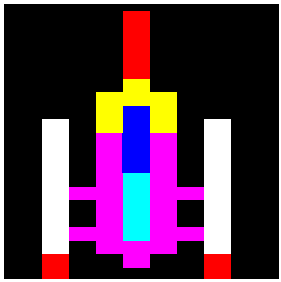

14

EXAMPLE GAME

This chapter is entirely devoted to describing one game. The program is only the very bare bones of a game, but it shows the principles upon which a game can be built.

The program is highly structured. This (hopefully) makes it very east to read and adapt. As it stands, the game is not particularly playable. This is due to a number of things:

1 Lack of high scores and sound effects.
2 Lack of originality.
3 Lack of speed.

Lack of sound can be overcome by using the sound routine in section 13.1, when in assembler. A high score routine is very easy to write, and even a table of high scores should present no real problem.

The second problem with the game is that it is unexciting. This is because thinking of new ideas is probably as difficult as writing a game, which is why most games are copies of others.

The final problem can be overcome by either writing the entire program in assembler or writing the

BASIC sections in optimised BASIC (single-letter integer variables, no spaces, multi-line statements, etc).

The first of these methods is undoubtedly the better of the two. If all of the program is written in assembler, the speed problem disappears, and you will find that you can probably handle about twenty shapes before the program slows down to the same speed as the original BASIC one. Also, writing the program in assembler means that you can preserve the structure, and, because there is no speed disadvantage in using long variable names, the program can be largely self-commenting.

The game has been written in such a way that it can be easily converted into assembler. All the variables are treated as integers and none of them has a range which exceeds a single byte. In addition, none of the procedures has more than two parameters and any parameters passed to them are only single bytes. This means that the procedures can be replaced by subroutines and the parameters can be passed in the registers.

The game has been listed mainly in BASIC rather than assembler because the assembler version would take up double the amount of room. Converting it has been left as an exercise for the reader.

Now for the listing:

```
    0 REM Demonstration Game For Book
   10
   20 REM Space Invader (!!)
   30
+-----------------------------------------------------+
| Top level of program                                |
+-----------------------------------------------------+
   40 MODE2:VDU23;8202;0;0;0;
   50 PROCinitialise
   60 PROCscore(0)
   70 PROClives(0)
   80 REPEAT
   90   PROCgame
  100   PROClives(-1)
  110   UNTIL lives=0
  120 *FX15
```

```
 130 END
+-----------------------------------------------------+
| Initialise all variables, assemble the machine-code |
| and plot the alien and the base.                    |
+-----------------------------------------------------+
 140
 150 DEFPROCinitialise
 160 DIM code 500
 170 PROCassemble(code,2)
 180 alien=0
 190 base=1
 200 bomb=2
 210 bullet=3
 220 alienX=40
 230 alienY=128
 240 baseX=40
 250 baseY=20
 260 bulletflag=FALSE
 270 bombflag=FALSE
 280 lives=3
 290 PROCplot(alien,alienX,alienY)
 300 PROCplot(base,baseX,baseY)
 310 ENDPROC
 320
+-----------------------------------------------------+
| Play the game, until the player is dead.            |
+-----------------------------------------------------+
 330 DEFPROCgame
 340 REPEAT
 350   PROCmovealien
 360   PROCmovebase
 370   PROCmovebomb
 380   PROCmovebullet
 390   IF FNdeadalien THEN PROCkillalien:PROCscore(10)
:PROCnewalien
 400   UNTIL FNdead
 410 ENDPROC
 420
+-----------------------------------------------------+
| Wipe the alien shape, update its coordinates, and   |
| plot the alien back up again.  Also give it a chance|
| to launch a bomb                                    |
+-----------------------------------------------------+
```

```
 430 DEFPROCmovealien
 440 PROCplot(alien,alienX,alienY)
 450 alienX=alienX+RND(5)-3
 460 alienY=alienY+RND(5)-3
 470 PROCplot(alien,alienX,alienY)
 480 IF RND(5) = 1 THEN PROCdropbomb(alienX+2,alienY-14)
 490 ENDPROC
+-----------------------------------------------------+
| Get keys from the keyboard to move the base and fire|
| bullets. The keys are 'Z' and 'X' to move left and  |
| right, and 'RETURN' to fire a bullet. Note that     |
| only one bullet may be in the air at once.          |
+-----------------------------------------------------+
 500
 510 DEFPROCmovebase
 520 PROCplot(base,baseX,baseY)
 530 baseX=baseX+INKEY-98-INKEY-67
 540 PROCplot(base,baseX,baseY)
 550 IF INKEY-74 THEN PROCfirebullet(baseX+1,baseY+8)
 560 ENDPROC
 570
+-----------------------------------------------------+
| Launch a bomb, by setting 'bombflag' to true, and   |
| initialising its coordinates. Also plot the bomb.   |
+-----------------------------------------------------+
 590 IF bombflag THEN ENDPROC
 600 bombflag=TRUE
 610 bombX=X
 620 bombY=Y
 630 PROCplot(bomb,bombX,bombY)
 640 ENDPROC
 650
+-----------------------------------------------------+
| Launch a bullet, in much the same way as the bomb   |
+-----------------------------------------------------+
 660 DEFPROCfirebullet(X,Y)
 670 IF bulletflag THEN ENDPROC
 680 bulletflag=TRUE
 690 bulletX=X
 700 bulletY=Y
 710 PROCplot(bullet,bulletX,bulletY)
 720 ENDPROC
 730
```

```
+-----------------------------------------------------+
| Update the bomb's position, and kill it off if it   |
| hits the ground.                                    |
+-----------------------------------------------------+
 740 DEFPROCmovebomb
 750 IF bombflag=FALSE THEN ENDPROC
 760 PROCplot(bomb,bombX,bombY)
 770 bombY=bombY-4
 780 IF bombY<=8 THEN bombflag=FALSE:ENDPROC
 790 PROCplot(bomb,bombX,bombY)
 800 ENDPROC
 810
+-----------------------------------------------------+
| Update the bullet's position, and stop the bullet if|
| it hits the ceiling                                 |
+-----------------------------------------------------+
 820 DEFPROCmovebullet
 830 IF bulletflag=FALSE THEN ENDPROC
 840 PROCplot(bullet,bulletX,bulletY)
 850 bulletY=bulletY+6
 860 IF bulletY >= 248 THEN bulletflag=FALSE : ENDPROC
 870 PROCplot(bullet,bulletX,bulletY)
 880 ENDPROC
 890
+-----------------------------------------------------+
| Check if player's bullet has hit the alien          |
+-----------------------------------------------------+
 900 DEFFNdeadalien
 910 IF bulletflag THEN IF ABS(bulletY-alienY)<5 AND
     ABS(bulletX-alienX)<5 THEN PROCplot(bullet,
     bulletX,bulletY):bulletflag=FALSE: =TRUE
 920 = FALSE
 930
+-----------------------------------------------------+
| If alien is dead, then knock the shape off the      |
| screen.                                             |
+-----------------------------------------------------+
 940 DEFPROCkillalien
 950 PROCplot(alien,alienX,alienY)
 960 ENDPROC
 970
+-----------------------------------------------------+
| Adjust the score, and display it                    |
+-----------------------------------------------------+
```

```
 980 DEFPROCscore(increment)
 990 score=score+increment
1000 PRINTCHR$(30);"Score :";score;
1010 ENDPROC
1020
+-----------------------------------------------------+
| Do much the same with the lives.                    |
+-----------------------------------------------------+
1030 DEFPROClives(increment)
1040 lives=lives+increment
1050 PRINTTAB(12,0);"Lives :";lives
1060 ENDPROC
1070
+-----------------------------------------------------+
| Initialise an alien.                                |
+-----------------------------------------------------+
1080 DEFPROCnewalien
1090 alienX=40
1100 alienY=128
1110 PROCplot(alien,alienX,alienY)
1120 ENDPROC
1130
+-----------------------------------------------------+
| Check if alien's bomb has hit player's base.        |
+-----------------------------------------------------+
1140 DEFFNdead
1150 IF bombflag THEN IF ABS(bombX-baseX)<4 AND
     ABS(bombY-baseY)<8 THEN bombflag=FALSE
     :P ROCplot(bomb,bombX,bombY): =TRUE
1160 =FALSE
1170
+-----------------------------------------------------+
| Nice interface for BASIC to the machine-code plotter|
+-----------------------------------------------------+
1180 DEFPROCplot(A%,X%,Y%)
1190 CALL plotshape
1200 ENDPROC
1210
+-----------------------------------------------------+
| Assemble the plotter routine. BASIC I users         |
| should convert all EQUB's and EQUW's to OPT FNequb's|
| and OPT FNequw's                                    |
+-----------------------------------------------------+
1220 DEFPROCassemble(origin,maxpass)
```

```
1230 P%=&70
1240 [OPT 0
1250 .addr
1260   EQUW 0
1270 .top
1280   EQUW &3000
1290 .rowcounter
1300   EQUB 0
1310 .counter
1320   EQUB 0
1330 .temp
1340   EQUB 0
1350 .temp1
1360   EQUB 0
1370 .depth
1380   EQUB 0
1390 .shape
1400   EQUW 0
1410 .offset
1420   EQUB 0
1430 ]
+-------------------------------------------------+
| This sets up the shapes. This means that, if you    |
| are using cassettes, the shapes must come AFTER this|
| program, or, alternatively, they may be on another  |
| cassette. Also, if on cassette, you should put a    |
| 'CLS' at, say, line 1585, so that the tape messages |
| are not left on the screen, when the game starts    |
+-------------------------------------------------+
1440 DIM shapeloaddr 3, shapehiaddr 3, shapesize 3,
shapedepth 3
1450 DIM shapes 300
1460 FOR I%=0TO3
1470 READfilename$
1480 OSCLI("LOAD "+filename$+" "+STR$~shapes)
1490 I%?shapeloaddr=shapes AND&FF
1500 I%?shapehiaddr=shapes DIV256
1510 READ I%?shapesize
1520 READ I%?shapedepth
1530 shapes=shapes+I%?shapedepth*I%?shapesize
1540 NEXT
1550 DATA alien, 5, 14
1560 DATA base, 4, 20
1570 DATA bomb, 2, 8
```

```
1580 DATA bullet, 2, 8
1590 FOR pass =0 TO maxpass STEP maxpass
1600 P%=origin
1610 [OPT pass
+-----------------------------------------------------+
| See section 13.2 for details on the following routines|
+-----------------------------------------------------+
1620.getaddr
1630   LDA #&00          Set the high byte of addr
1640   STA addr+1        to 0
1650   TYA               Invert the Y coordinate
1660   EOR #&FF          and save on the stack
1670   PHA
1680   LSR A             Divide Y coordinate by 8
1690   LSR A
1700   LSR A
1710   TAY               and leave in Y
1720   LSR A             Adjust carry for * &80
1730   STA temp          Save Y/16 in temp
1740   LDA #&00          Set bottom byte of addr to 0
1750   ROR A             Put carry into top bit
1760   ADC top           and add in top of screen
1770   PHP               Save carry flag
1780   STA addr          Store result in addr
1790   TYA               Get Y/8
1800   ASL A             Double it for top byte
1810   ADC temp          of addr. Add in Y/16
1820   PLP               Restore carry
1830   ADC top+1         and add in top of screen
1840   STA addr+1
1850   LDA #&00          Set temp to 0
1860   STA temp
1870   TXA               Get X coordinate
1880   ASL A             Perform 2-byte multiplication
1890   ROL temp          of temp by 8, because of
1900   ASL A             the memory map
1910   ROL temp
1920   ASL A
1930   ROL temp
1940   ADC addr          Add in rest of result to
1950   STA addr          far, and store it
1960   LDA temp
1970   ADC addr+1
1980   BPL ok            Check for hardware scroll
```

```
1990   SEC              If over 3000-8000 boundary
2000   SBC #&50         then correct address
2010.ok
2020   STA addr+1       And store it
2030   PLA              Restore inverted y coord
2040   AND #&07         Get row number in computed
2050   ORA addr         column, and add it in
2060   STA addr
2070   RTS              Return
2080
2090.plotshape
2100   PHA                   Save shape number
2110   JSR getaddr           Get address
2120   PLA                   Restore shape
2130   TAY
2140   LDA shapeloaddr, Y    Set up parameters
2150   STA shape             This assumes that
2160   LDA shapehiaddr, Y    the User has set
2170   STA shape + 1         up the relevent
2180   LDA shapesize, Y      tables
2190   STA counter           (shapeloaddr,
2200   LDA shapedepth, Y     shapesize, etc)
2210   STA depth
2220 \fall through to 'doplot'
2230.doplot
2240   LDA addr+1            If highbyte is 0
2250   BEQ return            then return
2260 .label
2270   LDY #&00              Set shape offset to 0
2280   LDA addr+1            Push screen address onto
2290   PHA                   stack, for later use
2300   LDA addr
2310   PHA
2320   LDA depth             Get depth of shape
2330   STA rowcounter
2340   LDA addr              Put offset in character
2350   AND #&07              cell into Y
2360   STA offset
2370   LDA addr              Adjust addr accordingly
2380   AND #&F8              Go to top of character
2390   STA addr                                 cell
2400.innerloop
2410   LDA (shape),Y         Get byte from shape
2420   INY
```

```
2430   STY temp
2440   LDY offset
2450   EOR (addr),Y
2460   STA (addr),Y
2470   INY                  Y holds offset on screen
2480   CPY #&08             Bottom of character cell?
2490   BEQ block            If so, then go down a line
2500.noblock
2510   STY offset
2520   LDY temp
2530   DEC rowcounter
2540   BNE innerloop
2550.nextblock
2560   LDA shape
2570   CLC
2580   ADC depth
2590   STA shape
2600   BCC nohi
2610   INC shape+1
2620.nohi
2630   CLC                   Go to top of next column,
2640   PLA                   by resetting address to top
2650   ADC #&08              of current column, and
2660   STA addr              moving to next character
2670   PLA                   cell
2680   ADC #&00
2690   BPL nobound1
2700   SEC
2710   SBC #&50
2720.nobound1
2730   STA addr+1
2740   DEC counter           Easier to DEC counter in
2750   BNE label             two places and test
2760.return
2770   RTS
2780.block
2790   LDY #&00              Go down a line
2800   LDA addr              addr=addr+&280
2810   CLC
2820   ADC #&80
2830   STA addr
2840   LDA addr+1
2850   ADC #&02
```

```
2860   BPL noboundary       If contents of addr
2870   SEC                  greater than &8000, then
2880   SBC #&50             subtract &5000
2890 .noboundary
2900   STA addr+1
2910   BNE noblock          Always jump
2920
2930]
2940NEXTpass
2950 ENDPROC
```

See inset colour illustrations for shape design.