vertical direction, implying that the height of each character is 32 units. The boundaries of the screen can now be worked out in terms of high resolution co-ordinates for the MOVE and DRAW commands. The Screen Layout illustration shows what these boundaries are.

We can now calculate the co-ordinates of the bottom left corner and the top right corner of the minefield (all other co-ordinates for the border follow from these two points). As we can see from the diagram, the co-ordinates of the bottom left corner of the border are (120,188). The co-ordinates of the top right corner are(1152,992).

The following procedure draws a border around the edge of the defined area. GCOL 0,1 sets the logical colour that will be used for graphics. The first number defines the type of plotting, which will be discussed later, and the second defines the colour. In mode 5, logical colour 1 is normally red. The MOVE commands move the graphics cursor (without drawing) from the origin to the bottom left corner of the border. The DRAW commands that follow it draw straight lines from the current position on the screen to the point specified.

```
2470DEF PROCdraw_border
2480GCOL 0,1
2490MOVE 120,188
2500DRAW 120,992
2510DRAW 1152,992
2520DRAW 1152,188
2530DRAW 120,188
2540ENDPROC
```

THE INTERNAL TIMER

The BBC and Electron have an internal timer that can be accessed easily from BASIC using the reserved variable, TIME. When asked to print the value of TIME, the computer will return a number that corresponds to the time, given in hundredths of a second, since the variable was last set to zero. The procedure 'set-time' prints the word 'Time', its starting value and sets the variable TIME to zero. This procedure is called during the set-up routine and starts the clock for the game.

```
2640DEF PROCset_time
2650PRINTTAB(2,27)"Time      02:00"
2660TIME=0
2670ENDPROC
```

During the main loop of the program, the time displayed on the screen must be updated. To display the time in seconds would be very straightforward; we would simply divide the variable TIME by 100, to convert to seconds, print this value to the screen, and so on. However, it is possible to convert TIME into minutes and seconds by making use of the BBC BASIC commands DIV and MOD. TIME DIV 100 would return the number of seconds as a whole number; (TIME DIV 100) MOD 60 would count the seconds from zero to 59 and then start again from zero. This is because the MOD 60 command gives the value of the remainder after division by 60. So, for example, 63/60 is 1, with a remainder of 3. (63/60) MOD 60 would therefore be 3. The minutes can be similarly isolated and displayed by using (TIME DIV 6000) MOD 60.

This is the procedure that will be used to update the time during the game:

```
2900DEF PROCupdate_time
2910sec$=STR$(((12100-TIME) DIV 100)MOD 60)
2920min$=STR$(((12100-TIME) DIV 6000)MOD 60)
2930REM ** ADD LEADING ZEROS **
2940sec$=LEFT$(zero$,2-LEN(sec$))+sec$
2950min$=LEFT$(zero$,2-LEN(min$))+min$
2960time$=min$+":"+sec$
2970PRINTTAB(11,27);time$
2980ENDPROC
```

As you can see from this procedure, we have gone a stage further. As well as being divided into minutes and seconds, the time will, in fact, be counted backwards from two minutes to zero. In addition, a short string-handling routine is included to ensure that the displays for the seconds and minutes always have two digits, by adding leading zeros if required.

Two other short procedures are still required to complete the setting up of the game. During the game, the player has four lives; therefore, we need to display, at the bottom of the screen, the number of lives remaining. Initially, this will be three lives, displayed as three of the 'assistant' characters we defined in the last instalment (see page 393). A variable 'count' will be used to determine the number of lives used. Initially, this will be one.

```
2690DEF PROCset_men
2700men$=CHR$(226)+CHR$(226)+CHR$(226)
2710count=1
2720COLOUR 1
2730PRINTTAB(2,30);men$
2740COLOUR 2
2750ENDPROC
```

The final set up procedure initialises the scores and displays them on the screen. The value of 'hiscore$' is not set within this procedure as the procedure is called each time the game restarts. Instead, we shall set its initial value at the beginning of the program only.

```
2770DEF PROCset_score
2780score=0:score$="00000"
2790PRINTTAB(2,28)"Score     00000"
2800PRINTTAB(2,29)"Hi score ";hi_score$
2810ENDPROC
```

Now that all the procedures for the setting up part of the program have been assembled, we can construct a higher-level procedure to call them all. In the last instalment we called all the procedures we had assembled directly from a short main program. You must now delete those lines and add these lines:

```
1880DEF PROCsetup
1890COLOUR 2
1900end_flag=0
1910PROCinitialise_variables
1920PROCdefine_characters
1940PROClay_mines(40)
1950PROCdraw_border
1960PROCset_time
1970PROCset_score
1980PROCset_men
1990PROCposition_chars
2000ENDPROC
```

We are now at the stage where we can write a short main program to call the 'setup' procedure and update the time in a REPEAT...UNTIL loop. Add these lines to your program:

```
10 REM **** CALLING PROGRAM ****
20 hiscore$="00000"
30 PROCsetup
40 REPEAT
50 PROCupdate time
60 UNTIL TIME>12099
70 END
```