4. USING SPRITES

4.1 OVERVIEW

When using this sprite package to create a computer game the first stage is to define a number of sprites using the special character definer supplied. Once defined, each sprite is individually stored away on cassette or disc.

The next step is to incorporate selected sprites into the machine code sprite handling routine. In fact there are two alternative handling routines, one for ordinary sprites, and one for super sprites. Super sprites are treated in section 5.5 of this manual. The handling routine containing your own sprites is then re-saved.

You may then proceed to write your game in Basic. This will access the sprite handling routine, which will need to be co-resident in the machine whenever the Basic program is run.

In practice the machine code sprite routines supplied, already contain 7 pre-defined sprites. We suggest that in the first instance you experiment with these before attempting to define and load your own sprites. The sprites supplied are as follows:

Sprite number	Description
1	Pacman shape
2	Man
3	Laser base
4	Two cherries
5	Monster
6	Monster
7	Monster

It is very easy to display and move sprites around the screen, and the next sections deal with the precise way in which this is achieved.

In preparation for experiments with the various sprite calls you should first load in the sprite handler, complete with default sprite definitions. To do this type:

```
*RUN M/CODE
```

This is the same for cassette or disc, but cassette users should not that M/CODE is the second file on the tape. It is also repeated a number of times later in the tape.

4.2 MEMORY USAGE

One of the first things that your Basic program should do is to enter mode 2 - the sprites only function in mode 2 - and reserve memory for the machine code sprite routine already loaded. It resides between &2800 and &3000, and is protected by setting HIMEM to &2800 after any mode change.

Early in your program a line similar to the following should therefore appear:

100 MODE 2:HIMEM=&2800 (&2500 for super sprites, see section 5.5)

HIMEM should be reset in this way after every mode change in your program. Failure to do this could cause corruption of the machine code routines.

4.3 INITIALISATION

One other essential is to initialise the sprite handler. This achieved by a call to P%. Thus all sprite programs should contain the following two lines at an early stage:

```
100 MODE 2:HIMEM=&2800 (&2500 for super sprites)
110 CALL P%
```

4.4 USE OF VARIABLES

The sprite routines use the static integer variables to pass to and from Basic (ie. A% to Z%). Your Basic program should not use these for purposes other than those outlined below; and they should not be directly used as a loop parameter in FOR NEXT loops. Of course any other integer variables (eg. AA%, a% and so on) may be freely used.

4.5 REFERENCING A SPRITE

Each of the seven individual sprites have separate integer variables reserved for the x and y co-ordinates of their location on the screen. These are as follows:

Sprite number	X co-ord	Y co-ord
1	A%	B%
2	C%	D%
3	E%	F%
4	G%	H%
5	I%	J%
6	K%	L%
7	M%	N%

The x coordinate may vary between 2 and 151, and the y coordinate between 2 and 139. There is an automatic wrap-around if values are out of these ranges. The wrap-around parameters may also be altered by the user; see section 4.11.

4.6 DRAWING A SPRITE ON THE SCREEN

This is very easy to do. W% is used to tell the computer which sprite you wish to draw, and S% is used to call the sprite plotting routine.

```
eg 40 W% = 2
   50 C% = 30:D% = 40
   60 CALL S%
```

would plot sprite 2 at (30,40)

If the sprite was already on the screen at some other location, plotting it in a new position (as in the above code) automatically erases the first image. Hence you do not need to worry about deleting old images as you cause movement, it is all done for you.

If you wish to try this, *RUN M/CODE as described in section 4.2, then run the following program:

```
  0 REM PROG1
 10 REM USES S% TO POSITION
 20 REM SPRITE NO 2
100 MODE2:HIMEM=&2800
110 CALLP%
120 W%=2
130 C%=30:D%=40
140 CALLS%
```

4.7 MOVING A SPRITE

There are 2 ways of doing this:

A. Plot the sprite on the screen, update the integer variables controlling its location, and simply plot it on the screen elsewhere - as described in section 4.6 above.

The program below uses this principle in conjunction with a FOR NEXT loop to move a sprite across the screen. If you wish to try it, *RUN M/CODE before running the program, as described in 4.2 above.

```
  0 REM PROG2
 10 REM USES S% TO MOVE
 20 REM SPRITE NO 2
100 MODE2:HIMEM=&2800
110 CALL P%:?&2EBE = 8
120 W%=2
130 D%=100
140 FOR AA%=2 TO 50
145 C%=AA%
150 CALLS%
```

```
160 NEXT
```

B. Use the special programmable directions, and CALL T%. This is achieved as follows:

```
W% = Sprite Number
Z% = Direction Number
CALL T%
```

eg

```
50 W% = 4
60 Z% = 5
70 CALL T%
```

would move sprite 4 in direction 5. This call assumes that the starting coordinates of the particular sprite are already defined. If this is not the case a statement of previous position should be made before the call; eg. G%=50:H%=100.

The directions for the call to T% are initially defined as follows, but may be reprogrammed as required:

```
 \    I     /
 1    2     3
-4    *     5-
 6    7     8
/     I     \
```

Note that moving the sprites using this method will automatically update the relevant position vectors. Thus in the above example with sprite 4, G% and H%, would have been automatically updated to reflect the sprite's new position.

As an example the following program uses this call to move a sprite randomly around the screen.

```
 0 REM PROG3
10 REM USES T% TO MOVE
```

```
  20 REM SPRITE NUMBER 2
 100 MODE2:HIMEM=&2800
 110 CALLP%
 120 C%=30:D%=40
 130 REPEAT:PROCRAND:UNTIL FALSE
1000 DEF PROCRAND
1010 W%=2
1020 Z%=RND(8)
1030 CALLT%
1040 ENDPROC
```

4.8 REDEFINING A PROGRAMMABLE DIRECTION

As mentioned above, it is possible to move a sprite using the special programmable directions and the variable T%. Initially these are defined to move one unit in each of the 8 possible directions, but these may also be reprogrammed as required. This is done as follows:

Z% = Number of direction to be reprogrammed
X% = desired positive movement on the x axis
Y% = desired positive movement on the y axis
CALL R%

So to alter direction 2, to move the sprite up 3 units, rather than the default value of 1, use:

```
Z% = 2
X% = 0
Y% = 3
CALL R%
```

Note that negative values of X% and Y% used in this way, will cause negative movements; that is right to left or top to bottom, respectively.

4.9 DELETING A SPRITE

As already mentioned the old image of a sprite is automatically deleted, when the sprite is redrawn elsewhere. However, you may wish to totally remove a sprite from the screen, and to do this simply set W% to the value of the sprite plus 256, and call S%.

For example, to delete sprite 3, use:

```
50 W% = 259
60 CALL S%
```

4.10 DELAY ROUTINES

Because of the high speed of movement provided by the sprites, you may need to slow them down a little to prevent them from shooting across the screen too quickly. This is fairly easy to do in Basic with simple REPEAT loops, however a machine code routine to do this is also included for your use.

SET X% to the required delay (from 1 to 255)
CALL O% (ie the letter O - not zero)

4.11 BOUNDARY DEFINITIONS

If an attempt is made to plot a sprite outside a given boundary on the screen, the program will automatically sense this and plot the sprite on the other side of the screen, producing a 'wrap around' effect. These boundaries are normally set to the edges of the screen, but may be reset using the following:

Boundary	Instruction	Range of Value
left side	?&85=value	(2 - 151)
right side	?&86=value	(2 - 151)
bottom side	?&87=value	(2 - 239)
top side	?&88=value	(2 - 239)

Thus for example, the following line:

```
100 ?&85 = 50
```

will reset the left boundary to 50.

The default values for these parameters are 2, 151, 2, and 239 respectively. To reset the four parameters to the default values, use:

```
CALL V%
```

Any call to P% will also reset them; though this will also reset other variables.