13 Teletext Graphics, Mode 7

Mode 7 on the BBC micro is the only mode in which the shapes of the characters are not explicitly stored in memory. Instead just one byte is used for each of the screen character locations which, in the simplest case, holds the ASCII code of the character to be displayed there. The screen display is then generated by a special micro-chip which contains the data for each character. The chip looks at the screen memory to see which character is to be displayed at that position and then includes the correct data for that character directly into the television signal. This means that the chip has to recalculate the data for the whole screen each time the television display is refreshed, 50 times a second (60 times a second in the U.S.A.).

The memory for the screen is arranged very simply and consists of 1000 locations (40 across by 25 down). Starting at location HIMEM (which is set to &7C00 in mode 7) the data for the screen are stored row by row, with forty locations per row. Hence the location equivalent to the block accessed by PRINT TAB(X, Y) is given by &7C00 + 40 * Y + X. As in chapter 5, we assume that there is no hardware scrolling of the screen.

There are two types of character in this mode: *alphanumeric* and *graphic*. Try the simple program in listing 13.1 which changes one single screen location at a time by storing the ASCII code of an alphanumeric character.

Listing 13.1

```
10 MODE 7
20 REPEAT
30 A=&7C00+(RND(25)-1)*40+RND(40)-1
40 ?A=RND(32)+64
50 UNTIL FALSE
```

Example 13.1

Invisible control codes may also be placed in these screen locations. These allow control of further display options such as colour, flashing, or distinguishing between alphanumeric and graphic characters. Rerun listing 13.1 (which drew alphanumeric characters by default) with line 40 replaced by ?A = RND(256) – 1. The effect of these codes is to mix up graphics and alphanumerics randomly on the screen.

Table 13.1 shows the ASCII codes that have special effects on the mode 7 teletext screen.

Table 13.1

128		144	
129	Alpha red	145	Graphic red
130	Alpha green	146	Graphic green
131	Alpha yellow	147	Graphic yellow
132	Alpha blue	148	Graphic blue
133	Alpha magenta	149	Graphic magenta
134	Alpha blue	150	Graphic cyan
135	Alpha white	151	Graphic white
136	Start flash	152	Conceal
137	End flash	153	Contiguous graphics
138		154	Separated graphics
139		155	
140	Normal height	156	Black background
141	Double height	157	New background
142		158	Hold graphics
143		159	Release graphics

Some of these characters are available (from OS 1.0 onwards) directly from the keyboard by using the shift or control keys with the red soft keys. Most useful are the colour codes alpha red to alpha white, which are available with shift on keys fl to f7 (see table 3.2), and colour codes graphic red to graphic white, which are available with control on the same keys. Other codes can be programmed for the function keys by using the '!' option (see user guide) to produce codes that are numerically greater than or equal to 128. For example, typing 'KEY1||!||M would set key fl to code 141 which corresponds to double height. The control keys H, I, J, K will move the text cursor (left, right, down, up): this will enable you to experiment by placing control codes all over the screen. For example, double-height letters may be placed on the screen by typing f1 (the control code 141) followed by the required text at the text cursor, and then repeating (or copying) the same string (code 141 followed by text) exactly one line above. You can see this in programmed form in the 'brickout' game of listing 13.5 .

Each hne of the display is treated as an individual unit. Unless told otherwise, the computer assumes that a black background and white alphanumerics are required. For example, to get red text you must type the colour code for alpha red (shift fl), which would appear as a space on the line, followed by any required text. The red code will be effective either for the remainder of the present line or

until another colour code is encountered. Naturally all control codes take up one screen block and result in a space being displayed at that position. These codes can be used to highlight REMarks in prograin listings when viewed in mode 7. The programs on the audio cassettes that accompany this book contain colour codes in the REM statements to make it easier to find individual procedures and to read the explanatory comments.

Listing 13.2

```
10 MODE 7
19 REM go through all the codes except the control codes ( < 32 )
20 FOR I=32 TO 255 STEP 32
30 VDU 129
40 FOR J=I TO I+31
49 REM don't output characters 127 or it will delete previous code.
50 IF J<>127 THEN VDU J
60 NEXT J
70 PRINT ‘
80 NEXT I
```

Example 13.2

Run listing 13.2. The program will display, in red, all the available *alphanumeric characters* (equivalent to ASCII codes 32 to 255). If we change line 30 from VDU 129 (alpha red) to VDU 145 (graphic red) you will see all the available graphic characters. The upper case characters are the same in both the graphics and alplianumeric colours, but the remaining characters are different. These other codes will be displayed as the *block graphics characters*, each containing 6 small square blocks (3 vertically by 2 horizontally). The 64 possible characters that are produced from such a combination of 6 blocks may be accessed by typing the normal alphanumeric characters for the ASCII codes 32 to 63 and 96 to 127 after a graphics code. Note that the ASCII codes 65 to 95 still produce their normal symbols. All the codes from 32 to 127 are duplicated on ASCII codes 160 to 255. Therefore should we wish to write a program that PRINTs teletext characters (including teletext codes) it is easier to use the lower ASCII code values which can be typed (into the string inside quotes following PRINT) directly from the keyboard. This, however, has the disadvantage that code 127, which should correspond to a totally filled character, is equivalent to the delete code. The VDU command allows use of the characters with codes 160 onwards. To calculate the appropriate ASCII character for any given pattern of blocks you should use the method detailed in the user guide, which finds the code value for any block character in the higher set (160 to 255) or you can see them in figure 13.1.

Example 13.3

It is tedious to access each individual block character, especially if we wish to use them for drawing low-resolution diagrams. A far more sensible approach is to create a library of procedures to do the manipulation for us. For example, the

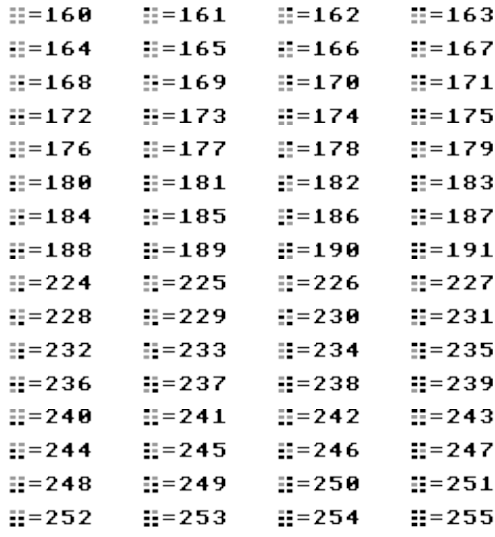

Figure 13.1

listing 13.3 draws a low-resolution picture of sine and cosine curves. The individual square pixels of this picture are the 1/6th blocks from within the graphics characters. Naturally we must allow for the addition (or deletion) of extra square pixels within a graphics character. We use the locations down the left-hand side of the screen for the graphic white code (or any other graphics colour). This means that the *x*-coordinates for our square pixel-blocks are in the range 2 to 79 (character columns 1 to 39: column 0 holds the colour code) and the *y*-coordinates are in the range 0 to 74 (rows 24 to 0). We introduce a 'plot' procedure which has three parameters, the coordinates of the pixel to be plotted and an integer (1, 2 or 3) that is used to define the type of plot – plot FOREGROUND, plot BACKGROUND or plot EOR respectively. We also give the 'draw' procedure which joins two specified points with an approximation to a straight line.

After mnning the program, hold down the RETURN key and you will delete all the control codes from the left-hand edge of the screen, and all the equivalent text characters become visible. Now type control Z, wllich will take the text cursor to the top of the screen. Then continuously press soft key f9 (set by line 180 ofthe program), which will reset the graphics control codes.

Listing 13.3

```
 10 MODE 7
 19 REM Array D stores values for positioning each 1/6th block
        inside character.
 20 DIM D(1,2)
 30 FOR I%=0 TO 1
 40 FOR J%=0 TO 2
 50 READ D(I%,J%)
 60 NEXT J%
 70 NEXT I%
 80 DATA 16,4,1,64,8,2
 89 REM place graphics white codes down side of screen.
 90 FOR L=0 TO 24
100 A=HIMEM+40*L : ?A=151
110 NEXT L
119 REM Draw axes.
120 PROCdraw(1,2,35,79,35)
130 PROCdraw(1,40,0,40,74)
139 REM Simultaneously plot curves.
140 FOR I=2 TO 79 STEP 1
150 PROCplot(1,I,COS(4-I/10)*30+35)
160 PROCplot(1,I,SIN(4-I/10)*30+35)
170 NEXT I
180 *KEY0|!|W|J|H
190 END

200 REM plot
210 DEF PROCplot(M,X,Y)
220 LOCAL C,XX,YY
229 REM Find address of character block which contains ‘pixel'.
230 A=&7C00+(25-(Y DIV 3))*40+X DIV 2
240 XX=X MOD2 : YY=Y MOD 3
249 REM Make sure we have character in higher set (160 to 255).
250 C=?A : IF C<128 THEN C=C+128
259 REM Modify character.
260 IF M=1 THEN C=C OR D(XX,YY)
270 IF M=2 THEN C=C AND (D(XX,YY) EOR &FF)
280 IF M=3 THEN C=C EOR D(XX,YY)
289 REM Replace character.
290 ?A=C
300 ENDPROC

400 REM draw
410 DEF PROCdraw(M,X1,Y1,X2,Y2)
420 DX=X2-X1 : DY=Y2-Y1 : SX=SGN(DX) : SY=SGN(DY)
430 DX=ABS(DX) : DY=ABS(DY)
440 IF DX=0 THEN ST=DY : SX=0 : GOTO 480
450 IF DY=0 THEN ST=DX : SY=0 : GOTO 480
460 IF DX>DY THEN SY=SY*DY/DX : ST=DX ELSE SX=SX*DX/DY : ST=DY
470 IF ST=0 THEN ENDPROC
479 REM Plot each pixel along line.
480 FOR I=0 TO ST STEP SGN(ST)
490 PROCplot(M,X1,Y1)
500 X1=X1+SX : Y1=Y1+SY
510 NEXT I
520 ENDPROC
```

Exercise 13.1
Change the value of NXNX and NYP1X in the 'start' procedure (listing 2.1) and alter 'moveto' and 'lineto' (listings 2.4 and 2.5) so that they produce calls to the 'plot' and 'draw' procedures. This will allow you to use any of our existing two-dimensional and three-dimensional graphics programs to draw 1ow-resolution teletext pictures. Figure 13.2 is a 1ow-resolution picture of a cube that was drawn by the program from chapter 9 which has been altered in this way.

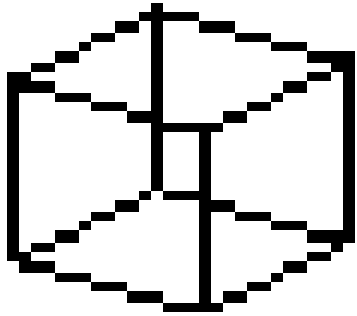

Figure 13.2

Listing 13,4

```
2000 DEF PROCkeys
2010 DEF PROCinitkeys
2020 *KEY0|!|L
2030 *KEY1|!|M
2040 *KEY2|!|Z
2050 *KEY3|!|\
2060 *KEY4|!|]
2070 *KEY5|!|^
2080 *KEY6|!|_
2090 ENDPROC
```

We now return to a discussion of the control co:ies. As we have seen, not all the codes are immediately available from the keyboard so we give a procedure 'initkeys' (listing 13.4) which simplifies the entry of control codes that are not a1ready available on the keyboard by redefining the soft keys. Table 13.2 lists the codes accessible from the keyboard.

Table 13.2 Control codes available on the function keys

Key			With shift		With control	
fl	141	Double height	129	Alpha red	145	Graphic red
f2	154	Separated graphics	130	Alpha green	146	Graphic green
f3	156	Black background	131	Alpha yellow	147	Graphic yellow
f4	157	New background	132	Alpha blue	148	Graphic blue
f5	158	Hold graphics	133	Alpha magenta	149	Graphic magenta
f6	159	Release graphics	134	Alpha cyan	150	Graphic cyan
f7			135	Alpha white	151	Graphic white
f8			136	Start flash	152	Conceal
f9			137	End flash	153	Contiguous graphics

The codes flashing (136) and non-f1ashing (137) are already available with shift on function keys 8 and 9. Like all control codes, these affect the remainder of the line (unless the opposite code is encountered) and they each appear as a space. Flashing displays the characters as alternately a blank background and then the normal character in quick succession. We have already seen double-height characters (code 141). This again will affect everything to the end of the line unless the normal height is restored with code 140.

To get different background colours in teletext we must first select a new colour (either graphics or text, it does not matter which) and follow it with the code for a new background (157). Since we are unable to select black as a foreground colour there must also be a special command to set a black background (code 156), and naturally you will have to introduce a new foreground colour. This means that the codes to set a new background will appear as two spaces on the screen. Code 152 enables you to conceal a string with background colour. If you overprint this code with a different code then the string becomes visible.

The last fuur codes to be considered all relate to the relative position of the graphics characters. Contiguous graphics (code 153) is assumed for all lines and means that 1/6th character blocks all touch. Separated graphics (code 154) slightly separates these square blocks. When pictures are drawn in teletext it is obviously a disadvantage to have a blank space between different colours. The hold graphics code (158) is designed so that subsequent control codes will not be displayed as blanks but will have the previous graphics character repeated and displayed at that location. The release graphics code (159) turns off this effect.

Example 13.4

We use some of these commands in the program given in listing 13.5 which plays a ‘brickout’ game in mode 7. The program first prints out a double-height logo,

and requests a skill factor (0 diicult to 10 easy). Five coloured walls are drawn, and on pressing the space bar a bat appears and a ball is served into play. The bat is moved left and right by the corresponding cursor keys, and is used to hit the ball back into play. Whenever the ball hits a wall it knocks a brick out and increments the score. If the ball hits the bottom of the screen then the serve is over. You have three serves per gaine. If the ball penetrates the five walls and hits the top line then the bat gets smaller.

Listing 13.5

```
  10 MODE 7 : HISCORE=0
  20 REPEAT
  30 PROCinit
  40 PROCgame
  50 PROCend
  60 UNTIL FALSE
  70 STOP

 100 REM Initialise game variables
 110 DEF PROCinit
 120 SCORE=0 : BALLS=3
 130 B$=" ppppp " : LB=5
 140 BRICK$="////////////////////////////////"
 150 B=2 : *FX15,0
 160 REPEAT : CLS
 170 PROClogo(3,2)
 179 REM print twice for double height letters
 180 PRINT TAB(9,12);"SKILL LEVEL (0 TO 10)"
 190 PRINT TAB(9,13);"SKILL LEVEL (0 TO 10)"
 200 PRINT SPC(40): PRINT TAB(17,14);:INPUT S
 210 UNTIL S>=0 AND S<=10
 220 SKILL=S*10
 230 PROCwall
 240 ENDPROC

 300 REM game
 309 REM repeat all stages of game. Note you have 3 chances to
       move bat each time ball moves
 310 DEF PROCgame
 320 REPEAT
 330 PROCthrow
 340 REPEAT
 350 PROCkey
 360 PROCball
 370 PROCkey
 379 REM slow down game
 380 FOR I=1 TO SKILL : NEXT I
 390 PROCkey
 400 UNTIL OUT
 410 BALLS=BALLS-1
 420 UNTIL BALLS=0
 430 ENDPROC

 500 REM start new ball
 510 DEF PROCthrow
 520 PRINT TAB(1,17); : VDU 141
 530 FOR I%=0 TO 1
 540 PRINT TAB(8,16+I%); : VDU 141,136
```

```
 550 PRINT"PRESS"; : VDU 130
 560 PRINT""SPACE"""; : VDU 151
 570 PRINT"FOR BALL"; : VDU 137
 580 NEXTI%
 590 REPEAT:UNTIL INKEY(-99)
 600 PRINT TAB(1,17); : VDU 32
 610 FOR I%=0 TO 1
 620 PRINT TAB(8,16+I%);"                          "
 630 NEXT I%
 640 X=RND(20)+10 : Y=22
 650 XD=SGN(RND(1)-.5) : YD=-1
 660 PRINT TAB(X,Y);"0"; : OX=X : OY=Y
 670 OUT=FALSE
 680 ENDPROC

 700 REM check keyboard
 710 DEF PROCkey
 720 IF INKEY(-122) AND B<38-LB THEN B=B+1
 730 IF INKEY(-26) AND B>2 THEN B=B-1
 740 PRINT TAB(B,23);B$;
 750 ENDPROC

 800 REM move ball
 810 DEF PROCball
 820 Y=Y+YD : IF Y>23 THEN OUT=TRUE : GOTO 930
 830 IF Y=22 AND YD=-1 AND BRICKOUT THEN PROCwall
 840 X=X+XD : IF X<4 THEN X=3 : XD=-XD
 850 IF X>37 THEN X=38 : XD=-XD
 860 IF Y=2 THEN YD=-YD : IF LB=5 THEN B$=" ppp " : LB=3
     : PRINT TAB(B,23);B$;" ";
 870 PRINT TAB(X,Y); : A%=135 : C=USR(&FFF4)
     : C=(C AND &FF00)/&100
 880 IF C=47 THEN PROCbrickout : YD=-YD :SOUND0,-15,2,1
 890 IF C<>112 THEN 910 ELSE YD=-YD : SOUND 1,-15,150,1
     : IF X<>(B+1+INT(LB/2))THEN XD=SGN(XD)*2 ELSE XD=SGN(XD)
 900 IF C=112 THEN 920
 910 PRINT TAB(X,Y);"0";
 920 IF OY=23 THEN 940
 930 PRINT TAB(OX,OY);" ";
 940 OX=X : OY=Y
 950 ENDPROC

1000 REM draw new wall
1010 DEF PROCwall
1020 CLS : VDU23,1,0;0;0;0;
1030 VDU 134 : PRINT"   SCORE ";SCORE
1040 PRINT TAB(22,0);"HISCORE ";HISCORE
1050 FOR I%=1 TO 22
1060 PRINT TAB(0,I%); : VDU 151
1070 PRINT" j"
1080 PRINT TAB(39,I%); : PRINT"5"
1090 NEXT I%
1100 FOR I%=5 TO 9
1110 PRINT TAB(0,I%); : VDU 140+I%
1120 PRINT TAB(3,I%); : PRINT BRICK$
1130 NEXT I%
1140 PRINTTAB(3,1);"````````````````````````````````````"
1150 PRINTTAB(0,23);:VDU150
1160 BRICKOUT=FALSE
1170 ENDPROC
```

```
1200 REM add score for a brick
1210 DEF PROCbrickout
1220 SCORE=SCORE+10-Y
1230 IF SCORE>HISCORE THEN HISCORE=SCORE
1240 PRINT TAB(10,0);SCORE : PRINT TAB(30,0);HISCORE
1250 IF SCORE MOD 540=0 THEN BRICKOUT=TRUE
1260 ENDPROC

1300 REM end of game
1310 DEF PROCend
1319 REM delete remaining bricks
1320 PRINT TAB(39,9); : I=0
1330 REPEAT
1340 PRINT" "; : VDU 8,8 : I=I+1
1350 A=RND(200)
1360 SOUND1,-10,A,1 : SOUND2,-10,1.5*A,1 : SOUND3,-10,A*1.75,1
1370 UNTIL INKEY(-99) OR I=200
1380 ENDPROC

1400 REM print double height brickout logo on screen
1410 DEF PROClogo(B,F)
1420 PRINT TAB(0,0);
1430 FOR I%=1 TO 25
1440 B=(B+2) MOD 6 : F=(F+2) MOD 6
1450 IF I%<>1 THEN PRINT
1460 VDU 129+B,157,129+F,141
1470 PRINT"********** BRICKOUT **********";
1480 NEXT I%
1490 ENDPROC
```

The game was written in a modular fashion similar to that of the worm game chapter 1. We shall not give a detailed description of the program since the technique of constmction should be self-evident from the listing. However, it is instmctive to note from the game logo that the background and foreground coiours for the two halves of a double-height character need not be the same.

We now consider the construction of the sort of pictures that are fainiliar to all owners of teletext televisions. It is possible to write a program to generate each individual picture, or painstakingly to create pictures by typing in codes and text from the keyboard. It is far better to use an interactive program for drawing displays like figure 13.3. We give such a program in listing 13.6.

Because the screen is used by the program for displaying messages as well as for drawing the picture, it is essential to store a copy of the picture elsewhere in tlle memory. We chose the locations between &7800 and &7C00, which we call a picture buffer, and set HIMEM to &7800 to protect this area. Any intentional changes made to the diagram on the screen (that is, to the screen memory) are also made to the corresponding buffer locations. Hence even though the screen is overwritten at several stages by the program, the diagram can be restored from ‘he memory buffer when required. The program includes an error-handling routine which ensures that any problems, apart from a deliberate BREAK, will

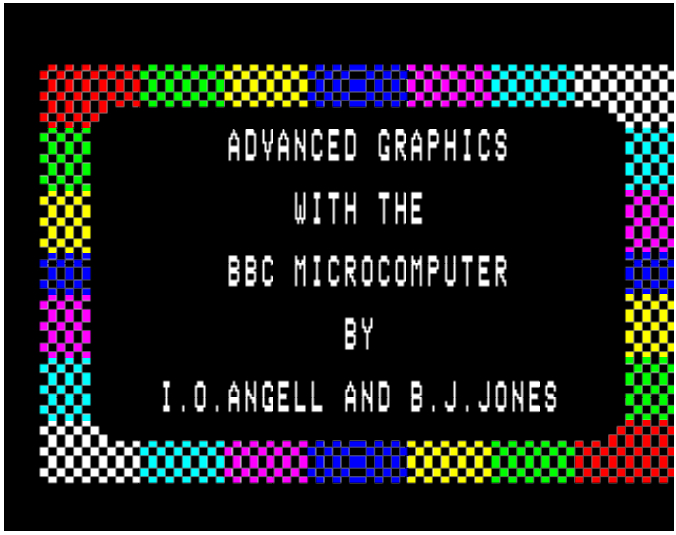

Figure 13.3

Listing 13.6

```
 10 ON ERROR GOTO 1510
 20 MODE 7 : HIMEM=&7800
 30 *LOAD EDPIC 7800
 39 REM array giving bit values for blocks in graphics characters
 40 DIM D(1,2)
 50 FOR I%=0 TO 1
 60 FOR J%=0 TO 2
 70 READ D(I%,J%)
 80 NEXT J%
 90 NEXT I%
100 DATA 16,4,1,64,8,2
110 VDU 15 : VDU 23,1,0;0;0;0;
120 *FX4,1
130 PROCinitkeys
139 REM initialise positions of text and graphics cursors
140 TX=20 : TY=10
150 GX=40 : GY=35
160 X1=GX : Y1=GY
169 REM display picture with menu
170 PROCbackon
180 PROCmenu
189 REM printing is done inside one line window at bottom of
        screen to appear flashing
190 REPEAT
200 PRINT'" TELETEXT EDITOR ";
209 REM FNkeys waits for a key press in the same way as INKEY$
        but allows the cursor to move
```

```
210 A$=FNkeys(100)
220 IF A$<>"" THEN PROCcommand
230 PRINT'" INPUT COMMAND ? ";
240 A$=FNkeys(100)
250 IF A$<>"" THEN PROCcommand
260 UNTIL FALSE
270 END
300 REM command
310 DEF PROCcommand
319 REM deal with any keypress
320 PRINT'" INPUT COMMAND ? ";A$;
330 IF A$=CHR$(13) THEN ENDPROC
340 P=INSTR("BGIMT",A$)
350 IF P=0 THEN PRINT" INVALID"; : A$=GET$ : GOTO 320
360 PRINT
370 IF A$="I" THEN PROCinit
380 IF A$="M" THEN PROCmenu
390 IF A$="T" THEN PROCtext
400 IF A$="B" THEN PROCbyte
410 IF A$="G" THEN PROCgraphic
420 ENDPROC

500 REM keys
510 DEF FNkeys(T%)
520 T=TIME
530 REPEAT
540 B$=INKEY$(0)
550 SHIFT=INKEY(-1)
559 REM if shift is pressed remove graphics cursor
560 IF SHIFT THEN PROCplot(1,3,GX,GY)
569 REM alter text/graphics cursor depending on shift
570 IF B$=CHR$(&8B) THEN PROCup : B$=""
580 IF B$=CHR$(&8A) THEN PROCdown : B$=""
590 IF B$=CHR$(&88) THEN PROCleft : B$=""
600 IF B$=CHR$(&89) THEN PROCright : B$=""
609 REM redraw graphics cursor
610 IF SHIFT THEN PROCplot(1,3,GX,GY)
619 REM if non-cursor key pressed or time is up then return
620 UNTIL TIME>T+T% OR B$<>""
630 =B$
700 REM up
710 DEF PROCup
719 REM move graphics cursor position
720 IF SHIFT THEN GY=GY+1 : IF GY=75 THEN GY=0
730 IF SHIFT THEN ENDPROC
739 REM erase text cursor move it and           redraw it
740 PROCcursoff : TY=TY-1 : IF TY=-1 THEN TY=24
750 PROCcurson : ENDPROC

800 REM down
809 REM see up
810 DEF PROCdown
820 IF SHIFT THEN GY=GY-1 : IF GY=-1 THEN GY=74
830 IF SHIFT THEN ENDPROC
840 PROCcursoff : TY=TY+1 : IF TY=25 THEN TY=0
850 PROCcurson : ENDPROC

900 REM right
909 REM see up
910 DEF PROCright
920 IF SHIFT THEN GX=GX+1 : IF GX=80 THEN GX=2
```

```
 930 IF SHIFT THEN ENDPROC
 940 PROCcursoff : TX=TX+1 :IF TX=40 THEN TX=0
 950 PROCcurson : ENDPROC

1000 REM left
1009 REM see up
1010 DEF PROCleft
1020 IF SHIFT THEN GX=GX-1 : IF GX=1 THEN GX=79
1030 IF SHIFT THEN ENDPROC
1040 PROCcursoff : TX=TX-1 : IF TX=-1 THEN TX=39
1050 PROCcurson
1060 ENDPROC

1100 REM get
1109 REM return the value stored on the screen at location X,Y
1110 DEF FNget(X,Y)
1120 A%=&7C00+Y*40+X : =?A%
1200 REM put
1210 DEF PROCput(A,X,Y,CHAR)
1219 REM store the CHARacter at X,Y either on the screen (A=1)
        or in the buffer (A=0)
1220 A%=&7800+A*&400+Y*40+X
1230 ?A%=CHAR : ENDPROC

1300 REM curson
1310 DEF PROCcurson
1319 REM remove graphics cursor, store current value at text cursor
     position and replace with a square, replace graphics
1320 PROCplot(1,3,GX,GY) : TST=FNget(TX,TY) : PROCput(1,TX,TY,255)
     : PROCplot(1,3,GX,GY)
1330 ENDPROC

1400 REM cursoff
1410 DEF PROCcursoff
1419 REM remove graphics cursor and replace text cursor by old
        screen value then replace graphics cursor
1420 PROCplot(1,3,GX,GY) : PROCput(1,TX,TY,TST) : PROCplot(1,3,GX,GY)
1430 ENDPROC

1500 REM error section
1509 REM if error occurs in loading EDPIC then carry on
1510 IF ERL=30 THEN 40
1519 REM if any other error occurs save the picture from buffer
        and then...
1520 *SAVE EDPIC 7800 7C00
1529 REM report error in usual way
1530 MODE 7 : REPORT : IF ERL<>0 THEN PRINT" at line ";ERL : VDU 14
1539 REM reset cursor keys and stop
1540 *FX4,0
1550 END

1600 REM backup
1610 DEF PROCbackup
1619 REM copy screen to buffer
1620 LOCAL I% : FOR I%=0 TO 996 STEP 4
1630 I%!&7800=I%!&7C00 : NEXT I%
1640 ENDPROC

1700 REM backon
1710 DEF PROCbackon
1719 REM copy buffer to screen
1720 LOCAL I% : FOR I%=0 TO 996 STEP 4
```

```
1730 I%!&7C00=I%!&7800 : NEXT I%
1739 REM replace cursors
1740 PROCplot(1,3,GX,GY) : PROCcurson
1750 ENDPROC

1800 REM menu
1810 DEF PROCmenu
1819 REM retrieve picture from buffer set window for menu
1820 PROCbackon
1830 VDU 28,2,21,39,4 : CLS
1839 REM put control codes for yellow background outside window
         so they can't scroll away
1840 FOR I%=4 TO 21 : PROCput(1,0,I%,131) : PROCput(1,1,I%,157)
     : NEXT I%
1849 REM print out instructions with pauses making operating system
         scroll window up
1850 PRINT TAB(0,16);" COMMAND M FOR MENU DISPLAY/REMOVAL"
1860 A$=INKEY$(10) : PRINT'" USE CURSOR KEYS TO MOVE TEXT CURSOR "
1870 A$=INKEY$(10) : PRINT'" USE SHIFT/CURSOR KEYS FOR GRAPHICS "
1880 A$=INKEY$(10) : PRINT'" COMMAND T TO PLACE TEXT AT CURSOR"
1890 A$=INKEY$(10) : PRINT'" COMMAND B TO ALTER BYTE AT CURSOR"
1900 A$=INKEY$(10) : PRINT'" COMMAND G FOR GRAPHICS FUNCTION"
1910 A$=INKEY$(10) : PRINT'" COMMAND I TO INIT TEXT/GRAPHICS"
1919 REM back to the top of window print double height header
1920 VDU30 : PRINT'SPC(12);CHR$(141);"COMMANDS"
1930 PRINT SPC(12);CHR$(141);"COMMANDS"
1939 REM alternate colours of two halves of header by colour codes
     while waiting
1940 REPEAT
1950 PROCput(1,10,5,133) : PROCput(1,10,6,130)
1960 C$=INKEY$(50) : IF C$="M" THEN 1990
1970 PROCput(1,10,6,133) : PROCput(1,10,5,130)
1980 C$=INKEY$(50)
1990 UNTIL C$="M"
1999 REM replace picture and remove window
2000 PROCbackon : VDU 28,0,24,39,24
2010 VDU 15,30
2020 ENDPROC

2100 REM plot
2110 DEF PROCplot(A,M,X,Y)
2120 LOCAL C,XX,YY
2129 REM calculate character position containing the point either
         on the screen (A=1) or in the buffer (A=0)
2130 A=A*&400+&7800+(25-(Y DIV 3))*40+X DIV 2
2139 REM find coordinates of point within the character
2140 XX=X MOD 2 : YY=Y MOD 3
2150 C=?A : IF C<128 THEN C=C+128
2160 IF M=1 THEN C=C OR D(XX,YY)
2170 IF M=2 THEN C=C AND(D(XX,YY) EOR &FF)
2180 IF M=3 THEN C=C EOR D(XX,YY)
2190 ?A=C
2200 ENDPROC

2300 REM draw
2310 DEF PROCdraw(A,M,X1,Y1,X2,Y2)
2319 REM draw a line from X1,Y1 to X2,Y2 either on the screen (A=1)
         or in the buffer (A=0)
2320 DX=X2-X1 : DY=Y2-Y1 : SX=SGN(DX) : SY=SGN(DY)
2330 DX=ABS(DX) : DY=ABS(DY)
2339 REM find the amounts by which one coordinate must change assuming
          the other has a larger distance and is moving in steps of one
```

```
2340 IF DX=0 THEN ST=DY : SX=0 : GOTO 2380
2350 IF DY=0 THEN ST=DX : SY=0 : GOTO 2380
2360 IF DX>DY THEN SY=SY*DY/DX : ST=DX ELSE SX=SX*DX/DY : ST=DY
2370 IF ST=0 THEN ENDPROC
2379 REM go along line adding step size to coordinates and
     plotting resulting point
2380 FOR I=0 TO ST STEP SGN(ST)
2390 PROCplot(A,M,X1,Y1)
2400 X1=X1+SX : Y1=Y1+SY
2410 NEXT I
2420 ENDPROC

2500 REM init
2510 DEF PROCinit
2520 REPEAT : INPUT" INITIALISE G OR T ? "B$ : UNTIL B$="G" OR B$="T"
2530 REPEAT : INPUT" WHICH COLOUR ( 1 TO 7 ) ? "C : UNTIL C>0 AND C<8
2540 REPEAT : INPUT" RANGE OF LINES ? "T,B
     : UNTIL T<B AND T>=0 AND B<=24
2550 IF B$="G" THEN C=C+144 ELSE C=C+128
2559 REM place a colour code (either text or graphics) at the start
         of each line in range both in buffer and on screen
2560 FOR I=T TO B : PROCput(1,0,I,C) : PROCput(0,0,I,C) : NEXT I
2570 ENDPROC

2600 REM text
2610 DEF PROCtext
2619 REM show whole picture and reset cursor so copy can be used
2620 PROCbackon
2630 *FX4,0
2640 VDU 23,1,1,0;0;0;0; : VDU 28,0,24,39,0
2650 PRINT TAB(TX,TY); : INPUT""ANY$ : IF LEN(ANY$)=0 THEN 2670
2659 REM put the string into buffer
2660 FOR I=1 TO LEN(ANY$) : PROCput(0,TX+I-1,TY,ASC(MID$(ANY$,I,1)))
: NEXT I
2669 REM reset mode 7 in case string was entered on bottom line
         scrolling screen
2670 VDU 22,7
2680 VDU 23,1,0,0;0;0;0; : VDU 28,0,24,39,24
2689 REM redisplay picture and reset cursor keys for program use
2690 PROCbackon
2700 *FX4,1
2710 ENDPROC

2800 REM byte
2810 DEF PROCbyte
2820 PROCbackon
2829 REM set one line window below byte and scroll to clear
         before printing value
2830 TT=(TY+1) MOD 25 : TS=(TY-1) MOD 25 : VDU 28,0,TT,39,TT
2840 PRINT'"  CURRENT VALUE IS ";?(&7800+40*TY+TX);
2849 REM set one line window above byte and input new value
         'return'=0 for no change
2850 VDU 28,0,TS,39,TS
2860 PRINT
2870 REPEAT : INPUT" WHAT ASCII CODE ( 32 TO 255 ) ? "C : UNTIL C>31
AND C<256 OR C=0
2880 IF C=0 THEN 2910
2890 PROCput(0,TX,TY,C) : PROCput(1,TX,TY,C)
2900 IF TY<> 24 THEN TST=C
2910 PROCbackon : VDU 28,0,24,39,24
2920 ENDPROC
```

```
3000 REM graphic
3010 DEF PROCgraphic
3019 REM plot point at cursor or draw line from last point
      specified to cursor
3020 REPEAT : INPUT" PLOT OR DRAW ( P OR D ) ? "A$
     : UNTIL A$="P" OR A$="D"
3030 REPEAT : INPUT" FOREGROUND,BACKROUND,EOR ( 1,2,3 ) ? "A
     : UNTIL A>0 AND A<4
3040 PROCbackon
3049 REM plot points directly into buffer then restore picture
3050 IF A$="P" THEN PROCplot(0,A,GX,GY)
3060 IF A$="D" THEN PROCdraw(0,A,X1,Y1,GX,GY)
3070 X1=GX : Y1=GY
3080 PROCbackon
3090 ENDPROC

3100 REM initkeys
3110 DEF PROCinitkeys
3119 REM set keys to produce more teletext codes directly
3120 *KEY0|!|L
3130 *KEY1|!|M
3140 *KEY2|!|Z
3150 *KEY3|!|\
3160 *KEY4|!|]
3170 *KEY5|!|^
3180 *KEY6|!|_
3190 ENDPROC
```

save the diagram as file EDPIC on backing store before ending. Before you start any constmction you may wish to clear the picture buffer locations by typing

CLS : PROCbackup

Initially the program will try to load the edit picture EDPIC from backing store; however if this is not available the program will continue on to the next line Whcn you press ESCAPE (if you are using tape backing store) or it will continue automatically (if you are using disks) – hence the need to clear the screen at the beginning of a diagram construction. Having loaded this picture (from the backing store or the buffer) the program will display it on the screen and then immediately display a menu over the top of it. This menu can be recalled or removed at any time by pressing key M. The other options available with this interactive program are Initialise, Graphics, Text and Byte. These commands are called by typing their initial letter and are described below.

Initialise is used to place a Graphics or Text colour code (of any colour) in the first column of each one of a range of lines (a subset of rows 0 to 24). This will affect the whole line unless countermanded by graphics/text colour codes further along that line. Text enables the typing of text and control codes on the screen (remember that if you want to start a text string with a space then you must enclose the whole entry in quotes). When text is being typed the copy cursor is enabled, so making it simple to copy parts of the screen to new locations. RETURN enables you to exit from this section ofthe program so that you can input another command.

Graphics allows you to use point plotting and line-drawing routines which include a choice of 'plot' options (background, foreground or EOR). The 'draw' cominand will draw from the last point specified either by plot or from the end of the last line drawn. The program creates a graphics cursor, by EORing a single point on the screen, which caii be moved by using the shift and cursor keys. When this graphics cursor runs through alplianumeric lines (as opposed to graphics) it will appear as a text character. Sometimes the graphics cursor is invisible, but holding down the shift key makes it flash, thus making it visible. The 'plot' and 'draw' procedures have an extra parameter which allows you to plot either on the screen as with the graphics cursor (which is not stored with the diagram), or directly into the buffer area as with line drawing. The text/graphics cursors must be moved into position prior to entedng the Text and Graphics sections.

Byte allows you to read directly the value currently positioned at the text cursor location. This enables you to see exactly which control code or character is at that position, and allows you either to alter it by typing in the replacement value or to RETURN. This is especially helpful if a graphics hold mode is operative, which effectively hides the control codes.

The program will automatically save the picture as file EDPIC on encounterin1 an error, such as typing ESCAPE by accident, so that any alterations are not lost. If you do not wish to save the picture, type BREAK to halt the program. The picture will still be in the buffer in memory.

Example 13.5: Animation

We can use the idea of a buffer to hold more than one picture in memory at any one time. This would be convenient for displaying simple teletext diagrams as a slide show in lectures or as advertisements. Since each picture occupies only 1K of store, at least twenty teletext pictures can be fitted into the memory at once. This is impossible for modes 0 to 2 since each screenful takes up 20K of memory in these modes. We simply set HIMEM to protect whatever area we wish to use for picture storage and then transfer each picture to the actual screen memory when required. Listing 13.7 contains an assembler routine to do this. Because user-defined characters are unavailable in mode 7 we can use this part of the memory (locations starting at &C00) to store the machine code produced by this program. It also stores the code as a file DISPLAY on backing store. This routine uses the OSBYTE call with the accumulator A set to 19, which allows the machine to wait for the start of the next refresh of the screen before copying the data to the screen: this helps to eliminate flickering. The program in listing 13.8 allows us to display any one of 20 pictures by typing a number between 0 and 19. Even when called from BASIC programs, this routine is fast enough to display a new picture within the time the machine takes to refresh the screen, and so it enables us to display animated pictures. We simply draw a set of pictures at varying stages of rotation, and arrange that the last frame is the same as the first. The movie section of the program (accessed by typing M) displays all the

pictures in rapid succession to get the animation effect. Listing 13.8 draws twenty pictures of a simple curve rotated through a further π/40 radians with each frame, and stores each picture at the correct location in memory by using a modified version of the 'backup' procedure of the teletext editor (listing 13.6). Then the program allows the slide show and movie procedures to display.

Listing 13.7

```
  9 REM display routine transfers a teletext picture to screen
 10 OSBYTE=&FFF4
 20 FOR I=0 TO 3 STEP 3
 30 P%=&C00
 39 REM multiply picture number in A by four and add to start of
       pictures to get hi-byte, use *FX19 to wait, then transfer
 40 [OPT I
 50 ASL A : ASL A
 60 CLC : ADC #&2C
 70 STA &71
 80 LDA #&7C : STA &73
 90 LDY #0 : STY &70 : STY &72
100 LDX #4
110 LDA #19 : JSR OSBYTE
120 .LOOP
130 LDA (&70),Y : STA (&72),Y
140 INY : BNE LOOP
150 INC &71 : INC &73
160 DEX : BNE LOOP
170 RTS
180 ]
190 NEXT
199 REM run address set to a RTS in ROM so that program can be
       loaded from disk by *<fsp>
200 *SAVE DISPLAY C00 +30 FFB8
```

Listing 13.8

```
 10 MODE 7
 20 HIMEM=&2C00
 30 DIM D(1,2)
 40 FOR I%=0 TO 1
 50 FOR J%=0 TO 2
 60 READ D(I%,J%)
 70 NEXT J%
 80 NEXT I%
 90 DATA 16,4,1,64,8,2
100 *LOAD DISPLAY
109 REM draw 20 frames of movie with curve rotating through PI/2
110 FOR F=0 TO 19
120 ANG=F*PI/40
129 REM for each frame put graphics white codes down the side
130 CLS
140 FOR I=0 TO 24 : PRINT TAB(0,I); : VDU 151 : NEXT I
149 REM draw clover leaf curve
150 OX=40+30*COS(ANG) : OY=38+30*SIN(ANG)
160 FOR I=0 TO 2*PI STEP PI/100
170 S=SIN(I+ANG) : C=COS(I+ANG)
180 S2=SIN(2*I) : C2=COS(2*I)
```

```
190 R=(C2^3+S2^3)*30
200 X2=40+R*C : Y2=38+R*S
210 PROCdraw(1,OX,OY,X2,Y2)
220 OX=X2 : OY=Y2
230 NEXT I
239 REM store frame F in memory
240 PROCbackup(F)
249 REM until all twenty frames are prepared and stored
250 NEXT F
260 PROCshow
270 END

300 REM plot
310 DEF PROCplot(M,X,Y)
320 LOCAL C,XX,YY
330 A=&7C00+(25-(Y DIV 3))*40+X DIV 2
340 XX=X MOD2 : YY=Y MOD 3
350 C=?A : IF C<128 THEN C=C+128
360 IF M=1 THEN C=C OR D(XX,YY)
370 IF M=2 THEN C=C AND (D(XX,YY) EOR &FF)
380 IF M=3 THEN C=C EOR D(XX,YY)
390 ?A=C
400 ENDPROC

500 REM draw
510 DEF PROCdraw(M,X1,Y1,X2,Y2)
520 LOCAL DX,DY,SX,SY,ST,I
530 DX=X2-X1 : DY=Y2-Y1 : SX=SGN(DX) : SY=SGN(DY)
540 DX=ABS(DX) : DY=ABS(DY)
550 IF DX=0 THEN ST=DY : SX=0 : GOTO 580
560 IF DY=0 THEN ST=DX : SY=0 : GOTO 580
570 IF DX>DY THEN SY=SY*DY/DX : ST=DX ELSE SX=SX*DX/DY : ST=DY
580 IF ST=0 THEN ENDPROC
590 FOR I=0 TO ST STEP SGN(ST)
600 PROCplot(M,X1,Y1)
610 X1=X1+SX : Y1=Y1+SY
620 NEXT I
630 ENDPROC

700 REM show
710 DEF PROCshow
719 REM ensure screen hasn't scrolled and that mode 7 is set
720 VDU 22,7
729 REM if you have a file of twenty frames already stored
730 INPUT" NAME OF SLIDE FILE OR PRESS <RETURN>   ",A$
740 IF A$="" THEN CLS : GOTO 770
749 REM use CLI to load file
750 $&7D00="LOAD "+A$+" 2C00"
760 X%=0 : Y%=&7D : CALL &FFF7
769 REM alternate between single frame slide show and coninuous movie
770 REPEAT
780 REPEAT
790 VDU 26 : INPUT A$
800 A%=VAL(A$)
809 REM put selected frame onto screen
810 CALL &C00
820 UNTIL A$="M"
829 REM if M is pressed start showing frames in quick succesion
830 REPEAT
840 FOR A%=0 TO 19
850 CALL&C00
860 NEXT A%
```

```
870 UNTIL INKEY(0)>-1
879 REM never leave until escape
880 UNTIL FALSE
890 ENDPROC
900 REM backup
910 DEF PROCbackup(FRAME)
920 LOCAL I%,A%
929 REM start address from frame
930 A%=&2C00+&400*FRAME
939 REM transfer data from screen
940 FOR I%=0 TO 996 STEP 4
950 I%!A%=I%!&7C00
960 NEXT I%
970 ENDPROC
```

Complete Programs

I Listing 13.1. No data required.

II Listing 13.2. No data required.

II Listing 13.3. No data required.

IV Listing 13.5. The ‘brickout’ game. Type in the skill level (try 10), then type a space to start the game and move the bat with the left and right cursors.

V Listing 13.6. The teletext interactive diagram constmction package. Try the following sample inputs (those underlined).

Type M, T, move the text cursor into position by using the cursor keys and type hello folks (note lower case characters). Then move the text cursor owr the ‘e’ and type B. This will show the ASCII code for the character that is at present occupying that screen position (namely ‘e’ = ASCII 101). Type 117 and RETURN. Now move the cursor away and you will see a ‘u’ replace the ‘e' . Then type I and T and you will be asked for a colour (try 1) and a range oflines (try the whole screen 0, 24). Everything should go red. Now type I, G, a colour (2), and range (again 0, 24), and the lower case red text characters change into green graphics characters.

VI Listings 13.7 and 13.8. First run 13.7 which creates the file DISPLAY. Then run listing 13.8 which reloads DISPLAY and either loads 20 pictures from backing store or on typing RETURN takes about 15 ntinutes to draw them. Then type an integer between 0 and 19 to get an individual frame, or M to get a movie: any key will stop it.