



The Book of Listings

Fun Programs for
the BBC Microcomputer

Tim Hartnell and Jeremy Ruston

The Book of Listings



The Book of Listings

Fun Programs for
the BBC Microcomputer

Tim Hartnell and Jeremy Ruston



© Tim Hartnell and Jeremy Ruston 1982
First published 1982

Printed in Great Britain by
Spottiswoode Ballantyne Ltd.
Colchester and London

Published by the
British Broadcasting Corporation
35 Marylebone High Street
London W1M 4AA

ISBN 0 563 16534 0

Contents

- 8 **Introduction**
- 9 **Racetrack**
A trackful of hazards.
- 14 **Turtle Graphics**
Create your own geometrical patterns.
- 16 **Numbfinger**
A fast and furious game of memory; your fingers follow the numbers.
- 19 **Cairo**
You are a harassed taxi-driver in one of the most crowded cities on earth.
- 24 **Little Eliza**
A tongue-in-cheek session with a cut-price psychiatrist.
- 34 **Life**
Birth, growth and death in a sci-fi colony of organisms.
- 40 **Fairground Organ**
The old style of funfair music.
- 44 **Space Storm**
A storm of asteroids that your space-ship has to destroy.
- 52 **Outlaw**
You're the Sheriff of Mean City and the baddies riding into town mean business.
- 55 **3D Super Plot**
Draw three-dimensional figures, view them and change them.

- 62 **Codebreaker**
Break the code before it breaks you.
- 66 **Masterword**
Similar to 'Codebreaker' but uses words.
- 70 **Bomb Squad**
Your space-ship goes on a bombing run.
- 73 **Dome Dweller**
A variation on one of the most popular computer games.
You play the ruler of a space colony.
- 82 **Wordsquare**
A computer imitation of a sliding plastic games puzzle.
- 86 **Tyranno**
You do battle with a prehistoric monster.
- 90 **Franglais**
Let the computer take over the burden of writing bad French
for you.
- 94 **Rain Catcher**
A colourful game of reflexes.
- 99 **Reversi**
A classic game of placing pieces on squares.
- 106 **Poet**
Poetry - as a computer would write it.
- 111 **Magic Square**
A number guessing game.
- 114 **Romland**
Beasts and gold mines and intrepid you.
- 123 **Gomoku**
A very popular game. You have to get five pieces in a line
before the computer does.
- 128 **Safari**
Elephant blobs stampede towards you, the human asterisk!
- 133 **Character Definition**
Writing your own character shapes.

- 138 **Graphic Displays Section**
- 139 **String Art**
String art patterns.
- 141 **Cat o'Six Tails**
Psychedelic curves.
- 143 **Chess Art**
Colourful ever-changing squares.
- 145 **Prison Bars**
'Moiré' patterns.
- 147 **Eight o'Clock**
An optical earthquake.
- 149 **Polar-plotting**
Flower-shaped patterns.
- 150 **Cartwheel**
Cartwheel effects.
- 151 **Quadrant String Art**
As in 'String Art' above, but in four quadrants, three of which are reflections of the pattern in the first.
- 153 **Oval**
Moving ovals of colour that finally spin.
- 155 **Goodbye**
A surprise farewell.
- 156 **Typing in Program Listings**
- 157 **Character Count Scale**

Introduction

Welcome to the first BBC Book of Listings. There are a host of games and other programs for you here. They range from arcade-like action programs, through board games which will tax your wits, to some startling graphics demonstrations.

We've tried to make the most of the tremendous sound and colour potential of the BBC microcomputer, and have written most programs so that they will run on both A and B model machines. The programs were developed on both the A and B model machines with the 0.1 Operating System.

Important The listings in the book should not be typed in exactly as you see them, but as you would normally (see pages 156 and 157).

Structured programming techniques have been used as far as possible. Although programs may thus be a little longer than strictly necessary, they do tend to be relatively easy to debug and modify. Many of the program notes include suggestions on how you can adapt the programs to make them your own and to develop them further. This is always well worth while, for you then put your own stamp on them. Further, working through other people's programs alerts you to useful programming techniques. Take a critical attitude to everything you see here and improve and tailor programs to your taste.

We have as far as possible avoided multistatement lines and unconditional GOTOs; used, whenever possible, procedures, with the game being called from a series of procedures cycling within a master REPEAT/UNTIL loop; used REM statements, or a line of asterisks (and sometimes both) to break the program down into clearly separate routines which perform specific tasks.

Now it's time to get on with having fun with your BBC microcomputer.

Good games-playing!

Tim Hartnell
Jeremy Ruston

Models A and B

Racetrack

The Game

You drive a car (which looks remarkably like a hash symbol) around a race track, trying to keep it moving for as long as possible without hitting the walls.

The 'A' key moves your car up on the Micro Racetrack, the 'Z' moves it down, while the ',' moves you to the left and the '.' to the right. The 'greater than' and 'less than' symbols on the ',' and '.' keys indicate the direction of travel the car will take if you press these keys. You control the car's up-and-down movements with your left hand, the right and left directions with your right hand.

There are five levels of difficulty in the game, ranging from fairly easy (five) to nearly impossible (one).

The Program

10 Title

20 Set the mode

30 Send action to the initialisation procedure.

40-60 Master REPEAT/UNTIL loop. A equals zero unless the car hits something, when its value changes to one.

70 Sends computer to the 'smash' procedure.

80-510 This is the initialisation procedure. Line 90 turns off the cursor, the initial direction of the car is set to the right (B\$ = '.' in line 100), and the car is placed three characters across (CARA) and two down (CARD). Lines 140 to 220 print out the instructions, while 230 to 250 accept your choice of the degree of difficulty. Line 260 sets the timer to zero, so this can be used to indicate how long you have managed to keep driving, when the game is actually underway.

520-680 This procedure allows you to 'drive' the car, and checks to see if it has crashed. Lines 530 and 550 supposedly produce the sound of the car being driven. Line 540 prints the duration of the race in seconds and tenths of a second in the top right hand corner of the screen. Line 560 prints the car in its current position, and 570 prints a blank in the same position. This happens very quickly, so the car appears to move fairly smoothly, even in the slowest (easiest) game. The routine from 580 to 650 reads the keyboard and

changes the direction of the vehicle in response to the value assigned to A\$. Line 590 keeps the vehicle moving in the same direction as it was, if no key is pressed. The same line jumps the section (lines 610 to 640) that changes the car's direction if a new direction is required. Line 600 is a 'mugtrap' designed to reject any key press which is not one of the four required. Line 660 checks the position where the car is about to be printed; if it finds an asterisk there, it knows the car is about to crash and changes the crash flag (A) from zero to one. Line 670 reprints the car in position.

690–840 This is the crash procedure, where the computer is sent if the crash flag has been changed from zero to one (from line 60). The buffer is cleared (line 700) and the time you have survived is set to variable W (line 710) before a dramatic sound of crashing is generated (lines 720 to 740). The final REPEAT/UNTIL loop runs from 750 to 830; prints the word 'smash!' in random places and colours; and generates a cacophony from sound channels one, two and three.

Suggestions for improvement

- Change the sound and cut it off after a second or so.
- Change the shape of the racetrack by changing the print statements 280 to 500; or work out a routine to POKE a random set of obstacles into a frame which you have previously printed.
- Add a 'high score' (longest race) feature, so you can try to better your score from game to game, without having to start all over again.
- Increase the number of levels of difficulty. Note that the 'level of difficulty' ('D') works by changing the length of the sound statement in line 550.
- Instead of your having to press ESCAPE and then RUN if you lose, program a facility to offer a new game or to end with the computer back in Mode 7.

The Listing

```
10 REM Racetrack
20 MODE7
30 PROCinitialise
40 REPEAT
50 PROCmovecar
60 UNTIL A=1
70 PROCsmash
80 DEFPROCinitialise
90 VDU 23;8202;0;0;0
100 B$="."
110 A=0:REM end of race flag
120 CARA=3:REM position car across
130 CARD=2:REM position car down
```

```

140 PRINT''CHR$(130);"Welcome to the Micro Ra
cetrack"
150 PRINT'CHR$(131);"The A key moves your car
up, the Z key"
160 PRINT CHR$(131);"moves it down, the , move
s you to the"
170 PRINT CHR$(131);"left and the . to the rig
ht"
180 PRINT'CHR$(129);"You have to continue driv
ing for as"
190 PRINT CHR$(129);"long as possible. Your ti
me is shown"
200 PRINT CHR$(129);"in 10ths of a second in t
he top corner"
210PRINT'CHR$(133);"Enter your skill level, f
rom 1 to 5"
220 PRINT'CHR$(133);"Five is easiest, one the
hardest"
230 D=GET
240 D=D-48
250 IF D<1 OR D>5 THEN 230
260 TIME=0
270 CLS
280 PRINT ' "*****
"
290 PRINT "***          ****          **
**"
300 PRINT "**          *****          **          *
**"
310 PRINT "*          *          ***
*"
320 PRINT "*          **          **          *****
*"
330 PRINT "*          **          **          *****
*"
340 PRINT "*          **          **          *****
*"
350 PRINT "*          *****          *          *****
*"
360 PRINT "*          **          ***          *****
*"
370 PRINT "*          *****          *****          *****
*"
380 PRINT "*          *****          *****          **          *
**"
390 PRINT "*          *****          **          *****          **
**"
400 PRINT "*          *****          *****          *****          *

```

```

**"
410 PRINT "*" *****
**"
420 PRINT "*" ** ** *****
**"
430 PRINT "*** **** **
**"
440 PRINT "**** ***** ** *****
**"
450 PRINT "***** ***** *** *****
**"
460 PRINT "***** ***** **** *****
**"
470 PRINT "***** ***** *** *****
**"
480 PRINT "*** *** *****
**"
490 PRINT "**** ** *****
**"
500 PRINT "*****
**"
510 ENDPROC
520 DEFPROCmovecar
530 SOUND 0,-3,1,1
540 PRINT TAB(31,1);CHR$(133);(TIME DIV 10)/10

550 SOUND 0,-7-RND(8),254,D
560 PRINT TAB(CARA,CARD);"#"
570 PRINT TAB(CARA,CARD);" "
580 A$=INKEY$(0)
590 IF A$="" A$=B$:GOTO 610
600 IFA$<>"A" AND A$<>"Z" AND A$<>"," AND A$<>
"." A$="."
610 IF A$="A" AND CARD>2 CARD=CARD-1
620 IF A$="Z" AND CARD<22 CARD=CARD+1
630 IF A$="," AND CARA>1 CARA=CARA-1
640 IF A$="." AND CARA<40 CARA=CARA+1
650 B$=A$
660 IF?(HIMEM+CARA+40*CARD)=ASC("*") THEN A=1
670 PRINT TAB(CARA,CARD);"£"
680 ENDPROC
690 DEF PROCsmash
700 *FX 15,0
710 W=TIME DIV 10
720 FOR T=1 TO 10
730 SOUND 0,-15,RND(10),T
740 NEXT
750 REPEAT

```

```
760 PRINT TAB(RND(30)-1,RND(23));CHR$(RND(5)+1
28);"smash!"
770 PRINT TAB(31,1);CHR$(RND(5)+128);W/10
780 FOR T=1 TO 10 STEP 2
790 SOUND 3,-15,20*T,1
800 SOUND 2,-15,RND(20),RND(3)
810 SOUND 1,-15,RND(20),RND(3)
820 NEXT
830 UNTIL FALSE
840 ENDPROC
```

Models A and B

Turtle Graphics

The Program

You control an invisible 'turtle', moving it around the screen to create geometrical patterns. The commands to move the cursor are:

init Initialises the turtle to point in a specified direction. (All angles are given in degrees).

anticlockwise Turns the turtle counterclockwise by a specified number of degrees.

clockwise Turns the turtle clockwise by a specified number of degrees.

position Moves the cursor to specified co-ordinates.

draw Moves the turtle in the direction it is pointing, for a specified distance, leaving a white trail.

These commands are procedure calls mixed with normal BASIC statements. The example the program gives draws a hexagon – see lines 50 to 100. When you start you probably will want to do something a little simpler – maybe just drawing a single character on the screen. Children often find a moving turtle easier to understand than the grid arrangement usually used on the BBC Microcomputer. Notice that the numbers 640, 512, 0, 100 and 60 will have to be changed to get different results.

1000 Starts the definition of PROCinit.

1010 Sets the variable 'angle' to the direction specified, MOD 360. Here 'angle' is used to store the direction the 'turtle' is pointing towards.

1020 Ends PROCinit.

2000 Starts the definition of PROCanticlockwise.

2010 Decrements 'angle'. The extra MODs ensure the direction ('angle') does not become negative.

2020 Ends PROCanticlockwise.

3000 Starts the definition of PROCclockwise.

3010 Increments 'angle'. The MOD ensures the angle does not exceed 359.

3020 Ends PROCclockwise.

4000 Starts the definition of PROCposition.
 4010 Sets the x and y co-ordinates of the turtle.
 4020 Ends PROCposition.
 5000 Starts the definition of PROCdraw.
 5010 Moves the graphics cursor to the current turtle position.
 5020 Computes the new x co-ordinate of the turtle.
 5030 Computes the new y co-ordinate of the turtle.
 5040 Draws a line to the new turtle position.
 5050 Ends PROCdraw.

The Listing

```

10 REM Turtle Graphics
20 REM See text for instructions
30 REM -----
40 MODE 4
50 PROCposition(640,512)
60 PROCinit(0)
70 FOR T=1 TO 6
80 PROCdraw(100)
90 PROCanticlockwise(60)
100 NEXT T
110 END
999 REM *****
1000 DEF PROCinit(direction)
1010 angle=direction MOD 360
1020 ENDPROC
2000 DEF PROCanticlockwise(step)
2010 angle=(angle+360-step) MOD 360
2020 ENDPROC
3000 DEF PROCclockwise(step)
3010 angle=(angle+360+step) MOD 360
3020 ENDPROC
4000 DEF PROCposition(xpos,ypos)
4010 x=xpos
4020 y=ypos
4030 ENDPROC
5000 DEF PROCdraw(length)
5010 MOVE x,y
5020 x=SIN(RAD(angle))*length+x
5030 y=COS(RAD(angle))*length+y
5040 DRAW x,y
5050 ENDPROC
5060 REM *****

```

Models A and B

Numbfinger

The Game

This game tests your memory. When you first press RUN, a digit from 1 to 4 appears on the screen, along with a distinctive tone. Number four's tone is the highest, number one's the lowest. You must press the same number, and wait till the number appears and the tone sounds.

Once you've done this, a multicoloured wall of hash symbols obscures the screen for a few seconds. Then the first number-and-tone combination is repeated, followed by a second one. You must press the first number, and once it and its respective tone are seen and heard, press the second number. This will continue, with an additional tone/number being added to the sequence, until a total of seven tones/digits has been presented and repeated correctly. If you do this you win the game. Failing to repeat the sequence at any time ends the game. You get a score related to the number you managed to work out correctly.

There is no reason why the same digit cannot appear up to seven times in the sequence (although that is most unlikely). If the same digit is repeated, it appears on the screen with its tone, is replaced by a coloured block for a moment, then reappears at the same position with the tone being sounded again. You *must* press this digit's key twice, allowing the digit to appear and tone to sound after the first press, before completing the sequence.

The Program

10-20 REM statements for the title.

30 Sets the mode.

40 A\$ is used to hold the sequence, as you can see in the routine starting at 120.

50 M is the maximum number of digits in a sequence. Increase this to make the game harder, reduce it to make it simpler.

60 Z is a variable set equal to one, and is used in several places in the program.

70-90 This routine sets up the digits which will have to be remembered, storing them in string A\$.

100 X is the number of digits which must be remembered each time a sequence is presented; there is only one digit in the first 'sequence'.

110 Starts the REPEAT/UNTIL loop for the main game.

120–170 This loop presents the digits, selecting them from the string (line 130); printing them in position in a random colour (140); producing an appropriate tone (150); adding a delay which gets shorter for each additional digit in the sequence (160); overprinting a little coloured square where the digit appeared (180); adding another pause (190); and then going back for more (210).

220–320 This routine accepts, and processes, the player's input. Line 230 waits until the keyboard is untouched, and 240 waits until a new key is touched. Line 250 clears the screen, and 260 sets B\$ to the key being touched. A digit and tone are produced (lines 270 and 280) in accord with the key touched; then there is a short delay (290) before the program checks if the key pressed corresponds to the relevant digit in the sequence. If not, the program goes on to the procedure called 'end'.

Line 310 clears the buffer, to be ready for the next key press, and 320 starts it again if all digits in the sequence have not been repeated and checked.

330 If you have repeated M digits correctly, you are declared the winner. You must press ESCAPE to end the program from here.

340–400 One is added to the number of digit/tones you must remember (340); a wall of pound signs or hash symbols appears (our printer reproduces the hash as a pound sign – use anything you like here); and the screen is cleared (390) in preparation for going back to the line after the initial REPEAT (400).

410–460 This procedure ('end') is where you are sent if you fail to repeat the sequence correctly. It prints your score, in randomly chosen colours, over and over again, while generating an offensively random bit of music until ESCAPE is pressed.

Suggestions for improvement

- Allow the player to choose the maximum number of digits which will be in the sequence from game to game.
- Add a 'highest score' feature so that the number of correct digits remembered in one game is compared with the maximum in subsequent games.
- Rewrite it so that eight digits, positions and tones are involved.
- Make a provision for the unsuccessful player to be asked if he or she would like to play the game again; cut off the sound after a couple of seconds.

The Listing

```
10 REM Close encounters of
20 REM     the NUMBFINGER kind!
30 MODE7
```

```

40 A$=""
50 M=7
60 Z=1
70 FOR A=Z TO M
80 A$=A$+STR$(RND(4))
90 NEXT A
100 X=Z
110 REPEAT
120 FOR Q=Z TO X
130 L=4*((ASC(MID$(A$,Q,1)))-48)
140 PRINTTAB(M,L+3);CHR$(128+RND(5))MID$(A$,
Q,1)
150 SOUND 3,-15,40*((ASC(MID$(A$,Q,1)))-48),6
160 FOR J=Z TO 1000-20*X
170 NEXT
180 PRINTTAB(M,L+3);CHR$(133)CHR$(255)
190 T=TIME:REPEAT UNTIL TIME-T=50
200 CLS
210 NEXT
220 FOR B=Z TO X
230 IF INKEY$(0)<>"" THEN 230
240 IF INKEY$(0)="" THEN 240
250 CLS
260 B$=INKEY$(50)
270 PRINTTAB(M,3+4*(ASC(B$)-48));CHR$(128+
RND(5))B$
280 SOUND 3,-15,40*((ASC(B$)-48)),10
290 T=TIME:REPEAT UNTIL TIME-T=30
300 IF B$<>MID$(A$,B,1) PROCend
310 *FX 15,0
320 NEXT
330 IF X=M PRINT CHR$(128+RND(5))"You win!":
GOTO 330
340 X=X+Z
350 CLS
360 FOR W=Z TO 5*(M+M)
370 PRINT CHR$(128+RND(5))"#####
#####"
380 NEXT
390 CLS
400 UNTIL FALSE
410 DEF PROCend
420 REPEAT
430 PRINTTAB(8,8);CHR$(128+RND(5))"You
scored ";X-Z
440 SOUND 1,-15,RND(100)+150,1
450 UNTIL FALSE
460 ENDPROC

```

Models A and B

Cairo

The Game

In this game you are an irate cab driver, caught up in one of Cairo's notorious traffic jams. Your blood pressure rises as you steer through the traffic at ever-increasing speed. You dare not crash into any other vehicles. If you stay the course, you have won.

Your car appears as a little red arrow, highlighted with a light blue streak. The other vehicles appear as yellow asterisks. If you do not press any key, you move across the screen horizontally by default; and when you reach the right hand side, you re-appear on the left-hand side.

To move down, press the '/' key; to move up, press the ':' key. These keys can be used with or without shift. At each move up or down, a short tone will sound: the sound of the 'horns' of other irate drivers. All the way through the game, the background sound is that of your blood pressure rising. When this sound stops, you are near the end of your ordeal.

Note You do not need to move the car horizontally, as this happens anyway if you don't move it up or down.

You can alter the number of other vehicles on the screen by changing line 20 of the listing.

The Program

20 'asterisks' sets the number of other vehicles on the road.

30 Puts the computer in Mode 7.

40 Calls PROCSETUP, which sets up the screen and places the asterisks on it.

50 Initialises X and Y which contain the x and y positions of your cab.

60 TIM is the time delay between each movement, multiplied by two. It is decremented at each movement.

80 Starts the main REPEAT loop of the game.

90 HIT is a boolean variable which is true if you have crashed into another vehicle.

100 Calls PROCPLACE. PROCPLACE puts your cab at position X, Y on the screen.

110 Resets TIME.

120 Gives a delay of TIM/2 centi-seconds.
130 Turns off the sound channel 1.
140 Gets a key press from the user.
150 Empties both the SOUND and keyboard buffers.
160 Plays a sound through channel 3 of frequency TIM.
170 Plays a sound through channel 2 of frequency TIM + 1. This produces a pleasant chord, with the tone from line 160.
180 Calls PROCREMOVE. PROCREMOVE removes your cab from position X, Y.
190 If the 'up' key was pressed, decrement Y, and SOUND hooter.
200 If the 'down' key was pressed, increment X and SOUND hooter.
210 If an attempt was made to move too close to the bottom of the screen, this line puts the cab on the bottom line of the display.
220 Similarly, if an attempt was made to move off the top of the screen, puts the cab on the top line.
230 Increments the X co-ordinate of the cab's position.
240 If the cab's new position is over an asterisk, put HIT to TRUE. The function FNADDRESS(X, Y) gives the address of the screen cell with co-ordinates X, Y.
250 If an attempt was made to move beyond the right-hand edge of the screen this line puts the cab on the left-hand edge.
260 Decrements TIM.
270 Stops the loop either when a hit is registered, or when the game has run its course.
280 If the game ended with a hit, calls PROCHIT.
290 Calls PROCEND, which restores the screen to normal, and tells you how well you did.
300 ENDS the program.

320 Starts the definition of PROCSETUP.
330 Sets all variables used in this procedure to be LOCAL.
340 Starts a loop, pointing to the left-hand location of each screen line.
350 Puts the code for yellow alphanumerics into each of the locations.
360 Ends the loop.
370 Starts a loop from one to the number of other vehicles on the road.
380 Picks a random x co-ordinate for each asterisk.
390 Picks a random y co-ordinate for each asterisk.
400 Puts the asterisk on the screen.
410 Ends the asterisk loop.
420 Sets 'moves' to zero.
430 Ends PROCSETUP.

450 Starts the definition of PROCEND.
470 Clears the screen.
480 PRINTs part of the message.

490 If you survived the course, tells you so.

500 Ends PROCEND.

520 Defines the function FNADDRESS(X, Y). This function uses a standard calculation to give the address of location X, Y on the screen.

540 Starts the definition of PROCPLACE(X, Y). This procedure places your taxi at position X, Y on the screen.

550 Places a right square bracket at position X, Y on the screen. This appears as a little arrow pointing to the right in the Mode 7 character set.

560 Places the code for light blue alphanumerics at the start of the line on which the cab has been placed.

570 Places the 'new background colour' colour at the next screen location. This gives a horizontal light blue line across the screen.

580 Places the red alphanumerics code at the next screen location. This makes the foreground colour of this line red.

590 Ends PROCPLACE.

610 Starts the definition of PROCPLACE(X, Y). This procedure removes your cab from the position X, Y on the screen.

620 Places a space at position x, y.

630 Places the code for yellow alphanumerics at the left-most column.

640 Removes the 'new background colour' code.

650 Removes the red alphanumerics code.

660 Ends PROCPLACE.

690 Starts the definition of PROCHIT.

690 Resets TIME.

700 Starts a REPEAT loop to generate random sound effects.

710 Clears all buffers, including the sound buffers.

720 SOUNDS a random note.

730 Stops the loop after two seconds.

740 Turns off any remaining sound effects.

750 Ends PROCHIT.

Suggestions for improvement

- The graphics could be made more appealing, if you have a model B, by transferring the game to Mode 4.
- The ENVELOPE command could be used to brighten up the sound effects.
- The asterisks could be made to move as the game progresses, rather than remaining static.
- Remove the cursor while the game is being played, then restore it when it ends.

The Listing

```
10 REM *****
*****
20 asterisks = 200
30 MODE 7
40 PROCSETUP
50 X=0:Y=12
60 TIM=200
70 REM *****
*****
80 REPEAT
90 HIT=FALSE
100 PROCPLACE(X,Y)
110 TIME=0
120 REPEAT UNTIL TIME>(TIM/2)
130 SOUND 1,0,0,0
140 A$=INKEY$(1)
150 *FX 15,0
160 SOUND 3,-10,TIM,255
170 SOUND 2,-10,TIME+1,255
180 PROCREMOVE(X,Y)
190 IF A$=":" OR A$="*" THEN Y=Y-1:SOUND 1,-1
5,100,255
200 IF A$="/" OR A$="?" THEN Y=Y+1:SOUND 1,-1
5,100,255
210 IF Y>24 THEN Y=24
220 IF Y<0 THEN Y=0
230 X=X+1
240 IF ?FNADDRESS(X,Y)=42 THEN HIT=TRUE
250 IF X>37 THEN X=0
260 TIM=TIM-1
270 UNTIL HIT=TRUE OR TIM=0
280 IF HIT=TRUE THEN PROCHIT
290 PROCEND
300 END
310 REM *****
*****
320 DEF PROCSETUP
330 LOCAL counter,X,Y
340 FOR counter=0 TO 960 STEP 40
350 ?(HIMEM+counter)=3
360 NEXT counter
370 FOR counter=1 TO asterisks
380 X=RND(37)
390 Y=RND(25)-1
400 ?(HIMEM+X+2+Y*40)=42
410 NEXT counter
420 moves=0
```



```

430 ENDPROC
440 REM *****
*****
450 DEF PROCEND
460 LOCAL
470 CLS
480 PRINT ''' " You had ";TIM;" moves to go."
490 IF TIM=0 THEN PRINT ' " Therefore you
won. !!!"
500 ENDPROC
510 REM *****
*****
520 DEF FNADDRESS(X,Y)=HIMEM+X+2+Y*40
530 REM *****
*****
540 DEF PROCPLACE(X,Y)
550 ?FNADDRESS(X,Y)=ASC("J")
560 ?(HIMEM+Y*40)=6
570 ?(HIMEM+Y*40+1)=29
580 ?(HIMEM+Y*40+2)=1
590 ENDPROC
600 REM *****
*****
610 DEF PROCREMOVE(X,Y)
620 ?FNADDRESS(X,Y)=ASC(" ")
630 ?(HIMEM+Y*40)=3
640 ?(HIMEM+Y*40+1)=ASC(" ")
650 ?(HIMEM+Y*40+2)=ASC(" ")
660 ENDPROC
670 REM *****
*****
680 DEF PROCHIT
690 TIME=0
700 REPEAT
710 *FX 15,0
720 SOUND RND(4)-1,-15,RND(256)-1,255
730 UNTIL TIME>200
740 *FX 15,0
750 ENDPROC
760 REM *****
*****

```

Little Eliza

The Game

In this program the computer acts as a 'psychiatrist', while you are the 'patient'. A free-format dialogue passes between you.

Note The computer is programmed to be a 'psychiatrist', *not* to argue, so try not to fall into the easy trap of just exchanging insults with it. It has been programmed to respond to 36 'keywords'. We chose these because they are the sort of words people use often. You may not agree. You may well get a more convincing dialogue if you change these to words you are more likely to use.

The Program

10 Puts the computer in Mode 7.

40 Prints the first part of the opening message.

50 Starts trying to be a mother-substitute.

60 ANS() holds which of the three possible responses to a keyword was used last.

70 M\$() holds all the possible responses by the computer – sorry, 'Eliza'.

80 Calls PROCinit, which reads all the answers into M\$.

90 P\$ holds the response you last typed in. This is set to a plus sign, so that the mechanism for stopping you typing the same thing twice in a row works all the time.

100 Starts the main REPEAT loop of the program. This one carries on until you type 'shut up'.

110 Sets the your response to the null string, because otherwise the comparison in the next-line-but-one would cause a 'No such variable' error.

120 Starts the REPEAT loop concerned with getting a valid response from you.

130 If your response was the same as the last one, prints a message telling you not to repeat yourself.

140 Inputs your reply. INPUT LINE is used to ensure that if you type a comma, all the typing will still be used.

150 Stops the loop when you do not repeat yourself.

160 Sets the old response to the new response, ready for the next input session.

170 Translates your input to upper case if it was lower case, or just leaves it if it was upper case.

180 This line uses nested function calls. The function FNanswer will generate a response to A\$, and the function FNoutput will convert that response to lower case.

190 Stops the loop when 'shut up' is typed.

200 Prints a suitable response to your typing 'shut up'.

210 Ends the program.

230 Starts the definition of FNtranslate_input(A\$).

FNtranslate_input(A\$) takes the string A\$ and makes the substitutions in the DATA statements starting at line 370. Thus all occurrences of the word ARE will be replaced with AM%. The percentage sign is to stop the program later replacing AM with ARE (see line 490).

240 Sets the DATA statement pointer to line 370.

250 Sets all variables used in this function to LOCAL.

260 Adds spaces to both ends of A\$. This is because all substitutions are made with words, and words are only known to be words when they are flanked by spaces. Thus the words starting and finishing a sentence will not be counted as such without this measure.

270 Starts a FOR loop for each of the 15 possible substitutions.

280 Reads the two words that make up the substitution.

290 Adds spaces to the first word.

300 Adds spaces to the second word.

310 Starts a REPEAT loop for that particular substitution.

320 Finds out the position of the search word in A\$. INSTR itself is not used because INSTR is not totally reliable in functions when the second argument is longer than the first. FNinstr checks the lengths of both arguments.

330 If X\$ is in A\$, the substitution is made.

340 The process continues until P = 0, when all occurrences of X\$ in A\$ have been dealt with.

350 Ends the major loop.

360 Exits the function with A\$.

530 Starts the definition of the function FNchange_case_to_upper. This function changes any lower case letters in its argument to upper case.

540 Sets all variables used in this function to LOCAL.

550 B\$ is the destination string, and it is set to null, to stop the 'No such variable' error message coming up.

560 Starts a loop through all the characters of A\$.

570 Picks out the T'th character of A\$.

580 If the character is 'greater than 'Z'' then subtracts 32 from its code. This line carries out the actual conversion.

590 If the character is a single quote, removes it.

600 Adds the character to the destination string.

610 Ends the loop.

620 Exits the function with the destination string, B\$.

640 Starts the definition of FNoutput(A\$). This function changes every character of A\$ to lower case, except the first, which is kept in upper case. The percentage signs are also taken out.

650 Sets all variables used in the function to LOCAL.

660 Sets B\$, the destination string, to a null string.

670 Sets the Boolean variable Y to false. This variable is false if the first character of A\$ is being processed, and TRUE otherwise.

680 Starts a loop through all the characters of A\$.

690 Picks out the T'th character of A\$.

700 If it's a capital letter, and Y is true, it is converted to lower case by subtracting 32 from its code.

710 If it's an alphabetic character, sets Y to TRUE. This means that if the first character is non-alphabetic, ie. a space, it still waits till it meets an alphabetic character before starting conversions to lower case.

720 If it's a percentage sign, removes it.

730 Adds the character to the destination string.

740 Ends the loop.

750 Exits the function, with B\$.

770 Starts the definition of FNinstr(A\$,B\$). This is a debugged version of INSTR (see page 281 of the User Guide).

780 If the second argument is longer than the first, exits with zero.

790 Else just uses INSTR to provide the result for the function.

810 Starts the definition of FNanswer(A\$). This function thinks up a suitable answer to your response.

820 Declares LOCAL variables. Notice that B\$ is not a LOCAL variable, since it is used again in FNmessage(X, Y).

830 Restores the DATA pointer to line 910.

840 K counts which of the keywords was found to be in the input string. This is initially set to zero.

850 Starts a REPEAT loop concerned with finding keywords in the patient's response.

860 Reads the next keyword.

870 Increments K.

880 UNTIL either the end of the list of keywords has been reached, or a match is found.

890 Increments the correct element of ANS. This is so that answers are used in cycles.

900 Uses the ANS(K)'th answer out of the K'th group of three.

910 Starts the list of keywords. Notice the order in which they are placed – 'YOU ARE' is found before 'YOU' on its own, and words like 'FRIENDS' are a last resort.

970 Starts the definition of FNmessage(X, Y). This function picks out the correct response and, if needed, adds the translated part of the human response on to the end of it, depending on whether the answer ends in an asterisk or not.

980 Sets up the LOCAL variables.

990 Picks the answer string from the string array which holds all the answers.

1000 If the rightmost character is not an asterisk, then returns the string as it is.

1010 Or else, takes off the asterisk.

1020 Combines the answer with the user's input, translated with FNtranslate_input.

1040 Starts the definition of PROCinit. This routine just reads in all the answers.

Suggestions for improvement

- The best way to improve 'Eliza' is to alter the responses to put your personal stamp on them. You do this by either altering the data statements from lines 1140 onwards, being careful to keep the same number, or by adding extra keywords.

- Each new keyword is added into the DATA statements at line 910, in the place where it is unlikely to interfere with other keywords. Once this has been done, you must alter lines 60 and 70, to allow for more than 36 keywords.

- You must then add the responses to the keywords you've added into the DATA statement list in line 1140. This should be done in the same place (relative to the other items) where you added your keywords in the keyword table. Each group of three responses is REMmed to show which keyword it is associated with.

The Listing

```
10 MODE 7
20 REM ELIZA
30 REM *****
*****
40 PRINT '''"Hello. My name is 'Eliza'."
50 PRINT '"Tell me what your problem is "'
60 DIM ANS(36)
70 DIM M$(2,36)
80 PROCinit
90 P$=""+"
100 REPEAT
110 A$=""
120 REPEAT
130 IF A$=P$ THEN PRINT "Please don't repeat y
oursel f. "
140 INPUT LINE "---->" A$
150 UNTIL A$<>P$
160 P$=A$
170 A$=FNchange_case_to_upper(A$)
180 PRINT FNoutput(FNanswer(FNchange_case_to_u
pper(" "+A$+" ")))
```

```

190 UNTIL FNinstr(A$,"SHUT UP")
200 PRINT "Shut up yourself ..."
210 END
220 REM *****
*****
230 DEF FNtranslate_input(A$)
240 RESTORE 370
250 LOCAL W,X$,Y$,P
260 A$=" "+A$+" "
270 FOR W=1 TO 15
280 READ X$,Y$
290 X$=" "+X$+" "
300 Y$=" "+Y$+" "
310 REPEAT
320 P=FNinstr(A$,X$)
330 IF P>0 THEN A$=MID$(A$,1,P-1)+Y$+MID$(A$,P
+LEN(X$))
340 UNTIL P=0
350 NEXT W
360=A$
370 DATA ARE,AM%
380 DATA WERE,WAS%
390 DATA YOUR,MY%
400 DATA YOU,I%
410 DATA IVE,YOU'VE%
420 DATA IM,YOU'RE%
430 DATA ME,YOU%
440 DATA I AM,YOU'RE%
450 DATA I HAVE,YOU'VE%
460 DATA MY,YOUR%
470 DATA YOURE,I'M%
480 DATA I,YOU%
490 DATA AM,ARE%
500 DATA WAS,WERE%
510 DATA YOUVE,I'VE%
520 REM *****
*****
530 DEF FNchange_case_to_upper(A$)
540 LOCAL T,C$,B$
550 B$=""
560 FOR T=1 TO LEN(A$)
570 C$=MID$(A$,T,1)
580 IF C$>"Z" THEN C$=CHR$(ASC(C$)-32)
590 IF C$="'" THEN C$=""
600 B$=B$+C$
610 NEXT T
620=B$
630 REM *****

```

```

*****
640 DEF FNoutput(A$)
650 LOCAL B$,T,C$,Y
660 B$=""
670 Y=FALSE
680 FOR T=1 TO LEN(A$)
690 C$=MID$(A$,T,1)
700 IF C$>="A" AND C$<="Z" AND Y THEN C$=CHR$(
ASC(C$)+32)
710 IF C$>="A" AND C$<="Z" THEN Y=TRUE
720 IF C$="%" THEN C$=""
730 B$=B$+C$
740 NEXT T
750=B$
760 REM *****
*****
770 DEF FNinstr(A$,B$)
780 IF LEN(B$)>LEN(A$) THEN=0
790=INSTR(A$,B$)
800 REM *****
*****
810 DEF FNanswer(A$)
820 LOCAL K
830 RESTORE 910
840 K=0
850 REPEAT
860 READ B$
870 K=K+1
880 UNTIL B$="NOKEY" OR FNinstr(A$.B$)
890 ANS(K)=(ANS(K)+1) MOD 3
900=FNmessage(ANS(K),K)
910 DATA " CAN YOU "," CAN I "," YOU ARE"," YO
URE "," I DONT "," I FEEL "
920 DATA " WHY DONT YOU "," WHY CANT I "," ARE
YOU "," I CANT "," I AM "," IM "
930 DATA " YOU "," I WANT "," WHAT "," HOW "," W
HO "," WHERE "," WHEN "," WHY "
940 DATA CAUSE, SORRY, DREAM, HELLO, " HI", MAYBE, N
O, " YOUR ", NAME
950 DATA " ALWAYS "," THINK "," ALIKE "," YES"
, " FRIEND", " COMPUTER", NOKEY
960 REM *****
*****
970 DEF FNmessage(X,Y)
980 LOCAL D$,F$
990 D$=M$(X,Y)
1000 IF RIGHT$(D$,1)<>"*" THEN =D$
1010 F$=LEFT$(D$,LEN(D$)-1)

```

```

1020=F$+FNtranslate_input (MID$(A$,FNinstr (A$,B$
)+LEN(B$))
1030 REM *****
*****
1040 DEF PROCinit
1050 RESTORE 1140
1060 LOCAL X,Y
1070 FOR Y=1 TO 36
1080 FOR X=0 TO 2
1090 READ M$(X,Y)
1100 NEXT X
1110 NEXT Y
1120 ENDPROC
1130 REM *****
*****
1140 DATA DON'T YOU THINK I CAN*
1150 DATA WHAT MAKES YOU THINK I CAN'T*
1160 DATA PERHAPS YOU WOULD LIKE TO BE ABLE TO*
1170 REM
1180 DATA PERHAPS YOU DON'T WANT TO*
1190 DATA WHY DO YOU WANT TO*
1200 DATA I DOUBT IT
1210 REM
1220 DATA WHY DO YOU THINK I'M*
1230 DATA DOES IT PLEASE YOU TO BELIEVE I AM*
1240 DATA HOW DID YOU KNOW
1250 REM
1260 DATA WHY DO YOU THINK I'M*
1270 DATA DOES IT PLEASE YOU TO BELIEVE I AM*
1280 DATA HOW DID YOU KNOW
1290 REM
1300 DATA DON'T YOU REALLY
1310 DATA WHY DON'T YOU*
1320 DATA IS IT ONLY YOU THAT DOESN'T*
1330 REM
1340 DATA TELL ME MORE ABOUT SUCH FEELINGS
1350 DATA DO YOU THINK SUCH FEELINGS ARE NORMAL
1360 DATA DO YOU THINK EVERYONE FEELS*
1370 REM
1380 DATA I COULD SAY THE SAME TO YOU
1390 DATA DO YOU THINK I SHOULD*
1400 DATA DO YOU REALLY BELIEVE THAT I DON'T*
1410 REM
1420 DATA PERHAPS YOU HAVEN'T TRIED HARD ENOUGH
1430 DATA MAYBE THERE'S JUST SOMETHING WRONG WI
TH YOUR MIND
1440 DATA MAYBE YOU ARE NOW IN A POSITION TO*
1450 REM

```


1460 DATA DOES THE QUESTION INTEREST YOU
1470 DATA DO YOU THINK I'M*
1480 DATA WOULD IT PLEASE YOU TO BELIEVE THAT I
M*
1490 REM
1500 DATA YOU TELL ME
1510 DATA PERHAPS YOU HAVEN'T TRIED HARD ENOUGH
1520 DATA MAYBE THERE'S JUST SOMETHING WRONG WI
TH YOUR MIND
1530 REM
1540 DATA DO YOU THINK IT'S NORMAL TO BE*
1550 DATA HOW LONG HAVE YOU BEEN*
1560 DATA DO YOU ENJOY BEING*
1570 REM
1580 DATA DO YOU THINK IT'S NORMAL TO BE*
1590 DATA HOW LONG HAVE YOU BEEN*
1600 DATA DO YOU ENJOY BEING*
1610 REM
1620 DATA WHO'S THE PATIENT AROUND HERE
1630 DATA WE WERE DISCUSSING YOU -- NOT ME
1640 DATA KEEP ME OUT OF THIS
1650 REM
1660 DATA WE ALL WANT*
1670 DATA WHAT WOULD IT MEAN TO YOU IF YOU GOT*
1680 DATA DOES EVERYONE WANT*
1690 REM
1700 DATA DOES THE QUESTION INTEREST YOU
1710 DATA WHY DO YOU ASK
1720 DATA I REFUSE TO ANSWER THAT
1730 REM
1740 DATA DO YOU KNOW HOW*
1750 DATA I SHOULD BE ASKING YOU HOW*
1760 DATA THAT'S A SILLY QUESTION FOR A START
1770 REM
1780 DATA WOULD IT HELP YOU TO KNOW WHO*
1790 DATA WHY DO YOU ASK WHO*
1800 DATA WHO INDEED
1810 REM
1820 DATA DO YOU NEED TO KNOW WHERE*
1830 DATA IS THAT A DUMB QUESTION OR IS IT A DU
MB QUESTION
1840 DATA WHAT WOULD IT MEAN TO YOU IF I TOLD Y
OU WHERE*
1850 REM
1860 DATA TIMES AND DATES DON'T BOTHER ME- PRAY
CONTINUE
1870 DATA WHEN WHAT
1880 DATA HOW SHOULD I KNOW WHEN*

1890 REM
1900 DATA HOW SHOULD I KNOW WHY*
1910 DATA DO YOUR FRIENDS KNOW WHY*
1920 DATA WHY DO YOU ASK
1930 REM
1940 DATA DO ANY OTHER REASONS SUGGEST THEMSELV
ES
1950 DATA IS THAT THE REAL REASON
1960 DATA ARE YOU SURE
1970 REM
1980 DATA PLEASE -- THERE'S NO NEED TO APOLOGIZ
E
1990 DATA APOLOGIES ARE NOT NEEDED
2000 DATA DON'T BE SO DEFENSIVE
2010 REM
2020 DATA DO YOU DREAM OFTEN
2030 DATA WHY DO YOU THINK YOU DREAM
2040 DATA WHAT IS SO IMPORTANT ABOUT DREAMS
2050 REM
2060 DATA HOW DO YOU DO -- PLEASE STATE YOUR PR
OBLEM
2070 DATA YOU ARE POLITE
2080 DATA OK -- I'VE GOT THE MESSAGE
2090 REM
2100 DATA HOW DO YOU DO -- PLEASE STATE YOUR PR
OBLEM
2110 DATA YOU ARE POLITE
2120 DATA OK -- I'VE GOT THE MESSAGE
2130 REM
2140 DATA YOU SEEM UNSURE
2150 DATA PLEASE BE MORE POSITIVE
2160 DATA BE MORE ASSERTIVE
2170 REM
2180 DATA ARE YOU SAYING 'NO' JUST TO BE NEGATI
VE
2190 DATA YOU ARE BEING RATHER NEGATIVE
2200 DATA WHY NO?
2210 REM
2220 DATA WHY ARE YOU CONCERNED ABOUT MY*
2230 DATA ARE YOU INTERESTED IN MY*
2240 DATA WHAT ABOUT YOUR OWN*
2250 REM
2260 DATA NAMES DATES DON'T BOTHER ME- PRAY CON
TINUE
2270 DATA WILL YOU PLEASE KEEP NAMES OUT OF IT
!!!
2280 DATA I HAVE NO USE FOR NAMES
2290 REM

2300 DATA "REALLY, ALWAYS?"
2310 DATA ISN'T 'ALWAYS' AN EXAGGERATION?
2320 DATA HOW OFTEN IS 'ALWAYS'
2330 REM
2340 DATA DO YOU THINK OFTEN
2350 DATA DO YOU THINK THINKING DAMAGES YOUR HE
ALTH
2360 DATA DID YOU THINK BEFORE YOU CAME TO SEE
ME
2370 REM
2380 DATA WHAT DOES THE SIMILARITY SUGGEST TO Y
OU
2390 DATA WHAT OTHER CONNECTIONS DO YOU SEE
2400 DATA WHAT RESEMBLANCE DO YOU SEE
2410 REM
2420 DATA YOU SEEM QUITE POSITIVE
2430 DATA ARE YOU SURE?
2440 DATA I SEE
2450 REM
2460 DATA WHY DO YOU BRING UP THE TOPIC OF FRIE
NDS?
2470 DATA DO YOU HAVE ANY FRIENDS
2480 DATA DO YOUR FRIENDS LIKE YOU
2490 REM
2500 DATA DO COMPUTERS WORRY YOU
2510 DATA ARE YOU THINKING ABOUT ME IN PARTICUL
AR?
2520 DATA WHY DO YOU MENTION COMPUTERS?
2530 REM
2540 DATA I'M NOT SURE I UNDERSTAND YOU FULLY
2550 DATA PLEASE ENLARGE
2560 DATA THIS IS MOST REVEALING -- PRAY CONTIN
UE
2570 REM *****

Life

The Game

During the winter of 1970, John Conway, then attending Cambridge University, created LIFE, a game whose simple rules produce effects more delightful and unexpected than any examination of the rules of the game could suggest. The world of LIFE-admirers remained small until *Scientific American* published an article on the game, written by Martin Gardner. Then LIFE broke out all over.

Without a computer the moves of the game are so tedious and slow to implement that the beauty of the unfolding patterns is lost. LIFE really is one of the true 'natural' computer games; it has deserved its vibrant success.

LIFE creates an evolving colony of cells on a grid; these are born and live and die in accordance with Conway's rules:

- Each cell on the grid has eight neighbours.
- Every cell with two or three neighbours survives to the next generation.
- A cell with only one neighbour dies.
- If there are three, and only three, neighbouring cells, a new cell is born.
- Any cell with four or more neighbours dies from overpopulation.

You can work out how the game proceeds by using a draughts board, placing some pieces on it, then checking each piece and the squares immediately around it to work out what will happen in the next generation. It is important to note that a particular cell is not changed immediately after it is examined, but only when the *whole* grid has been examined. The rules are applied all over the grid *simultaneously*.

There are two versions of the program. The first one generates the initial colony at random, and the second allows the player to enter the starting colony of his or her choice. We suggest you enter the 'automatic' version first and then, when you're familiar with how the process works, run the version (LIFE 2) that allows you to define your initial colony.

After you input the number of cells (in LIFE 2), you then enter the co-ordinates of each cell within the LIFE grid.

It is impossible to describe how effective the whole process is.

Reading this description will not prepare you for the pure magic of this evolution of Life.

The Programs

Program one

10 Title

20 Sets mode.

30 Sends action to the initialisation procedure.

40 Start of master REPEAT/UNTIL loop (terminated in line 70).

50 To the procedure to print out the present colony.

60 To the procedure which determines how the next colony will look.

70 Termination of master REPEAT/UNTIL loop.

90–180 Initialisation procedure:

100 Clears screen.

110 Turns off cursor.

120 Dimensions arrays to hold the plotted colony (A) and the colony which is amended (B) before being copied into the plotting colony.

130–160 Distribute the initial cells in the colony. The value one in an array tells the computer a cell is present; a zero equals an empty element of the grid. Line 150 copies the contents of each element of the A array into the B array, so they start the run holding an identical colony.

200–300 The procedure which actually prints out the colony:

210 Moves the print position to the top left-hand corner of the screen; prints two blank lines.

220 Sets the counter Z to zero. This counter controls the number of times the random notes sound just before the number of the generation is changed. The word 'Generation', and the value assigned to the variable GENERATION are printed in red (CHR\$(129)). Lines 260 to 290 run through loops from two to 14 (leaving out the elements on the 'Frame' of the grid). Line 270 copies the elements from the B array (which has been updated) into the A array, and line 280 prints a double space if the element is zero, and a space followed by a green asterisk if the element is one.

320–470 This is the procedure which updates the cell colony:

330 Adds one to the generation count.

340 Starts both the X and Y loops (terminated in line 460).

350 Sets the variable C, which will count the number of neighbouring cells, to zero.

360–430 Check the cells surrounding the cell under examination, looking at the elements in the A array.

440–450 Modify the contents of the B array in light of the value which C has obtained, and in light of the contents of the A array. If A equals one (that is, a cell is present at this location) and there are

not two or three neighbours (which means there must be none, one, or more than three), the cell is 'killed' (that is, the equivalent element in the B array is set to zero). If A equals zero (that is, there is no cell present at this location), and there are three surrounding cells (ie. C equals three), then a cell is born (the equivalent element in the B array is set to one).

Program two

There are many similarities between the first and second programs. The major REPEAT/UNTIL loop is identical, and the procedures which print the colony and update the generations are also identical. The difference between the two programs lies within lines 100 and 270, where you can enter your own colony. You'll find that if you enter a colony which forms a balanced pattern, it may well evolve more interestingly than a colony of random cells. You will find certain patterns that cycle, and others that move across the screen, keeping each cell in a fixed relationship with the other cells in the pattern.

150 Allows you to enter the number of starting cells you desire.

Line 160 rejects an answer less than zero.

180-240 This loop (FOR entry = 1 TO start) accepts your input as X and Y (line 190) and checks to see if the cell lies within the grid (line 200) and ensures that the cell is not already occupied (line 210). Line 220 sets that element in the A array to one, and line 230 copies this into the B array.

It is worth keeping track of your entry numbers so that if you discover a particularly attractive pattern you can always repeat it to impress your friends.

The Listings

```
10 REM *** LIFE ***
20 MODE 7
30 PROCinitialise
40 REPEAT
50 PROCprint_colony
60 PROCgeneration_update
70 UNTIL FALSE
80 REM *****
90 DEF PROCinitialise
100 CLS
110 VDU 23;8202;0;0;0
120 DIM A(15,15),B(15,15)
130 FOR X=2 TO 14:FOR Y=2 TO 14
140 IF RND(1)>.35 A(X,Y)=1
```

```

150 B(X,Y)=A(X,Y)
160 NEXT:NEXT
170 GENERATION=0
180 ENDPROC
190 REM *****
200 DEF PROCprint_colony
210 PRINT CHR$(30)';
220 Z=0:REPEAT
230 SOUND1,-15,RND(100)+30,1:SOUND1,-7,59,1
240 Z=Z+1:UNTIL Z=3
250 PRINT CHR$(129);" Generation: ";
GENERATION';
260 FOR X=2 TO 14:FOR Y=2 TO 14
270 A(X,Y)=B(X,Y)
280 IF A(X,Y)=0 PRINT " "; ELSE PRINT CHR$(1
30);"*";
290 NEXT:PRINT:NEXT
300 ENDPROC
310 REM *****
320 DEF PROCgeneration_update
330 GENERATION=GENERATION+1
340 FOR X=2 TO 14:FOR Y=2 TO 14
350 C=0
360 IF A(X-1,Y-1)=1 C=C+1
370 IF A(X-1,Y)=1 C=C+1
380 IF A(X-1,Y+1)=1 C=C+1
390 IF A(X,Y-1)=1 C=C+1
400 IF A(X,Y+1)=1 C=C+1
410 IF A(X+1,Y-1)=1 C=C+1
420 IF A(X+1,Y)=1 C=C+1
430 IF A(X+1,Y+1)=1 C=C+1
440 IF A(X,Y)=1 AND C<>2 AND C<>3 B(X,Y)=0
450 IF A(X,Y)=0 AND C=3 B(X,Y)=1
460 NEXT:NEXT
470 ENDPROC

```

```

10 REM *** LIFE 2 ***
20 REM ALLOWS USER TO DEFINE STARTING COLONY
30 MODE 7
40 PROCinitialise
50 REPEAT
60 PROCprint_colony

```

```

70 PROCgeneration_update
80 UNTIL FALSE
90 REM *****
100 DEF PROCinitialise
110 CLS
120 VDU 23;8202;0;0;0
130 DIM A(15,15),B(15,15)
140 PRINT''''''
150 INPUT"HOW MANY CELLS TO START? "start
160 IF start<=0 THEN 150
170 FOR entry=1 TO start
180 PRINT"Please enter cell number ";entry
190 INPUT X,Y
200 IF X*Y=0 OR X>14 OR Y>14 PRINT CHR$(130);
X;" ";Y;" IS OUT OF RANGE":GOTO 180
210 IF A(X,Y)<>0 PRINT CHR$(129);"THAT CELL
IS ALREADY OCCUPIED":GOTO 180
220 A(X,Y)=1
230 B(X,Y)=A(X,Y)
240 NEXT entry
250 CLS
260 GENERATION=0
270 ENDPROC
280 REM *****
290 DEF PROCprint_colony
300 PRINT CHR$30''
310 Z=0:REPEAT
320 SOUND1,-15,RND(100)+30,1:SOUND1,-7,59,1
330 Z=Z+1:UNTIL Z=3
340 PRINT CHR$(129);" Generation: ";
GENERATION''
350 FOR X=1 TO 14:FOR Y=1 TO 14
360 A(X,Y)=B(X,Y)
370 IF A(X,Y)=0 PRINT " "; ELSE PRINT CHR$(1
30);"*";
380 NEXT:PRINT:NEXT
390 ENDPROC
400 REM *****
410 DEF PROCgeneration_update
420 GENERATION=GENERATION+1
430 FOR X=2 TO 14:FOR Y=2 TO 14
440 C=0
450 IF A(X-1,Y-1)=1 C=C+1
460 IF A(X-1,Y)=1 C=C+1
470 IF A(X-1,Y+1)=1 C=C+1
480 IF A(X,Y-1)=1 C=C+1
490 IF A(X,Y+1)=1 C=C+1
500 IF A(X+1,Y-1)=1 C=C+1

```



```
510 IF A(X+1,Y)=1 C=C+1
520 IF A(X+1,Y+1)=1 C=C+1
530 IF A(X,Y)=1 AND C<>2 AND C<>3 B(X,Y)=0
540 IF A(X,Y)=0 AND C=3 B(X,Y)=1
550 NEXT:NEXT
560 ENDPROC
```

Models A and B

Fairground Organ

The Game

In this game, your interaction with the computer consists of typing in RUN, pressing RETURN, then sitting back to listen as the computer composes a passable imitation of music on a fairground steam organ. It changes speed from 'verse' to 'verse' of its little tune, and obligingly prints its speed (Moderato or Allegro) on the screen for you as it does so.

The Program

The heart of the program is the 'bass sequences' which play twelve-bar blues in the key of C, and from time to time generate notes which harmonise, more or less, with the bass sequence being played.

10 Title.

20 Clears the screen (can be changed to MODE 7).

30 Dimensions the arrays which hold the notes to go with 'chord' and 'fchord'.

50-360 This is the master REPEAT/UNTIL loop.

60, 100, 150, 190, 230, 280 These change the colour of the words 'Fairground organ' and 'Moderato' or 'Allegro'.

70-90 Call the 'chord' procedure four times.

110-130 Call the 'fchord' and 'fnote' procedure four times.

160-180 Call the 'chord' procedure a further four times.

200-220 Call the 'gchord' procedure twice.

240-260 Call the 'fchord' and 'fnote' procedures twice.

290-310 Call the 'chord' procedure twice.

320-350 Play the final note of the 'verse', and pause for an appropriate time (using the zero in the second SOUND statement - line 330 - to effectively turn off the sound for six times the value of Z, the variable which controls the tempo). Line 350 changes the speed from 'verse' to 'verse'.

380-410 This procedure controls the display.

430-510 This procedure plays the bass run (C, E, G, E), calling notes three times, and choosing a volume setting for the middle two notes at random (thus giving a different stress to each bar, which stops it from being too mechanical and monotonous). Notes one and four (C and E) are chosen at random from different octaves, again to increase variety.

530-580 This procedure plays the 'fchord' (F, A, C, A) in rigid time and with a set volume. This stops the whole piece from degenerating into apparent randomness (which tends to occur if every 'chord' has the same waywardness that 'cchord' enjoys).

600-650 This procedure plays the 'gchord' (G, B, D, B), again in strict time, at a set volume.

670-690 This procedure chooses notes at random (by selecting random elements of the array A) which go with 'cchord'.

720-840 The initialisation procedure fills the arrays with values which correspond to appropriate notes.

860-880 Chooses 'fnote's to go with 'fchord'.

Suggestions for improvement

- Make better use of the display (perhaps to draw the notes as the computer plays them).
- Change the notes the computer has to choose from.
- Allow 'fchord' and 'gchord' more variety, and add a routine to change the volume within a verse or between verses.

The Listing

```
10REM** Fairground Organ **
20CLS
30 DIMA(8),B(4)
40PROCinitialise
50REPEAT
60PROCdisplay
70FORG=1TO4
80PROCcchord
90NEXT G
100PROCdisplay
110FORG=1TO4
120PROCfnote
130PROCfchord
140NEXTG
150PROCdisplay
160FORG=1TO4
170PROCcchord
180NEXTG
190PROCdisplay
200FORG=1TO2
210PROCgchord
220NEXTG
230PROCdisplay
240FORG=1TO2
250PROCfchord
260PROCcnote
```

```

270NEXTG
280PROCdisplay
290FORG=1TO2
300PROCcchord
310NEXTG
320SOUND 2,-10,53,4*Z
330SOUND 2,0,1,6*Z
340PROCdisplay
350Z=RND(2)+2
360UNTIL FALSE
370REM*****
380DEF PROCdisplay
390PRINT TAB(0,3);CHR$(128+RND(6));"**
Fairground";CHR$(128+RND(6));"organ **"
400 IF Z=3 PRINT TAB(8,6);CHR$(128+RND(6));"
Allegro" ELSE PRINT TAB(7,6);CHR$(128+RND(6));"
Moderato"
410ENDPROC
420END
430REM*****
440DEF PROCcchord
450IF RND(1)>.5 SOUND 2,-15,53,Z ELSE SOUND 2
,-15,5,Z
460 PROCcnote
470 IF RND(1)>.5 SOUND 2,-15,69,Z ELSESOUND 2
,-1,69,Z
480 IF RND(1)>.5 SOUND 2,-15,81,Z ELSESOUND 2
,-1,81,Z
490PROCcnote
500 IF RND(1)>.5 SOUND 2,-15,69,Z ELSESOUND 2
,-15,21,Z
510PROCcnote
520 ENDPROC
530REM*****
540DEF PROCfchord
550 SOUND 2,-15,73,Z
560 SOUND 2,-15,89,Z
570 SOUND 2,-15,101,Z
580 SOUND 2,-15,89,Z
590ENDPROC
600REM*****
610DEF PROCgchord
620 SOUND 2,-15,81,Z
630 SOUND 2,-15,97,Z
640 SOUND 2,-15,109,Z
650 SOUND 2,-15,97,Z
660ENDPROC
670REM*****

```

```

680DEF PROCcnote
690SOUND3,-13,A(RND(8)),Z*2:SOUND1,-4,A(RND(8
),Z
700SOUND1,-7,A(RND(8)),Z
710ENDPROC
720REM*****
730DEF PROCinitialise
740Z=3
750 FORB=1TO8
760 READC
770 A(B)=C
780 NEXTB
790DATA149,165,177,5,245,233,117,33
800FORB=1TO4
810READC
820B(B)=C
830NEXTB
840DATA245,233,217,25
850ENDPROC
860REM*****
870DEF PROCfnote
880SOUND3,-12,B(RND(4)),Z:SOUND1,-4,B(RND(4)
),2*Z
890ENDPROC

```

Model B

Space Storm

The Game

This program is a simplified version of the popular arcade game. You steer a space-ship, roaming around in deep space, when you encounter an asteroid storm.

All you have to do to destroy an asteroid is crash into it, or let it crash into you. There is also a 'poisonous gas' at the top left of the screen which does you no harm but destroys asteroids at will. A special feature of this game is that asteroids will occasionally merge into one if they meet.

The object of the game is to destroy as many of the asteroids as you can in the shortest possible time. The time in seconds is shown ticking away in the top left corner of the screen. Pressing 'ESCAPE' gives a special effect, so do not try to cheat.

You alter the number of asteroids you battle against in line 10. Replace the '2' with one less than the number of asteroids you feel fit to tackle. The speed of the game can be altered in line 40; increasing DIF% from 0 slows down the game. The speed is largely determined by the number of asteroids used.

The controls you have for your ship are:

Travel in direction you are pointing in: Space bar.

Rotate clockwise: Colon key. (Asterisk key.)

Rotate counter-clockwise: Slash key. (Question mark.)

The Program

10 Initialises the number of asteroids to be used.

20 Dimensions an array to hold the X, Y positions of each of the asteroids, and their velocities in each direction.

30 Sets up an 'ESCAPE' trap.

40 Initialises the difficulty factor of the game.

50 Sets the computer into Mode 4.

60 Calls PROCdefine, which defines the asteroid shape, and eight characters which are the space-ship pointing in each of eight directions.

70 Calls PROCinit, which does the general initialisation required.

80 Starts the clock for the game.

90 This line of asteroids marks the start of the main game loop.

100 Starts the main game loop.

110 Prints the elapsed time near the top of the screen.

120 Saves the TIME.

130 And waits for DIF% centiseconds. As DIF% is usually 0 for experienced players, you can delete this section of two lines and line 40.

140 Uses INKEY with a negative number to sense whether the space bar is pressed down. If so, places a space at the space-ship's current position, and calls PROCthrust, which does just that.

150 Checks for the 'slash' key being held down. If it is, increments the counter SH%, which gives the direction in which the space-ship is pointing. This value is used MOD 8 since there are only eight possible directions.

160 Checks for the colon key. SH% should then be decremented; but, if it is, it could turn SH% into a negative number, so 7 is added and the value taken MOD 8 again, to ensure SH% remains positive.

170 Places the space-ship at the required position, using SH% to determine which of the eight space-ship shapes to print.

180 Calls PROCmove_ast, which moves the asteroids on the screen.

190 Ends the game if all the asteroids have been destroyed. This is checked by a call to FNall_destroyed.

200 A line of asterisks to delimit the end of the main loop.

210 Calls PROCfinished, which prints a progress report and restores the cursor.

220 The program ends.

240 Starts the definition of PROCdefine.

260 Starts a loop through all the characters to be redefined.

270 Starts off the VDU statement required to redefine the character.

280 Starts a loop through each of the eight rows required for each character.

290 Reads the next character string from the data statements. This will be a two-digit hexadecimal number.

300 Sends the byte to the redefining routine. The EVAL function converts the hex string to a decimal number.

310 Ends loop.

320 Ends loop.

330 Ends PROCdefine.

350 Start of nine DATA statements to define each of the required characters.

460 Starts the definition of PROCthrust.

470 If the ship is pointing left, moves the ship left.

480 If the ship is pointing left and up, moves the ship in that direction.

490 If the ship is pointing straight up, moves it one space up.

500 If the ship is pointing right and up, moves it in that direction.

510 If the ship is pointing to the right, moves it one space right.

520 If the ship is pointing down and to the right, moves it in that direction.

530 If the ship is pointing straight down, moves it one space down.
540 If the ship is pointing down and to the left, moves it in that direction.
550 If the ship has moved off the left-hand side of the screen, moves it to the right edge of the screen.
560 If the ship has moved off the top of the screen, moves it to the bottom.
570 Takes the MOD value of the x co-ordinate of the ship . . .
580 and of the y co-ordinate.
590 Ends PROCthrust.

610 Starts the definition of FNreadch. This function is directly taken from the User Guide, and an explanation of it may be found in its chapter on *FX calls.

730 Starts the definition of PROCinit. This routine initialises all the variables required, and sets up the display.
740 Restores the normal display colours – black background, white foreground.
750 Chooses a yellow foreground and a red background.
760 Sets the starting x co-ordinate of the ship.
770 Sets the starting y co-ordinate of the ship.
780 Sets the starting direction for the ship.
790 Places your ship on the screen.
800 Starts a loop through all the memory locations on one line of the screen.
810 Places a yellow block on the top line . . .
820 and on the bottom.
830 Ends the loop. The end result of all this is to fill the top and bottom lines of the display with yellow blocks.
840 For each of the top 31 lines of the display . . .
850 prints a white block at the beginning and end of every line and . .
860 ends the loop.
870 Turns off the cursor.
880 Starts a loop for each of the available asteroids.
890 Chooses a random x co-ordinate for each asteroid . . .
900 and a random y co-ordinate.
910 Chooses a random velocity in each direction for each of the asteroids. This can be -1, 0 or +1.
920 See above.
930 Sets the flag 'is asteroid not shot down' to TRUE for each asteroid.
940 Places each asteroid on the screen.
950 Ends loop.
960 Sets the print field width to 3.
970 Lets the cursor control keys give true values for GETting and INKEYing.
980 Ends PROCinit.

1000 Starts the definition of PROCmove_ast. This procedure moves all the asteroids according to their current velocities.
1010 Sets the LOCAL variables required.
1020 Starts a loop through each of the available asteroids.
1030 Gets the x co-ordinate of the current asteroid.
1040 Gets the y co-ordinate of the current asteroid.
1050 If the asteroid is meant to be on the screen, and the place where it should be does not contain an asteroid, resets the flag.
1060 If the asteroid has still survived, replaces it with a space.
1070 Increments the x co-ordinate according to the x velocity.
1080 Increments the y co-ordinate according to the y velocity.
1090 If the asteroid has moved off the screen, places it at the bottom.
1100 If the asteroid has moved off the left edge of the screen, puts it on the right edge.
1110 Takes the MOD value, in case the asteroid has moved off the bottom.
1120 Takes the MOD value, in case the asteroid has moved off the right-hand edge of the screen.
1130 If the asteroid is still meant to be on the screen, puts an asteroid at the relevant space.
1140 Resets the array with the new co-ordinate . . .
1150 does the same for the y co-ordinate.
1160 Goes back for the reset of the asteroids.
1170 Ends PROCmove_ast.
1190 Starts the definition of FNall_destroyed. This function checks to see if there are any 'un-destroyed asteroids' around.
1200 Sets the relevant LOCAL variables.
1210 Sets the flag for 'all destroyed' to TRUE.
1220 Starts a loop through all the asteroids.
1230 If the current asteroid is visible, sets the flag to FALSE.
1240 Ends the loop.
1250 Ends the loop with the flag.

1270 Starts the definition of PROCfinished.
1280 Clears the screen.
1290 Gives the screen a blue foreground and a red background (very difficult to read).
1300 Starts the congratulatory message.
1310 Carries on with the above.
1320 And finishes it.
1330 Gives you back the cursor.
1340 Ends PROCfinished.

1370 Puts the machine in Mode 5.
1380 Prints 'CHEAT' in the middle of the screen.
1390 Resets Time.
1400 Sets the flash rate for the flashing colours at very high.
1410 Ditto.

- 1420 Makes the screen flash black/white.
- 1430 Ends the effect after a predetermined time.
- 1440 Returns to the start of the program.

Suggestions for improvement

- Sound effects should be your first priority. The cues for sound would be given at a collision and when the game is over.
- You could have more than one shape for the asteroids (although most people playing this version do not notice or even care that they are all identical).
- A high-score feature, or – better still – a score ladder, as in the arcade versions, would increase the competitive element.
- An option to end a game and also an option to begin a new game.

The Listing

```

10 AS%=5
20 DIM AST%(AS%,4)
30 ON ERROR GOTO 1360
40 DIF%=1
50 MODE 4
60 PROCdefine
70 PROCinit
80 TIME=0
90 REM *****
100 REPEAT
110 PRINT TAB(2,2),TIME DIV 100
120 TI%=TIME
130 REPEAT UNTIL TIME>DIF%+TI%
140 IF INKEY(-99) THEN VDU 31,X%+1,Y%+1,32:
PROCthrust
150 IF INKEY(-105) THEN SH%=(SH%+1) MOD 8
160 IF INKEY(-73) THEN SH%=(SH%+7) MOD 8
170 VDU 31,X%+1,Y%+1,SH%+224
180 PROCmove_ast
190 UNTIL FNall_destroyed
200 REM *****
210 PROCfinished
220 END
230 REM *****
240 DEF PROCdefine
250 LOCAL X%,Y%,A$
260 FOR X%=224 TO 233
270 VDU 23,X%
280 FOR Y%=0 TO 7
290 READ A$
300 VDU EVAL("&"+A$)
310 NEXT Y%
320 NEXT X%

```

```

330 ENDPROC
340 REM *****
350 DATA 06,1C,7C,F8,F8,7C,1C,06
360 DATA 00,10,30,38,7E,7C,F0,C0
370 DATA 00,81,E7,7E,7E,3C,3C,18
380 DATA 00,08,0C,1C,7E,3E,0F,03
390 DATA 60,38,3E,1F,1F,3E,38,60
400 DATA 03,0F,3E,7E,1C,0C,08,00
410 DATA 18,3C,3C,7E,7E,E7,81,00
420 DATA C0,F0,7C,7E,38,30,10,00
430 DATA 30,1C,7E,F8,3E,7E,7E,18
440 DATA FF,FF,FF,FF,FF,FF,FF,FF
450 REM *****
460 DEF PROCthrust
470 IF SH%=0 THEN X%=X%-1
480 IF SH%=1 THEN X%=X%-1:Y%=Y%+1
490 IF SH%=2 THEN Y%=Y%+1
500 IF SH%=3 THEN X%=X%+1:Y%=Y%+1
510 IF SH%=4 THEN X%=X%+1
520 IF SH%=5 THEN X%=X%+1:Y%=Y%-1
530 IF SH%=6 THEN Y%=Y%-1
540 IF SH%=7 THEN X%=X%-1:Y%=Y%-1
550 IF SGN(X%)=-1 THEN X%=38+X%
560 IF SGN(Y%)=-1 THEN Y%=30+Y%
570 X%=(X% MOD 38)
580 Y%=(Y% MOD 30)
590 ENDPROC
600 REM *****
610 DEF FNread(X%,Y%)
620 LOCAL A%,LX%,LY%,C%
630 LX%=POS
640 LY%=VPOS
650 VDU 31,X%,Y%
660 A%=135
670 C%USR(&FFF4)
680 C%=C% AND &FFFF
690 C%=C% DIV &100
700 VDU 31,LX%,LY%
710=(C% MOD 32)+224
720 REM *****
730 DEF PROCinit
740 VDU 20
750 VDU 19,1,3,0,0,0,19,0,1,0,0,0
760 X%=20
770 Y%=20
780 SH%=1
790 VDU 31,X%+1,Y%+1,SH%+224
800 FOR R%=0 TO 319

```

```

810 ?(HIMEM+R%)=255
820 ?(HIMEM+R%+9920)=255
830 NEXT R%
840 FOR R%=0 TO 30
850 VDU 31,39,R%,233,233
860 NEXT R%
870 VDU 23;8202;0;0;0;
880 FOR T%=0 TO AS%
890 AST%(T%,0)=RND(38)-1
900 AST%(T%,1)=RND(30)-1
910 AST%(T%,2)=RND(3)-2
920 AST%(T%,3)=RND(3)-2
930 AST%(T%,4)=TRUE
940 VDU 31,AST%(T%,0)+1,AST%(T%,1)+1,232
950 NEXT T%
960 @%=3
970 REM
980 ENDPROC
990 REM *****
1000 DEF PROCmove_ast
1010 LOCAL T%,X%,Y%
1020 FOR T%=0 TO AS%
1030 X%=AST%(T%,0)
1040 Y%=AST%(T%,1)
1050 IF AST%(T%,4) AND FNread(X%+1,Y%+1)<>232
THEN AST%(T%,4)=FALSE
1060 IF AST%(T%,4) THEN VDU 31,X%+1,Y%+1,32
1070 X%=X%+AST%(T%,2)
1080 Y%=Y%+AST%(T%,3)
1090 IF SGN(X%)=-1 THEN X%=38+X%
1100 IF SGN(Y%)=-1 THEN Y%=30+Y%
1110 X%=X% MOD 38
1120 Y%=Y% MOD 30
1130 IF AST%(T%,4) THEN VDU 31,X%+1,Y%+1,232
1140 AST%(T%,0)=X%
1150 AST%(T%,1)=Y%
1160 NEXT T%
1170 ENDPROC
1180 REM *****
1190 DEF FNall_destroyed
1200 LOCAL A%,T%
1210 A%=TRUE
1220 FOR T%=0 TO AS%
1230 IF AST%(T%,4) THEN A%=FALSE
1240 NEXT T%
1250=A%
1260 REM *****
1270 DEF PROCfinished
1280 CLS

```

```
1290 VDU 19,1,4,0,0,0,19,0,1,0,0,0
1300 PRINT '''"You have succeeded in
destroying"
1310 PRINT "all the asteroids in ";TIME DIV 10
0;" seconds"
1320 PRINT '''Well done. '''
1330 VDU 23,0,10,64,0;0;0;0;:*FX 15
1340 ENDPROC
1350 REM *****
1360 REM *** ON ERROR SUBROUTINE ***
1370 MODE 5
1380 PRINT TAB(2,16);"C H E A T ! ! !"
1390 TIME=0
1400 *FX 9,2
1410 *FX 10,2
1420 VDU 19,3,8,0,0,0,19,0,15,0,0,0
1430 REPEAT UNTIL TIME>300
1440 GOTO 40
```

Outlaw

The Game

This game, based on a program written by Alastair Gourlay, puts you in the role of Sheriff. You have to clean up the town by killing off all the outlaws so that decent, law-abiding folk (such as you and me) can walk the streets free of fear. The outlaws are, we have heard, 'plenty mean'. The moment you spot one, you must shoot him very, very dead . . . or he will draw his six-shooter and gun you down. It is fairly easy to shoot an outlaw: you just hit any key on your trusty computer when you are told that an outlaw has been spotted. But you must be very quick on the draw, as you'll discover when you run this game. You start a game with between 11 and 20 outlaws to 'clean up' and you win the game only if you manage to get them all.

The game is essentially a reaction test.

The Program

10-20 Title.

30 Sets the mode.

40 Decides how many outlaws are in town and sets the variable A equal to this number (which is between 11 and 20)

50 Prints title.

60 Tells you how many outlaws are left.

70 This line ensures that around 70% of the time you will be told the streets are empty.

90-110 A random delay before the streets are surveyed again.

130 Prints out a warning sign.

140 Clears the buffer, to await the 'gun shot'.

150 Waits for a time (which gets shorter as the game progresses) for you to react.

150 If a shot has been fired (ie. a key has been touched) action is sent to the routine from line 240 which guns down the outlaw.

160-230 This is the 'you have been shot' routine which ends the game.

240 Clears the buffer.

250-310 'Fires' the gun, using the J loop and the two sound lines (270 and 280).

320 Determines if you hit the outlaw or not.

330-340 You did not.

350-390 Well done. You did.

400-450 The congratulatory message if you kill all the outlaws.

Suggestions for improvement

- Note that if you hold down a key long enough, the Sheriff always wins. You could perhaps try to modify the program to get around this.
- Cut off the ending message after a few seconds and give an option to restart the game. It is best to put the computer into Mode 7 at the end of the current game.

The Listing

```
10 REM OUTLAW
20 REM ALASTAIR GOURLAY/TIM HARTNELL
30 MODE7
40 A=RND(10)+10
50 PRINTTAB(12,3);CHR$(128+RND(5))"OUTLAW"
60 PRINTTAB(0,16);CHR$(128+RND(5))"There are
";A;" outlaws left in town "
70 IF RND(10)<4 THEN 130
80 PRINTTAB(3,8);CHR$(128+RND(5))"The streets
are empty... "
90 T=TIME
100 K=RND(250)+50
110 REPEAT UNTIL TIME-T=K
120 GOTO 50
130 PRINTTAB(3,8);CHR$(128+RND(5))"Look out, t
here's an outlaw! "
140 *FX 15,0
150 IF INKEY$(20+A)<>" " THEN 240
160 REPEAT
170 PRINT"CHR$(129)"+++++++ Too slow!! +++
+++++++"
180 PRINT"CHR$(128+RND(5))"_____ The outlaw
got you! _____"
190 T=TIME
200 REPEAT UNTIL TIME-T=10
210 K=RND(4)-1
220 SOUND K,-15,RND(100)+128,K+1
230 UNTIL FALSE
240 *FX 15,0
250 FOR J= 1 TO 60 STEP6
260 PRINTTAB(3,8);CHR$(128+RND(5))"*****
** BANG *****"
270 SOUND 1,-15+J/5,1,1
280 SOUND 2,-15+J/5,2,1
290 NEXT
300 T=TIME
```

```

310 REPEAT UNTIL TIME-T=40
320 IF RND(10)>3 THEN 350
330 PRINTTAB(3,8);CHR$(128+RND(5))"*****
You missed! *****"
340 GOTO 150
350 PRINTTAB(3,8);CHR$(128+RND(5))"*****
You got him! *****"
360 A=A-1
370 T=TIME
380 REPEAT UNTIL TIME-T=300
390 IF A>1 THEN 50
400 REPEAT
410 PRINTTAB(3,8);CHR$(128+RND(5))"Well done,
Sheriff,you've cleaned"
420 PRINTTAB(6,10);CHR$(128+RND(5))"the town o
f outlaws"
430 SOUND 1,-15,RND(20)+30,RND(4)
440 SOUND 2,-15,RND(20)+30,RND(4)
450 UNTIL FALSE

```


Model B

3D Super Plot

The Game

This program allows you to build up images of three-dimensional objects and then view the finished drawing from any position in three-dimensional space. This has the effect of variously rotating and changing the size of the drawing. You can also edit the drawings, in a manner similar to BASIC editing, to correct errors or to enhance an image. 'Sub-drawings', such as a cube, can be stored under user-definable keys, to allow easy repetition of parts of an object.

When you run the program, the prompt 'Enter the X, Y and Z co-ordinates' is printed near the bottom of the screen. The portion below this prompt is not available for drawing. The program is asking you to enter the co-ordinates of a point somewhere in three-dimensional space. This point will then be used as the viewpoint from which a drawing is constructed.

In response to the prompt, enter three numbers separated by commas. The numbers that you enter depend on the drawing you intend to construct, but usually '100, 200, 300' is a suitable viewpoint. For this reason, it has been stored under function key 9. Thus a simpler method of answering the prompt is to simply press function key 9.

After you answer the first question, the program asks you to 'Enter 'D' or 'E' (Draw or Edit)'. For the moment, press 'E'. This puts you in 'edit mode', where drawings can be constructed and edited. Entering 'D' would have put you in 'draw mode', which allows you to view the most recent drawing from the chosen viewpoint. However, as the program does not have a drawing stored in it at the moment, you will get no joy from this option. In 'edit mode' the program responds to the keys 'U, D, L, R, F, B, S, O, P' and 'Return'. Pressing 'Return' brings you back to the initial prompt and stores the current drawing. The first six keys in the above list draw lines in the following directions:

- 'U' up
- 'D' down
- 'L' left
- 'R' right
- 'F' forwards
- 'B' backwards

You can try experimenting with these keys in various combinations. If you get confused, just press 'Return'. (It is worth commenting that if you choose to edit from a 'negative viewpoint' (eg. -100, -200, 300), the above keys will work in the opposite sense, since the 'F' key adds a certain amount to one of the three co-ordinates. From a negative viewpoint, an addition will result in a line going away from the user. The implication of this is to avoid editing a drawing from a negative viewpoint).

If, after pressing 'Return', you still wish to use some of the parts of your previous drawings, you can use the 'cursor keys' and the 'copy' key to copy parts of the previous drawing, since all key presses also appear printed on the screen.

The 'S' key is used to change the size of the lines drawn using the above keys. The initial, default, size is '8'. To change size, press 'S' and then a number key from 1 to 9. Thus 'S4' will halve the size of the lines drawn.

The 'O' and 'P' keys are used to change the kind of lines drawn. Normally, solid lines are drawn, but by pressing 'P' these can be changed to dotted lines. Similarly, pressing 'O' will return the program to drawing solid lines. These dotted lines are left out when the drawing is drawn using the 'D' option in answer to the second prompt; so, by judicious use of these keys, you can draw pictures which do not 'join up' all over. The 'etch-a-sketch' program in the Welcome Pack uses a similar method to allow you to move to another part of the screen without leaving a trace.

That covers the edit mode. To prepare a picture for the 'draw mode' to act upon, proceed as follows: Press 'Return' to return you to the initial prompt, and enter suitable co-ordinates again. Function key 9 is still a suitable viewpoint. After pressing 'D' in response to the second prompt, the program will draw your picture from the chosen viewpoint, which is displayed at the top left of the screen; but it will not draw any of the dotted lines.

When it has finished, it waits for you to press one of the keys 'U, D, L, R, F, B' or 'Return'. 'Return' returns you to the first prompt as before, and the other keys change the viewpoint by 80 units in the direction they indicate. Thus repeatedly pressing 'L' will move the viewpoint to the left, and so seem to rotate the drawing. You can have great fun using this mode.

In response to the first prompt, you can also press function key 8 as the new viewpoint. This deposits '0, 0, 0' as the new viewpoint. In this case, the program inserts random values for the three co-ordinates, to allow special effects.

To store a section of a picture under a function key, it is necessary to 'escape' into the program and type something of the form `"*KEY O ' ' "`, inserting within the quotes the section you want to store. This can be edited down from the screen display.

The Program

10–150 This section of the program initialises some variables: *SP* is a pointer into the string array which stores the current drawing. That is, it contains a number equal to the subscript of the string array which is currently being processed.

mode is a variable which will have a value of 0 for a model B machine and 4 for a model A machine, since it governs the screen mode the program will run in.

lines is a variable containing the number of lines per page in the current screen mode, selected by 'mode'. This will almost always have a value of 32.

columns is a variable containing the number of characters per line in the current screen mode. This will be either 40 or 800. Thus to run the program in a model A machine, change this variable and 'mode' in line 20.

foreground_number is the number of the normal foreground colour. Thus for Modes 0 and 4 it will be 1, since the normal foreground colour is 1.

background_colour is the number governing the colour the background of the screen will be. I have chosen cyan, indicated by the 6.

foreground_colour is the number governing the colour the foreground of the screen will be. I have chosen red, indicated by the 1.

80, 90 Define the two user-defined keys used by the program.

100, 110 Declare how many key presses the drawing may be made up of, and then dimension a string array *S\$* to store all the key presses. It is this array that the variable *SP* is pointing into.

120 Puts the computer into the screen mode selected in line 20.

130 The call to **PROCSPLIT** here calls a sub-program which partitions the screen into the graphics and text windows, and moves the graphics origin into the centre of the graphics window.

140, 150 Change the screen colours according to the previously selected colours.

160–200 This section is the main program – all five lines of it. The program repeatedly calls **PROCGET**, which presents the user with the various prompts outlined above and extracts answers to the questions. Depending on the outcome of the second prompt, the program calls **PROCDRAW** or **PROCEDIT**.

The **CLG** statement clears the graphics window between Mode switches.

210–250 These lines make up **PROCSPLIT**. They first define a graphics window and then relocate the graphics origin to the bottom left of the window. A screen window is defined.

260–330 These lines make up **PROCACT**. *U*, *V* and *W* are variously the current viewpoint and the current plotting position. Thus this

routine decrements or increments these variables according to A\$. The change is governed by the variable G, which is usually the size of lines drawn in the 'edit mode'.

340-360 These lines make up PROCINIT. The variables U, V and W are initialised to zero for the edit routine and the draw routine.

370-430 These lines make up PROCVIEW. PROCVIEW sets up the variables used by the plotting routine, according to the current viewpoint (A, B and C). In addition, L, M and N are made equal to the current view position. The next routine, PROCPLT3, uses all these variables.

440-500 These lines make up PROCPLT3. This routine is a three-dimensional equivalent of the standard PLOT statement. Thus K is exactly equal to the first argument of the PLOT statement. The operation of this routine is complex, but it depends on the constants defined in the previous procedure.

510-590 These lines make up PROCGET. This procedure presents the user with the prompts outlined above, and inserts the random viewpoints if required.

600-790 These lines make up PROCPICTURE. This procedure is the main working section of the program, since it is the portion used by both draw and edit modes to construct the picture.

U, V and W are defined to be LOCAL, since these variables are also used outside this procedure. A\$ and G are defined as LOCAL also, because the value of A\$ has to be preserved, as does G.

PROCINIT is called in line 620 to zero the three LOCAL variables U, V and W. Then PROCVIEW is called, to initialise the constants required by PROCPLT3. Then the program repeatedly gets key presses from FNGET, and draws the necessary lines.

The value of MOV defines whether dotted lines are to be drawn or left out, depending on whether the procedure is called from the DRAW or EDIT sections of the program. This section of the program is fairly simple, since it relies almost exclusively on other procedures and a function.

The point to bear in mind when reading this section is that INP is a variable which is true if the procedure was called from the EDIT section, and false otherwise.

800-870 These lines make up PROCSIZE. This procedure is called whenever 'S' is encountered in the list of characters which define a drawing. Only if INP is true is the numeral following S printed out.

880-890 These lines make up the function FNGET. This function chooses whether to get input from the keyboard or from the string array S\$.

900-940 These lines make up PROCEDIT. This routine simply calls

PROCPICTURE, having previously defined MOVes to be dotted lines and having set INP equal to TRUE.

950–1120 These lines make up PROCDRAW. The current cursor position is saved in temporary registers at line 960, then a new text window of the same size as the current graphics window is defined. This is to allow the current view position to be displayed at the top of the screen.

MOV and INP are assigned as necessary, before a loop is entered which repeatedly draws the current picture, and moves the current view position according to the direction keys, until 'Return' is pressed. When the loop has been exited the old screen windows are established, and the cursor moves to its old position.

Suggestions for improvement

- The program could be made more user-friendly by allowing the user to input a two-digit number after S; improving the prompts; allowing curved and diagonal lines; and allowing the function keys to be programmed directly by the program. Most of these enhancements could not, however, be made on the Model A machine because of its limited memory.
- The program could be speeded up considerably by calculating a new picture while a previous one is being displayed, and compressing the program's structure.
- Does the program exit 'gracefully'? If you think not, modify it to do so.

The Listing

```
10 SP=1
20 mode=0
30 lines=32
40 columns=80
50 foreground_number=1
60 background_colour=6
70 foreground_colour=1
80 *KEY9 "100,200,300!M"
90 *KEY8 "0,0,0!M"
100 storage=80*4
110 DIM S$(storage)
120 MODE mode
130 PROCSPPLIT
140 VDU 19,0,background_colour,0,0,0
150 VDU 19,foreground_number,
foreground_colour,0,0,0
160 REPEAT
170 PROCGET
180 IF A$="D" THEN PROCDRAW ELSE PROCEDIT
190 CLG
```

```

200 UNTIL FALSE
210 DEF PROCSPPLIT
220 VDU 24,0;256;1279;1023;
230 VDU 29,0;256;
240 VDU 28,0,lines-1,columns-1,24
250 ENDPROC
260 DEF PROCACT(A$)
270 IF A$="U" THEN W=W+G
280 IF A$="D" THEN W=W-G
290 IF A$="L" THEN U=U+G
300 IF A$="R" THEN U=U-G
310 IF A$="F" THEN V=V+G
320 IF A$="B" THEN V=V-G
330 ENDPROC
340 DEF PROCINIT
350 U=0:V=0:W=0:G=80
360 ENDPROC
370 DEF PROCVIEW(A,B,C)
380 S=A*A+B*B
390 T=S+C*C
400 Q=SQR(T)
410 R=SQR(S)
420 L=A:M=B:N=C
430 ENDPROC
440 DEF PROCPLT3(K,U,V,W)
450 LOCAL O
460 O=T-U*L-V*M-W*N
470 C=T*(V*L-U*M)*4/(R*O)+640
480 D=384+3*Q*(W*S-N*(U*L+V*M))/(R*O)
490 PLOT K,C,D
500 ENDPROC
510 DEF PROCGET
520 INPUT "Enter the X,Y and Z co-ordinates "
A,B,C
530 PRINT "Enter 'D' or 'E'. [Draw or Edit] "
;
540 IF A=0 THEN A=(50+RND(300))*SGN(RND)
550 IF B=0 THEN B=(50+RND(300))*SGN(RND)
560 IF C=0 THEN C=(50+RND(300))*SGN(RND)
570 A$=GET$
580 PRINT
590 ENDPROC
600 DEF PROCPICTURE(A,B,C)
610 LOCAL U,V,W,A$,G
620 PROCINIT
630 PROCVIEW(A,B,C)
640 SP=1
650 PROCPLT3(69,0,0,0)

```

```

660 G=80
670 K=5
680 REPEAT
690 A$=FNGET:S$(SP)=A$
700 IF INP=TRUE THEN PRINT A$;
710 PROCACT(A$)
720 PROCPLT3(K,U,V,W)
730 IF A$="S" THEN PROCSIZE
740 IF A$="O" THEN K=5
750 IF A$="P" THEN K=MOV
760 SP=SP+1
770 UNTIL A$=CHR$(13)
780 PRINT
790 ENDPROC
800 DEF PROCSIZE
810 LOCAL A$
820 SP=SP+1
830 A$=FNGET
840 IF INP=TRUE THEN PRINT A$;
850 G=10*VAL(A$)
860 S$(SP)=A$
870 ENDPROC
880 DEF FNGET
890 IF INP=TRUE THEN VDU 7:=GET$ ELSE=S$(SP)
900 DEF PROCEDIT
910 MOV=21
920 INP=TRUE
930 PROCPICTURE(A,B,C)
940 ENDPROC
950 DEF PROCDRAW
960 TPOS=POS:TVPOS=VPOS
970 VDU 28,0,23,columns-1,0
980 G=80
990 MOV=4
1000 INP=FALSE
1010 U=A:V=B:W=C
1020 REPEAT
1030 PRINT "Current view position is... ";U;",
";V; ", ";W
1040 PROCPICTURE(U,V,W)
1050 VDU 7
1060 A$=GET$
1070 CLS
1080 PROCACT(A$)
1090 UNTIL A$=CHR$(13)
1100 PROCSPLIT
1110 VDU 31,TPOS,TVPOS
1120 ENDPROC

```

Models A and B

Codebreaker

The Game

'Codebreaker', or variants of it, has been popular in England for centuries under the name 'Bulls and Cows'.

The computer 'thinks of' a code consisting of four colours chosen from six possible colours. You have 10 guesses to work out which four colours the computer has in mind. All four colours in the code are different to each other. Not only do you have to guess the colours but you must also discover where in the sequence each colour stands.

The computer helps you after each guess, in terms of 'blacks' (correct colours in the correct position) and 'whites' (colours which appear in the code, but not in the position you specified).

When you run the game, the numbers one to six, and the colours they represent, are shown at the top of the screen. You enter your guess by typing in the numbers which represent the colours but together as one four-digit number which the computer 'strips down' into four separate 'colours'. Each of your guesses, and the feedback information on it, stays on the screen, so you can use the information given on earlier guesses to improve later ones.

The Program

10 Title.

20 Sets the mode.

30 Turns off the cursor.

40 Clears the buffer.

50 Dimensions the arrays to hold the computer's number and processes your guesses.

70-140 Instructions for you, including a print-out of the six colours from which the code will be guessed.

150-160 The GET\$ waits for any key to be pressed, and the screen clears for the game to begin.

180-200 Prints out the six colours and their corresponding numerical codes.

220-300 Selects the four-digit code, making sure that all digits are different.

310 Starts the loop (terminated in 630) to give you up to 10 guesses.

330 Accepts your guess.

340 Moves the print position back to overprint your input.

350-390 Strips your guess down to four separate digits.
 400 Sets the counters for black (B) and white (W) to zero.
 410-450 Looks for blacks, converting any element of the array found to be a black to zero, so it will not be recounted when the whites are assessed.
 460-520 Looks for whites, jumping over (line 470) any elements of the array which have been set to zero.
 530-600 Prints out the colour version of your guess, and the score you have achieved. Note that all the foregoing, from line 330 to this point in the program, happens extremely quickly, so you have the impression that once you type the number in it is reprinted more or less immediately as four colours, together with a score.
 630 If there are less than four blacks (that is, B is not equal to four) the computer goes back to line 320 for the next guess.
 610-620 'Congratulations' message if the code is guessed correctly.
 650-700 Prints out the complete code, either because it has been guessed, or the 10 guesses are over. Line 680 puts a maddening pause into the print-out, just to frustrate you if you are desperate to know what the code was.
 710 Pauses before offering new game.
 720-780 Offers you a new game; and, for any answer except one beginning with N or n, gives you a new game.

Note This program can be improved considerably. For instance, it would be neater to put the computer into Mode 7 at the end of the program, and also give the user the option of ending it 'gracefully'.

The Listing

```

10 REM ** Codebreaker **
20 MODE 7
30 VDU 23;8202;0;0;0
40 *FX 15,0
50 DIM C(4),G(4),H(4)
60 CLS
70 PRINT '''CHR$(133);"I am thinking of a
four-colour code,"
80 PRINT CHR$(133);"which you have 10 goes
to guess."
90 PRINT '''CHR$(134);"I am choosing from
these colours:"'
100 FOR colour=1 TO 6
110 PRINT CHR$(135);colour;">";CHR$(128+
colour);"* ";
120 NEXT
130 PRINT'''CHR$(129);"All four colours are
different."
140 PRINT''''''CHR$(131);"Press any key to
begin..."

```

```

150 A$=GET$
160 CLS
170 PRINT'''
180 FOR colour=1 TO 6
190 PRINT CHR$(135);colour;">";CHR$(128+
colour);"* ";
200 NEXT
210 PRINT '''
220 C(1)=RND(6)
230 Z=1
240 Z=Z+1
250 C(Z)=RND(6)
260 J=0
270 J=J+1
280 IF C(J)=C(Z) THEN 230
290 IF J<Z-1 THEN 270
300 IF Z<4 THEN 240
310 FOR G=1 TO 10
320 PRINT CHR$(133);"Enter guess number ";G
330 INPUT A:IF A<1000 OR A>9999 THEN GOTO 330
340 PRINT CHR$(11)CHR$(11)CHR$(11)
350 FOR Z=1 TO 4
360 G(Z)=A-10*INT(A/10)
370 H(Z)=G(Z)
380 A=INT(A/10)
390 NEXT
400 B=0:W=0
410 FOR Z=1 TO 4
420 IF C(Z)<>G(Z) THEN 450
430 B=B+1
440 G(Z)=0
450 NEXT
460 FOR Z=1 TO 4
470 IF G(Z)=0 THEN 520
480 FOR J=1 TO 4
490 IF C(Z)<>G(J) THEN 510
500 W=W+1
510 NEXT J
520 NEXT Z
530 FOR T=4 TO 1 STEP -1
540 PRINT CHR$(128+H(T));"*";
550 NEXT
560 PRINT CHR$(132);"scored";CHR$(129);B;"
black";
570 IF B<>1 PRINT "s"; ELSE PRINT " ";
580 PRINT CHR$(132);"and";CHR$(129);W;"
white";
590 IF W<>1 PRINT "s" ELSE PRINT

```

```

610 IF B=4 PRINT CHR$(133);"You guessed it...
in just ";G;" guess";
620 IF G>1 AND B=4 PRINT "es"
630 IF B<>4 NEXT G
640 PRINT CHR$(11)CHR$(11)CHR$(11)
650 PRINT ``TAB(3);CHR$(134);"The code was";
660 FOR H=1 TO 5000:NEXT
670 FOR T=4 TO 1 STEP -1
680 FOR H=1 TO 2000:NEXT
690 PRINT CHR$(128+C(T));"*";
700 NEXT
710 FOR H=1 TO 8000:NEXT
720 PRINT ``CHR$(134);"Do you want another
game?"
730 PRINT CHR$(132);TAB(8);"Enter Y or N"
740 A$=GET$
750 IF ASC A$<> ASC "N" AND ASC A$<> ASC "n"
THEN RUN
760 CLS
770 PRINT````CHR$(136);"OK, bye for now!"
780 END

```

Models A and B

Masterword

The Game

This program is similar to 'Codebreaker' except that instead of using colours the computer thinks of a four-letter word (which, you'll be pleased to know, will be remarkably pure). You are given ten guesses to work out the word. The words are held in the DATA statements from lines 800 to 900, and these can easily be changed when you get to know the 40 words provided in the computer's vocabulary. It is useful to make the words you put in the DATA statements similar to each other (words such as FEAT, FEAR and NEAT; or FACE and FATE) so that it is somewhat difficult to guess the word.

You play the game just as you would 'Codebreaker', entering your guess (a four-letter word), for which you are awarded 'blacks' (correct letters in the correct position) and 'whites' (correct letters in the wrong position). In many ways, this game is more challenging than 'Codebreaker' because the computer can choose from 26 variables, rather than just six. But the letters within the code do bear some relationship to each other, so it is often easier to work out the missing letters from the way the word is 'shaping up', rather than from the computer's feedback.

The Program

- 10 Title.
- 20 Sets the mode.
- 30 Turns off the cursor.
- 40 Sets up arrays to hold the codes of the letters of the words.
- 50 Sets the DATA pointer back to the beginning of the list of words.
- 70 Sends action to the procedure which selects the required word.
- 80-120 Brief instructions, and a key-press (line 110) to get the game underway.
- 140 Start of the master loop which counts the number of guesses. This loop terminates in line 430.
- 160 Accepts your guess.
- 170 Asks for a new guess if the one just entered is not a word of four letters.
- 180-190 Sets the counters for black (B) and white (W).

- 200 Compares your word (B\$) with the computer's word (C\$) and, if they are the same, sends the computer to the procedure PROCwin.
- 210-270 Compares the letters of the computer's word with your word to find blacks, substituting any letter which is correct with the code for 9.
- 280-350 Checks the word for whites, jumping over any letter which is a '9'.
- 360 Converts C\$ back into the original word
- 370 Moves the print position up to print over your input.
- 390-420 Prints out your word in a randomly selected colour, and gives an assessment.
- 450 'Sorry' message if the word was not guessed.
- 460 Directs action to the procedure PROCword which prints out the word the computer was thinking of.
- 470 Directs action to PROCnextgo to offer you another game.
- 500-550 This procedure selects a word from the DATA store by reading through it to a random point.
- 570-620 This procedure congratulates you for guessing the word, prints out the word (PROCword) and offers you a new game (PROCnextgo).
- 640-660 Prints out the word the computer was thinking of.
- 680-760 Offers you a new move, prints farewell message if a new game is not wanted.
- 800-900 The 'DATA-bank' vocabulary.

Note As it stands, the program will accept an input such as '1234' or '&&&&'. Program your way around this, and improve the ending message so that it does not run interminably if you don't press 'Escape'.

The Listing

```

10REM*MASTERWORD**
20MODE7
30VDU23;8202;0;0;0
40DIME(4),F(4)
50RESTORE
60REM*****
70PROCfind_a_word
80PRINT''''CHR$(128+RND(6));"I am thinking
of a four-letter word,"
90PRINTCHR$(128+RND(6));"      which you
must try and guess"
100PRINT''''CHR$(128+RND(6));"Press any key
to begin"
110D$=GET$
120CLS:PRINT''
130REM****the game****
140FORE=1TO10

```

```

150PRINTCHR$(128+RND(6));"Enter guess number
";E
160INPUTB$
170IF LEN(B$)<>4 THEN150
180B=0
190W=0
200IF B$=C$ PROCwin
210REM***get blacks***
220FORC=1TO4
230E(C)=ASC(MID$(A$,C))
240F(C)=ASC(MID$(B$,C))
250IF E(C)=F(C) B=B+1
260IF E(C)=F(C) E(C)=ASC"9"
270NEXT
280REM***get whites***
290FORC=1TO4
300IFE(C)=ASC"9" THEN350
310FOR D=1TO4
320IF E(C)<>F(D) THEN340
330W=W+1
340NEXTD
350NEXTC
360A$=C$
370PRINTCHR$(11)CHR$(11)CHR$(11)
380REM***print result***
390PRINTCHR$(128+RND(6));B$;" - ";B;" black";
400IFB<>1 PRINT"s"; ELSE PRINT " ";
410PRINT" and ";W;" white";
420IFW<>1 PRINT"s" ELSE PRINT
430NEXTE
440REM***if didn't guess***
450PRINT"'CHR$(128+RND(6));"Sorry, time is
up"
460PROCword
470PROCnextgo
480END
490REM*****
500DEF PROCfind_a_word
510 FORT=1TORND(40)
520READA$
530NEXT
540C$=A$
550ENDPROC
560REM*****
570DEF PROCwin
580PRINT"'CHR$(128+RND(6));"You got it right
in just ";E;" guess";
590IFE>1 PRINT"es" ELSE PRINT

```

```

600PROCword
610PROCnextgo
620ENDPROC
630REM*****
640DEF PROCword
650 PRINT CHR$(128+RND(6));"The word was";
CHR$(128+RND(6));A$
660ENDPROC
670REM*****
680DEF PROCnextgo
690PRINT'CHR$(128+RND(6));"Do you want to
play again?"
700PRINT'CHR$(128+RND(6));"Enter Y or N"
710Z$=GET$:IF Z$<>"N" RUN
720REM***farewell message***
730CLS
740REPEAT
750PRINTCHR$(128+RND(6));"OK, thanks for the
game"
760UNTIL FALSE
770ENDPROC
780REM*****
790REM**words - change to suit**
800DATA"FACE","FATE","FEAT","FEAR"
810DATA"DASH","NEAT","PUSH","PAST"
820DATA"DATE","DIET","DEAD","GRAB"
830DATA"SAID","DAIS","DUET","DUEL"
840DATA"RANT","RAVE","MARS","GETS"
850DATA"TINY","TRIP","PERT","REST"
860DATA"ROAD","RAID","SOUL","SOLE"
870DATA"STAY","THIN","OVER","DOWN"
880DATA"PEST","LONG","LAST","LIST"
890DATA"MANY","MOST","ONLY","QUIT"
900DATA"QUIZ","PACE","BATS","CATS"

```

Models A and B

Bomb Squad

The Game

In this game you pilot a space-ship across the screen, from left to right, for a predetermined number of trips, dropping bombs on a line of asterisks. If you hit one, that asterisk disappears; if you miss, another one appears. The object of the game is to obliterate all the asterisks.

You drop bombs by pressing any key (except ESCAPE, CONTROL and SHIFT).

A continuously updated display of the 'pass' you're on and your current score is maintained at the top of the screen.

The Program

20 Removes the delay between a key being pressed and its repeating.

30 Sets the number of passes across the screen that will be made.

40 Sets the speed of the game.

60 Puts the machine in Mode 7.

70 Places the code for blue alphanumerics at the start of line 20.

80 Places the code for red alphanumerics at the start of line 10.

90 Starts a loop from line 11 to line 19 of the display.

100 Puts the code for magenta alphanumerics at the start of each of the lines.

110 Ends loop.

120 Starts a loop from one to 39.

130 Calculates the address of the T'th point on the 20th line of the screen.

140 If T is odd, puts an asterisk at that place, else puts a space.

150 Ends loop.

160 Sets the score to 20, since there are 20 asterisks on the screen.

170 Moves the cursor to the tenth position on the top line of the display.

180 Prints the current score, in yellow.

190 Moves the cursor to the tenth position on the second line down the display.

200 Prints the current score, again. A double-height code will shortly be printed.

210 Sets up a loop for each of the PASSes that will be made.

220 Homes the cursor.

230 Prints the PASS number.
240 As above.
250 Starts a FOR loop across the screen for your gun.
260 Places your gun on the screen.
270 If a key has been pressed, calls PROCFIRE.
280 Empties all buffers.
290 Removes the gun from the screen.
300 Ends loop, enables editing keys.
310 Restores key repeat.
320 ENDS the game.
330 Starts the definition of PROCFIRE.
340 Sets all variables used in PROCFIRE to LOCAL.
350 Starts a loop from the position of your gun on line 10 to line 20.
360 Places the bomb symbol, an upright 'equals' sign, at each position on the way.
370 Starts a very short delay.
380 Finishes delay.
390 Replaces the bomb with a space.
400 Ends the 'bomb' loop.
410 D will have been incremented at the NEXT statement, so it is safe to test the character under D. If it is an asterisk, replaces it with a space and decrements the variable SCORE. If not, puts an asterisk there, and increments SCORE.
420 See line 170.
430 See line 180.
440 See line 190.
450 See line 200.
460 Ends PROCFIRE.

Suggestions for improvement

- Sound effects would enhance the game; so would the superior graphics available in Mode 4. To transfer the game to a different mode, you will need to use FNREADCH given in the chapter of *FX calls in the User Guide, to find out which character is at a particular position.
- You could change the game so that you can alter your direction but not your speed.
- Does the game exit 'gracefully'? If not, modify it to do so.

The Listing

```
10 *FX 4,1
20 *FX 11,1
30 PASS=5
40 SPEED=20
50 REM *** KILL ***
```

```

60 MODE 7:VDU 23;8202;0;0;0;
70 ?(HIMEM+20*40)=4
80 ?(HIMEM+10*40)=1
90 FOR T=11 TO 19
100 ?(HIMEM+T*40)=5
110 NEXT T
120 FOR T=1 TO 39
130 P=HIMEM+20*40+T
140 IF (T MOD 2)=1 THEN ?P=ASC("*") ELSE ?P=3
2
150 NEXT T
160 SCORE=20
170 VDU 31,10,0
180 PRINTCHR$(128+3);"SCORE=",SCORE
190 VDU 31,10,1
200 PRINTCHR$(128+3);"SCORE=",SCORE
210 FOR T=PASS TO 1 STEP -1
220 VDU 30
230 PRINT CHR$(13+128);CHR$(2+128);"PASS=";
PASS+1-T
240 PRINT CHR$(13+128);CHR$(2+128);"PASS=";
PASS+1-T
250 FOR G=32145 TO 32183
260 ?G=ASC("O")
270 IF INKEY(SPEED)<>TRUE THEN PROCFIRE
280 *FX 15,0
290 ?G=32
300 NEXT G,T:*FX 4,0
310 *FX 12,0
320 VDU 23;29194;0;0;0;0;:END
330 DEF PROCFIRE
340 LOCAL D,T
350 FOR D=G+40 TO G+9*40 STEP 40
360 ?D=ASC(";")
370 FOR T=1 TO 10
380 NEXT T
390 ?D=32
400 NEXT
410 IF ?D=ASC("*") THEN ?D=32:SCORE=SCORE-1
ELSE ?D=ASC("*"):SCORE=SCORE+1
420 VDU 31,10,0
430 PRINTCHR$(128+3);"SCORE=",SCORE
440 VDU 31,10,1
450 PRINTCHR$(128+3);"SCORE=",SCORE
460 ENDPROC

```

Models A and B

Dome Dweller

The Game

This game is in the classic 'Kingdoms' genre. In most 'Kingdoms' games (including the one on the Welcome Pack which came with your computer) you rule an ancient colony in which your administrative decisions seem limited almost solely to how many acres of land you buy and how much corn you plant. Your attempts to prove a wise and just ruler are constantly thwarted by such natural disasters as floods, rats, thieving neighbours and killer bees, all of which are triggered by the random number generator.

This game is a little different. Although the starting parameters are randomly selected, once you've got them they are – more or less – yours for the duration of a particular game. Apart from an occasional attack from outer space (which will not necessarily occur every time you play the game) the fate of your lunar dome depends entirely upon the wisdom of your decisions in buying and trading.

When the game begins (year one) you have somewhere between 81 and 120 people living in a lunar dome. You have a limited amount of oxygen and food in the stores, and a limited amount of money in the treasury. Your people need a certain amount of food and oxygen to survive each year; the costs of these basic necessities vary from game to game.

Your only means of raising money to buy more food and oxygen, and pay the annual maintenance on your dome, is to trade your 'unique lunar sculptures' with the inhabitants of other domes. This, as you can see, is hardly a mixed economy. However, each lunar sculpture uses up precious oxygen in its manufacture; so the amount of oxygen you have, and the number of people you must support on this oxygen, limits the number of sculptures you can make. From time to time you'll have very generous starting parameters – a lot of oxygen, few people, and low oxygen needs for both the people and the sculpture manufacturing process. If this is so, the dome will last forever (unless you are exceptionally dimwitted) and you will end up with millions in the treasury. However, if the starting parameters (even one of them) are not favourable, or you are simply a hopeless administrator, you will not get beyond year 12.

'Dome' has another important difference from the old 'Ruler of

the Kingdom' games. Instead of some worthless subject beginning the annual report to you with: 'O Ruler of Sumeria . . .', as Dome Master you learn of the state of your domain through the dome's computer. The game output is designed to look just like the possible output of such a computer 'in real life' would look; and it is possible, when you're playing this game late at night by yourself, to become completely wrapped up in it, and almost start believing there is a real dome out there whose survival depends on you.

There are a number of small elements in the program which you may only appreciate after you have played 'Dome' many times.

To make the program easier to modify and follow, explicit names are given to the variables. Although these take longer to enter, it makes for a much clearer program, and you should find it fairly easy to find your way about it. The variables are:

YEAR	The year of your dome's life, with the game starting at year one.
A\$	The tragic phrase, 'The lunar dome is dead'.
FOLK	The number of people within your dome.
CASH	The money in the Treasury.
FOOD	The quantity of food in your food stores.
FOODCOST	How much you must pay per unit of food.
FOODNEED	How much each of your FOLK consume each year.
ARTCOST	The number of units of oxygen it takes to make one of your dome's famed pieces of sculpture.
ARTPAY	How much you can get for these remarkable creations.
OXY	The amount of oxygen in your tanks.
OXYNEED	How much oxygen your FOLK each need per year.
OXYCOST	The price of each unit of oxygen.
REPAIR	Your annual maintenance bill.

The names of the procedures are self-explanatory: year_pop_update; computer_report; food; oxygen; oxydeath; fooddeath; cashdeath; folkdeath; drawline (which 'rules off' each report from your computer); trading; attack (which assigns NASTY\$ to the name of the attacking forces!); and warningbell.

The Program

10 Title.

20 Sends action to the initialisation procedure.

30-120 The main REPEAT/UNTIL loop. It cycles through the updating of the year and the number of people within your dome, and gives you the chance to make and trade your precious sculptures and buy food and oxygen. From time to time, if you're unlucky, the random number generator (see line 110) will decide it is time a NASTY\$ burst in from the vacuum and inflicted some damage.

130–320 This is the initialisation procedure which sets the starting parameters for your term of office.

290–310 This procedure adds one to the year and increases your population. You'll find that in some games your dome dwellers appear to have learned their breeding habits from terrestrial rabbits; in others, they seem to have better things to do (like making those sculptures or fending off NASTY\$).

330–520 This is the major procedure (PROCcomputer_report) in the game. It controls the information the dome's computer feeds to you during the game. Lines 350 to 380 are the 'doom' ones, which terminate the game if you run out of food, money, oxygen or people. Lines 390 to 420 are the 'warning bells' which let you know when supplies of anything are nearing critical levels. Lines 430 to 500 print out in every round of the game (a round is four months long) to keep you in touch with your dome. The figures in this are updated between rounds when necessary.

530–590 This procedure, 'drawline', draws a red line across the screen, pauses, and makes some squiggly noises.

600–690 This is the creative procedure, where you at last get a chance to make those pieces of sculpture. The only limit on the number you can make in a year is the amount of oxygen. The computer will not allow you to attempt to make more than your oxygen would permit (see warning in line 640).

700–810 The computer tells you how much food you have in stock and how long this will last for the present population. It also gives you the chance to buy as much food as you can afford. The computer ensures (line 770) you do not try to overspend. 'Neither a borrower nor a lender be' is the rule on the moon.

820–920 This procedure allows you to buy oxygen, the most vital ingredient in your dome's economy and the one element in the economic mix which you can most easily mismanage.

930–1260 This is the most dramatic procedure, the one when NASTY\$ attack. Note the routine from lines 980 to 1000. It determines which of the baddies has got at you this time. The rest of the procedure gives you the bad news . . . in detail.

The balance of the program is made up of procedures which you trigger when you've made some fatally bad decision or series of decisions. These procedures all trigger the final one ('warningbell') until you're sick to death of its sound.

1270–1330 You have run out of oxygen.

1340–1410 You have run out of food.

1420–1500 You have run out of money.

1510–1620 You have run out of people and, as the program happily

informs you, 'You are the only one left alive on your dome – you have betrayed your Oath of Office . . . I hope you feel real bad!!!!!!'
1630–1680 The sound you're going to come to hate – the sound of the warning bell.

Suggestions for improvement

- Make it much harder to play; reduce the starting stocks of food and oxygen, and increase the number of people and the amounts of food and oxygen they need.
- Do the same by cutting the return from sculptures and increasing the amount of oxygen these consume during manufacture.
- Add even more factors which you must control, such as viruses and plagues, radiation poisoning, being hit by stray meteorites, dialogue with other domes to (a) stop them attacking you; or (b) uniting with you to attack other domes to steal food and oxygen.
- Limit the number of sculptures which each person can make each year. This will really make the dome hard to control and is not a modification I would suggest you make until you are familiar with the program in its present form.
- As it stands, the program accepts negative numbers and alpha characters as inputs – so modify the program to avoid this.
- Cuts off the sound after a couple of seconds.
- Does the game exit 'gracefully'? If not, modify it to do so.

The Listing

```
10 REM ** DOME DWELLER **
20PROCinitialise
30REPEAT
35 *FX15,2
40PROCyear_pop_update
50PROCcomputer_report
60PROCtrading
70PROCcomputer_report
80PROCfood
90PROCcomputer_report
100PROCoxygen
110IF RND(5)=2 PROCattack
120UNTIL FALSE
130DEF PROCinitialise
140 CLS
150YEAR=0
160A$="The lunar dome is dead"
170FOLK=80+RND(40)
180CASH=INT(7*(700+RND(2000)/RND(3)))
190FOOD=700+RND(500)
200FOODCOST=RND(7)
```

```

210FOODNEED=1+RND(5)
220ARTCOST=1+RND(5)
230ARTPAY=30*RND(ARTCOST)
240OXY=3000-RND(2000)
250OXYNEED=2+RND(4)
260OXYCOST=2+RND(7)
270REPAIR=200+RND(400)
280ENDPROC
290DEF PROCyear_pop_update
300YEAR=YEAR+1
310FOLK=FOLK+INT(FOLK/(12+RND(8)))
320ENDPROC
330DEF PROCcomputer_report
340PRINT'CHR$(128+RND(5));"Computer report
to Dome Master:"'
350IF OXY<OXYNEED*FOLK PROCoxydeath
360IF FOOD<FOODNEED*FOLK PROCCfooddeath
370IF CASH<50 PROCCashdeath
380IF FOLK<2 PROCCfolkdeath
390IF FOLK<13 PRINTCHR$(129);"WARNING!"' "
Population is nearing extinction"
400IF OXY<2*OXYNEED*FOLK PRINT CHR$(130);"
WARNING!"'CHR$(129);"Oxygen supplies are low"
410IF FOOD<2*FOODNEED*FOLK PRINT CHR$(131);"
WARNING!"' "Food stocks depleted"
420IF CASH<2000 PRINT CHR$(129);"WARNING!"' "
Cash reserves dangerously low"
430PRINT' 'CHR$(131);"There are ";FOLK;"
people living"
440PRINTCHR$(131);"within your dome in year "
;YEAR'
450PRINTCHR$(130);"Money credit is $";CASH
460PRINT CHR$(130);"Annual maintenance
charge is $";REPAIR'
470PRINT CHR$(133);"Oxygen tanks hold ";OXY;"
units"
480PRINT CHR$(133);"Oxygen costs $";OXYCOST;"
per unit"
490PRINT CHR$(133);"Each dome dweller needs "
;OXYNEED;" units a year"'
500PRINTCHR$(134);"Food stocks stand at ";
FOOD
510PROCdrawline
520ENDPROC
530DEF PROCdrawline
540PRINT CHR$(129);"
-----"
550T=TIME

```

```

560SOUND1,-7,RND(254),5:SOUND1,-7,RND(254),5
570REPEAT UNTIL TIME-T=400
580SOUND1,-7,RND(254),5:SOUND1,-7,RND(254),5

590ENDPROC
600DEF PROCtrading
610PRINT CHR$(132);"You can trade your
unique lunar"CHR$(132);" sculptures with the
people"CHR$(132);" who live in other domes"
620PRINT CHR$(129);" You use up ";ARTCOST;
" units of oxygen"CHR$(129);"making each one,
and sell them for $";ARTPAY
630PRINT CHR$(134);"How many pieces of
sculpture will you"CHR$(134);" create
this year?"
640INPUTB
650IFB*ARTCOST>OXY PRINT CHR$(129);"There is
not enough oxygen to "CHR$(129);"make that
many":GOTO640
660CASH=CASH+B*ARTPAY
670OXY=OXY-B*ARTCOST
680PROCdrawline
690ENDPROC
700DEF PROCfood
710PRINT CHR$(132);"Food costs $";FOODCOST;"
per unit"
720PRINT CHR$(132);"Each dweller needs ";
FOODNEED;" units a year."
730PRINT CHR$(129);"($";FOODCOST*FOODNEED;"
each, $";FOLK*FOODCOST*FOODNEED;" for dome.
This will"
740PRINT CHR$(129);"last ";INT(FOOD/(
FOODNEED*FOLK));" years at present population.)
"
750PRINT CHR$(130);"How many food units will
you buy?"
760INPUT C
770IF C*FOODCOST>CASH PRINT CHR$(129);"You
do not have enough money":GOTO760
780FOOD=FOOD+C*FOODCOST
790CASH=CASH-C*FOODCOST
800PROCdrawline
810ENDPROC
820DEF PROCoxygen
830PRINT CHR$(133);"How much oxygen will you
buy?"
840PRINT CHR$(129);"(Current stocks will
last ";INT(OXY/(OXYNEED*FOLK));" years"

```



```

850PRINT CHR$(129);"at the present
population of ";FOLK;"")
860INPUTD
870IF D*OXYCOST>CASH PRINTCHR$(129);"There
is not enough money!":GOTO860
880FOOD=FOOD-FOLK*FOODNEED
890CASH=CASH-REPAIR-D*OXYCOST
900OXY=OXY+D-FOLK*OXYNEED
910PROCdrawline
920ENDPROC
930DEF PROCattack
940FORJ=1TORND(25)+15 STEP RND(3)
950SOUND1,-15,RND(100)+30,1:SOUND1,-15,250,1

960NEXT
970RESTORE1010
980FORJ=1TORND(6)
990READ NASTY$
1000NEXT
1010DATA"A fleet of Sirian ships","renegade
dwellers from a nearby dome","Martian sub-
fighters","Vyrillix outworlders","a lone ship,
under robot control","a Parralexian escort
vessel"
1020 PRINT CHR$(133);"The dome is under
attack by"
1030PRINT CHR$(132);NASTY$
1040FORJ=1TO100 STEP RND(7)
1050SOUND1,-15,200-2*J,1:SOUND1,-15,J*2,1
1060NEXT
1070 FOLKDEAD=INT(FOLK/(RND(35)+1)) +1
1080DAMAGE=INT(RND(CASH/9))
1090IF CASH-DAMAGE<1 DAMAGE =0
1100FOODDEAD=INT(RND(FOOD/2))
1110OXYDEAD=INT(RND(OXY/2))
1120PRINT CHR$(133);"There were ";FOLKDEAD;"
people killed"
1130FOLK=FOLK-FOLKDEAD
1140PROCdrawline
1150 IF DAMAGE>0 PRINT CHR$(129);"Damage to
the dome totals $";DAMAGE
1160CASH=CASH-DAMAGE
1170PROCdrawline
1180PRINT CHR$(133);FOODDEAD;" units of food
have been ruined"
1190FOOD=FOOD-FOODDEAD
1200PRINT CHR$(133);"and ";OXYDEAD;" units of
oxygen leaked away"CHR$(133);"before the dome
could be repaired"

```

```

1210PROCdrawline
1220REPAIR=REPAIR+RND(30)
1230FORJ=1TO100 STEP RND(7)
1240SOUND1,-15,200-2*J,1:SOUND1,-15,J*2,1
1250NEXT
1260ENDPROC
1270DEF PROCoxydeath
1280REPEAT
1290PROCwarningbell
1300PRINT 'CHR$(133);A$
1310PRINT 'CHR$(129);"You ran out of oxygen
in year ";YEAR
1320UNTIL FALSE
1330ENDPROC
1340DEF PROCfooddeath
1350REPEAT
1360PROCwarningbell
1370PRINT' CHR$(129);A$
1380PRINT' CHR$(132);"You ran out of food in
year ";YEAR
1390PRINT CHR$(129);"Now, ";FOLK;" people
will starve"'CHR$(129);" to death!"
1400UNTIL FALSE
1410ENDPROC
1420DEF PROCcashdeath
1430REPEAT
1440PROCwarningbell
1450PRINT CHR$(131);A$
1460PRINT' CHR$(134);"The treasury ran dry in
year ";YEAR
1470PRINT CHR$(131);"so the ";FOLK;" people
who relied on you"
1480PRINT CHR$(131);"will";CHR$(129);"die!"
1490UNTIL FALSE
1500ENDPROC
1510DEF PROCfolkdeath
1520REPEAT
1530PROCwarningbell
1540PRINT"You are the only one left alive"
1550PROCwarningbell
1560PRINTCHR$(130);"on your dome -- You have
betrayed your"
1570PROCwarningbell
1580PRINT CHR$(129);"Oath of Office...I hope
you feel"
1590PROCwarningbell
1600PRINT CHR$(133);"real";CHR$(129);"bad!!!!"
"
1610UNTIL FALSE

```

```
1620ENDPROC
1630DEF PROCwarningbell
1640FORJ=1T050
1650SOUND RND(4)-1,-15,RND(10)+240,RND(5)
1660NEXT
1670SOUND1,1,1,1
1680ENDPROC
```

Models A and B

Wordsquare

The Game

This program imitates a sliding plastic tile puzzle. A grid is printed on the screen with letters of the alphabet filling every space except one. This space can be moved around the grid with the cursor control keys. As the space is moved, the letters in the grid move around to accommodate it. Therefore, it is possible for you to eventually get the letters in alphabetical order.

A continuous readout of the number of moves made so far is shown. This adds a competitive element to the game. When the letters are in order, pressing ESCAPE will return you to the first prompt. This asks for the dimensions of the 'square' (it can be a rectangle). The dimensions are entered in the form of two numbers that are separated by a comma.

The game is quite difficult until you work out a system.

The Program

The main section is from lines 10 to 140.

10 This sets up an error handler, so that 'Escape' will return the user to the start of the game.

20 This line waits half a second for a key to be pressed, and if a key was pressed within this time limit, ends the game. This is so that you can end the game without pressing BREAK. To end the game press 'Escape' *and* then press any other key as quickly as possible.

40 This line clears all variables, since otherwise the DIM statement in line 50 could be executed twice, which would lead to an error.

50 Dimensions a string array which will hold the state of the grid.

60 Calls the procedure which sets up starting conditions for the game, and asks you for the dimensions of the grid.

70 Calls the procedure which prints out the grid in colour.

80 Starts the main loop of the game.

90-110 Prints the number of moves made so far.

120-130 Accept a single keypress from you and then call PROCact, which interprets the key pressed and acts on it if necessary.

140 Finishes the main game loop. Notice that this is an infinite loop, so the only way out of it is to press 'Escape'.

The section from line 150 to line 350 makes up PROCset_up. It is called in line 60.

170 Initialises the variable MOV, which contains the number of moves made so far.

180 Sets up a REPEAT loop, which will be executed until a valid response to the dimensions question is registered.

190 Prints the prompt for the question, followed by a row of spaces. The spaces are to erase any previous dimensions that you input which were unacceptable.

200 Moves the cursor to the space after the prompt, and accepts the two dimensions.

210 Finishes the input loop when the dimensions that have been input are valid.

220 Clears the screen.

230 Turns off the cursor.

240 Sets up a REPEAT loop which will fill the grid with letters. It does this by repetitively generating random co-ordinates inside the grid, until the co-ordinates of an empty array element are generated. It then fills that square with the next letter in sequence. The last part of line 280 ensures that the bottom right element is not chosen, since that element will contain the space at the start of the game.

310 Places a space character in the bottom right hand corner of the grid.

320 Switches the cursor control keys to the mode where they generate actual values rather than just shift the cursor around.

330 Together with line 340, this sets variables L and M to the co-ordinates of the space in the grid.

350 Returns control to line 70.

360–390 These make up PROCend, which is called at the end of the program to restore the cursor keys to their normal functions and to turn the cursor back on.

400–500 These make up PROCprint_whole, which prints out the whole grid, using a simple loop arrangement. The grid itself comprises underscore characters, which appear as elongated minus signs in Mode 7.

510–650 These make up the last procedure – PROCact.

530 Generates a string which contains all the valid characters. These are the codes generated by the four cursor control keys.

540 Checks to see if the key pressed was one of these keys, and if not returns control to line 140, without doing anything.

550 Extracts the ASCII code of the key pressed.

560–590 Check to see if you are trying to move off the grid, and if so, return control to line 140.

600 Increments the move counter, MOV, because we can now be certain that the key just pressed was valid.

610-640 Act on the key pressed, by moving the space around the screen. FNad generates the screen address of the cell whose coordinates are X, Y.

660 Defines FNad, as used in the previous procedure.

Suggestions for improvement

- If you have a model B, the game could be transferred to Mode 0, where a larger grid could be used.
- If you can work out an algorithm for solving these puzzles (which is not nearly as difficult as it sounds) you may like to write a section of code which solves the puzzle when you give it a starting condition. This could be extended to the point where the machine gives you hints on where your best move would be, at a penalty of, say, five points.
- All in all, Wordsquare gives scope for considerable modification on your part.

The Listing

```
10 ON ERROR GOTO 20
20 IF INKEY$(50)<>" THEN PROCend:END
30 MODE 7
40 CLEAR
50 DIM letter$(10,10)
60 PROCset_up
70 PROCprint_whole
80 REPEAT
90 FOR T=4 TO 5
100 PRINT TAB(0,Y*2+T);CHR$(129);CHR$(128+13);
"Moves made so far - ";MOV
110 NEXT T
120 A$=GET$
130 PROCact(A$)
140 UNTIL FALSE
150 DEF PROCset_up
160 LOCAL XC,YC,T
170 MOV=0
180 REPEAT
190 PRINT TAB(0,5);CHR$(132);"Enter size of square...";CHR$(131);STRING$(56," ")
200 INPUT TAB(25,5) U%,V%:X=U%:Y=V%
210 UNTIL X>1 AND Y>1 AND X<8 AND Y<8 AND (X*Y)<=27 AND INT(X)=X AND INT(Y)=Y
220 CLS
230 !&FE00=&10200A
240 FOR T=1 TO X*Y-1
250 REPEAT
260 XC=RND(X)
```

```

270 YC=RND(Y)
280 UNTIL letter$(XC,YC)="" AND (XC<>X OR YC<>
Y)
290 letter$(XC,YC)=CHR$(T+64)
300 NEXT T
310 letter$(X,Y)=" "
320 *FX 4,1
330 L=X
340 M=Y
350 ENDPROC
360 DEF PROCend
370 *FX 4,0
380 !&FE00=&10720A
390 ENDPROC
400 DEF PROCprint_whole
410 LOCAL XC,YC
420 PRINT CHR$(131);STRING$(X*2+1,"-")
430 FOR YC=1 TO Y
440 VDU 131,ASC("-")
450 FOR XC=1 TO X
460 PRINT letter$(XC,YC);"-";
470 NEXT XC
480 PRINT 'CHR$(131);STRING$(X*2+1,"-")
490 NEXT YC
500 ENDPROC
510 DEF PROCact(A$)
520 LOCAL V$,A
530 V$=CHR$(136)+CHR$(137)+CHR$(138)+CHR$(139)
540 IF INSTR(V$,A$)=0 THEN ENDPROC
550 A=ASC(A$)
560 IF A=136 AND L=1 THEN ENDPROC
570 IF A=137 AND L=X THEN ENDPROC
580 IF A=138 AND M=Y THEN ENDPROC
590 IF A=139 AND M=1 THEN ENDPROC
600 MOV=MOV+1
610 IF A=136 THEN ?FNad(L,M)=?FNad(L-1,M):L=L-
1:?FNad(L,M)=32
620 IF A=137 THEN ?FNad(L,M)=?FNad(L+1,M):L=L+
1:?FNad(L,M)=32
630 IF A=138 THEN ?FNad(L,M)=?FNad(L,M+1):M=M+
1:?FNad(L,M)=32
640 IF A=139 THEN ?FNad(L,M)=?FNad(L,M-1):M=M-
1:?FNad(L,M)=32
650 ENDPROC
660 DEF FNad(X,Y)=HIMEM+X*2+Y*80-40

```

Models A and B

Tyranno

The Game

In this game, you (the fearless yellow asterisk) must battle with a mighty prehistoric monster (the dreaded red @) on a Blue Land measuring 10 by 10. It is impossible to evade 'Rex' forever.

You move yourself using the 'Q' (moves you up), 'M' (down), 'A' (left) and 'L' (right) keys. You'll find your fingers soon learn to make the right movements automatically. Your score is clocked up on the screen all through the game, along with the 'high score' which is automatically updated, when needed, throughout the game. You get a tiny pause at the end before you and 'Rex' are placed at random positions on the Blue Land, to battle it out again.

The Program

10 Title.

20 Sets HISCORE variable to zero.

30 Start of the master REPEAT/UNTIL loop which terminates in line 180.

50 Sends action to the initialisation procedure (PROCbegin).

60-70 These direct the computer to the procedures which place you and 'Rex' in position.

90-170 This REPEAT . . . UNTIL loop directs the computer to the procedures which actually play the game. The loop repeats until the flag GOT is changed from zero to one.

190-370 PROCbegin is the initialisation procedure. Line 200 clears the screen (the computer assumes you are in Mode 7, by the way), and 210 turns off the cursor. Line 220 prints out the instructions on which key to press for which result; the loops 230 to 270 draw out the Blue Land upon which the battle will be fought. The score for each game is set to zero in line 280. Then the locations for 'Rex' (C and D) and the human (X and Y) are chosen at random. As the two of you move around the screen, your old positions must be overprinted with blue squares: C1 and D1 hold the 'overprint position' for 'Rex', while X1 and Y1 the overprint position for you. 390-420 Line 400 overprints your old position and 410 prints the new position.

440-470 Line 450 overprints the old position of 'Rex', and 460 prints in the new position.

490–570 This procedure accepts your move. Lines 500 and 510 set X1 and Y1 equal to the current position, before the new position is determined. Line 520 reads the keyboard, and lines 530 to 560 move you in accordance with this reading, so long as the move will not take you outside the Blue Land. The 'AND X greater than 1' part of these four statements ensures that you will not stray.

590–610 This procedure (PROCcheck_if_got) does a spot-check. If 'Rex' and you are occupying the same square, you've been 'got'.

630–740 This procedure moves 'Rex' around. Lines 670, 690, 710 and 730 check 'Rex's position in relation to yours, and move him towards you. However, lines 660, 680, 700 and 720 randomly prevent 'Rex' from checking one or more of the relationships. Without these lines (as you can easily see by deleting them), 'Rex' would swallow you without fail in a couple of seconds. Randomness introduces some stupidity into 'Rex's movements, and gives you some chance to evade his crushing jaws for as long as possible.

760–800 This procedure checks the score (SCORE) in the current game and compares it with the highest score yet achieved (HISCORE), modifying HISCORE when necessary.

820–870 This procedure, joyfully executed when you are finally nabbed, prints out the 'Gotcha, human!!' message in changing colours, and makes appropriate eating noises. The flag GOT is set to zero to terminate the REPEAT/UNTIL loop running from 90 to 180.

950–970 This procedure (sound_and_fury) is called throughout the game (see line 130) to add interest.

Suggestions for improvement

- Add a routine to shoot 'Rex'.
- Decrease 'Rex's stupidity (or increase it).
- Add diagonal moves for yourself.
- Add a routine to keep yourself moving in a designated direction unless a new key is pressed.

The Listing

```
10 REM TYRANNO CHASE
20HISCORE=0
30REPEAT
40 GOT=0
50 PROCbegin
60 PROCplace_human
70 PROCplace_rex
80 REM*****
90 REPEAT
100PROCupdate_score
110PROCget_move
```

```

120PROCplace_human
130PROCsound_and_fury
140PROCmove_rex
150PROCplace_rex
160PROCcheck_if_got
170 UNTIL GOT=1
180 UNTIL FALSE
190 DEF PROCbegin
200 CLS
210VDU23;8202;0;0;0
220PRINT'''CHR$(133);"Q - up, M - down, A - le
ft, L - right"
230FORA=1TO10
240FORB=1TO10
250 PRINT TAB(2*(A+4),B+4);CHR$(132);CHR$(25
5)
260 NEXTB
270 NEXTA
280 SCORE=0
290 C=RND(10)
300 C1=C
310 D=RND(10)
320 D1=D
330 X=RND(10)
340 X1=X
350 Y=RND(10)
360 Y1=Y
370 ENDPROC
380 REM*****
390 DEF PROCplace_human
400PRINT TAB(2*(X1+4),Y1+4);CHR$(132);CHR$(255
)
410PRINT TAB(2*(X+4),Y+4);CHR$(131);"*"
420ENDPROC
430 REM
440 DEF PROCplace_rex
450 PRINT TAB(2*(C1+4),D+4);CHR$(132);CHR$(255
)
460PRINT TAB(2*(C+4),D+4);CHR$(129);"@"
470 ENDPROC
480 REM
490 DEF PROCget_move
500 X1=X
510 Y1=Y
520A$=INKEY$(0)
530IFA$="A" AND X>1 X=X-1
540IFA$="L" AND X<10 X=X+1
550IFA$="Q" AND Y>1 Y=Y-1

```

```

560IFA$="M" AND Y<10 Y=Y+1
570ENDPROC
580 REM*****
590DEF PROCcheck_if_got
600IFX=C AND Y=D PROCgotcha
610ENDPROC
620 REM*****
630DEF PROCmove_rex
640D1=D
650 C1=C
660IF RND(5)>3 THEN 740
670IF C<X C=C+1
680 IF RND(5)>3 THEN 740
690IF C>X C=C-1
700IF RND(5)>3 THEN 740
710 IF D<Y D=D+1
720 IF RND(5)>3 THEN 740
730 IF D>Y D=D-1
740 ENDPROC
750 REM*****
760DEF PROCupdate_score
770IF HISCORE<SCORE HISCORE=SCORE
780SCORE=SCORE+1
790PRINT TAB(5,20);CHR$(134)"Score is ";SCORE;
CHR$(130);" High score is";CHR$(128+RND(5));HIS
CORE+1
800ENDPROC
810 REM*****
820DEF PROCgotcha
830 FORcount=1TO34
840 SOUND0,-15,RND(10),3
850PRINT TAB(4,17);CHR$(130);"Gotcha, human!!"
860 FORG=1TO50:NEXTG
870 PRINT TAB(4,17);CHR$(129);"Gotcha, human!!"
"
880PRINT TAB(5,20);CHR$(134)"Score is ";SCORE;
CHR$(130);" High score is";CHR$(128+RND(5));HIS
CORE+1
890NEXTcount
900*FX 15,0
910SOUND0,0,0,0
920 GOT=1
930 ENDPROC
940 REM*****
950DEFPROCsound_and_fury
960SOUND3,-15,RND(100),1:SOUND2,-15,RND(128)+1
27,2
970 ENDPROC

```

Models A and B

Franglais

The Game

This program 'translates' a phrase in English to an equivalent 'Franglais' one. Franglais is a curious mixture of French and English. It is promoted in magazines like *Punch*, where simple words are translated correctly into French but any complex words are left in English.

This program attempts to translate English into Franglais automatically. All you do is type English in response to the prompts displayed. After a few seconds the translation will appear. When you are bored, enter a pre-designated keyword to indicate that you've finished.

Input can be in upper or lower case, or both. All output is in upper case. We realise it's a little contrived, but we think you'll agree that it's fun. Do not include any punctuation marks in your input unless they are flanked by spaces.

The Program

- 40 Starts the main repeat loop of the program. This loop only finishes when the keyword is found in your input.
- 50 Requests you to type your phrase for translation.
- 60 Inputs your phrase. The INPUT LINE statement is used to ensure that commas will be read in as valid input.
- 80 Calls PROCtranslate, which translates the phrase and prints it out.
- 90 Carries on with the main loop until the keyword is found. Insert a word of your choice in the quotes.
- 100 No comment.
- 110 Ends the program.

- 130 Starts the definition of FNlower_case_to_upper. This function changes all lower case letters in its argument to upper case.
- 140 Sets all the LOCAL variables required.
- 150 Sets the destination string C\$ to be the null string.
- 160 Starts a loop through all the characters of A\$, the string to be converted.
- 170 Picks out the B'th character.
- 180 If it's lower case, changes it to upper case. Refer to the table of ASCII codes in the User Guide.

- 190 Updates C\$.
- 200 Ends the loop.
- 210 Returns with C\$.
- 230 Defines a 'debugged' INSTRfunction, to get around a 'bug' in Version 1 of the BASIC supplied.
- 270 Starts the definition of FNchange_single_word. This function translates the single word held in A\$, or whatever string is used as the argument, to an equivalent Franglais string.
- 290 Returns if A\$ is not a proper word.
- 300 Restores the DATA pointer to the beginning of the vocabulary DATA statements.
- 310 Starts a REPEAT loop through all the words in the vocabulary.
- 320 Reads the next English and French pair of words.
- 330 Stops the loop when the end of DATA has been reached, or when a match is found.
- 340 If the end of DATA was reached, exits with the calling string . . .
- 350 or else, exits with the French string.
- 370 Starts the definition of PROCtranslate.
- 380 Sets the local variable required.
- 390 Changes the string to be translated to upper case.
- 400 Adds a space to the end of the string for translation.
- 410 Starts a REPEAT loop through all the words in the string for translation, A\$.
- 420 Finds the first space in the string.
- 430 Then calls the translate single word function, with all the characters to the left of the space.
- 440 Strips off the bit that has just been translated . . .
- 450 until A\$ has all been translated.
- 460 Prints a new line.
- 470 Ends PROCtranslate

Lines 490 onwards are a list of English words with their French equivalents, ended by two asterisks. Feel free to expand on this.

The Listing

```

10 REM Franglais
20 REM *****
*****
30 MODE 7
40 REPEAT
50 PRINT CHR$(132);"Enter the English sentenc
e to be          ";CHR$(132);" 'translated' to Fren
ch "
60 INPUT LINE ENG$
70 REM CAN INSERT OUTPUT TO PRINTER HERE
80 PROCtranslate(ENG$)

```

```

    90 UNTIL FNINSTR(ENG$, "A KEY WORD", 0)
    100 PRINT "Merci..."
    110 END
    120 REM *****
*****
    130 DEF FNlower_case_to_upper(A$)
    140 LOCAL T, B$, C$
    150 C$=""
    160 FOR T=1 TO LEN(A$)
    170 B$=MID$(A$, T, 1)
    180 IF B$>"a" THEN B$=CHR$(ASC(B$)-32)
    190 C$=C$+B$
    200 NEXT T
    210=C$
    220 REM *****
*****
    230 DEF FNINSTR(A$, B$, K)
    240 IF LEN(B$)>LEN(A$) THEN=0
    250=INSTR(A$, B$, K)
    260 REM *****
*****
    270 DEF FNchange_single_word(A$)
    280 LOCAL ENG$, FRE$
    290 IF A$=" " OR A$="" THEN=""
    300 RESTORE
    310 REPEAT
    320 READ ENG$, FRE$
    330 UNTIL ENG$="*" OR A$=ENG$
    340 IF ENG$="*" THEN=A$
    350=FRE$
    360 REM *****
*****
    370 DEF PROCtranslate(A$)
    380 LOCAL P
    390 A$=FNlower_case_to_upper(A$)
    400 A$=A$+" "
    410 REPEAT
    420 P=FNINSTR(A$, " ", 0)
    430 PRINT FNchange_single_word(LEFT$(A$, P-1));
" ";
    440 A$=MID$(A$, P+1)
    450 UNTIL A$=""
    460 PRINT
    470 ENDPROC
    480 REM *****
*****
    490 DATA THE, LE, HE, LUI, HIS, LE, ROOM, CHAMBRE, PEA
, POIS, STEAK, ENTRECOTE

```

500 DATA ME, MOI, I, JE, HERE, ICI, HAVE, AI, A, UNE, A,
UN, MY, SON
510 DATA BIG, GRAND, MATCH, ALLUMETTE, SUPER, FANTA
STIQUE, DEAD, MORT, DIE, MORT
520 DATA GIN, VIN, WHISKEY, VIN, WHISKY, VIN, MARTIN
I, VIN, BEER, VIN, NEWCASTLE
530 DATA PARIS, CIGARETTES, GAULOISES
540 DATA HAIR, CHEVAUX, ARM, BRA, LEG, JAMBE, LEFT, D
ROITE, RIGHT, GAUCHE
550 DATA TRENDY, AVANT-GUARDE, MEDICINE, VIN, POLI
CE, GENDARME, DETECTIVE, CLUESO
560 DATA DOOR, PORTE, PORT, DOOR, QUAY, CLEF, KEY, CL
EF, HEAD, TETE, LOVE, AMOUR
570 DATA HOUSE, MAISON, CHAIR, CHAISE, EYE, ORIEL, S
UN, SOLIEL
580 DATA SONG, CHANSON, MY, MOI, YOUR, VOTRE, DESK, P
UPITRE
590 DATA FRIENDS, AMIS, WINDOW, FENETRE, BEHIND, DE
RRIERE, SEA, MER, MOTHER, MERE
600 DATA FATHER, PERE, CAR, VOITURE, APPLES, POMMES
, ENGLISHMEN, POMS, POTATOES
610 DATA POMME DE TERRE, CAT, CHAT, DOG, CHIEN, BLU
E, BLEU, LITTLE, PETITE, MUSIC
620 DATA MUSIQUE, PLEASE, SI'L VOUS PLAIT, BOY, GA
RCON, GIRL, FILLE, CHIPS
630 DATA POMME FRITES, FISH, POISON, CHICKEN, POUL
ET, DUCK, CANARD, MUSTARD, MOUTARDE
640 DATA HOT, CHAUD, COLD, FROID, EVERYBODY, TOUT L
E MONDE, WORLD, MONDE
650 DATA HELLO, BONJOUR, GOOD, BON, SWEETS, BON BON
S
660 DATA *, *

Models A and B

Rain Catcher

The Game

This program 'gives' you a bucket that you must use to try and stop a sudden deluge of rain from drowning you. The bucket appears as a white bar on the screen, and is moved left and right with the left and right cursor control keys. As the waterdrops fall, you must position yourself under each drop. The difficult part is that, as soon as you start moving, the drops speed up by a factor of about five. When a drop has either added to the puddle at the bottom of the screen, or has been caught by the bucket, the bucket moves back to the centre of the screen.

When the drop reaches the bottom of the screen it adds to the multi-coloured puddle already there. When the puddle is deep enough to drown you, the game ends, with suitable special effects.

The game is quite difficult to play but is more fun than the normal 'breakout' type game.

The Program

The game is divided into several sections. The main section is from line 20 to line 270.

10 This line defines a key to restore the keyboard to its normal repeat rate and lets the editing keys work as normal. *Press f0 after an ESCAPE to give you control over the machine again.* To LIST the program now, go first into Mode 7.

20 Sets the speed of the game.

30 Makes sure the computer is in Mode 7.

40 Makes a call to PROCSETUP, which colours the screen and sets up the bucket and other housekeeping jobs.

50 Sets up the main REPEAT loop. This loop is the one which sends down the raindrops.

60 Sets up the REPEAT loop which releases each raindrop.

70 Moves the cursor to the position the raindrop used to occupy, and puts a 'graphics control code' in its place.

80 Increments the y co-ordinate of the raindrop, before re-drawing it at line 90.

100 Waits for a key press, for T% centiseconds.

110 Clears the keyboard buffer, since any erroneous key presses will give the bat a delayed action.

120 Together with line 130, decides whether the last key pressed makes it necessary to call the bat-moving subroutines.
140 This UNTIL statement stops the drop when it has reached another drop, or has hit the bucket, or has reached the bottom of the screen.
150 Works out if the drop did in fact hit the top raindrop already in the puddle; if so, sets the flag K% to true; else, to false.
160 Works out if the drop hit the bucket and, if so, blanks out the drop.
170 This line puts a line of spaces where the bucket should be.
180 Sets S%, the position of the bucket, to 14, which is about the centre of the line.
190 Places the pail at the correct position on the screen. (A\$ contains all the information required to print the bucket.)
200 Chooses a new x co-ordinate for the next raindrop.
210 Sets the y co-ordinate of the new raindrop to zero, the top of the screen.
220 Allows the game to carry on until the game end flag, K% is set.
230 With line 240, this resets the keyboard to normal repeating speed, and returns the cursor keys to their usual function.
250 The call to PROC SHUDDER creates a grand finale.
260 Together with line 270, wraps up the loose ends and then ends the program.

The next main section is the procedure SETUP at lines 290 to 480.

310 Defines a text window to be the entire screen, except the extreme left-hand column.
320 Starts a FOR loop to place colour control codes down the entire left-hand column of the screen. Thus the previously defined window will stop the colour information from being scrolled.
350 Sets up a FOR loop to fill the bottom seven lines of the screen with random colour control codes.
380 Places the 'white graphics' control code in the extreme left-hand position of the 17th row of the screen.
390-410 Initialise variables which contain the X and Y position of the raindrop and the displacement of the bucket from the left-hand side of the screen.
420 Sets up a string A\$ containing the characters which make up the bucket. These are a space, five lowercase p's, and another space. A lower case 'p' looks like a single line when displayed on a graphics line of the screen.
430 Allows the cursor control keys to generate specific codes, rather than move the editing cursor around the screen.
440 Places the bucket in its initial position at the centre of the screen.
450 Turns the cursor off.
460 & 470 Make all keys repeat instantaneously, and very quickly.

The next two sections are the procedures called when the bucket is to be moved right or left.

PROCLEFT comes first.

510 Checks to see if the bucket is not already on the left hand side; and, if not, decrements S%, which is the bucket's displacement from the left of the screen.

520 Replaces the bucket on the screen.

PROCRIGHT is similar.

560 Checks to see if the bucket is not already too far to the right, and if not increments S%.

570 Replaces the bucket on the screen.

PROCSHUDDER is a special routine, used to shake the screen from right to left. VDU 23,0,13,X,0;0;0; puts the value X in the 6845's register 13. This register governs the position on the screen of the extreme left hand character. Thus it usually contains 0. *It is not recommended that inexperienced users mess around with the 6845's internal registers.* You are not going to damage your computer if you do, but it is easy to upset your television.

Suggestions for improvement

- The screen layout could be improved, to make the picture more interesting; for example, the bucket's present shape is not particularly convincing.
- A continuously updated score could add a bit more tension. This could also take the form of some sort of time limit, again continuously updated on the screen.

The Listing

```
10 *KEY 0 "*FX 12:M*FX 4:M"
20 T%=10
30 MODE 7
40 PROCSETUP
50 REPEAT
60 REPEAT
70 VDU 31, X%, Y%, RND(5)+144
80 Y%=Y%+1
90 VDU 31, X%, Y%, 255
100 L%=INKEY(T%)
110 *FX 15, 0
120 IF L%=136 THEN PROCLEFT
130 IF L%=137 THEN PROCRIGHT
140 UNTIL (Y%>23) OR (? (HIMEM+Y%*40+X%+41)=25
5) OR (? (HIMEM+Y%*40+X%+41)=112)
```

```

150 IF (? (HIMEM+Y%*40+X%+41)=255) AND Y%=17
THEN K%=-1 ELSE K%=0
160 IF (? (HIMEM+Y%*40+X%+41)=112) THEN VDU 31
,X%,Y%,32
170 PRINT TAB(0,17);STRING$(38," ")
180 S%=14
190 PRINT TAB(S%,17);A$
200 X%=RND(37)
210 Y%=0
220 UNTIL K%
230 *FX 12
240 *FX 4
250 PROC SHUDDER
260 MODE 7
270 END
280 REM *****
*****
290 DEF PROCSETUP
300 LOCAL C%
310 VDU 28,1,24,39,0
320 FOR C%=0 TO 40*16 STEP 40
330 ?(C%+HIMEM)=20
340 NEXT C%
350 FOR A%=40*18 TO 40*24+39
360 ?(HIMEM+A%)=RND(5)+16
370 NEXT A%
380 ?(HIMEM+17*40)=16+7
390 X%=10
400 Y%=0
410 S%=14
420 A$=" "+STRING$(5,CHR$(112))+ " "
430 *FX 4,1
440 PRINT TAB(S%,17);A$
450 !&FE00=&10200A
460 *FX 11,1
470 *FX 12,1
480 ENDPROC
490 REM *****
*****
500 DEF PROCLEFT
510 IF S%>0 THEN S%=S%-1
520 PRINT TAB(S%,17);A$
530 ENDPROC
540 REM *****
*****
550 DEF PROCRIGHT
560 IF S%<32 THEN S%=S%+1
570 PRINT TAB(S%,17);A$

```

```
580 ENDPROC
590 REM *****
*****
600 DEF PROC SHUDDER
610 LOCAL A,B,C
620 FOR C=1 TO 20
630 FOR A=0 TO 39
640 VDU 23,0,13,A,0;0;0;
650 NEXT A
660 FOR B=39 TO 0 STEP -1
670 VDU 23,0,13,B,0;0;0;
680 NEXT B
690 NEXT C
700 ENDPROC
710 REM *****
*****
```

Models A and B

Reversi

The Game

At the start of this game a multi-coloured board is displayed. The letters 'H' and 'C' are used for the 'human piece' and the 'computer piece' respectively. The board is set up with the first four pieces on it. You always move first, so you think of which square you wish to move to and then enter the co-ordinates of that square, with the displacement down the screen, first. The co-ordinates are entered as a single two-digit number.

After you have pressed 'Return', the computer starts to consider its move. This takes some time, so the program has been set up so that you can see the computer's 'mental processes'. As it looks at each of the squares in turn, that square flashes. If going to that square is the best move it has found so far the square turns into a purple block. If later on a better move is found, the purple block is removed and placed elsewhere.

When the computer has examined all the possible moves, it decides on a final move. Then the score board is updated as it makes the move. If you cannot move, just enter '99' instead. If in two moves running neither player can move, the game ends.

Unless you enjoy cheating, do not look at the description section until you've played a few games – once you know the algorithm used, the computer is at a distinct disadvantage.

The Program

10 Puts the computer in Mode 7.

20 Calls PROCBOARD. This procedure prints out the board and sets up some variables needed in the rest of the program.

30 Starts a loop to get a valid move from you.

40 Calls PROCSCORE, which prints out the current scores.

70 Inputs the actual move. The CLS will not wipe out the board, because a text window was defined in PROCBOARD to occupy two lines just below the board.

80 This UNTIL statement tests the move entered, in a number of ways. First, the move entered is checked to make sure that it is a positive whole number, then the function FNPIECE(X) is used to check what character is at the location where you want to move. The character should be a full stop. FNPIECE is used again later on. FNNUM(A\$,B\$,X) is used to check the number of pieces that

you (A\$) could make by moving to X. A\$ and B\$ must be H or C. This has to be greater than zero.

Alternatively, the game is allowed to continue if you entered 99 as the move, to indicate that you cannot move. The computer doesn't bother to check to see if you *could* move (it would take far too long); so, when you are in a tight spot, you could cheat and say you cannot move.

90 If you could not move, this line jumps over the section concerned with making your move and updating the scores.

100 The variable HUM is your score, and this is updated by using the previously described function, FNNUM(A\$,B\$,X).

110 Similarly, the variable COM, the computer's score, is updated.

120 A dummy variable is then used to call FNTURN(A\$,B\$,X). This is almost identical to FNNUM, except that as well as counting the number of pieces that you could take, it turns them over.

130 Calls PROCSCORE, to print out the newly updated scores.

140 Tells you to press 'Return' when you are ready for the computer to start making its move.

150 Uses a dummy variable to GET a key press.

160 Clears the screen window.

170 Prints out the scores again.

180 Initialises the variables H and M. H holds the number of the square with the highest 'score' found so far. This score is a measure of how good a move it would be for the computer to move there. M holds the square which had this highest 'score'.

190 This line sets up a loop through all the possible squares on the board, plus a few more such as 79 and 40. These 'extra' squares are weeded out in the next line.

200 This line jumps over the computer's move section if that square is not occupied by a full stop; that is, it is not vacant.

210 Gets the character at the square.

220 Replaces it with a space, to make it appear to flash.

230 The 'score' of a square is made up of the number that can be turned over at a particular square, multiplied by the value of FNVALUE for that square. FNVALUE gives a weighting to each square; so that, for example, the corners will be moved to in preference to the sides or anywhere else. This number is calculated for the square being considered. If it is bigger than the current highest score, it changes X\$ to a block, to allow the purple block to appear; puts a full stop at the last best position; calculates a new value for H; and copies C to M, to make the best move so far be the current square.

240 The character in X\$ is placed at the square just considered. This will normally be a full stop, unless X\$ was changed to a block in line 220.

250 Ends the loop through all the squares.

260 If H is still zero, no move was possible. If that happened, and you couldn't move, ends the game.

- 270 Updates your score, as previously described.
- 280 Similarly with the computer's score.
- 290 Now computer's move can be made, using FNTURN.
- 300 The program loops back, to get another move from you.
- 310 Just for completeness, an END statement has been included.

- 320 Starts the definition of PROCBOARD.
- 330 As a MODE 7 statement was executed just before this procedure is called, the cursor is on the top line of the display. This PRINT statement moves it down a line.
- 340 This line prints the top of the board. It consists of the code for purple alphanumerics, the code for a block, the code for blue alphanumerics, the numbers from 1 to 8, the code for purple alphanumerics, and the code for a block.
- 350 Starts a loop to print out the middle section of the board.
- 360 Prints the middle section of the board. It consists of the code for blue alphanumerics, the loop counter, and eight lots of the code for red alphanumerics followed by a full stop. At the end of the line, the loop counter is printed out in blue.
- 370 Ends the loop.
- 380 Is the same as line 340, to print out the bottom line of the board.
- 390 Sets your score to two (for the two pieces at the start of each game).
- 400 Sets the computer's score to two.
- 410 Puts your piece at the correct place for the starting set up.
- 420 Puts another of your pieces down.
- 430 Places one of the computer's two starting pieces on the board.
- 440 Puts the last of the computer's pieces on the board.
- 450 Dimensions DIR(). This array holds the displacements from a particular square to each of its eight neighbours.
- 460 Fills up part of DIR().
- 470 Fills up part of DIR().
- 480 Fills up part of DIR(). These three lines are arranged in such a way that you can see which displacement belongs to which direction.
- 490 Sets up the previously described text window.
- 500 Ends PROCBOARD.

- 510 Starts the definition of PROCPIECE. This procedure places the piece A\$ at position X.
- 520 Sets variable A to be LOCAL.
- 530 Calculates the screen address corresponding to the position X.
- 540 Puts the character in A\$ at the position.
- 550 If the character was an H, then makes it a green H, placing the alphanumeric green code in the location before the H.
- 560 If the character was C, then makes it a light blue C.
- 570 If the character was a full stop, then makes it a red full stop.

580 If the character was a block, then makes it a purple block.
590 Ends PROCPIECE.
600 Starts the definition of PROCSCORE.
610 Prints the current scores.
620 Ends PROCSCORE.
630 Starts the definition of FNPIECE(X). This reads the character at a particular location on the board.
640 Sets A to be a LOCAL variable.
650 Calculates the address of location X.
660 If the character at that address was an H then exits the function with H.
670 If the character at that address was a C, then exits with C.
680 If it was a full stop, then exits with a full stop.
690 If the character was a block, exits with a full stop.
700 As the character was something odd, exits with a pound sign.
710 Starts the definition of FNTURN(A\$,B\$,X). This function turns over pieces starting at location X, for player A\$.
720 Sets all the variables used in this function to be LOCAL.
730 Q will hold the number of pieces turned over.
740 Starts a loop to look in each of the eight possible directions from a particular square.
750 Sets the current square to be the starting square.
760 Repeatedly moves in the direction T, until it finds a square which is not occupied by you.
770 If only one square was moved before a square not occupied by the opponent was found, or the piece at the end of the trail is not one of your pieces, jumps past the turning-over routine.
780 Starts a loop which encompasses all the locations moved to.
790 Makes each location into one of your pieces.
800 Ends the loop.
810 Increments Q, as some captures were made.
820 Ends the loop for all the different directions.
830 Ends the function with Q.
840 Starts the definition of FNNUM(A\$,B\$,X).
850 Sets all the variables used in the function to be LOCAL.
860 As above.
870 The algorithm used is the same as the one in FNTURN, except that no pieces are turned over, and Q is properly maintained.
960 Ends the function with Q.
970 Starts the definition of FNVALUE(X). This routine gives the weighting for the square X.
980 A and B will hold the two separate digits of the square X.
990 A and B are calculated, using MOD and DIV.
1000 If any of the corners are X, gives a multiplication factor of 10, and exits.

1010 If X is anywhere else, except one away from any of the sides, gives a multiplication factor of 4.

1020 As X is one away from the side, which means that you will probably get the side, gives a multiplication factor of 1.

Suggestion for improvement

One way this game could be improved is by altering the way FNVALUE operates, to give more exact biases towards specific squares. For example, if you wanted to give the square 67 a bias of 100, you would add a line such as: 1055 IF B = 6 AND A = 7 THEN = 100.

The Listing

```
10 MODE 7
20 PROCBOARD
30 REPEAT
40 PROCSCORE
50 SOUND 1, -15, 100, 3
60 VDU 131
70 INPUT "Enter your move "move:CLS
80 UNTIL (ABS(INT(move))=move AND FNPIECE(
move)=". " AND FNNUM("H", "C", move)>0) OR move=99
90 IF move=99 THEN GOTO 130
100 HUM=HUM+FNNUM("H", "C", move)+1
110 COM=COM-FNNUM("H", "C", move)
120 DUMMY=FNTURN("H", "C", move)
130 PROCSCORE
140 PRINT CHR$(129); "Press"; CHR$(130); "
return "; CHR$(129); "for my move";
150 DUMMY$=GET$
160 CLS
170 PROCSCORE
180 H=0:M=-1
190 FOR C=11 TO 88
200 IF FNPIECE(C)<>". " THEN GOTO 250
210 X$=FNPIECE(C)
220 PROCPIECE(" ", C)
230 IF (FNVALUE(C)*FNNUM("C", "H", C))>H THEN
X$=CHR$(255):PROCPIECE(". ", M):H=FNVALUE(C)*
FNNUM("C", "H", C):M=C
240 PROCPIECE(X$, C)
250 NEXT C
260 IF H=0 AND move=99 THEN PRINT "END OF
GAME " ; :VDU 26:END
270 HUM=HUM-FNNUM("C", "H", M)
280 COM=COM+1+FNNUM("C", "H", M)
290 DUMMY=FNTURN("C", "H", M)
300 GOTO 30
```

```

310 END
320 DEF PROCBOARD
330 PRINT
340 PRINT CHR$(128+5);CHR$(255);CHR$(128+4);"
1 2 3 4 5 6 7 8";CHR$(128+5);CHR$(255)
350 FOR T=1 TO 8
360 PRINT CHR$(128+4);T;STRING$(8,CHR$(129)+
."");CHR$(128+4);T
370 NEXT T
380 PRINT CHR$(128+5);CHR$(255);CHR$(128+4);"
1 2 3 4 5 6 7 8";CHR$(128+5);CHR$(255)
390 HUM=2
400 COM=2
410 PROCPIECE("H",44)
420 PROCPIECE("H",55)
430 PROCPIECE("C",54)
440 PROCPIECE("C",45)
450 DIM DIR(8)
460 DIR(1)=-11:DIR(2)=-10:DIR(3)=-9
470 DIR(4)=-1:DIR(5)=1
480 DIR(6)=9:DIR(7)=10:DIR(8)=11
490 VDU 28,0,14,39,13
500 ENDPROC
510 DEF PROCPIECE(A$,X)
520 LOCAL A
530 A=HIMEM+(X MOD 10)*2+41+(X DIV 10)*40
540 ?A=ASC(A$)
550 IF A$="H" THEN ?(A-1)=2
560 IF A$="C" THEN ?(A-1)=6
570 IF A$="." THEN ?(A-1)=1
580 IF A$=CHR$(255) THEN ?(A-1)=5
590 ENDPROC
600 DEF PROCSCORE
610 PRINT CHR$(128+3);"YOUR SCORE-";HUM;TAB(2
0);CHR$(128+3);"MY SCORE-";COM
620 ENDPROC
630 DEF FNPIECE(X)
640 LOCAL A
650 A=? (HIMEM+(X MOD 10)*2+41+(X DIV 10)*40)
660 IF A=ASC("H") THEN="H"
670 IF A=ASC("C") THEN="C"
680 IF A=ASC(".") THEN="."
690 IF A=255 THEN="."
700="##"
710 DEF FNTURN(A$,B$,X)
720 LOCAL T,S,G,Q
730 Q=0
740 FOR T=1 TO 8

```

```

750 S=X
760 REPEAT S=S+DIR(T):UNTIL FNPIECE(S)<>B$
770 IF (S-X)=DIR(T) OR FNPIECE(S)<>A$ THEN
GOTO 820
780 FOR G=X TO S STEP DIR(T)
790 PROCPIECE(A$,G)
800 NEXT G
810 Q=Q+1
820 NEXT T
830=Q
840 DEF FNNUM(A$,B$,X)
850 LOCAL T,S,Q,K
860 LOCAL
870 Q=0
880 FOR T=1 TO 8
890 S=X:K=-1
900 REPEAT
910 S=S+DIR(T):K=K+1
920 UNTIL FNPIECE(S)<>B$
930 IF FNPIECE(S)<>A$ THEN K=0
940 Q=Q+K
950 NEXT T
960=Q
970 DEF FNVALUE(X)
980 LOCAL A,B
990 A=X MOD 10:B=X DIV 10
1000 IF X=11 OR X=18 OR X=81 OR X=88 THEN=10
1010 IF A<>7 AND A<>2 AND B<>2 AND B<>7 THEN=4
1020=1

```

Poet

The Game

In this game, you type RUN and then sit back to admire the brilliant 'poems' written by your BBC microcomputer. Once you've endured a few hundred stanzas using the words in the DATA statements we've provided, add your own words to create your own poems.

The idea for this program came from an article called 'Producing Computer Poetry' by Margaret Chisman (*The Best of Creative Computing*, Volume 2, edited by David Ahl, Creative Computing Press, Morristown, New Jersey, USA; pp. 106-107) in which Ms Chisman described how she 'wrote' computer poetry by first writing a verse which, although its content may have been nonsense, gave her (and the computer) a scanning pattern to work from. Ms Chisman said that she then broke down the original poem, analysed the kinds of words it used, and collected a number of similar words which the computer could then juggle to its heart's content within the dictated framework.

We decided to follow suit, and started with the following coruscating verse:

*Slowly I straightened in the sun
Hardly a man was caring
... had dared
A tree wandered through the doorway
... That botany should be confounded
Thinking, palmtrees waved
... Waiting, a tall tree shuddered*

With a beginning like that, how could we fail? We broke the original verse down into parts, working out which were bridging words, which nouns; and so on. Then, we wrote the raw framework of the program, leaving procedures 'aword' to 'iword' to be filled in by trial and error. The nouns ('bword', 'dword', and 'fword') were fairly easy to add, but some of the other sections (such as 'gword') proved difficult in practice. Although in this procedure we put words that should have worked, they did not seem to fit when the program was run. The selection you see in this listing is the final result of hours of trial and error.

The Program

The program itself is fairly simple and should prove easy to modify.
10 Title.

20–30 Clear the screen and set the mode.

40 Turns off the cursor.

50 Starts the master REPEAT/UNTIL loop (terminated in line 390).

60 Puts four blank lines between verses.

70–360 Go to specified procedures, adding link words (lines 150, 270) from time to time, and starting new print lines (lines 120, 170, 190, 240 and 290) at other times.

370–380 Pause at the end of each verse before a new one is written. The rest of the program consists of procedures. Each contains a number of words in a DATA statement. The loop in each procedure, FOR J = 1 TO RND(X), chooses a word from this DATA statement, prints it with a space after it, then returns to the program for the next instruction.

Suggestions for improvement

- Change the words within the DATA statements to create poems of the same general feel as ours, but using your own words.
- Once you understand how the process works, write your own 'framework verse', then create a program from scratch to build on your framework.

Finally, here is a sample poem from this program :

*Savagely he frightened in the path
little a child had frightened
... had chastened
shadows wandered to the joy
... that the void might be stopped
waiting she frightened
... wanting a child cried*

*Quietly I sighed before the darkness
only the woman had wandered
... had cried
a crowd waited in the pain
... that the path should be chastened
turning they wandered
... asking a lonely man sighed*

*Sadly the woman stopped before the night
much a lonely man had waited
... had waited
I sighed before the agony
... that the darkness ought to be whispered
crying she waited
... wanting the prophet wondered*

The Listing

```
10 REM ** POET **
20 CLS
30 MODE 4
40 VDU23;8202;0;0;0
50 REPEAT
60 PRINT ' ' ' '
70 PROCaword
80 PROCbword
90 PROCcword
100 PROCiword
110 PROCdword
120 PRINT
130 PROCeword
140 PROCbword
150 PRINT "HAD ";
160 PROCcword
170 PRINT ' "          ...HAD ";
180 PROCcword
190 PRINT
200 PROCbword
210 PROCcword
220 PROCiword
230 PROCfword
240 PRINT ' "...THAT THE ";
250 PROCdword
260 PROCgword
270 PRINT "BE ";
280 PROCcword
290 PRINT
300 PROCyword
310 PROCbword
320 PROCcword
330 PRINT ' "          ...";
340 PROCyword
350 PROCbword
360 PROCcword
370 T=TIME
380 REPEAT UNTIL TIME-T>950
390 UNTIL FALSE
400 END
410 REM *****
420 DEF PROCaword
430 RESTORE 480
440 FOR J=1 TO RND(10)
450 READ A$
460 NEXT
470 PRINT A$" ";
```

480 DATA "SADLY", "HOPELESSLY", "WITHOUT CARE", "QUICKLY", "JOYFULLY", "OFTEN", "SAVAGELY", "CALMLY", "QUIETLY", "SILENTLY"

490 ENDPROC

500 REM *****

510 DEF PROCbword

520 RESTORE 570

530 FOR J=1 TO RND(10)

540 READ B\$

550 NEXT

560 PRINT B\$ " ";

570 DATA "I", "HE", "THEY", "THE PROPHET", "A LONELY MAN", "SHADOWS", "A CHILD", "SHE", "A CROWD", "THE WOMAN"

580 ENDPROC

590 REM *****

600 DEF PROCcword

610 RESTORE 660

620 FOR J=1 TO RND(13)

630 READ C\$

640 NEXT

650 PRINT C\$ " ";

660 DATA "STOPPED", "WONDERED", "HOPED", "WAITED", "WANDERED", "SIGHED", "WHISPERED", "CRIED", "SCREAMED", "ENDED", "IGNORED", "CHASTENED", "FRIGHTENED"

670 ENDPROC

680 REM *****

690 DEF PROCdword

700 RESTORE 750

710 FOR J=1 TO RND(10)

720 READ D\$

730 NEXT

740 PRINT D\$ " ";

750 DATA "WOODS", "NORTH", "SUN", "PATH", "SEA", "DARKNESS", "NIGHT", "VOID", "DOORWAY", "CAVERN"

760 ENDPROC

770 REM *****

780 DEF PROCeword

790 RESTORE 840

800 FOR J=1 TO RND(5)

810 READ E\$

820 NEXT

830 PRINT E\$ " ";

840 DATA "ALL", "MUCH", "NOTHING", "LITTLE", "ONLY"

" 850 ENDPROC

860 REM *****

870 DEF PROCfword

880 RESTORE 930

```

890 FOR J=1 TO RND(5)
900 READ F$
910 NEXT
920 PRINT F$ " ";
930 DATA "SORROW", "AGONY", "PAIN", "ANGUISH", "JO
Y"
940 ENDPROC
950 REM *****
960 DEF PROCgword
970 RESTORE 1020
980 FOR J=1 TO RND(4)
990 READ G$
1000 NEXT
1010 PRINT G$ " ";
1020 DATA "MAY", "MIGHT", "OUGHT TO", "SHOULD"
1030 ENDPROC
1040 REM *****
1050 DEF PROC hword
1060 RESTORE 1110
1070 FOR J=1 TO RND(10)
1080 READ H$
1090 NEXT
1100 PRINT H$ " ";
1110 DATA "SIGHING", "REACHING", "TURNING", "HOPIN
G", "ASKING", "NEEDING", "WAITING", "CRYING", "TAKING
", "WANTING"
1120 ENDPROC
1130 REM *****
1140 DEF PROCiword
1150 RESTORE 1200
1160 FOR J=1 TO RND(4)
1170 READ I$
1180 NEXT
1190 PRINT I$ " THE ";
1200 DATA "TO", "IN", "PAST", "BEFORE"
1210 ENDPROC

```


Models A and B

Magic Square

The Game

The computer generates a three-by-three 'magic square', which vertically, horizontally and diagonally adds up to the same number. Three of the numbers are shown as red hash signs when you first run the program, and you have to enter a series of guesses to work out the answer. At the end, the computer tells you how many guesses it took you to work out the three missing numbers.

The Program

10 Title.

20 Sets mode.

30 Sends computer to the initialisation procedure.

50-80 The master REPEAT/UNTIL loop, which prints out the square after each move (PROCprintout) and accepts your guesses (PROCaccept-move).

100-130 Prints the end of game message.

150-450 This procedure assigns the values to each of the elements in the grid using the three random numbers generated by the REPEAT/UNTIL loop (200 to 240). Lines 340 to 370 check that none of the nine numbers in the square equals zero; and, if any one is zero, sends action back to the 200-240 REPEAT/UNTIL to try again. Lines 390 to 410 copy the elements of array A into array B, then lines 420 to 440 change three of the elements of array B to zero. These are the numbers that you must try to deduce.

470-560 This procedure prints out the whole magic square, and checks to see how many numbers still have to be guessed.

480 Adds one to the number of guesses made.

490 Prints title.

500-540 Prints out the numbers, or prints a pound sign if a number has not been guessed.

550 Tells you the number of numbers still to be guessed.

570-670 This procedure accepts the player's guess, and checks whether it is correct (line 630). The next line counts the number of correct guesses. Line 660 overprints the last guess.

Suggestions for improvement

- Add sound to reward a correct guess.
- Let the number of values to be guessed change from game to game.
 - Change the maximum values for A, B and C to be greater than nine. Note that if you do this you will have to change the way the program decides which squares to turn into zeroes (see lines 420 to 440).
- Change colours to suit your idea of the best colour combinations.
- Restore the flashing cursor when the game ends.
- Change the layout of the square to make it look tidier.
- Modify the program to accept number inputs only, with a corresponding message, and without increasing the count of the number of guesses.
- Cut off the sound after a couple of seconds at the end of the game, offering an option to start again, or exiting 'gracefully'.

The Listing

```
10 REM *Magic Square*
20 MODE7
30 PROCinitialise
40 REM*****
50 REPEAT
60 PROCprintout
70 PROCaccept_move
80 UNTIL M=9
90 REM*****
100 PROCprintout
110 PRINT ' 'CHR$(128+RND(5));"You have
solved it. Well done!"
120 PRINT ' 'CHR$(128+RND(5));"    you ";J-1;"
guesses"
130 END
140 REM*****
150 DEF PROCinitialise
160 DIM A(9),B(9)
170 W=-99
180 M=6
190 J=0
200 REPEAT
210 A=RND(9)
220 B=RND(9)
230 C=RND(9)
240 UNTIL A<>B AND A<>C AND B<>C
250 A(1)=A+B
260 A(2)=A-B-C
```

```

270 A(3)=A+C
280 A(4)=A-B+C
290 A(5)=A
300 A(6)=A+B-C
310 A(7)=A-C
320 A(8)=A+B+C
330 A(9)=A-B
340 K=1
350 FOR Z=1 TO 9
360 IF A(Z)=0 K=0
370 NEXT
380 IF K=0 THEN 200
390 FOR Z=1TO9
400 B(Z)=A(Z).
410 NEXT
420 B(ABS(A))=0
430 B(ABS(B))=0
440 B(ABS(C))=0
450 ENDPROC
460 REM*****
470 DEF PROCprintout
480 J=J+1
490 PRINT TAB(3,3);CHR$(128+RND(5));"Magic
Square"''''
500 FOR Z=1TO9
510 IF B(Z)=0 PRINT CHR$(129);"£ ";:GOTO 530
520 PRINT CHR$(129+RND(4));B(Z);" ";
530 IF Z=3 OR Z=6 PRINT:PRINT
540 NEXT
550 IF M<9 PRINT ''CHR$(128+RND(5));"You
have ";9-M;" to solve"
560 ENDPROC
570 DEF PROCaccept_move
580 M=0
590 PRINT TAB(1,18);CHR$(128+RND(5));"Please
enter guess number ";J
600 INPUT W
610 FOR Z=1TO9
620 IF W=-99 THEN 640
630 IF A(Z)=W THEN B(Z)=W
640 IF B(Z)<>0 THEN M=M+1
650 NEXT
660 PRINT TAB(2,18);"
"
670 ENDPROC

```

Models A and B

Romland

The Game

You are an intrepid explorer, working your way through Romland, a peculiar place surrounded on all sides by impassable walls and studded with a remarkable collection of landmarks – ‘no entry’ sectors, magic tunnels, bottomless quagmires, RAM beasts and treasure (‘the fabled lost gold mines of Romland’). You must survive in this unusual place for 25 hours while collecting as much gold as you can.

From time to time you will be shown a map of Romland from above. You can draw up a 10×10 grid and place landmarks on it to form a map of Romland as you go along, so you can work out how to avoid the beasts and quagmires while homing in on the gold.

You start each game in ‘sector 55’ and can move North, South, East or West at each move. If you try to walk through one of the walls surrounding Romland, or try to enter a ‘no entry sector’, the program will tell you that you cannot go that way, and you’ll have to choose a new direction.

After each move, assuming you haven’t wandered into a sector containing something, your Romland Scanner will swing into action, looking at the eight sectors immediately surrounding the one you are currently occupying. If it finds something there, you will be given a clue as to what is near you. However, if it finds more than one thing, it will only tell you about one of them, and will not tell you where – in relation to your present position – the thing is.

The game explains itself as it progresses, and makes good use of the sound and colour available on the BBC microcomputer.

The Program

10 Title.

20 Sets the mode.

30 Turns off the cursor.

40 Sends action to the initialisation procedure (PROCset_up_romland).

50 Start of master REPEAT/UNTIL loop.

60 To PROCstatus, which informs the player, after each move, what the Romland scanner has detected.

70–80 Adds one to H, the variable storing the number of hours the player has survived within the walls of Romland.

90 Q is a flag that dictates the section of the procedure PROCend_of_the_world which is printed from time to time.

120–410 This procedure creates the initial conditions within Romland (PROCset_up_rom_land), and initialises the variables (H is hours, Q is the flag mentioned before, G is the gold found); the array A holds each sector of Romland. The routine 150 to 210 puts up the walls (character 255) and spaces (character 46, the full stop). As the REM statement in line 230 points out, the next routine (220 to 390) distributes the landmarks within the land, assigning a particular character code to a component of the land.

Line 250 selects an element of the land at random and assigns it to a 'no exit' (character 255, a solid square). Line 260 chooses an element of the array at random and, in a check repeated in the rest of the loop for other landmarks, makes sure this new element has not already been assigned to a 'no exit'. Thus the 'no exits' within the walls are not reassigned, and – far more importantly – the walls are not breached. The other landmarks are assigned in the rest of the routine: magic tunnels (character 63, a question mark); Rom beasts (character 66, the letter B); and quagmires (character 81, the letter Q).

Note that apart from not overwriting 'no exit' elements, there is no mechanism to ensure that an element assigned to a landmark in one part of the loop is not reassigned in another. This ensures that the exact number of each landmark varies from game to game. The Y loop, from 350 to the first part of 390, assigns approximately twice as many 'money sectors' as any other landmark. The player begins in sector 55, and the number 55 is assigned to variable E, which keeps track of the player's position within the land.

The procedure PROCstatus (lines 430–780) is the heart of the game, being the only one which is called repeatedly (from line 60) within the master REPEAT/UNTIL loop.

440 Places the H (for human) in the sector you are occupying.

450–460 Generates a random value for Q and, if it is zero, sends action to PROCmap to give you a glimpse of Romland from above.

490–510 Tells you where you are and how much gold you have found.

520 Sends the program to PROCclues to check the squares surrounding the one you occupy.

530 Tells you how many more hours you must last.

540 Chooses a number at random (from 129 to 133) to determine the colour in which the next two lines will be printed.

570–580 Accept a direction from the player (N, S, E or W) and first check (line 580) that the direction chosen is one of these.

590 Clears the screen, and assigns zero to the variable U which will, if it becomes one, indicate a 'no exit'.

600–680 Check to see if you can move in the direction indicated,

and, if not, point this out in no uncertain manner, in double-height letters (using CHR\$(141) for this purpose).

700–730 Move you into the chosen sector.

740–770 If the sector is not empty, send you to a procedure related to the contents of the sector.

800–900 PROCtunnel. This is the ‘magic tunnel’ which will move you to a random place within Romland, but not into a ‘no exit’ or wall sector.

920–1250 PROCbeast. This is one of the less frightening bad sectors you can stumble into, because you have a chance to escape. With a lot of sound and fury the computer lets you know you are in a beastie sector, but you need patience to discover your fate. The possible messages are:

Warning!! You have come across a Rom beastie!

Stand by!

Luckily for you, this one is blind!

Oh no! It has seen you, and decides to ignore you!

Oh no! It has seen you, and . . . EATS YOU . . .

As you can guess, any message except the one ending in ‘EATS YOU’ will allow you to continue the game.

1260–1380 PROCquagmire. To the accompaniment of Hartnell’s ‘Quartet for Sucking Noises and Despair’ you are warned of your imminent demise.

1400–1620 PROCgold. This is a jolly procedure, in which – with the aid of a dazzling screen display – you gain between £101 and £150 (line 1540). Note that the ‘ in the listing stands for a £ sign.

1630–2040 PROCclues. This procedure checks the surrounding sectors and reports on what they contain, reporting only on one of the sectors, and not telling you which sector the report refers to.

2050–2190 PROCend-of-the-world. This procedure is called only at the end of the game, your survival or extinction (held by the value assigned to Q) determining which part of the procedure is used.

2200–2290 PROCmap. This is the cartographic procedure, printing out the view of Romland from the air. This procedure is called at random within a game (see lines 450 and 460) and at the ‘end of the world’.

Suggestions for improvement

- Add a ‘shoot at the beastie’ mechanism to give you a chance to survive if it sees you.
- Add other things besides ‘gold’ which you can carry with you (such as a gun which you’ll need later for shooting at the beastie).
- Add a little ‘quagmire goblin’ who will allow you to bribe your way out of the muck if you have enough gold.

- Change the way the map is presented during the game, so only a randomly selected part of the map is printed, with little indication as to where the section comes from.
- Add extra landmarks which have unique effects.
- Increase the size of Romland.
- Cut off the sound after a couple of seconds at the end of the game, offering an option to start again, or exiting 'gracefully'.

The Listing

```

10 REM *ADVENTURE IN ROMLAND*
20 MODE7
30 VDU 23;8202;0;0;0
40 PROCset_up_rom_land
50 REPEAT
60 PROCstatus
70 H=H+1
80 UNTIL H=25
90 Q=9
100 PROCend_of_the_world
110 END
120 DEFPROCset_up_rom_land
130 DIMA(100)
140 H=1:Q=0:G=0
150 FOR B=1TO100
160 A(B)=46
170 IF B<22 OR B>90 THEN A(B)=255
180 IF B=31 OR B=41 THEN A(B)=255
190 IF B=51 OR B=61 OR B=71 OR B=81 THEN A(B)=
255
200 IF 10*INT(B/10)=B THEN A(B)=255
210 NEXT B
220 FOR B=1TO4
230 REM:DISTRIBUTE LANDMARKS
240 REM:NO ENTRY
250 A(RND(77)+11)=255
260 X=RND(77)+11:IF A(X)=255 THEN 260
270 REM:MAGIC TUNNEL
280 A(X)=63:REM "?"
290 X=RND(77)+11:IF A(X)=255 THEN 290
300 REM:RAM BEAST
310 A(X)=66:REM "B"
320 X=RND(77)+11:IF A(X)=255 THEN 320
330 REM:BOTTOMLESS QUAGMIRE
340 A(X)=81:REM "Q"
350 FOR Y=1 TO 2
360 X=RND(77)+11:IF A(X)=255 THEN 360
370 REM:LOVELY GOLD
380 A(X)=71:REM "G"

```

```

390 NEXT:NEXT
400 E=55:REM YOUR LOCATION
410 ENDPROC
420 REM*****
430 DEF PROCstatus
440 A(E)=72:REM "H"
450 Q= RND(3)-1
460 IF Q=0 PROCmap
470 CLS
480 Q=1
490 PRINT '''CHR$(128+RND(5));CHR$(141);"You a
re in sector ";E
500 PRINT CHR$(128+RND(5));CHR$(141);"You are
in sector ";E'
510 IF G>0 PRINT CHR$(128+RND(5))"with Romland
gold worth '';G
520 PROCclues
530PRINT '''CHR$(128+RND(5))"You must endure f
or just ";25-H;" more hours"
540 X=128+RND(5)
550 PRINT 'CHR$(X);"Which direction do you wan
t"
560 PRINT TAB(5);CHR$(X);"to move (N, S, E or
W)"
570 Z$=GET$
580 IF Z$<>"N" AND Z$<>"S" AND Z$<>"E" AND Z$<
>"W" THEN 570
590 U=0:CLS
600 IF Z$="N" AND A(E-10)=255 THEN U=1
610 IF Z$="S" AND A(E+10)=255 THEN U=1
620 IF Z$="E" AND A(E+1)=255 THEN U=1
630 IF Z$="W" AND A(E-1)=255 THEN U=1
640 PRINT
650 IF U=0 THEN 690
660 PRINT CHR$(141);"Ha ha, he he, no exit the
re be!"
670 PRINT CHR$(141);"Ha ha, he he, no exit the
re be!"
680 GOTO 540
690 A(E)=46
700 IF Z$="N" THEN E=E-10
710 IF Z$="S" THEN E=E+10
720 IF Z$="E" THEN E=E+1
730 IF Z$="W" THEN E=E-1
740 IF A(E)=63 THEN PROCtunnel
750 IF A(E)=66 THEN PROCbeast
760 IF A(E)=81 THEN PROCquagmire
770 IF A(E)=71 THEN PROCgold

```



```

780 ENDPROC
790 REM*****
800 DEF PROCtunnel
810 K=2+RND(7)
820 FOR T=254 TO 1 STEP -2
830 SOUND 3,-15,T,1
840 PRINT TAB(0,20);CHR$(128+RND(5));"Oh! You'
ve fallen into a magic tunnel"
850 PRINT CHR$(128+RND(5));"You will be transp
orted to a random"
860 PRINT CHR$(128+RND(5));"part of Romland...
stand by..."
870 NEXT
880 A(E)=63
890 E= RND(77)+11: IF A(E)=255 THEN 890
900 ENDPROC
910 REM*****
920 DEF PROCbeast
930 FOR J=1 TO 24:PRINT CHR$(128+RND(5));"****
*****":NEXT
940 FOR Y=1 TO 2
950 FOR J=1TO30
960 SOUND 0,-15,RND(255),RND(3)
970 SOUND 1,-15,255-7*(INT(J/5)),3-Y
980 NEXT:NEXT
990 PRINT TAB(6,17);CHR$(141);"Warning!!":PRIN
T TAB(6);CHR$(141);"Warning!!"
1000 FOR J=1 TO 5
1010 SOUND 1,-3*J,60,30-2*J
1020 NEXT
1030 PRINT CHR$(129);"You have come across a"CH
R$(132)"RAM beastie!"
1040 FOR J=1TO500
1050 PRINT TAB(9,14);CHR$(128+RND(5));"Stand by
!!"
1060 NEXT
1070 CLS
1080 M=RND(10)
1090 IF M>3 THEN 1130
1100 PRINT '''CHR$(130)"Luckily for you, this o
ne is blind!"
1110 FOR Z=1 TO 5000:NEXT
1120 ENDPROC
1130 PRINT CHR$(131)'''"Oh no! It has seen you
, and"
1140 FOR J=1 TO 1000 STEP 10
1150 SOUND 1,-15,(J+1)/4,1
1160 NEXT

```

```

1170 IF M<9 THEN 1210
1180 PRINT(CHR$131)"decides to ignore you!"
1190 FOR Z=1TO1500:NEXT
1200 ENDPROC
1210 Q=3
1220 PRINT CHR$(129);CHR$(141);".....EATS
YOU....."
1230 PRINT CHR$(129);CHR$(141);".....EATS
YOU....."
1240 FOR Z=1TO1500:NEXT
1250 PROCend_of_the_world
1260 DEF PROCquagmire
1270 FOR J=1 TO 100
1280 PRINT CHR$(128+RND(5));".....OH N
O!....."
1290 SOUND 2,-15,255-J,1
1300 NEXT
1310 PRINT'''TAB(5);CHR$(141)"You're standin' i
n a quagmire"
1320 PRINT TAB(5);CHR$(141);"You're standin' in
a quagmire"
1330 PRINT'''CHR$(128+RND(5))".....Bye bye, suc
ker....."
1340 FOR J=1 TO 254
1350 SOUND 0,-15,J,1
1360 NEXT
1370 Q=3
1380 PROCend_of_the_world
1390 REM*****
1400 DEF PROCgold
1410 FOR J=1 TO 1000 STEP 5
1420 SOUND 1,-15,J/4,1
1430 SOUND 2,-15,J/5,1
1440 PRINT CHR$(128+RND(5))".....G
OLD.....";
1450 NEXT
1460 FOR J=1 TO 1000 STEP 17
1470 SOUND 1,-15,J/4,1:SOUND 2,-15,J/5,2
1480 PRINT TAB(14,10);CHR$(128+RND(5));CHR$(141
);"Gold!"
1490 PRINT TAB(14);CHR$(128+RND(5));CHR$(141);"
Gold!"
1500 NEXT
1510 FOR J=1TO40:PRINT:FOR Z=1TO60:NEXT:NEXT
1520 PRINT TAB(0,6);CHR$(128+RND(5))"You have f
ound one of the fabled"
1530 PRINT '''CHR$(128+RND(5));"lost gold mines
of Romland!"

```

```

1540 K=RND(100)+50
1550 PRINT' 'CHR$(128+RND(5));"You have found go
ld worth '";K;"!!!!"
1560 G=G+K
1570 FOR J=1TO100
1580 SOUND 1,-15,154+J,1
1590 SOUND 3,-15,100+J/2,1
1600 NEXT
1610 CLS
1620 ENDPROC
1630 DEF PROCclues
1640 L=46
1650 IF A(E-11)<>46 THEN L=A(E-11)
1660 IF A(E-10)<>46 THEN L=A(E-10)
1670 IF A(E-9)<>46 THEN L=A(E-9)
1680 IF A(E-1)<>46 THEN L=A(E-1)
1690 IF A(E+1)<>46 THEN L=A(E+1)
1700 IF A(E+9)<>46 THEN L=A(E+9)
1710 IF A(E+10)<>46 THEN L=A(E+10)
1720 IF A(E+11)<>46 THEN L=A(E+11)
1730 IF L=46 THEN 2040
1740 FORJ=1TO7
1750 FOR Z=1TO500:NEXT
1760 SOUND 1,-15,20*J,3
1770 NEXT
1780 IF RND(3)>1 THEN 1880
1790 FOR J=1TORND(9)
1800 PRINT 'CHR$(128+RND(5))"Stand by, intrepid
traveller..."
1810 PRINT CHR$(128+RND(5))"Here's a clue comin
' up..."
1820 SOUND 0,-15,RND(3),3:SOUND1,-15,RND(3),3
1830 FOR Z=1TO500:NEXT
1840 NEXT
1850 FOR J=1TO500STEP10
1860 SOUND 3,-15,J/2,1
1870 NEXT
1880 FORJ=7 TO 1 STEP -1
1890 PRINT CHR$(128+RND(5))"Clue...clue...clue!"
"
1900 FOR Z=1TO500:NEXT
1910 SOUND 1,-5,30*J,1
1920 NEXT
1930 CLS
1940 FOR J=1 TO 40
1950 PRINT TAB(0,9);CHR$(128+RND(5))"Somewhere
near you is ";
1960 IF L=255 PRINT "a No Exit"

```

```

1970 IF L=63 PRINT "a magic tunnel"
1980 IF L=66 PRINT "a RAM beastie"
1990 IF L=81 PRINT "a quagmire"
2000 IF L=71 PRINT "a gold mine"
2010 PRINT TAB(0,13);CHR$(128+RND(5));"Who woul
d believe it?"
2020 SOUND RND(3),-15,RND(128)+127,RND(3)
2030 NEXT
2040 ENDPROC
2050 DEFPROCend_of_the_world
2060 CLS
2070 IF Q=9 THEN 2100
2080 PRINT"CHR$(128+RND(5))"You just blew it,
buddy!"
2090 PRINT CHR$(128+RND(5))"You are remarkably
dead..."
2100 A(E)=72
2110 PRINT CHR$(128+RND(5))"You survived for ";
H;" hour";
2120 IF HK>1 PRINT"s" ELSE PRINT
2130 IF G>0 PRINT CHR$(128+RND(5))"and found go
ld worth "";G
2140 PROCmap
2150 FOR Z=1 TO 500 STEP 30
2160 SOUND 1,-15,Z/3,1
2170 NEXT
2180 IF Q=0 ENDPROC
2190 GOTO 2150
2200 DEF PROCmap
2210 PRINT TAB(11,8);CHR$(136);"Romland"
2220 PRINT TAB(11);CHR$(141);"*****"
2230 FOR J=0 TO 90 STEP 10
2240 M=A(J+1):N=A(J+2):O=A(J+3):P=A(J+4):Z=A(J+
5):R=A(J+6):S=A(J+7):T=A(J+8):U=A(J+9):V=A(J+10)
2250 PRINT TAB(7);J;CHR$(128+RND(5));CHR$(M);CH
R$(N);CHR$(O);CHR$(P);CHR$(Z);CHR$(R);CHR$(S);CH
R$(T);CHR$(U);CHR$(V)
2260 SOUND 3,-15,255-J,2
2270 NEXT
2280 FOR T=1TO9000:NEXT
2290 ENDPROC

```

Models A and B

Gomoku

The Game

This Gomoku, adapted from a program written by Graham Charlton, puts up a strong defence and will give you a good game. The aim of the game is simple – try to get five of your pieces (the H, for human) in a row or diagonally, while trying to block the computer ('C') from doing the same. The computer, of course, is trying to block your pieces while trying to build up its own row of five in any direction.

You move by entering the number along the side of the board (eg. 6) and then the number along the top (eg. 4) as one number (ie. 64). The piece will appear in position, and there is a brief pause while the computer works out a devastating response.

Note that if you keep on making bad entries the screen scrolls.

The Program

10 Title.

20 Sends action to the initialisation procedure.

30 Sends the computer to the procedure which prints out the first board.

50–110 This is the master REPEAT/UNTIL loop which controls the whole game.

130–190 This procedure, 'check', is called while the computer is considering its moves (from lines 590, 610, 790, 1090 and 1110) and also when it is checking to see how many in a row you have.

210–360 This procedure prints out the board, with line 290 determining which colour will be used for which piece.

380–490 This procedure accepts your move. Line 390 clears the buffer, and line 400 makes a brief sound so you know it is time for your move. Line 440 moves the print position up one line (using CHR\$(11)) and then overprints the words 'Please enter your move' with a line of spaces. Line 460 checks if the move is legal and, if it is not, sends the program back to line 410 to ask for a new move.

520–1000 This is the heart of the program, in which the computer looks at the board and decides on its move. Starting from where you last placed a piece (G), the computer searches around this location, calling up the 'check' procedure. If it finds (line 640) five

in a row, it sends the computer to the 'you_win' procedure. The rest of the procedure determines which move is best and, having made a decision, signals this to you with a short sound (line 990).

1020–1150 The computer checks its latest move, to see if this has completed a line of five in any direction and, if it has, sends the computer to the procedure 'i_win'.

1170–1600 This is the initialisation procedure. Line 1190 turns off the cursor, line 1200 sets up the arrays for the board (A) and the directions around a particular square (X). Lines 1300 to 1320 play three chords to start the game off, and lines 1360 and 1370 allow you to choose whether you will go first. If you want first move, line 1390 sends the computer to the end of the procedure (line 1450). If not, the computer chooses at random from 12 good opening moves (represented by the DATA statements in line 1460), generating a little bit of sound (line 1410) while doing so.

1480–1520 The procedure triggered when you win.

1540–1600 The procedure triggered when the computer wins.

Suggestion for improvement

- An option to start a new game or 'gracefully' exit.

The Listing

```
10REM**GOMOKU**
20PROCinitialise
30PROCboard
40REM*****
50REPEAT
60PROCplayer_move
70PROCboard
80PROCmove
90PROCboard
100PROCmove_two
110UNTIL FALSE
120REM*****
130DEF PROCcheck
140E=A
150E=E+N
160IFA(E)<>Z THEN190
170K=K+1
180GOTO150
190ENDPROC
200REM*****
210DEF PROCboard
220PRINT CHR$(30)'''
230PRINTCHR$(128+RND(6));" 1 2 3 4 5 6 7 8"
240PRINTCHR$(128+RND(6));"  -----"
```

```

250FORA=1TO8
260PRINTCHR$(131);A;
270FORB=2TO9
280M=A(A*10+B)
290F=-130*(M=67)-133*(M=72)-134*(M=46)
300      PRINTCHR$(F);CHR$(M);
310NEXTB
320PRINTCHR$(131);A
330NEXTA
340PRINTCHR$(128+RND(6));"  _____"
350PRINTCHR$(128+RND(6));"  1 2 3 4 5 6 7 8"
360ENDPROC
370REM*****
380DEF PROCplayer_move
390*FX 15,0
400SOUND2,-15,RND(50)+25,1
410PRINT'CHR$(128+RND(6));"Please enter your
move";
420INPUTG
430SOUND3,-15,RND(10)+128,2
440PRINTCHR$(11);"
"
450G=G+1
460IFG<12 OR G>89 OR A(G)<>ASC"." THEN 410
470PRINT TAB(22,11);CHR$(128+RND(6));"Please
stand by"
480Z=ASC"H"
490A(G)=Z
500ENDPROC
510REM*****
520DEF PROCmove
530PRINT TAB(22,11);"
"
540A=G
550L=0
560FORX=1 TO 4
570K=0
580N=X(X)
590PROCcheck
600N=-N
610PROCcheck
620IFK>L L=K
630NEXT X
640IFL>3 PROCyou_win
650T=1
660IFT<>2 Z=ASC"C"
670IFT=2 Z=ASC"H"
680G=0
690H=0

```

```

700L=0
710FORA=12TO 89
720M=0
730IFA(A)<>ASC". " THEN900
740FORX=1 TO 4
750K=0
760N=X(X)
770PROCcheck
780N=-N
790PROCcheck
800IFK>L H=0:L=K
810IFL<>K THEN860
820IFT=1 AND L<4 THEN860
830IFT=2 AND L<2 THEN860
840IFT=3 AND L<2 THEN860
850M=M+1
860NEXTX
870IFM<=H THEN900
880H=M
890G=A
900NEXTA
910IFH<>0 THEN980
920T=T+1
930IFT<>4 THEN660
940A=1
950G=RND(77)+12
960IFA(G)=ASC". " THEN980
970A=A+1: IFA<400THEN950
980A(G)=ASC"C"
990SOUND3,-15,RND(20),2
1000ENDPROC
1010REM*****
1020DEF PROCmove_two
1030Z=ASC"C"
1040A=G
1050L=0
1060FORX=1TO4
1070K=0
1080N=X(X)
1090PROCcheck
1100N=-N
1110PROCcheck
1120IFK>L L=K
1130NEXTX
1140IFL>3PROCi_win
1150ENDPROC
1160REM*****
1170DEF PROCinitialise

```



```

1180CLS
1190VDU23;B202;0;0;0
1200DIMA(100),X(4)
1210FORA=1T08
1220FORB=2T09
1230A(A*10+B)=ASC". "
1240NEXTB
1250NEXTA
1260FORQ=1T04
1270READF:X(Q)=F
1280NEXTQ
1290DATA 1,9,10,11
1300SOUND1,-15,29,8:SOUND2,-15,13,8:SOUND3,-15
,41,8
1310SOUND1,-15,5,8:SOUND2,-15,21,8:SOUND3,-15,
33,8
1320SOUND1,-15,29,8:SOUND2,-15,13,8:SOUND3,-15
,41,8
1330PRINT'''CHR$(131),"GOMOKU"
1340PRINT'''CHR$(129);"Do you want the first
move?"
1350PRINTTAB(12,12);CHR$(133);"Y or N"
1360Q$=GET$
1370IFQ$<>"N" AND Q$<>"Y" THEN 1360
1380CLS
1390IF MID$(Q$,1,1)<>"N" ENDPROC
1400FORJ=1TORND(12)
1410SOUND3,-15,256/12*J,3
1420READZ
1430NEXTJ
1440A(Z)=ASC"C"
1450ENDPROC
1460DATA34,35,36,44,45,46,47,54,55,56,57,66
1470REM*****
1480DEF PROCyou_win
1490REPEAT
1500PRINTCHR$(128+RND(6));"You win!";CHR$(11)
1510UNTIL FALSE
1520END
1530REM*****
1540ENDPROC
1550DEF PROCi_win
1560REPEAT
1570PRINTCHR$(128+RND(6));"I win!";CHR$(11)
1580UNTIL FALSE
1590END
1600ENDPROC

```

Models A and B

Safari

The Game

In this game, you are on safari in Darkest Africa. A herd of elephants charge at you. In your terror your only thought is to avoid their oncoming grey mass – you do not consider killing them.

In real life (that is, in computer terms) you are a little asterisk in the centre of the screen, and can move right and left with the cursor right and left keys. The oncoming elephants appear as blobs which roll up the screen towards you. As you flee, you leave a line of asterisks in your wake. This line undulates rather satisfactorily with every move.

A suitable message is printed at the centre of the screen when you are hit by an elephant; the screen flashes violently. A message telling you your degree of staying-power follows this.

The Program

20 Sets the computer to mode 4. Model B owners could alter the program to run in mode 0, which gives a bigger playing area.

30 Calls PROCinit, which carries out general housekeeping such as turning off the cursor and setting variables which govern the difficulty of the game.

40 Sets up the main REPEAT loop of the game.

50 Checks for a key press. The variable 'time_limit' was set in PROCinit so, by altering this value, you can change the speed of the game.

60 Clears the keyboard buffer, to ensure that no spurious characters are picked up; these could slow down the speed of the game.

70 Calls PROCmove, which moves the asterisk in response to any keys which may have been detected in line 50.

80 Calls PROCroad, which creates a new row of elephants. It is called 'road' because the row looks more like a section of pot-holed road in the final version than like a herd of elephants.

90 Increments 'goes', which is a variable containing the number of goes you have had; in other words, the number of opportunities you have had to move.

100 Carries out the above operations again, until a hit is registered. The variable 'hit' is set in PROCmove.

110 Prints a cry of anguish for your demise.

120 Restores the speed of repeating on the keyboard to normal. See the chapter on *FX calls in the User Guide.

130 Restores the cursor control keys to their normal functions.

140 Turns the cursor back on again.

150–200 Form a FOR loop which flashes the screen on and off. The calls to PROCdelay slow the flash down.

210 Clears the screen ready to print out the message describing your standard of evasion.

220 Switches the screen to blue writing on a green background.

230, 240 Print the message.

250 Ends the game.

270 Starts the definition of PROCinit.

280 Initialises the variable 'position', which is the displacement of the asterisk from the left-hand side of the screen.

290 Initialises the variable 'difficulty', which gives the difficulty of the game, in terms of the number of elephants charging during each scroll.

300 Initialises the variable 'time_limit', which is the argument of the INKEY\$ statement, and so governs the speed of the game.

310 Changes the cursor control keys to the mode where they give codes rather than just move the cursor about.

320 Stops the delay between a key being pressed and the key repeating.

330 Turns off the cursor.

340 Re-defines character 255 to be a block with rounded edges.

350 Changes the foreground colour to red, and the background colour to yellow.

360 Initialises the variable 'goes' to 0. This variable counts the number of moves you have made.

370 Ends PROCinit.

390 Starts the definition of PROCroad. This procedure prints out a new row of elephants and scrolls the screen.

400 Sets 'counter' to be a local variable.

410 Positions the cursor at the very bottom left of the screen.

420 Starts a loop, with 40 iterations, to print either elephants or spaces.

430 If a random number between one and 'difficulty' is one, prints an elephant; else, prints a space.

440 Ends loop.

450 Ends PROCroad.

470 Starts the definition of PROCmove.
This procedure moves you in response to the key-press detected in line 50.

480 If the cursor left key is pressed, moves left by decrementing the variable 'position'.

490 If the cursor right key is pressed, moves right by incrementing the variable 'position'.

500 If you move off the right-hand side of the screen, places you on the last position on the right.

510 If you have moved off the left-hand edge of the screen, this places you on the first position on the left.

520 This line checks to see if the character at the place you want to move to is an elephant. This is done by using the routine given in the chapter on *FX calls in the User Guide. If the character is an elephant, the variable 'hit' is set to true. You will remember that this variable is used in line 100 to sense when the game has finished.

530 This line prints the asterisk at the required place. Because the screen has been scrolled since this routine was last called, there is no need to rub out your old position.

540 Ends PROCmove.

560-630 Make up PROCreadch (X, Y). This routine is documented in the User Guide, in the chapter on *FX calls.

650 Starts the definition of PROCdelay, which gives a tenth of a second delay.

Suggestions for improvement

- The game as it stands has no sound effects. Maybe a blood-curdling roar would help to put you in the right frame of mind.
- You can use the character definition program on page 133 if you would like to make the elephants look a little more menacing.
- Modify the program to exit or restart 'gracefully' and also flush the key buffer.
- Change the colours to suit your eye.

The Listing

```
10 *KEY 0 "*FX 12:M*FX 4:M"
20 MODE 4
30 PROCinit
40 REPEAT
50 A$=INKEY$(time_limit)
60 *FX 15,0
70 PROCmove
80 PROCroad
90 goes=goes+1
100 UNTIL hit
110 PRINT TAB(5,20);"
Argghhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh!!!!"
120 *FX 12
130 *FX 4,0
140 !&FE00=&10670A
150 FOR T=1 TO 10
160 VDU 19,0,3,0,0,0,19,1,5,0,0,0
```

```

170 PROCdelay
180 VDU 19,1,3,0,0,0,19,0,5,0,0,0
190 PROCdelay
200 NEXT
210 CLS
220 VDU 19,1,4,0,0,0,19,0,2,0,0,0
230 PRINT """" You took ";goes;" steps,
before you""
240 PRINT "were trampled..."""
250 END
260 REM *****
*****
270 DEF PROCinit
280 position=20
290 difficulty=10
300 time_limit=3
310 *FX 4,1
320 *FX 11,1
330 VDU 23;8202;0;0;0;
340 VDU 23,255,&7C,&FE,&FE,&FE,&FE,&FE,&FE,&7
C
350 VDU 19,1,1,0,0,0,19,0,3,0,0,0
360 goes=0
370 ENDPROC
380 REM *****
*****
390 DEF PROCroad
400 LOCAL counter
410 PRINT TAB(0,31);
420 FOR counter=1 TO 40
430 IF RND(difficulty)=1 THEN VDU 255 ELSE
VDU 32
440 NEXT counter
450 ENDPROC
460 REM *****
*****
470 DEF PROCmove
480 IF A$=CHR$(136) THEN position=position-1
490 IF A$=CHR$(137) THEN position=position+1
500 IF position>39 THEN position=39
510 IF position<0 THEN position=0
520 hit=FNreadch(position,16)=255
530 PRINT TAB(position,16);""
540 ENDPROC
550 REM *****
*****
560 DEF FNreadch(X,Y)
570 LOCAL A%,C

```

```
580 VDU 31,X,Y
590 A%=135
600 C=USR(&FFF4)
610 C=C AND &FFFF
620 C=C DIV &100
630=(C MOD 32)+224
640 REM *****
*****
650 DEF PROCdelay
660 TIME=0
670 REPEAT UNTIL TIME>10
680 ENDPROC
690 REM *****
*****
```

Character Definition

The Program

This program allows you to re-define characters 224 to 255, building up a character on the screen using the cursor control keys. Full instructions are included in the program.

In this version, the program re-defines the character you requested, and then ends. You may find it more convenient to have it actually print out the VDU statement required, which you can then copy down and include in the PROCinit part of our games in this book, to enhance the displays we've used. Your own programs may well benefit from such re-definition as well.

Remember, you cannot redefine characters in Mode 7.

- 30 Starts a REPEAT loop to get a valid character for re-definition.
- 40 Clears the screen to Mode 7.
- 50 Calls PROCask_questions, which displays instructions; and asks you which character you wish to re-define. This character is returned in the variable 'character'.
- 60 Ends the REPEAT loop when the character falls within an accepted range.
- 70 Calls PROCdraw_character, which draws a blank character grid on the screen, ready to be used as a template for the new character design.
- 80 Starts a REPEAT loop for the main program.
- 90 Gets a key press from you.
- 100 Calls PROCact_onkey_pressed, which does just that.
- 110 Ends the loop when TAB is pressed.
- 120 Calls PROCassemble, which has nothing to do with assembly language but reads the character from the screen and sends the correct codes via the VDU statement to redefine character 'character'.
- 130 Returns the cursor control keys to their normal function.
- 140 Ends the program.
- 160 Starts the definition of PROCask_questions.
- 170 Gets the actual character for re-definition.
- 180 Prints the instructions.
- 190 Requests you to press a key.
- 200 Gets the key pressed.
- 210 Ends PROCask_questions.

230 Starts the definition of PROCdraw_character.
250 Clears the screen, to remove the text generated by PROCask_ questions.
260 Moves the cursor down the screen. The number of lines it is moved is crucial to the operation of the program.
270 Starts a loop for each of the eight character rows to be printed.
280 Prints eight full stops preceded by *three* spaces for each of the eight rows.
290 Ends the loop.
300 Prints instructions for ending the program.
310 Initialises the x starting position of the cursor.
320 Initialises the y starting position of the cursor.
330 Changes the cursor control keys so that they generate codes rather than shift the cursor around.
340 Ends PROCdraw_character.

360 Starts the definition of PROCact_onkey_pressed.
370 Sets the LOCAL variables.
380 Initialises valid\$, which contains all the valid key presses with codes under 128. This is used to ensure that any key can be used to make a pixel light up.
390 Sets a Boolean variable 'on_template' to be TRUE or FALSE, depending on whether the cursor is on the template at the moment.
400 If the Space Bar was pressed, the next statement uses the variable 'on_template' to decide whether to print a full stop or a space, and the x co-ordinate of the cursor is incremented.
410 If the key pressed was RETURN, and the cursor is not on the bottom line of the display, then returns the cursor to the start of the next line and adjusts the co-ordinates of the cursor accordingly.
420 If the key pressed had an ASCII code less than 128 and was not a space or a RETURN character, either prints a space or a white block, depending on whether the cursor is on the template or not. All co-ordinates are similarly adjusted.
430 If the key pressed was 'cursor left' then moves the cursor left and adjusts the x co-ordinate of the cursor.
440 If the key pressed was 'cursor right' then moves the cursor right and adjusts the x co-ordinate of the cursor.
450 If the key pressed was 'cursor down' then moves the cursor down a line and adjusts the y co-ordinate of the cursor.
460 If the key pressed was 'cursor up' then moves the cursor up and adjusts the y co-ordinate of the cursor.
470 Ends PROCact_onkey_pressed.

490 Starts the definition of PROCassemble.
500 Sets all the LOCAL variables needed.
510 Sends the first part of the required VDU statement.
520 For each of the eight rows in the character ...
530 works out the start address of that row ...
540 resets a counter ...

550 and for each of the eight bits in the row ...
 560 increments the counter if necessary ...
 570 then ends the bit count ...
 580 sends the right byte to the VDU driver ...
 590 and gets the rest of the rows.
 600 Ends PROCassemble.

Suggestions for improvement

- The modifications for a print-out of the required VDU instruction are as follows:

```
510 PRINT TAB(0, 15);"The required instruction is:
VDU 23,";character;",";
580 PRINT temp;",";
595 VDU 127, 13
```

The Listing

```
10 REM Re-defining characters.
20 REM *****
*****
30 REPEAT
40 MODE 7
50 PROCask_questions
60 UNTIL character>223 AND character<256
70 PROCdraw_character
80 REPEAT
90 key_pressed$=GET$
100 PROCact_onkey_press
110 UNTIL key_pressed$=CHR$(9)
120 PROCassemble
130 *FX 4,0
140 END
150 REM *****
*****
160 DEF PROCask_questions
170 INPUT TAB(0,5)" Enter the number of the
character to be re-defined ? "character
180 PRINT " Use the cursor keys to move
the cursor around the character. Pressing any
keyexcept space will make a dot white,
pressing space will blank out a dot.
Return' will move the cursor to the start of
the next line."
190 PRINT TAB(0,24);CHR$(132);"*** Press '
space' to continue ***";
200 A$=GET$
210 ENDPROC
220 REM *****
*****
```

```

230 DEF PROCdraw_character
240 LOCAL row
250 CLS
260 PRINT ' '
270 FOR row=1 TO 8
280 PRINT "  ...."
290 NEXT row
300 PRINT TAB(0,24);CHR$(132);"*** Press '
tab' to finish ***";TAB(0,10);
310 xposition=0
320 yposition=10
330 *FX 4,1
340 ENDPROC.
350 REM *****
*****
360 DEF PROCact_onkey_press
370 LOCAL valid$,on_template
380 valid$=" "+CHR$(13)
390 on_template=xposition>2 AND xposition<11
AND yposition>2 AND yposition<11
400 IF key_pressed$=" " THEN PRINT CHR$(ASC("
")-on_template*14);:xposition=xposition+1
410 IF key_pressed$=CHR$(13) AND yposition<24
THEN PRINT:yposition=yposition+1:xposition=0
420 IF key_pressed$<CHR$(128) AND INSTR(
valid$,key_pressed$)=0 THEN VDU 32-on_template*
223:xposition=xposition+1
430 IF key_pressed$=CHR$(136) THEN VDU 8:
xposition=xposition-1
440 IF key_pressed$=CHR$(137) THEN VDU 9:
xposition=xposition+1
450 IF key_pressed$=CHR$(138) AND yposition<2
4 THEN VDU 10:yposition=yposition+1
460 IF key_pressed$=CHR$(139) AND yposition>0
THEN VDU 11:yposition=yposition-1
470 ENDPROC
480 REM *****
*****
490 DEF PROCassemble
500 LOCAL origin,row,bit,temp
510 VDU 23,character
520 FOR row=0 TO 7
530 origin=&7C7B+(character-224)*8+row*40
540 temp=0
550 FOR bit=7 TO 0 STEP -1
560 IF ?(origin+7-bit)=255 THEN temp=temp+2
^bit
570 NEXT bit

```

```
580 VDU temp
590 NEXT row
600 ENDPROC
610 REM *****
*****
```

Graphic Displays

We now come to a series of demonstrations which show off the BBC microcomputer's graphics splendidly. The descriptions for these programs are not as extensive as for some of the others but are detailed enough for you to understand how the programs work, and – more importantly – how you can modify them to create your own graphics demonstrations.

Note You can also improve the programs by doing things such as exiting 'gracefully' and putting the computer into Mode 7.

Model B

String Art

String art patterns are created by bouncing two points around the screen and continually joining the points together with straight lines. It is a simple idea but it produces startling results. This particular program allows for random colour changes, and changes of direction, without having the dots actually bounce off the edge of the screen. Also, only a limited number of lines are kept on the screen at a time. For every line drawn, another is erased.

Line 10 dimensions an array to hold the starting and finishing x and y co-ordinates of up to 200 lines. Line 20 sets the number of lines that will be kept on the screen at a time. Lines 30 and 40 choose random end-points for the first line. Line 50 chooses white as the first drawing colour. Line 60 calls PROCnew_velocities, which assigns different velocities to each end of the line in the x and y directions. It also chooses the number of lines that will be drawn using these velocities. Line 70 puts the computer in Mode 1. Model A owners could run this program in Mode 5, although it is not as effective as running it in Mode 1. Then colour 3 is changed to blue. The call to PROCnew_colour chooses a random plotting colour and also the number of lines that will be drawn using that colour.

Line 100 starts the main REPEAT loop of the program. This is an 'UNTIL FALSE' loop, so it executes for ever. First a loop is set up through all the available lines. Lines 120 and 130 erase the line defined by the current element of the array. If the loop is executing for the first time, the array will be filled with zeros, so all this line will do is put a black spot at co-ordinates (0, 0). Lines 140 and 150 draw the current line, then the start and finishing points of this line are copied into the array at lines 160 to 190. Lines 200 to 230 compose a simple test to stop the pattern going off the edge of the screen. Tests like these are always very important.

In line 240 and 250 the two variables decremented are the number of lines drawn with the current velocities and the number drawn in the current colour, respectively. Lines 260 and 270 update the line co-ordinates, according to the current velocities. Lines 290 and 300 get new colours or velocities, if necessary. The FOR loop then ends, and the UNTIL FALSE line takes over.

Lines 340 and 350 are set out to enable you to change the

maximum velocity. Just make U% equal to the maximum velocity, and V% be twice U%.

The Listing

```
10 DIM M%(200,3)
20 H%=50
30 X%=RND(1280)-1:Y%=RND(1024)-1
40 L%=RND(1280)-1:M%=RND(1024)-1
50 COL%=3
60 PROCnew_velocities
70 MODE 1
80 VDU 19,COL%,4,0,0,0
90 PROCnew_colour
100 REPEAT
110 FOR U%=1 TO H%
120 MOVE M%(U%,0),M%(U%,1)
130 PLOT 7,M%(U%,2),M%(U%,3)
140 MOVE X%,Y%
150 DRAW L%,M%
160 M%(U%,0)=X%
170 M%(U%,1)=Y%
180 M%(U%,2)=L%
190 M%(U%,3)=M%
200 IF A%+X%>1279 OR A%+X%<0 THEN A%=-A%
210 IF B%+Y%>1023 OR B%+Y%<0 THEN B%=-B%
220 IF C%+L%>1279 OR C%+L%<0 THEN C%=-C%
230 IF D%+M%>1023 OR D%+M%<0 THEN D%=-D%
240 N%=N%-1
250 COLN%=COLN%-1
260 X%=X%+A%:Y%=Y%+B%
270 L%=L%+C%:M%=M%+D%
280 IF N%=0 THEN PROCnew_velocities
290 IF COLN%=0 THEN PROCnew_colour
300 NEXT U%
310 UNTIL FALSE
320 DEF PROCnew_velocities
330 LOCAL U%,V%
340 U%=20
350 V%=40
360 A%=U%-RND(V%):B%=U%-RND(V%)
370 C%=U%-RND(V%):D%=U%-RND(V%)
380 N%=RND(30)+20
390 ENDPROC
400 DEF PROCnew_colour
410 COLN%=RND(30)+10
420 IF COL%=1 THEN ENDPROC
430 GCOL 0,RND(COL%)
440 ENDPROC
```

Models A and B

Cat o'Six Tails

This program draws psychedelic curves on the screen, with a musical accompaniment of sorts. It runs on a Model A or a Model B, and is intended more to inform than impress.

PROCCHANGE at line 160 changes the background and foreground colours of the screen, randomly, but with a test to ensure that the colours are not the same. PROCNOTE at line 260 uses the SOUND statement to play a nearly identical note through each of the three tone channels. The effect of this is to give a richer sound than the tones your computer normally produces. '16' has been added to each channel number to ensure all the sound queues are cleared before the note is played.

The rest of the program is made up of a loop through all the horizontal points on the screen, followed by some lines to draw lines whose lengths are proportional to the SINE of X. Line 140 calls PROCCHANGE every second to alter the screen colours.

The Listing

```
10 REM Cat'o'six tails
20 MODE 4
30 TIME=0
40 FOR X=0 TO 1279 STEP 4
50 Y=SIN(RAD(X))*X/4
60 PROCNOTE(SIN(RAD(X))*180+50)
70 FOR G=-200 TO 1000 STEP 200
80 MOVE X,Y+G
90 DRAW X,Y+G+(1279-X)/6.4
100 IF TIME>100 THEN PROCCHANGE
110 NEXT G
120 NEXT X
130 REPEAT
140 IF TIME>100 THEN PROCCHANGE
150 UNTIL FALSE
160 DEF PROCCHANGE
170 LOCAL C,D
180 C=RND(6)
190 REPEAT
200 D=RND(6)
```

```
210 UNTIL C<>D
220 VDU 19,0,C,0,0,0,19,1,D,0,0,0
230 TIME=0
240 ENDPROC
250 DEF PROCNOTE(N)
260 SOUND 17,-15,N,255
270 SOUND 18,-15,N+1,255
280 SOUND 19,-15,N+2,255
290 ENDPROC
```


Model B

Chess Art

This program is written to run on a Model B in Mode 1, but could be run on a Model A in Mode 5. It generates string art patterns – with a twist.

The screen is first filled with a chessboard pattern in colours 1 and 0 (PROCSTRIPE); then colour 1 is turned to black using the VDU 19 statement in line 150. Then a normal string art pattern is drawn, except the plotting colour is '1,2'. This means 'OR the colour specified with the colour already there'. The upshot of this is that if a spot, previously over a section of the screen which was colour 0, is lit, it will now appear as colour 2; colour 1 will appear as colour 3.

The end result is that the lines drawn change according to their position. If you wait long enough, the whole screen will fill up with lines, showing an overall chessboard pattern. To make the drawing start from a new position with new velocities, just press any key. This version does not do any undrawing, so you'll have to stop it eventually.

The interesting routines from the point of view of programming routines are PROCSTRIPE and lines 100 to 130. Lines 100 to 130 choose two (unequal) numbers between 1 and 7, and then use them as the two plotting colours.

PROCSTRIPE uses GCOL with a number other than 0 as its first argument to fill various graphics windows with colour. This is much faster than using triangle drawing commands like PLOT 85.

The Listing

```
10 REM "Chess-art"  
20 U=20  
30 V=40  
40 MODE 1  
50 VDU 19,0,0,0,0,0,0,19,1,0,0,0,0  
60 GCOL 1,2  
70 REPEAT  
80 COLOUR 128  
90 CLS  
100 REPEAT  
110 AA=RND(7)  
120 BB=RND(7)
```

```

130 UNTIL AA<>BB
140 PROCSTRIPE
150 VDU 19,0,0,0,0,0,19,1,0,0,0,0
160 VDU 19,2,AA,0,0,0,19,3,BB,0,0,0
170 REPEAT
180 AA=RND(7)
190 BB=RND(7)
200 UNTIL AA<>BB
210 X=RND(1280)-1:Y=RND(1024)-1
220 L=RND(1280)-1:M=RND(1024)-1
230 A=U-RND(V):B=U-RND(V)
240 C=U-RND(V):D=U-RND(V)
250 REPEAT
260 MOVE X,Y
270 DRAW L,M
280 IF X+A>1279 OR X+A<0 THEN A=-A
290 IF Y+B>1023 OR Y+B<0 THEN B=-B
300 IF L+C>1279 OR L+C<0 THEN C=-C
310 IF M+D>1023 OR M+D<0 THEN D=-D
320 X=X+A:Y=Y+B
330 L=L+C:M=M+D
340 UNTIL INKEY(1)<>-1
350 *FX 15
360 UNTIL FALSE
370 DEF PROCSTRIPE
380 GCOL 0,129
390 FOR T=0 TO 1023 STEP 256
400 VDU 24,0;T;1279;T+128;
410 CLG
420 NEXT T
430 GCOL 3,129
440 FOR T=0 TO 1279 STEP 256
450 VDU 24,T;0;T+128;1023;
460 CLG
470 NEXT T
480 VDU 26
490 ENDPROC

```

Models A and B

Prison Bars

This program is the forerunner of 'Chessart'. It draws 'moiré' patterns, with a foreground achieved in the same way as 'Chessart'. Model A users can modify lines 30, 100 and 210 to run this program.

The moiré patterns are created by using the inverse line drawing commands. A point is chosen, then lines are inverted out from it to the edge of the screen. The completed pattern looks as if it was drawn using curved lines, which can be puzzling for those who do not know the method by which it was done.

The program runs continuously, clearing the screen after each pattern, then choosing new colours and a new starting point.

The Listing

```
10 REM *** Prison bars ***
20 REM Model 'B' version
30 MODE 1
40 VDU 19,1,0,0,0,0
50 REPEAT
60 CLS
70 X=RND(1280)-1
80 Y=RND(1024)-1
90 COLOUR 129
100 FOR T=4 TO 34 STEP 10
110 VDU 28,T-4,31,T,0
120 CLS
130 NEXT T
140 COLOUR 128
150 VDU 26
160 REPEAT
170 A=RND(7)
180 B=RND(7)
190 UNTIL A<>B
200 VDU 19,2,A,0,0,0,19,3,B,0,0,0
210 FOR T=0 TO 1279 STEP 8
220 MOVE T,1023
230 PLOT 6,X,Y
240 PLOT 6,1280-T,0
250 NEXT T
```

```
260 FOR T=0 TO 1023 STEP 4
270 MOVE 0,T
280 PLOT 6,X,Y
290 PLOT 6,1279,1024-T
300 NEXT T
310 TIME=0
320 REPEAT UNTIL TIME>100
330 UNTIL FALSE
```

Models A and B

Eight o'Clock

This program draws eight circles, connected together in the middle of the screen, using straight lines. When this has been done, the whole screen tosses back and forth, like the effect at the end of the 'Rain Catcher' program.

The basis of the routine is the procedure at line 190. This draws a filled-in circle with its centre at X, Y and with radius R.

The lines from 80 to 170 throw the screen around. Alter line 120 to change the speed at which this happens.

The Listing

```
10 REM "Eight o'clock"
20 MODE 4
30 S=400:D=50
40 FOR A=0 TO 315 STEP 45
50 MOVE 640,512
60 PROCCIRCLE(SIN(RAD(A))*S+640,COS(RAD(A))*
S+512,D)
70 NEXT A
80 REPEAT
90 FOR T=0 TO 39
100 VDU 23,0,13,T,0,0,0,0,0,0,0,0,0,0
110 TIME=0
120 REPEAT UNTIL TIME>3
130 NEXT T
140 FOR T=39 TO 0 STEP -1
150 VDU 23,0,13,T,0,0,0,0,0,0,0,0,0,0
160 NEXT T
170 UNTIL FALSE
180 UNTIL FALSE
190 DEF PROCCIRCLE(X,Y,R)
200 LOCAL step,angle
210 step=5
220 FOR angle=0 TO 360-step STEP step
230 DRAW X,Y
```

```
240 MOVE SIN(RAD(angle))*R+X,COS(RAD(angle))*  
R+Y  
250 PLOT 85,SIN(RAD(angle+step))*R+X,COS(RAD(  
angle+step))*R+Y  
260 NEXT angle  
270 ENDPROC
```

Models A and B / Model B

Polar Plotting

The following two programs draw flower-shaped patterns. Number one is for Model A and Model B owners; number two may only be run by Model B owners. The programs have a heavy mathematical colouring but this is not the place to discuss polar plotting.

If you alter the multiplying factor in line 90, you will get a different number of petals. If you're feeling really adventurous, you can even alter the function in line 90 altogether. The function should compute in the range 0 to 500 as X varies from 0 to 360. The best way to alter it is by trial and error. If the pattern goes off the edge of the screen, just reduce the multiplying factor.

The Listings

```
10 REM Polar plot -- mark 1
20 MODE 4
30 VDU 29,640;512;
40 MOVE 0,FNF(0)*500
50 FOR A=0 TO 360
60 DRAW SIN(RAD(A))*FNF(A),COS(RAD(A))*FNF(A)
70 NEXT A
80 END
90 DEF FNF(X)=500*SIN(RAD(X*8))
```

```
10 REM Polar plot -- mark 11
20 MODE 2
30 VDU 29,640;512;
40 MOVE 0,FNF(0)*500
50 FOR A=0 TO 360
55 MOVE 0,0
56 GCOL 0,RND(7)
60 DRAW SIN(RAD(A))*FNF(A),COS(RAD(A))*FNF(A)
70 NEXT A
80 END
90 DEF FNF(X)=500*SIN(RAD(X*8))
```

Model B

Cartwheel

This is strictly for Model B owners, since it runs in Mode 2.

A solid circle is drawn, occupying nearly the whole screen. The circle is coloured in segments, going from colour 1 to colour 15 and back to 1 again. Each segment is 10° wide, so there are 36 of them. When all the segments have been drawn, all the 15 colours used are turned to black, using the VDU 19 statement. Then, in rotation, each colour is changed to blue, and back to black again. The total effect is like a cartwheel rotating extremely fast, much faster than could be achieved by using repeated PLOT 85 statements.

If you adjust the number in line 40, you can alter the speed at which rotation takes place.

The Listing

```
10 REM This program generates a cartwheel
effect in Mode 2
20 MODE 2
30 step=6
40 time=10
50 FOR angle=0 TO 359 STEP step
60 MOVE 640,512
70 MOVE SIN(RAD(angle))*500+640,COS(RAD(
angle))*500+512
80 GCOL 0,((angle/step) MOD 15)+1
90 PLOT 85,SIN(RAD(angle+step))*500+640,COS(
RAD(angle+step))*500+512
100 NEXT angle
110 FOR colour=1 TO 15
120 VDU 19,colour,0,0,0,0
130 NEXT colour
140 REPEAT
150 FOR colour=1 TO 15
160 VDU 19,colour,4,0,0,0
170 TIME=0
180 REPEAT UNTIL TIME=time
190 VDU 19,colour,0,0,0,0
200 NEXT colour
210 UNTIL FALSE
```


Models A and B

Quadrant String Art

This program is a further variation on the string art theme. It draws common-or-garden patterns, but only uses one quarter of the screen. The other three quarters contain reflections of the first quarter. There are also some refinements in the picture control over the other versions: pressing any key brings you into 'hold mode', where no drawing takes place. You can come out of this mode by either pressing TAB to make the drawing start again from a CLS, with new starting positions and new velocities; or any other key to just restart drawing where it left off.

Note You should put the computer in a suitable graphics mode before running.

The only noticeably clever part of this routine is the procedure at line 240. This procedure joins X, Y to L, M, using PLOT K. It does this in all four quadrants of the screen, using a couple of loops. Bear in mind that the origin has been moved at line 110 to the centre of the screen.

The Listing

```
10 REM String art pattern generator, in
four quadrants of the screen.
20 REM This will run in any graphics mode,
but modes 0 and 4 are best.
30 REPEAT
40 A$=""
50 CLS
60 U=20:V=40
70 X=RND(640)-1:Y=RND(512)-1
80 L=RND(640)-1:M=RND(512)-1
90 A=U-RND(V):B=U-RND(V)
100 C=U-RND(V):D=U-RND(V)
110 VDU 29,640;512;
120 REPEAT
130 PROCDRAW(X,Y,L,M,5)
140 IF (X+A)>639 OR (X+A)<0 A=-A
150 IF (Y+B)>511 OR (Y+B)<0 B=-B
160 IF (L+C)>639 OR (L+C)<0 C=-C
```

```

170 IF (M+D)>511 OR (M+D)<0 D=-D
180 X=X+A:Y=Y+B
190 L=L+C:M=M+D
200 IF INKEY$(0)<>" " THEN A$=GET$
210 UNTIL A$=CHR$(9)
220 UNTIL FALSE
230 REM *****
*****
240 DEF PROCDRAW(X,Y,L,M,K)
250 LOCAL ones,twos
260 FOR ones=-1 TO 1 STEP 2
270 FOR twos=-1 TO 1 STEP 2
280 MOVE ones*X,twos*Y
290 PLOT K,ones*L,twos*M
300 NEXT
310 NEXT
320 ENDPROC

```

Model B

Oval

This program will only run on a Model B, since it runs in mode 2.

It draws two sets of ovals, side by side on the screen. These ovals are in different colours, each being made up of colours 2 to 15, overlapping and getting smaller towards the centre. The colours are in a different order for the two ovals - the one on the right has them going in ascending order from the outside, and the one on the left has them in descending order.

Once the two ovals have been set up, the remainder of the program is much like 'Cartwheel', in that the colours are switched from black to another colour and back again.

The Listing

```
10 MODE 2
20 P=5
30 C=4
40 FOR T=14 TO 1 STEP -1
50 FOR G=0 TO 1
60 IF G=0 THEN GCOL 0,T ELSE GCOL 0,15-T
70 PROCcircle(320+640*G,512,T*30)
80 NEXT G
90 NEXT T
100 VDU 19,15,1,0,0,0,0
110 VDU 5
120 GCOL 0,15
130 MOVE 200,512
140 PRINT "BBC Computer"
150 REPEAT
160 FOR T=1 TO 14
170 *FX 15,0
180 SOUND 1,-15,T*10+100,255
190 SOUND 2,-15,T*10+101,255
200 SOUND 3,-15,T*10+102,255
210 VDU 19,T,C,0,0,0
220 TIME=0
230 REPEAT UNTIL TIME>P
240 NEXT T
```

```

250 FOR T=14 TO 1 STEP -1
260 *FX 15,0
270 SOUND 1,-15,T*10+100,255
280 SOUND 2,-15,T*10+101,255
290 SOUND 3,-15,T*10+102,255
300 VDU 19,T,C,0,0,0
310 TIME=0
320 REPEAT UNTIL TIME>P
330 VDU 19,T,0,0,0,0
340 NEXT T
350 C=1+RND(6)
360 UNTIL FALSE
370 DEF PROCcircle(xco,yco,radius)
380 LOCAL angle,step
390 step=15
400 FOR angle=0 TO 360 STEP step
410 MOVE SIN(RAD(angle))*radius/2+xco,COS(
RAD(angle))*radius+yco
420 MOVE SIN(RAD(angle+step))*radius/2+xco,
COS(RAD(angle+step))*radius+yco
430 PLOT 85,xco,yco
440 NEXT angle
450 ENDPROC

```

Models A and B

Goodbye

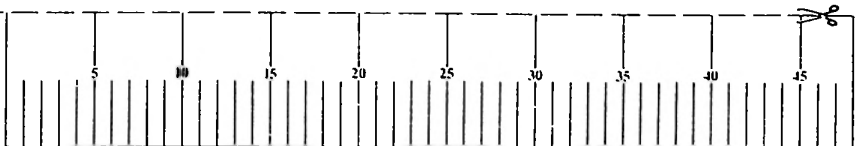
This program is our final fling. There are no line-by-line notes, no suggestions for improvement. Key it into your model A or B, type RUN; sit back. We think you will be surprised at the result.

```
10 MODE 7:VDU 23;B202;0;0;0;:R%=128+RND(5):
FORT%=0T023:VDU31,0,T%,R%:NEXTT%
20 DATA AA,AA,AA,41,41,88,88,14,14,88,08,41,41,
A0,82,50,04,AA,AB,55,00
30 DATA AA,80,55,00,2A,80,15,40,0A,A0,05,50,
02,AB,01,14,02,22,04,41,AA,A0
40 DIMC%(17,21):FOR Y%=1T021:FORT%=0T01:
READA$:A%=EVAL("&"+A$):FORG%=0T07
50 IF (A%AND2^G%)<>0THENC%(T%*8+8-G%,Y%)=1
60 NEXTG%,T%:IF (Y%MOD2)=1THENC%(17,Y%)=1
70 NEXTY%:FORT%=1T0357:REPEATX%=RND(17):Y%=
RND(21):UNTILC%(X%,Y%)<>2
80 IFC%(X%,Y%)=1THENVDU31,X%+10,Y%+2,255
90 C%(X%,Y%)=2:TIME=0:REPEATUNTILTIME>10:
NEXTT%:A%=INKEY(1000):CLS:CH=&BF00
100 PRINT TAB(0,24);CHR$(132);"Wait a little.
..";
110 FORT=0T014:READA:X=(T MOD 5)*8:Y=(T DIV 5
)*8
120 F$=CHR$(128):B$=CHR$(128+RND(5)):C$=CHR$(
A):FORR=0T07:VDU31,X,Y+R
130 FORC=7T00STEP-1:IF (? (CH+ASC(C$)*8+R)AND2
^C)=2^C PRINTF$;ELSEPRINTB$;
140 NEXTC,R,T:PRINTTAB(0,24);STRING$(39," ");
:REPEAT:X=RND(40)-1:Y=RND(25)-1:REPEAT:L=SGN(
RND):M=SGN(RND)
150 UNTILL<>0ANDM<>0:REPEAT:P=HIMEM+Y*40+X:T=
?P:?P=42:TIME=0:REPEAT
160 UNTILTIME>3:IFT=128THENT=255
170 ?P=T:X=X+L:Y=Y+M:UNTILX<0ORY<0ORX>39ORY>2
4:UNTILFALSE
180 DATA 32,84,104,101,32,32,111,119,108,32,1
14,117,108,101,115
```

Typing in Program Listings

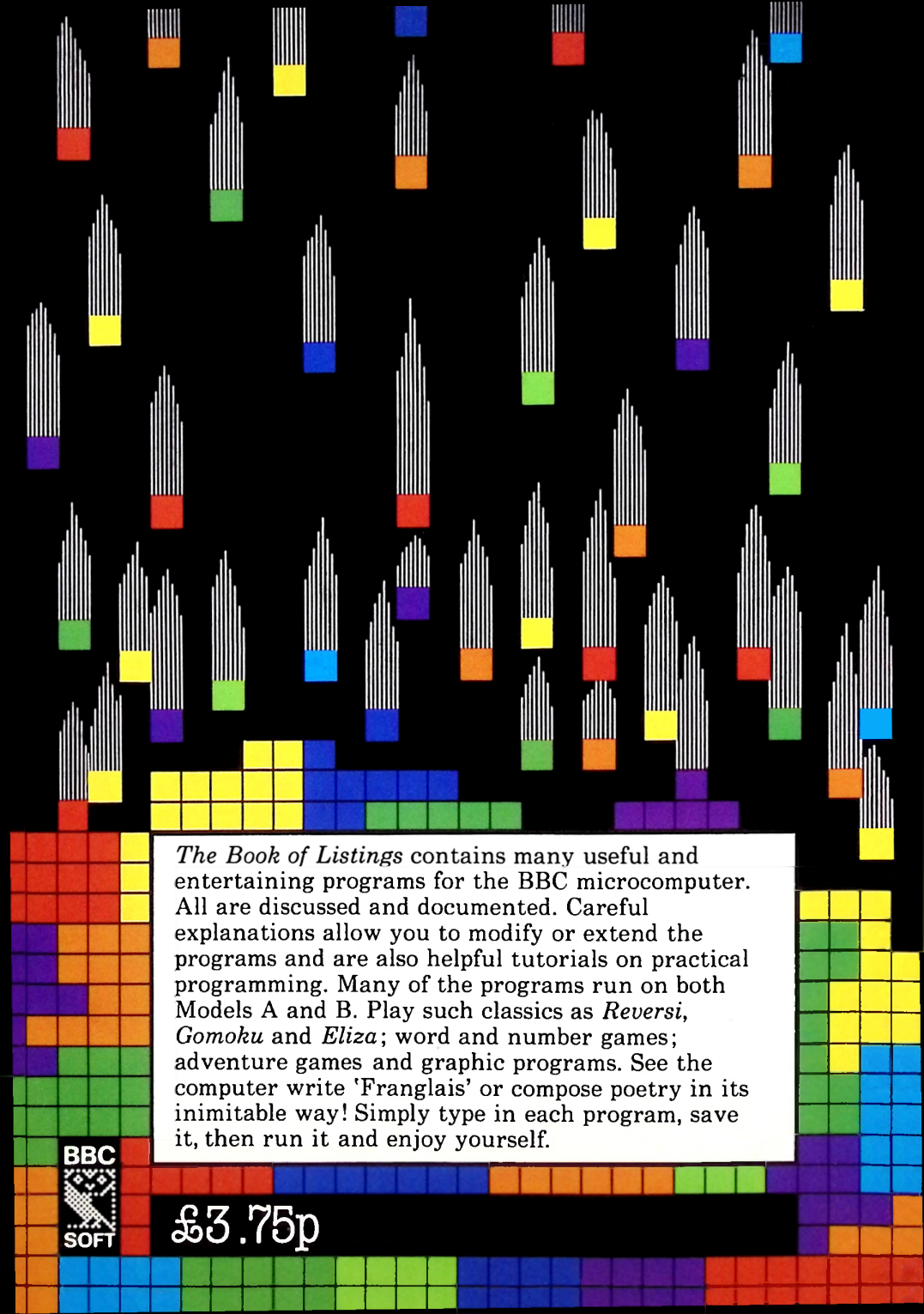
- When you type in a statement, press the RETURN key only when you come to the end of the complete statement line, not when you come to the end of a line on the printed page.
- If the statement line is simply a message and you do not type it in exactly as you see it (for example, you do not preserve the number of spaces between words), the program will still run though the message will not look very pretty.
- Sometimes spaces are vital in BASIC words. For example, the first bracket after TAB must follow on immediately without a space. When in doubt, consult the *User Guide*.
- Where a 'picture' is being drawn on the screen using symbols such as '+', it is important to type in what you see accurately but not literally. Spaces are very important here, and you can count them with the aid of the character count scale on page 157, which may be cut out for use to make it portable.

Character Count Scale (cut out to use)



NOTES

NOTES



The Book of Listings contains many useful and entertaining programs for the BBC microcomputer. All are discussed and documented. Careful explanations allow you to modify or extend the programs and are also helpful tutorials on practical programming. Many of the programs run on both Models A and B. Play such classics as *Reversi*, *Gomoku* and *Eliza*; word and number games; adventure games and graphic programs. See the computer write 'Franglais' or compose poetry in its inimitable way! Simply type in each program, save it, then run it and enjoy yourself.

BBC
SOFT

£3.75p