Sprites Alive Manual - Basic Version

Submitted by Tommy Pereira on 24 September 2003
Kindly scanned and PDFed by Steve R Sopp

"Thank you Kevin Thacker for putting it online for everyone
on the - Unofficial Amstrad WWW Resource"

-=-

[//\ //\//\ | G O]

http://mondodizzy.members.easyspace.com/
Kiss My ARSEnal - Pixel Art Drawrings

http://dizzypetition.members.easyspace.com/
Bring Back that Loveable EGG!

MAIN MENU

THE SUPERVISOR

THE DESIGNER

I would like to take this opportunity to thank Dawn, my long suffering girlfriend. Without her patience this program would never have been created.

THE SUPERVISOR

GLEN COOK

SPRITES SUPERVISOR PROGRAM

INTRODUCTION

The CPC range of Amstrad computers is supplied with an excellent hardware specification. However the BASIC supplied with the computer, good as it is, does not show off the computer's full potential in the graphics and sound departments.

This suite of programs will remedy this by extending the BASIC with a number of extra commands. These extra commands will allow the user to manipulate graphics to a far greater extent than was ever possible using the BASIC supplied.
On the disc there are a number of demonstration programs written entirely in BASIC to show you the type of program that you will be able to create for yourself. You do not need to have any knowledge of machine code.
Try the demonstrations for yourself.

RUN "DEMO1" and press ENTER

There are five demonstration programs on the disc. They are called DEMO1, DEMO2, DEMO3, DEMO4 and DEMO5. After reading through this manual and getting used to the extra commands you will be able to write programs of the same sort of quality as this.

The program has the following features:-

1) 64 User definable sprites

2) Animation sequencing

3) Smooth pixel movement

4) True collision detection

5) Use of mode 0 or mode 1

6) Joystick or Keyboard control

7) Comprehensive sprite designer

8) Automated commands

9) Simple to understand instructions

The program has been designed to eliminate large amounts of BASIC, thereby making the programs easier to write and understand. This is done by making the sprite program do most of the decision making for you. It is quite possible to write a game that only has three or four lines of BASIC for the main loop. The rest of the program would set up the sprite and music data.

If you take a look at the listings of the demonstration programs you will be able to see just how simple they are. You should be able to write. programs just like them in no time at all.

AN OVERVIEW OF THE SPRITES AND DRAWINGS

It is very important that you understand this page before you start to use the sprite program. Take your time to fully familiarize yourself with the points made on this page. Without understanding the basics of sprites and this program you will find the program impossible to use to its full extent.

It is important to familiarize yourself to the difference between a drawing and a sprite. The drawing is the actual shape of the character that will be stored in memory. The sprite is not an actual drawing but it can be linked to a drawing. That may sound a bit confusing but I will give you an example to try and explain what I mean.

Please bear in mind that every individual character on the screen is a separate sprite, be it a spaceship, a man or a missile.

Take as an example the game of Breakout, the game where a ball bounces around the screen knocking out blocks at the top. You control a bat at the bottom of the screen and try to keep the ball in play by bouncing the ball off your bat. To simplify things all the blocks at the top of the screen are the same shape, size and colour. Each of the blocks at the top of the screen are individual sprites, the ball is a sprite and the bat is a sprite.

We could have 40 blocks at the top of the screen, say four rows of ten. Each of those 40 blocks is a sprite. in total we have 42 sprites on the screen at the start of the game (blocks,bat and ball). However we would only have to define 3 drawings, that of the block, the bat and the ball. Even though we have forty blocks they are all identical so we simply link all 40 sprites to the one drawing.

I hope that starts to make sense. There is no point in storing all 40 blocks in memory because it is just a waste of memory.

Taking the idea one step further. We have a man walking across the screen in an animated sequence. That man is a single sprite. However to get the animation effect we have to swap the drawings of the man everytime that he is moved. This is a case of 1 sprite being linked to a number of drawings.

As another example, consider a printing press. The press has been set to produce a picture of a girl. From that one drawing we can have a number of copies printed onto paper. Your drawing stored in memory is the printing press and the copies on paper are your sprites. We only need one drawing to produce any number of sprites from it.

Please remember the difference between your drawings and your actual sprites. Any confusion you have will clear once you have used the program a number of times. Practice does make perfect !!

SPRITES SUPERVISOR PROGRAM

IMPORTANT NOTES ON USING THIS PROGRAM

You may use the Sprites In either Mode 0 or Mode 1.

The screen PAPER should always be set to zero. You may choose any colour you like for the paper colour by using the command INK 0,n. If you use the PAPER command with a variable other than zero the sprites will not operate correctly.

When using the sprites you MUST NOT scroll the screen. If you want to print something on the bottom line of the screen you should use a semi-colon after the information you want printing. If the screen does scroll the sprites coordinate system will not work correctly.

When you are linking the colours to the inks with the INK i,c you can not use flashing colours. If you try to use flashing colours an error message will be displayed.

The coordinate system used within the Supervisor program is very similar to the graphics coordinate system the Amstrad uses; 0,0 being the bottom left hand edge of the screen. The differences between the two systems are as follows.

The Amstrad system has a number of coordinates on the screen pointing to the same pixel.

ie Mode 0 : 0,0 : 1,0 : 2,0 and 3,0 are the same points on the screen.

he supervisor system has a different point on the screen for each individual coordinate.

ie Mode 0 : 1,0 is a different point to 0,0.

This applies to both the X and Y coordinates. The maximum screen coordinates for each screen and mode can be found in Appendix 1.

Some of the commands described in the manual will pass values back to BASIC. These commands will have the '@' character before the variable name. It is VERY important that you include this character when it is shown in a command. Failure to do this could result in your program being corrupted and the computer may crash.

It is important that you save your program before you 'RUN' it. If you have made a mistake and you loose your program it is very frustrating to have to type it back into the computer again.

You MUST use either the |ERASE or the |RESET command as the first instruction within your program.

If you are going to use a different size screen to the standard Amstrad screen, the LOCATE command will not work correctly. You should use trial and error to find the correct value for the LOCATE command.

SCREEN AND DRAWING COMMANDS

|SCREEN,n

This command will allow the user to change the shape of the screen. There are nine different screen sizes to choose from, these range, from |SCREEN,0 to |SCREEN,7. To return to the standard screen simply use the |SCREEN command with no number after it. You should experiment with this command with the border set to a different colour to the paper so you can see the exact shape the screen will take. The sizes of the different screens can be found in appendix 1.

NOTE The default setting is the standard Amstrad screen.

|DGET,d,x,y

This is a command that will get the drawing into the computers memory. d is the drawing number, x and y are the drawing dimensions. To get a drawing into memory the drawing must be put into the top left hand corner of the screen. You can do this by either defining characters and printing them in the top left hand corner, or by using the PLOT command to put the character into the top left hand side.

You should then use the |DGET command to store the drawing. The x and y values should be the width and height of the drawing in pixels. In order to conserve as much memory as possible, the top of the drawing should be touching the top of the screen and the left hand edge of the drawing should be touching the left hand edge of the screen. The rest of the screen should be completely clear of any graphics. It is very important that there are no graphics to the right hand edge of the drawing you are going to store.

If you are using the sprite designer program, which we strongly recommend, you will not need to use this command.

NOTE : The correct mode must be set, ie MODE 0 or MODE 1. Also the size of the screen must be set to its default size ie |SCREEN.

You can define from drawing 0 up to drawing 63. The maximum size of a drawing is 32 x 32 pixels.

Please remember that the x and y dimensions need to be specified in pixels. DO NOT use the Amstrad coordinate system.

Before you get any of the drawings into memory, you should ensure that you have reserved enough memory with the MEMORY command. It you do not reserve enough memory you may corrupt your program.

To calculate the amount of memory you will need please, see the equation on the next page.

MEMORY CALCULATION OF DRAWING SIZE

If you are going to get the drawings into memory by using the |DGET command you will need to know how much memory the drawings are going to take up. You will need to do this in order to calculate the correct value for the MEMORY command.

There are two different methods of calculation, depending on which screen mode you are going to use.

MODE 0 CALCULATION

1) DIVIDE X DIMENSION BY 2 DISCARDING ANY FRACTION (INT)

2) ADD 1 TO RESULT

3) MULTIPLY ANSWER BY Y DIMENSION

4) MULTIPLY RESULT OF 3 BY 2

ie MEM = INT (X DIMENSION / 2 + 1) * Y * 2

You must do this for all the drawings you are going to |DGET.

When you have a total for all the drawings subtract answer from 23389 to find the result you should use for the MEMORY command.

MODE 1 CALCULATION

1) DIVIDE X DIMENSION BY 4 DISCARDING ANY FRACTION (INT)

2) ADD 1 TO RESULT

3) MULTIPLY ANSWER BY Y DIMENSION

4) MULTIPLY RESULT OF 3 BY 4

ie MEM = INT (X DIMENSION / 4 + 1) * Y * 4

You must do this for all the drawings you are going to |DGET.

When you have a total for all the drawings subtract answer from 23389 to find the result you should use for the MEMORY command.

You can test your calculations once you have DGETed all of your drawings into memory. Once all the drawings are stored type in the following line

```
A%=0:|SMEM,@A%:PRINT A%-1
```

The value that is printed onto the screen should be the roughly the same value used for the MEMORY command.

LOADING DRAWINGS FROM THE SPRITE DESIGNER PROGRAM

We strongly advise the user to design his/her drawings using the sprites designer program. This has a number of distinct advantages over manually getting the data from the top of the screen. If however the user wants to use sprites in mode 1 he/she will have to use the manual method. This method is described in detail on page 9 under the heading |DGET.

Users of the sprite designer program should follow the instructions detailed below to avoid any problems that may occur

1) Fully design your drawings and make sure they are ready to be used by your main program.

2) Before you select option (8) CREATE SPRITE DATA in the sprites designer program, find out how much memory the drawings take up. To do this press option (6) SPRITE DATA and then press Y. At the end of the list of drawings will be the total memory used. Write down this number.

3) Now select option (8) CREATE SPRITE DATA. Insert the disc you want the drawings stored to and enter a filename.

4) You are now ready to load the drawings into your program. Before you do this you must insert a MEMORY command. To calculate the memory command subtract the total memory used from 23389. The answer to the sum should be used for the MEMORY command.

5) You should now use the |DRAW command. This will load all of the drawings into memory. If you would like the colours to be altered to the colours used within the SPRITE DESIGN program use the |COLOUR command.

We will now describe the |DRAW and |COLOUR commands in more detail.

|DRAW,"filename"

This command can be used in either immediate mode or it can be included within your program. This command will load in the drawing information for the filename you supply. The drawing data stored on the disc has the file name xxxxxxxx.DRW. Do not include the DRW when you specify the name.

You should not use this command until you have used the MEMORY command as described above.

You may repeat the test shown on the previous page to ensure that you have used the correct value for the MEMORY command.

GENERAL SCREEN COMMANDS

The colours you use within your program are stored in memory. By doing this you can achieve professional results when putting the sprites onto the screen. If you turn all the inks to black, put the sprites onto the screen and then set the inks to their correct colour it will look as if the sprites have all been put onto the screen at the same time.

The memory block used to store the colours is 16 bytes long.

The memory block is updated when you load drawings into the computer using the DRAW command. If you load any drawings into the computer, the colours you selected using the designer program will be transferred into this memory block. You can access the colours by using the |COLOUR command as described below.

The only limitation of using this method is that you may not use flashing colours. If you try to use the Amstrad INK command with more than two variables (ink and colour) an error message will be displayed.

|COLOUR

This command will turn all the inks on the screen to their normal colours (as defined by the Amstrad INK i,c command). This command may be used in conjunction with either the |INKBLACK command or the |DRAW command.

When using this command in conjunction with the |DRAW command it will change the inks to the colours selected by the sprite designer program.

|INKBLACK

This command will change all the colours on the screen, including the border and paper, to black. The colours used will be remembered, you will not need to define all the colours again, you would simply use the |COLOUR command as described above.

|WP

This command will clear the screen. The screen will change to Mode 2, the screen will revert to its normal size and the pen and paper colours will be changed to make them easier to read.

|CLS

If you are going to use a different size screen to the normal Amstrad screen when you issue the normal CLS command a small section of the screen way not have been cleared correctly. To clear the screen completely you should use this command.

SPECIAL SPRITES AND HOW TO USE THEM

I have spent quite some time trying to decide where this page should appear in the manual. On first reading it may seem totally confusing, however, if I didn't include it near the beginning of the manual you may not understand why some sprites behave differently to others. My suggestion is that you should read this page now and then refer back to it when you understand the program a bit more.

Sprite 0 and sprite 1 are controlled by the joystick and keyboard, all other sprites (2 63) are controlled by your program.

If you do not intend to use a sprite controlled by joystick or keyboard you should not use sprite 0 or sprite 1.

The differences between these special sprites and the more general purpose sprites are as follows :

- Sprite 0 and sprite 1 will not bounce at screen edge.
- Sprite 0 and sprite 1 will not bounce when they collide with another sprite.
- Sprite 0 and sprite 1 will behave differently when they are in collision with other sprites.
- Sprite 0 and sprite 1 have separate missile commands to all other sprites.

It is very easy to forget when you are writing your programs about these differences (I've done it a number of times). If you find one of your sprites behaving differently check your program to ensure you are not using sprite 0 or sprite 1 for any purpose other than joystick/keyboard control.

Sprite 0 is the main character sprite. It will operate under control of the joystick.

Sprite 1 should only be used in two player games or in games which use the keyboard instead of the joystick.

SPRITES, DRAWINGS AND PUTTING THEM ONTO SCREEN

|SGET,s,d

This command will allow you to link a sprite to a previously defined drawing. s is the sprite number and d is the drawing number. By now you should have your drawings stored in memory. To make use of your drawings you must link it to a sprite.

s may be any non defined sprite in the range 0 to 63. d must be a defined drawing in the range 0 to 63.

Using the Breakout game as an example. You have defined 3 drawings. Drawing 0 is the bat, drawing 1 is the ball and drawing 2 is the block. We will make sprite 0 the bat, sprite 2 the ball and sprites 3-42 the blocks. To do this we would use the following commands.

```
|SGET,0,0:|SGET,2,1:FOR I%=3 TO 42:|SGET,I%,2:NEXT I%
```

We use the % symbol because this means we aye using integer variables, this will be explained in detail later.

We would use sprite 0 as the bat as this is controlled by the joystick. However if we wanted the bat to be controlled by the keyboard we would need to use sprite 1.

The line above links the sprites to the correct drawing. If we wanted to make the bat sprite 1 and the ball sprite 2, then we would have used

```
|SGET,1,0:|SGET,2,1:FOR . . .
```

IMPORTANT : You must set the relevant |SCREEN before using |SGET command. If you wish to to use a different screen size to the standard Amstrad screen you must change the size of the screen with the |SCREEN command before issuing an |SGET command, failure to do this will result in the sprites behaving incorrectly.

|SPUT,s,x,y

This command is used for putting the sprite onto the screen. s is the sprite number, x and y are the coordinates for the position of the sprite to go onto the screen. s must be a sprite that you have defined, x and y must be within the legal coordinates allowed by that screen. The coordinates refer to where the top left hand edge of the sprite will go.

Also bear in mind that if you are putting a sprite onto the screen near the bottom or right hand edge you must take into account the height and width of the sprite. ie If a sprite has a height of ten pixels then it would be no good trying to put it at coordinate 100,8 simply because the sprite would go off the bottom of the screen.

GETTING THE SPRITES ON AND OFF THE SCREEN

|SPUT,s

This command is used to take a sprite off the screen. s is the sprite number. If you have taken a sprite off the screen then by using another |SPUT,s command the sprite will come back onto the screen at the same place that it was taken off. In other words the |SPUT,s command will toggle the sprite off and on the screen.

Example.

```
|SPUT,0,100,100 - Put sprite 0 at location 100,100
|SPUT,0         - Take sprite 0 off the screen
|SPUT,0         - Put sprite 0 back onto the screen
```

NOTE In order to use the |SPUT,s command the sprite must have been put onto the screen initially with a |SPUT,s,x,y

|SPUT,s1,s2

This command is identical to the |SPUT,s command except that it will either remove or put onto the screen a number of sprites. s1 being the first sprite and s2 being the last sprite. It does not matter if you have not defined all of the sprites in the range s1 - s2 as the program will ignore any sprite that has not been defined.

|SPUTALL

This command will remove all the sprites off the screen. This is useful at the end of a game when you need the screen clearing.

IMPORTANT: If you have put a sprite onto the screen then you should not try and put that same sprite onto a different part of the screen without first removing the original sprite.

```
ie |SPUT,0,100,100:|SPUT,0,120,30         - This is incorrect
   |SPUT,0,100,100:|SPUT,0:|SPUT,0,120,30 - This Is correct
```

|RESET

This command will reset all of the settings within the program back to their default values. The data that links the sprites to the drawings will be erased. The drawing data will be kept intact.

|ERASE

This command will do everything the RESET command does, but it will also erase the drawing data as well. THIS COMMAND SHOULD BE THE FIRST COMMAND USED AT THE START OF ANY OF YOUR PROGRAMS.

SETTING THE ATTRIBUTES OF THE SPRITES

|XEDGE,s,n

This command will determine the property of the sprite when it collides with the left or right hand screen edge. s is the sprite number, n is the number that determines what the sprite will do when it hits the screen edge. The table below will show you how to set tip the correct attribute.

NUMBER	ATTRIBUTE
1	Disappear
2	Stop
3	Bounce
4	Wrap

Disappear : The sprite will be taken off the screen when it hits the screen edge.

Stop : The sprite will stop moving but stay on the screen.

Bounce : The sprite will bounce off the screen edge as if it had bit a wall.

Wrap : The sprite will be removed from the screen and put back onto the screen on opposite screen boundary.

|YEDGE,s,n

This command is identical to the |XEDGE command except that it determines what the sprite will do if it hits the top or bottom screen boundarys. It uses the same attribute number as the |XEDGE command.

The |XEDGE and |YEDGE commands can be set to different values. ie The |YEDGE could be set to bounce whilst the |YEDGE could be set to wrap.

NOTE : The default settings are for the general sprites is BOUNCE at all edges.

The default settings for sprite 0 and sprite 1 are for the sprites to STOP at all edges.

You may not alter the edge attributes for any sprites you define to be missiles. The missile sprites will be described in more detail later in the manual.

Sprite 0 and sprite 1 may only be set to STOP or WRAP. If you try to set them to bounce or disapear an error message will be displayed.

GETTING THE SPRITES MOVING

|SDIR,s,xs,ys

This command determines the speed at which the sprite will move. s is the sprite number, xs and ys are the sprite speeds. The speed variable determines how many pixels at a time the sprite will move. We do not recommend that this value goes above 8. This should be fast enough for most applications. To get the sprite to move to the left then you should use a negative velocity for xs. To get the sprite to move down the screen then you should use a negative velocity for ys.

NOTE : Under normal circumstances the speed of sprite 0 and sprite 1 will be set by the |STIXSPEED and |KEBSPEED commands.

You may not set the speed of the missiles using this command. The speed for missiles is set using the |MISSILE command.

You have a number of commands which will allow you to control the sprites either by the joystick, keyboard or both. They range from simply reporting the position of the joystick to actually moving the sprite under joystick or keyboard control.

The commands in this section deal with moving the sprites in the direction of either the joystick or keyboard.

As you will already know, sprite 0 and sprite 1 are controlled by using the joystick and keyboard. The commands below and on the next page will determine how sprite 0 and sprite 1 are controlled.

|STIX,n

The |STIX command allows sprite 0 to move under the control of the Joystick. The number after the command determines in which direction that sprite will be allowed to move. This allows for the sprite to be moved only left-right; or left,right, up, down but not diagonally; or left, right, up, down and diagonally. This number will also determine whether or not the fire button will be activated. The sprite will move under joystick control every time sprite 0 tries to move. i.e |MOVEALL or |MOVE,0,20 etc.

|KEB,n

This command is identical to |STIX command except that it will control sprite 1 and it will determine which way the sprite will move under control of the keyboard. The sprite will move under keyboard control every time sprite 1 tries to move. i.e |MOVEALL or |MOVE,0,20 etc.

USING THE JOYSTICK AND KEYBOARD

In order to inform the program of the directions that sprite 0 and sprite 1 are allowed to move you should use the following table.

NUMBER	DIRECTION	COMMENTS
0	NO DIRECTION	JOYSTICK/KEYBOARD DISABLED
1	UP	ALLOW SPRITE TO GO UP
2	DOWN	ALLOW SPRITE TO GO DOWN
4	LEFT	ALLOW SPRITE TO GO LEFT
8	RIGHT	ALLOW SPRITE TO GO RIGHT
16	DIAGONAL	ALLOW THE SPRITE TO MOVE DIAGONALLY
32	FIRE	FIRE BUTTON ON
64	CONTINUOUS	SPRITE WILL NEVER STOP

To use this table simply decide which features you want turned on and add up the numbers to get a value. This is the number you put after the |STIX or |KEB command.

Example.

Sprite 0 to move Left-Right with fire button turned on.

```
4 + 8 + 32 = 44                    |STIX,44
```

Sprite 1 to move Up, Down, Left and Right with no diagonal movement and no fire button.

```
1 + 2 + 4 + 8 = 15                 |KEB,15
```

if you wanted the fire button turned on as well as the directions then you would use

```
1 + 2 + 4 + 8 + 32 = 47            |KEB,47
```

If you wanted diagonal movement as well.

```
1 + 2 + 4 + 8 + 16 + 32 = 63       |KEB,63
```

Continuous movement means the sprite will not stop, even if you centre the joystick. The sprite will stop only when it hits a screen edge.

If you are not going to use the joystick you should not put sprite 0 onto the screen and you should not use the |STIX command.

It you are not going to use the keyboard you should not put sprite 1 onto the screen and you should not use the |KEB command.

NOTE : The default settings for both the joystick and the keyboard are for them both to be disabled.

USING THE JOYSTICK AND KEYBOARD

|STIXSPEED,u,d,l,r

The |STIXSPEED command informs the program of the speed that sprite 0 will move whilst under joystick control. All the variables must be used, ie use four values. This is the case even though the joystick may only be programmed to move in two directions. The values must all be positive numbers.

|KEBSPEED,u,d,l,r

This is identical to the STIXSPEED command except that it will control the speed of sprite 1.

Example.

|STIXSPEED,1,1,2,2	This will move sprite 0 at a speed of 1 pixel at a time whilst moving up and down and 2 pixels whilst moving left or right.
|KEBSPEED,4,4,2,2	This will allow sprite 1 to move at a speed of 2 pixels at a time whilst moving left or right, and 4 pixels at a time moving top and down.

NOTE : The default settings for both the |STIXSPEED and the |KEBSPEED commands are all directions set to speed 1.

|KEBDEF,u,d,l,r,f

This command will allow the user to define which keys will move sprite 1. The command accepts key numbers only. This information can be found in chapter 7 page 23 of the Amstrad 6128 user manual. The information is also displayed on the 6128 disc drive case. You most define all the key numbers even though you may not use them all. ie You may not want to have a fire button defined, however you must use a key number for it.

Example. |KEBDEF,58,63,60,61,18

The command on the previous page would define the keyboard as follows

Up - E, Down - X Left - S, Right - D, Fire - RETURN

Default setting: arrow keys for direction, space to fire.

USING THE JOYSTICK AND KEYBOARD

Due to the way in which the computer tests for certain keys it will sometimes return the wrong key number. This can effect the way the sprites move if you are using both sprite 0 and sprite 1. If you find the sprites are not moving in the direction you have selected or change direction of their own accord, you should try altering the key numbers you selected using the |KEBDEF command. This problem will also occur with the |READKEB and |READJOY commands.

All the above commands deal with actually moving the sprite under control of the joystick or keyboard. This type of control is fine if you want the sprite to move in the direction that you move the joystick (space invaders, pacman). However there are games and situations when you will need to know in which direction the joystick/keyboard is pointing without actually moving the sprite in that particular direction. Take as an example the game of Asteroids. If you move the joystick to the left the ship should rotate anticlockwise, instead of moving to the left.

The way to get around this problem is to establish the position of the joystick/keyboard and then act upon this information. The commands below will test if a certain direction is being pressed.

|READSTIX,k,@v%

This command will test the joystick to see if is being moved in a particular direction. k should contain a number between 0 and 9. This number will refer to the direction to test.

```
        1
    2       5
3       9       6
    4       7
        8
```

If you wish to test to see if the joystick is pushed straight up the value of k should be 1. To test for diagonally left and down k should have the value of 4, etc.

Number 9 tests for the fire button.

Number 0 tests for NO direction. If a value of 0 is returned in your variable it means the joystick is in the central position, if a value of 1 is returned it means that the joystick is being pushed in a certain but unknown direction.

The command will return a value in v%. This can either be 0 (not pressed) or 1 (pressed).

|READKEB,k,@v%

This command acts in exactly the same way as the |READJOY command except it tests for the direction pressed on the keyboard. The direction keys are defined by the |KEBDEF command.

GETTING THE SPRITE MOVING

|MOVE,s

This command will move sprite s. The sprite must be on the screen and either under joystick/keyboard control or have been given a direction to move in with the |SDIR command. If the sprite is not on the screen the command will be ignored.

|MOVE,s1,s2

This command will move all the sprites that are on the screen that are within the s1 - s2 limits. s1 is the lowest sprite, s2 is the highest sprite.

|MOVEALL

This command will move all the sprites on the screen.

Example.

```
10 MODE 0                        ; SET MODE TO 0
20 ?"0":|DGET,0,7,7:CLS          ; GET DRAWING 0
30 |SGET,2,0:|SGET,3,0           ; SET UP SPRITE 2 AND 3
40 |SDIR,2,-1,-1:|SDIR,3,1,-1    ; SET THE DIRECTIONS
50 |SPUT,2,40,30:|SPUT,3,60,70   ; PUT SPRITES ON SCREEN
60 |MOVEALL:GOTO 60              ; MOVE THE SPRITES AND LOOP
```

This program does not set the |XEDGE and |YEDGE commands as we will use the default values of bounce at screen edge. Try experimenting with this program by altering screen mode or sprite speeds. You could even try to alter the edge attributes by putting in a line at 45

ie 45 |XEDGE,2,4:|YEDGE,2,4

See if you can make sprite 2 act under control of the joystick in all four directions. As a hint you will need to alter the sprite number to 0.

You may notice that when the sprites collide they bounce off each other. This is covered on the next page.

|SWINDOW,s,xl,xh,yl,yh

This command will set up a window in which the sprite can move. Each sprite can have an independent window. When the |SGET command is used the window is automatically set to the full size of the screen. You may alter the size of the window by using this command. xl is the left hand edge, xh is the right hand edge, yl is the bottom edge and yh is the top edge. This command should be used after the SGET command.

The four values should be expressed in sprite coordinates.

COLLISION DETECTION

|COLLIDE,s,n

This command will determine what the sprite will do if it hits anything other than the screen edge. s is the sprite number, n is a value that will determine what the sprite will do when in collision.

The table below will show you how to set the correct attribute for sprite s when it is in collision.

NUMBER	ATTRIBUTE
1	Disappear
2	Stop
3	Bounce

Disappear : The sprite will be remove from the screen when it hits an object other than the screen edge.

Stop : The sprite will stop moving but stay on the screen.

Bounce : The sprite will bounce off the object that it has hit.

NOTE : The default settings for general purpose sprites is to bounce when in collision.

The default setting for sprite 0 and sprite 1 is to STOP when in collision.

You may not alter the collision attribute of any sprites defined as missiles. The missile sprites will he described in detail later in the manual.

Sprite 0 and sprite 1 can only STOP when in collision. You may not alter the collision attribute for these two sprites. If you do try to alter the attribute an error message will be displayed.

|REPON,s

In order for you to write your program you will need to be able to tell if certain sprites are in collision. There are a number of commands to help you achieve this. You may not need details about every collision on the screen, and this is why we have included the |REPON and |REPOFF commands.

The |REPON command will turn on the collision reporting for sprite s. This collision reporting has no say in what the sprites will do when they collide. If the reporting is switched on for a particular sprite and has its attributes set to bounce and the sprite collides with something the sprite will bounce and its sprite number will be remembered.

COLLISION DETECTION

If the same sprite has its reporting switched off and it was in collision, it would still bounce but the sprite number would not be remembered.

It is best if you keep the number of sprites with reporting turned on to a minimum. Some of the collision detection commands are relatively slow, therefore the less sprites that they have to test the faster your program will be.

NOTE : The default setting for collision reporting for all sprites is OFF. In order to use the collision detection commands the collision reporting must be turned on.

You cannot turn on the collision detection for missile sprites. These sprites have their own detection routines. We will deal with the missile commands further into the manual.

|REPOFF,s

This command is the opposite to the |REPON command. it will turn the collision detection for sprite s off.

The following commands all test the collision reporting flag. it the flag is turned off and a sprite is in collision the sprite number will not be reported.

|COLLTEST,s,@v%

This command will tell the user whether a particular sprite is in collision. s is the sprite number, v% is the variable that you want the information stored in. Please be aware that you must use the @ just before the variable. This is because the sprite program is passing a variable back to basic.

After using this command the variable you have used will contain a number. This number will be either 0, 64, 128 or 255.

```
  0 - Sprite in collision
 64 - Collision detection turned off
128 - Sprite not on screen
255 - Sprite not collided
```

SUGGESTION : When you are putting sprites onto the screen randomly you can test to ensure a sprite has not been put onto another sprite.

```
10 A%=0:FOR I%=2 TO 30:|SGET,I%,3: |REPON,I%
20 |SPUT,I%,RND(8)*140,RND(8)*160+30
30 |COLLTEST,I%,@A%:IF A%=0 THEN |SPUT,I%:GOTO 20
40 NEXT
```

If a sprite is in collision, it is removed and the program tries again.

COLLISION DETECTION

|REPORT,@v%

This command is used to check all sprites that are on the screen to see if they are in collision. By issuing this command, every sprite on the screen that has collision reporting turned on is checked for collision. All the sprites that are in collision are stored in a table. The variable v% tells you the amount of sprites that were in collision when the command was issued. Therefore if v% returns a value of 0, then none of the sprites with collision reporting turned on were in collision.

By using this command together with the |NEXTREP command the user will have a list of the sprites that were in collision.

|REPORT,s1,s2,@v%

This command is identical to the |REPORT,@v% command except that it will check a range of sprites instead of all the sprites. s1 is the starting sprite number whilst s2 is the ending sprite number.

|NEXTREP,@v%

This command will get the next value from the table that was set up by the |REPORT command. When using this command the variable v% will contain the number of a sprite that was in collision. When you have this number, that sprite number will bc removed from the table. By doing another |NEXTREP,@v% command the next sprite number will be taken from the table. If the variable v% contains the value 255 then that means all the sprite numbers in the table have been reported.

An important point to remember is that only the sprites that were in collision at the time of the |REPORT command are stored in the table. If you move any sprites after the |REPORT command and some of the new movement causes a collision that will not be reported.

You can issue another |REPORT command before you have removed all of the sprite numbers from the table. The old sprite numbers that were in the table will be erased to make way for the new numbers.

An important point to remember is that this routine is quite slow. You should not call it needlessly.

|CLEAREP

This command will clear the report table. If a |NEXTREP command is issued after this command the variable will contain 255. ie no more sprites in collision.

COLLISION DETECTION

|HIT,s,@v%

This command will allow the user to find out which sprite has hit another sprite. ie You may know sprite 4 is in collision but there is no way to find out which sprite that it has collided with. By using this command you will be able to find out.

s is the sprite number that you know is in collision, v% is the variable that will hold the number of the sprite that has hit sprite a. If v% contains 255 then the sprite must have hit a piece of scenery or a sprite that does not have collision reporting turned on.

|MOVEHIT,s,@b%

This command can be used after you have just moved a sprite. s is the sprite number, b% is the variable that has the result of the collision test in it. If b%=255 then the sprite was not in collision, if b%=0 the sprite has collided with something.

The advantage of using this command in preference to |COLLTEST command is that this command is a great deal faster. During the sprite movement routine the sprite is tested for collision automatically. This routine simply checks the flag within the movement routine to see if the collision flag has been set.

It you move the sprite again then the collision flag will be cleared. It is important that you fully understand how this command works in order to use it to its full advantage.

Example.

You move sprites 0 to 10 with the |MOVE,0,10 command. When sprite 0 is moved it does not hit anything therefore the collision flag is not set. When sprite 1 is moved it hits sprite 0 therefore the collision flag is set for sprite 1. If you did a |MOVEHIT,0,@b% command, testing to see if sprite 0 is in collision, the routine would return a value of 255 (sprite not in collision). This is because when sprite 0 was moved it did not hit anything. It you did a |MOVEHIT,1,@b% command, testing to see if sprite 1 was in collision, the routine would return a value of 0 (sprite in collision).

If instead of doing the |MOVEHIT command after the |MOVE command you did a |COLLTEST,0,@b% command then the routine would return a value of 0 (sprite in collision). This is because the |COLLTEST command checks the sprite after everything has moved.

Please note : The collision reporting must be turned on In order for this routine to work.

IMPORTANT NOTES

Throughout the collision detection commands we have been using a variable v%. There is a '@' character before this command. We are using the '@' character because that tells BASIC that we are going to pass a value from the sprites program back to your own program. It is extremely important to include this character before your variable name. Failure to do this could result in your program being destroyed.

We have used v% as a variable name throughout the collision detection commands. You may use any variable name.

We have used the '%' character after the variable name. This is because the sprite program is passing integer variables. You must always use integer variables whilst either passing or receiving variables from the sprite program. Therefore it is important that you understand fully how to use integer variables.

An important point to remember about passing integer variables back to BASIC is that the variable must be known to the program. At the start of the program the variables must contain a number, this can be zero. ie a%=0:v%=0:f%=10

It is always a good idea to save your program before you 'RUN' it. Whilst every effort has been made to ensure that any errors that you make are reported by the error routine there are a few instances were an error might be missed by the program.

ERROR MESSAGES

If the program finds an error, then the command the error was found in will be displayed along with the error number. The program will then pause and the cursor will appear on the screen. It is now up to you to break into the program by pressing the 'BREAK' key once. If you press any key other than the BREAK key the program will continue.

If you have missed pressing the BREAK key and the program has continued you should stop the program by pressing the BREAK key twice. You should now sort out the error.

The error numbers can be found in appendix II.

After you have pressed the BREAK key the computer will report BREAK IN LINE xxxx. This may not be the line in which the error occurred. The error may have occurred in the previous line. If an error occurs in the last statement of a line when the BREAK key is pressed the computer will display the line number as the line following the erroneous line.

ie Your program starts at line 10 and ends at line 250 going up in steps of 10. If the computer displays BREAK IN LINE 60 the error could be in line 60, or it could be the last command on the previous line. ie 50.

MISCELLANEOUS COMMANDS

|SXPOS,s,@x%

This command will tell the user the X coordinate of a particular sprite. s is the number of the sprite, @x% is the variable that you want the sprite X coordinate to be stored in.

|SYPOS,s,@y%

This command will tell the user the Y coordinate of a particular sprite. s is the number of the sprite. @y% is the variable that you want the sprite Y coordinate to be stored in.

|SDRAW,s,@d%

This command will tell the user the drawing number that the sprite currently is. s is the number of the sprite. @d% is the variable that you want the drawing number to be stored in.

|SMEM,@m%

This command will tell the user the amount of memory that the drawings have taken up in memory. This value. should always he higher than the HIMEM value. @m% is the variable that you want the memory information stored in.

|XDIR,s,@x%

This command will tell the user the speed at which the sprite is moving on the horizontal plane. s is the number of the sprite, @x% is the variable that you want the sprite speed to be stored in.

|YDIR,s,@y%

This command will tell the user the speed at which the sprite is moving on the vertical plane. @y% is the number of the sprite. @y% is the variable that you want the sprite speed to be stored In.

|CSPRITE,s

This command will clear sprite s from memory. If the sprite is on the screen when this command is issued, the sprite will be removed. You may now, if you wish use the |SGET command to link this sprite to another drawing.

|WAIT,n

This command will cause your program to pause. n is the amount of time in 1/20th's of a second it will wait. If n was 100 the program would wait for 5 seconds.

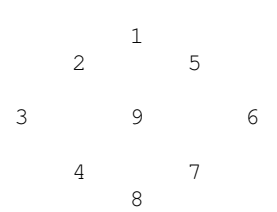
ANIMATING THE DRAWINGS

The user can define 16 animation sequences. The animation sequences consist of a list of drawing numbers in sequence for the eight directions that the sprite can move in.

The drawings to be used in animation must all have the game x and y dimensions. They must also be in sequence one after another. You define the drawing sequences for the sprite for the eight directions in which the sprite can move. You may not want to use all directions, indeed you may only want the sprite to he animated in one or two directions, however you must define the sprite movement for all eight directions.

The sprites program will automatically select the correct drawing sequence depending on which direction the sprite is moving, once the drawing sequence has been defined.

A number of sprites may use the same sequence of drawings, therefore you may have a great deal more than 16 sprites animated on the screen at the same time.



The way you define your sequence is as follows

These are the eight directions that your sprite can move in.

You must define a drawing sequence for each direction.

|SEQUENCE,sn,1l,1h,2l,2h,3l,3h,4l,4h,5l,5h,6l,6h,7l,7h,8l,8h

This is quite a long command, you only need do it once for each sequence and not everytime you want to animate a sprite. The l and h in the command example are for the low and high numbers of a sequence.

sn is the sequence number. This should not he confused with the sprite number. You will learn how to link a sequence to a sprite shortly. The sequence number can be between 0 and 15.

When you define a sequence from low to high, every drawing within that sequence must have the same x,y dimensions. Also there should not be any drawings missing. The sequences can be repeated for a number of different directions.

Example : You have designed a number of drawings of a man, drawings 0 to 5 are. the man walking left, drawings 6 to 11 are the man walking right. The man can walk left and right. The command should look something like this

|sequence,0,0,5,0,5,0,5,0,5,6,11,6,11,6,11,6,11

This command sets up sequence 0. If the man happened to start moving at a diagonal the figure would still be animated.

ANIMATING THE DRAWINGS

Once you have set up your animation sequences you will then need to link the sequences to a sprite. You should do this after you have issued an |SGET command. When you do link a sprite to a drawing, the drawing number you should use should be lower than the highest drawing number defined in the sequence.

Ie. 'You have linked drawing 10 to sprite number 0. The sequence you are going to use must have a drawing number in it which is greater than 10.

Once the sprite is animated the drawing number is incremented until it is equal to the last sequence number, then it is reset to the start sequence number. If you issued an |SGET command with a drawing number higher than the highest sequence number, an error would occur.

If you wish to use only one drawing for each different direction, you should use the same drawing number for both the high and the low values.

Example.

You have designed eight drawings of a spaceship, one drawing for each direction the ship can move in. The sequence command should look something like this.

|SEQUENCE,1,0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7

|ANIMATE,s,sn

This is a command that links a sprite to an animation sequence. s is the sprite number, sn is the sequence number. The sprite should have already been linked to a drawing with the |SGET command.

|ANIMOFF,s

This command turns the animation of sprite s off. The drawing that the sprite was using at the time the command was issued will now be frozen. ie not animated. If the sprite is now moved the sprite will be linked to the last drawing.

NOTES ON ANIMATION

If a sprite hits another sprite whilst it is animated then it is possible for the sprites to stick together. This is due to the nature of the animation and the collision detection. Under normal circumstances as soon as a sprite collides with another by one pixel the sprite will bounce. However if at the time of collision the drawing overlaps the other drawing by more than one pixel due to the change in the shape of the sprite through animation, a stick may occur. There are a number of things you can do to get around this (if it occurs). There are some notes towards the back of the manual to help you.

USING MISSILES

There have been a number of commands incorporated within the supervisor program to allow the sprites to fire missiles. The commands may seem complicated at first, but with experimentation the commands will soon become easy to use.

You must first design a missile drawing. You do this in the same way as you would define any drawing, be it a small circle or a more rectangular shaped missile.

The missile must then be linked to every DRAWING you wish to fire missiles. The reason you link a missile to a drawing rather than to a sprite is simple.

It you have defined a sprite to be a spaceship (as in the animation example), the sprite can move in eight directions. Each of these eight directions has a different drawing associated with it. Because we have linked the missile to the drawing rather than the sprite we know which direction to fire the missile in, we also know the offset at which to first put the missile onto the screen.

You are probably totally confused by that description. We will now go through the commands associated with the missiles and we will give you some examples after that. There is a demonstration program on the disc called DEMO1 that you can look through to give you a better idea of the commands.

There are three types of missiles type 0, 1 and 2. Type 0 missiles are fired from sprite 0, type 1 missiles are fired form sprite 1 and type 2 missiles are fired from sprites 2 - 63.

|MISSTYPE,ty,d

You should have by now defined your drawing(s) for your missile(s). You may have a separate drawing for each type of missile or all types may use the same drawing. The above command will inform the supervisor of which drawing to use for the missile for each missile type.

ie |MISSTYPE,0,2:|MISSTYPE,2,2

This would define missile type 0 to drawing 2 and missile type 2 to drawing 2. Remember type 0 missiles are fired from sprite 0 and type 2 missiles are fired from sprites 2 - 63. Both are using drawing 2 as the missile drawing.

|MISSTYPE,0,3:|MISSTYPE,1,4

This would define missile type 0 to drawing 3 and type 1 to drawing 4.

The drawings you design as missile drawings must be above drawing 1.

USING MISSILES

|MISSILE,Dn,ty,Xo,Yo,Xs,Ys

This is the command that will link a missile type to a drawing number. Dn is the drawing that will fire the missile. ty is the missile type (0,1,2).

If you require sprite 0 to fire missiles and sprite 0 is animated with drawings 0 to 7, we would need to use 8 |MISSILE commands starting |MISSILE,0,0,xo,yo,xs,ys and ending |MISSILE,7,0,xo,yo,xs,ys. Remember the second variable is the missile type and not the drawing number of the missile.

The advantage of linking a missile to a drawing rather than the sprite for use in different types of game is obvious. The game of asteroids has a ship that can rotate through eight directions. When you fire a missile, you fire it in the direction the ship is pointing and moving. However the game of space invaders has a ship moving to the left and right but the missiles always fire upwards.

The |MISSTYPE command must be issued before you start to use this command.

Xo and Yo are the X and Y offsets for the missile. To calculate the offsets you should use a piece of graph paper or a calculator.

CALCULATING MISSILE OFFSETS

There is a utility on the disc to help you to calculate the offsets. This utility is called "OFFSETS". Full instructions are included within the program.

To simplify things we shall call the drawing that is going to fire the missile drawing 'A', and the missile itself drawing 'B'.

As you should know by now, all the sprites are put onto the screen with the origin being the top left hand corner.

Using either graph paper or by calculating yourself you should make a picture of drawing A. You should then picture where you want drawing B to be. You should bear in mind that drawing A should not be touching drawing B. You should then calculate the distance between the top left hand corner of drawing A to the top left hand corner of drawing B. This is the offset, you should do this for both X and Y.

The offsets may be either positive or negative.

Xs and Ys are the speeds at which the missile will travel. You should not try to set the speeds of a missile with the |SDIR command.

USING MISSILES

The x speed and y speed of the missile need to he signed numbers. If you want the missile to travel left along the x axis you should use a negative number for the x speed. If you want the missile to travel down the screen you should use a negative number for the y speed.

By now you should have defined the drawings to be used as missiles and also which drawings will actually fire missiles. You should now have decided how far and how often a sprite can fire missiles. The next two commands will determine these factors.

|MISSDELAY,ty,de

This command will limit how fast the missiles can be fired. ty is the missile type, as described on the previous page (1,2). de is the delay between firing. de will contain a number, this number will determine how many moves a missile must move before the next missile can be put on the screen.

ie. If you set de to a value of 4, the missile that has most recently been fired must move 4 times before the next missile can be put on the screen.

This command has a number of advantages. It enables the previously fired missile to move out of the way before the next missile is fired, this will stop missiles from colliding with each other. The other advantage is that you can alter how frequently a sprite can fire missiles throughout the game.

The minimum value for de is 2 and the maximum value is 63.

|MISSDIST,di

This command will determine how far all of the missiles will travel before they are removed from the screen. di should contain the number of moves a missile can make before it is removed. ie. If the value of di was 5, then after the fifth MOVE statement for that sprite, the sprite would be removed. If you do not want the sprite to be removed then simply use a value of 127.

The minimum value for di is 5 and the maximum value is 127.

The commands concerning missiles that have appeared so far in this manual are in the same order as they should appear in your program. If you fail to put them in this order the missiles may not work correctly. The order they should appear in your program is

|MISSTYPE, |MISSILE, |MISSDELAY, |MISSDIST, |BULLET

USING MISSILES

We are now ready to define a number of sprites to be used as missiles.

|BULLET,ty,sp1,sp2

This command defines how many sprites are going to be used as missiles. ty is the missile type. There are three separate banks of missiles you may use, 0,1 and 2.

Sprite 0 has its own bank (0). Sprite 1 has its own bank (1). Sprites 2-63 have their own bank (2).

sp1 is the lowest sprite that can be used as a missile, sp2 is the highest sprite that can be used as a missile.

A sprite can have more than one missile on the screen at the same time.

EXAMPLE.

You wish for sprite 0 to have up to 10 missiles on the screen at the same time. You way then simply use the command

|BULLET,0,20,29

This command will define sprites 20 to 29 to be missile sprites.

To get the sprite to actually fire a missile is simple. For sprites 0 and 1 you may simply turn on the fire button using the |STIX or |KEB commands. Everytime the button is pressed a missile will be fired.

The other way to fire a missile, this works with all sprites from 0 to 63, is to use the |SHOOT command which will be described a little later.

NOTE : When defining the lower and upper sprite numbers to be used as missiles you should not use these sprites for any other purpose. Also the lowest sprite number cannot be lower than 2.

The three banks of missiles should not overlap.

When your program is working and you start to fire the missiles you may get an error 'MISSILE TYPE MISMATCH'. This occurs when you try to fire a missile of the wrong type.

eg

You have defined drawings 0 to 4 to fire missiles of type 0 with the |MISSILE command. If you issue a |SHOOT,2 command and sprite 2 is using drawing 1 an error will occur because |SHOOT,2 is trying to fire a type 2 missile and drawing 1 is only allowed to fire type 0 missiles. BE CAREFUL !!

USING MISSILES

|SHOOT,sp

This command will allow a sprite to fire a missile whenever this command is issued, sp being the sprite that will fire the missile.

If you have defined only a small number of sprites to be used as missiles and you have only used a small number for the delay you will find that the sprite will fire missiles in bursts. This is due to the fact that only the number of missiles you have defined can be on screen at the same time. If you continue to press the fire button or use the |SHOOT command the action will be ignored until a missile has been removed from the screen and so freeing this missile to be used again.

When the missiles collide with the screen edge they disappear and can be used again. If a missile collides with a piece of scenery, or a sprite that does not have collision reporting turned on, the missile will also disappear and can be used again.

If the missile hits a sprite with collision reporting turned on (|REPON,s) the missile will stop moving and it will stay on the screen. The sprite it has collided with will also stop moving (if it was moving).

We now have two commands that deal with missiles once they have collided with a sprite, that had collision reporting turned on.

|MISSHIT,@v%

This command store in variable v% the number of missiles that are currently in collision and are still on the screen. It does not take into account the type of missile. If v% contains 0, no missiles that are currently on the screen are in collision. If v% contained the value 3 then 3 missiles on the screen are in collision with a sprite.

|MISSHIT,ty,@v%

ty is the type of missile (0,1,2). When this command is issued, if a missile of the type you specified in ty was in collision, the missile will be removed from the screen and v% will contain the number of the sprite that the missile had collided with. The sprite that the missile had hit will now be able to move again. It is up to you to decide what to do with the sprite that the missile hit. You have the sprite number in v%, therefore if you wished to remove that sprite you could use the command |SPUT,v%.

USING MISSILES

It is important that you check regularly to ensure if any missiles have collided with sprites. If you do not your game may become cluttered with sprites that are not moving. Also if you have only specified to use only a small number of bullets with the |BULLET command you may not use the missile sprite again until it is removed from the screen.

NOTES ON USING THE MISSILES

If you should find that when firing your missiles they do not work correctly, there may be a number of reasons for this.

1) The offset you have chosen is to close to the main drawing. When the missile is put onto the screen it collides with the sprite that is firing it. You should increase the offset between the sprite and the missile.

2) You are not moving the missile fast enough. If the missile has not been moved completely away from the main firing sprite and you press the fire button again, the next missile will collide with the old missile. You should either increase the delay between firing missiles with the |MISSDEL command, or include more |MOVE routines to move the missile faster, or increase the speed the missile moves from within the |MISSILE command.

3) You are not checking for the missiles colliding often enough. As soon as you find a missile in collision you should act upon it. You should check for missile collisions REGULARLY. Remember the missiles do not remove themselves when they hit something, its up to YOU to remove the missile with the |MISSHIT command.

4) If you find the missile does not fire correctly when you are moving the sprite in the direction you are firing, you should increase the offset distance between the main firing drawing and the missile.

|MERGE,s,d

This command may be used once a sprite is on the screen. It will swap the drawing that is currently linked to the sprite with another drawing. s is the sprite that should be on the screen, d is the new drawing number to use. The old drawing and the new drawing should have the same X and Y dimensions.

This command could be used if you do not wish to use the movements that the |STIX command allows. The |STIX command is fine if you want to program a game like Space Invaders, if you move the joystick to the left the sprite will move to the left. If however you wish to program a game like Asteroids, where moving the stick to the left will rotate the spaceship without actually moving it, then this command should be used in your program in conjunction with the |READSTIX command.

You should use sprite 0 in this instance, even though the sprite is not directly controlled by the joystick.

MORE MISCELLANEOUS COMMANDS

|SCENERY,dn,x,y

This command allows the user to put drawings onto the screen to act as scenery. dn is the drawing number, the x and y numbers are the coordinates where you are going to place the scenery. You may use this command to put borders around games or to put scenery onto the screen. The position a drawing is put onto the screen is not remembered and so to remove it you must clear the screen. You may use a drawing any number of times. You may use a drawing that is linked to a sprite, this does not matter. This command has nothing at all to do with the actual sprites.

The coordinate system used for the scenery is the game system as used by the sprites. REMEMBER, you cannot go over the edge of a screen.

|EXPLODE,spr,spot,inc,lim,del

This command will make a sprite explode on the screen. Whilst the explosion is taking place everything on the screen will stop moving until the explosion dies down.

spr is the sprite that is to explode, it must be on the screen. spot is the number of pixels that are put on the screen before the explosion expands. inc is the size of the increase of the explosion in pixels. lim is the maximum size of the explosion in pixels. del is a delay to slow the explosion down.

The only way of getting used to this command is to try it with different values to see the effect it has on the screen.

```
spots : range 0 - 32767 , average 100 - 400
inc   : range 1 -32     , average 2 -- 10
lim   : range.10 - 128  , average 20 - 100
del   : range 1 - 255   , average 10 - 100
```

A program to test the explode command may look like this

```
 10 MODE 0:LOCATE 1,1:PRINT "0":|DGET,0,7,7
 20 |SGET,2,0:A%=0:B%=0:C%=0:D%=0
 30 CLS
 40 INPUT "SPOTS ",A%
 50 INPUT "INC   ",B%
 60 INPUT "LIM   ",C%
 70 INPUT "DEL   ",D%
 80 WHILE INKEY$=""
 90 |SPUT,2,75,95:|EXPLODE,2,A%,B%,C%,D%
100 WEND:|SPUTALL:GOTO 30
```

MORE MISCELLANEOUS COMMANDS

|FRAME

This command will wait until the frame flyback interrupt occurs before allowing the propram to continue. It is useful for slowing down graphics if they are moving too fast. It can also make the graphics in certain circumstances appear to move more smoothly, although I find it doesn't really make all that much difference.

|STUCK,s,@v%

This command will tell the user if a particular sprite has become stuck to another sprite or a piece of scenery. This can occur during animation, and is described in the animation section. If a sprite has become stuck you should |SPUT it off the screen, alter the coordinates and |SPUT it back onto the screen. s is the sprite number to test. v% will then contain a number 0 (not stuck) or 1 (stuck).

NOTE: This routine is not foolproof. It will only return a value of stuck if it cannot move in a certain direction. If you are using a joystick to control a sprite and you try to move the sprite into some scenery the stuck flag may be turned on for that sprite even though it can still move in other directions.

A none joystick/keyboard controlled sprite will try to free itself by moving in all directions. Only when it cannot move in any direction will the stuck flag be set. A sprite may look stuck to you, however it may be bouncing backwards and forwards a matter of pixels and therefore the stuck flag will not be set (the sprite is moving).

|SPEEK,x,y,@v%

This command will test a pixel on the screen. The x and y are the screen coordinates to test. v% will contain the ink number of that particular pixel. This command will only work in Mode 0 or Mode 1.

|SPOKE,x,y,i

This command will plot a pixel onto the screen. The x and y are the screen coordinates, i is the ink. You should use the sprites coordinate system (Not Amstrad).

USING AUTOMATIC SOUNDS

The supervisor allows for the sprites to automatically generate sounds when certain conditions apply. Whenever this condition is met the supervisor will generate a sound. There are two commands that deal with automatically generating sounds.

|SOUND,nn,ch,ae,te,tp,np,ia,du

This command is identical to the standard Amstrad sound command, with two exceptions. The first variable (sn) is the sound number. You can define up to 16 different sound effects (0 - 15). The only other difference between this command and the Amstrad version, is that you must use all the variables.

sn - Sound number (0 - 15)
ch - Channels to use and rendezvous requirements (0 - 255)
ae - Amplitude envelope to use (0 - 15)
te - Tone envelope to use (0 - 15)
tp - Tone period (1 - 4095)
np - Noise period (0 - 31)
ia - Initial amplitude (0 - 15)
du - Duration of sound (-32768 - +32767)

Please see your Amstrad user manual for a full description of this command.

As stated above you must use all 8 variables. If you do not require a certain function, you should use a value of zero.

To define the amplitude and tone envelopes, you should use the standard Amstrad commands ENV and ENT.

Once you have defined the sounds with the above commands you will need to link them to the sprites and their actions.

|SATTR,sp,ac,sn

You can get the sprites to issue a sound whenever one of the following situations occur

1 Fire a missile
2 Explode
3 Bounce at window edge
4 Bounce with anything else

sp is the number of an already defined sprite, ac is the action (1-4 above) when the sprite will issue a sound. sn is the number of a sound you have created with the |SOUND command.

eg |SATTR,0,1,2 : Issue sound 2 when sprite 0 fires a missile.

USING NODES

In certain games (Pacman etc) the sprites are only allowed to move within the confines of a maze. There are a number of commands built into the supervisor that will allow the user to simulate this type of game.

To create a maze for the sprites, we have to define a number of nodes. A node is the point within a maze which will allow the sprites to change direction.

To demonstrate this, consider a large square on the screen. The square has four corners, therefore there will need to he four nodes. (top left, top right, bottom left and bottom right).

To inform the supervisor of the shape of your maze we need to specify three variables. The first two variables are the x and y coordinates of the position that the sprites are allowed to change direction. The third variable is a value which informs the sprites the directions they are allowed to leave from that particular node.

To make things a little easier for you, we have written a program that will help you to define your nodes. The program has its own instructions. You will find the program on side one of your system disc. In order to use this program to its full advantage I will now try and explain nodes in a little more detail.

A node is an invisible point on the screen. You cannot see a node, however you can see the actions of a node. The demonstration program DEMO5, shows you the nodes in action. In this demonstration the sprites move around the screen and seem to change direction at fixed, but invisible, points. These points are nodes. To make the game playable you will need to show the paths the sprites can take. If you now look at DEMO6 we have included these paths. The paths are put there to make the game playable. Imagine trying to play Pacman without being able to see where you can change direction. It is up to you to make these paths visible. It is a very tedious job, but the results are worth it.

As stated above you will need to specify the x and y coordinates for every node. You will also need to specify the direction a sprite is allowed to leave the node. Whilst the node designer program will be of great use to you, we suggest you sit down with a piece of graph paper and VERY carefully plan your maze. You will need to take into account the width and height of the sprites that will be used within the maze.

ie If you have a sprite with a width of 12 pixels and you are designing a maze in mode 0 with the standard size Amstrad screen and the sprites are set to bounce at screen edge it would be no use putting a node at point 154,40 simply because the sprite would never reach that point. It would have bounced when it reached 147,40.

USING NODES

You will also need to take into account the height and width of the sprites when you are designing the paths the sprites will take. If the path is too narrow or low the sprite will collide with the path and get itself stuck.

The sprites change direction whenever the top left hand corner of the sprite has the same coordinates as the node. If you have put a node at 120,40 and you have put the sprite onto the screen at 111,40 moving right with a speed of two the sprite will bypass the node moving from 119,40 to 121,40. You will need to be careful to ensure that the speed of the sprite does not allow the sprite to pass over the node. This can be used to your advantage, allowing certain sprites to ignore specific nodes whilst other sprites do not.

When you are using the node designer program and you come to view your nodes, the sprites will only change direction when they hit a node. When the paths the sprites are allowed to take are drawn onto the screen and the paths cross each other and there is no node at that point where they cross, the sprites will not change direction. When all the paths are drawn the directions the sprites can leave the nodes are shown by a light coloured line leaving the node.

The nodes (when designed) use up a certain amount of memory. The amount of memory they use is displayed after you have saved the nodes (for use with the supervisor) to disc. You will need to make a note of this number. When you are calculating the value for your memory command, once you have an answer you should then subtract from the answer the amount of memory the nodes will use. You should then use the new figure for your memory command.

We will now go through the supervisor commands associated with the nodes.

|NODE,"filename"

This command will load the nodes you have defined with the node designer program into memory. You should have already taken into account the extra memory the nodes will use and altered the MEMORY command.

You can only load the nodes into memory AFTER ALL the drawings have been stored in memory. If after executing this command you try to load drawings or DGET drawings into memory an error message will he displayed.

The |RESET command will not clear the nodes from memory. The |ERASE command will clear the nodes as well as the drawings.

When you specify the filename you should not use an extension. ie GAME1.nde. The extension will be added automatically. This command will load files with the extension .NDE .

USING NODES

|NODESPEED,sp,xs,ys

This command sets the speeds the sprites will travel around the maze, sp is the number of an already defined sprite. xs and ys are the sprite speeds. The speeds you specify must both be positive numbers between 1 and 10.

The node speeds will not be used until a sprite hits its first node. Therefore if you are not going to initially SPUT a sprite directly onto a node you will also need to define the speed of the sprite with the |SDIR command. Once a sprite has hit a node the values used in the |SDIR command will be overwritten.

You do not need to specify the speeds for sprite 0 and sprite 1. The speeds for these two sprites is taken from the |STIXSPEED and |KEBSPEED commands.

Please remember the point made earlier about the speeds of the sprites, make sure the speeds defined will not allow the sprites to skip over nodes (unless you specifically want them to).

|NODEATTR,sp,at

This command will determine how the sprites you have defined will move around the maze. Sprite 0 and sprite 1 move around the maze under control of the joystick/keyboard. However the rest of the sprites (2 - 63) need to be told how to move. This command does just that. sp is the number of an already defined sprite (2 - 63), at is the attribute number that determines how the sprite will move.

The normal sprites (2 - 63) can either move randomly around the maze, chase after sprite 0/1, or flee away from sprite 0/1.

To determine which of these actions the sprite will take, please use the following table.

1 - Random move
2 - Chase after sprite 0/1
3 - Flee from sprite 0/1

You should use one of the numbers above for the at variable.

NOTE : If you are going to use option 2 or option 3 the nodes can chase/flee from either sprite 0 or sprite 1, they cannot chase/flee from both sprites. The sprite that they are going to chase or flee from is defined with the command on the next page. If the sprite that they are going to chase/flee is not on the screen then the sprites will assume option 1 until sprite 0 or sprite 1 is put back onto the screen.

USING NODES

|NODESPRITE,sp

This command will determine which sprite all other sprites will chase/flee. sp is the sprite number 0 or 1. If sp is 0 the sprites will chase/flee sprite 0, which is controlled by the joystick. If sp is 1 the sprites will chase/flee sprite 1, which is controlled by the keyboard.

|NODEON,sp

This command will turn on the maze function for a specific sprite. All of the sprites do not need to follow the directions the nodes dictate. Some of the sprites may ignore the nodes totally. Sprites that have been defined as missiles will also ignore the nodes. If you try to use this command on a missile sprite an error message will be displayed.

Before using this command you should have defined the NODESPEED and the NODEATTR for the sprite.

sp is the number of the sprite that will travel the maze. You can not use this command on sprite 0 and sprite 1 as they are automatically turned on when a maze is defined.

|NODEOFF,sp

This command will turn the maze feature off for sprite sp. sp can be any sprite number between 2 and 63.

|NODEALTER,x,y,nd

This command will alter the directions a sprite is allowed to leave a node. You cannot add extra nodes with this command, only alter existing nodes.

x and y are the coordinates of an already defined node. nd is the variable that will hold the new directions that a sprite can move on leaving the node. To calculate the new directions use the following table.

1 - Up
2 - Down
4 - Left
8 - Right

Decide which directions the sprites can now leave the node and add up the values to the left of the direction. The total should be the value used for the nd variable. You can not use a value of 0.

eg New direction left,right,up : 1 + 4 + 8 Therefore nd = 13

|MAZEON

This command will turn on the maze for all the sprites. After you have set up the nodes and defined all their attributes and speeds and turned them on with the |NODEON command, they will not follow the maze until this command has been issued.

|MAZEOFF

This command will turn the maze off. If you wish to turn the maze back on again you will not need to turn all the sprites back on individually, simply use this command. All the sprites that were turned on when this command is issued will be remembered.

These two commands are the master switches for file maze facility.

MORE NOTES ON SPRITES AND NODES

Sprites can not move diagonally through the maze. If a sprite has been defined to use the maze but has not hit its first node it can move diagonally. Even though a sprite has been turned on with the |NODEON command it will not follow the rules of the maze until it hits its first node.

In order for the sprites to use the maze straight away, they need to he SPUTed onto a node or they should he SPUTed onto the screen with a direction set with the SDIR command so that they hit a node.

It sprite 0 or sprite 1 is SPUT onto the screen and they are not on a node they can move around the screen freely until they do hit a node.

Whenever a sprite enters a node, the sprite will not leave by the direction it has entered unless there is only one exit from the node. ie the sprites will not reverse direction.

You can create one way systems within the maze. Please see the example. below.

Consider a T junction, the node path runs up and down the screen and another node path connects from the left. At the connection point if the node only allowed the sprite to leave either up or down, you have created a one way system. A sprite travelling up the path reaches the node and can only continue to travel upwards. Another sprite travelling from the left reaches the node and can then travel up or down. By constructing a number of junctions, a very complex maze can be set up.

SPEED OF YOUR PROGRAM

Every effort has been made to make the supervisor machine code routines as fast as possible. The problem that occurs with this sort of program is that the more general purpose you make the routines the slower they become.

We have tried to speed up the program execution as much as possible by automating many of the commands. There comes a time when we have to draw a line.

The problem with writing games in BASIC is speed. You will need to use BASIC to link all the commands within the supervisor together. This will slow your programs down. It is up to you to write your games as efficiently as possible.

Try to use the supervisor to do your work for you. Incorporate as many of the automated commands as possible (MISSILE, SOUND EFFECTS, MOVEHIT etc). kiln

Some of the commands used within the supervisor are relatively slow compared to others. As an example the REPORT command has to check every sprite on the screen. Plan your game so that only a certain group of sprites could be in collision and limit the REPORT command to this section of sprites.

Try to keep the sprites as small as possible.

When defining sprites keep them in a block. Keep the sprite numbers as low as possible. If you define sprite 0 and sprite 63 and you issue a command that operates on all the sprites ie MOVEALL all the sprites need to be checked (even though they may not have been defined).

If after all this your program runs at a snails pace there are a number of alternatives.

1) We will be selling a compiler to convert your BASIC program in to machine code. This will dramatically improve the speed of your program.

2) If you have knowledge of machine code, we will be selling a cutdown version of the supervisor with documentation of all the entry points and variable information.

Please contact us about any of the above options for availability.

If you do have problems with the supervisor program please write to us and we will try to resolve the problem as quickly as possible.

It you have any comments or useful ideas for improvements to the program please contact us. if we like the idea you will receive a revised and updated copy of the program free of charge.

SPRITES SUPERVISOR PROGRAM

APPENDIX I

SCREEN DIMENSIONS

MODE 0

SCREEN	CHARACTERS	DIMENSIONS
N	20 x 25	160 x 200
0	19 x 16	152 x 208
1	18 x 28	144 x 224
2	17 x 30	136 x 240
3	16 x 31	128 x 248
4	21 x 24	168 x 192
5	22 x 23	176 x 184
6	23 x 22	184 x 176
7	24 x 21	192 x 168

MODE 1

SCREEN	CHARACTERS	DIMENSIONS
N	40 x 25	320 x 200
0	38 x 16	304 x 208
1	36 x 28	288 x 224
2	34 x 30	272 x 240
3	32 x 31	256 x 248
4	42 x 24	336 x 192
5	44 x 23	352 x 184
6	46 x 22	368 x 176
7	48 x 21	384 x 168

SPRITES SUPERVISOR PROGRAM

APPENDIX II

ERROR MESSAGES

01	incorrect number of variables used in command.
02	The drawing number is greater than 63.
03	The sprite number is greater than 63.
04	The drawing has not been defined.
05	The sprite has not been defined.
06	The starting number is greater than finish number.
07	The sprite has not been on the screen.
08	The sprite is already on the screen.
09	The drawing has been defined already.
10	The sprite has already been defined.
11	The sprite is outside window coordinates.
12	The X coordinate equals zero.
13	The Y coordinate equals zero.
14	The X coordinate is greater than 32.
15	The Y coordinate is greater than 32.
16	The variable used is greater than 7.
17	The X speed is greater than 10.
18	The Y speed is greater than 10.
19	The screen mode is greater than 1.
20	The edge variable equals zero.
21	The edge variable is greater than 4.
22	The collision attribute equals zero.
23	The collision attribute is greater than 3.
24	The collision switch is greater than 1.
25	The variable used is greater than 63.
26	The left speed is greater than 10.
27	The right speed is greater than 10.
28	The up speed is greater than 10.
29	The down speed is greater than 10.
30	The window is to wide.
31	The window is to narrow (X).
32	The window is to low.
33	The window is to narrow (Y).
34	The window is to tall.
35	The colours have not been defined.
36	The sequence is greater than 15.
37	The animation sequence is greater than 10 drawings.
38	The drawing within sequence has not been defined.
39	The X dimension within sequence is different.
40	The Y dimension within sequence is different.
41	The sequence has not been defined.
42	The current sprite drawing number is greater than the highest number within sequence.
43	The current X coordinate is different to X coordinate within sequence.
44	The current Y coordinate is different to Y coordinate within sequence.
45	The missile type has not been defined
46	The missile type is greater than 2.
47	The missile type is already defined.
48	The missile drawing has not been defined.
49	The missile X offset is greater than 40.
50	The missile Y offset is greater than 40.

ERROR MESSAGES (CONT)

51 The distance is less than 2.
52 The colour is greater than 26.
53 The ink is greater than 15.
54 The number of spots equals zero.
55 The explosion increase is greater than 32.
56 The explosion limit is greater than 128.
57 The variable cannot be zero.
58 The variable is greater than 9.
59 The missile X speed -is greater than 10.
60 The missile Y speed is greater than 10.
61 The missile drawing is greater than 63.
62 The missile drawing equals 0.
63 Sprite 0 and Sprite 1 cannot be missiles.
64 There are more than 10 missiles defined.
65 The sprite used for a missile is already defined.
66 The missile type has been mismatched.
67 The missile delay is greater than 63.
68 The missile distance is greater than 127.
69 The missile delay equals 0.
70 You cannot do that to a missile sprite.
71 You cannot alter that on sprite 0 or sprite 1.
72 Missile overflow. Type 0.
73 Missile overflow. Type 1.
74 Missile overflow. Type 2.
75 You cannot use flashing colours.
76 The coordinate is outside window (x).
77 The coordinate is outside window (y).
78 The drawings must be defined before the nodes.
79 The speed cannot equal zero.
80 The nodes have not been loaded onto the supervisor.
81 The sprites node speeds have not been defined.
82 The sprites node attribute has not been defined.
83 The node attribute cannot equal zero.
84 The node attribute is greater than 31.
85 The sound number can not be greater than 15.
86 The amplitude envelope can not be greater than 15.
87 The tone envelope can not be greater than 15.
88 The tone period can not be greater than 4095.
89 The tone period can not be greater than 31.
90 The initial amplitude can not be greater than 15.
91 The sprite sound action can not equal zero.
92 The sprite sound action can not be greater than 4.
93 The sound has not been defined.
94 The sprite number is greater than 1.
95 The x coordinate has not been defined.
96 The node direction is not correct.
97 The y coordinate has not been defined.
98 The x coordinate is greater than 383.
99 The y coordinate is greater than 247.

SPRITES SUPERVISOR PROGRAM

APPENDIX III

ADVANCED USER NOTES

Due to the technique used by the supervisor program to detect for collisions, collisions between certain inks will not be ignored. This can be used to your advantage. The following table shows which inks when collided will not register a collision. Find the two inks used along the top and the left hand side of the table, follow the paths to where the two inks intercept. If there is a cross character at the intersection the supervisor will not register a collision. We have used Hex notation around the boundrys of the table, if you do not understand Hexadecimal then substitute A=10, B=11, C=12, D=13, E=14, F=15.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 :	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1 :	X	0	X	0	X	0	X	0	X	0	X	0	X	0	X	0
2 :	X	X	0	0	X	X	0	0	X	X	0	0	X	X	0	0
3 :	X	0	0	0	X	0	0	0	X	0	0	0	X	0	0	0
4 :	X	X	X	X	0	0	0	0	X	X	X	X	0	0	0	0
5 :	X	0	X	0	0	0	0	0	X	0	X	0	0	0	0	0
6 :	X	X	0	0	0	0	0	0	X	X	0	0	0	0	0	0
7 :	X	0	0	0	0	0	0	0	X	0	0	0	0	0	0	0
8 :	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0
9 :	X	0	X	0	X	0	X	0	0	0	0	0	0	0	0	0
A :	X	X	0	0	X	X	0	0	0	0	0	0	0	0	0	0
B :	X	0	0	0	X	0	0	0	0	0	0	0	0	0	0	0
C :	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0
D :	X	0	X	0	0	0	0	0	0	0	0	0	0	0	0	0
E :	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F :	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

When the sprites overlap each other their colour will change. The following table will show the colour the sprite will change to. We will describe on the next page the advantages of using these tables.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 :	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1 :	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2 :	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3 :	3	2	1	0	7	6	5	6	B	A	9	8	F	E	D	C
4 :	4	5	6	7	0	1	2	4	C	D	E	F	8	9	A	B
5 :	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6 :	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7 :	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8 :	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9 :	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A :	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B :	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C :	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D :	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E :	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F :	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

APPENDIX III

ADVANCED USER NOTES

We have stated throughout that once a sprite collides with another sprite they both change direction. This is not strictly true, there are ways to get around this.

Consider a platform game. The character will probably need to climb up a ladder. The ladder cannot be the same ink as the background colour because you would not see it. It you look at the tables on the previous page you will notice that collisions between certain ink combinations can not be detected. If the main character was one off these inks and the ladder was another, the character could move up the ladder without the collision detection routine knowing about it. However there is still one problem. When two inks are merge together they form a new ink. This is were the second table comes in. If we set the new ink to the same colour as the main character nobody will be able to tell when the character moves over the ladder. I am using a ladder as an example, you can apply this technique to many games.

Example.

The main character sprite is defined in ink 1 and the ladder is defined in ink 10 (A). We can see from the first table that a collision cannot be detected between these two inks. Looking at the second table we find that a collision between ink 1 and ink 10 (A) results in a change of ink the ink 11 (B). If we now make ink 11 the same colour as ink 1, you will not be able to detect the change in colour.

Taking this process one step further, if you study the first table you will notice that ink 8 will not have collisions detected if it collides with inks 0 to 7. Looking at the second table when inks 0 to 7 collide with ink 8 they all produce an ink greater than 7. Therefore we can have two colours for background colours, however we have limited the amount of foreground colours to 7.

If you find it acceptable to only have seven foreground colours and you want two background colours, follow these simple instructions.

The two background inks must be 0 and 8. When you have decided which colours you need for your foreground colours, these colours should also be copied into inks 9 to 15.

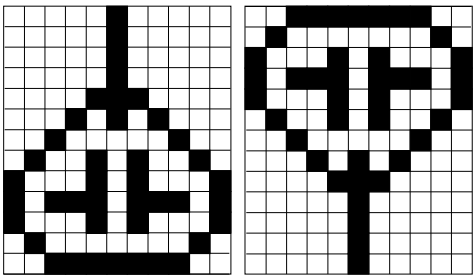
ie

```
     ink 1 = ink 9,   ink 2 = ink 10,  ink 3 = ink 11,
     ink 4 = ink 12,  ink 5 = ink 13,  ink 6 = ink 14,
                      ink 7 = ink 15
```

Sprite Supervisor Program

If you find your animated sprites are sticking to each other or to scenery, the following instructions should eleviate the problem.

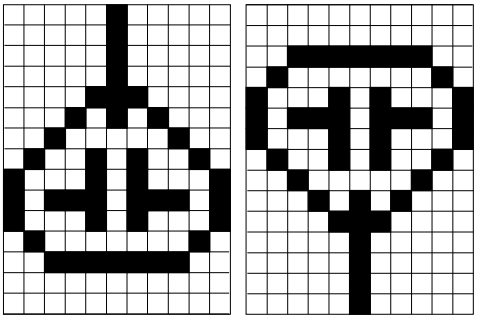
Under normal circumstances your drawings should look like this, no space around the outside of the sprite. The problem of sticking occurs when the sprite changes direction. If the first sprite was going up the screen and hits an object, it would reverse (shown in drawing B).



You have a tank which is animated to move up and down the screen pictured above. If you find the tank sticks with another object the best method to avoid the problem is to make the area around the sprite larger.

When the tank reverses by swapping drawings the new drawing may cover more of the hit object than the last drawing did.

To get around this problem you should leave a space behind the drawing that is liable to stick. If a stick still occurs try increasing the space to 3 or 4 pixels. If this does not solve the problem you will need to redesign your drawing.



SPRITES SUPERVISOR PROGRAM

APPENDIX III

ADVANCED USER NOTES

The previous page has suggested a way of avoiding collisions by making the drawing size larger than the actual drawing. There is another method you may use to prevent animated drawings from sticking together.

If you draw a box around your drawing in an ink that will nearly always detect collisions ie ink 7 and then set this ink to the same colour as the background colour. The user will never see the box, however the supervisor will detect whenever that box is in collision and allow the sprite to change direction without becoming stuck.

We have used this principle in the nodes demonstrations (DEMO5 and DEMO6).

In case you were wondering how we managed to get the scenery looking like it did in DEMO6, we used the Advanced Art Studio program.
Its not a simple job and it would take many pages of instructions if we told you how to do it in a step by step manner. I will explain the basic principles to you and you can then work it out for yourselves.
It you are going to load drawings into the supervisor with the |DRAW command, you will need to make a note of the colours you have selected. When you enter the Art Studio yo will need to alter the palette colours to suit the colours selected within the sprites designer program.
Next design your screen layout. We defined a pattern and filled the screen with the pattern, then we went to town with the spray can.
Next you will need to alter a pattern with the pattern editor to be the same size as the sprites you are going to use. Once you have a pattern the same size, you should clear the pattern to ink 0 (the background colour). Select the Brush option and using the cursor keys cut the tracks out of your painted screen.
You will need to note the coordinates every time you change direction (create a node). Once you have finished save th screen without compression turned on.
You should now have a list of all the coordinates where the nodes should be. The coordinates used are the Amstrad coordinates and these will need to be converted to sprite coordinates before entering them into the nodes program.
The Art studio gives you the coordinates for the centre of the brush. You will need to convert these to the top left hand corner. To do this divide the size of the brush by 2 for both the x and the y. Subtract the x size from the converted Art studio x coordinate. Add the y size to the converted Art studio coordinate. The result of this should be the, coordinates you should type into nodes program.

Sorry we have ran out of space.

SUPERVISOR INDEX

SPRITES SUPERVISOR PROGRAM

COMMAND	PAGE	BRIEF DESCRIPTION
MOVE	21	Move a sprite.
MOVEALL	21	Move all the sprites.
MOVEHIT	25	Check for collision of the last sprite that was moved.
NEXTREP	24	Report next sprite ill collision.
NODE	40	Load nodes into memory.
NODEALTER	42	Alter leaving directions of node,
NODEATTR	41	Set attributes fur sprite.
NODEOFF	42	Turn a sprite off to use nodes.
NODEON	42	Turn a sprite on to use nodes.
NODESPEED	41	Set speed of sprites in maze.
NODESPRITE	42	Determine chaselflee sprite.
READKEB	20	Test a direction from the keyboard.
READSTIX	20	Test a direction from the joystick.
REPOFF	23	Turn collision reporting off.
REPON	22	Turn collision reporting on.
REPORT	24	Cheek all sprites for collision and store result in a list,
RESEF	15	Clear sprite variables.
SATTR	38	Set sound attributes for sprite,
SCENERY	36	Put a drawing onto the screen.
SCREEN	9	Alter the shape of the screen,
SDIR	17	Define the speed of a sprite.
SDRAW	27	Report the drawing a sprite is using.
SEQUENCE	28	Define an animation sequence.
SGET	14	Link a sprite to a drawing.
SHOOT	34	Fire a missile from a sprite.
SMEM	27	Report amount of memory free.
SOUND	38	Define a sound.
SPEEK	37	Report ink at coordinates.

SPRITES SUPERVISOR PROGRAM

SPRITES DESIGNER

GLEN COOK

SPRITE DESIGNER PROGRAM

The Sprite Designer program on your system disk will enable you to design 64 multicoloured sprites with ease. The program will only run on an AMSTRAD 6128, as the program uses the extra banks of memory.

To use the program, simply insert the correct side of the disc into the drive and type

RUN "DESIGN"

You will then be asked to insert a disc, containing your drawing data, into the drive. Don't panic if you do not have any data saved yet. We have included a number of sample files on the disc for you to use. Press any key on the keyboard. After a short time you will be presented with a list of the drawing files stored on the disc.

1) DATA1
2) GAME
3) CREATE NEW FILE

This is how a typical menu may look. This menu is telling you that there are two drawing files stored on the disc called 'DATA1' and 'GAME'. The third option allows you to create your own drawing file.

Select option 1 by pressing the '1' key

After a short delay, the MAIN menu will be displayed.

THE OPTIONS MENU

1) EDIT A DRAWING
2) VIEW THE DRAWINGS
3) ANIMATE THE DRAWINGS
4) DISC OPERATIONS
5) DRAWING EDITOR
6) SPRITE INFORMATION
7) ALTER THE COLOURS
8) CREATE SPRITE DATA
9) END THE PROGRAM

PLEASE MAKE YOUR SELECTION

We will look at each individual option shortly. To select an option, press the relevant number. You must use the numbers along the top of the keyboard as the function keys on the right have been disabled.

Select option 2.

You will he told the number of drawings defined and the drawing range. The program is asking you if you would like to see all the drawings. Press 'Y'.
You are now shown the drawings that are stored in memory at the moment. You can use these drawings to experiment with the various options within the program.

SPRITE DESIGNER PROGRAM

If at any time throughout the program you make a mistake, ie select the wrong option, you can press the 'ESC' key to return to the MAIN menu.

We will now go through each individual option.

OPTION 1 : EDIT THE DRAWINGS

After selecting this option you will be asked

PLEASE ENTER DRAWING NUMBER

You may type any number between 0 and 63. After you have typed this number press 'RETURN'.

If the drawing number you have typed is a drawing that has already been defined the program will display the design grid with the drawing displayed in it, however if it is a new undefined drawing number you will be asked

PLEASE ENTER DRAWING COORDINATES

You then have to enter the drawings x and y coordinates. You may use any coordinate between 1 and 32. the maximum size of drawing you may use is 32 x 32.

PLEASE ENTER DRAWING DESCRIPTION

You may enter a description upto 10 characters in length. You must enter at least one character for the description or you will not be able to continue with the program.

After entering a description you will be presented with the design grid. This is the screen with which you will design all your drawings.

Along the top of the screen are the 16 colours that you may use to design your drawings. Below these colours are the numbers 0 to 9 and the letters A to F. To select a colour you simply press the relevant number or letter.

In the middle of the screen is the main grid, this consists of a number of squares, the number of squares there are depends on the size of the coordinates you selected. The squares will also vary in size depending on the drawing dimensions you have chosen. The grid will always be to a scale size. ie a larger version of the actual drawing.

The area to the right hand side of the design grid is where the drawing will be placed. This drawing is normal size.

At the bottom of the screen is the information area. This area informs the user of a number of things.

```
DRAWING NO.  : 05          DESCRIPTION  :  BIRD LEFT
DIMENSIONS   : 16 x 20     SPRITE SIZE  :  00360    X=01  Y=01
SPRITE SPACE : 12324       SPACE LEFT   :  25776
```

SPRITE DESIGNER PROGRAM

The DRAWING NUMBER and DESCRIPTIONS are self explanatory.

The DIMENSIONS are the x and y dimensions of the grid on the screen.

The SPRITE SIZE is the amount of memory it will take to store the drawing that you will design on the screen. It is important to use as little memory as possible. This can be achieved by making sure there are no spaces left around the outside of your drawing. If there are spaces left after you have finished your drawing, you may alter the dimensions as described later on in the manual.

The SPRITE SPACE is the amount of memory that all the drawings, you have so far designed, take up.

The SPACE LEFT is the amount of memory you have left with which to design your drawings. You originally have 38100 Bytes. Please bear in mind that you also have to write a program to control the sprites in memory as well. Once again try to use as little memory as possible.

The X=01, Y=01 are the coordinates of the cursors within the grid.

You can move the cursors around the outside of the grid by pressing the arrow keys. As you move the cursors around the grid you will note that the X and Y coordinates change.

To design a drawing you should move the cursors to the square you want a colour on and press the relevant colour key. If you make a mistake you may clear it by pressing '0' (the background colour) on the square you made the mistake on.

You way notice that when you select a colour, the colour you selected appears at the top right hand side of the screen. This shows the user the selected colour. You may now fill in squares within the grid with that colour by pressing the SPACEBAR. To change the selected colour, simply press a different colour number or letter.

You may now try the options menu. To do this press the 'ESC' key once. The bottom of the screen will clear and the options menu will appear.

f1	RENAME	f4	DELETE	f7	MOVEMENT
f2	COPY	f5	CLEAR	f8	COLOUR SEARCH
f3	COLOUR LATCH ON	f6	DIMENSIONS	f9	MAIN MENU

To select an option press the appropriate function key.

To return to the design grid press the 'ESC' key again.

This is the only screen where pressing 'ESC' does not return you to the MAIN menu. To return to the MAIN menu when using the design screen you should press ESC, to get the above menu and then select function key 9.

f1 RENAME

Pressing f1 will allow the user to alter the description of the current drawing.

After pressing f1 the old description is displayed on the screen. The user may now enter a new description. If you do not wish to do this, press 'ESC' to return to the menu.

After entering a new description press 'RETURN' to return to the design grid.

f2 COPY

Pressing f2 will allow the user to copy the current drawing to another drawing area. This is important for animation as the user can then alter just a small section of the drawing instead of having to do the whole drawing from the start.

After pressing f2 the user will be prompted to enter

COPY THIS DRAWING TO DRAWING NUMBER :

If you do not want to copy to another drawing press 'ESC'.

You should now type in the drawing number you want to copy this drawing to and press 'RETURN'. If the drawing number you entered already has a drawing stored there, you will be shown the drawing you would overwrite at the bottom right of the screen and asked

YOU WILL OVERWRITE THE DRAWING ON THE BOTTOM RIGHT,
CONTINUE Y/N

If you select N then you will return to the design screen.

If you select Y the drawing will be overwritten by the current drawing.

PLEASE NOTE : WHEN YOU COPY A DRAWING AND RETURN TO THE MAIN DESIGN GRID, THE CURRENT DRAWING IS NOW THE DRAWING THAT HAS JUST BEEN COPIED. LOOK AT THE DRAWING NUMBER. BE WARNED !!

f3 COLOUR LATCH

Pressing f3 will toggle the colour latch between on and off.

The colour latch option is the function that allows the spacebar to remember the colour that was last typed in. When the colour latch is on every time you press the spacebar in the design screen the colour that is at the top right hand side of the screen is placed at the X,Y cursor position. If you turn the colour latch off, the spacebar will not remember the last colour pressed, it will instead act as an eraser. It will clear the colour at the X,Y cursor position to the background colour.

Press 'ESC' to return to design grid.

f4 DELETE

Pressing f4 will allow the user to delete the current drawing from memory.

After pressing f4 the user will be asked

ARE YOU SURE YOU WANT TO DELETE THIS DRAWING Y/N

If you select 'N' you will return to the design grid with your drawing still intact.

If you select 'Y' the drawing will be erased and you will return to the MAIN menu.

f5 CLEAR

Pressing f5 will allow the user to clear the current grid but keep the grid in memory.

After pressing f5 the user will be asked

ARE YOU SURE YOU WANT TO CLEAR THIS DRAWING Y/N

If you press 'N' you will return to the design grid with your drawing still intact.

If you press 'Y' the drawing grid will clear and you will return to the design grid.

PLEASE NOTE : You should not clear the grid and then return to the MAIN menu. Even though there is no drawing stored in the grid it will still count as a drawing. If you intend to return to the MAIN menu without a drawing stored, you should use the f4 option and delete the grid from memory.

f6 DIMENSIONS

Pressing f6 will allow the user to alter the dimensions of the current grid. You may increase or decrease the size of the grid.

After pressing f6 the user will be asked

```
OLD DIMENSIONS  : 16 x  20
NEW DIMENSIONS  :    x
```

You should now enter the dimensions that you would like the grid to be.

If you are enlarging the grid then the design grid will be redrawn with the larger dimensions.

If you are reducing the grid, the area that will be cut off will be shaded. This will enable the user to see if they will lose any of the drawing.

After the area to be cut off is shaded the user will be asked

WITH DRAWING DIMENSIONS OF 10 x 10 YOU WILL LOSE THE DRAWING DATA IN THE SHADED AREA. DO YOU WISH TO CONTINUE Y/N

If you select 'N' the full grid will be redrawn and you will return to the design grid.

It you select 'Y' the grid will be redrawn with the new coordinates and you will return to the design screen.

USEFUL TIP: When you are going to reduce the size of the grid, if you move the x,y cursors to the squares you want to reduce the drawing to, by noting the x,y coordinates at the bottom of the screen you should save yourself counting the squares by hand.

f7 MOVEMENT

Pressing f7 will allow the user to manipulate the drawings. You may make a mirror image of the drawing or invert the drawing.

After pressing f7 the user will be asked

PLEASE MAKE YOUR SELECTION : M - MIRROR IMAGE
: I - INVERT THE DRAWING

If you do not wish to do either of these options, pressing 'ESC' will return you to the design grid.

Pressing 'M' will produce a mirror image of the drawing. This is useful if used along with the copy command. You only need to design a drawing for one direction. If you then copy the drawing to a free area and press f7 the drawing will be produced to move the sprite in the opposite direction.

Pressing 'I' will invert the current drawing.

f8 COLOUR SEARCH

Pressing f8 will allow the user to replace any colour on the grid with another colour.

After pressing f8 the user will be told to

PLACE X,Y CURSORS OVER THE COLOUR YOU WISH TO CHANCE AND PRESS THE COPY KEY.

You can now move the cursors around the grid until they intersect on the colour you want to change. Press the COPY key to select the desired colour. You will then be asked to confirm the colour.

PLEASE CONFIRM YOU WANT TO CHANGE COLOUR : 5 Y/N

If you press 'N' then uou will return to the design grid without out any alterations being made.

If you press 'Y' you will then be asked

CHANGE COLOUR : 5 TO NEW COLOUR :

You must now enter the colour number or letter that you want the old colour changed to.

After entering this number/letter the message will read

YOU CANNOT ALTER ANY MISTAKES AFTERWARDS. ARE YOU SURE Y/N

If you press 'N' you will return to the design screen without any alterations being made.

If you press 'Y' the old colour will be replaced with the new colour and you will return to the design grid.

WARNING : If you alter a colour to a colour that is already being used on the design grid, and you try to change it back you will alter all of the colour and not just the colour you originally changed. Be careful !!

f9 MAIN MENU

Pressing f9 will allow the user to return to the MAIN menu. The drawing that is currently in the design grid will be saved to memory.

NOTES on option 1:

If you are storing a large number of drawings in memory keep a careful eye on space left. If instead of displaying a number it shows 'OUT OF MEMORY' you do not have enough memory left to use option 8 on the MAIN menu.

SHIFTING THE DRAWINGS

The user may move the drawings around inside the design grid by pressing the SHIFT key and then the appropriate arrow key. If you shift part of the drawing over the edge of the grid it will be lost and you will not be able to recover it.

OPTION 2 : VIEW THE DRAWINGS

After selecting this option you will be shown

DRAWINGS DEFINED : 20
DRAWING RANGE : 00 - 31

DO YOU WISH TO SEE ALL THE DRAWINGS Y/N

This is giving the user certain information about the drawings that have been stored in memory so far.

DRAWINGS DEFINED : Tells the user the number of drawings that have so far been designed.
DRAWING RANGE : Tells the user the lowest and highest drawings that have so far been designed.

If you press "Y" to the question the screen will show upto 16 drawings. If more than 16 drawings have been designed you will see the rest of the drawings after pressing any key.

If you press 'ESC' whilst the drawings are being displayed you will return to the above option menu. Likewise if you press 'ESC' at this menu you will return to the MAIN menu.

If you press 'N' to the question you will then be asked

PLEASE ENTER DRAWING RANGE: x

You should now enter the lowest drawing number you wish to see, followed by the highest drawing number. After entering these two values the drawings will be displayed.

OPTION 3 : ANIMATING THE DRAWINGS

After selecting this option you will be presented with a grid of 64 numbers ranging from 0 to 63. These numbers represent the drawings stored in memory. Underneath drawing 0 there is an arrow.

If any drawings have been defined the numbers of that particular drawing will be a different colour. If the colours have not been changed with option 7, the undefined drawing numbers will be blue and the defined drawing numbers will be red.

At the bottom of the screen there will be the description of the drawing that the arrow is pointing to. If the arrow is pointing to an undefined drawing the bottom of the screen will display 'NOT DEFINED'.

If the drawing is described the x,y dimensions of the drawing will be displayed at the bottom left of the screen.

IMPORTANT NOTE : All drawings that are to be animated must have the same x and y coordinates.

You may move the arrow around the screen by using the arrow keys. To select a drawing to be animated press the SPACEBAR. The drawing numbers you have selected will be listed along the bottom of the screen. If you make a mistake press the 'DEL' key and the drawing number will be removed.

You may select up to 8 drawings to be animated.

After selecting all the drawings you want animating press the RETURN key.

If you have made a mistake (drawings having different x,y dimensions) the error screen will be displayed.

The error screen lists all the drawings you have selected and it will highlight any drawing with different x,y. coordinates. After pressing a key you will be returned to the selection menu to try again.

Once you have selected the drawings and there are no errors you will be presented with the animation options screen.

The screen looks like this :

SPEED OF SPRITE :

SELECT DIRECTION (U,D,L,R) :

DELAY BETWEEN MOVES :

The speed of sprite option will determine how many pixels the sprite will move at a time. To get the smoothest movement you should select '1' and press RETURN. You may use any number up to a maximum of 9. If you want the sprite to be animated but not to move you should select '0'.

The select direction option will allow the user to determine the direction in which the sprite is to move, Up , Down , Left or Right. If you have selected the sprite to be stationary by selecting '0' at the last option you may type any direction, the sprite will be stationary. There is no need to press RETURN on this option.

The delay between moves option allows the user to select the speed between sprite movements. You may use any number between 0 and 99. An average number is between 30 - 40.

It you select '0' the sprite will move everytime you press a key. This feature is useful to check the slow motion of a sprite to ensure the animation is correct.

After pressing RETURN on this option the screen will clear and you will then see the sprite being animated. To return to the MAIN menu press the 'ESC' key.

OPTION 4 : DISC OPERATIONS

It is important for you to know how the information is stored on the disc. This should prevent any mishaps and possible heartache.

On the disk there is a file called 'DISKDATA.SPR'. This file holds information about the sprite files that are stored on the disk. If you decide to erase any of the files off the disc please use the erase function within this option.

If you erase any of the drawing files off the disc without using this option, the load menu will still display the file as being present. This is because the drawing file names are stored in DISKDATA.SPR. Only the erase function within this option will erase the relevant information from this file.

If you erase the DISKDATA.SPR file from the disc you will not be able to recover any of the drawing data files that might be stored on the disc. BE CAREFUL !

If you wish to copy some drawing files from one disc to another, you should load the drawing data file you wish to copy off the disc, insert the disc you want to copy it to and then save the file.

A blank formatted disc can store 4 sets of drawing information. Each file of drawing information takes 36k. Try to ensure when you are going to save drawing data to disc you have enough space left on the disc.. Don't lose sleep over it though, the program does check to ensure there is enough space. It just saves you a bit of time thats all !

The program does its own housework ie. it automatically erases any backup copies it creates.

The drawings are stored in the format

filename.da1 : first bank of memory (17k)
filename.da2 : second bank of memory (17k)
filename.da3 : drawing colour info (2k)

If you want to store drawing data on a new disk, ensure that it is formatted. It does not matter that there is no DISKDATA.SPR file on the disc, the program will create one.

If you make a mistake and the disc drive has started, DO NOT try to remove the disc. It's a had habit and you could lose all the data on the disc. The DISKDATA.SPR file is erased before it is updated.

Alter selecting option 4 you will be presented with the menu

1) LOAD DRAWINGS
2) SAVE DRAWINGS
3) ERASE DRAWINGS
4) RETURN TO MAIN MENU

LOAD THE DRAWINGS

After selecting this option the warning message will be displayed

YOU WILL DESTROY INFORMATION IN MEMORY CONTINUE Y/N

If you select 'N' you will return to the above menu.

SPRITE DESIGNER PROGRAM

If you select 'Y' you will see displayed the message that is shown when you first run the program.

PLEASE INSERT DISC WITH DRAWING DATA INFORMATION AND
PRESS ANY KEY

You should now insert the disc that you would like to load and press any key.

If you have inserted a disc that does not have a DISKDATA.SPR file on it you are given the following options

DATA NOT FOUND

1) CREATE NEW DATA DISC
2) TRY ANOTHER DISC

It you select option 2 you are prompted to insert another disc and try again.

Selecting option 1 will create a DISKDATA.SPR file on the disc.

If the disc you inserted has no drawing files stored on it or you have just created a new DISKDATA file you will get the message

NO DRAWING FILES STORED AS YET
PRESS ANY KEY TO CONTINUE

Pressing any key will display the MAIN menu.

If the disc you inserted had an files on it, the files will be displayed along with a number to the filenames lefthand side. Typing this number will load the relevant file.

At the end of the filenames will be a CREATE NEW DATA option. This option will allow the user to start a new file which he can save with the save option that is documented next.

If there has been a mishap and the file you try to load is missing or corrupt the program will check the complete disc to see if any other files are missing. If any other files are found to be missing their names will be reported and the DISKDATA.SPR file will be updated. The user will then be offered a choice of loading a file from the updated catalogue.

SAVE THE DRAWINGS

If the file you are going to save is not a newly designed file ie has been loaded off the disc, you will have the following message displayed.

CURRENT FILENAME : DATA1

DO YOU WISH TO SAVE TO THIS FILE Y/N

If you select 'N' you will then be asked

ENTER NEW FILENAME :

You may now enter a filename of up to eight letters.

The file will now be saved to either the old filename if you selected 'Y' as an option or, saved as the new filename if you selected 'N' as the option.

If there is not enough room on the disc to store the file to, you will he told to insert another disc and try again.

ERASING THE DRAWINGS

After selecting this option you will be prompted to

PLEASE INSERT DISC WITH DRAWING DATA FILE YOU WOULD LIKE
TO ERASE AND PRESS ANY KEY

After inserting the disc with the file(s) you want to erase and pressing any key, you will be displayed a list of the files on the disc. It will look like this

ERASING THE DRAWINGS

1) DATA1
2) TEST
3) TEST
4) RETURN TO MENU

PLEASE MAKE YOUR SELECTION

You may now select one of the files by typing the relevant number. If you select the option RETURN TO MENU you will be returned to the MAIN sprite program menu.

After selecting the file to erase, the screen will clear and you will be asked to confirm the file you want to erase.

PLEASE CONFIRM YOU WANT TO ERASE

DATA 1 Yes or No

If you decide not to erase that particular file you should Press the 'N' key, this will return you to the directory of files screen.

If you select 'Y' the file will he erased from the disc and you will be returned to the MAIN sprites menu.

RETURN TO MAIN MENU

This option will simply return you to the MAIN sprites menu.

SPRITE DESIGNER PROGRAM

OPTION 5 DRAWING EDITOR

After selecting this option you will be presented with a grid of 64 numbers ranging from 0 to 63. These numbers represent the drawings stored in memory. Underneath drawing 0 there is an arrow.

If any drawings have been defined the numbers of that particular drawing will be a different colour. If the colours have not been altered using option 7. the undefined drawing numbers will be blue and the defined drawings will be red.

At the bottom of the screen, in the middle, there will be the description of the drawing that the arrow is pointing to. If the arrow is pointing to an undefined drawing the screen will display 'NOT DEFINED'.

Underneath the drawing description is a message

Delete , Rename , Copy , ESC to end

Pressing the relevant key (D,R,C,ESC) will enable that function to act on the drawing number the arrow is pointing to.

You can move the arrow key around the screen by using the arrow keys on the right hand side of the keyboard.

If you select any of the options above (except ESC) when the arrow is pointing at an undefined drawing, the computer will emit a beep and nothing will happen.

We will now look at the functions in more detail.

DELETE

Pressing the 'D' key when the arrow is pointing at a defined drawing will display the message

ARE YOU SURE YOU WANT TO DELETE Y/N

If you select 'N' this message will clear and the drawing will stay intact.

If you select 'Y' the drawing will he erased from memory, the colour of the particular number the arrow is pointing to will change and the description will now display NOT DEFINED.

RENAME

Pressing the 'R' key when the arrow is pointing at a defined drawing will display the message

OLD : old description NEW:

You are now prompted to enter a new description for that particular drawing. If you decide you want to keep the drawing description the same you may press the ESC key.

COPY

Pressing the 'C' key when the arrow is pointing at a defined drawing will display the following message

COPY DRAWING : 03 TO DRAWING :

You are now prompted to enter a drawing number. The program will now make a copy of the first drawing you selected and copy it to the drawing number you have just entered.

If you have entered a drawing number that has already been defined, an error message will be displayed.

YOU CANNOT OVERWRITE EXISTING DRAWINGS

This is self explanatory. If you do wish to overwrite a drawing, you must use the copy command from within EDIT A DRAWING.

If you did enter an undefined drawing number, the drawing will be copied.

ESC

Pressing the ESC key will return the user to the sprites MAIN Menu.

OPTION 6 SPRITE INFORMATION

After selecting this option the following screen will be displayed

DRAWINGS DEFINED : 30

DRAWING RANGE : 00 - 43

DO YOU WANT INFORMATION ON ALL DRAWINGS Y/N

This screen tells you the user, information about the drawings that are in memory at the moment. See option 2 VIEW THE DRAWINGS for this information.

If you want information on all the drawings press 'Y'.

If you press 'N' you will then be asked

ENTER DRAWING RANGE : -

You must now enter the lowest drawing and the highest drawing you want information on.

Then you will be asked

OUTPUT TO Screen OR Printer

You may select one by pressing 'S' or 'P'.

PRINTER SELECTED

It you selected printer the next question you will be asked

DO YOU REQUIRE CONTROL CODES SENDING TO PRINTER Y/N

If you want to send control codes to your printer. Press the 'Y' key. These control codes could be to change printer to a different font, etc.

If you do require control codes sending you should now see displayed

PLEASE TYPE EACH INDIVIDUAL CONTROL CODE AND PRESS RETURN

CONTROL CODE :

At this point you should enter each control code and press RETURN. When you have finished entering the control codes you should press the 'ESC' key. As you enter the control codes they will be displayed along the bottom of the screen. If you make a mistake press the 'DEL' key.

NOTE : You may enter a maximum of 6 control codes. They cannot be greater than 63.

You will now be asked

DO YOU WANT THE LINE FEED CONTROL CODE SENDING TO PRINTER Y/N

You should answer 'Y' to this question. If you find your printer is printing a blank line between each piece of information then in future you should answer 'N'.

PLEASE ENSURE YOUR PRINTER IS ONLINE AND READY TO PRINT
PRESS ANY KEY WHEN YOU ARE READY

Will now be displayed. After checking your printer, press any key. Your printer should print out the information on the drawings you have selected.

SCREEN SELECTED

The information that will be printed is as below

DRAW	DIMENSION	DESCRIPTION	MEMORY
00	32 x 32	MAN	1088
01	32 x 32	MAN	1088
		TOTAL MEMORY	2176

This information is needed for reference when using the main sprites program. The total memory value will be needed to calculate which value to use for the basic MEMORY command.

OPTION 7 : ALTERING THE COLOURS

After selecting this option you will be asked

DO YOU WISH TO ALTER

0) MODE 0 COLOURS
1) MODE 1 COLOURS

MODE 0 COLOURS : These are the colours that you will design your drawings with.

MODE 1 COLOURS : These are the colours for the menus and messages.

To select the type of colours you want to change press either '0' or '1'.

MODE 0 COLOURS

You will see displayed on the screen three bands of colours. The top two bands of colours are the 27 colours available on this computer. You may only count 26 colours, the background colour is also there but you can not see it.

The third band of colours are the colours that are currently selected for mode 0. Underneath this band of colours are the numbers 1 - 9 and the letters A - F. You can only use 16 colours in mode 0.

At the top of the screen underneath the first colour is an arrow. You can move this arrow from colour to colour by using the arrow keys.

To alter a colour, move the arrow to the colour you want to select and press the number/letter of the colour you want to change. The new colour will now be displayed in the bottom band.

You way also alter the background colour by moving the arrow to the colour you would like the background to be and press the SPACEBAR.

When you are using the design screen you will notice that the information at the bottom of the screen is in mode 2. The colour of the text is initially pink, colour 16. You may also alter this colour. You do not have to use one of the sixteen colours you have selected for mode 0. To change this colour, move the arrow key to the colour you want and press the COPY key. The information at the bottom of the selection screen (also in mode 2) will change. This will enable you to decide if this colour is readable against the background colour.

When you have made your final selection press the 'ESC' key.

NOTE : The colours for the background and the mode 2 foreground cannot be the same, if you try to do this the command will be ignored.

The colours you have selected will be saved along with the drawings, therefore you will not need to alter the colours when you load the drawings back into the program.

The colours will also be saved with the drawing data when you load the drawings into the main sprites program. To access the colours in the sprites program use the |COLOUR command.

MODE 1 COLOURS

Altering these colours will alter all the menus and messages that are displayed in mode 1. This information is saved when you save drawing data to disc.

When you select this option you will see displayed

```
BACKGROUND 0  CURRENT : 00   CHANGE TO :
FOREGROUND 1  CURRENT : 16   CHANGE TO :
FOREGROUND 2  CURRENT : 06   CHANGE TO :
FOREGROUND 3  CURRENT : 02   CHANGE TO :
```

You may now type in the numbers of the colours you want (0 - 27). When you have entered the four numbers the colours will change and the message

ARE THESE COLOURS OK Y/N

will be displayed If you press 'N' the colours will revert back to the colours that were used before the command was executed. You will then be asked to try again. If you do not wish to alter the colours press 'ESC'.

If you press 'Y' in answer to the above question the colours will be permanently changed and the MAIN menu will be displayed.

OPTION 8 CREATE SPRITE DATA

This is the command that will convert your drawings into a form that the main sprites program will understand.

When you have selected this option the following message will be displayed.

WARNING

YOU WILL NOT BE ABLE TO RETURN TO
THE SPRITE DESIGNER PROGRAM AFTER
YOU HAVE CREATED THE SPRITE DATA

SPRITE DESIGNER PROGRAM

YOUR DRAWINGS MUST BE FULLY READY
TO BE USED BY THE MAIN SPRITES
PROGRAM

ARE YOU SURE YOU WANT TO CONTINUE Y/N

In order for the program to convert the drawings into a form the main sprites program can understand the program has to erase a large chunk of itself. If you answer 'Y' to this question you will not be able to return to the designer program without reseting the computer and 'running' the program again.

IMPORTANT : SAVE YOUR DRAWINGS TO DISC BEFORE USING THIS OPTION

It you do not want to continue with creating the sprite data, pressing 'N' will return you to the MAIN sprite menu.

If you want to continue press 'Y'. You will now be prompted for a filename.

SAVE AS FILENAME :

You may use any filename up to eight letters long. After entering the filename press RETURN. The program will prompt you to

PLEASE INSERT THE DISC YOU WANT THE
DRAWINGS SAVING TO AND PRESS ANY KEY

You should now insert the disc on which you want the drawings storing. We advise against using the system disc.

After pressing any key the screen will change colours a few times. This is not unusual, the program uses the screen as a buffer for the drawings. Finally the screen will clear and display.

THE FILE HAS BEEN SAVED
YOU MAY NOW RESET THE COMPUTER BY
PRESSING SHIFT-CONTROL-ESC

The program has now ended.

The drawing file that hag been created will have the format

username.DKW

OPTION 9 : END THE PROGRAM

This option is to end the program and return back to BASIC.

After selecting this option the following message will he displayed.

SPRITE DESIGNER PROGRAM

PROGRAM TERMINATION

YOU WILL LOSE ALL THE INFORMATION IN

MEMORY IF YOU END THE PROGRAM I HOPE

YOU HAVE SAVED THE DRAWINGS TO DISK.

END THE PROGRAM Y/N

Pressing 'Y' will then reset the computer.

Pressing 'N' will return you to the MAIN menu with the data still intact.

HAPPY DESIGNING

(This page was missing from the original PDFs I was sent)