SPRITES

GAME WRITERS'
UTILITY PACK

FOR THE BBC MICRO

SUPPLIED ON
CASSETTE OR DISC

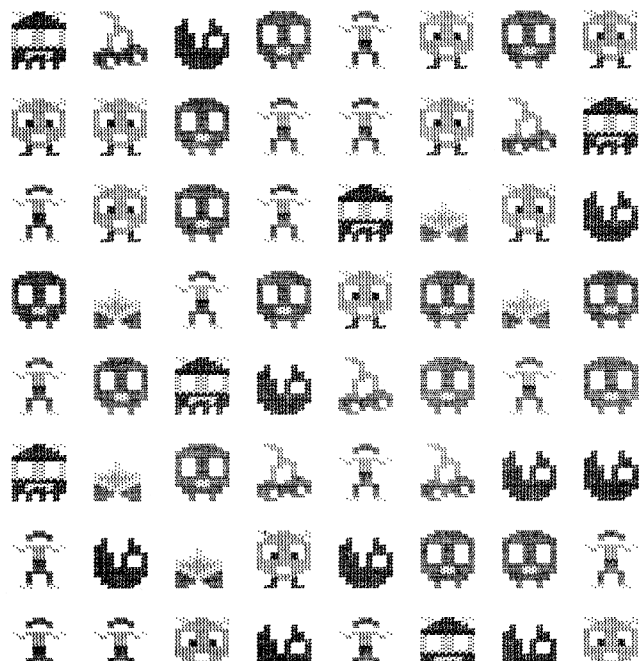
BEEBUGSOFT

Published by BEEBUG Publications Ltd.,
PO BOX 50, St Albans, Herts, England.

SPRITES

BEEBUG GRAPHICS PACKAGE FOR THE BBC MICRO MODEL B

by Stephen Allen



CONTENTS

1 INTRODUCTION

2 FEATURES OF THE SPRITES PACKAGE

3 GETTING STARTED

4 USING SPRITES
- 4.1 Overview
- 4.2 Memory usage
- 4.3 Initialisation
- 4.4 Use of variables
- 4.5 Referencing a sprite
- 4.6 Drawing a sprite
- 4.7 Moving a sprite
- 4.8 Redefining directions
- 4.9 Deleting a sprite
- 4.10 Delay routines
- 4.11 Boundary definitions

5 ADVANCED FEATURES
- 5.1 Clones
- 5.2 Animation
- 5.3 Checking for collisions
- 5.4 Allocating sprites
- 5.5 Super sprites

6 DEFINING SPRITES
- 6.1 Using the definer
- 6.2 Incorporating User Sprites

7 DEMONSTRATION PROGRAMS

APPENDIX

Programs supplied on cassette

1. INTRODUCTION

The major difference between games written in Basic (or other high level languages) and those written in assembly languages is their speed of operation. This is partly due to their graphics handling. In Basic it is simply not possible to plot intricate objects quickly; even the method of plotting user defined graphics on the screen is clumsy since each colour must be plotted separately. The usual result is that Basic games tend to have jerky graphics, or if they are smooth, the game must be kept simple. Sprites provide an easy way of generating smooth multicoloured graphics.

This sprite package contains routines for creating and moving sprites around the screen at will, for creating sprite clones, and super sprites giving enhanced animation facilities. Special routines are included to check if a sprite collides with another, and all routines are accessible from Basic, allowing arcade style games to be written with a minimum of code. The package also contains a set of example programs.

2. FEATURES OF THE BEEBUG SPRITES PACKAGE

1. Sprites are quickly created on an 8 by 16 grid, using a special sprite definer program. Each of the 8 by 16 (128) individual parts which make up the sprite may be any of the 16 colours available in mode 2 (8 steady + 8 flashing).

2. Each sprite is easily displayed or moved, using integer variables. For example the following sequence:

```
W%=1: A%=30: B%=40:  CALL S%
```

will draw sprite 1 at location (30,40) on the screen ie 30 positions along from the left of the screen, and 40 positions up.

3. Plotting a sprite at a new position will automatically delete its previous image, and this is achieved in such a way that backgrounds are left unchanged.

4. The sprite routines permit the use of an automatic wrap-around screen so that a sprite moving off the left of the screen, will reappear on the right.

5. A special machine code routine will automatically check for collisions between sprites, allowing you to take appropriate action.

6. Each sprite may be defined twice, and a special facility causes one image to be displayed if the x co-ordinate is even, and the other if it is odd. This is quite automatic, and may be used to provide animation effects; for example to create the impression of a man running.

7. Up to 7 different sprites may be used and moved around the screen independently. Each of these may be defined twice as above. As well as this, each sprite may have up to 2 clones, (ie exact copies) of itself, allowing up to 21 moving sprite pairs on the screen at once.

8. A super sprite facility is also incorporated, allowing 4 variations of each sprite to be created rather than the normal two. The image chosen to represent a super sprite at a given time could be made to depend on the direction of travel, so that, for example, a monster could automatically face the direction in which it was moving.

3. GETTING STARTED

The tape or disc supplied with this pack contains a number of programs. These are as follows:

1. An overview and an introductory display of sprites moving about the screen.

2. A sprite definer program.

3. A machine code sprite routine, into which you may load your sprite definitions.

4. An alternative machine code routine, for use with super sprites.

5. Seven demonstration programs, to show exactly how to use various features of the sprite pack.

These features will be explained in the course of this manual, but to get started, you may care to run the Introduction file on the cassette or disc. This makes a few general comments about sprites, and displays a number of sprites on the screen.

Cassette users should do this by typing:

```
CHAIN ""
```

and pressing Return. Cassette users should refer to the Appendix for a full list of the programs on the tape.

Disc users should press Shift-Break (ie. hold down the Shift key, then hit Break), to display the menu, and then select option 1.

At this point you may also like to look at the demonstration programs which form a part of the package. In this case please refer to section 7 of this manual.

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 note 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 is 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 239. 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:

```
 Ø REM PROG1
 1Ø REM USES S% TO POSITION
 2Ø REM SPRITE NO 2
 1ØØ MODE2:HIMEM=&28ØØ
 11Ø CALLP%
 12Ø W%=2
 13Ø C%=3Ø:D%=4Ø
 14Ø 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.

```
 Ø REM PROG2
 1Ø REM USES S% TO MOVE
 2Ø REM SPRITE NO 2
 1ØØ MODE2:HIMEM=&28ØØ
 11Ø CALL P%: ?&2EBE = 8
 12Ø W%=2
 13Ø D%=1ØØ
 14Ø FOR AA%=2 TO 15Ø
 145 C%=AA%
 15Ø CALLS%
 16Ø NEXT
```


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

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

eg

```
5Ø W% = 4
6Ø Z% = 5
7Ø 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%=5Ø:H%=1ØØ.

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.

```
  Ø REM PROG3
 1Ø REM USES T% TO MOVE
 2Ø REM SPRITE NUMBER 2
 1ØØ MODE2:HIMEM=&28ØØ
 11Ø CALLP%
 12Ø C%=3Ø:D%=4Ø
 13Ø REPEAT:PROCRAND:UNTIL FALSE
 1ØØØ DEFPROCRAND
 1Ø1Ø W%=2
 1Ø2Ø Z%=RND(8)
 1Ø3Ø CALLT%
 1Ø4Ø 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 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% = Ø
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:

```
5Ø  W% = 259
6Ø  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.

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% and 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 a 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 for 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 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 CHRASHES.
100 MODE 2:HIMEM=&2800
110 CALL P%
120 VDU 23;8202;0;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 Z% = 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.

5.5 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 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 &25ØØ whenever the mode is changed, rather than &28ØØ for normal sprites.

Many of the commands for super sprites are similar to those for normal sprites. However it is not posible 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:

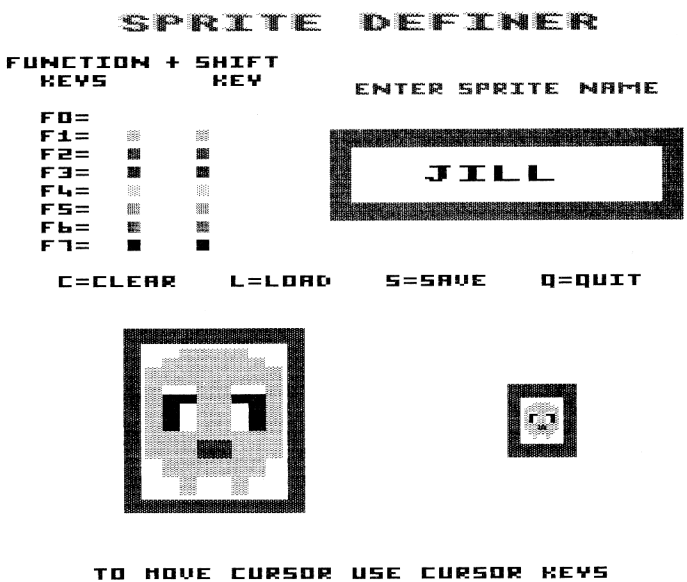
```
W% = Super sprite number
Z% = Phase (1 or 2)
CALL U%
```

For example:

```
1Ø W% = 3
2Ø Z% = 2
3Ø 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.



6. DEFINING SPRITES

6.1 USING THE SPRITE DEFINER

Sprites are each defined separately, and each is saved as a separate file. It is therefore possible to build up a library of sprites, which can be selected independently for use in programs. Instructions on how to load sprites into a program can be found in section 6.2 below.

Each sprite is defined in two parts, the first being displayed when the X co-ordinate of the sprite's position is even, and the second when it is odd. The purpose of this feature is to allow for the employment of animation techniques, eg. to generate a running man or a flying bird. When using the sprite definer, you can select which is being displayed by typing "1" for part 1 and "2" for part 2. Each is defined on an 8 by 16 grid. Unless the second part is defined, it is assumed to be identical to the first.

Super sprites have four parts. These should be considered as two pairs (or phases) of normal sprites amalgamated into one. Super sprites are defined by defining two normal sprites, each of which may have two parts as usual. The two normal sprite definitions are saved exactly as for normal sprites. But when they are loaded back into the machine code sprite handler, a different piece of code is used (ie. SS/CODE rather than M/CODE), and the two pairs are saved into it separately. See section 6.2, which gives the load addresses for the two phases of each super sprite.

To define a sprite proceed as follows:

(a) Load the sprite definer program. Disc users should select it from the menu, and tape users should load it with

CHAIN "DEFINE"

It is the fifth file on the cassette.

(b) Press "1".

(c) Move the cursor (a small dot in the left-hand grid) by pressing the cursor control keys; and press the relevant function keys to paint the square at the position of the cursor as required. Continue until the sprite is complete.

(d) If the second part of the sprite is required to be defined differently, then press "2" and repeat (c).

(e) Press "S" to save the sprite on tape or disc (see the end of this section for more precise details).

(f) If you are creating a super sprite, repeat (b) to (e) above to generate and save (separately) its second phase.

Note that you may clear the grid to any chosen colour by selecting a colour with the function keys, and then pressing "C".

	Key alone	Key with shift
f0	black	black/white
f1	red	red/cyan
f2	green	green/magenta
f3	yellow	yellow/blue
f4	blue	blue/yellow
f5	magenta	magenta/green
f6	cyan	cyan/red
f7	white	white/black

The colours generated with the shift key and a function key are all flashing colours.

NOTES

The smaller square grid shows the sprite at actual size.

To alter a previously defined sprite, type "L" (for Load), load the sprite, alter it, and save it again.

Saving sprites on tape.

(a) Make sure that there is a clean part of the tape in the tape recorder. Any tape can be used except the one containing the sprite package.

(a) Press the S key.

(b) Type in the name of your SPRITE (up to seven letters).

(c) Press the RECORD button on your tape-recorder.

(d) Press RETURN.

(e) After the sprite has been saved press the STOP button on your tape-recorder.

Loading sprites from tape back into the definer.

Once your sprite has been recorded on tape it can be loaded back into the sprite definer program at any time to be altered.

(a) Load the sprite definer program into your computer.

(b) Line up the start of the part of the tape on which you have recorded your sprite.

(c) Press the L key.

(d) Type in the name that you gave to the sprite.

(e) Press the play button on the tape-recorder.

(f) After your sprite has loaded press the STOP button on the tape-recorder.

Saving sprites on disc.

(a) Insert a spare formatted disc into the disc drive. If a double drive is being used, drive Ø should be selected.

(b) Press the S key.

(c) Type in the name of your SPRITE (up to seven letters).

Once your sprite has been saved on disc it can be loaded back into the sprite definer program at any time to be altered.

Loading sprites from disc back into the definer

(a) Load the sprite definer program into your computer.

(b) Press the L key.

(c) Type in the name that you gave to the sprite.

6.2 INCORPORATING USER DEFINED SPRITES

Before you use your sprites in a Basic program they must be placed in the relevant machine code sprite handling routine (M/CODE for normal sprites, SS/CODE for super sprites). This subroutine must then be re-saved containing the new sprites, so that it can run together with the main Basic program. It is advised that the altered program is saved under a different name. The operation should be performed in mode seven, with HIMEM set to &2800 for normal sprites and &2500 for supersprites.

(a) Type MODE 7: HIMEM = &2800 (or &2500)

(b) Run the machine code subroutine using:

*RUN M/CODE (normal sprites)

*RUN SS/CODE (super sprites)

Cassette users should note that M/CODE is the second file on the cassette, and SS/CODE the eighth.

(c) Load in the required sprites which should have been previously saved on tape or disc by typing:

*LOAD name nnnn

where name is the sprite name
and nnnn is the relevant address (see table below)

Normal sprites	sprite number	nnnn
	1	2800
	2	2880
	3	2900
	4	2980
	5	2A00
	6	2A80
	7	2B00

Super sprites	Sprite number	nnnn (phase 1)	nnnn (phase 2)
	1	2500	2580
	2	2600	2680
	3	2700	2780
	4	2800	2880
	5	2900	2980
	6	2A00	2A80
	7	2B00	2B80

If you only want to use two sprites in your Basic program for example then only load the two sprites that you require into the machine code routine. The remaining sprites will remain the default designs.

(d) To re-save the machine code program that now contains the redefined sprites type:-

*SAVE name 2800 +900 3000 (for normal sprites)

*SAVE name 2500 +C00 3000 (for super sprites)

This saved program will now contain the redesigned sprites and can be run exactly as if it were the original M/CODE or SS/CODE.

7. DEMONSTRATION PROGRAMS

Included in the sprites package is a series of seven demonstration programs, of which six (1-5 and 7) deal with ordinary sprites and one (number 6) deals with super sprites. Each program is listable, and the user is strongly recommended to experiment with these programs, trying to alter speeds, directions of movement, sprites plotted and so on. These programs are an ideal way to increase familiarity with sprites quickly. The user will then be better equipped to design programs of his own.

Users should note that the final space-bar pressing in each demonstration causes the program to be listed on the screen. To repeat the demonstration, just type RUN. When examining the program, please note that the function of the last five lines of each is to perform the auto-listing.

Cassette users:

All the demostration programs, with the exception of DEMO:6 and DEMO:7 may be run using:

```
*RUN M/CODE
CHAIN "DEMO:n"    n is the program number
```

DEMO:6 is run by:

```
*RUN SS/CODE
CHAIN "DEMO:6"
```

and DEMO:7 by:

```
*RUN CODE_2
CHAIN "DEMO:7"
```

Disc users

Disc users should call the programs from the menu. At the end of each demonstration the easiest way to move to the next is to access the menu again using Shift-Break.

The functions of the various programs are summarized briefly below.

DEMO:1 This simply plots a number of sprites on the screen, without movement.

DEMO:2 This moves the sprites around the screen, using the facility for displaying two images of one sprite to create an effect of animation for the cherry. The two images are plotted depending on whether the x co-ordinate of the position is even or odd. See section 5.2.

DEMO:3 This moves the seven main sprites (1 - 7) around the screen. It also demonstates how it is possible to allocate the same shape to more than one of the sprites. All the clones (9 - 15, 17-23) take the same form as the sprites of their respective base numbers.

DEMO:4 This shows the movement of all of the sprites and all of the clones. The program must keep a record of where each sprite is, and this is done using PROCswapin and PROCswapout which assign the actual clone positions to their relative variables.

DEMO:5 This generates a repeat pattern of sprites. Normally, unless you use clones, it is not possible to have the same sprite in more than one position on the screen, since plotting it in a second position automatically deletes it from the first. A way round this is to re-initialize the variables every time the character is to be re-plotted. A CALL to P% achieves this.

DEMO:6 This demonstates the use of super sprites, and shows one moving around the screen facing left or right depending on the direction of travel. There are other super sprites already defined in the machine code program and the user may like to experiment with them.

DEMO:7 This is a short sequence showing the use of sprites in combination with plot and fill graphics.

APPENDIX

PROGRAMS SUPPLIED ON THE CASSETTE

The programs are located on the tape in a convienient order. Where two programs are required in sequence they are placed in order. The files are as follows:

Introduction	INTRO	
	(M/CODE)	(Normal sprite code)
	(PART_1)	
	(PART_2)	
Sprite definer	DEFINE	
	(DEF_1)	
	(DEF_2)	
Super sprite code	SS/CODE	
Demonstrations	M/CODE	
	DEMO:1	
	M/CODE	
	DEMO:2	
	M/CODE	
	DEMO:3	
	M/CODE	
	DEMO:4	
	M/CODE	
	DEMO:5	
	SS/CODE	
	DEMO:6	
	CODE_2	
	DEMO:7	

If the program name is enclosed in brackets this means that it is automatically called by a previous program.

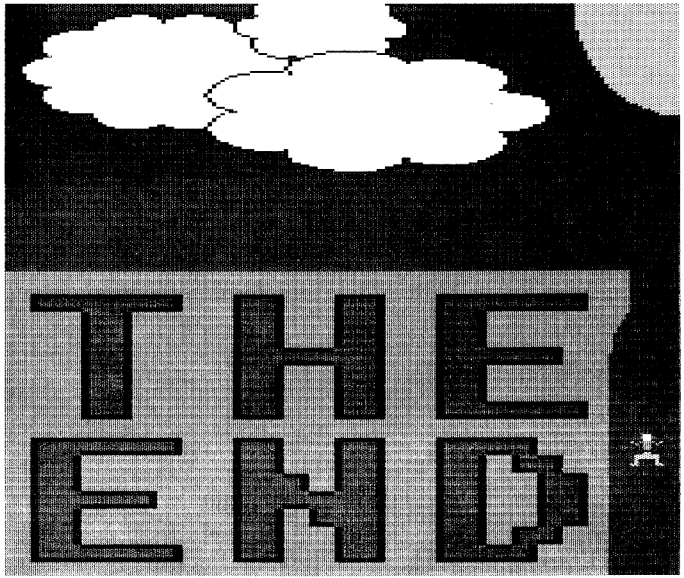
The intoduction should be loaded by

```
CHAIN "INTRO"
```

and the sprite definer by:

```
CHAIN "DEFINE"
```

Access to the machine code routines, and to the sprite definer are described in sections 4 and 6 respectively.



NOTES