5. ADVANCED FEATURES

It is advised that the features covered in this section should only be used once familiarity has been gained with those outlined in section 4.

5.1 CLONES

Each of the 7 sprites may have up to 2 additional clones (ie exact copies) of themselves on the screen at any time. Hence there may be up to 21 independently moving objects on the screen at once.

Each clone is allocated an identity, and it is with this that it is identified, called and moved. These are as follows:

Primary Sprite Number	First Clone Number	Second Clone Number	X co-ord	Y co-ord
1	9	17	A%	B%
2	10	18	C%	D%
3	11	19	E%	F%
4	12	20	G%	H%
5	13	21	I%	J%
6	14	22	K%	L%
7	15	23	M%	N%

As you will observe, the variables used to define the position of the clones, are the same as those used to control the main sprite. For example A% and B% are used to indicate the position of sprite 1 and both of its clones (sprites 9 and 17).

Hence, to move the sprite and both of its clones will require care on your behalf to ensure that you do not call, say, sprite 1 and then sprite 9, without updating the values of A% or B% to the new position of sprite 9.

5.2 ANIMATION

If the sprite that you are moving is, say, a man walking along, simply redrawing him in different positions will give the appearance of him gliding along - not actually walking. This is not important for moving shapes such as a pacman, car or tank, but is essential for moving, say, a flying bird or a man climbing a ladder.

The BEEBUG sprite pack takes care of this automatically. As mentioned earlier, the x co-ordinate of a sprite's location may vary between 2 and 151, however in actual fact there are only half this number of different screen locations. Consequently, a sprite will appear at the same position on the screen, if its x co-ordinate is, say, 10 or 11. This fact is made use of to place alternate images of a sprite at the same position so as to simulate animation.

When using the sprite definer, (as explained later in this manual) it is possible to create 2 different versions of the same sprite. (Do not confuse this with clones or super sprites). If you do this, the first image of the sprite will be displayed whenever the x co-ordinate of the sprite is an even number, and the second image displayed on odd numbers.

5.3 CHECKING FOR A COLLISION BETWEEN SPRITES

In most games, it is essential to know when one object on the screen hits another, for example a bullet hitting an alien or pacman hitting a monster. The continual checking for crashes in Basic, will tend to slow down the operation of any but the most simple programs. The BEEBUG sprite pack includes a special machine code routine, which is quickly called from Basic, to do this you.

To check for a collision between sprite number m and sprite number n, use the following:

```
W%=m
Z%=n
CALL Q%
```

Now simply test for the value of X%. If it is unity, then a collision has occurred, and appropriate action can be taken. The routine defines a crash as any overlap involving a central area of a given sprite of 4 pixels wide by 16 pixels high. This is something of a compromise in that sprites defined by the user will be of varying size.

It is however possible to increase the area used for the collision routine to cover the full 8 by 16 sprite size. To do this, type the following:

```
?&2EBE=8
```

Once you have done this, you can re-save your sprite handling routine (whether for normal or super sprites), and the modification will be saved along with it. To return to the default value, use:

```
?&2EBE=4
```

The collision checking routine contains a further facility to assist here. Once Q% has been called, Y% will return a value depending on the accuracy of the collision. A value of 255 indicates no crash, while lower values indicate progressively greater overlaps between the two colliding sprites.

Note that if either of the two sprites concerned are clones, you must ensure that their position vectors hold their correct x and y coordinates before any collision check can be made; and that W% or Z% holds the number of their parent sprite (rather than the clone number).

The program below moves two sprites randomly around the screen. When a collision is detected a noise is sounded. There is a continuous printout of the value of Y% on the screen, to demonstrate the way in which this parameter can measure the closeness of a collision.

```
 0 REM PROG4
10 REM MOVES TWO SPRITES RANDOMLY,
  20 REM AND CHECKS FOR CRASHES
 100 MODE 2:HIMEM=&2800
 110 CALL P%
 120 VDU 23;8202;0;0;0
 130 A%=100:B%=100
 140 C%=100:D%=100
 150 REPEAT
 160   time%=TIME + RND(600)
 170   Z1%=RND(8)
 180   Z2%=RND(8)
 190   REPEAT
 200     PROCmove1
 210     PROCmove2
 220     PROCcrash
 230     UNTIL time%<TIME
 240 UNTIL FALSE
1000 DEFPROCmove1
1010 W%=1
1020 Z%=Z1%
1030 CALL T%
1040 ENDPROC
1050 DEFPROCmove2
1060 W%=2
1070 Z%=Z2%
1080 CALL T%
1090 ENDPROC
1100 DEFPROCcrash
1110 W% = 1
1120 W% = 2
1130 CALL Q%
1140 IF X%<>0 THEN SOUND &10,-15,6,4:time%=0
1150 PRINT CHR$(30);Y%;" "
1160 ENDPROC
```

5.4 ALLOCATING SPRITES

You may reach a point in your game, when you need to have several versions of the same sprite moving on the screen at the same time. As already mentioned, there are 7 available sprites which may be defined as required, and so one method to achieve this would obviously be to define sprites 1, 2 and 3 (say) as the same character. Another method would be to use clones, as already explained in this manual.

However, a third method is available which is particularly useful if your game requires all 7 sprites to be defined as different characters, and then requires the display of, say 3 or 4 versions of one particular sprite. Quite simply, calling U% will enable a sprite to be displayed not as itself but as one of the other sprites. This is known as 'allocating a sprite', and is achieved as follows:

W% = Number of sprite to be allocated
Z% = Number of sprite whose image is to be copied
CALL U%

For example, if sprite 1 is a monster and sprite 2 is a man;

```
50 W% = 1
60 Z% = 2
70 CALL U%
```

will cause both sprites 1 and 2 to display the man.

Note that it is advisable to delete previously displayed images of the sprite to be replaced.

Clones take on the new forms of their allocated parent sprite. It is possible to allocate one sprite shape to more than one sprite, and likewise to have a sprite shape that is not allocated to any sprite. Merely because any sprite shape is not allocated to any sprite number at a particular time does not mean that it no longer exists: it will just remain idle until required.

5.3 SUPER SPRITES

Normal sprites may have two alternative and alternating manifestations. This allows simple animation effects. A super sprite has four such manifestations and is created as a pair of normal sprites. Each normal sprite in a super sprite is called a phase, and the user determines which phase is being displayed at any given moment. As an example, the sprite could be used to represent a running man. By defining one phase with him facing left and the other phase with him facing right it is possible to have him facing the same way as he moves, all using one sprite. Furthermore it would be easy to introduce animation into both phases using the odd/even effect on the x co-ordinate.

It is not possible to combine the use of normal sprites and super sprites in one program since super sprites require a separate machine code handling routine. To access the super sprite code on cassette or disc, use the following:

```
*RUN SS/CODE
CALL P% (sets default values)
```

Cassette users should note that SS/CODE is the eighth file on the cassette.

To protect this code it is necessary to set HIMEM to &2500 whenever the mode is changed, rather than &2800 for normal sprites.

Many of the commands for super sprites are similar to those for normal sprites. However it is not possible to switch sprite shapes and sprite numbers in the same way as is possible with ordinary sprites.

When defining the sprites for use in SS/DEMO each sprite is allocated two phases. For example sprite one has a phase facing left, and a phase facing right. These two phases are defined and saved quite separately, as explained in section 6 below.

It is possible to determine which phase of a given sprite will be drawn using a call of the following kind:

```
W% = Super sprite number
Z% = Phase (1 or 2)
CALL U%
```

For example:

```
10 W% = 3
20 Z% = 2
30 CALL U%
```

This will allocate phase 2 to super sprite 3. All further manifestations of super sprite 3 will be in phase 2 until it is altered by another call to U%. Note that this call has a different effect when using normal sprites.

It is advisable to delete the relevant supersprites before changing their phase.