-IX:IY=IY^IY
25Ø  IF Q=1ØØ THEN IY=-IY
26Ø  IF Q=15 THEN GOSUB 513:GOSUB 47Ø
27Ø  IF Q=77 OR Q=78 THEN IX=-IX:IY=-IY
28Ø  IF Q=233 THEN IX=-1:IY=1
29Ø  IF Q=16Ø THEN IX=Ø:IY=1
3ØØ  IF Q=223 THEN IX=1:IY=1
31Ø  FOR D=1 TO 5Ø:NEXT D
32Ø  GET D
33Ø  ON D GOSUB 449,459
34Ø  X1=X+IX:Y1=Y+IY:T=33728+X1-4Ø*Y1
35Ø  IF T<33768 THEN 22Ø
36Ø  POKE P,32
362  IF S1=S THEN N=N-1
363  N=N+1
365  IF N<4 THEN 197
367  PRINT "[17 CD]"
368  IF CR=Ø THEN 533
37Ø  PRINT "PRESS 'R' FOR NEXT GAME"
38Ø  GOSUB 52Ø
39Ø  IF A$="R" THEN 11Ø
44Ø  STOP
449  IF B=33411 THEN RETURN
45Ø  POKE B+1,32:POKE B,223:POKE B-1,16Ø:POKE B-2,223:
     B=B-1
452  RETURN
459  IF B=33419 THEN RETURN
46Ø  POKE B-1,32:POKE B,233:POKE B+1,16Ø:POKE B+2,223:
     B=B+1
463  RETURN
47Ø  D=INT(RND(1)*3-1):IF D=IY THEN 47Ø
48Ø  IY=D
49Ø  D=INT(RND(1)*3-1):IF D=IX THEN 49Ø
5ØØ  IX=D:IF IX=Ø AND IY=Ø THEN 49Ø
51Ø  RETURN
511  POKE T,32:E=E+1
512  IF E=6 THEN N=N-1
513  S=S+1:PRINT "S";S:IF S<5Ø THEN RETURN
514  IF S=5Ø THEN 517
515  IF INT((S-5Ø)/2Ø)=(S-5Ø)/2Ø THEN 517
516  RETURN
517  CR=CR+1
518  IF CR<1Ø THEN POKE 32876,CR+48:RETURN
519  D=INT(CR/1Ø):POKE 32875,D+48:POKE 32876,CR-D*1Ø+48:
     RETURN
52Ø  GET A$:IF A$="" THEN 52Ø
53Ø  RETURN
533  PRINT "FOR ANOTHER GAME INSERT 1ØP COIN"
534  PRINT "(OR RUN THE PROGRAM AGAIN)"
54Ø  END
```

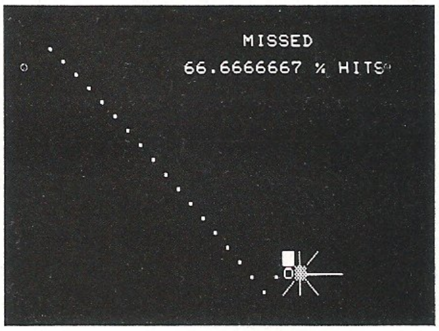
Pete Howells

STOMPER

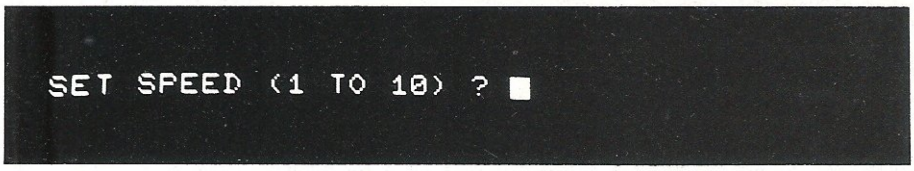
A game which should appeal to the meaner side of your nature!



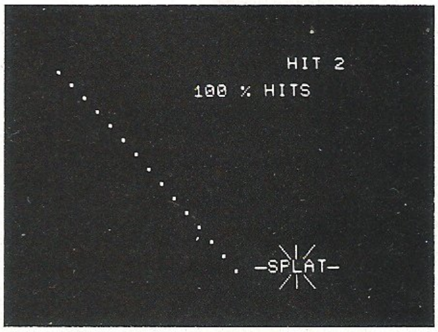
```
THE OBJECT OF THE GAME IS TO 'STOMP'
ON THE INSECT. TO DO THIS YOU MUST
MOVE YOURSELF (WHITE) OVER THE INSECT'S
BODY AND, ONCE OVER IT, PRESS THE
'S' KEY. THE INSECT, HOWEVER, DOES
NOT STAY STILL: THE SPEED IS SET AT
THE START OF THE GAME TO A VALUE OF
BETWEEN 1 AND 10. TO MOVE YOURSELF
USE THE NUMBER KEYS (1 - 9, BUT NOT 5)
'N' RESTARTS THE GAME AT ANY TIME

PRESS ANY KEY TO START
```



```
                    MISSED
            66.6666667 % HITS
```



```
SET SPEED (1 TO 10) ? ■
```



```
                    HIT 2
              100 % HITS

                    -SPLAT-
```

PROGRAM STRUCTURE

Statement	Function	Action
Lines 100-210	Set Up	Move to the instruction routine, set up graphics characters for the target and set the delay for the speed.
Line 220	Clear Screen	Clears the screen.
Lines 300-320	Move Cursor	Position the cursor by means of various key presses.
Lines 330-340	Screen Limits	Stop the cursor going off the top and bottom of the screen.
Line 360	Speed Delay	Controls the speed of the target.
Lines 380-470	Blank Target	The target is blanked.
Lines 480-530	Movement	Control the movement of the target.
Lines 540-620	Restore Image	Restore the image on the screen.
Line 630	Hit Test	If the position of the cursor and the target coincide, a test for a 'hit' is made.
Lines 650-680	Miss	Print a 'missed' message if you did not stomp on the insect.
Line 740	Hit	Prints a 'hit' message if insect has been stomped on.
Lines 800-950	Instructions	Print rules of the game.

Stomper was one of the first moving graphics games and shows some of the weaknesses of the early systems that it was written for. Originally written to fit on the 8K Old ROM PETs, it is presented here as a source of inspiration rather than as an example of excellent programming technique!

HOW TO PLAY

The basic object of the game is given in the instructions and is quite simple — you must position the cursor and 'stomp' on the randomly moving insect.

There are numerous ways in which the game could be improved: sound, more than one insect and slowing the insect down if you 'stomp' a leg off are just a few of the options. None of the statements included in the program should cause any trouble to the avid converter, the only requirement is for a memory mapped screen and PEEK and POKE statements. The choice of graphics characters to make up the insect are fairly arbitrary (they work on the PET).

```
100  PRINT "[CLS]DO YOU WANT INSTRUCTIONS (Y OR N)"
110  GET A$
120  IF A$="" THEN 110
130  IF A$="Y" THEN 800
140  DIM B$(8),C(8)
150  YY=160:MM=102:SS=46:SY=32
160  DATA "7",-41,"8",-40,"9",-39,"4",-1,"6",1,"1",39,
     "2",40,"3",41
170  FOR K=1 TO 8:READ B$(K),C(K):NEXT K
180  INPUT "[CLS]SET SREED (1 TO 10)";DF
190  IF DF>10 OR DF<1 THEN 180
200  DF=DF/50
210  J=32768
220  PRINT "[CLS]"
230  I=33267
240  POKE J,YY
250  GET A$
260  IF A$="" THEN 350
270  IF A$="S" THEN 630
280  IF A$="N" THEN 940
290  POKE J,SS
300  FOR K=1 TO 8
310  IF A$=B$(K) THEN J=J+C(K)
320  NEXT K
330  IF J>33767 THEN J=J-40
340  IF J<32768 THEN J=J+40
350  POKE J,YY
360  IF RND(TI)>DF THEN 250
370  X=RND(TI)
380  POKE I-41,32
390  POKE I-40,32
400  POKE I-39,32
410  POKE I-1,32
420  POKE I,SY
430  POKE I+1,32
440  POKE I+2,32:POKE I+3,32
450  POKE I+39,32
460  POKE I+40,32
470  POKE I+41,32
480  IF X<0.25 THEN I=I-40
490  IF X>0.25 AND X<0.5 THEN I=I-1
500  IF X>0.5 AND X<0.75 THEN I=I+1
510  IF X>0.75 THEN I=I+40
520  IF I>33767 THEN I=I-40
530  IF I<32768 THEN I=I+40
540  POKE I,MM
550  POKE I-41,77
560  POKE I-40,66
570  POKE I-39,78
580  POKE I-1,87
590  POKE I+1,64:POKE I+2,64:POKE I+3,64
600  POKE I+39,78:POKE I+40,66
610  POKE I+41,77
620  GOTO 250
630  IF I=J THEN 710
640  POKE J,YY:POKE I,MM
650  PRINT "[HOM][16 CR]MISSED"
660  MX=MX+1
670  PRINT "[HOM][2 CD][10 CR]";100*(N/(N+MX+1E-30));
     "% HITS"
680  FOR KK=1 TO 1000:NEXT KK
690  PRINT "[CLS]"
700  GOTO 370
710  N=N+1
720  POKE I-2,19:POKE I-1,16:POKE I,12:POKE I+1,1:
     POKE I+2,20
730  POKE I-3,64
740  PRINT "[HOM][19 CR]HIT";N
750  PRINT "[HOM][2 CD][10 CR]";100*(N/(N+MX+1E-30));
     "% HITS"
760  FOR KK=1 TO 1000:NEXT KK
770  PRINT "[CLS]"
780  J=32768
790  GOTO 250
800  PRINT "[CLS]THE OBJECT OF THE GAME IS TO 'STOMP'"
810  PRINT "ON THE INSECT. TO DO THIS YOU MUST"
820  PRINT "MOVE YOURSELF (WHITE BLOCK) OVER THE
     INSECT'S"
830  PRINT "BODY AND, ONCE OVER IT, PRESS THE"
840  PRINT "'S' KEY. THE INSECT, HOWEVER, DOES"
850  PRINT "NOT STAY STILL: THE SPEED IS SET AT"
860  PRINT "THE START OF THE GAME."
870  PRINT "TO MOVE YOURSELF USE THE NUMBER KEYS"
880  PRINT "(1 - 9 BUT NOT 5)."
890  PRINT "'N' RESTARTS THE GAME AT ANY TIME."
900  PRINT "[2 CD]PRESS ANY KEY TO START"
910  GET A$
920  IF A$="" THEN 910
930  GOTO 140
940  N=0:MX=0
950  GOTO 180
```

Trevor Lusty

KIRK vs THE

All photographs courtesy of UIP (Paramount).

While playing an old version of Startrek recently, I wondered what the real thing would be like. I could not help feeling that the real James T Kirk would have been zapped by a Klingon long before the short-range scan was halfway up the screen.

The Failings of VDUs

The reason for most games behaving like a piece of electronic paper scrolling slowly with a vertical motion is largely historical. Most of the software used at present is adapted from the time when your interface with a computer was either a set of punched cards or a printer connected to a mainframe at the other end of a telephone line. Unfortunately, for many people, those 'good old days' are still with us. There were compensations, 32K BASIC interpreters and 64K Startreks being just two of the advantages! What I wanted was to be able to print output to selected areas of the screen without disturbing material already present. There appeared to be two basic methods.

1) I could POKE everything to my memory mapped screen, but this was going to be time consuming to program when text had to be printed.

2) I could use the cursor movements provided by the manufacturer to signal the start positions of a print statement, but when a given output might appear anywhere on the screen this method might prove difficult too.

Are there other methods? After a little thought I decided that it must be possible to refine each of the above methods as follows:-

1a) I could write a machine code subroutine to be called by the main program. This meant that I could get my assembler program to do the decoding of text for me.

2a) I could delve into page zero of memory, find where the machine stored the cursor position and see if I could use this to simplify the movement required.

The Final Solution

In the end I decided to use all the above methods except for machine code. Each method was used where it seemed most appropriate and the final division was:

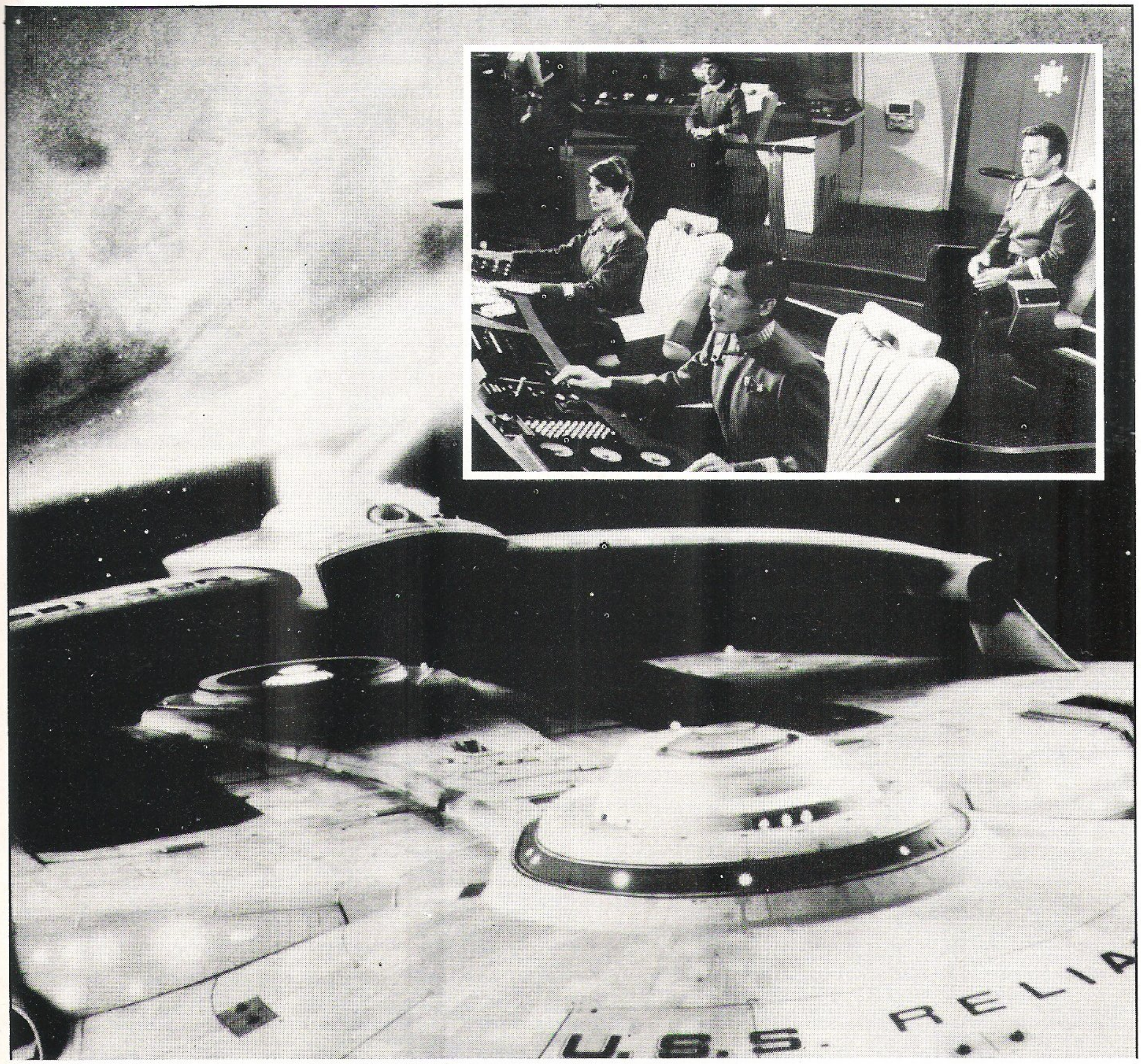
Method 1: used to simulate the final detonation. This always occurred in the same place and therefore it was easy to store the exact locations and characters in a data statement.

Method 2: used for instructions and messages. There is little point in being clever just for the sake of it.

Method 2a: used to plot the alien spacecraft. Here we

CURSOR

Captain Kirk never had it so good! Liven up any graphics game with our simple techniques.



have a string of characters representing the craft which wander about the screen. Because they are always the same they fit nicely into a print statement, but normal cursor control would have been messy.

What I wanted to be able to do was specify the starting position of the spacecraft string using the two co-ordinates XC and YC which save the displacements across and down from the top left-hand corner of the screen. I found that locations 196 and 197 (Old ROMs 224, 225) of my PET hold the exact position of the cursor in Hex. The decimal value is always between 32K and 33K and may be found using the function 256* PEEK (197) + PEEK(196). In the program we require the inverse function and this is coded in lines 2860 to 2920 of the listing.

I won't claim that this is the greatest program I have ever written, but I did learn a great deal about cursor control while writing it. It is an enjoyable game by itself, but I feel that it will really come into its own as a subroutine to my Startrek program.

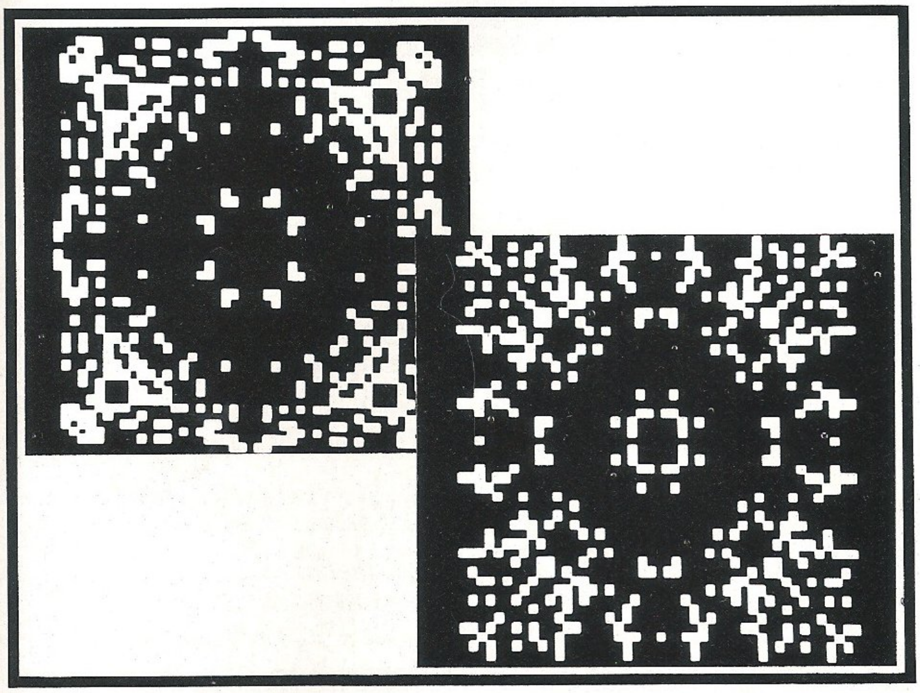
After all, James T Kirk ought to have something extra now that they have made him a Commodore!

▶

```
1180  PRINT "[CLS][8 CD]"
1200  PRINT "[11 SPC][REV][13 SPC][OFF]"
1220  PRINT "[11 SPC][REV] SPACE ATTACK [OFF]"
1240  PRINT "[11 SPC][REV][13 SPC][OFF]"
1260  FOR I=1 TO 1ØØØ:NEXT I
1280  POKE 59468,14
1300  PRINT "[CLS]SIX ALIEN INVADERS HAVE PENETRATED"
1320  PRINT "EARTH'S OUTER DEFENCES."
1340  PRINT:PRINT
1360  PRINT "AS COMMANDER OF THE LAST FIGHTER DEFENCE";
1380  PRINT "SHIP, YOUR MISSION IS TO DESTROY THEM."
1400  PRINT:PRINT
1420  PRINT "YOU MUST POSITION YOUR PHASER SIGHT"
1440  PRINT "SO THAT THE ALIEN CRAFT IS IN THE CENTRE";
1460  PRINT "AND THEN FIRE YOUR WEAPONS."
1480  PRINT:PRINT
1500  PRINT "PRESS  5  TO FIRE"
1520  PRINT "PRESS  8  TO MOVE YOUR SIGHT UP"
1540  PRINT "PRESS  2  TO MOVE YOUR SIGHT DOWN"
1560  PRINT "PRESS  4  TO MOVE YOUR SIGHT LEFT"
1580  PRINT "PRESS  6  TO MOVR YOUR SIGHT RIGHT"
1600  PRINT:PRINT
1620  PRINT "DAMAGE FROM THE ENEMY ATTACK WILL GRADUALLY"
1640  PRINT "DESTROY YOUR AIM --- SO DON'T DELAY"
1660  PRINT
1680  PRINT "PRESS SPACE BAR WHEN READY"
1700  GET Q$:IF Q$<>"[SPC]" THEN 17ØØ
1720  POKE 59468,12
1740  PRINT "[CLS]RATINGS : B  BEGINNER"
1760  PRINT "[8 SPC]: N  NOVICE"
1780  PRINT "[8 SPC]: V  VETERAN"
1800  INPUT "[5 CD][9 SPC]RATING ";R$
1820  IF R$="B" THEN RA=35Ø:GOTO 188Ø
1840  IF R$="N" THEN RA=2ØØ:GOTO 188Ø
1860  IF R$="V" THEN RA=5Ø
1880  PRINT "[CLS]"
1900  YC=1Ø:XC=8:GOSUB 286Ø
1920  PRINT "TALLY HO AND GOOD LUCK"
1940  FOR N=1 TO 5ØØ:NEXT N
1960  MI=1:HI=1:PRINT "[CLS]"
1980  GOSUB 41ØØ:REM ** FIND AND PRINT RATING
2000  GOSUB 238Ø:REM ** PRINT PHASER SIGHT
2020  YC=INT(8*RND(1)+1Ø)
2040  XC=INT(2Ø*RND(1)+5)
2060  GOSUB 286Ø:REM ** POSITION CURSOR
2080  GOSUB 274Ø:REM ** PRINT ALIEN CRAFT
2100  PRINT "[HOM]"
2120  GOSUB 378Ø:REM ** INCREMENT TIME AND TEST
2140  GET D$:IF D$="" THEN 212Ø
2160  GOSUB 286Ø:REM ** POSITION CURSOR
2180  IF D$="4" THEN XC=XC+1
2200  IF D$="6" THEN XC=XC-1
2220  IF D$="8" THEN YC=YC+1
2240  IF D$="2" THEN YC=YC-1
2260  IF D$="5" THEN 3ØØØ
2280  GOSUB 36ØØ:REM ** REMOVE OLD POSITION
2300  GOSUB 286Ø:REM ** SET NEW POSITION
2320  GOSUB 274Ø:REM ** PRINT ALIEN CRAFT
2340  GOTO 21ØØ
2360  FOR I=1 TO 2ØØØ:NEXT I:END
2380  PRINT "[HOM][4 CD]":PRINT "[1Ø SPC][2Ø^"][OFF]"
2400  PRINT "[19 SPC][2^%]"
2420  PRINT "[19 SPC][2^%]"
2440  PRINT "[19 SPC][2^%]"
2460  PRINT "[REV][^'][OFF][38 SPC][^']"
2480  PRINT "[REV][^'][OFF][38 SPC][^']"
2500  PRINT "[REV][^'][OFF][5^#][28 SPC][5^#][^']"
2520  PRINT "[REV][^'][OFF][5^#][13 SPC][^[][14 SPC][^']"
2540  PRINT "[REV][^'][OFF][38 SPC][^']"
2560  PRINT "[REV][^'][OFF][38 SPC][^']"
2580  PRINT "[19 SPC][2^%]"
2600  PRINT "[19 SPC][2^%]"
2620  PRINT "[19 SPC][2^%]"
2640  PRINT "[1Ø SPC][REV][2Ø^"][OFF]"
2660  RETURN
2680  REM **
2700  REM ** PRINT ALIEN CRAFT
2720  REM **
2740  PRINT "[^Z]-[^Q]-[^Z]"
2760  P=YC:Q=XC
2780  RETURN
2800  REM **
2820  REM ** SET CURSOR
2840  REM **
2860  S1=32768+4Ø*YC+XC
2880  S3=INT(S1/256)
2900  S2=S1-256*S3
2920  POKE 196,S2:POKE 197,S3:RETURN
2940  REM **
2960  REM ** TEST FOR HIT
2980  REM **
3000  IF YC=12 THEN 3Ø6Ø
3020  GOSUB 322Ø
3040  GOTO 2Ø6Ø
3060  IF XC=17 THEN 312Ø
3080  GOSUB 322Ø
3100  GOTO 2Ø6Ø
3120  YC=12:XC=16:GOSUB 286Ø:GOSUB 432Ø:GOSUB 238Ø
3140  YC=21:XC=1Ø:GOSUB 286Ø
3160  PRINT "ENEMY DESTROYED =";HI
3180  HI=HI+1:IF HI=7 THEN 336Ø
3200  GOTO 2Ø2Ø
3220  P=YC:Q=XC:YC=22:XC=1Ø:GOSUB 286Ø
3240  PRINT "NUMBER MISSED =";MI:MI=MI+1
3260  YC=P:XC=Q:GOSUB 286Ø
3280  RETURN
3300  REM **
3320  REM ** PRINT RESULTS
3340  REM **
3360  YC=22:XC=1Ø:GOSUB 286Ø
3380  MI=MI-1
3400  PRINT "[HOM]":YC=1:XC=Ø:GOSUB 286Ø
3420  PRINT "[8 SPC]PLANET EARTH HAS BEEN SAVED"
3440  PRINT "[CLS][8 SPC]PERFORMANCE =";INT(6/(6+MI)*1ØØ);
      "%"
3460  IF 6/(6+MI)*1ØØ>75 THEN RA=RA-75:IF RA<15 THEN
      RA=25
3480  IF 6/(6+MI)*1ØØ<51 THEN RA=RA+5Ø
3500  GOSUB 41ØØ:YC=24:XC=Ø:GOSUB 286Ø
3520  PRINT "[6 SPC][REV]DO YOU WANT ANOTHER MISSION ?"
3540  GET Q$:IF Q$="" THEN 354Ø
3560  IF Q$<>"Y" THEN END
3580  GOTO 196Ø
3600  S1=32768+4Ø*P+Q
3620  S3=INT(S1/256)
3640  S2=S1-256*S3
3660  POKE 196,S2:POKE 197,S3
3680  PRINT "[5 SPC]"
3700  RETURN
3720  REM **
3740  REM ** INCREMENT TIME
3760  REM **
3780  T=T+1
3800  IF T>RA THEN 39ØØ
3820  RETURN
3840  REM **
3860  REM ** MOVE AND TEST ALIEN
3880  REM **
3900  IF XC>36 THEN 398Ø
3920  IF YC<12 THEN 398Ø
3940  XC=XC-1:YC=YC+1:GOSUB 36ØØ:GOSUB 286Ø:GOSUB 274Ø:
      PRINT "[HOM]"
3860  GOTO 4ØØØ
3980  XC=XC+1:YC=YC-1:GOSUB 36ØØ:GOSUB 286Ø:GOSUB 274Ø:
      PRINT "[HOM]"
4000  IF YC>18 OR YC<5 THEN PRINT "[CLS][7 CD][1Ø SPC]
      THEY GOTCHA":GOTO 236Ø
4020  T=Ø:RETURN
4040  REM **
4060  REM ** SET RATINGS
4080  REM **
4100  IF RA<=5Ø THEN Q$="RATING = VETERAN PILOT":GOTO
      416Ø
4120  IF RA>=35Ø THEN Q$="RATING = BEGINNER":GOTO 416Ø
4140  Q$="RATING = NOVICE PILOT"
4160  YC=2:XC=8:GOSUB 286Ø
4180  PRINT "[26 SPC]"
4200  YC=2:XC=8:GOSUB 286Ø
4220  PRINT Q$
4240  RETURN
4260  REM **
4280  REM ** SIMULATE HIT
4300  REM **
4320  RESTORE
4340  FOR J=1 TO 15
4360  READ L,M
4380  POKE L,M
4400  NEXT J
4420  RETURN
4440  DATA 33267,42,33266,42,33268,42
4460  DATA 33227,42,333Ø7,42,33267,32
4480  DATA 33265,42,33269,42,33266,32
4500  DATA 33268,32,33227,32,333Ø7,32
4520  DATA 33269,32,33265,32,33267,91
```

NASCOM PATTERNS

A graphic illustration of the NASCOM Graphics ROM's functions.



This program generates a random, but highly symmetrical, pattern which gradually builds up, stays a while and is then replaced by a new sequence. Typical examples of the patterns produced are shown in the illustration. They have reflectional symmetry about the diagonals and about the vertical and horizontal axes passing through the centre. The program produces a (nearly) square array of 48 by 48 points using the SET(x,y) function and so can only be used when the Graphics ROM is available. The patterns produced are quite pleasing in black and white, but would be fabulous if adapted for use with a colour board.

Logical Progression

The logic flow is as follows. A random number pair (x,y) is generated in the range (1,1) to (24,24), corresponding to the upper left-hand quadrant of the pattern. The program makes sure that x>y, so that the point lies in the upper half of the quadrant. The subroutine at line 2000 centres the pattern and reflects the point about the horizontal and vertical axes passing through the centre of the screen. The original values of x and y are interchanged (line 250) to give reflection about the diagonals, and the subroutine is called again. The values of x and y are then incremented by ± 1 or 0; the program checks that the point is not already set and that it still lies within the starting segment. Each point thus grows as a randomly shaped blob until these conditions fail and then a new random point is started. A more disconnected pattern can be produced by removing line 320 and setting K in line 350 to 75. The two photographs were actually taken with line 320 removed.

The patterns are generated with x and y values lying between 1 and 48; the SET function has x values from 0 to 95 and y from 0 to 47. The x values are all incremented by 22 to bring the pattern into the centre of the screen before SETting. The unscrolled line 16 in the NASCOM is printed as the top line above lines 1 to 15, and has to be unscrambled to produce a symmetrical pattern. This is taken care of in the subroutine, which decreases each y value by four (you would expect it to be three since SET divides each character into three vertically as well as two horizontally, but x values start at one while SET runs from 0) but if y< 4 it is increased by 48 to produce the top line.

```
  50  K=Ø:CLS:DX=Ø:DY=Ø
 10Ø  X=INT(RND(Ø.5)*24+1):Y=INT(RND(Ø.1)*24+1)
 12Ø  IF X<Y THEN Y=25-Y
 14Ø  DX=INT(RND(Ø.3)*3-1):DY=INT(RND(Ø.2)*3-1)
 15Ø  X=X+DX:Y=Y+DY
 16Ø  IF X<25 AND X>Ø AND Y<25 AND Y>Ø AND Y<=X THEN 18Ø
 17Ø  GOTO 1ØØ
 18Ø  IF POINT(X,Y)=Ø THEN 2ØØ
 139  REM ** YOU NOW HAVE A STARTING POINT IN CORRECT
      SEGMENT
 19Ø  GOTO 1ØØ
 2ØØ  GOSUB 2ØØØ
 25Ø  Z=X:X=Y:Y=Z
 259  REM ** INTERCHANGE X AND Y, REFLECTS ABOUT THE
      DIAGONALS
 3ØØ  GOSUB 2ØØ
 32Ø  Z=X:X=Y:Y=Z
 329  REM ** CHANGE X AND Y BACK AGAIN
 35Ø  K=K+1:IF K<175 THEN 12Ø
 359  REM ** DETERMINES NUMBER OF POINTS SET
 4ØØ  FOR T=1 TO 5ØØØ:NEXT T:GOTO 5Ø
 4Ø9  REM ** T DETERMINES DELAY BETWEEN PATTERNS
1999  REM ** SUBROUTINE REFLECTS ABOUT CENTRAL AXES,
      CENTRES PATTERN AND PUTS LINE 16 AT THE BOTTOM
2ØØØ  A=X+22:IF Y<4 THEN B=Y+44:GOTO 22ØØ
21ØØ  B=Y-4
22ØØ  SET(A,B):SET(7Ø-X,B)
23ØØ  P=X+22:Q=44-Y
24ØØ  SET(P,Q):SET(7Ø-X,Q)
25ØØ  RETURN
```

THE NEW MAGAZINE THAT WILL PROBABLY BE OF LITTLE INTEREST TO **YOU!**
BUT . . . .

FIRST ISSUE AT ALL GOOD NEWSAGENTS NOW!

GREAT VALUE WITH 100 PAGES FOR ONLY 65p!

ON SALE THE FIRST FRIDAY OF EVERY MONTH

. . . . .FOR ALL OF THOSE CONSIDERING BUYING THEIR FIRST MICROCOMPUTER AND FOR THOSE SEEKING TO EXPAND OR UPGRADE THEIR EXISTING MACHINE OR WHO ARE DECIDING UPON THE NEXT MICROCOMPUTER THEY WILL BUY

Personal Computing Today WILL BECOME ESSENTIAL READING!

TELL YOUR FRIENDS ABOUT IT! OR EVEN HAVE A LOOK YOURSELF!

UTILITIES

'Scope Simulation — A full simulation of that most versatile of research tools, the digital storage 'scope, for the classroom.
Originally published in Computing Today, December 1981.

Screen Print — Adding a direct screen copy facility to the Sorcerer/MX80 combination.
Originally published in Computing Today, December 1981.

Line Plotter — Using the Microtan's chunky graphics to your advantage.
Originally published in Computing Today, October 1981.

PET Lister — Convert those awkward graphics characters to the standard codes from those awfully nice CT people.
Originally published in Computing Today, September 1981.

Graph Plotter — An excellent routine to give you neat and tidy graphical output.
Originally published in Computing Today, November 1980.

Cross Hatcher — Check out your TV with the DAI's colour graphics.
Originally published in Computing Today, December 1981.

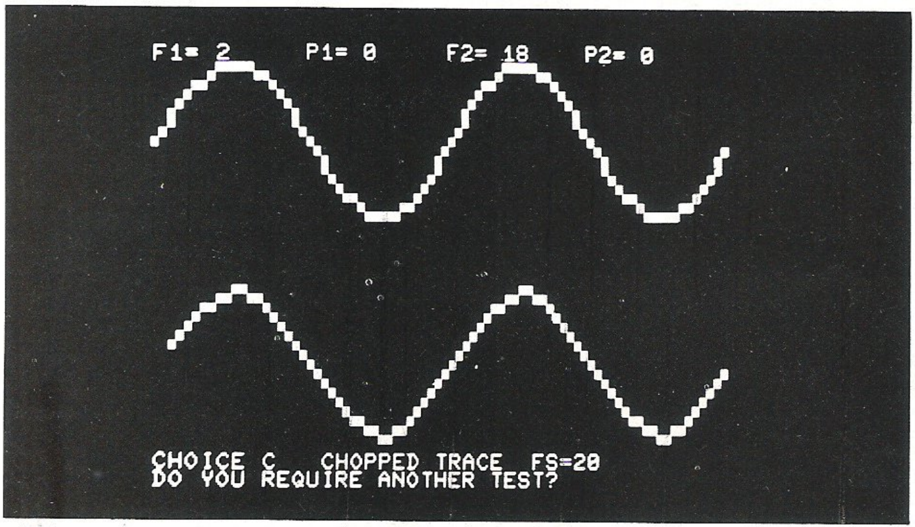
Double Density — Double your plotting capacity with this routine.
Originally published in Computing Today, December 1980.

Fast Plotter — Speed up the plotting of all those complex functions with this superbly documented program.
Originally published in Computing Today, April 1981.

Len Topple

'SCOPE SIMULATION

A full simulation of that most useful of research tools, the digital storage 'scope. Ideal for the classroom.



```
F1= 2     P1= 0     F2= 18    P2= 0

CHOICE C   CHOPPED TRACE   FS=20
DO YOU REQUIRE ANOTHER TEST?
```

A normal oscilloscope is a very versatile piece of test equipment and is one of the basic tools to be found in most development laboratories and service workshops. While it has the fast response times necessary for studying the operations of modern digital ICs, it keeps no permanent record of the signal. Once the screen phosphorescence has died away you have lost the information, unless you use photographic techniques.

So, why not use an ultraviolet recorder? For slow response systems this may be adequate, although the recording paper is expensive. However, the speed of a UV recorder is about as slow compared to a 'scope as a bus is to Concorde! Perhaps we can use the oscilloscope in a different way?

The Digital Oscilloscope

One of the ways in which an oscilloscope can be enhanced is by fitting a 'storage' tube. This is very expensive, does not provide a permanent picture and can be damaged by misuse.

The logical alternative is to use computer technology and convert the analogue signal into a digital one with an A to D converter, store the resulting digital signal in RAM and then convert back through a D to A for display. The information stored can then be captured, replayed, or otherwise inspected *ad infinitum* on a standard 'scope tube. We can even take small parts of the stored signal and expand them — something no other storage system can do.

All this sounds too good to be true — there has to be a catch somewhere. That catch is a phenomenon called aliasing. Consider what happens when you increase the frequency of a sinusoidal input; to make things simple, let's assume that the A to D is sampling every microsecond. If the frequency of the input signal is 10 kHz, then each cycle on the display will consist of some 100 dots and the waveform will appear smooth and undistorted. If we now increase the input frequency to 100 kHz, we reduce the number of samples per cycle to 10 and the nature of the trace will become obvious. Worse still, if the input signal is greater than half that of the sampling frequency, the display will actually appear to be of a *lower* frequency!

The main purpose of producing this program for classroom demonstration was to show this together with the effects of limited resolution on a display system.

The Program Criteria

As well as demonstrating the effects of resolution and aliasing, the program was also designed to show the modes of oscilloscope operation: single or dual trace, chopped or alternate sampling and to show the effects of dot-joining on the displayed waveform.

The entire program is menu driven and the user can select any of the modes of operation from this main display (see the accompanying photographs).

For any of the chosen modes of operation, test data concerning the frequency of the input and its relative phase can be input together with joined or un-joined traces. The main flowchart for the program is given in Fig. 1.

It can be seen immediately from the photographs that the resolution of the demonstration is poor; it uses the equivalent of a four-bit A to D (five-bit in the case of single trace) as compared to at least eight-bit converters in a digital 'scope.

TECHNICAL DETAILS

The program listed is written specifically for the Commodore PET computer fitted with the PIC CHIP available from Insel Computers. Routines are available which will perform the double density graphics facility of the PIC CHIP, both in BASIC and machine code, but this method is somewhat easier!

The PIC CHIP is enabled by the SYS 36864 command at the start of the program, line 350, and the following four functions are used at various points in the program.

!WP This plots a pixel (¼ sized block) at a screen position of X,Y ***relative*** to the chosen origin.

!WL This draws a line of pixel points between X1, Y1 and the chosen X2, Y2 position.

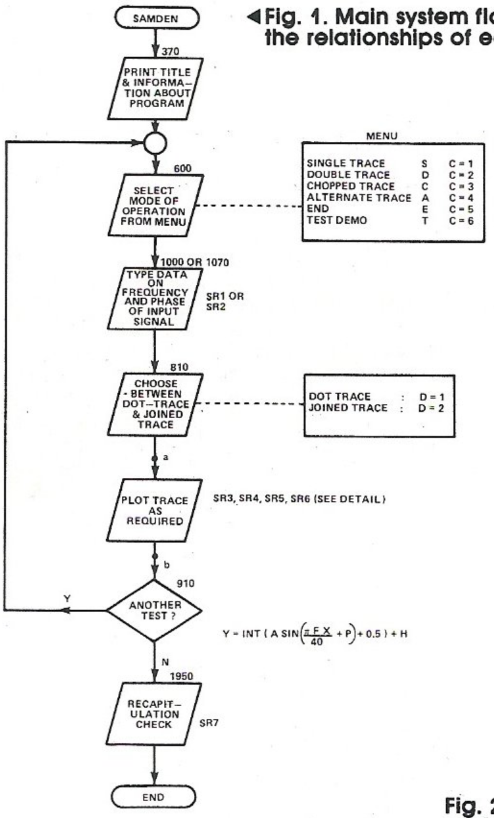
!WC As !WL but in second and subsequent calls, the new value of X1, Y1 is automatically set as the old value of X2, Y2.

!CW This positions the cursor so that normal PRINT commands can be combined with traces produced by PIC CHIP.

Although the resolution is limited to a quarter sized block, the cost of fitting a high resolution graphics option to the PET is excessive enough to make the existing situation acceptable.

The main flowchart shown in Fig. 1 indicates the main areas of the program and the functions that they can be expected to perform. The second flowchart in Fig. 2 is an expanded detail of the section between points 'a' and 'b' on Fig. 1.

All the main system variables are declared at the start of the program and the various routines are all commented.



◀**Fig. 1. Main system flowchart showing the relationships of each main section.**

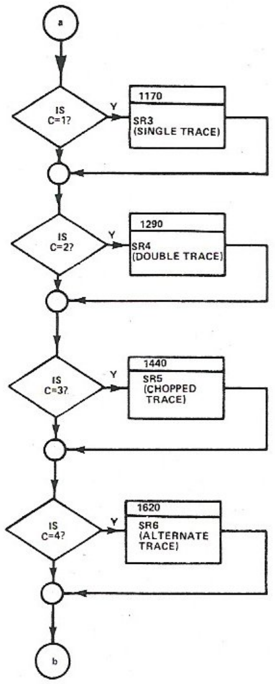


Fig. 2. Expanded detail between 'a' and 'b' of Fig. 1.▲

HOW TO USE THE PROGRAM

One way or another, you should now be ready to test the program. Although the program behaves exactly like a digital storage 'scope in most respects, you ***cannot*** alter the sampling rate of the A to D converter. This, however, is of little importance in a simulation such as this because the display obtained is unchanged if both input frequency and sampling frequency are changed by the same factor.

It is worth outlining the component parts of a digital storage 'scope and the effect each has on the overall performance.

1) The resolution of the A to D converter controls the resolution of the Y axis.
2) The amount of memory available for signal storage determines the X axis resolution.
3) The D to A converter must have the same resolution as the A to D.
4) The clock for each of the converters should be capable of independent adjustment to give control over sampling rate and allow a clean display.

The problem of aliasing is common to ***all*** sampling methods and should be borne in mind whenever connecting your micro to the outside world. Shannon's sampling theorem states that the minimum sampling frequency should be ***twice*** the frequency of the input frequency.

Doubtless the problem can be expanded and improved upon but in its current form it provides a useful and demonstrable system which contains all the major features of a true digital storage 'scope. ▶

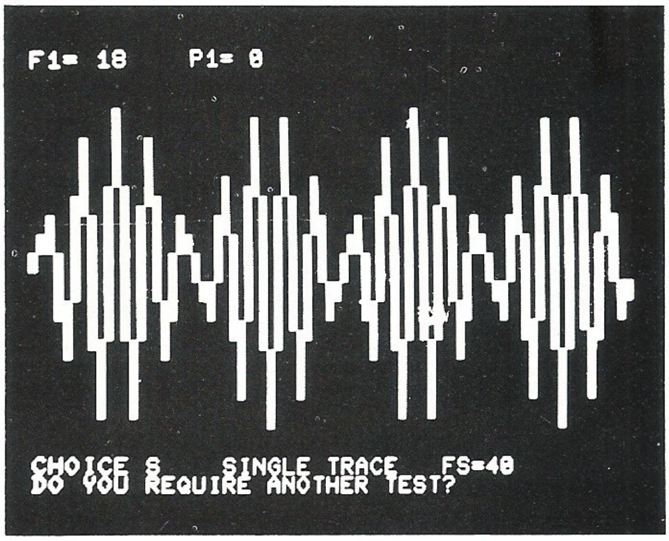
PROGRAM STRUCTURE

Subroutine	Function	Starts at Line
SRI	Single trace data input	1000
SR2	Dual trace data input	1070
SR3	Single trace plot	1170
SR4	Dual trace plot	1290
SR5	Chopped trace plot	1440
SR6	Alternate trace plot	1620
SR7	Exit routines	1950
SR8	Demonstration	2080
SR9	Delay	2410
SR10/11	Temporary store	1820/1890

```
170Ø  REM ** VARIABLES AND FUNCTIONS
180Ø  A=Ø:REM ** AMPLITUDE OF WAVEFORM
190Ø  C=Ø:REM ** MODE SELECTION CONTROL
200Ø  D=Ø:REM ** DOT JOINING CONTROL
210Ø  H=Ø:REM ** X AXIS HEIGHT
220Ø  K=PI/40:REM ** CONSTANT USED IN DEF
230Ø  F1=Ø:REM ** FREQUENCY OF UPPER TRACE
240Ø  F2=Ø:REM ** FREQUENCY OF LOWER TRACE
250Ø  P1=Ø:REM ** PHASE OF UPPER TRACE IN RADS
260Ø  P2=Ø:REM ** PHASE OF LOWER TRACE IN RADS
270Ø  XØ=Ø:YØ=Ø:REM ** USER ORIGIN
280Ø  X1=Ø:Y1=Ø:REM ** USED FOR DOT JOINING
290Ø  X2=Ø:Y2=Ø:REM ** USED WITH !WL AND !WC
300Ø  X3=Ø:Y3=Ø:REM ** TEMP STORE FOR X1,Y1
      UPPER TRACE
310Ø  X4=Ø:Y4=Ø:REM ** TEMP STORE FOR X1,Y1
      LOWER TRACE
320Ø  DEF FNA(X)=A*SIN(K*F1*X+P1)
330Ø  DEF FNB(X)=A*SIN(K*F2*X+P2)
340Ø  DEF FNC(Y9)=INT(Y9+.5)+H
350Ø  SYS 36864:REM ** TURN ON PIC CHIP
360Ø  REM ** TITLE AND MODE CHOICE
370Ø  PRINT "[CLS]"
380Ø  PRINT:PRINT "[3 SPC]DIGITAL STORAGE
      OSCILLOSCOPE"
390Ø  PRINT "[11 SPC]SIMULATION"
400Ø  PRINT:PRINT
410Ø  PRINT "ALL THE INPUTS USED IN THIS
      PROGRAM HAVE SINE WAVEFORMS"
420Ø  PRINT "YOU WILL BE ASKED TO SELECT THE
      TYPE OF TRACE REQUIRED"
430Ø  PRINT "YOU MAY CHOOSE THE INPUT
      FREQUENCIES AND PHASES"
440Ø  PRINT "YOU MAY ALSO CHOOSE BETWEEN DOT
      AND CONTINUOUS PLOTS"
450Ø  PRINT
460Ø  PRINT "FIRST TIME USERS ARE ADVISED TO
      SELECT 'T' AT THE FIRST RUN"
470Ø  PRINT "THIS DEMO WILL GIVE YOU AN IDEA
      OF THE FACILITIES AVAILABLE"
480Ø  PRINT
490Ø  PRINT "USE YOUR OWN TEST DATA TO
      EXAMINE"
500Ø  PRINT "SAMPLING DISTORTION AND
      ALIASING"
510Ø  GOSUB 24ØØ:GOSUB 24ØØ:REM ** DELAY
520Ø  PRINT:PRINT "[2 SPC]CODE[3 SPC]
      DESCRIPTION[6 SPC]SAMPLING FREQ."
530Ø  PRINT "[4 SPC]T[4 SPC]DEMO TEST PROG."
540Ø  PRINT "[4 SPC]S[4 SPC]SINGLE TRACE
      [8 SPC]4Ø"
550Ø  PRINT "[4 SPC]D[4 SPC]DOUBLE TRACE
      [8 SPC]4Ø"
560Ø  PRINT "[4 SPC]C[4 SPC]CHOPPED TRACE
      [7 SPC]2Ø"
570Ø  PRINT "[4 SPC]A[4 SPC]ALTERNATE TRACE
      [5 SPC]4Ø"
580Ø  PRINT "[4 SPC]E[4 SPC]END OF PROGRAM"
590Ø  PRINT:PRINT
600Ø  PRINT "SELECT FROM T,S,D,C,A,OR E"
610Ø  GET A$:IF A$="" THEN 61Ø
620Ø  C=Ø
630Ø  IF A$="S" THEN C=1
640Ø  IF A$="D" THEN C=2
650Ø  IF A$="C" THEN C=3
660Ø  IF A$="A" THEN C=4
670Ø  IF A$="T" THEN C=6
680Ø  IF A$="E" THEN GOSUB 195Ø:REM ** END
      ROUTINE, SR7
690Ø  IF C=5 THEN 244Ø:REM ** END
700Ø  IF C=6 THEN GOSUB 2Ø7Ø:GOTO 91Ø
710Ø  REM ** WRONG ENTRY
720Ø  PRINT
730Ø  IF C=Ø THEN PRINT "INCORRECT RESPONSE":
      GOTO 59Ø
740Ø  REM ** ONE TRACE OR TWO
750Ø  PRINT
```

```
760Ø  IF C=1 THEN GOSUB 99Ø:REM ** SINGLE TRAC
      INPUT DATA
770Ø  IF C<>1 THEN GOSUB 1Ø6Ø:REM ** DUAL TRAC
      INPUT DATA
780Ø  PRINT
790Ø  REM ** DOT JOINING
800Ø  D=Ø
810Ø  PRINT "DO YOU REQUIRE DOT JOINING";
820Ø  INPUT B$:IF LEFT$(B$,1)="Y" THEN D=1
830Ø  REM ** MAIN PROGRAM
840Ø  PRINT "[CLS]"
850Ø  REM ** MODE SELECTION
860Ø  IF C=1 THEN GOSUB 116Ø:REM ** SINGLE
      TRACE MODE, SR3
870Ø  IF C=2 THEN GOSUB 128Ø:REM ** DOUBLE
      TRACE MODE, SR4
880Ø  IF C=3 THEN GOSUB 143Ø:REM ** CHOPPED
      TRACE MODE, SR5
890Ø  IF C=4 THEN GOSUB 161Ø:REM ** ALTERNATE
      TRACE MODE, SR6
900Ø  REM ** DO IT AGAIN
910Ø  PRINT "DO YOU REQUIRE ANOTHER TEST?";
920Ø  GET A$:IF A$="" THEN 92Ø
930Ø  PRINT "[CLS]"
940Ø  REM ** YES OR NO
950Ø  IF LEFT$(A$,1)="Y" THEN 52Ø
960Ø  IF LEFT$(A$,1)="N" THEN GOSUB 195Ø:
      REM ** SR7
970Ø  IF C=5 THEN 244Ø
980Ø  PRINT "INCORRECT RESPONSE. PLEASE TYPE
      Y OR N":PRINT:GOTO 91Ø
990Ø  REM ** SINGLE TRACE INPUT DATA, SR1
1000Ø PRINT:PRINT "INPUT FREQUENCY AND PHASE
      DATA"
1010Ø PRINT
1020Ø INPUT "F=";F1
1030Ø INPUT "P=";P1
1040Ø RETURN
1050Ø REM
1060Ø REM ** DUAL TRACE INPUT DATA, SR2
1070Ø PRINT:PRINT "UPPER TRACE FREQUENCY AND
      PHASE DATA"
1080Ø PRINT
1090Ø INPUT "F=";F1
1100Ø INPUT "P=";P1
1110Ø PRINT:PRINT "LOWER TRACE DATA"
1120Ø PRINT
1130Ø INPUT "F=";F2
1140Ø INPUT "P=";P2
1150Ø RETURN
1160Ø REM ** SINGLE TRACE PLOT, SR3
1170Ø A=16:H=23:X1=Ø:Y1=FNC(FNA(Ø))
1180Ø PRINT:PRINT "F1=";F1,"P1=";P1
1190Ø FOR X=Ø TO 8Ø STEP 2
1200Ø Y=FNC(FNA(X))
1210Ø IF D=Ø THEN D=Ø:!WP:REM ** TRACE WITHOUT
      DOT JOINING
1220Ø REM ** D=Ø IS A DUMMY INSTRUCTION
1230Ø IF D=1 THE X2=X:Y2=Y:!WC:REM ** !WC IS
      THE AUTO LINE JOINING FUNCTION
1240Ø NEXT X
1250Ø X=Ø:Y=1:!CW
1260Ø PRINT "CHOICE S[3 SPC]SINGLE TRACE
      [3 SPC]FS=4Ø"
1270Ø RETURN
1280Ø REM ** DOUBLE TRACE PLOTS, SR4
1290Ø A=8:H=37:X3=Ø:Y3=FNC(FNA(Ø))
1300Ø PRINT "F1=";F1,"P1=";P1,
      "F2=";F2,"P2=";P2
1310Ø FOR X=Ø TO 8Ø STEP 2
1320Ø H=37
1330Ø Y=FNC(FNA(X))
1340Ø IF D=Ø THEN D=Ø:!WP
1350Ø IF D=1 THEN GOSUB 181Ø
1360Ø H=13
1370Ø Y=FNC(FNB(X))
1380Ø !WP
```

```
1390  NEXT X
1400  X=0:Y=1:!CW:REM ** !CW POSITIONS THE
      CURSOR
1410  PRINT "CHOICE D[3 SPC]DOUBLE TRACE
      [3 SPC]FS=40"
1420  RETURN
1430  REM ** CHOPPED TRACE PLOTS, SR5
1440  A=8:X3=0
1450  H=37:Y3=FNC(FNA(0))
1460  X4=2
1470  H=13:Y4=FNC(FNB(2))
1480  PRINT "F1=";F1,"P1=";P1,
      "F2=";F2,"P2=";P2
1490  FOR X=0 TO 80 STEP 2
1500  H=37:Y=FNC(FNA(X))
1510  IF D=0 THEN D=0:!WP
1520  IF D=1 THEN GOSUB 1810
1530  X=X+2
1540  H=13:Y=FNC(FNB(X))
1550  IF D=0 THEN D=0:!WP
1560  IF D=1 THEN GOSUB 1880
1570  NEXT X
1580  X=0:Y=1:!CW
1590  PRINT "CHOICE C[3 SPC]CHOPPED TRACE
      [3 SPC]FS=20"
1600  RETURN
1610  REM ** ALTERNATE TRACE PLOTS, SR6
1620  A=8:X3=0
1630  H=37:Y3=FNC(FNA(0))
1640  H=13:Y4=FNC(FNB(0))
1650  X4=0
1660  PRINT "F1=";F1,"P1=";P1,
      "F2=";F2,"P2=";P2
1670  FOR X=0 TO 80 STEP 2
1680  H=37:Y=FNC(FNA(X))
1690  IF D=0 THEN D=0:!WP
1700  IF D=1 THEN GOSUB 1810
1710  NEXT X
1720  REM
1730  FOR X=0 TO 80 STEP 2
1740  H=13:Y=FNC(FNB(X))
1750  IF D=0 THEN D=0:!WP
1760  IF D=1 THEN GOSUB 1880
1770  NEXT X
1780  X=0:Y=1:!CW
1790  PRINT "CHOICE A[3 SPC]ALTERNATE TRACE
      [3 SPC]FS=40"
1800  RETURN
1810  REM ** TEMP STORE, SR10, USED WITH SR4,
      SR5, SR6
1820  X1=X3:Y1=Y3
1830  X2=X:Y2=Y
1840  !WL
1850  X3=X2
1860  Y3=Y2
1870  RETURN
1880  REM ** TEMP STORE, SR11, USED WITH SR5
      AND SR6
1890  X1=X4:Y1=Y4
1900  X2=X:Y2=Y
1910  !WL
1920  X4=X2:Y4=Y2
1930  RETURN
1940  REM ** END SUBROUTINE, SR7
1950  PRINT "[CLS]"
1960  PRINT "END OF PROGRAM"
1970  C=5:REM ** CONTROL FOR END
1980  PRINT:PRINT "YOU SHOULD BY NOW
      UNDERSTAND THE FOLLOWING"
1990  PRINT:PRINT "1. CHOPPED AND ALTERNATE
      TRACE MODES OF OPERATION"
2000  PRINT "2. THE EFFECT OF LIMITATIONS IN
      X & Y RESOLUTION"
2010  PRINT "3. DISTORTION EFFECTS DUE TO A
      SMALL NO OF SAMPLES PER CYCLE"
2020  PRINT "4. WHAT IS MEANT BY 'ALIASING'"
2030  PRINT:PRINT:PRINT "IF NECESSARY REPEAT
      SOME OF THE TESTS USING NEW DATA"
2040  PRINT:PRINT
2050  PRINT "GOODBYE"
2060  RETURN
2070  REM ** TEST DEMO, SR8
2080  PRINT "[CLS]"
2090  PRINT "DEMO TEST PROGRAM"
2100  C=1:D=0
2110  READ F1,P1
2120  GOSUB 1160:REM ** SR3
2130  GOSUB 2400:REM ** SR9
2140  REM
2150  C=2:D=1
2160  READ F1,P1,F2,P2
2170  PRINT "[CLS]"
2180  GOSUB 1280:REM ** SR4
2190  GOSUB 2400
2200  REM
2210  C=3
2220  READ F1,P1,F2,P2
2230  PRINT "[CLS]"
2240  GOSUB 1430:REM ** SR5
2250  GOSUB 2400
2260  REM
2270  C=4
2280  READ F1,P1,F2,P2
2290  PRINT "[CLS]"
2300  GOSUB 1610:REM ** SR6
2310  GOSUB 2400:GOSUB2400:REM ** DOUBLE NORMA
      DELAY
2320  PRINT "[CLS]"
2330  RESTORE
2340  REM ** DATA FOR DEMO
2350  DATA 5,2
2360  DATA 12,0,12,0
2370  DATA 8,0,12,0
2380  DATA 37,1,53,1
2390  RETURN
2400  REM ** DELAY SUBROUTINE, SR9
2410  T1=TI:REM ** TI IS PET'S INTERNAL CLOCK
      COUNTER
2420  IF TI-T1<420 THEN 2420
2430  RETURN
2440  REM ** RESTORE NORMAL FUNCTIONS
2450  !CO:REM ** PIC CHIP OFF
2460  SYS 45056:REM ** TOOLKIT ON
2470  REM ** THAT'S IT!
2480  END
```



```
F1= 18    P1= 0

CHOICE S    SINGLE TRACE    FS=40
DO YOU REQUIRE ANOTHER TEST?
```

A single trace plot with dot joining. The sampling frequency is 40Hz and the highly distorted display is a result of beating between the alias frequency of 22Hz and the input frequency of 18Hz. To avoid this kind of distortion, a sampling frequency of some five times that of the highest input component must be used.

Don Thomasson

SCREEN PRINT

Add a screen copy facility to your Exidy Sorcerer/Epson printer system.

Not so very long ago I wrote a short article called 'Getting Into Print'. In this I stated that it was not possible to transfer the contents of the Sorcerer screen onto the MX80 printer. As statements of this kind seem to have a habit of turning on one I was not surprised, some few days before the article appeared, to find that there is, indeed, a method of getting the Sorcerer to print its screen onto the MX80!

So, in an attempt to put things right, here is the necessary information and a short routine so that you can all benefit from the discovery.

Inside The Epson

The Epson MX80 printer employs a pair of microprocessors to control its actions, an 8049 and an 8041. The program for the 8049 is quite large, extending to 6K, and the behaviour of the printer can be varied extensively by using different programs.

The original program provided a number of type styles, vertical and horizontal tabulation, variable line pitch and a number of other facilities. A later version dropped some of these facilities, but added 'Bit Mode', of which more anon. The most recent version seen at the time of writing covers most of the features offered by either of its predecessors, plus italic type and reverse video types. Since none of these programs appear to have identifying references, it is necessary to be specific when enquiring about them.

It should be added that some of the programs are available in three-ROM form, and to use these it is necessary to cut a link on the main circuit board to disable the program held in the 8049 microprocessor. Others are supplied as a 4K ROM and a specially programmed 8049. A little confusing, until you get the main idea.

Bit Mode

The most interesting facility offered by these programs is 'Bit Mode', which allows every dot position in the whole printout area to be defined as black and white. The only snag is that this can involve quite a lot of dots, up to about 7,400 per square inch. An A4 page could accommodate 650,000 dots, and storing that would involve more than 80K of storage!

For some types of work, such as graph plotting, the amount of data can be cut down by specifying the position of black dots and counting off the white dots from the left-hand margin, but even that can involve some complex programming.

For those who find themselves frustrated by their inability to make adequate use of Bit Mode, Screenprint may provide an answer.

HOW TO USE THE PROGRAM

Screenprint is a machine code program for the Z80, and though described here for the

Sorcerer it can be adapted quite easily for other computers with memory mapped displays.

The Sorcerer stores its screen data in 1,920 bytes of RAM, each byte relating to a given character position on the 30-line by 64-character screen. Each byte holds an ASCII code, which is translated into a pattern of 64 dots by reference to the standard character RAM or the graphics RAM. The latter can be set by software to any desired pattern, though the lower half of the graphics range is reset to standard forms when Clear Screen is called.

Screenprint begins by setting IX to F080 Hex, the start of screen RAM, this being the screen pointer. An output sequence 1B, 41, 08 is then sent to the printer to set up a line spacing of 8/72". Some, but not all, MX80 programs require this to be followed by the sequence 1B,32 to confirm the setting.

Bit Mode with 512 characters per line is then set by the sequence 1B,4C,00,02. This has to be done afresh for every line.

HL is now set to F800 Hex, the start of the character definition area, and the first character is read into A. The result is multiplied by eight and added to HL to form a pointer, each character definition occupying eight bytes.

The next operation involves storing the eight bytes defining the character, after which the first bit of each of the eight bytes is assembled in A to form the first data output to the printer. This process is necessary because the bytes define eight horizontal dots, whereas the printer requires eight vertical dots.

When A has been set, a NOP byte is provided; changing this to 2F reverses the print action to white on black, like the screen image, but black on white is clearer.

The byte is then output, and the program loops to J4 to assemble the next output byte. When eight bytes have been transferred, a jump is made to J2 to obtain the next character.

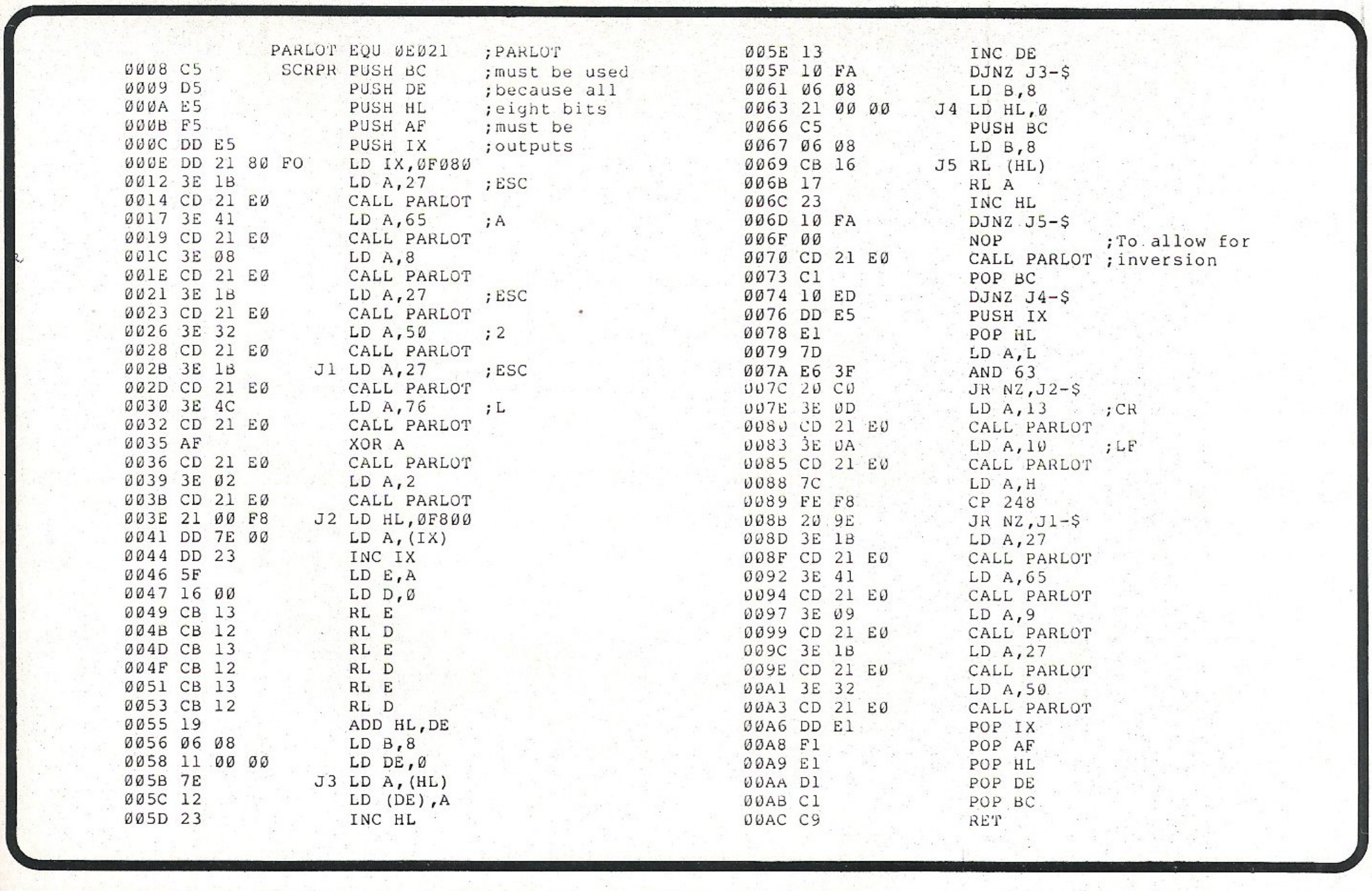
When the line is complete, IX AND 3F = 0 being used to induce a jump back to J1 to start a fresh line, unless IX has reached F800 Hex which is one location beyond the end of screen RAM.

Finally, a sequence 1B,41,9 is output to restore the line spacing to the normal 1/6" pitch. Here again, some programs may require the sequence 1B,32 to confirm the new setting.

TECHNICAL DETAILS

The time taken to print a screen is about one minute. The print quality produced is good. There is a slight discrepancy between the vertical width of a line and the nearest vertical spacing, but this is not too obvious.

An important consideration is that a manual call to Screenprint will show up on the screen and thus on the printed copy, so it is usually wise to arrange for an automatic call at an appropriate point in the program which creates the display. If this is not possible, then the intruding text can be covered by using cursor left to regain the start of the line, spacing forward to erase the text, and then pressing Return. The Monitor does not object, and ignores the redundant part of the input.

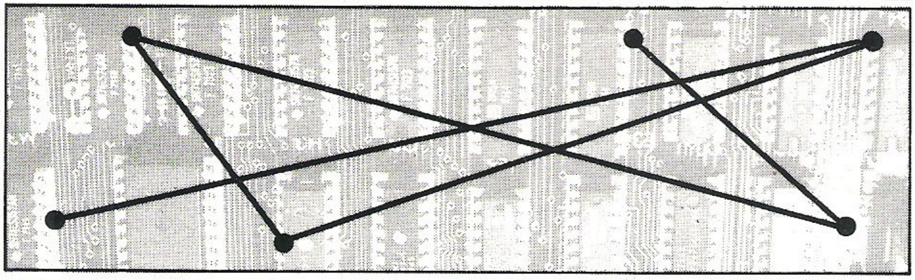


```
                    PARLOT EQU ØEØ21   ;PARLOT
ØØØ8 C5              SCRPR PUSH BC     ;must be used
ØØØ9 D5                    PUSH DE     ;because all
ØØØA E5                    PUSH HL     ;eight bits
ØØØB F5                    PUSH AF     ;must be
ØØØC DD E5                 PUSH IX     ;outputs
ØØØE DD 21 8Ø FO           LD IX,ØFØ8Ø
ØØ12 3E 1B                 LD A,27     ;ESC
ØØ14 CD 21 EØ              CALL PARLOT
ØØ17 3E 41                 LD A,65     ;A
ØØ19 CD 21 EØ              CALL PARLOT
ØØ1C 3E Ø8                 LD A,8
ØØ1E CD 21 EØ              CALL PARLOT
ØØ21 3E 1B                 LD A,27     ;ESC
ØØ23 CD 21 EØ              CALL PARLOT
ØØ26 3E 32                 LD A,5Ø     ;2
ØØ28 CD 21 EØ              CALL PARLOT
ØØ2B 3E 1B              J1 LD A,27     ;ESC
ØØ2D CD 21 EØ              CALL PARLOT
ØØ3Ø 3E 4C                 LD A,76     ;L
ØØ32 CD 21 EØ              CALL PARLOT
ØØ35 AF                    XOR A
ØØ36 CD 21 EØ              CALL PARLOT
ØØ39 3E Ø2                 LD A,2
ØØ3B CD 21 EØ              CALL PARLOT
ØØ3E 21 ØØ F8           J2 LD HL,ØF8ØØ
ØØ41 DD 7E ØØ              LD A,(IX)
ØØ44 DD 23                 INC IX
ØØ46 5F                    LD E,A
ØØ47 16 ØØ                 LD D,Ø
ØØ49 CB 13                 RL E
ØØ4B CB 12                 RL D
ØØ4D CB 13                 RL E
ØØ4F CB 12                 RL D
ØØ51 CB 13                 RL E
ØØ53 CB 12                 RL D
ØØ55 19                    ADD HL,DE
ØØ56 Ø6 Ø8                 LD B,8
ØØ58 11 ØØ ØØ              LD DE,Ø
ØØ5B 7E                 J3 LD A,(HL)
ØØ5C 12                    LD (DE),A
ØØ5D 23                    INC HL
ØØ5E 13                    INC DE
ØØ5F 1Ø FA                 DJNZ J3-$
ØØ61 Ø6 Ø8                 LD B,8
ØØ63 21 ØØ ØØ           J4 LD HL,Ø
ØØ66 C5                    PUSH BC
ØØ67 Ø6 Ø8                 LD B,8
ØØ69 CB 16              J5 RL (HL)
ØØ6B 17                    RL A
ØØ6C 23                    INC HL
ØØ6D 1Ø FA                 DJNZ J5-$
ØØ6F ØØ                    NOP            ;To allow for
ØØ7Ø CD 21 EØ              CALL PARLOT ;inversion
ØØ73 C1                    POP BC
ØØ74 1Ø ED                 DJNZ J4-$
ØØ76 DD E5                 PUSH IX
ØØ78 E1                    POP HL
ØØ79 7D                    LD A,L
ØØ7A E6 3F                 AND 63
ØØ7C 2Ø CØ                 JR NZ,J2-$
ØØ7E 3E ØD                 LD A,13        ;CR
ØØ8Ø CD 21 EØ              CALL PARLOT
ØØ83 3E ØA                 LD A,1Ø        ;LF
ØØ85 CD 21 EØ              CALL PARLOT
ØØ88 7C                    LD A,H
ØØ89 FE F8                 CP 248
ØØ8B 2Ø 9E                 JR NZ,J1-$
ØØ8D 3E 1B                 LD A,27
ØØ8F CD 21 EØ              CALL PARLOT
ØØ92 3E 41                 LD A,65
ØØ94 CD 21 EØ              CALL PARLOT
ØØ97 3E Ø9                 LD A,9
ØØ99 CD 21 EØ              CALL PARLOT
ØØ9C 3E 1B                 LD A,27
ØØ9E CD 21 EØ              CALL PARLOT
ØØA1 3E 32                 LD A,5Ø
ØØA3 CD 21 EØ              CALL PARLOT
ØØA6 DD E1                 POP IX
ØØA8 F1                    POP AF
ØØA9 E1                    POP HL
ØØAA D1                    POP DE
ØØAB C1                    POP BC
ØØAC C9                    RET
```

Paul B Kaufman

LINE PLOTTER

Making more out of the Microtan's chunky graphics.



This pair of machine code routines allows lines to be drawn on the Microtan screen at high speed between any two points and at any angle. Both routines are directly accessible from BASIC using the USR command.

Listing 1 is an extended version of the Microtan manual's graphics routine. XCOORD and YCOORD are set up with the x and y co-ordinates respectively. MODE is set to one of three values:

1)$FF — Erases graphics dot at position XCOORD, YCOORD

2)$01 — Sets graphics dot at position XCOORD, YCOORD

3)$00 — Tests graphics dot at position XCOORD, YCOORD

Mode is returned as 1 if bit is set, 0 if not set.

This routine may be called independently of the program in Listing 2.

Drawing Lines

There are several ways of drawing lines on a microcomputer. One way would be to start at one end of the line and continually increment the x and y co-ordinates while plotting until the end of the line is reached. This may be simply expressed as:

$$X=X0+I*(X1-X0)$$
$$Y=Y0+I*(Y1-Y0)$$

where X and Y are new co-ordinates to be plotted, X0 and Y0 are start co-ordinates of the line, X1 and Y1 are end co-ordinates of the line and I is the increment which varies from 0 to 1.

This may be considered as the fraction of the whole line which is to be plotted at any one time. Although this method is reasonably simple to code in BASIC, problems arise for the machine-code programmer in handling fractional numbers which are needed to represent I.

Another method, which is the one used here, is to repeatedly divide the line in half, saving the results of each division until division cannot go any further. The resulting co-ordinates are a single point. This point is then plotted and the division process then starts again and continues until the end of the line is reached. The most efficient method of storing the intermediate points is to push them onto the stack. This makes for faster processing and economy of memory usage.

HOW TO USE THE PROGRAMS

Enter the code from Listings 1 and 2. If you are using BASIC, answer 'MEMORY SIZE' with 7670 to protect the machine code area. Listings 3 and 4 are the same routines expressed as data statements which may be easier for the BASIC programmer. For convenience a 'clear-screen' routine is included in the listings. The following steps are then followed to run the routines.

1) To clear screen: JSR$1F94 or in BASIC:

```
POKE 34,148:POKE 35,31:DUM=USR(DUM)
```

2) Set, Clear or Test graphics bit: Enter x and y co-ordinates at $40 and $41 with values between $0-$3F. Enter the MODE value at $3F as above and JSR$1F40. If testing a bit, checking $3F will tell you if the bit is set or not. In BASIC use:

```
POKE 63,x co-ordinate:POKE 65,
     y co-ordinate (0-63)
POKE 63,mode(0=test bit,1=set
     bit,255=clear bit)
POKE 34,64:POKE 35,31:DUM=USR(DUM)
```

3) Draw or Delete line: Set MODE ($3F) to required value. Set $40 and $41 with x and y co-ordinates of the start of the line and $42 and $43 with co-ordinates of the end of the line then JSR$1E00. In BASIC do:

```
POKE 63,mode:POKE 64,start
     x:POKE 65,start y
POKE 66,end x:POKE 67,end y
POKE 34,00:POKE 35,30:DUM=USR(DUM)
```

These routines are a useful tool for the BASIC and machine-code programmer alike and could be the basis of many interesting games or demonstration programs. They also show what can be done with the limited (64 by 64) definition of Tangerine's chunky graphics.

References And Further Reading

The Mathematics of Computer Graphics, Byte, September 1978 and July 1979.

Vector Graphics for Raster Displays, Byte, October 1980.

```
003F 00           MODE   BYTE
0040 00           XCOORD BYTE
0041 00           YCOORD BYTE
0042 00           VDULO  BYTE
0043 00           VDUHI  BYTE

1F40 AD F0 BF     START  LDA $BFF0   TURN GRAPHICS
1F43 A9 3F               LDA #$3F    ON
1F45 C5 40               CMP $40
1F47 30 40               BMI $1F89   CO-ORDINATES
1F49 EA                  NOP         OUT OF RANGE
1F4A 38                  SEC         CALCULATE
1F4B E5 41               SBC $41     SCREEN
1F4D EA                  NOP         ADDRESS
1F4E 48                  PHA
1F4F 29 03               AND #$03
1F51 AA                  TAX
1F52 68                  PLA
1F53 EA                  NOP
1F54 29 3C               AND #$3C
1F56 0A                  ASL A
1F57 0A                  ASL A
1F58 0A                  ASL A
1F59 85 42               STA $42
1F5B A9 02               LDA #$02
1F5D 85 43               STA $43
1F5F 90 02               BCC $1F63
1F61 E6 43               INC $43
1F63 A5 40               LDA $40
1F65 4A                  LSR A       CALCULATE
1F66 A8                  TAY         GRAPHICS BYTE
1F67 A9 01               LDA #$01
1F69 90 01               BCC $1F6C
1F6B 0A                  ASL A
1F6C CA                  DEX
1F6D 30 04               BMI $1F73
1F6F 0A                  ASL A
1F70 0A                  ASL A
1F71 D0 F9               BNE $1F6C
1F73 AA                  TAX
1F74 A5 3F               LDA $3F     CHECK MODE
1F76 30 08               BMI $1F80   CLEAR PIXEL ?
1F78 F0 10               BEQ $1F8A   TEST PIXEL ?
1F7A 8A                  TXA
1F7B 11 42               ORA ($42),Y
1F7D 91 42               STA ($42),Y TURN PIXEL ON
1F7F 60                  RTS
1F80 8A           CLEAR  TXA         DELETE PIXEL
1F81 31 42               AND ($42),Y
1F83 F0 04               BEQ $1F89
1F85 51 42               EOR ($42),Y
1F87 91 42               STA ($42),Y
1F89 60                  RTS
1F8A 8A           TEST   TXA         TEST PIXEL
1F8B 31 42               AND ($42),Y
1F8D F0 04               BEQ $1F93
1F8F A9 01               LDA #$01
1F91 85 3F               STA $3F
1F93 60                  RTS
1F94 A9 00        CLEAR  LDA #$00    CLEAR SCREEN
1F96 AA                  TAX
1F97 9D 00 02            STA $200,X
1F9A 9D 00 03            STA $300,X
1F9D E8                  INX
1F9E D0 F7               BNE $1F97
1FA0 60                  RTS
```

Listing 1.

```
0040 00           STARTX BYTE
0041 00           STARTY BYTE
0042 00           ENDX   BYTE
0043 00           ENDY   BYTE

1E00 A9 FF        ENTRY  LDA #$FF    PREPARE STACK
1E02 48                  PHA
1E03 48                  PHA
1E04 A5 40               LDA $40     IF START>END
1E06 C5 42               CMP $42     THEN CHANGE
1E08 30 10               BMI $1E1A
1E0A A8                  TAY
1E0B A5 42               LDA $42
1E0D 85 40               STA $40
1E0F 98                  TYA
1E10 85 42               STA $42
1E12 A4 43               LDY $43
1E14 A5 41               LDA $41
1E16 84 41               STY $41
1E18 85 43               STA $43
1E1A A5 42        LOOP   LDA $42     STARTX=ENDX ?
1E1C C5 40               CMP $40
1E1E F0 2B               BEQ $1E4B   YES
1E20 48                  PHA         NO
1E21 18                  CLC
1E22 65 40               ADC $40     FIND CENTRE X
1E24 4A                  LSR A       CO-ORDINATE
1E25 85 42               STA $42     SAVE IT
1E27 AA                  TAX
1E28 E8                  INX
1E29 A5 43               LDA $43
1E2B 48                  PHA
1E2C A5 41               LDA $41     STARTY=ENDY ?
1E2E C5 43               CMP $43
1E30 F0 11               BEQ $1E43
1E32 18                  CLC         NO
1E33 65 43               ADC $43     FIND CENTRE Y
1E35 4A                  LSR A       CO-ORDINATE
1E36 85 43               STA $43     SAVE IT
1E38 C5 41               CMP $41
1E3A B0 04               BCS $1E40
1E3C E6 43               INC $43
1E3E 50 03               BVC $1E43
1E40 18                  CLC
1E41 69 01               ADC #$01    ADJUST CENTRE
1E43 A8                  TAY         SAVE CO-ORD
1E44 8A                  TXA
1E45 48                  PHA
1E46 98                  TYA
1E47 48                  PHA
1E48 4C 1A 1E            JMP $1E1A   START AGAIN
1E4B A5 41               LDA $41
1E4D C5 43               CMP $43
1E4F F0 0D               BEQ $1E5E
1E51 A5 42               LDA $42     SAVE Y CO-ORD
1E53 48                  PHA
1E54 A5 43               LDA $43
1E56 48                  PHA
1E57 A6 40               LDX $40
1E59 A5 41               LDA $41
1E5B 4C 32 1E            JMP $1E32
1E5E 8A                  TXA
1E5F 48                  PHA
1E60 20 40 1F            JSR $1F40   PLOT CO-ORDS
1E63 68                  PLA         BRING BACK
1E64 AA                  TAX
1E65 68                  PLA
1E66 85 41               STA $41
1E68 68                  PLA
1E69 85 40               STA $40
1E6B C9 FF               CMP #$FF    NO MORE ?
1E6D D0 01               BNE $1E70
1E6F 60                  RTS
1E70 68                  PLA
1E71 85 43               STA $43
1E73 68                  PLA
1E74 85 42               STA $42
1E76 4C 1A 1E            JMP $1E1A   CONTINUE
```

Listing 2.

```
20000  FOR I=8000 TO 8096:READ V:POKE I,V:NEXT I
21000  DATA 173,240,191,169,63,197,64,48,64,234
21500  DATA 56,229,65,234,72,41,3,170,104,234,41,60
22000  DATA 10,10,10,133,66,169,2,133,67,144,2,230,67
22500  DATA 165,64,74,168,169,1,144,1,10,202,48,4,10,10
23000  DATA 208,249,170,165,63,48,8,240,16,138,17,66
23500  DATA 145,66,96,138,49,66,240,4,81,66,145,66
24000  DATA 96;138,49,66,240,4,169,1,133,63
24500  DATA 96,169,0,170,157,0,2,157,0,3
25000  DATA 232,208,247,96
```

Listing 3.

```
40000  FOR I=7680 TO 7800:READ V:POKE I,V:NEXT I
41000  DATA 169,255,72,72,165,64,197,66,48,16,168,165,66
41500  DATA 133,64,152,133,66,164,67,165,65,132,65,133
42000  DATA 67,165,66,197,64,240,43,72,24,101,64,74
42500  DATA 133,66,170,232,165,67,72,165,65,197,67,240
43000  DATA 17,24,101,67,74,133,67,197,65,176,4,230
43500  DATA 67,80,3,24,105,1,168,138,72,152,72,76,26
44000  DATA 30,165,65,197,67,240,13,165,66,72,165,67
44500  DATA 72,166,64,165,65,76,50,30,138,72,32,64,31
45000  DATA 104,170,104,133,65,104,133,64,201,255,208,1
45500  DATA 96,104,133,67,104,133,66,76,26,30
```

Listing 4.

Paul Williams

PET LISTER

Convert the PET's graphic symbols into CT's standard codes for a more readable listing.

When PET BASIC programs are listed on a printer, the cursor controls and shifted characters are printed as cryptic symbols which are often difficult to decipher. This machine code program lists BASIC programs on paper, spacing out the statements (if necessary) and showing cursor control and shifted characters as more easily identifiable characters; these generally correspond to the CT Standards.

A Clear Screen (normally ♡') is printed as '[CLS]'
A Cursor Down (normally 'Q') is printed as '[CD]'

Shifted characters are printed as their un-shifted versions but in square brackets ([ ]).

HOW TO USE THE PROGRAM

To list a BASIC program in this way, load the lister, type NEW and then load the BASIC program. If SYS 30000 is now typed, the program will be listed in this special way on the printer, as fast as a normal listing. After one SYS 30000, the area of memory used for the lister program will be protected from being overwritten by strings, as the top of memory pointers are set by the machine code program.

The lister can be entered using an assembler, or using TIM. If you are using an assembler on an 8K machine, change BEGIN= $7530 to BEGIN= $1EDC and execute the program with a SYS 7900 instead. Because a number of zero-page and ROM addresses are used by the program, it will not work without considerable alteration on the Old ROM machines.

TECHNICAL DETAILS

As BASIC statements in a program are stored as single bytes called 'tokens' (eg 128 for END and 153 for PRINT) to save memory and speed up the interpreter, reference has to be made to the ROM table of statements to print the correct characters for each command. If the lister finds a BASIC token byte while listing on the printer, it does not directly print it but finds the correct word in the ROM string starting at $C092.

At $758D in the lister, if the LDA #0 is changed to LDA #32 (A9 00 Hex to A9 20 Hex), a space will automatically be printed after each keyword (eg PRINT, GOTO,/,—,etc). This amendment can make program listings even clearer, but often it is better to leave $ 758D as LDA #0.

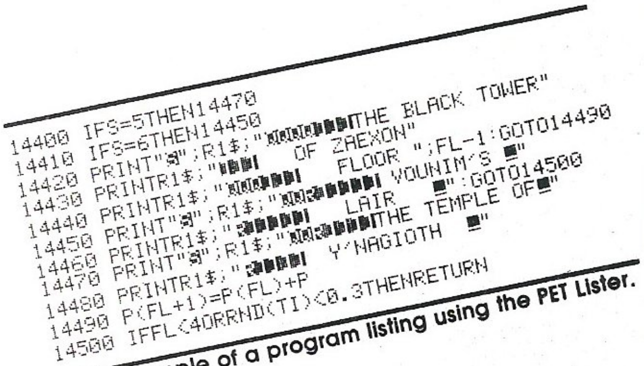
The actual method used by the lister is explained in the assembler listing, and the program takes just over 256 bytes.

The changes made to the cursor control characters in the output of the lister are shown in Table 1. All shifted graphics symbols are printed as unshifted characters in square brackets.

SYMBOL	MEANING	NEW CHARACTERS
Q	DOWN CURSOR	[CD]
]	CURSOR RIGHT	[CR]
◙	CURSOR UP	[CU]
▌	CURSOR LEFT	[CL]
S	HOME CURSOR	[HOME]
♡	CLEAR SCREEN	[CLS]
R	REVERSE FIELD	[RVS]
▁	OFF REVERSE FIELD	[OFF]

Table 1. The changes to the cursor control characters in the lister output.

```
                 QUOTES $00
                 NEXTLN $01
                 PTR    $0F
                 TXTST  $28
                 HIMEM  $34
                 LENFN  $D1
                 LOGFL  $D2
                 SECAD  $D3
                 DEVICE $D4
                 BEGIN  $7530
                 PRTLN  $DCD9
                 OPEN   $F524
                 SETOUT $F7BC
                 RESTOR $F272
                 CLOSE  $F2AC
                 STOP   $F301
                 SPACE  $FDCD
                 CRLF   $FDD0
                 PRINT  $FFD2
                 BASIC  $C092

7530 A5 30              LDA <BEGIN   ;SET TOP OF
7532 85 34              STA HIMEM    ;MEMORY TO
7534 A5 75              LDA >BEGIN   ;PROTECT
7536 85 35              STA HIMEM+1  ;PROGRAM
7538 A9 00              LDA #0
753A 85 D1              STA LENFN
753C 85 D3              STA SECAD
753E A9 04              LDA #4
7540 85 D2              STA LOGFL
7542 85 D4              STA DEVICE
7544 20 24 F5           JSR OPEN     ;OPEN FILE TO
7547 A6 D2              LDX LOGFL    ;PRINTER
7549 20 BC F7           JSR SETOUT
754C A5 28              LDA TXTST
754E A6 29              LDA TXTST+1
7550 85 0F       NEWLIN STA PTR      ;SET LISTER
7552 86 10              STA PTR+1    ;POINTERS
7554 20 70 75           JSR LAST
7557 85 01              STA NEXTLN
7559 20 6A 75           JSR NEXT
755C 85 02              STA NEXTLN+1
755E D0 15              BNE MOR100
7560 20 D0 FD    FINISH JSR CRLF
7563 20 72 F2           JSR RESTOR
7566 20 AC F2           JSR CLOSE
7569 60                 RTS
756A E6 0F       NEXT   INC PTR
756C D0 02              BNE LAST
756E E6 10              INC PTR+1
7570 A0 00       LAST   LDY #0
7572 B1 0F              LDA (PTR),Y
7574 60                 RTS
7575 20 01 F3    MOR100 JSR STOP     ;TEST FOR
7578 F0 E6              BEQ FINISH   ;STOP KEY
757A A9 00              LDA #0
757C 85 00              STA QUOTES
757E 20 6A 75           JSR NEXT
7581 AA                 TAX
7582 20 6A 75           JSR NEXT
7585 EA                 NOP
7586 EA                 NOP
7587 20 D9 DC           JSR PRTLN
758A 20 CD FD           JSR SPACE
758D A9 00       EXTRA  LDA #0       ;PUT IN SPACE
758F 20 D2 FF    PRT    JSR PRINT
7592 20 6A 75           JSR NEXT
7595 D0 0A              BNE MOR200
7597 20 D0 FD           JSR CRLF
759A A5 01              LDA NEXTLN
759C A6 02              LDX NEXTLN+1
759E 4C 50 75           JMP NEWLIN
75A1 C9 22       MOR200 CMP #'"'
75A3 D0 08              BNE NOTQUT
75A5 A5 00              LDA QUOTES
75A7 49 01              EOR #1
75A9 85 00              STA QUOTES
75AB A9 22              LDA #'"'
75AD A6 00       NOTQUT LDX QUOTES
75AF D0 27              BNE INQUOT
75B1 AA                 TAX
75B2 10 DB              BPL PRT
75B4 C9 FF              CMP #$FF     ;IS IT PI?
75B6 F0 D7              BEQ PRT
75B8 29 7F              AND #$7F
75BA A0 FF              LDY #$FF
75BC AA                 TAX
75BD F0 09              BEQ FOUND    ;END OF BASIC
75BF C8          CYCLE  INY
75C0 B9 92 C0           LDA BASIC,Y
75C3 10 FA              BPL CYCLE
75C5 CA                 DEX
75C6 D0 F7              BNE CYCLE
75C8 C8          FOUND  INY
75C9 B9 92 C0           LDA BASIC,Y
75CC 08                 PHP
75CD 29 7F              AND #$7F
75CF 20 D2 FF           JSR PRINT
75D2 28                 PLP
75D3 10 F3              BPL FOUND
75D5 4C 8D 75           JMP EXTRA
75D8 C9 22       INQUOT CMP #'"'
75DA F0 B3              BEQ PRT
75DC C9 80              CMP #$80
75DE B0 04              BCS CHECK    ;SHIFTED?
75E0 C9 20              CMP #$20
75E2 B0 AB              BCS PRT
75E4 A9 5B       CHECK  LDA #'['     ;PRINT [
75E6 20 D2 FF           JSR PRINT
75E9 20 70 75           JSR LAST
75EC A0 07              LDY #7
75EE D9 13 76    SEARCH CMP KEYCHR,Y
75F1 F0 0D              BEQ YES
75F3 88                 DEY
75F4 10 F8              BPL SEARCH
75F6 29 7F              AND #$7F
75F8 20 D2 FF           JSR PRINT
75FB A9 5D       SQCLOS LDA #']'     ;PRINT ]
75FD 4C 8F 75           JMP PRT
7600 BE 1B 76    YES    LDX OFFSET,Y
7603 E8          MOR400 INX
7604 BD 23 76           LDA NEWCHR,X
7607 08                 PHP
7608 29 7F              AND #$7F
760A 20 D2 FF           JSR PRINT
760D 28                 PLP
760E 10 F3              BPL MOR400
7610 4C FB 75           JMP SQCLOS
7613 11          KEYCHR .BYT $11,$1D,$91,$9D
7614 1D                              ;TABLE OF
7615 91                              ;CONTROL
7616 9D                              ;CHARACTERS
7617 93                 .BYT $93,$13,$12,$92
7618 13
7619 12
761A 92
761B FF          OFFSET .BYT $FF,$01,$03,$05
761C 01                              ;OFFSETS TO
761D 03                              ;'NEWCHR'
761E 05                              ;TABLE
761F 07                 .BYT $07,$0A,$0E,$11
7620 0A
7621 0E
7622 11
7623 43          NEWCHR .BYT 'C',$C4
7624 C4                              ;CURSOR DOWN
7625 43                 .BYT 'C',$D2
7626 D2                              ;CURSOR RIGHT
7627 43                 .BYT 'C',$D5
7628 D5                              ;CURSOR UP
7629 43                 .BYT 'C',$CC
762A CC                              ;CURSOR LEFT
762B 43 4C              .BYT 'CL',$D3
762D D3                              ;CLEAR SCREEN
762E 48 4F 4D           .BYT 'HOM',$C5
7631 C5                              ;HOME CURSOR
7632 52 56              .BYT 'RV',$D3
7634 D3                              ;REVERSE
7635 4F 46              .BYT 'OF',$C6
7637 C6                              ;REVERSE OFF
7638 00                 .BYT $00,$00,$00
7639 00                              ;END OF TABLE
763A 00
763B                    .END
```



```
14400 IFS=5THEN14470
14410 IFS=6THEN14450
14420 PRINT"■";R1$;"■■■■■■THE BLACK TOWER"
14430 PRINTR1$;"■■■   OF ZAEXON"
14440 PRINTR1$;"■■■■■   FLOOR ";FL-1:GOTO14490
14450 PRINT"■";R1$;"■■■■■■■ VOUNIM'S ■"
14460 PRINTR1$;"■■■■■   LAIR   ■":GOTO14500
14470 PRINT"■";R1$;"■■■■■■■THE TEMPLE OF■"
14480 PRINTR1$;"■■■■■  Y'NAGIOTH  ■"
14490 P(FL+1)=P(FL)+P
14500 IFFL<4ORRND(TI)<0.3THENRETURN
```

An example of a program listing using the PET Lister.

D J Stephenson

GRAPH PLOTTER

A utility program to make all your graphs neat and tidy.

Many programs already exist which take advantage of the high resolution graphics capability of the ITT 2020 and Apple series of microcomputers. Those which produce a graph plot of a mathematical function usually split the x and y axes into n equal parts and display the value which each scale division represents. This procedure invariably results in an ugly string of mixed digits spreading half way across the screen, requiring tedious mental approximations before points on the curve can be evaluated.

Solving The Problem

The problem is overcome in this program by positioning the axis divisions according to a straightforward power of 10 rule. For example, if the x axis limits for a particular equation are entered as −50 and +50, the program will accurately place five division 'pips' in each direction from the zero intercept and produce a text output:

```
X AXIS*10
```

The y axis is processed in a similar manner. If the x axis limits are entered as −30 and +63 (or similar unruly figures) the division 'pips' are still presented in simple powers of ten.

Existing programs appear to use the DEF FN statements for placing the equations into specific line numbers. Thus, to plot the graphs of X squared and X cubed on the same axes, it is necessary to type out:

```
DEF FNF(X)=X^2
DEF FNG(X)=X^3
```

This method was found tedious and it was decided to ditch it in favour of a subroutine. This allows a simpler and less error-prone entry as follows:

```
Y=X^2
Z=X^3
```

There appears to be no appreciable difference in execution time as a result.

A further advantage is that either one or two functions can be plotted at each RUN without the dreaded 'UNDEF'D FUNCTION ERROR' polluting the screen and halting execution. An error-handling subroutine is provided to deal with division-by-zero errors which can occur when attempting to plot curves of the 1/X or TAN(X) forms. Apart from this, the program includes the usual mundane features such as auto scaling and computing the y axis limits. Two equations can be processed simultaneously by finding the highest and lowest y co-ordinates of both equations and then setting y axis limits accordingly.

HOW TO USE THE PROGRAM

The program is written in 'PALSOFT' BASIC for the ITT 2020 but should also run on an Apple II (Applesoft) providing line 102 is amended to:

```
102 W=279:H=159
```

(This is necessary to compensate for the reduced resolution of the Apple.)

The display invites you to enter the equations in line numbers 3000 and/or 4000

PROGRAM STRUCTURE

Statement	Action
Lines 10-60	Instructions for use, enter equations.
Line 102	Sets max HPLOT co-ordinates (depending on machine).
Lines 110-125	Input and order x axis limits.
Lines 130-145	Input options, find plotting increment.
Lines 147-160	Initialise y axis limits (auto mode).
Lines 165-185	Input and order y axis limits (manual mode).
Lines 197-260	Fill arrays with y and z values and process y axis limits (auto mode).
Lines 267-270	Set graphics and text mode.
Lines 277-300	Find origin and draw axes.
Lines 307-360	Find and draw axes division.
Lines 367-370	Label axes.
Lines 377-410	Label axes for offset origin.
Lines 417-420	Plot 1st graph.
Lines 427-430	Plot 2nd graph.
Lines 437-450	Press any key to continue routine.
Lines 510-570	Input option.
Lines 997-1070	Scaling subroutine.
Lines 1997-5000	Equations subroutine.
Lines 6997-7040	Division of axes subroutine.
Lines 7997-8040	Division by zero error-handling subroutine.

according to the following example format:

```
3000 Y=SIN(X)
4000 Z=COS(X)
```

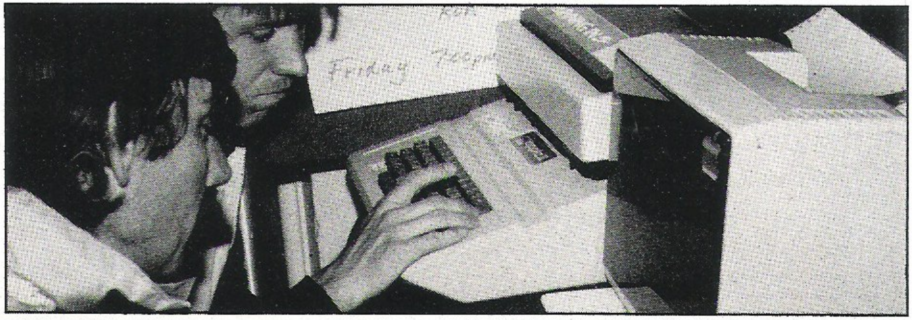
For convenience, constants can be defined in line 104, using any spare variables. You will be asked for x axis limits but it is unimportant in which order these are entered because they are automatically arranged correctly by the program. The number of plotting points (the plotting density) can be chosen within the range 100-300 per graph although it will be appreciated that execution time increases with plotting density.

It may also be obvious that the auto y axis limits feature must be overridden for graphs with inherent discontinuities such as 1/X and TAN(X).

Graphs with offset origins can be plotted in any quadrant since the relevant values X max, X min, Y max and Y min are always displayed.

For those who are interested, the particular equations shown in the listing plots a simple sine wave and superimposes a second curve portraying the fundamental, third and fifth harmonic. This shows how a 'square wave' can be built up by the addition of the odd harmonics according to the Fourier series. The program will run in a 16K machine providing the REM statements are omitted.

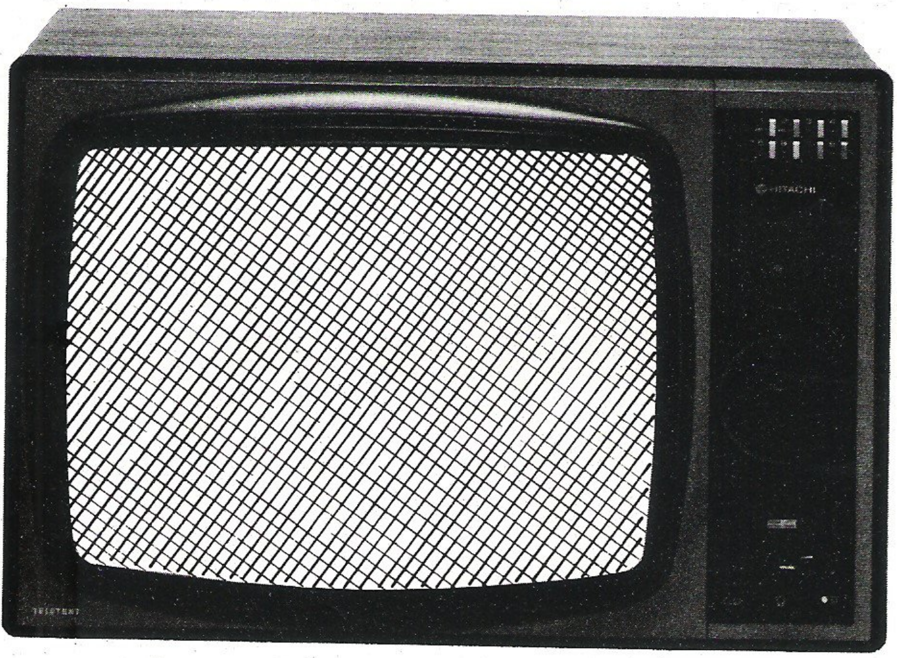
It is also a wise move to set HIMEM to 8192. However, if the REM statements are typed in, the program will overspill into the high resolution graphics page one of memory. If the machine is 32K or over, set LOWMEM to 16384 to avoid this problem.



```
  1Ø  TEXT:HOME:PRINT TAB(14)"GRAPH PLOT":PRINT:PRINT
  2Ø  PRINT "PROVISION OF Y AXIS LIMITS IS MANDATORY":
      PRINT "FOR NON CONTINUOUS GRAPHS ONLY"
  3Ø  PRINT:PRINT "ENTER EQUATIONS IN LINES 3ØØØ AND/OR":
      PRINT "4ØØØ IN THE FORM:-"
  4Ø  PRINT:PRINT "3ØØØ Y=FUNCTION(X)":PRINT "4ØØØ Z=
      FUNCTION(X)":PRINT
  5Ø  PRINT "ENTER EQUATIONS THEN TYPE RUN 1ØØ"
  6Ø  PRINT:END
 1ØØ  DIM Y(3Ø1),Z(3Ø1),A$(1)
 1Ø1  REM ** SET MAX HI-RES PLOT CO-ORDINATES
      IN LINE 1Ø2
 1Ø2  W=359:H=159
 1Ø5  HOME:INPUT "LABEL X AXIS LIMIT (1) ";A:PRINT
 11Ø  INPUT "LABEL X AXIS LIMIT (2) ";B:PRINT
 115  IF A<B THEN XL=A:XR=B:GOTO 13Ø
 12Ø  IF A>B THEN XL=B:XR=A:GOTO 13Ø
 125  GOTO 1Ø5
 13Ø  INPUT "ENTER PLOTTING DENSITY (1-3) ";A:PRINT
 135  IF A>3 OR A<1 THEN 13Ø
 137  REM ** FIND PLOTTING INCREMENT
 14Ø  K=A*1ØØ:INC=(XR-XL)/K
 145  INPUT "Y AXIS LIMITING (Y/N)[8 SPC]";A$:PRINT:
      PRINT
 147  REM ** INITIALISE Y AXIS LIMITS VIA LAST X,Y
 15Ø  IF A$="N" THEN X=XR:GOSUB 2ØØØ:YT=Y:YB=Y:GOTO 19Ø
 155  IF A$="Y" THEN 165
 16Ø  GOTO 145
 165  INPUT "LABEL Y AXIS LIMIT (1) ";A:PRINT
 17Ø  INPUT "LABEL Y AXIS LIMIT (2) ";B
 175  IF A<B THEN YB=A:YT=B:GOTO 19Ø
 18Ø  IF A>B THEN YB=B:YT=A:GOTO 19Ø
 185  GOTO 165
 19Ø  HOME:VTAB 21:HTAB 8:PRINT "TABULATION IS
      PROCEEDING"
 197  REM ** FIND Y AND Z VALUES PLUS Y AXIS LIMITS
 2ØØ  N=Ø:FOR X=XL TO XR STEP INC:N=N+1:GOSUB 2ØØØ:
      Y(N)=Y:Z(N)=Z:IF A$="Y" THEN NEXT:GOTO 26Ø
 21Ø  IF YT<Y THEN YT=Y
 22Ø  IF YB>Y THEN YB=Y
 23Ø  IF YT<Z THEN YT=Z
 24Ø  IF YB>Z THEN YB=Z
 25Ø  NEXT
 26Ø  XX=(XR-XL):YY=(YT-YB)
 267  REM ** SET GRAPHICS PLUS TEXT MODE
 27Ø  HGR:HCOLOR=3:POKE 34,2Ø:CALL -936
 277  REM ** FIND ORIGIN/DRAW AXES
 28Ø  X=Ø:Y=Ø:GOSUB 1ØØØ:Y1=(Y2-5):X1=(X2+5):HPLOT X2,Ø
      TO X2,H:HPLOT Ø,Y2 TO W,Y2
 29Ø  IF Y1<1Ø THEN Y1=(Y2+5)
 3ØØ  IF X1>W-1Ø THEN X1=(X2-5)
 3Ø7  REM ** FIND AND DRAW SCALE AXES DIVISIONS
 31Ø  IF ABS(XL)>=ABS(XR) THEN B=XL:GOSUB 7ØØØ:P=B*1Ø^E:
      Q=XR:R=1Ø^E:GOTO 33Ø
 32Ø  B=XR:GOSUB 7ØØØ:P=B*1Ø^E:Q=XL:R=-1*1Ø^E
 33Ø  FOR X=P TO Q STEP R:GOSUB 1ØØØ:HPLOT X2,Y2 TO
      X2,Y1:NEXT
 34Ø  IF ABS(YT)>=ABS(YB) THEN B=YT:GOSUB 7ØØØ:P=B*1Ø^E:
      Q=YB:S=-1*1Ø^E:GOTO 36Ø
 35Ø  B=YB:GOSUB 7ØØØ:P=B*1Ø^E:Q=YT:S=1Ø^E
 36Ø  X=Ø:FOR Y=P TO Q STEP S:GOSUB 1ØØØ:HPLOT X2,Y2 TO
      X1,Y2:NEXT
 367  REM ** LABELS
 37Ø  CALL -936:PRINT:PRINT "X AXIS *";ABS(R);TAB(21);
      "Y AXIS *";ABS(S)
 377  REM ** LABEL AXES FOR AN OFFSET ORIGIN
 38Ø  IF YB>Ø THEN VTAB 23:PRINT TAB(21);"Y(MIN)= ";YB
 39Ø  IF YT<Ø THEN VTAB 23:PRINT TAB(21);"Y(MAX)= ";YT
 4ØØ  IF XL>Ø THEN VTAB 23:PRINT "X(MIN)= ";XL
 41Ø  IF XR<Ø THEN VTAB 23:PRINT "X(MAX)= ";XR
 417  REM ** PLOT FIRST GRAPH
 42Ø  N=Ø:FOR X=XL TO XR STEP INC:N=N+1:Y=Y(N):
      GOSUB 1ØØØ:HPLOT X2,Y2:NEXT
 427  REM ** PLOT SECOND GRAPH
 43Ø  N=Ø:FOR X=XL TO XR STEP INC:N=N+1:Y=Z(N):
      GOSUB 1ØØØ:HPLOT X2,Y2:NEXT
 437  REM ** ANY KEY TO CONTINUE
 44Ø  X=PEEK(-16384):IF X<127 THEN 44Ø
 45Ø  POKE -16368,Ø:TEXT:HOME
 51Ø  VTAB(1Ø):PRINT "THE FOLLOWING OPTIONS ARE
      AVAILABLE:-":PRINT:PRINT
 52Ø  PRINT "(1) REPLOT (SAME AXES)"
 53Ø  PRINT "(2) REPEAT (DIFFERENT AXES)"
 54Ø  PRINT "(3) ENTER NEW EQUATIONS"
 55Ø  PRINT "(4) END PROGRAM"
 56Ø  PRINT:PRINT:INPUT "ENTER OPTION ";A:IF A>4 OR A<1
      THEN 56Ø
 57Ø  ON A GOTO 27Ø,1Ø5,1Ø,58Ø
 58Ø  HOME:END
 997  REM ** SCALING SUBROUTINE
1ØØØ  X2=INT(W*(X-XL)/XX)
1Ø1Ø  IF Y<YB OR Y>YT THEN Y=Ø
1Ø2Ø  Y2=INT(H*(YT-Y)/YY)
1Ø3Ø  IF Y2<Ø THEN Y2=Ø
1Ø4Ø  IF X2<Ø THEN X2=Ø
1Ø5Ø  IF Y2>H THEN Y2=H
1Ø6Ø  IF X2>W THEN X2=W
1Ø7Ø  RETURN
1997  REM ** EQUATIONS SUBROUTINE
2ØØØ  ON ERR GOTO 8ØØØ
3ØØØ  Y=SIN(X)
4ØØØ  Z=SIN(X)+(1/3*SIN(3*X))+(1/5*SIN(5*X))
5ØØØ  RETURN
6997  REM ** DIVISION OF AXES
7ØØØ  E=Ø:BB=B:B=ABS(B)
7Ø1Ø  IF B>=1Ø THEN B=B/1Ø:E=E+1:GOTO 7Ø1Ø
7Ø2Ø  IF B>=1 AND B<1Ø THEN B=INT(B):IF BB<Ø THEN B=-B:
      GOTO 7Ø4Ø
7Ø3Ø  IF B<1 THEN B=B*1Ø:E=E-1:GOTO 7Ø1Ø
7Ø4Ø  RETURN
7997  REM ** DEAL WITH DIVISION BY ZERO
8ØØØ  A=PEEK(2Ø2):POKE 216,Ø
8Ø1Ø  IF A=133 THEN 8Ø3Ø
8Ø2Ø  RESUME
8Ø3Ø  IF X=XR THEN XR=XR+INC/1Ø:GOTO 15Ø
8Ø4Ø  VTAB 23:PRINT "TRYING TO RECTIFY DIVISION BY ZERO
      ERROR":XL=XL-INC/1Ø:GOTO 2ØØ
```

CROSS HATCHER

Use the DAI's colour graphics to set up your colour TV.



This program may prove of practical use to television engineers like myself. It is written for the DAI Personal Computer and is fairly self-explanatory.

The CHR$(12) in line 10 clears the screen and line 12 generates a 400 Hz test tone from one of the three internal oscillators.

There are four test options; a straight crosshatch (300-400), a large circle (200-240), a colour bar pattern (500-600) and a raster scan in primary colours (700-760).

The graphics commands are only transferrable to a system with equally high resolution colour graphics.

```
  1Ø  PRINT CHR$(12):CURSOR 14,23:PRINT "COLOUR
      TELEVISION TEST PROGRAM"
  12  SOUND Ø 1 15 Ø FREQ(4ØØ.Ø)
  15  PRINT:PRINT
  16  PRINT "CROSS HATCH PATTERN[3 SPC]TYPE A"
  2Ø  PRINT:PRINT
  21  PRINT "PICTURE GEOMETRY[6 SPC]TYPE B"
  3Ø  PRINT:PRINT
  31  PRINT "COLOUR BARS[1Ø SPC]TYPE C"
  4Ø  PRINT:PRINT
  41  PRINT "PURITY[15 SPC]TYPE D"
  5Ø  PRINT:PRINT:PRINT:PRINT
  51  PRINT "TO RETURN FROM ANY ROUTINE PRESS THE SPACE
      BAR"
  6Ø  P=GETC
  61  IF P=65.Ø THEN 3ØØ
  62  IF P=66.Ø THEN 1ØØ
  63  IF P=67.Ø THEN 5ØØ
  64  IF P=68.Ø THEN 7ØØ
  65  GOTO 6Ø
 1ØØ  MODE 6
 2ØØ  FOR X=Ø.Ø TO PI*2.Ø STEP 5E-2
 21Ø  DOT 17Ø+8Ø*COS(X*2.Ø),13Ø+8Ø*SIN(X*2.Ø) 15
 22Ø  NEXT X
 23Ø  DRAW XMAX/2,Ø XMAX/2,YMAX 15
 24Ø  DRAW Ø,YMAX/2 XMAX,YMAX/2 15
 25Ø  GOSUB 2ØØØ
 3ØØ  MODE 5
 31Ø  A=XMAX:B=YMAX:C=2Ø.Ø:D=Ø.Ø
 32Ø  DRAW C,D C,B 15
 33Ø  C=C+2Ø.Ø
 34Ø  IF C>=335.Ø THEN C=2Ø.Ø:GOTO 36Ø
 35Ø  GOTO 32Ø
 36Ø  DRAW D,C A,C 15
 37Ø  C=C+2Ø.Ø
 38Ø  IF C>=255.Ø THEN 4ØØ
 39Ø  GOTO 36Ø
 4ØØ  GOSUB 2ØØØ
 5ØØ  MODE 5
 51Ø  FILL Ø,Ø 4Ø,YMAX 15
 52Ø  FILL 4Ø,Ø 85,YMAX 12
 53Ø  FILL 85,Ø 125,YMAX 14
 54Ø  FILL 125,Ø 17Ø,YMAX 5
 55Ø  FILL 17Ø,Ø 215,YMAX 2
 56Ø  FILL 215,Ø 26Ø,YMAX 3
 57Ø  FILL 26Ø,Ø 3Ø5,YMAX 1
 58Ø  FILL 3Ø5,Ø XMAX,YMAX Ø
 6ØØ  GOSUB 2ØØØ
 7ØØ  MODE 5:A=1.Ø
 71Ø  FILL Ø,Ø XMAX,YMAX A
 72Ø  P=GETC
 73Ø  IF P=32.Ø THEN 75Ø
 74Ø  GOTO 72Ø
 75Ø  A=A+2.Ø:IF A>5.Ø THEN GOSUB 2ØØØ
 76Ø  GOTO 71Ø
2ØØØ  P=GETC:IF P=Ø.Ø THEN 2ØØØ
2ØØ1  IF P=32.Ø THEN MODE Ø:GOTO 1Ø
2ØØ2  GOTO 2ØØØ
```

DOUBLE DENSITY

Double your PET's plotting capacity with this routine.

The following simple program listing allows plotting of characters on an 80 by 50 grid on the PET screen, thus enabling more precise graphs and pictures to be drawn. The first two lines of the program (lines 1 and 2) should be included at the beginning of the program which is to use the double-density feature; they initialise the two arrays required. The plotting section (the latter two lines) can be called by a GOSUB 1000 during the program run, after an x and y value has been specified. The x value should be between −39 and 39, and the y value between −24 and 24.

Where To Go

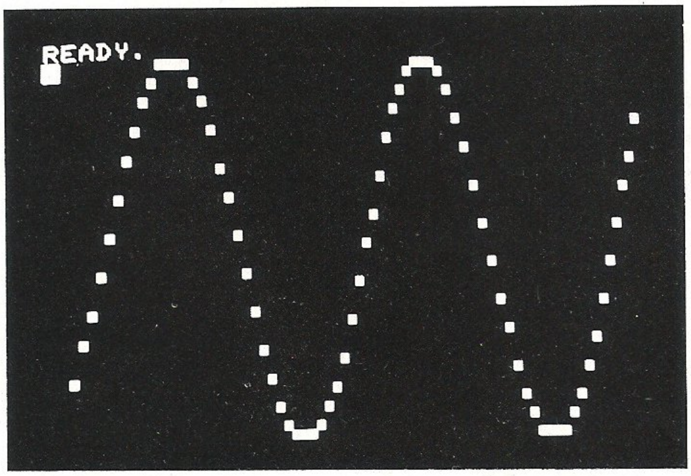
Assigning 0 to both x and y will produce a dot in the centre of the screen, −39 for x and 24 for y will produce a dot in the top left-hand position of the screen, and 39 for x and −24 for y will be in the bottom right-hand corner of the screen. Thus, the positions radiate as for a normal graph from the centre of the screen.

The program works by arranging the codes for the sixteen different double-density graphics in such a way that if the position of the code already on the screen is ORed in binary with the position in the array of the code that you want to put on the screen, the resulting position will give the code containing both the characters that you want to plot.

Array S contains the list of all sixteen codes, and array T is used for decoding the PEEK code from the screen into a position for use with array S. This method is best explained by looking at the array S. Table 1 shows the contents in graphical form. For example, if the character ▖ was on the screen, and you required the character ▚ to be added, the position of the first character, 0001, is ORed with the position of the second character, 0110. The result obtained is 0111 which, in the table, is the character ▙, which is the one required to POKE on to the screen. Line 1010 of the subroutine does this, as well as calculating which character needs to be added to the screen.

POSITION IN ARRAY	BINARY POSITION	DECODED CHARACTER	
0	0000	□	SPACE
1	0001	▖	SHIFTED ;
2	0010	▘	SHIFTED >
3	0011	▌	SHIFTED !
4	0100	▗	SHIFTED ,
5	0101	▄	SHIFTED "
6	0110	▚	SHIFTED ?
7	0111	▙	RVS SHIFTED <
8	1000	▝	SHIFTED <
9	1001	▞	RVS SHIFTED ?
10	1010	▀	RVS SHIFTED "
11	1011	▛	RVS SHIFTED ,
12	1100	▐	RVS SHIFTED !
13	1101	▟	RVS SHIFTED >
14	1110	▜	RVS SHIFTED ;
15	1111	█	RVS SPACE .

Right: The block graphics characters and their binary and character key designations for producing the double density effect.

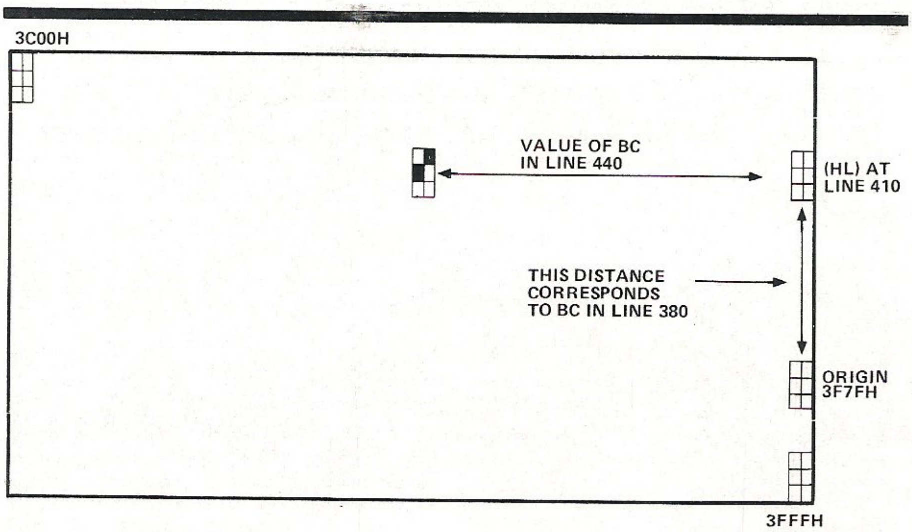

READY.

```
   1  DIM S(15),T(255):FOR T=0 TO 15:READ S(T):
      T(S(T))=T:NEXT T:T=0
   2  DATA 32,123,126,97,108,98,127,252,124,255,226,
      236,225,254,251,160
1000  S=33267+(X/2)-INT(Y/2)*40
1010  POKE S,S(T(PEEK(S)) OR (2^((X/2-INT(X/2))*4+
      ((Y/2-INT(Y/2))*2)^2))):RETURN
```

A Lacy

FAST PLOTTER

Let your TRS-80 take the strain for plotting all those complex functions with this superbly documented program.



Above: This diagram shows how the byte position is calculated in the plotter routine (line numbers refer to the assembly listing). In this example, graphics characters B and C are shown turned on — this means that bits 1, 2 and 7 are logic '1' in that location. The byte will look like that shown above.

Right: This represents a Hex value of 86 Hex (134 Dec).

1	▨	0	0	0	1	1	0
7							0

In the dim and distant past I remember gazing at microcomputer advertisements showing (apparently) all manner of graphs and mathematical symbols flowing across the screen. When I finally got my hands on a machine I soon found out the sad truth. The TRS-80 certainly has graphics capability in the form of SET and RESET functions, but ever so slow!

POKE and PEEK also give access to the display but the speed is not much better. The fastest method of all in BASIC is to PRINT a string containing graphics characters. This last method is very succesful when small areas of the display are to be moved, but I still wanted to see those sine waves rippling across the screen!

The method shown here is a machine code program which sometimes needs to be slowed down to give a viewable display. I shall first describe the machine code program itself, then show you how to interface such a program to a BASIC language program.

One For The Code

This is for your information only, don't worry, you don't have to type in any assembly code to use the graph plotter. All of the references to line numbers in this section are for the assembly code listing. Lines 10-120 are the equivalent of REM statements in BASIC; I include these in my 'library' of source programs because I find assembly code very 'opaque', that is, the program itself does not suggest how it works. This is also the reason for all the comments down the right-hand side of the listing.

The CALL on line 170 is used to get information from the BASIC program. After this call has been made, the HL register pair contains a value corresponding to the value V in the BASIC statement:

```
10 X=USR(V)
```

Lines 200-260 are mainly concerned with setting up loop parameters, the equivalent of the FOR. . .NEXT statement. As in any program, the input variables need to be tested and the appropriate action taken if they are out of the desired range. This is done in lines 230-240; if the variable is greater than 40 then the loop contents will be skipped and the next variable will be processed. I chose a value of 40 because the screen is 48 graphics characters high and space might be needed for axis and other information. The values in the program will give one free line at the top and three at the bottom. Similarly 'XAXIS' defines the display width as numbers of graphics characters. The maximum is 128, and I chose 120 giving some free space at the screen edges.

If the check in line 240 is not made then values could be input which cause memory locations other than screen memory to be loaded, possibly in the areas of RAM used by the TRS-80's housekeeping routines. Most likely you would have to reset the machine to get any more sense out of it!

At this point you need to know how TRS-80 graphics are accessed from machine language. In the TRS-80, there are two video graphics chips, one contains all the information required for the ASCII character set (and more if you know how to get it out) and the other is really a bit of TTL which switches on graphics blocks at the right instant of time in the screen scan. If bit 7 in the screen memory location being accessed is set, at logic '1', then the graphics generator will turn on, otherwise the ASCII generator will be enabled. So we know that we must turn on bit 7 at the required location.

But what is that location? Well, a bit of arithmetic is needed to calculate it and this calculation is what comprises the bulk of the program. Each graphics block corresponds to a byte of memory and is three graphics characters high and two wide. The characters themselves correspond to bits in the memory byte as shown in Table 1. We must determine the bit to be set as well as the correct location, the procedure used is listed:

1) Divide the variable by three. 260-290
2) Save the remainder. 300
3) Multiply quotient by 64. 350-360
4) Subtract it from baseline. 410
5) Get the horizontal position. 420
6) If odd then add 1 to remainder. 450-460
7) Subtract position from origin. 500
8) Convert remainder to a bit position. 520-580
9) Is it already a graphics location? 590
10) If not then set bit 7 610
11) And reset bit 5. 620
12) Put the information on the screen. 640
13) Check to see if finished. 690
14) Get the next variable 200
15) And carry on!

Most of the other operations in the program are concerned with setting up registers prior to the above or with loop counting. In the TRS-80, if a machine code routine has been called from BASIC then a RET instruction will return control to the next BASIC statement.

The information for the graph plot is stored in an integer array as a set of values between 0 and 40. This is rather wasteful of space since each element of the array is contained in two bytes and only the least significant byte is being used. It does make life easier though when filling such an array in BASIC.

The code shown is relocatable, that is, it doesn't mind where it is loaded in memory. This is achieved by avoiding references to absolute addresses within the program, in other words, any jumps or branches are specified as forwards or backwards relative to the current position in the program.

HOW TO USE THE PROGRAM

Type in the BASIC listing and RUN it! This will give you an idea of the speed of plotting as each frame seems to appear instantly. Now try various functions on line 220.

PROGRAM STRUCTURE

Statement	Function	Action
Line 40	Arrays	Contains GG% (N), an array that stores the machine code subroutine and DD%(n,m), the 'target' array. The program treats the latter as a list of m arrays, each of single directions, and displays them in quick succession giving the impression of movement.
Lines 50-100	DATA	The DATA values represent the subroutine.
Line 110	READ	All DATA statements start with a 255 and end with a series of 0s. This avoids having to be too precise about the number of READs. The first number in line 50 then, is a dummy number — take it out if you are not going to use line 110.
Lines 140-160	Array Input	A way of getting the right bytes in place in the integer array. If you are POKEing the subroutine then you don't need this.
Line 190	Message	Prints a message to let you know that all is going according to plan.
Lines 200-230	Delay	Causes a delay while the array, DD%, fills with the 1200 values.
Line 260	Subroutine Directory	Tells the computer where to go to start the machine code subroutine. The statement USR passes the location of the start of the array ***not*** the plotting routine, so that it knows where to get the relevant values.
Line 300	Moving Display	Loops back to give a continuously moving display.

▶

Remember, you have two independant variables to play with, I2 and I1. Line 260 can appear anywhere in your own program as many times as you wish, so there is plenty of scope for experiment.

For example, a program could be written to alter a few of the target array elements while it is running, maybe under keyboard control. This could give a moving display which also changes over a longer time period.

GRAPHIC CHARACTER DISPLAYED	BIT POSITION	EQUIVALENT VALUE (HEX)
A	0	1
B	1	2
C	2	4
D	3	8
E	4	10
F	5	20
NOT USED	6	40
ALWAYS 1 FOR GRAPHICS	7	80

A	B
C	D
E	F

Table 1. This shows the relationship between display memory bytes and the character displayed on the screen.

```
                  FAST PLOTTER
                  THIS PROGRAM IS INTENDED FOR USE AS A
                  USR CALL FROM BASIC. IT WILL RESPOND
                  TO Ø<=A<4Ø, VALUES OUTSIDE THIS RANGE
                  WILL NOT CAUSE A CRASH BUT WILL BE
                  IGNORED. HL MUST POINT TO THE FIRST
                  ELEMENT OF A 12Ø ELEMENT INTEGER
                  ARRAY. Y=Ø CHR POSITION 4
                  Y=119 CHR POSITION 63
                  X=Ø IS ON LINE 13
                  X=39 IS ON LINE 2
BFØØ                       ORG   BFØØ HEX
ØA7F              GETHL    EQU   ØA7F HEX
ØØ78              XAXIS    EQU   12Ø
3F7F              ORIGIN   EQU   3F7F HEX
BFØØ CD 7F ØA     START    CALL  GETHL
BFØ3 Ø6 78                 LD    B,XAXIS
BFØ5 ØE ØØ                 LD    C,Ø
BFØ7 7E           LOOPØ    LD    A,(HL)
BFØ8 E5                    PUSH  HL
BFØ9 C5                    PUSH  BC
BFØA FE 28                 CP    4Ø
BFØC 3Ø 3C                 JR    NC,LOOP4
BFØE Ø6 FF                 LD    B,FF HEX
BF1Ø Ø4           LOOP1    INC   B
BF11 D6 Ø3                 SUB   3
BF13 FE 28                 CP    4Ø
BF15 38 F9                 JR    C,LOOP1
BF17 2F                    CPL
BF18 68                    LD    L,B
BF19 26 ØØ                 LD    H,Ø
BF1B CB 27                 SLA   A
BF1D Ø6 Ø6                 LD    B,6
BF1F 29           LOOP2    ADD   HL,HL
BF2Ø 1Ø FD                 DJNZ  LOOP2
BF22 E5                    PUSH  HL
BF23 C1                    POP   BC
BF24 21 7F 3F              LD    HL,ORIGIN
BF27 B7                    OR    A
BF28 ED 42                 SBC   HL,BC
BF2A C1                    POP   BC
BF2B C5                    PUSH  BC
BF2C CB 38                 SRL   B
BF2E 38 Ø1                 JR    C,LOOP3
BF3Ø 3C                    INC   A
BF31 48           LOOP3    LD    C,B
BF32 Ø6 ØØ                 LD    B,Ø
BF34 B7                    OR    A
BF35 ED 42                 SBC   HL,BC
BF37 47                    LD    B,A
BF38 Ø4                    INC   B
BF39 AF                    XOR   A
BF3A 37                    SCF
BF3B 17           LOOP5    RLA
BF3C 1Ø FD                 DJNZ  LOOP5
BF3E 47                    LD    B,A
BF3F 7E                    LD    A,(HL)
BF4Ø CB 7F                 BIT   7,A
BF42 2Ø Ø4                 JR    NZ,SET
BF44 CB FF                 SET   7,A
BF46 CB AF                 RES   5,A
BF48 BØ           SET      OR    B
BF49 77                    LD    (HL),A
BF4A C1           LOOP4    POP   BC
BF4B E1                    POP   HL
BF4C 23                    INC   HL
BF4D 23                    INC   HL
BF4E 1Ø B7                 DJNZ  LOOPØ
BF5Ø C9                    RET
```

The machine code listing for Fast Plotter.

```
 4Ø  DIM GG%(41),DD%(12Ø,1Ø)
 5Ø  DATA 255,2Ø5,127,1Ø,6,12Ø,14,Ø,126,229,197,254,4Ø,
     48,6Ø,6,255
 6Ø  DATA 4,214,3,254,4Ø,56,249,47,1Ø4,38,Ø,2Ø3,39,6
 7Ø  DATA 6,41,16,253,229,193,33,127,63,183,237,66,193,
     197,2Ø3,56
 8Ø  DATA 56,1,6Ø,72,6,Ø,183,237,66,71,4,175,55,23
 9Ø  DATA 16,253,71,126,2Ø3,127,32,4,2Ø3,255,2Ø3,175,
     176,119,193
1ØØ  DATA 225,35,35,16,183,2Ø1,Ø,Ø,Ø,Ø,Ø
11Ø  READ G9:IF G9<>255 THEN 11Ø
12Ø  FOR X9=Ø TO 41
13Ø  READ Y9:READ Z9
14Ø  X8=256*Z9+Y9
15Ø  IF X8>32768 THEN X8=X8-65536
16Ø  GG%(X9)=X8
17Ø  NEXT X9
18Ø  REM ** END OF DATA READ
19Ø  CLS:PRINT@512,"DATA READ COMPLETE, FILLING ARRAY"
2ØØ  FOR I1=Ø TO 1Ø
21Ø  FOR I2=Ø TO 12Ø
22Ø  DD%(I2,I1)=SIN(I2/2Ø+I1/1.57)*19+2Ø
23Ø  NEXT I2,I1
24Ø  CLS:INPUT "PRESS ENTER FOR DISPLAY";D
25Ø  FOR I2=1 TO 1Ø
26Ø  DEF USR3=VARPTR(GG%(Ø)):X9=USR3(VARPTR(DD%(Ø,I2)))
27Ø  REM ** FOR X=1 TO 5Ø:NEXT X:REM ** IF YOU WANT IT
     SLOWED DOWN
28Ø  CLS
29Ø  NEXT I2
3ØØ  GOTO 25Ø
```

The BASIC program listing.

INFORMATION

Graphic Details

Most personal microcomputers possess memory mapped screens and graphics character sets allowing the user to produce all kinds of graphics displays. However, few machines are equipped with compatible graphics character sets making program conversion from one machine to another quite a difficult task.

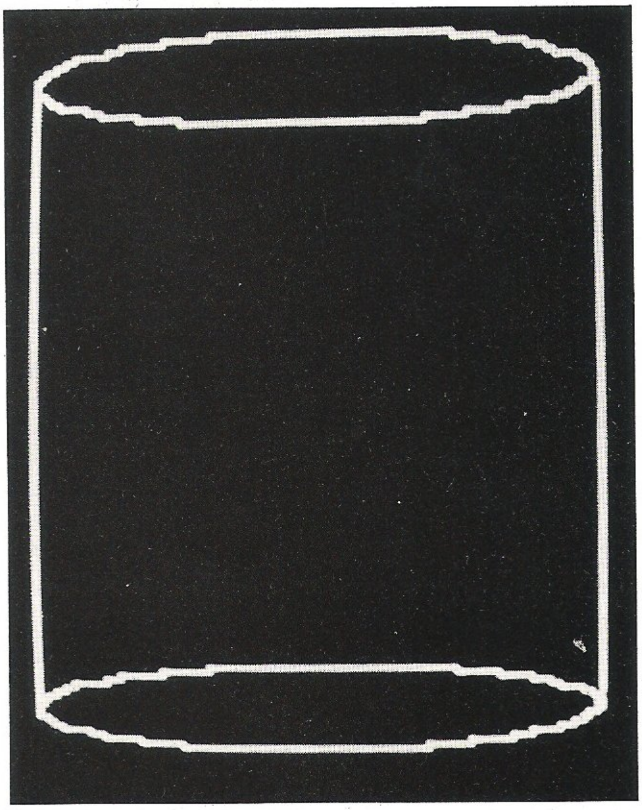
Until now that is. For the first time, we are publishing all the Graphic Details features from past issues of Computing Today as well as a few extra ones we thought you couldn't do without. All you have to do now is to look up the code used in a particular machine's program, cross reference to your machine and select a suitable graphic and its code. Couldn't be simpler, could it?

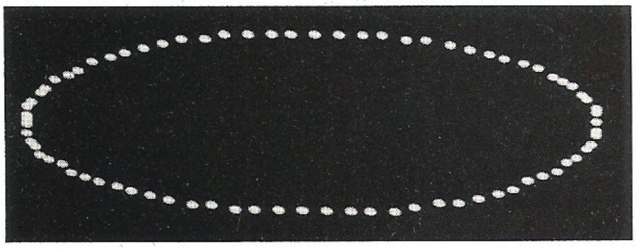
Graphics Directory

This is an entirely new feature detailing the complete graphics capabilities of over 30 micros.

Full of facts on the type of graphics, resolution, memory required, screen format, address range and commands, there is also a section explaining the eccentricities of each machine as well as the extras available.

How have you managed without this information up until now, we ask ourselves!

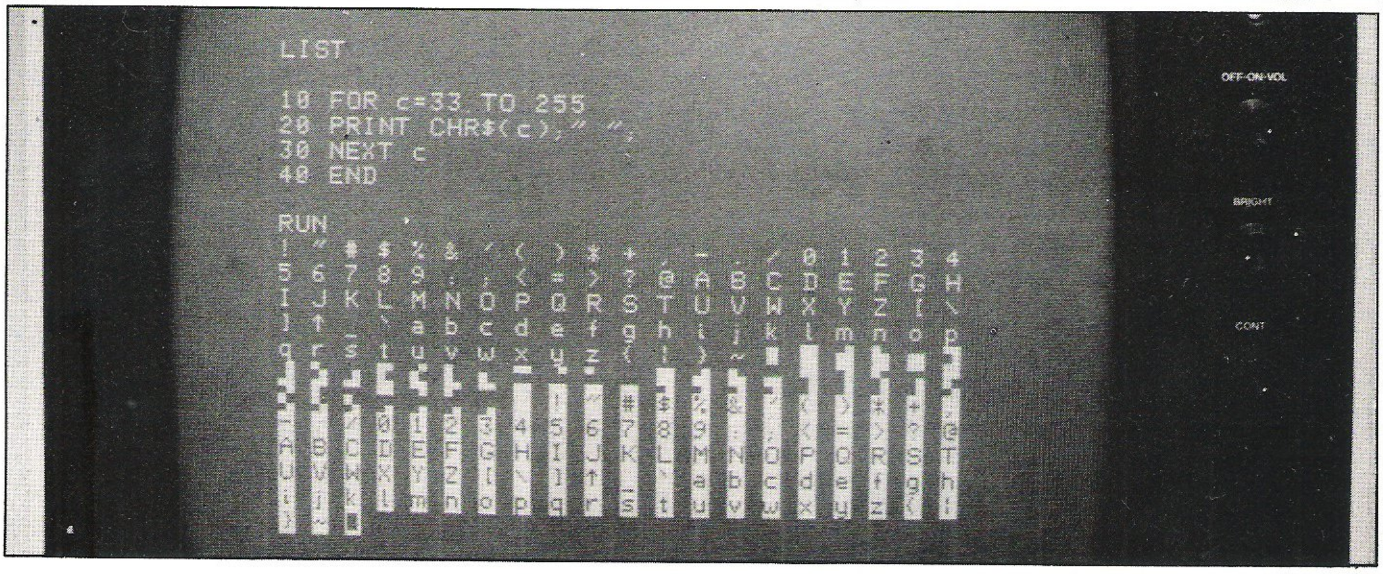




Henry Budgett

GRAPHIC DETAILS

The graphics character sets of the most popular systems — makes program conversion that much simpler.



```
LIST

10 FOR c=33 TO 255
20 PRINT CHR$(c);" ";
30 NEXT c
40 END

RUN
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4
5 6 7 8 9 : ; < = > ? @ A B C D E F G H
I J K L M N O P Q R S T U V W X Y Z [ \
] ↑ _ ` a b c d e f g h i j k l m n o p
q r s t u v w x y z { | } ~
```

Before you get too engrossed in this feature, it might be useful to those newcomers amongst you to introduce yourselves to the standard codes for graphics and other non-printable graphics used in our sister publication, Computing Today.

All standards tend to be irksome to adhere to but the ones laid out here are fairly simple and tend to make software easier to maintain by the programmer and simpler to understand for others.

Controlling That Cursor

The original standards published in CT have most certainly stood the test of time. Machines such as the Commodore VIC which have a dual Shift capability can now be incorporated, as can those systems which use Control key functions.

The recently introduced BBC system offers pre-programmed function keys which, we are glad to say, can also be handled by our original coding system. It's nice to see just how well adapted the original standards have become over the last two years! (Indeed, a whole series of books is using them as its *de facto* standard). The standards for the cursor controls are given in Fig.1. To indicate more than one of these an optional number can be placed within the brackets, for example (4 CL), etc.

The use of square brackets has raised one or two queries. The reason for this choice is that *most* of the common microcomputer BASICs don't use them for specific functions. In fact, at least one machine provides an added bonus by returning a Syntax Error if they are found — a useful check in case you type them in by mistake.

The code [SPC] was added to the list of cursor control codes to get over the problem of indicating just how many spaces are contained in the gap in the printout. The other common variant of the code for spaces is used by the ZX people. Their choice was '_*_' and this crops up in the various newsletters they publish.

The code [RVS] has caused a few headaches. This is really specific to the PET, where the character set can be displayed in reversed video. On machines which don't have this facility,you should either find a character in the set which is the reversed image of the one you want and use that or simply ignore it and use anything else you fancy! Don't forget you may have to look up and alter the values used elsewhere in the program.

The Graphic Solution

It soon became obvious that the techniques applied to the confusing cursor controls could also be applied to the graphics symbols. The following standard is now in general use in programs published in Computing Today.

If a graphics character or characters are to be displayed in a listing (as opposed to POKE codes or CHR$() codes) then they are indicated by the method shown in Fig. 2.

Several people have asked what the relationship between the POKE value for a character and that of its shifted graphic might be. In general, the shifted version of any character will be 64 greater than the value of that character. This applies to both PET and MZ-80K systems in all cases.

This can be taken further to include machines which use a pixel graphics set rather than pre-programmed PET-style characters and the series of codes for these is given in Fig. 3. As is nearly always the case,there is one machine to which the standard shown in Fig. 3 does not apply — Tangerine's Microtan/Micron. This machine uses a four by two cell structure for its pixel graphics instead of the Prestel/Teletext three by two cell. The method for calculating the value to assign to 'P' is shown in Fig. 4, and is fortunately nice and simple.

Making REMarks

Many people scorn the use of REMs within programs but, during the development at least, they are extremely useful. One of the documentation methods that we use is to keep our back-up copy of our programs on a 300 Baud CUTS tape with all the REMs in place; the working co‸y, be it on tape or disc, is REMless in order to save space.

It is also good programming 'manners' to give your REMs odd line numbers:

```
3999 REM ** CRASH PROOF INPUT
4000 INPUT "THE NUMBER OF
ENTRIES ";A$
```

A remarkable number of submitted programs have jumps that go not to the relevant point in the program, but to the REM statement. This can cause severe problems when re-numbering after removing the REMs.

Documentitis

Ideally, program documentation (both for high level and machine code) should consist of a general description of the program's function, a detailed explanation of any parts that are specific to your computer or that do clever things, and suggestions as to how the program might be implemented on another machine.

As essential ingredient of any documentation is a list of any special characters, for example POKE codes, and variables used within the program and the functions they perform. The inclusion of a flowchart within your documentation can also prove an invaluable asset when you come to look at your program at a later date.

[CLS]	CLear Screen
[HOM]	HOMe cursor
[CL]	Cursor Left
[CR]	Cursor Right
[CU]	Cursor Up
[CD]	Cursor Down
[REV]	REVerse video on
[OFF]	Turn it OFF
[SPC]	SPaCe
[CTL]	ConTroL key
[fn]	Function key (BBC)
[G<]	Graphic left (VIC/MZ-80A)
[G>]	Graphic right (VIC/MZ-80A)

Fig. 1. The extended set of cursor control standards includes four new functions.

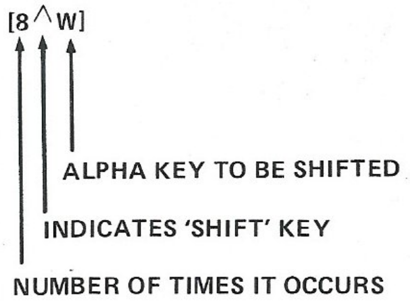


Fig. 2. The way block graphics are indicated on machines like the PET and Sharp; the VIC and Sharp MZ-80A systems of Shift Left and Shift Right are shown in Fig. 1.

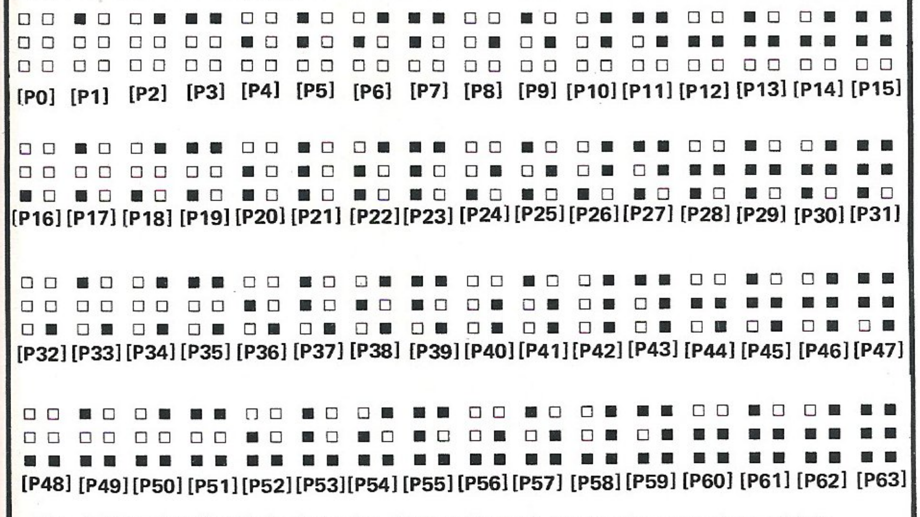

[P0] [P1] [P2] [P3] [P4] [P5] [P6] [P7] [P8] [P9] [P10] [P11] [P12] [P13] [P14] [P15]
[P16] [P17] [P18] [P19] [P20] [P21] [P22] [P23] [P24] [P25] [P26] [P27] [P28] [P29] [P30] [P31]
[P32] [P33] [P34] [P35] [P36] [P37] [P38] [P39] [P40] [P41] [P42] [P43] [P44] [P45] [P46] [P47]
[P48] [P49] [P50] [P51] [P52] [P53] [P54] [P55] [P56] [P57] [P58] [P59] [P60] [P61] [P62] [P63]

Fig. 3. The standard pixel codes; they will work on most computers which employ this technique as well as for Teletext and Prestel.

1	2
4	8
16	32
64	128

Fig. 4. To convert a Tangerine pixel code into its blocks, simply decode the number into its binary or Hex value and fill in the relevant squares.

ASCII

Many currently available personal microcomputers are equipped with memory mapped screens and graphics character sets. These facilities allow the user to produce pictorial and graphic displays (the resolution generally being somewhat crude) and play all those interesting games. But what if you want to translate a program written for another machine which uses another graphics set and has a different screen memory area?

Now, if you had a series of charts showing all the standard codes and screen positions, you could look up a character on the appropriate chart, cross-reference to your machine and select the correct graphics character and its code. Here we give a selection of graphics sets belonging to some of the more popular machines along with a variety of useful notes.

The ASCII Set

The standard character code set for computers is known as ASCII, the acronym for American Standard Code for Information Interchange.

It is based around a seven bit natural binary sequence thus providing a total of 127 different alphanumeric and control codes. Although $2^8 = 128$, we usually regard 'all zeroes' and 'all ones' as NULL codes hence the figure of 127 unique codes. In many systems an eight bit code is used with the extra bit functioning as a parity check.

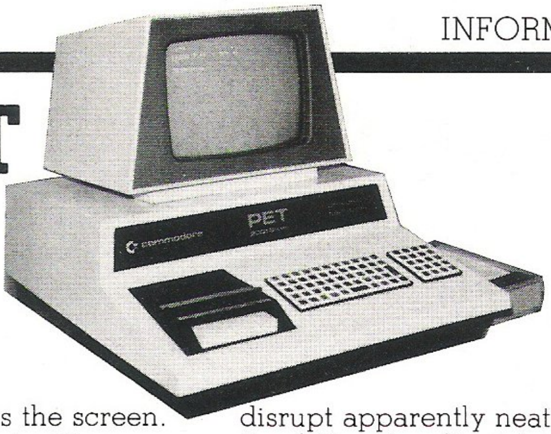
The first table gives the complete ASCII character set. The ASCII codes from 1 to 32 have special control functions. The ones of most use to the general programmer are as follows: 7-Bell, 10-Line Feed, 12-Form Feed (can be used as a Clear Screen), 13-Carriage Return, 32-Space. On some machines code 35 will be a # (hash) symbol.

Character Codes

All the alphagraphic code sets are similar in a number of ways to the ASCII set in that their alphanumeric codes follow the same sort of pattern, for example, code E being a number four greater than code A. In general, the first 31 codes are used for graphics as are the extra 127 codes not used by the ASCII set. It should be noted at this point that these numbers are ***not*** replacements for the ASCII code but numbers to be used in conjunction with the BASIC PEEK and POKE commands which access a referenced location in memory. If you wish to use the ASCII set then the BASIC function CHR($) should be used, for example, PRINT CHR$ (12) clears the screen by using the appropriate ASCII control code, whereas POKEing code 12 would output the respective graphic character.

CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL
0	NUL	32	SP	64	@	96	
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	EXT	35	£	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	¦
29	GS	61	=	93	]	125	}
30	RS	62	>	94	↑	126	~
31	US	63	?	95	←	127	DEL

Commodore PET



Screen memory:- 32768-33767
8000 – 83E7
Hex

Format: 25 lines of 40 characters

Notes: Graphics characters may be converted to lower case alphabetics with POKE 59468,14 and back with POKE 59468,12. CHR$ (147) clears the screen. Note that when outputting screen based information, the PET uses an absolute TAB rather than spaces which can disrupt apparently neat formats. For full and well explained details on the PET see the 'PET Revealed' from Computabits, price £10.

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0	@	32		64	─	96	SP	128	@	160	SP	192	─	224	SP
1	A	33	!	65	♠	97	▌	129	A	161	!	193	♠	225	▌
2	B	34	"	66	│	98	▄	130	B	162	"	194	│	226	▄
3	C	35	#	67	─	99	▔	131	C	163	#	195	─	227	▔
4	D	36	$	68	─	100	▁	132	D	164	$	196	─	228	▁
5	E	37	%	69	─	101	▏	133	E	165	%	197	─	229	▏
6	F	38	&	70	─	102	▒	134	F	166	&	198	─	230	▒
7	G	39	'	71	│	103	▕	135	G	167	'	199	│	231	▕
8	H	40	(	72	│	104	▒	136	H	168	(	200	│	232	▒
9	I	41	)	73	╮	105	◤	137	I	169	)	201	╮	233	◤
10	J	42	*	74	╰	106	▕	138	J	170	*	202	╰	234	▕
11	K	43	+	75	╯	107	├	139	K	171	+	203	╯	235	├
12	L	44	,	76	⎣	108	▗	140	L	172	,	204	⎣	236	▗
13	M	45	—	77	╲	109	└	141	M	173	—	205	╲	237	└
14	N	46	.	78	╱	110	┐	142	N	174	.	206	╱	238	┐
15	O	47	/	79	⎡	111	▁	143	O	175	/	207	⎡	239	▁
16	P	48	0	80	⎤	112	┌	144	P	176	0	208	⎤	240	┌
17	Q	49	1	81	●	113	┴	145	Q	177	1	209	●	241	┴
18	R	50	2	82	─	114	┬	146	R	178	2	210	─	242	┬
19	S	51	3	83	♥	115	┤	147	S	179	3	211	♥	243	┤
20	T	52	4	84	▏	116	▎	148	T	180	4	212	▏	244	▎
21	U	53	5	85	╭	117	▍	149	U	181	5	213	╭	245	▍
22	V	54	6	86	╳	118	▐	150	V	182	6	214	╳	246	▐
23	W	55	7	87	○	119	▀	151	W	183	7	215	○	247	▀
24	X	56	8	88	♣	120	▀	152	X	184	8	216	♣	248	▀
25	Y	57	9	89	▕	121	▆	153	Y	185	9	217	▕	249	▆
26	Z	58	:	90	♦	122	▕	154	Z	186	:	218	♦	250	▕
27	[	59	;	91	┼	123	▖	155	[	187	;	219	┼	251	▖
28	\	60	<	92	▒	124	▝	156	\	188	<	220	▒	252	▝
29	]	61	=	93	│	125	┘	157	]	189	=	221	│	253	┘
30	↑	62	>	94	π	126	▘	158	↑	190	>	222	π	254	▘
31	←	63	?	95	◥	127	▚	159	←	191	?	223	◥	255	▚

Apple II

Screen memory: As shown opposite.

Format—Text: 24 lines of 40 characters (characters of 5 by 7 dots)
Low-Res: 48 rows of 40 characters
High-Res: 192 by 280 dots (each dot is the size of a character dot from the text mode)

Notes: The Apple also has the facility to mix text and graphics modes giving four 40 character text lines below the graphics area. The locations given here act as toggle switches so POKE 49232,0 would set GRAPHICS mode and POKE 49233,0 would reset to TEXT mode. Apple can support colour operation with the addition of a colour card but it should be noted that it is not possible to display two High-Res dots of different colours next to one another. Both block and line graphics commands are supported by Applesoft BASIC and because of the two display areas, it is possible to switch rapidly between two separate pictures producing a simple animated display.

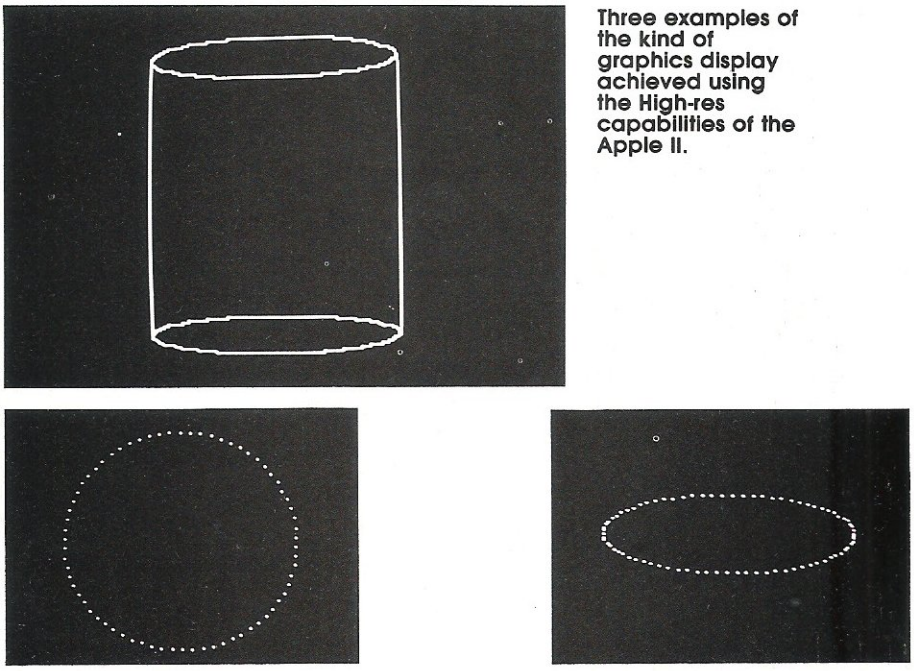
CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL
0	@	32	SP	64	AS COLUMN 1 BUT FLASHING	96	AS COLUMN 2 BUT FLASHING	128	@	160	SP	192		224	
1	A	33	!	65		97		129	A	161	!	193		225	
2	B	34	"	66		98		130	B	162	"	194		226	
3	C	35	#	67		99		131	C	163	#	195		227	
4	D	36	$	68		100		132	D	164	$	196		228	
5	E	37	%	69		101		133	E	165	%	197		229	
6	F	38	&	70		102		134	F	166	&	198		230	
7	G	39	'	71		103		135	G	167	'	199		231	
8	H	40	(	72		104		136	H	168	(	200		232	
9	I	41	)	73		105		137	I	169	)	201		233	
10	J	42	*	74		106		138	J	170	*	202		234	
11	K	43	+	75		107		139	K	171	+	203		235	
12	L	44	,	76		108		140	L	172	,	204		236	
13	M	45	-	77		109		141	M	173	-	205		237	
14	N	46	.	78		110		142	N	174	.	206		238	
15	O	47	/	79		111		143	O	175	/	207		239	
16	P	48	0	80		112		144	P	176	0	208		240	
17	Q	49	1	81		113		145	Q	177	1	209		241	
18	R	50	2	82		114		146	R	178	2	210		242	
19	S	51	3	83		115		147	S	179	3	211		243	
20	T	52	4	84		116		148	T	180	4	212		244	
21	U	53	5	85		117		149	U	181	5	213		245	
22	V	54	6	86		118		150	V	182	6	214		246	
23	W	55	7	87		119		151	W	183	7	215		247	
24	X	56	8	88		120		152	X	184	8	216		248	
25	Y	57	9	89		121		153	Y	185	9	217		249	
26	Z	58	:	90		122		154	Z	186	:	218		250	
27	[	59	;	91		123		155	[	187	;	219		251	
28	\	60	<	92		124		156	\	188	<	220		252	
29	]	61	=	93		125		157	]	189	=	221		253	
30	^	62	>	94		126		158	^	190	>	222		254	
31	_	63	?	95		127		159	_	191	?	223		255	

Mode	**Page 1**	**Page 2**
Text	1024-2047 (0400-07FF)	2048-3071 (0800-0BFF)
Low-Res	As Text	As Text
High-Res	8192-16383 (2000-3FFF)	16384-24575 (4000-5FFF)

The three 'pure' modes and their corresponding screen addressing.

Dec	**Hex**	**Colour**
0	0	Black
1	1	Magenta
2	2	Dark Blue
3	3	Purple
4	4	Dark Green
5	5	Grey 1
6	6	Medium Blue
7	7	Light Blue
8	8	Brown
9	9	Orange
10	A	Grey 2
11	B	Pink
12	C	Light Green
13	D	Yellow
14	E	Aquamarine
15	F	White

The available colours and their codes for Low-res graphics.



Three examples of the kind of graphics display achieved using the High-res capabilities of the Apple II.

Sharp MZ–80K

Screen memory: 53248-54247
D000–D3E7
Hex

Format: 25 lines of 40 characters

Notes: Taking the top left-hand corner of the screen as coordinate 0,0 the commands SET and RESET can be used to turn on or off any cell on a 50 by 80 grid thus allowing limited double density plotting. Normal graphic codes are accessed by POKE; CHR$(198) performs a [CLS].

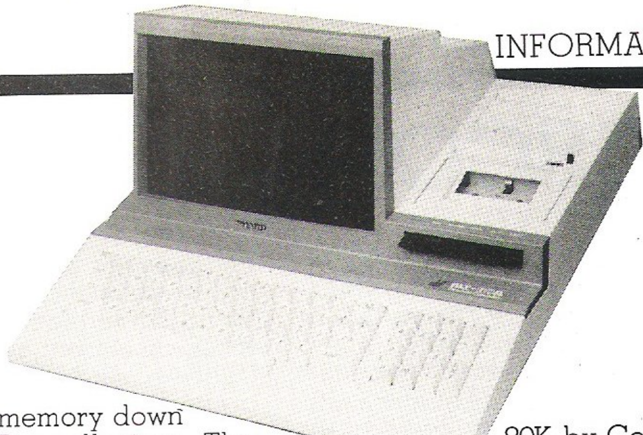
```
STOCK CONTROL OPTIONS

1...............STOCK REGISTRATION
2...............STOCK AMENDMENT
3...............STOCK ENQUIRY
4...............STOCK VALUATION
5...............OUT OF STOCK LIST
6...........7...OUTSTANDING ORDERS
7...............SUPPLIERS FILE
8...............EXIT PROGRAM
```

▶

Sharp MZ-80K (cont)

CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL
0	SP	32	0	64	SP	96	π	128	SP	160	◺	192	⤓	224	┘
1	A	33	1	65	♠	97	!	129	a	161	☰	193	↓	225	┐
2	B	34	2	66	◥	98	"	130	b	162	Ⅲ	194	↑	226	┘
3	C	35	3	67	■	99	#	131	c	163	▦	195	→	227	∼
4	D	36	4	68	♦	100	$	132	d	164	▨	196	←	228	⊀
5	E	37	5	69	←	101	%	133	e	165	▧	197	H	229	⊓
6	F	38	6	70	♣	102	&	134	f	166	▩	198	C	230	Σ
7	G	39	7	71	●	103	'	135	g	167	◿	199	◈	231	⊳
8	H	40	8	72	○	104	(	136	h	168	◸	200	H	232	⋊
9	I	41	9	73	?	105	)	137	i	169	◹	201	I	233	K
10	J	42	—	74	◘	106	+	138	j	170	β	202	大	234	K
11	K	43	=	75	╭	107	*	139	k	171	ü	203	⊀	235	≻
12	L	44	;	76	╮	108	▞	140	l	172	ö	204	⊁	236	⫞
13	M	45	/	77	◣	109	X	141	m	173	Ü	205	¥	237	÷
14	N	46	▪	78	◢	110	╰	142	n	174	Ä	206	●	238	≶
15	O	47	,	79	⁚	111	╯	143	o	175	Ö	207	☺	239	▩
16	P	48	▔	80	↑	112	□	144	p	176	◸	208	▩	240	SP
17	Q	49	▏	81	<	113	□	145	q	177	◸	209	▩	241	▘
18	R	50	▁	82	[	114	□	146	r	178	◹	210	▩	242	▝
19	S	51	▕	83	♥	115	□	147	s	179	◹	211	▩	243	▀
20	T	52	═	84	]	116	═	148	t	180	◺	212	▩	244	▖
21	U	53	║	85	@	117	║	149	u	181	◺	213	▩	245	▌
22	V	54	▀	86	◤	118	╱	150	v	182	◿	214	▩	246	▞
23	W	55	▌	87	>	119	╲	151	w	183	◿	215	▩	247	▛
24	X	56	─	88	↓	120	─	152	x	184	◹	216	▩	248	▗
25	Y	57	│	89	\	121	│	153	y	185	◟	217	╲	249	▚
26	Z	58	▄	90	→	122	▀	154	z	186	◞	218	╲	250	▐
27	£	59	▐	91	▚	123	▌	155	ä	187	╱	219	○	251	▜
28	┗	60	□	92	▗	124	▁	156	╱	188	¥	220	⁙	252	▄
29	┛	61	□	93	▖	125	▕	157	╭	189	⊞	221	╱	253	▙
30	╞	62	▂	94	╡	126	▄	158	◹	190	◡	222	╱	254	▟
31	╧	63	▕	95	╤	127	▐	159	╲	191	◎	223	ω	255	█

Sharp MZ-80A



Screen memory: 53248-55296
D000 – DFFF
Hex

Format: 25 lines of 40 characters

Notes: The screen area is a 2K block memory which scrolls over the 1K window of displayed information; Control E scrolls the memory down and Control D scrolls it up. The entire screen area can be reversed by Control @. It is also possible to simulate the screen configuration of the MZ-80K by Control [(this provides a static 1K screen from 53248) and you can revert back to the 'A' format with Control].

CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL	CODE	SYMBOL
0	SP	32	0	64	│	96	π	128	▕	160	◺	192	⤓	224	┼
1	A	33	1	65	♠	97	!	129	a	161	≡	193	⬇	225	┼
2	B	34	2	66	◥	98	"	130	b	162	‖‖	194	⬆	226	┴
3	C	35	3	67	■	99	#	131	c	163	▦	195	➡	227	∿
4	D	36	4	68	♦	100	$	132	d	164	`	196	⬅	228	ҳ
5	E	37	5	69	←	101	%	133	e	165	~	197	H	229	▧
6	F	38	6	70	♣	102	&	134	f	166	▩	198	C	230	≻
7	G	39	7	71	●	103	'	135	g	167	◿	199	✵	231	⋊
8	H	40	8	72	○	104	(	136	h	168	◸	200	H	232	⋉
9	I	41	9	73	?	105	)	137	i	169	◹	201	I	233	K
10	J	42	─	74	◘	106	+	138	j	170	ß	202	大	234	K
11	K	43	═	75	╭	107	*	139	k	171	ü	203	⋇	235	≻
12	L	44	;	76	◟	108	▞	140	l	172	ö	204	⋋	236	┤├
13	M	45	/	77	◣	109	☒	141	m	173	Ü	205	¥	237	÷
14	N	46	.	78	◢	110	◞	142	n	174	Ä	206	☻	238	∫
15	O	47	,	79	:	111	◜	143	o	175	Ö	207	☺	239	▩
16	P	48	▔	80	↑	112	▔	144	p	176	◸	208	▩	240	▨
17	Q	49	▏	81	<	113	▏	145	q	177	◜	209	▐	241	▘
18	R	50	└	82	[	114	└	146	r	178	◝	210	▌	242	▝
19	S	51	┘	83	♥	115	┘	147	s	179	◞	211	▀	243	▀
20	T	52	─	84	]	116	─	148	t	180	◟	212	▄	244	▖
21	U	53	│	85	@	117	│	149	u	181	╲	213	▞	245	▌
22	V	54	▔	86	◤	118	╱	150	v	182	◞	214	▚	246	▞
23	W	55	▎	87	>	119	╲	151	w	183	╯	215	▚	247	▛
24	X	56	▁	88	↓	120	─	152	x	184	╲	216	▞	248	▗
25	Y	57	▕	89	\	121	│	153	y	185	╰	217	◹	249	▚
26	Z	58	▄	90	→	122	▀	154	z	186	╯	218	◸	250	▐
27	£	59	▐	91	▚	123	▌	155	ä	187	╱	219	°	251	▜
28	┘	60	▁	92	┌	124	▁	156	╱	188	{	220	⁙	252	▄
29	└	61	▕	93	┐	125	▕	157	╭	189	⊞	221	╱	253	▙
30	├	62	▂	94	┤	126	▬	158	╮	190	^	222	╱	254	▟
31	┴	63	▐	95	┬	127	▐	159	╲	191	─	223	~	255	█

NASCOM

Screen memory: 2048-3071
0800– 0BFF
Hex

Format: 16 lines of 48 characters

Notes: A total of 256 bytes of video RAM are lost in the margins and should not be accessed by the user. These are the initial ten locations (0800-0809 Hex) and the last six (OBFA-OBFF Hex) as well as 15 groups of 16 bytes between each line. The top line of the display is not scrolled and may be used for titles, etc. The top line addresses follow on from those of the bottom line which can cause problems for the unwary. The NASCOM 2 offers an optional on-board graphics set whose codes are from 128 up.

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0	□	32	SP	64	@	96		128	▒	160	SP	192	PIXEL CHARACTERS	224	PIXEL CHARACTERS
1 ·	┌	33	!	65	A	97	a	129	⁙	161	¼	193		225	
2	⊥	34	"	66	B	98	b	130	╲	162	½	194		226	
3	┘	35	#	67	C	99	c	131	╱	163	¾	195		227	
4	⇟	36	$	68	D	100	d	132	◥	164	α	196		228	
5	⊠	37	%	69	E	101	e	133	◤	165	β	197		229	
6	✓	38	&	70	F	102	f	134	◣	166	γ	198		230	
7	♁	39	'	71	G	103	g	135	◢	167	δ	199		231	
8	↰	40	(	72	H	104	h	136	⁄	168	μ	200		232	
9	⇢	41	)	73	I	105	i	137	﹨	169	π	201		233	
10	≡	42	*	74	J	106	j	138	⟍	170	Δ	202		234	
11	↓	43	+	75	K	107	k	139	⟋	171	Σ	203		235	
12	⚓	44	,	76	L	108	l	140	‖	172	≠	204		236	
13	⇠	45	—	77	M	109	m	141	=	173	≈	205		237	
14	⊗	46	.	78	N	110	n	142	◪	174	±	206		238	
15	⊙	47	/	79	O	111	o	143	◩	175	÷	207		239	
16	⊟	48	0	80	P	112	p	144	┌	176	»	208		240	
17	◷	49	1	81	Q	113	q	145	┐	177	«	209		241	
18	◶	50	2	82	R	114	r	146	└	178	√	210		242	
19	◵	51	3	83	S	115	s	147	┘	179	^	211		243	
20	◴	52	4	84	T	116	t	148	│	180	¥	212		244	
21	⤫	53	5	85	U	117	u	149	├	181	웃	213		245	
22	⊓	54	6	86	V	118	v	150	┼	182	웃	214		246	
23	┤	55	7	87	W	119	w	151	┤	183	웃	215		247	
24	⧗	56	8	88	X	120	x	152	─	184	○	216		248	
25	ǂ	57	9	89	Y	121	y	153	┴	185	●	217		249	
26	ʕ	58	:	90	Z	122	z	154	┬	186	□	218		250	
27	⊖	59	;	91	[	123	{	155	ʃ	187	▣	219		251	
28	◰	60	<	92	\	124	¦	156	ʅ	188	♠	220		252	
29	◱	61	=	93	]	125	}	157	–	189	♥	221		253	
30	◲	62	>	94	↑	126	—	158	#	190	♦	222		254	
31	◳	63	?	95	—	127	▨	159	∞	191	♣	223		255	

RML 380Z

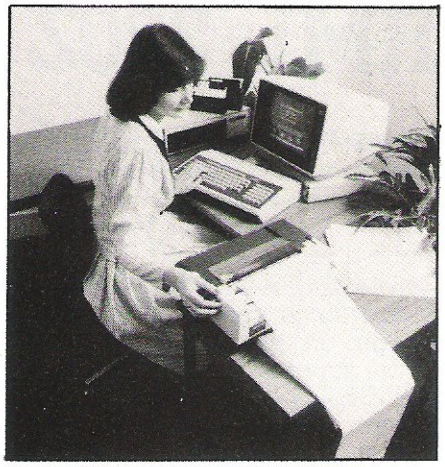
Screen memory: 61440-62209 F000 – F5FF Hex

Format: 24 lines of 40 characters with a 25 character (19 Hex) margin on the right-hand side of the screen. These positions will display but in a non-ordered fashion.

Notes: Apart from the usual PEEK and POKE commands, the RML Extended BASIC offers several graphics commands as follows: GRAPH—sets the top 20 lines to graphics mode, leaving the bottom four for scrolled text; TEXT—resets the screen to full scrolling; PLOT—used for plotting points, characters or strings in the top 20 lines with the bottom left-hand corner being referenced 0,0 and the top right being 79,59. LINE—draws a straight line from the last coordinates to the specified position, and POINT—returns the character value stored at the given location. All the graphics characters can be plotted in two 'shades' of white.

CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL	CODE	SYM-BOL
0		32	SP	64	@	96	—	128	PIXEL CHARACTERS	160	PIXEL CHARACTERS	192		224	
1		33	!	65	A	97	a	129		161		193		225	
2		34	"	66	B	98	b	130		162		194		226	
3		35	£	67	C	99	c	131		163		195		227	
4		36	$	68	D	100	d	132		164		196		228	
5		37	%	69	E	101	e	133		165		197		229	
6		38	&	70	F	102	f	134		166		198		230	
7		39	'	71	G	103	g	135		167		199		231	
8		40	(	72	H	104	h	136		168		200		232	
9		41	)	73	I	105	i	137		169		201		233	
10		42	*	74	J	106	j	138		170		202		234	
11		43	+	75	K	107	k	139		171		203		235	
12		44	,	76	L	108	l	140		172		204		236	
13		45	—	77	M	109	m	141		173		205		237	
14		46	.	78	N	110	n	142		174		206		238	
15		47	/	79	O	111	o	143		175		207		239	
16		48	0	80	P	112	p	144		176		208		240	
17		49	1	81	Q	113	q	145		177		209		241	
18		50	2	82	R	114	r	146		178		210		242	
19		51	3	83	S	115	s	147		179		211		243	
20		52	4	84	T	116	t	148		180		212		244	
21		53	5	85	U	117	u	149		181		213		245	
22		54	6	86	V	118	v	150		182		214		246	
23		55	7	87	W	119	w	151		183		215		247	
24		56	8	88	X	120	x	152		184		216		248	
25		57	9	89	Y	121	y	153		185		217		249	
26		58	:	90	Z	122	z	154		186		218		250	
27		59	;	91	←	123	¼	155		187		219		251	
28		60	<	92	½	124	ǁ	156		188		220		252	
29		61	=	93	→	125	¾	157		189		221		253	
30		62	>	94	↑	126	÷	158		190		222		254	
31		63	?	95	#	127	▪	159		191		223		255	

PC-8001B



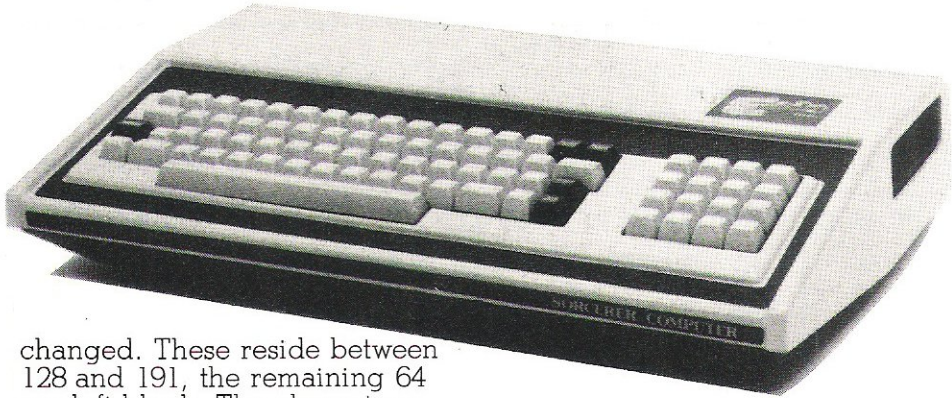
Screen memory: 62208-65207
F300-FEB7
Hex

Format: 20 lines of 40 characters (default)
25 lines of 80 characters

Notes: Although 3K of RAM is provided, only 2K is actually used for the display. The extra 1K is arranged as a further 40 columns on the right-hand side of the screen and stores the display attributes. A dot resolution of 160x80 dots is possible and there are seven colours plus black.

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0		32		64	@	96		128	▁	160		192	α	224	═
1	SH	33	!	65	A	97	a	129	▂	161	≥	193	ʟ	225	╞
2	SX	34	"	66	B	98	b	130	▃	162	·	194	Δ	226	╪
3	EX	35	#	67	C	99	c	131	▄	163	˙	195	β	227	╡
4	ET	36	$	68	D	100	d	132	▅	164	≤	196	ξ	228	◢
5	EQ	37	%	69	E	101	e	133	▆	165	ˊ	197	η	229	◣
6	AK	38	&	70	F	102	f	134	▇	166	˙	198	θ	230	◥
7	BL	39	▾	71	G	103	g	135	█	167	↑	199	¹	231	◤
8	BS	40	(	72	H	104	h	136	▏	168	½	200	±	232	♠
9	HT	41	)	73	I	105	i	137	▎	169	↓	201	ν	233	♥
10	LF	42	*	74	J	106	j	138	▍	170	←	202	π	234	♦
11	HM	43	+	75	K	107	k	139	▌	171	→	203	∧	235	♣
12	CL	44	,	76	L	108	l	140	▋	172	⁺	204	²	236	●
13	CR	45	−	77	M	109	m	141	▊	173	⁽	205	⊕	237	○
14	SO	46	.	78	N	110	n	142	▉	174	⁾	206	¯	238	╱
15	SI	47	/	79	O	111	o	143	┼	175	¼	207	τ	239	╲
16	DE	48	0	80	P	112	p	144	┴	176	∞	208	¢	240	╳
17	D1	49	1	81	Q	113	q	145	┬	177	³	209	ω	241	┄
18	D2	50	2	82	R	114	r	146	┤	178	γ	210	≈	242	┆
19	D3	51	3	83	S	115	s	147	├	179	⁴	211	√	243	┴
20	D4	52	4	84	T	116	t	148	▔	180	⁵	212	⁷	244	┬
21	NK	53	5	85	U	117	u	149	▁	181	⁶	213	⁸	245	┤
22	SN	54	6	86	V	118	v	150	│	182	ε	214	⁹	246	├
23	EB	55	7	87	W	119	w	151	▕	183	ρ	215	ι	247	┼
24	CN	56	8	88	X	120	x	152	┌	184	σ	216	φ	248	
25	EM	57	9	89	Y	121	y	153	┐	185	ψ	217	≠	249	
26	SB	58	:	90	Z	122	z	154	└	186	Ω	218	χ	250	
27	EC	59	;	91	[	123	{	155	┘	187	Γ	219	°	251	
28	→	60	<	92	\	124	¦	156	╭	188	⁰	220	Q	252	
29	←	61	=	93	]	125	}	157	╮	189	ϑ	221	ζ	253	
30	↑	62	>	94	^	126	~	158	╰	190	κ	222	λ	254	
31	↓	63	?	95	_	127		159	╯	191	Σ	223	μ	255	

Sorcerer



Screen memory: 61568-63487
F080 – F7FF
Hex

Format: 30 lines of 64 characters

Notes: The Sorcerer supports 128 fixed characters and 128 programmable characters — the first user defined characters are defined at switch-on but may be changed. These reside between 128 and 191, the remaining 64 are left blank. The characters are defined on an eight by eight grid and can be saved on tape and re-loaded, useful for simulating other machines.

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0	□	32		64	@	96	`	128	│	160	■	192		224	
1	┌	33	!	65	A	97	a	129	│	161	■	193		225	
2	┴	34	"	66	B	98	b	130	│	162	│	194		226	
3	┘	35	#	67	C	99	c	131	│	163	♦	195		227	
4	⤡	36	$	68	D	100	d	132	●	164	♣	196		228	
5	⊠	37	%	69	E	101	e	133	│	165	▚	197		229	
6	✓	38	&	70	F	102	f	134	│	166	▞	198		230	
7	▫	39	'	71	G	103	g	135	│	167	▌	199		231	
8	↖	40	(	72	H	104	h	136	○	168	▐	200		232	
9	→	41	)	73	I	105	i	137	─	169	▬	201		233	
10	≡	42	*	74	J	106	j	138	─	170	▬	202		234	
11	Ψ	43	+	75	K	107	k	139	─	171	╱	203		235	
12	✱	44	,	76	L	108	l	140	─	172	╲	204		236	
13	←	45	–	77	M	109	m	141	π	173	┼	205		237	
14	⊠	46	.	78	N	110	n	142	✕	174	─	206		238	
15	⊡	47	/	79	O	111	o	143	╱	175	─	207		239	
16	⊟	48	0	80	P	112	p	144	╲	176	▔	208		240	
17	◳	49	1	81	Q	113	q	145	┌	177	▒	209		241	
18	◲	50	2	82	R	114	r	146	┐	178	┴	210		242	
19	◱	51	3	83	S	115	s	147	◤	179	▬	211		243	
20	◰	52	4	84	T	116	t	148	◥	180	├	212		244	
21	⤤	53	5	85	U	117	u	149	▪	181	▒	213		245	
22	⎍	54	6	86	V	118	v	150	▪	182	▌	214		246	
23	┬	55	7	87	W	119	w	151	─	183	▐	215		247	
24	⌧	56	8	88	X	120	x	152	♠	184	▒	216		248	
25	╪	57	9	89	Y	121	y	153	♥	185	┤	217		249	
26	⁖	58	:	90	Z	122	z	154	╲	186	▬	218		250	
27	⊟	59	;	91	[	123	{	155	╱	187	┬	219		251	
28	◨	60	<	92	\	124	¦	156	└	188	┌	220		252	
29	◧	61	=	93	]	125	}	157	┘	189	┐	221		253	
30	◩	62	>	94	^	126	~	158	◣	190	└	222		254	
31	◪	63	?	95	_	127	▒	159	◢	191	┘	223		255	

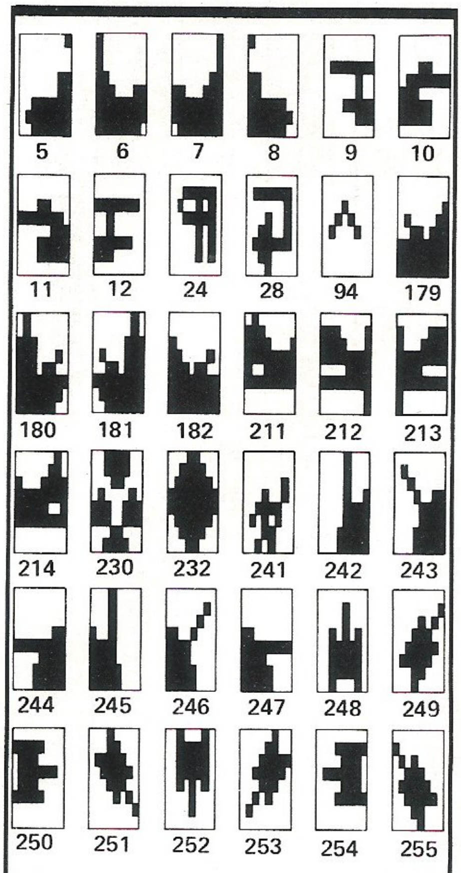
CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0	▶	32		64	@	96	▲
1	▤	33	!	65	A	97	♣
2	▥	34	"	66	B	98	♦
3	◀	35	#	67	C	99	♥
4	▒	36	$	68	D	100	♠
5	⋱	37	%	69	E	101	♧
6	⋰	38	&	70	F	102	♢
7	□	39	'	71	G	103	♡
8	└	40	(	72	H	104	♤
9	┘	41	)	73	I	105	▏
10	‖	42	*	74	J	106	▕
11	≡	43	+	75	K	107	▁
12	⧖	44	,	76	L	108	▔
13	■	45	–	77	M	109	┌
14	◇	46	.	78	N	110	┐
15	○	47	/	79	O	111	├
16	●	48	0	80	P	112	┤
17	◙	49	1	81	Q	113	╱
18	▫	50	2	82	R	114	╲
19	◘	51	3	83	S	115	─
20	◤	52	4	84	T	116	│
21	◢	53	5	85	U	117	▬
22	◥	54	6	86	V	118	▮
23	◣	55	7	87	W	119	╋
24	▀	56	8	88	X	120	✚
25	▄	57	9	89	Y	121	⧖
26	⧗	58	:	90	Z	122	█
27	◪	59	;	91	[	123	↑
28	◩	60	<	92	\	124	↓
29	◣	61	=	93	]	125	←
30	◢	62	>	94	↑	126	→
31	▼	63	?	95		127	▒

TRITON

Screen memory: 4096-5119
1000–13FF
Hex

Format: 16 lines of 64 characters

Notes: Direct access is available to the VDU control chip with the VDU 0,n command in BASIC where n is one of a number of control codes. Some useful ones are: 8—Backspace, 9—Cursor Right, 10—Line Feed, 11—Cursor Up, 12—Clear Screen, 13—Carriage Return erasing remainder of line, 27—Scrolling Line Feed, 28—Home Cursor and 29—non-destructive Carriage Return. Normal screen access is by the VDU x,y format where x is the position and y is the selected character. On some early versions of the TRITON you must have a delay after clearing the screen; a 150 FOR...NEXT loop normally suffices.



The following 36 characters are from the Superboard II and should be inserted in the table opposite when using that system.

UK 101/Superboard II

UK101
Screen memory: 53248-54271
(visible screen 53259-54269)
D000–D3FF
Hex
(visible screen D00B-D3FD Hex)

Format: 16 lines of 64 characters (visible screen — 16 lines of 51 characters)

Superboard II
Screen memory: 53379-54171
D083-D39B
Hex

Format: 25 lines of 25 characters

Notes: Among the more popular single board machines equipped with graphics are the Superboard II and its UK competitor, the UK101. The two systems are basically very similar but they do have differences both in their graphic sets and the layout of the screen memory.

The 'non-printing' functions for the various monitors.

FUNCTION	MON 01	MON 02	CEGMON
Carriage Return	13	13	13
Cursor Left	—	08	—
Cursor Right	—	09	11
Cursor Up	—	11	—
Cursor Down	10	10	10
Home	—	—	12
Clear Screen	—	12	26
Clear Window	—	—	30

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0	╪	32		64	@	96		128	—	160	█	192	(	224	(
1	╫	33	!	65	A	97	a	129	—	161	█	193	^	225	)
2	)(	34	"	66	B	98	b	130	—	162	▀	194	)	226	O
3	⊃⊂	35	#	67	C	99	c	131	—	163	■	195	/	227	C
4	♛	36	$	68	D	100	d	132	—	164	█	196	/	228	Ͻ
5	▛	37	%	69	E	101	e	133	—	165	▪	197	\	229	♥
6	▯	38	&	70	F	102	f	134	—	166	▪	198	\	230	♣
7	▙	39	'	71	G	103	g	135	—	167	▖	199	\	231	♠
8	▣	40	(	72	H	104	h	136	│	168	▗	200	\	232	♦
9	⍋	41	)	73	I	105	i	137	│	169	▚	201	/	233	▢
10		42	*	74	J	106	j	138	│	170	▞	202	/	234	◀
11	^	43	+	75	K	107	k	139	│	171	⁞	203	└	235	▶
12	≡	44	,	76	L	108	l	140	│	172	⁞	204	┌	236	⇧
13	♠	45	-	77	M	109	m	141	│	173	▪	205	┐	237	⇨
14	⌂	46	.	78	N	110	n	142	│	174	▪	206	┘	238	⇩
15	▓	47	/	79	O	111	o	143	│	175	◤	207	┐	239	⇦
16	↑	48	0	80	P	112	p	144	═	176	◢	208	┘	240	†
17	↗	49	1	81	Q	113	q	145	│	177	◥	209	└	241	α
18	→	50	2	82	R	114	r	146	│	178	◣	210	┌	242	β
19	↘	51	3	83	S	115	s	147	│	179	⇒	211	√	243	ω
20	↓	52	4	84	T	116	t	148	▬	180	⇐	212	∫	244	δ
21	↙	53	5	85	U	117	u	149	│	181	⇕	213	▪	245	ψ
22	←	54	6	86	V	118	v	150	■	182	△	214	⁑	246	Ω
23	↖	55	7	87	W	119	w	151	■	183	▒	215	┴	247	γ
24	£	56	8	88	X	120	x	152	▌	184	▒	216	├	248	π
25	¬	57	9	89	Y	121	y	153	▐	185	▒	217	┬	249	Σ
26	⊕	58	:	90	Z	122	z	154	■	186	▒	218	┤	250	λ
27	▯	59	;	91	[	123	{	155	■	187	▓	219	┼	251	φ
28	◂	60	<	92	\	124	¦	156	▌	188	X	220	╰	252	θ
29	⋀	61	=	93	]	125	}	157	▐	189	/	221	╭	253	ξ
30	Ŷ	62	>	94	↑	126	÷	158	■	190	\	222	)	254	ν
31	⋁	63	?	95	_	127	⁄	159	█	191	V	223	)	255	γ

Tandy TRS-80 Model 1

Screen memory: 15360-16383
3C00–3FFF
Hex

Format: 16 lines of 64 characters, selectable to 32 characters

Notes: Character codes from 0 to 31 are control codes. Notable ones are: 14—Cursor on, 15—Cursor off, 23—32/64 character select, 29—Reset cursor to start of line, 30—Erase to end of line, 31—Erase to end of frame. Pixel graphics are accessed by codes 129 to 191 inclusive and the remaining 64 are used as TAB generators from 0 spaces to 63 spaces for space commission in programs.

CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL	CODE	SYM–BOL
0		32	SP	64	@	96	NON DISPLAYABLE CHARACTERS	128	PIXEL CHARACTERS	160	PIXEL CHARACTERS	192	CHARACTER COMPRESSION CODES	224	CHARACTER COMPRESSION CODES
1		33	!	65	A	97		129		161		193		225	
2		34	"	66	B	98		130		162		194		226	
3		35	#	67	C	99		131		163		195		227	
4		36	$	68	D	100		132		164		196		228	
5		37	%	69	E	101		133		165		197		229	
6		38	&	70	F	102		134		166		198		230	
7		39	'	71	G	103		135		167		199		231	
8	BS	40	(	72	H	104		136		168		200		232	
9		41	)	73	I	105		137		169		201		233	
10	LF	42	*	74	J	106		138		170		202		234	
11	FF	43	+	75	K	107		139		171		203		235	
12	FF	44	,	76	L	108		140		172		204		236	
13	CR	45	—	77	M	109		141		173		205		237	
14	CURON	46	.	78	N	110		142		174		206		238	
15	CUROF	47	/	79	O	111		143		175		207		239	
16		48	0	80	P	112		144		176		208		240	
17		49	1	81	Q	113		145		177		209		241	
18		50	2	82	R	114		146		178		210		242	
19		51	3	83	S	115		147		179		211		243	
20		52	4	84	T	116		148		180		212		244	
21		53	5	85	U	117		149		181		213		245	
22		54	6	86	V	118		150		182		214		246	
23	32/64	55	7	87	W	119		151		183		215		247	
24	[CL]	56	8	88	X	120		152		184		216		248	
25	[CR]	57	9	89	Y	121		153		185		217		249	
26	[CD]	58	:	90	Z	122		154		186		218		250	
27	[CU]	59	;	91	↑	123		155		187		219		251	
28	[HOM]	60	<	92	↓	124		156		188		220		252	
29		61	=	93	←	125		157		189		221		253	
30	ERL	62	>	94	→	126		158		190		222		254	
31	ERF	63	?	95	—	127		159		191		223		255	

Dr G J Marshall

GRAPHICS DIRECTORY

All the facts and figures about most popular micros

This article presents a survey of the graphics facilities available on personal computers. The computers covered range from the cheapest single board systems through the established machines to the most recently available offerings, including the Sinclair Spectrum and NEC's personal computer. The graphics facilities themselves vary from the crudest pixel graphics to highly sophisticated colour systems with commands which, for example, permit a shape to be filled with a specified colour.

The classification of the graphics facilities, devised by the Editor, seems quite straightforward, and is designed to show what kind of facilities are possessed by each machine and how they are provided. However, some machines manage to defy classification. In each case a classification *is* given, but it is a good idea to read the 'Comments' section where an explanation or a qualification of a particular categorisation may appear.

The section 'Type of Graphics' shows if a system provides colour or monochrome displays, and also if the graphics system operates with line, block or pixel graphics. In the latter case, several systems possess more than one of the types and, for the purpose of categorisation, line takes precedence over block, which, in turn, takes precedence over pixel.

Under 'Resolution', the highest available resolution using a standard machine is given. It would seem to be straightforward to give the size and address range of the screen memory, but even here there are exceptions, and, to give an example, the Sinclair ZX81 has a variable size screen memory that does not occupy a fixed position.

The 'Commands' section should give a good idea of the power of a graphics system from the user's point of view. It includes only those commands that are exclusively for graphics, so that PEEK and POKE, for instance, are not included because this is not their exclusive purpose.

Under 'Extras Available', only those items supplied by the manufacturer of the micro in question are considered. Even here, only those items that are ***known*** to be available (as opposed to being promised) are listed. For this reason the number of extras mentioned may be on the conservative side.

This survey is intended as an aid to evaluating the graphics capabilities of the personal computers that are covered in the survey. It should also permit comparisons to be made and, finally, it does reveal the increasing sophistication of the graphics facilities that are now readily available.

Acorn ATOM

Type of graphics:	Line Black and white
Resolution:	256 x 192
Memory required:	6K
Screen format:	Columns 0 to 255, rows 0 to 191, cell (0,0) at bottom left
Address range:	32768 to 38911 (8000 to 97FF Hex)
Commands:	CLEAR, PLOT, MOVE, DRAW

Comments: The ATOM has nine graphics modes. To obtain the highest resolution of 256 x 192, it is necessary to have the full memory complement of 12K of ROM and 12K of RAM installed. The 16 forms of the PLOT command permit moving and drawing to positions specified either as absolute positions on the screen or given relative position; plotting a point at an absolute or a relative position; drawing or plotting in black and white; and erasing lines or points. The MOVE and DRAW commands give the same effect as particular cases of PLOT. Graphic displays cannot contain letters or numbers (unless they are designed by the programmer and plotted on the display). The ATOM also supplies for low resolution graphics the full complement of pixel graphics characters on a 2 x 3 mosaic: 64 in black and white and 64 in black and grey.

Extras available: For colour graphics, Acorn provide a PAL UHF colour encoder. A colour graphics extension ROM provides the extra COLOUR command. The highest resolution for colour graphics is 128 x 192 with four colours available.

Apple II

Type of graphics: Line
Colour

Resolution: 280 x 192

Memory required: 8K

Screen format: Columns 0 to 279, rows 0 to 191, cell (0,0) at top left.

Address range: 16384 to 24575 (4000 to 5FFF Hex)

Commands: For low-resolution graphics: GR, COLOR, PLOT, HLIN, VLIN, SCRN
For high-resolution graphics: HGR, HGR2, HCOLOR, HPLOT
For mobile graphics: SHLOAD, DRAW, XDRAW, ROT, SCALE

Comments: HGR2 selects the highest resolution of 280 x 192 and clears the screen. HGR gives a resolution of 280 x 160 with, below the graphics area, four lines available for text, but using a different memory address range (8192 to 16383). HCOLOR can set one of eight colours for plotting, but two of them are white and two more are black! Further, when individual dots are plotted they do not appear in the specified colour unless the adjacent horizontal dot is also plotted. HPLOT will plot a point (HPLOT 10,10), a line (HPLOT 10,10, TO 20,20) or a sequence of linked line segments (HPLOT 10,10 TO 20,20 TO 20,30). The commands for mobile graphics permit a table giving a binary representation of a shape to be loaded from tape. Once loaded it can be drawn, erased, rotated and scaled. These commands permit the generation of realistic, rapidly moving displays from BASIC.

Extras available: Apple provide a number of utility packages to assist in the use of high-resolution graphics. The Apple Graphics Tablet consists of a stylus, an 11″ square digitising surface and an interface card. It permits the direct input of images to the Apple, giving a resolution of 167 points to the inch. Also Apple Pascal incorporates Turtle graphics which provide the commands PENCOLOR, TURNTO, TURN and MOVE.

Atari 400

Type of graphics: Line
Colour

Resolution: 320 x 192

Memory required: 8K

Screen format: Columns 0 to 319, rows 0 to 191, cell (0,0) at top left

Address range: Memory is scattered and depends on the configuration in use

Commands: GRAPHICS, SETCOLOR, COLOR, POSITION, PLOT, DRAWTO, X10

Comments: As Atari 800. The main differences from the Atari 800 are in the keyboard and the size of the RAM. This only affects the graphics in so far as there is less RAM available to the user.

Extras available: As Atari 800.

Atari 800

Type of graphics: Line
Colour

Resolution: 320 x 192

Memory required: 8K

Screen format: Columns 0 to 319, rows 0 to 191, cell (0,0) at top left

Address range: Memory is scattered and depends on the system configuration

Commands: GRAPHICS, SETCOLOR, COLOR, POSITION, PLOT, DRAWTO, X10

Comments: There are nine graphics modes; some of them give a four line text area below the plotting area. The colour and colour intensity of the background and of the plotting colour can be changed using SETCOLOR. COLOR selects the plotting colour. X10 will fill a shape with a specified colour provided that the shape has previously been properly described for it. The background and plotting colours can each be any of 16 colours and can have any of 16 intensities. The combination of resolution, colour variety and colour intensity range allow remarkable colour displays to be generated; this is exhibited as well as anything by a remarkable aircraft landing simulation program.

Extras available: Atari provide a range of graphics program utilities for many applications including the generation of three-dimensional displays. The Versawriter is a digitiser for the direct input of images to the Atari. Its digitising surface has an active area of 8″ x 12.½″ and gives a resolution exceeding 3/100. Atari Pilot incorporates Turtle graphics and includes the commands TURN and DRAW.

BBC Model A

Type of graphics:	Line Colour
Resolution:	320 x 256
Memory required:	10K
Screen format:	Columns 0 to 319, rows 0 to 255, cell (0,0) at top left
Address range:	22528 to 32767 (5800 to 7FFF Hex)
Commands:	CLG, GCOL, MODE, POINT, PLOT, MOVE, DRAW

Comments: There are 16 colours available and four graphics modes. Model A has insufficient memory to get the other four modes of the Model B. Only two colours can be used in the highest resolution mode, although four are available with a resolution of 160 x 256, and when used in Teletext mode (with pixels on a 2 x 3 mosaic), all 16 colours are available. CLG clears the graphics screen, GCOL is used to select the colours for both foreground and background in plotting, and MODE sets the graphics mode. The colour at any point on the screen can be determined with POINT. PLOT can be used in very flexible fashion for moving and drawing, and MOVE and DRAW are both equivalent to particular cases of PLOT. When using PLOT, coordinates can be given in absolute or relative form, the plotting colour can be specified, continuous or broken lines can be drawn and filled triangles can be plotted. The significance of the triangle is that it can be used to construct drawings of solid objects. Finally, block graphics can be defined by the user on an 8 x 8 dot matrix, so that the displays of other micros can be simulated.

Extras available: Extras? Have you got a decent manual yet?

BBC Model B

Type of graphics:	Line Colour
Resolution:	640 x 256
Memory required:	20K
Screen format:	Columns 0 to 639, rows 0 to 255, cell (0,0) at top left
Address range:	12288 to 32767 (3000 to 7FFF Hex)
Commands:	CLG, GCOL, MODE, POINT, PLOT, MOVE, DRAW

Comments: There are 16 colours available and eight graphics modes. Only two colours can be used in the highest resolution mode, although four are available with a resolution of 320 x 256 and all 16 at the 160 x 256 resolution. CLG clears the graphics screen, GCOL is used to select the colours for both foreground and background in plotting, and MODE sets the graphics mode. The colour at any point on the screen can be determined with POINT. PLOT can be used in very flexible fashion for moving and drawing, and MOVE and DRAW are both equivalent to particular cases of PLOT. When using PLOT, coordinates can be given in absolute or relative form, the plotting colour can be specified, continuous or broken lines can be drawn and filled triangles can be plotted. The significance of the triangle is that it can be used to construct drawings of solid objects. Block graphics can also be defined by the user on an 8 x 8 dot matrix, so that the displays of other micros can be simulated.

Extras available: See this section for the Model A Machine . . .the same applies!

Commodore 2001, 3000 & 4000

Type of graphics:	Block Black & white
Resolution:	80 x 50
Memory required:	1K
Screen format:	25 lines of 40 characters
Address range:	32768 to 33767 (8000 to 83EF Hex)
Commands:	None

Comments: There is a repertoire of 62 pixel graphics characters. Each one is designed on an 8 x 8 dot matrix. Pseudo high-resolution graphics can therefore be obtained to a resolution of 320 x 200 by using them ingeniously because, for example, there are eight different horizontal line, and vertical line, graphics characters. The PET's graphics characters are the original set on which all others have been more or less closely based. However, shapes tend to be typically either rectangular approximations or of the 'stick man' variety. Since no graphics commands are provided, the graphics characters are placed on the screen by using the PRINT command or a POKE to screen memory. In this way, letters and numbers can appear anywhere on graphic displays. For plotting lines or curves, quartered (2 x 2) pixel graphics characters can be used to obtain a resolution of 80 x 50.

Extras available: Graphics packages are available, for example, to give pseudo high-resolution line drawing. 'PET graphics' by Nick Hampshire gives many routines for generating graphics as well as giving a design for a light pen.

Commodore VIC-20

Type of graphics:	Block
	Colour
Resolution:	44 x 46
Memory required:	1K
Screen format:	23 lines of 22 characters
Address range:	7680 to 8185 and 38400 to 38905 (IE00 to 1FF9 Hex and 9600 to 97F9 Hex)
Commands:	None

Comments: The VIC has 62 block graphics characters on an 8 x 8 dot matrix which are identical to those of the PET. Pseudo high-resolution graphics can be obtained to a resolution of 176 x 184 because, as with the PET, there are eight different horizontal line graphics characters. The memory area from 7680 to 8185 is used to hold the codes for the characters displayed on the screen while the area from 38400 to 38905 stores the colour information. Characters can be plotted in any of eight colours, while the background can be in any of 16 and the border around the plotting area of any eight. Thus, a character can be placed on the screen by using two POKE commands — one for the character and one for its colour. Alternatively, this can be done with a PRINT command with the colour specification included in it. The resolution of 44 x 46 is obtained using the quarter (2 x 2) pixel graphics characters.

Extras available: A cartridge is available to give high-resolution graphics to a resolution of 176 x 158. It provides commands that include PAINT, POINT, DRAW, COLOUR, CIRCLE and REGION. Joysticks and paddles are available and a light pen is reputed to be on the way.

Compukit UK 101

Type of graphics:	Block
	Black and white
Resolution:	48 x 16
Memory required:	1K
Screen format:	16 lines of 48 characters
Address range:	53248 to 54271 (D000 to D3FF Hex)
Commands:	None

Comments: A wide range of graphics characters is available on an 8 x 8 dot matrix. The inclusion of eight horizontal line characters and eight vertical line characters make possible pseudo high-resolution graphics to a resolution of 384 x 128. The characters can be placed in the screen using PRINT. The use of POKE is less convenient since the memory is mapped in lines of 64 characters of which up to 48 per line can be displayed. Although the UK101 possesses a number of pixel characters, it has no complete set on any mosaic.

Extras available: None.

DAI

Type of graphics:	Line
	Colour
Resolution:	336 x 256
Memory required:	23K
Screen format:	Columns 0 to 335, rows 0 to 255, cell (0,0) at bottom left
Address range:	25600 to 49151 (6400 to BFFF Hex)
Commands:	MODE, COLORT, COLORG, DOT, DRAW, FILL, SCRN

Comments: Any of 12 graphics modes can be set with MODE; there are three different resolutions and the option to have a text window below the plotting area with each. In some modes 16 different colours can be used, while in others any four of the 16 are available. However, when using all 16 colours there are restrictions on how many of them can be used in each eight columns-wide band. This means that plots have to be carefully planned, but there is an 'ANIMATE' facility which takes advantage of the restriction and works by making colours appear or disappear against the background so that animation effects result. COLORG sets the colours in a four colour mode, COLORT sets up background and plotting colours, DOT plots a point, DRAW draws a line and FILL fills a rectangle when two of its corners are specified.

Extras available: None.

Exidy Sorcerer

Type of graphics:	Block Black and white
Resolution:	128 x 60
Memory required:	2K
Screen format:	30 lines of 64 characters
Address range:	– 3968 to – 2058 (F080 to F7FF Hex)
Commands:	None

Comments: The Sorcerer provides the capability for 128 block graphics characters on an 8 x 8 dot matrix. All 128 characters can be designed by the user, although 64 of them are set at switch-on. The definitions of the graphics characters are held in locations FC00 to FFFF Hex. The default characters include the quarter (2 x 2) pixel graphics to give a resolution of 128 x 60. A judicious choice of graphics characters can give a pseudo high-resolution capability of 512 x 240. The ability to define up to 128 graphics characters means, for example, that other alphabets such as Greek or Arabic can be defined and used, or that all pixel graphics on a 2 x 3 mosaic can be defined, so increasing the resolution.

Extras available: None.

Grundy NewBrain

Type of graphics:	Line Colour
Resolution:	640 x 250
Memory required:	Information not provided
Screen format:	30 lines of 80 characters
Address range:	Information not provided
Commands:	MOVE, MOVEBY, TURN, TURNBY, PLACE, BACKGROUND, COLOUR, WIPE, DRAW, DRAWBY, DOT, CENTRE, FILL, RANGE, AXES, PEN

Comments: The plotting commands are given in the form 'PLOT plotlist', so that a typical command might appear as PLOT MOVEBY(5), TURNBY(90). WIPE clears the screen. The MOVE command, unlike that on any other micro, permits a line to be drawn; DRAW allows a line to be drawn in a specified colour. TURN causes the direction in which the drawing 'pen' faces to turn to a specified direction. MOVEBY, DRAWBY and TURNBY all operate as MOVE, DRAW and TURN, but relative to the current position. PLACE moves the pen to a specified position and DOT marks a single dot in a specified colour. BACKGROUND sets the background colour and COLOUR the plotting colour. With CENTRE, the origin can be relocated, RANGE sets the horizontal and vertical ranges, and two labelled axes can be drawn with AXES. PEN is used to find the position, orientation and other states of the pen. There are many options for configuring the screen to mix areas displaying high-resolution graphics and text. The resolutions can also be set to any of several alternatives. A wide range of characters is provided including a complete set of pixels on a 2 x 3 mosaic and the Greek alphabet.

Extras available: It is not clear that all the facilities implied by the specification are yet available, including in particular, the colour capability.

NASCOM 1

Type of graphics:	Pixel Black and white
Resolution:	96 x 48
Memory required:	1K
Screen format:	16 lines of 48 characters
Address range:	2048 to 3071 (800 to BFF Hex)
Commands:	None

Comments: The NASCOM 1 is programmed in machine code and so has no special commands for graphics. The resolution is obtained by placing pixels with a 2 x 3 mosaic on the screen. The locations in memory to which the screen is mapped are awkwardly arranged in lines of 64 characters of which only 48 are displayed. The addresses start at the second line down, and the addresses for the top line follow those of the bottom line! The top line of the screen is static and does not scroll.

Extras available: None.

NASCOM 2

Type of graphics:	Pixel Black and white
Resolution:	96 x 48
Memory required:	1K
Screen format:	16 lines of 48 characters
Address range:	2048 to 3071 (800 to BFF Hex)
Commands:	CLS, SET, RESET, POINT

Comments: The resolution is obtained by using pixels with a 2 x 3 mosaic. CLS clears the screen, SET plots a point and RESET can erase a point. With POINT, it is possible to determine if a point is plotted at a particular place. When using these commands, the coordinates are a little awkward. The top three rows on the screen are rows 45 to 47. The remaining rows are rows 0 to 44, the columns are columns 0 to 95 and so for this part of the screen the cell (0,0) is at the left of the fourth line down from the top of the screen.

Extras available: None.

NASCOM 3

Type of graphics:	Pixel Black and white
Resolution:	96 x 48
Memory required:	1K
Screen format:	16 lines of 48 characters
Address range:	2048 to 3071 (800 to BFF Hex)
Commands:	CLS, SET, RESET, POINT

Comments: The resolution is obtained using pixels with a 2 x 3 mosaic. CLS clears the screen, SET plots a point and RESET can erase a point. POINT can be used to determine if a point is plotted at a particular place.

Extras available: An Advanced Video Controller card gives high-resolution colour graphics. A resolution of 800 x 256 can be obtained with two colours. A resolution of 400 x 256 is also available with eight foreground and eight background colours.

NEC PC-8001

Type of graphics:	Line Colour
Resolution:	160 x 100
Memory required:	3K
Screen format:	25 lines of 80 characters
Address range:	62208 to 65159 (F300 to FE87 Hex)
Commands:	COLOR, LINE, PSET, PRESET, POINT, WIDTH

Comments: The memory is mapped in lines of 120 characters of which up to 80 can be displayed. The number of characters per line, and the number of lines, can be set using WIDTH. The highest resolution is obtained using a 2 x 4 pixel characters. One of the eight colours can be set using COLOR. The undisplayed part of the screen memory holds the colour information for the display. With PSET a point is plotted and PRESET can erase one; POINT is used to examine any point. The LINE command can be used in several ways: it can cause a line to be drawn with the two points as its corners and, optionally, to fill it with a specified colour. There are also 56 graphics characters on an 8 x 8 dot matrix for use in block graphics.

Extras available: The PC-8023 dot matrix printer can print block and dot graphics.

OSI Superboard II

Type of graphics: Block
Black and white

Resolution: 25 x 25

Memory required: 0.8K

Screen format: 25 lines of 25 characters

Address range: 53379 to 54171 (D083 to D39B Hex)

Commands: None

Comments: A wide range of graphics characters is available on an 8 x 8 dot matrix. The characters include eight horizontal line characters and eight vertical line characters, so that pseudo high-resolution graphics to a resolution of 200 x 200 can be obtained. The characters can be placed on the screen using PRINT or POKE, but the use of POKE is somewhat awkward as the memory is mapped in lines of 32 characters of which only 25 are displayed. Although there is a group of 36 unusual block graphics characters, there is no complete set of pixel characters on any mosaic with which a consistent resolution exceeding 25 x 25 could be achieved.

Extras available: None.

Research Machines 380Z

Type of graphics: Block
Black and white

Resolution: 80 x 72

Memory required: 1.7K

Screen format: 24 lines of 40 characters

Address range: 61440 to 62209 (F000 to F301 Hex)

Commands: None

Comments: 32 graphics characters are provided as well as all the pixel characters with the 2 x 3 mosaic. These pixels give the 80 x 72 resolution. The other graphics characters give little scope for accessing the full dot resolution with pseudo high-resolution plots. RML Extended BASIC provides the commands GRAPH, which gives an 80 x 60 resolution area for graphics with the origin at the bottom left and four-line text window below it; PLOT, to plot a point; LINE, to draw a line and POINT, to determine if a point is plotted at a particular place.

Extras available: A high-resolution graphics board gives a highest resolution of 320 x 192 with four colours. It has its own memory on the board and so makes no call on the system's memory. The commands RESOLUTION, CLEAR, PLOT, LINE, COLOUR, FILL (to fill a rectangle), SETCOL (to set colours for later display — invoked by VIEW) are among those provided. The graphics package GINO-F is also available; this package is a library of subroutines which can be called from a FORTRAN program for the easy generation of two- and three-dimensional drawings.

Sharp MZ-80A

Type of graphics: Block
Black and white

Resolution: 80 x 50

Memory required: 2K

Screen format: 25 lines of 40 characters

Address range: 53248 to 55295 (D000 to D7FF Hex)

Commands: SET, RESET

Comments: Because 2K of memory is reserved for a display which itself requires only 1,000 locations, the screen can be scrolled either up or down. Control + D causes the display to scroll up and Control + E causes it to scroll down. The block graphics symbols, on an 8 x 8 dot matrix, are modelled fairly loosely on those of the PET and are the same as those of the MZ-80K. Pseudo high-resolution graphics to a resolution of 320 x 200 can be obtained because the repertoire of graphics characters includes eight different vertical line, and horizontal line, characters. Cleverly designed graphics characters make it possible to draw lines in directions other than the vertical and horizontal, to draw curves, to plot circuits with symbols which include transistors and capacitors, and, in contrast, to make faces with nose and eye symbols. With SET and RESET, any point can be turned on or off using quartered (2 x 2) pixels.

Extras available: None.

Sharp MZ-80B

Type of graphics:	Line Black and white
Resolution:	320 x 200
Memory required:	8K
Screen format:	Columns 0 to 329, rows 0 to 199, cell (0,0) at top left.
Address range:	57344 to 65535 or 24576 to 32767 (E000 to FFFF Hex or 6000 to 7FFF Hex)
Commands:	GRAPH, SET, RESET, LINE, BLINE, POSITION, PATTERN, POINT, POSH, POSV

Comments: The 8K memory area for the screen graphics is in a special area rather than in the user RAM. When a graphic display is required the graphics RAM is treated like ordinary RAM, but the addresses assigned to it depend on how the main RAM is being used. GRAPH indicates the graphic display mode and automatically selects the graphics RAM in the appropriate way. SET plots a point and RESET can erase it. LINE draws a line or connected line segments while BLINE can erase them. POSITION is equivalent to 'MOVE' and with PATTERN, any pattern of dots can be drawn as specified in the command. The screen can be examined by using POINT, POSH and POSV.

Extras available: A second graphics RAM can be installed and used in the same way as the first one, giving the capability, for example, to create mobile graphics by switching between the two screens. The Pascal available for the MZ-80B includes the graphics commands graph, gset, grset, line, bline, position, pattern, point, posh and posv, all of which correspond to the similarly named BASIC commands.

Sharp MZ-80K

Type of graphics:	Block Black and white
Resolution:	80 x 50
Memory required:	1K
Screen format:	25 lines of 40 characters
Address range:	53248 to 54247 (D000 to D3E7 Hex)
Commands:	SET, RESET

Comments: The block graphics symbols of the MZ-80K are based on an 8 x 8 dot matrix and are modelled fairly loosely on those of the PET. Pseudo high-resolution graphics to a resolution of 320 x 200 can be obtained because the repertoire of graphics characters includes eight different horizontal line, and vertical line, characters. Cleverly designed graphics characters make it possible to draw lines in directions other than the vertical and horizontal, to draw curves, to plot circuits with symbols which include transistors and capacitors, and, in contrast, to make faces with nose and eye symbols. With SET and RESET, any point can be turned on or off using quartered (2 x 2) pixels. Other graphics symbols are placed on the screen by using PRINT or with a POKE to the screen memory.

Extras available: None.

Sharp PC-1500

Type of graphics:	Block Black and white
Resolution:	7 x 156
Memory required:	Input buffer
Screen format:	One line of 26 characters
Address range:	N/A
Commands:	For one-line display: GLS, GPRINT, GCURSOR, POINT For printer/plotter: CSIZE, ROTATE, COLOR, LF, LPRINT, LCURSOR, SORGN, GLCURSOR, LINE, RLINE

Comments: The one-line display possessed by the PC-1500 can display 26 characters, each on a 5 x 7 dot matrix. However, every dot in the display is accessible. CLS clears the display, GCURSOR permits the cursor to be positioned on any column of the display and POINT reveals what is displayed in any column. With GPRINT any character can be created and then plotted on the display. The printer/plotter which can be attached to the PC-1500 can create plots in four colours using the ball-point pens mounted in its pen-holder. When creating plots on this device, COLOR selects the colour to be used, SORGN establishes the origin of coordinates, GLCURSOR positions the pen and LINE causes a line to be plotted. RLINE also causes lines to be drawn, but uses co-ordinates relative to the current pen position rather than absolute ones. For plotting characters, CSIZE establishes their size, LCURSOR positions them and LPRINT causes them to be drawn. They can be drawn in any one of four directions established with ROTATE.

Extras available: The printer/plotter is correctly known as the printer/cassette interface.

Sinclair ZX Spectrum

Type of graphics:	Line Colour
Resolution:	256 x 176
Memory required:	6K
Screen format:	Columns 0 to 255, rows 0 to 175, cell (0,0) at bottom left
Address range:	16348 to 22528 (3FDC to 5800 Hex)
Commands:	PLOT, DRAW, CIRCLE, POINT, INK, PAPER, FLASH, BRIGHT, INVERSE, OVER

Comments: The same screen arrangement is used as with the ZX81: 22 usable rows each with 32 character positions and 8 x 8 dot pixels combine to give the Spectrum's resolution of 256 x 176. The cleverly named INK and PAPER give plotting and background colours respectively; both can be chosen from eight colours. BRIGHT permits two levels of brightness to be obtained and with FLASH, parts of the screen can be made to flash. INVERSE permits foreground and plotting colours to be interchanged, while OVER allows what would be called overprinting when using paper. With PLOT, any point can be plotted and DRAW not only permits lines to be drawn, but also arcs and circles. CIRCLE gives a circle when the centre and radius are specified.

Extras available: First, get your Spectrum!

Sinclair ZX81

Type of graphics:	Pixel Black and white
Resolution:	64 x 44
Memory required:	Up to 0.8K
Screen format:	Columns 0 to 63, rows 0 to 41, cell (0,0) at bottom left
Address range:	Not fixed
Commands:	PLOT, UNPLOT

Comments: The screen representation held in memory can vary in size, since each screen line is terminated by a Newline and spaces at the end of a line need not be included. This is to economise on memory usage in the small RAM area provided with the standard ZX81. The screen memory resides above the area where a program is stored, and that begins at 16509. Thus, the screen memory occupies different locations when different programs are entered, implying that it is not feasible to use POKE commands to generate displays although clever software allows it. PLOT causes a point to be plotted while UNPLOT erases one. There are six graphics characters besides the 16 pixel graphics characters. They are placed on the screen by using PRINT.

Extras available: The ZX printer can reproduce graphics. With a COPY command, the screen contents are reproduced on the printer.

Tandy TRS-80 Model 1

Type of graphics:	Pixel Black and white
Resolution:	128 x 48
Memory required:	1K
Screen format:	16 lines of 64 characters
Address range:	15360 to 16383 (3C00 to 3FFF Hex)
Commands:	CLS, SET, RESET, POINT

Comments: The screen is cleared with CLS, SET causes a point to be plotted and RESET can erase a point. With POINT, it can be determined if a point has been plotted at a particular place. Thus, the facilities provided by the TRS-80 Model 1 for graphics are quite primitive. Nevertheless, the drawing of lines and curves is not at all difficult.

Extras available: None.

Tandy TRS-80 Model 3

Type of graphics:	Pixel Black and white
Resolution:	128 x 48
Memory required:	1K
Screen format:	16 lines of 64 characters
Address range:	15360 to 16383 (3C00 to 3FFF Hex)
Commands:	CLS, SET, RESET, POINT

Comments: The screen is cleared with CLS, SET causes a point to be plotted and RESET can erase a point. With POINT, it is possible to determine if a point has been plotted at a particular place. These quite primitive facilities are identical to those of the TRS-80 Model 1, but additionally the Model 3 possesses 96 characters on an 8 x 8 dot matrix which can be placed on the screen with a POKE to the screen memory or with a PRINT command.

Extras available: None.

Tandy Color Computer

Type of graphics:	Line Colour
Resolution:	256 x 192
Memory required:	6K
Screen format:	Columns 0 to 255, rows 0 to 191, with cell (0,0) at top left
Address range:	Selectable by the user in 1.5K blocks between 1536 and 13823 (600 and 35FF Hex)
Commands:	SCREEN, PMODE, PCLS, PCLEAR, PSET, PRESET, COLOR, LINE, CIRCLE, DRAW, PAINT, PUT, GET

Comments: On the Tandy Color Computer with Extended BASIC, the graphics screen and choice of colours are re-set with SCREEN, and PMODE selects the resolution. The highest resolution mode permits the use of two colours, while other modes permit four; foreground and background colours are selected with COLOR. PCLS clears the screen, PSET plots a point and PRESET can erase a point. With PCLEAR, the section of the graphics memory which is required can be nominated. The LINE, CIRCLE, DRAW and PAINT commands are all very powerful and flexible. LINE allows a line to be drawn between two points or to be erased, but with an additional parameter, a rectangle with two points as corners can be drawn and, if required, filled with a specified colour. With CIRCLE, a circle, ellipse or any part of either can be drawn in any colour. DRAW permits a shape to be drawn according to commands given with it in a stylised fashion. PAINT lets the screen be painted in a given colour from a specified point to a specified boundary. For mobile graphics, GET allows a shape to be transferred from the screen to an array in memory, and with PUT it can subsequently be placed somewhere else on the screen.

Extras available: None.

Tangerine Micron/ Microtan 65

Type of graphics:	Pixel Black and white
Resolution:	64 x 64
Memory required:	0.5K
Screen format:	16 lines of 32 characters
Address range:	512 to 1023 (200 to 3FF Hex)
Commands:	None

Comments: The resolution is obtained by POKEing to the screen memory the codes for the 'chunky' pixel graphics. These are on a 2 x 4 mosaic, which on the 32 character by 16 line screen gives a 64 x 64 resolution. In putting, say, one dot on the screen at this resolution, some bit manipulation is necessary to find the required character and a knowledge of the memory map is also required to place it correctly. If plotting a point is awkward, drawing a line is correspondingly more difficult. The graphics facilities of the Tangerine must be said to be primitive (at least they are no more sophisticated than many other block graphics based systems).

Extras available: There is a graphics board giving a resolution of 256 x 256, carrying 8K of static RAM and permitting text and graphics to be mixed.

Texas Instruments TI-99/ 4A

Type of graphics:	Block Colour
Resolution:	24 x 32
Memory required:	Information not provided
Screen format:	24 lines of 32 characters
Address range:	Information not provided
Commands:	CLEAR, COLOR, GCHAR, HCHAR, VCHAR, SCREEN

Comments: The graphics facilities in TI BASIC are provided by subroutines, so that the form of the program statement for clearing the screen is CALL CLEAR. To create graphic displays, users can define their own block characters on an 8 x 8 dot matrix (32 of them). In this way, judicious character design can give pseudo high-resolution displays to a resolution of 192 x 256. CALL CHAR is used for defining characters. There are 16 colours available, CALL SCREEN sets the colour of the screen, and CALL COLOR is used to set the foreground and background colours of a character to be displayed on the screen. CALL HCHAR permits a character to be displayed at a given location on the screen and then to be repeated horizontally as specified. CALL VCHAR works similarly and permits repetition vertically. With CALL GCHAR, the screen can be interrogated.

Extras available: TI Extended BASIC contains extra commands, including DISPLAY AT and DISPLAY USING. It also permits 'sprites' composed of up to four graphics characters to be created and then moved rapidly around the screen using its full 192 x 256 resolution. TI Logo is also available.

Transam Tuscan

Type of graphics:	Block Black and white
Resolution:	16 x 64
Memory required:	1K x 7 bits
Screen format:	16 lines of 64 characters
Address range:	The screen memory does not intrude on the system memory map
Commands:	None

Comments: The Tuscan has 22 graphics characters each situated on an 8 x 12 dot matrix. The characters themselves, however, cannot freely occupy any dots within the matrix, being subject to certain restrictions, particularly that the top row and bottom four rows are always identical. In this way, these characters are not entirely suitable for (and nor are they really intended for) generating lines and curves. It is possible to customise the 128 characters in the character generator for particular applications.

Extras available: There is a memory mapped VDU system to plug into the S100 bus.

Video Genie 1 and 2

Type of graphics:	Pixel Black and white
Resolution:	128 x 48
Memory required:	1K
Screen format:	Columns 0 to 127, rows 0 to 47, cell (0,0) at top left
Address range:	15360 to 16383 (3C00 to 3FFF Hex)
Commands:	CLS, SET, RESET, POINT

Comments: The screen is cleared with CLS, SET causes a point to be plotted and RESET can erase a point. With POINT, it can be determined if a point has been plotted at a particular place. Thus, the facilities provided for graphics by the Video Genie are quite primitive. Nevertheless, the drawing of lines and curves is not at all difficult.

Extras available: None.

CORRECTIONS

It appears that in our last issue of Personal Software, despite using listings produced by the BBC Micro itself, errors have still crept into several of the programs. We know that all the programs worked at one point because we ran them to take the photographs which appear in the book!

However, there appears to be a somewhat more serious problem. Owners of Model A systems found that they could not load programs which, according to their understanding of the machine, ought to fit into the 16K of memory. The fact is that the BBC Model A only posesses 16K of user RAM when it is switched off. If you consult page 225 of the Provisional User Guide you will see that an area of some 3300 bytes is taken away from you for various system variables and workspace. Your 16K system has now become a 13K system to all intents and purposes. (The new User Guide gives details of this as well and the figures it quotes are even larger — less room for you.) We included the sizes of each program to the nearest 500 bytes on the contents page of each section of the magazine but, sadly, it seems that this was not enough to help many of you sort the problems out.

When using MODE 4 graphics on the Model A, as many of the programs in the book did, the user is left with around 3K to fit the program and variables into — not an easy task. Some readers have found that removing the text and any REMs from the program made it small enough to run but this also brought another problem to light. Several enterprising programmers started to remove spaces from the programs to squash them in and then rang us to ask why they had an even bigger problem than when they started. The answer to this is that the BBC Micro allows long variable names. For example, if you remove the spaces in:

```
IF H AND D=2 OR P AND
S=6 THEN 200
```

you create several new variables:

```
IF HAND=2 OR PANDS=6
THEN 200
```

The following represents all the errors we know of in the last issue:

SURROUND p14

Line 60 should read:

```
MODE 4:CLS:VDU
19,1,0,0,0,0:
VDU 19,0,3,0,0,0
```

FOX & HOUNDS p36

The SOUND commands in lines 870 and 880 are missing their final parameter. This is 15 and must be separated from the previous value by a comma.

There also appears to be a quirk in the algorithm which shows up as the machine refusing to admit that it has been beaten. This can be cured by changing the jump at the end of line 1040 to 690 and the jump at the end of line 1060 to 660. There also appears to be a rather obscure occurence of a fox occupying the same square as a hound. This can be resolved by changing line 740 to read:

```
740 IF A(X+1,Y+C)=0 AND
    X<8 AND Y+C>0 AND
    Y+C<9 THEN E=X+1:
    F=Y+C:GOTO 790
```

THE WHITE BARROWS p40

Line 30 should read:

```
N=42:@%=&00000303
```

The substitution of 1 for the % symbol also occurs in Meter Minder.

LEAPFROG p61

The contents of lines 1020 and 1030 should be exchanged.

Line 1850 should read:

```
PRINT TAB (5,23); "PRESS
ANY KEY';
```

HOME FINANCE p70,71

Two of the arrays DIMensioned in line 50, M1 and S1, should be M% and S%. They also appear in lines 360, 1310, 1320, 1670, 2500, 2510, 2770 and 2790 and the appropriate M% or S% should be substituted.

Due to a layout error lines 2200 to 2360 appear in the wrong place. No lines are missing and the BASIC will accept them if typed in as printed.

In some copies it appears that Line 3070 is somewhat faint. It should read:

```
IF R$="Y" THEN 2930
```

It also appears that there are faults in the original versions of Calendar and Morse Trainer which have been brought across into the BBC versions during translation. Both these programs are being extensively re-written and if it proves practical, we will re-publish them in a later issue.

Personal SOFTWARE

Personal Software is a quarterly publication dedicated to all aspects of software for the most popular micros.

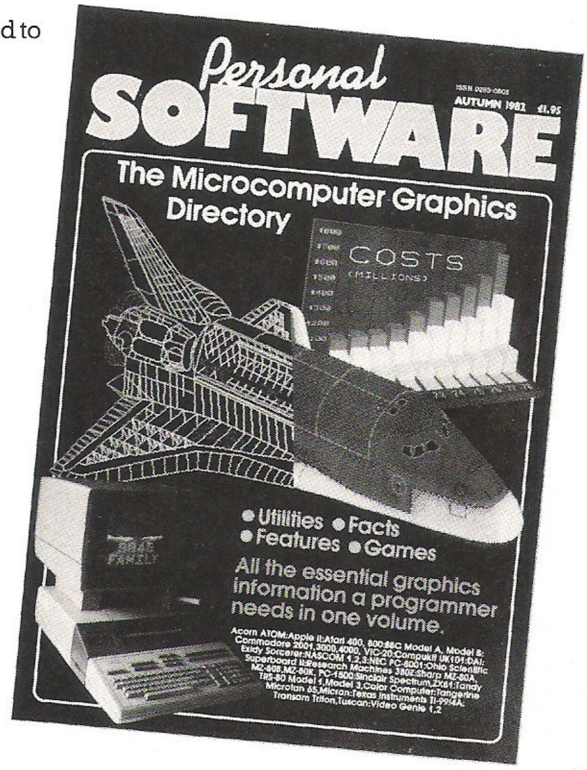
Forthcoming issues will contain everything you ever wanted to know about machine code, utility programs and simple software techniques.

Personal Software can be ordered directly from us at £7.80 per annum or £1.95 per copy. To ensure a single copy or a complete year's supply, fill in the form below, cut it out and send it with your cheque or postal order (made payable to ASP Ltd) to:

Personal Software
Subscriptions,
513 London Road,
Thornton Heath,
Surrey CR4 6AR.

– you can even spread the load with your credit card!

Don't miss out – subscribe now.


Personal SOFTWARE
ISSN 0263-0803
AUTUMN 1982 £1.95
The Microcomputer Graphics Directory
COSTS (MILLIONS)
• Utilities • Facts
• Features • Games
All the essential graphics information a programmer needs in one volume.
Acorn ATOM:Apple II:Atari 400, 800:BBC Model A, Model B:
Commodore 2001,3000,4000, VIC-20:Compukit UK101:DAI:
Exidy Sorcerer:NASCOM 1,2,3:NEC PC-8001:Ohio Scientific
Superboard II:Research Machines 380Z:Sharp MZ-80A,
MZ-80B,MZ-80K, PC-1500:Sinclair Spectrum,ZX81:Tandy
TRS-80 Model 1,Model 3,Color Computer:Tangerine
Microtan 65,Micron:Texas Instruments TI-99/4A:
Transam Triton,Tuscan:Video Genie 1,2

SUBSCRIPTION ORDER FORM

Cut out and SEND TO : Personal SOFTWARE
513 LONDON ROAD,
THORNTON HEATH,
SURREY,
CR4 6AR.

POST FREE SUBSCRIPTION

Please use BLOCK CAPITALS and include post codes.

Name (Mr/Mrs/Miss) . . . .
delete accordingly

Address . . . .

. . . .

. . . .

Signature . . . .

Date . . . .

82 Autumn

Please commence my subscription to Personal Software with the . . . . issue.

SUBSCRIPTION RATES (tick □ as appropriate)

£7.80 for 4 issues UK □

£1.95 for a single copy of the . . . . . issue □

I am enclosing my (delete as necessary) Cheque/Postal Order/International Money Order for £. . . . .
(made payable to ASP Ltd)
OR
*Debit my Access/Barclaycard**
*(*delete as necessary)*

Insert card no. □□□□□□□□□□□□□

BIBLIOGRAPHY

There is undoubtedly a need for some good books on graphics because although the micro manuals usually include one or two programs designed to show the machine's capabilities to good effect, they are usually rather limited in the help that they offer to the user in mastering the full potential of the graphics. The four books under review are among the few available about graphics on micros.

PET Graphics: This publication is clearly specific to the PET, explaining how to generate displays using the PET's block graphics.

The book starts from absolute basics by explaining how displays can be produced using first PRINT and then POKE commands. While programs written in BASIC are presented throughout the book, attention quickly moves to machine code programs.

There are chapters on screen and block scrolling, double density plotting and the displaying and moving of large characters.

My only real quibbles, and they are minor, are that in such a good book the quality of the English and the spelling are in places so poor as to be distracting, and the table showing the block graphics characters is nowhere near the quality of the ones printed elsewhere in this publication.

PET Graphics by Nick Hampshire is published by Computations Ltd at £10.00 for 218 pages.

Computer Graphics Primer: Mitchell Waite's book contains three chapters and two short appendices. The first chapter is a truly dire general introduction. It begins: 'Rod leaned slightly forward, his eyes intently fixed on the screen before him.' Enough said, I should think. I would recommend that nobody waste any time reading this chapter. Do, however, look at the pictures as they include some fine examples, in colour, of what computer graphics can achieve.

The second chapter deals mainly with the hardware techniques used by computer graphics equipment. The third chapter, called 'Graphics programming', deals with programming the Apple II in high resolution graphics mode. It covers general plotting, shape tables, transformations and animation.

Some of the material covers the same ground as Apple's Applesoft manual, but the book does complement and extend the treatment given in the manual. To me, though, this book is a pretty expensive way of obtaining a minor extension of the Apple manual which, as it happens, is rather good.

Computer Graphics Primer by Mitchell Waite is published by Sams at £10.45 for 184 pages. ISBN 0-672-21650-7

Graphics on Microcomputers: This book is another in the NCC's 'Computing in 80's' series. It claims to review 'the current trends in graphics on low-cost microprocessor-based systems' and to provide 'information on a number of commercially available systems'.

Well it does — but it contains very little that could not be found in an 18-month old copy of Computing Today. The book presents the specifications of the PET, Apple and Acorn Atom, among others, but it makes no mention of the BBC Microcomputer, the Atari machines or of Hewlett-Packard's microcomputer-based graphics equipment.

A look at 'picture building' techniques is also promised, and this would have been interesting and valuable. However, what is presented is a copy of two magazine articles which have been re-written sufficiently to avoid violating copyrights.

I found this book very disappointing.

Graphics on Microcomputers by J E Lane is published by NCC at £4.00 for 59 pages. ISBN 0-85012-333-X

A Practical Introduction to Computer Graphics: This is not really aimed at micro users at all. Its programs are written in FORTRAN: the graphics commands are based on the Calcomp library, which is a library of FORTRAN subroutines providing graphics facilities originally intended for use with Calcomp graphic plotters. If this does not sound promising, do not despair. The programs presented in this book can all be readily translated to BASIC, and the graphics routines either have familiar names and purposes or can be readily related to the commands available for graphics on any micro.

Besides providing a practical introduction to graphics (as promised in its title) the book also gives the best introduction to the underlying principles of computer graphics that I have read. It deals with two-dimensional geometry in a painless fashion, followed by two-dimensional transformation.

The book then moves on to deal with three-dimensional objects, showing how to model, transform and generate perspective views of them. This leads on to a treatment of some rather advanced topics including the removal of hidden lines and surfaces in three-dimensional scenes and animatiom. Throughout the book there are many superb examples of computer-generated images.

A Practical Introduction to Computer Graphics by IO Angell is published by MacMillan at £5.50 for 146 pages. ISBN 0-333-31083-7

GAMES PACK

Beat this for value! Five 16K programs Plus two 1K programs

3-D Battle (M/code - 1K)
City Bomb (M/code - 1K)
Warp Wars (Basic & M/code - 16K)
Snake (Basic 16K)
Sweet Tooth (Basic & M/code - 16K)
Slalom (Basic 16K)
Black Holes (Basic 16K)

ALL ON ONE CASSETTE FOR ONLY £4.95 ($9.90)

NEW

GRAPHICS TOOLKIT

(another masterpiece by Paul Holmes)

22 exciting MACHINE CODE routines that give you control over your screen as never before!

(ZX81 - 16K RAM ONLY)

Draw/Undraw draws or deletes your multi-character shape which is defined in a REM statement.

Foreground On/Off use this to 'protect' existing characters on your screen.

Border/Unborder draws a border round the edges of your screen area. Edit lines can be used if required.

Fill fills any number of lines you specify, starting at any line you specify, by your chosen character.

Reverse converts all characters to their inverse video, control as in FILL.

Print Position Controls
UP, DOWN, LEFT, RIGHT. Alter your next PRINT position in the direction indicated.

Editprint moves next PRINT position to first edit line.

Scroll Facilities
UPSCROLL, DOWNSCROLL, RIGHTSCROLL, LEFTSCROLL. Scroll your screen in the direction indicated.

Onscreen/Offscreen turns your screen on or off.

Background On/Off fills your screen by your specified character.

Search and Replace will search the screen for every occurrence of the character you specify and replace it with your new character.

Square draws a square or rectangle from your specified co-ordinates.

All these routines are in machine code for SUPER-FAST response! GRAPHICS TOOLKIT uses only 2K of your RAM and that includes space to load the programmers TOOLKIT described above.

ALL FOR ONLY £5.95 ($11.90)

This includes a cassette with 2 copies of the program, 2 copies of a demonstration program plus a comprehensive instruction booklet with examples.

All prices are fully inclusive.

An ESSENTIAL addition to your 16K RAM ZX81

TOOLKIT

(written by Paul Holmes)

Provides the following additional facilities:-

Line renumber - you state starting number and increment value. GOTO's and GOSUB's included in line renumber.

Search and List - searches for and lists every line containing specified character.

Search and replace - changes every occurance of a character as you require.

Free space - tells you how many free bytes you have left.

SPECIAL GRAPHICS ROUTINES

Hyper graphics mode - graphics never seen on a ZX81 before.

Fill - fills your screen instantly with your specified character.

Reverse - changes each character on your screen to its inverse video.

TAPE ROUTINE - provides a system WAIT condition until a signal is received in the cassette ear jack.

All these routines are written in machine code and together take up only 1K of your precious RAM -an incredible achievement!!

FOR 16K *ONLY £4.95 ($9.90)*

NEW

BATTLESHIPS & CRUISERS GAME

(16K - ZX81)

First computer version of this popular game of strategy and tactics. Both grids on view together. Establish the position of your fleet then locate and destroy the computer's fleet. Visual display shows hits and misses made and gives running score.

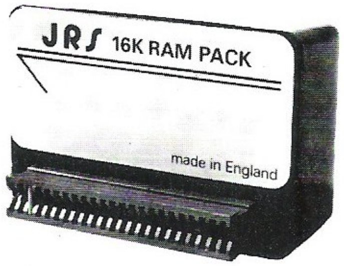
ONLY £4.95 ($9.90)

As reviewed in 'Your Computer' March 1982

16K RAM PACK

ONLY £26.50 ($49.95)

Quite simply the best available plus FREE 'Alien Attack' (7K-M/code) on cassette - Value £5.75 ($11.00).



Fully built tested and guaranteed.
Uses existing power supply (min. 600 m.a.)
Compatible with printer.
No wobble problems.
Gold plated edge connector for perfect contact with your ZX81.
Normally despatched within 10 days of receipt of your order.

With advances in microtechnology we are now able to offer the **64K RAM PACK.**
Same quality as the 16K but giving massive memory to your ZX81, with external dimensions no larger than the 16K Ram Pack (above).

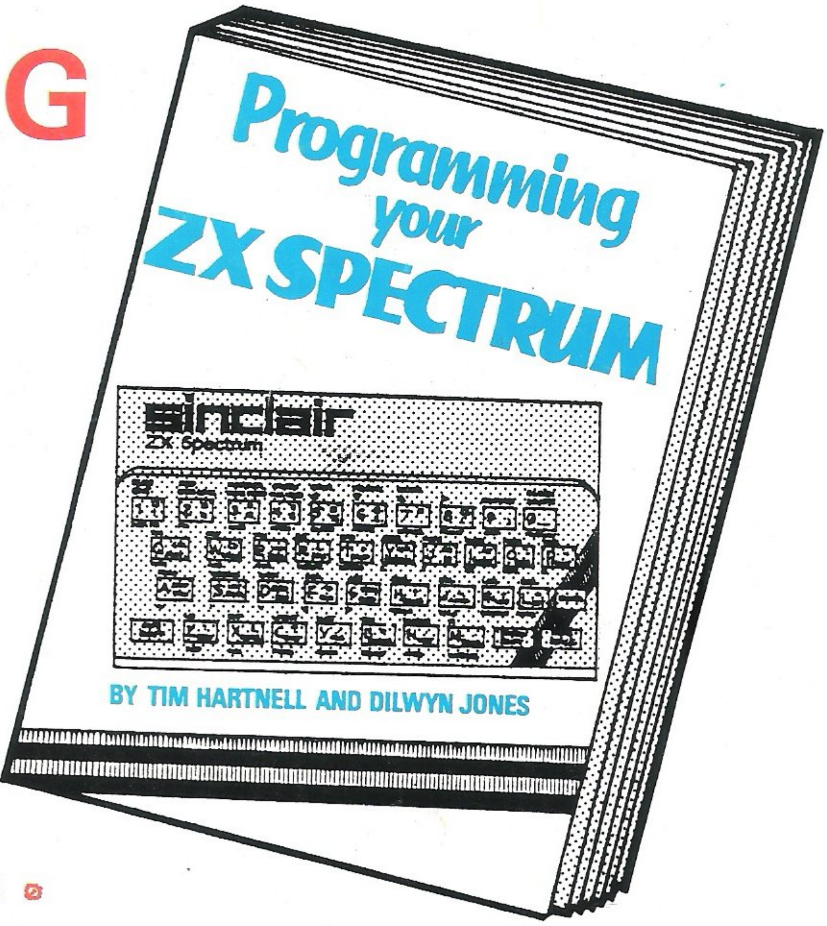
ONLY £62.95 ($119.95).

OVERSEAS CUSTOMERS PLEASE NOTE: ALL PRICES ARE APPLICABLE FOR EXPORT, INCLUDING AIRMAIL POSTAGE. PAYMENT MAY BE MADE IN STERLING (MONEY ORDER AVAILABLE AT YOUR BANK) OR YOUR OWN U.S. $ CHEQUE, MADE PAYABLE TO *JRS SOFTWARE*. DESPATCH NORMALLY 7 DAYS FROM RECEIPT OF ORDER.

JRS SOFTWARE

19 WAYSIDE AVENUE, WORTHING, SUSSEX. BN13 3JU Telephone: (0903) 65691

THE BOOK YOU'VE BEEN WAITING FOR!


Programming your ZX SPECTRUM
sinclair
ZX Spectrum
BY TIM HARTNELL AND DILWYN JONES

Examines and explains
every function
on the ZX Spectrum

Just £6.95

Try this program from the book:

```
 5 PAPER 0: CLS : BORDER 0
10 FOR X=0 TO 255
20 PLOT X,0
30 DRAW OVER 1;255-X*2,175
40 NEXT X
50 FOR Y=0 TO 175
60 PLOT 0,Y
70 DRAW OVER 1;255,175-Y*2
80 NEXT Y: REM © J. Ruston
```

Takes you through programming your ZX Spectrum from first principles right through to such things as defining your own graphics, MERGE, READ/DATA, SCREEN$, POINT and DRAW.

ZX Spectrum

THIS BOOK IS TOTALLY ORIGINAL. IT IS NOT BASED ON ANY OTHER BOOK, & THE PROGRAMS ARE ALL NEW

Interface,
Dept. PS,
44–46 Earls Court Road,
London W8 6EJ.

Please send me the following:

( ) PROGRAMMING YOUR ZX SPECTRUM — £6.95
( ) GETTING ACQUAINTED WITH YOUR ZX81 — £5.95
( ) 39 TESTED PROGRAMS FOR THE ACORN ATOM — £6.45
( ) LET YOUR BBC MICRO TEACH YOU TO PROGRAM — £6.45
( ) THE BBC MICRO REVEALED — £7.95
( ) PASCAL FOR HUMAN BEINGS — £4.95
( ) A sample issue of INTERFACE, the monthly ZX magazine published by the National ZX Users' Club — £1.00

I enclose a total of £__________

Name ____________________________

Address ____________________________

____

BBC Micro

User-defined graphics chapter tells you how to create your own DOTMAN game!