VZ SPRITE GENERATOR

REQUIREMENTS: VZ300/VZ200 Computer

No expansion memory needed

Joysticks optional

OVERVIEW: This package is only a utility, a program to make ‘BASIC’ Graphic programming much easier.

It is written in very fast machine language. It is also very easy to use if you follow a few simple steps outlined. It is designed for use with high-resolution graphics or MODE (1).

A sprite is a small graphic image that is very easy to move. Some computers have sprites ‘built in’, that is to say, they are resident in the computer all the time. Now this program gives you the same, if not more power than many other computers, many times the price. Once loaded, the utility stays until the computer is turned off. This package comes with many enhanced features not normally found on most other built in sprites.

SOME FEATURES INCLUDE: -

1/ Easy positioning of 8 X 8 pixel image (dot image).

2/ After the image is moved, the old background is restored as before.

3/ Joystick and keyboard testing is automatically done.

4/ The collision registers tell the programmer what colours have just been collided with.

5/ Boarders or windows for image may be set.

6/ A trail may be left behind the Image if desired.

7/ Automatic positioning is available.

To get the best Idea of the extra power of this utility, simple 'CLOAD' and 'RUN' the first program on the tape, which is called 'SPRITE DEMO #1’. Only two out of these features were used in this simple demonstration. The very fast speed is also shown up here. This demo program is only a small simple 8-line program and it only has to follow 3 special steps. These steps are fully explained later on.

To save the program follow just one more step, and your basic program and my machine language utility, will both be saved on the tape at the same time. This means that anyone can load your program and run it at once. How easy.

PROGRAMMER: Trent George

EXTRAS: A Sprite Editor to create the shape of your image. Five demonstration programs that show different use’s. Full instructional tutorials to help you get started.

LOAD TIME: Under 1 minute.

A BRIEF DESCRIPTION ON HOW IT WORKS.

The actual sprite utility program is written totally in machine language. To load this in simply 'CLOAD' in 'SPRITE PACKAGE’, the second program on the tape, and run it.

The machine language takes under 1 K of memory. The rest is for your BASIC program to use.

After you run the program, 'SUPER SPRITE EDITOR' will be displayed. This is where you design the Image you need. Refer to the editor page, for full description on all the commands available to edit.

After you are happy with their design, assemble them to memory, with the <Z> command, and reset your computer. This leaves you ready to program in BASIC with your newly designed sprites ready to be used.

This utility can also be customised exactly to your needs.

This is done with a simple POKE statement.

Lists of useful modifications are given in the customising section at the back. POKEing the addresses with the values given will activate the different features of the utility.

The command that drives the whole utility is the USR statement in the form of D=USR(N) where N is a number either representing a co-ordinate or -1 (automatic motion positioner).

Every time this command is executed, several things happen. First the joysticks and keyboard are read and the results are stored in a memory location. If the sprite has moved, the background under the sprite is restored to its original status before the sprite was displayed. The new position background is then saved and the sprite is then displayed at its new position. A count of each colour pixel being superimposed is stored in four colour collision memories.

To leave a trail you simple switch out background restore mode.

To find the value of these memory locations (listed at the back with the customising details) use a V=PEEK (M) where V is the variable to store the value and M is the memory location to be viewed.

The new position is reached in one of two ways. The first way is to give the actual co-ordinates of the new position, where N=X*256+Y. Here X Is an Integer (X=INT(X)) representing the X co-ordinate and Y represents the Y co-ordinate. Co-ordinate (0,0) is in the top left hand corner of the screen. Secondly, if N=-1, the automatic position mode is activated, and its new position will depend on the old position and the result of keyboard/joystick input. If the joystick is up and the sprite is not at the very top, the sprite will move up.

Only loading in another program that does not use the sprite, or by turning off the machine, will you lose the sprite utility. A NEW command will only clear your basic program, not the machine language.

TO START FROM THE BEGINNING. THE EDITOR.

You first CLOAD "SPRITE PACKAGE" and RUN it.

You have now just entered the sprite editor program. A list of commands are given down the left hand side of the screen for easy reference. The status of the editor is given in a summary at the bottom. The flashing cursor is moved by the joystick or keyboard.

If you find you are having trouble positioning the cursor, you will find by disabling the joystick <J> the keyboard may be more accurate. The joystick equivalents are: UP=<Q>, DOWN=<A>, LEFT=<M> and RIGHT=<, >.

When you see a letter enclosed by < >, this represents a command. To activate commands, simply press the corresponding key while the cursor is flashing. If the cursor is not flashing the program is probably In the middle of doing something, and it will not respond to commands. Just wait for the flashing cursor. Then edit the shapes into the desired form.

This is done by modifying the shape currently stored in memory. If you decide you want totally different Looking sprites, clear the sprites with the <R> remove command.

There are two sprites available for storage. They can be switched from the editor with the <N> command. The two images can be switched then from basic fairly simply with a POKE.

Choose the image you want to edit with the <N> command. Then position the flashing cursor with the joystick or keyboard. You may either reset a pixel (ARM) or set a pixel (FIRE). From the keyboard FIRE=<.> and ARM=<SPACE>.

Choose the right colour with the numbers <l> to <4>. <1>=GREEN, <2>=YELLOW, <3>=BLUE and <4>=RED. If using a colour TV the Images are very clear, but if using a black and white TV, simply switch to B/W mode with the <B> command. This will display all colours by their corresponding letters (e.g. G=GREEN, Y=YELLOW). The colour set can also be changed. If you Intend to use the sprites in hi-res. on COLOUR,1 or second colour set, simply press <C>. This converts the first four colours to the second four colours, so while you edit you can view the end result.

If you want two fairly similar images, use the <D> duplicate command. This moves the other image to the current image. You can then modify this copy. (This was done In the first demo program where one man has his hands up and the second man has his hands down. These two Images are swapped as the man bounces up and down to give an animated affect). If totally dissatisfied with your current Image you may clear it with the <R> remove command.

When satisfied with both Images, they must be assembled to memory, with the <Z> command. This stores the image in the right format for the utility to use. If you do not assemble, the old image will stay as the old sprites, no matter how you edited the images. After assembling the Image to memory, you may test them in the sprite mode with the <X> command. This will switch to hi-res. and let you move the Images. You may aiso swap images, change colour, set and disable the joysticks, all from test mode using normal command letters. Return to edit mode by pressing <X> again. Sprites should always be tested before you finish. If they don't look right, check that you have assembled them. This is indicated at the bottom of the edit screen. If you particularly like your images, and think you will use them for many other programs, save the new images to tape after assembling with the <U> command. Make sure the cassette is wound past the leader and the recorder has the play and record depressed before you press <U>.

One last thing must now be done and that is reset the computer. To reset, simply use the SHIFT<8> command. This will ask if you are sure. Reply with <Y> if you are otherwise reply with <N>.

Two hints can be given here. Firstly, if you have just finished editing two images and you feel the old original ones were better, simply break out of the program and run It again. This will bring back the last assembled images. Secondly, If you are going to use a series of the same Images, simply save the whole thing with the <U> command. This will save the last assembled image and editor program. This can be loaded in at any future date and it will save you having to re-edit your images.

Take care and read instructions carefully before using the program. It is powerful and when you understand how to use it properly, you will find the editor and machine language utility both very useful and powerful.

YOUR BASIC PROGRAM. HOW TO MAKE USE OF THE UTILITY.

This is the part of the tutorial that will explain what all these Mysterious POKEs are going to do. The techniques for using this utility are divided into several steps and each is described clearly. Just follow the text and you will not have any trouble. The program was designed for ease of use. Unfortunately the utility is so powerful that initially, the many different functions available may confuse you as to how to make the best use of it. A wise step is to read this section and look at and list the demo programs, to find out how they work.

After you have left the editor with SHIFT <8> - reset command, you are now, ready to make use of the utility. As mentioned before, just follow the simple steps, and you will not run into any trouble.

WARNING: Many pokes are used throughout and each has its purpose. Be very careful using the numbers inside these POKEs as one wrong number may wipe your program. Just follow the addresses and values given, and no problems will arise. If the worst comes to the worst, you can always turn off the computer and start again. No permanent damage can ever occur. Power down will always rectify the problem.

At the very first line of your basic program, you must specify the location of the sprite utility. In simple terms just copy the following line down as the first of each basic program:

```
10 POKE 30862,0:POKE 30863,124
```

After this step you must write any more programming you need before you use the sprite for the first time. Before the sprite Is used for the first time two things must be done. You must switch to hi-res. with the MODE(1) basic command. Any pattern must now be displayed (e.g. background maze or grids). Once all these preliminary tasks have been completed, you can Initialise, the sprite. This is done with the following basic line:

```
30 POKE31631, V1:D=USR(N):POKE31631,V2
```

Substitute V1 as follows: with 3 to initialise without displaying the sprite or 2 to Initialise and display the sprite.

Substitute N with -1 or X*256+Y as described before.

Substitute V2 as follows:

With 0 to set normal mode, with 2 to set non-restoring mode or 3 to leave sprite disabled.

If a combination of 3 and 0 are used, the sprite will not be displayed until it is first moved. When Initialising, It is also common to start the sprite at a particular position. This means letting N=X*256+Y. The most common combination to Initialise is V1=2 and V2=0. This will display the sprite at the Initial position and leave it in normal restoring mode.

Example: - 30 POKE 31631,2:D=USR(256*60+28):POKE 31631,0

This example will display the sprite in the middle of the screen and leave it in normal mode.

NOTE: The variable D may be substituted for any unused variable. The line numbers in this reference may also be substituted for any logical fine numbers.

The sprite is now Initialised and ready to move. The main reason for this utility is to be able to move objects in hi-res. with great ease. You are now ready after this short but necessary Initialisation, to take advantage of the powerful features of sprites. The most obvious and useful advantage of a sprite is its ability to move over background without disturbing it. Without sprites, this task In the BASIC language is far too slow and difficult. You would have to SET(X,Y) the Image and RESET(X,Y) the image every time it moved. This would also only affect one pixel. The sprite generator not only does all this for you, but also replaces the background and can also calculate the new position for you. It not only does this for one pixel, but for 8 by 8 or 64 pixels, quicker than basic executes two set commands.

Keeping track Of Position may be done in two ways. The easiest way is to make use of the automatic motion feature. This Is done using D=USR(-1) command. This will move the Image from the current position with respect to the state of the keyboard and/or the joysticks. The Image will only move each time a USR statement is executed, and the player is trying to make the Image move. The shorter the period between each USR statement execution, the smoother and faster the Image will move. You can double or triple the normal fastest motion by POKEing the relevant factor of speed or velocity into the velocity X and velocity Y (two memory locations listed in the customising section). Even though this will make the image seem to 'fly' along at top speed, the motion will not be as smooth as earlier, slower velocities. This is due to the fact that to make the Image go ‘faster' the machine language is moving the Image two or three pixels each move.

No other programming is necessary for basic movement. The next line is sufficient to make it move:

```
40 D=USR(-1 ) : GOTO40
```

The GOTO statement will simply repeat the USR statement continuously. The USR statement will do the rest. This is obviously a very simplified example. In more common use, you would execute the USR statement. Test for any collision with particular background colours. Test position of X, Y co-ordinates by PEEKing the X and Y memory locations. Testing for ARM or FIRE depressions and maybe branch off for some special routine for firing and such. After all the filling programming is done, you must GOTO the USR statement at the beginning of the loop and execute all the tests again.

Any branching should be in the form of a G0SUB and RETURN. These routines would ideally be given high line numbers and would be positioned outside the main loop body. The end of the routine should be completed with a RETURN statement to return back to the main loop. If these guidelines are followed it will be much easier to edit and modify the program at a later date, If necessary.

The other way of controlling the motion more directly is to position the sprite with direct co-ordinates. Everything already mentioned is still relevant, but you must now keep track of co-ordinates yourself. You must also calculate new positions all the time. The advantage of this method is that you can use formulas to calculate co-ordinates or you may add forces such as gravity, etc, to the motion of the object. To position a sprite at a co-ordinate, you simply exchange all references of D=USR(-1) to D=USR(X*256+Y). Here X and Y represent X and Y co-ordinates. The only stipulation for this method, is that X be an Integer. If you feel X may become a decimal number at some stage, Insert a X=INT(X) statement before each USR statement. This will ensure that X is converted to an Integer.

Roughly the program will look as follows:

```
10 POKE 30862,0:POKE 30863,124
20 MODE(1)
XX Background
XX Set
XX UP
30 POKE 31631,2:D=USR(-1):POKE 31631,0
40 D=USR(-1)
XX Testing of
XX Collisions,
XX Arm, Fire
XX or Position.
200 GOTO 4 0
XXX Any
XXX Branched
XXX Subroutines.
999 END
```

The final step is to disable the sprite. This may be necessary if you switch from MODE(1) to text MODE(0), or if you simply want to turn off the sprite. The following line will disable the sprite. When I say disable, I mean make the sprite disappear from the screen altogether. The position, background and motion are all still active whenever you use the USR statement though. When disabled you can use the USR statement in text mode for motion detection, which mixes keyboard and joystick Input.

```
1000 POKE 31631,3:X=USR(-1)
```

Before saving your program use the following line before the CSAVE command. There are two pokes that must be done before saving to tape. These POKEs will then save your basic program and the machine language to tape at the same time. If you POKEd and SAVEd the programs, by LISTing, you would find your program had been lost. To regain your program simply type RUN and then break out of your program. You will find it has returned.

The line to type is the following.

```
POKE 30884,233:POKE 30885,122:POKE31464,0
CSAVE" filename"
```

The first of these lines must be typed exactly otherwise the program will be totally lost and you will have to type It in from the start.

The above method is only necessary if you feel you are saving the program for the last time. If you are just saving the program between edits, just CSAVE without all the above procedure. This will just save the basic program. To load the program saved this way, the sprite utility must already be in the computer memory. The sprite utility, once loaded from an editor package, will only be lost if you turn off the computer or CLOAD any program that does not use the sprite utility. Please make sure you understand the difference between the methods of saving. The first method is more correct and the second is faster, but you must only use, the second if you intend to save and load while the sprite machine language utility is resident in memory.

Read the next section for details on specialised testing and customising. Please read all sections carefully before starting and work slowly to start with. Once you have mastered the basics you can start using different techniques. You will find this utility gives your VZ300/VZ200 much more power and freedom to program with a lot more imagination. The secret to quality programming is simplicity and hopefully this utility will promote simplicity in program design. Load, list and analyse the demo programs supplied, these may give you a few ideas.

VARIOUS TESTING TECHNIQUES

These are to be used in the major loop in your program. The major loop in your program should be the one that starts with the USR statement. Then it is filled by the following testing techniques. It must return with a GOTO, back to the start of the loop, where the USR started it. Each BASIC program should have the following elements:

(1) The initialisation of the USR statement, MODE (1) and background displays and of course, the sprite Initialisation line.

(2) The main body or continuous loop of the program. This should start with main USR statement, the repetitive programming portions such as testing and maybe a pair of INKEY$ statements for any extra user Input, and return to start of the loop. Any testing that causes branching should then G0SUB to a specialised subroutine located at the end of file program (section 4).

(3) If the program is not an endless repetition, there should be some type of conclusion or ending to the program. This may include disabling the sprite or giving some text message. This is the only unessential section of the program.

(4) This section should be positioned at the end of the program and consists of the subroutines needed to fill conditions of the previous testing. Each of these subroutines should ideally start on each thousand line number. For example, the firing routine might be located at line number 2000 and the collision routine may start at fine 3000.

This section is basically centered around helping you with the second element

```
60 M=PEEK(31628)
```

The following inputs have corresponding values:

UP.................... 1	LEFT.....................4	F IRE............................ 16
DOWN............ 2	RIGHT.................. 8	ARM............................ 32

For every direction activated, Its value is added into the motion value.

For example, If you were only going right, the value of M would be 8.
If you were also going up and pressing fire, the value of M would be equal to the addition of all these values. 8+1+16 equals 25. Therefore, the value of M would equal 25.

There is a particular term sometimes used in programming. The word is MASK. The word mask refers to highlighting particular Information and disregarding other Information. Usually In normal situations, a single number can only inform us about one certain condition.
In computer logic though, a single number may represent several conditions that are Independent of each other. Each condition may be singled out independently of the rest of the Information with special commands. The command is the BASIC 'AND' command. This command when used properly can split one single number into eight different pieces of information or conditions.

While testing for a right, you could use the test ‘IF M=8 THEN' . This test will only be true if right and only right is activated. We showed that M could equal 25 while right had been activated. The way to avoid the obvious problem is to mask out all the other directions while testing the value.

To test for the right direction, irrespective of other directions, you can mask through the right direction with the basic ‘8 AND M' command.

To test then for activation of a right, you would use the following lines:

```
70 IF (M AND 8)=8 THEN G0SUB 2000
```

The brackets in the test are important. If right is activated the program will G0SUB 2000. Just for your own proof, type 'PRINT 25 AND 8’ on your computer. You will find the answer is 8. This answer is what we are testing for. The test 'IF (M AND 8)=8 THEN' can be cut down in size to the following: - 'IF M AND 8 THEN'.

```
80 IF M AND 1 THEN POKE 32230,184
90 IF M AND 2 THEN POKE 32230,152
```

These two lines will select IMAGE #1 when going up and IMAGE #2 when going down. The following line will branch to subroutine 3000 if the FIRE is activated.

```
100 IF M AND 16 THEN G0SUB 3000
```

You may also mask more than one direction at a time with the AND command. The following examples may give you an idea.

M AND 48 ....either ARM or FIRE depressed.
(M AND 48)=48 ....both ARM and FIRE depressed.
(M AND 48)=32 ....only ARM depressed, fire Is Inactive.
(M AND 15)=4 ....out of all 'DIRECTIONS' only left activated.

Note that the mask is calculated by adding relevant condition values.

COLLISION TESTING.

To understand this fully, study the examples & descriptions, then check with the customising section for addresses and values used.

```
110 IF PEEK(31538)<>0 THEN POKE 31629,2
120 IF PEEK(31540)<>0 THEN POKE 31629,1
130 RC=PEEK(31542):IF RC<>0 THEN SOUND RC,1
```

Line 110 will test for a collision with a yellow pixel. If the value is zero, there are no yellow background pixels that have been overlayed by the sprite. If there is yellow collision, the test will prove true and the POKE 31629,2 will be executed.
This POKE will double horizontal speed of travel. This is obviously only relevant for automatic motion use.

Line 120 will change the X velocity back to normal if the sprite goes over a blue pixel.

Line 130 not only depends on collision but also on the number of pixels superimposed.
The more red pixels, the higher the pitch of the sound.

Remember the collision registers actually give the number of pixels overlayed, not just indicate collision as most collision detection systems do. The other important thing is that the sprite must not be in cumulative Collision mode.
All these collision registers will only be made good use of if a background has been laid. Background may be laid anytime and anywhere except actually on top of the sprite.
You may SET or RESET any pixel next to or distant from the sprite and these will stay. Anything actually set upon active sprite pixels will be removed as soon as the sprite is updated. By active sprite pixels I mean those that are non-transparent.

The only exception to the above rule is occasionally pixels SET near -the sprite immediately after initialisation may not stay, even if the initialisation was for a disable mode. The simple and obvious way to avoid this small problem is to lay the background before initialisation or after the second USR statement after initialisation. Background may be continuously changed during the program as long as it does not involve changes of pixels directly below an active sprite.

The co-ordinates of the sprite may be examined at any point of time, especially if you are using automatic motion mode. The following two lines will load XX and YY with the corresponding co-ordinates. You will have to obviously substitute these symbols with valid variables
E.g. X1 =PEEK(31633) .

```
XX=PEEK (31633)
YY=PEEK (31634)
```

These may be tested or used like any variable and they represent the position of the top left hand pixel of the sprite on the screen at anytime.

```
140 X =PEEK (31633)
150 IF X < 1 0 THEN G0SUB 3000
```

These two lines will test if the X co-ordinate is less then 10. If this condition proves true the program will branch off to a subroutine at 3000.

The motion of any other object can also be Included in this main body loop. To make another body move you will have to use the old basic methods of either a pair of SET and RESET commands while keeping track of position or you may use a POKE into video memory to 'set and reset' the object. A delay may also be placed before the end of the loop.

The last task to complete in this main loop is to simply go back to the start of the loop (where the USR command is located) with a goto statement.

There is one other way to make the sprite move. So far, the only method of positioning is through the automatic motion mode e.g. USR(-1).
To make this image move, you use the keyboard or joystick and activate the direction in which you want to move. This is quite sufficient for positioning an object to a desired position. Sometimes in games though, your image is not supposed to move under ideal conditions. For example, you may not want an object to go left simply because you are pressing left. If the object was going right you might want it to slow down, stop, and then go left. This method of motion is vastly different from the first direct automatic motion method. The whole concept behind this other type of motion, is the fact that you are adding realism. The way of creating more realistic motion is to use the joystick or keyboard input as acceleration and not velocity. Velocity in this form is the size of movement. Moving two pixels is the same as a velocity of 2. Using automatic motion, If a direction is active then the image is moved in that direction. To create acceleration you need to use the co-ordinate mode.

The effect of velocity is very easy to create in basic. You must set up two variables for each axis One must be position and the other must be velocity. This contrasts with the fact that before you did not need any variables as all motion was done for you. The results of the Input must then be added to the velocity. A changing velocity, by definition has acceleration, and this acceleration is simply the response to your Input.

You must then add the velocity to the old sprite position. After these two simple calculations, you can display the sprite at the new position. To add quality features, you could add a component of gravity by adding a constant amount to the Y-axis velocity each time through the loop.
You could also, at boarders reverse the velocities sign (positive and negative) as if you are bouncing off the edges. Remember that these calculations must be done for both X and Y-axis’s.

To position the sprite by co-ordinate simply exchange any reference of D=USR(-1) to the function D=USR(X*256+Y). Remember that X must be an integer and not a decimal.

The following is a small program that uses some techniques explained above. The program has been described line by line for your ease of understanding. Please type in the following program just after you have reset the computer from the editor program with the SHIFT<8> command. Design any image you want to use before resetting

```
1 0 POKE 30862,0: POKE 30863,124
20 MODE (1)
30 FOR C=2 TO 4: COLOR C,0
40 FOR I=1 TO 50
50 SET (RND (127),RND (63))
60 NEXT I: NEXT C
70 X=20: Y=20: A=0.3: G=0.2: R=0.8
80 POKE 31631,2: D=USR(X*256+Y) : POKE 31631,0
```

The first line is self-explanatory. This fine must be the first line in every basic program using the utility. The second line sets up graphics mode. Line 30 starts a loop for the last 3 colours available in the graphics mode and activates the colour with the COLOR C command. Line 40 specifies that there will be 50 dots per colour. Line 50 sets a random pixel on the screen in the current colour. Line 60 completes both loops so that all the pixels for each colour are set and each colour is chosen. Line 70 gives the Image a co-ordinate to start, sets a rate for acceleration and gravity for later use. The variable R is used for a rebound factor. A value of one gives perfect elastic collision. A value of 0.5 will retard the amount of bounce. A value of zero will give no rebound. It will give the affect of stopping when hitting the edge. Line 80 initialises the sprite. NOTE that background has been laid before Initialisation.

```
90 D=USR(INT(X)*256+Y)
100 M=PEEK(31628)
110 IF M AND 1 THEN VY=VY-A
120 IF M AND 2 THEN VY=VY+A
130 IF M AND 4 THEN VX=VX-A
140 IF M AND 8 THEN VX=VX+A
```

Line 90 is the beginning of the main loop of the program. It Is the line that positions the sprite at co-ordinates X, Y.

NOTE that the function INT(X) has been placed within the USR statement. This is Important to get accurate positioning when using decimals.

Line 100 loads M with the motion value. Line 110 to 140 tests for each direction and adjusts the velocities for each active direction.

```
150 IF X+VX > 120 THEN VX=-VX*R
160 IF X+VX < 0 THEN VX=-VX*R
170 IF Y+VY > 56 THEN VY=-VY*R
180 IF Y+VY < 0 THEN VY=-VY*R
```

These four lines 150 to 180 will test if the co-ordinates would go over the limit. If they would have gone over the limit, the velocities are reversed. This gives the affect of a rebound off the edges. NOTE these three things. Firstly, the velocities after a rebound are opposite from their direction before the rebound. Secondly, the positions are calculated and tested for limits before they are actually assigned to the co-ordinate variables. In this way you can adjust the velocity before you pass the limits. Thirdly, at the point of rebound the velocity is reduced. Realistically, everything reduces speed after collision

```
190 X =X +VX
200 Y=Y+VY
210 GOTO 90
```

These, last lines recalculate the new position using co-ordinates and velocities. NOTE velocities are changed by input but are not equal to input.

After you type this in and run it, you may modify it to include gravity. To create the illusion of gravity simply add a bit to the Y axis velocity. This is the same as giving a constant downward acceleration. In physics, gravity is a constant acceleration downwards. Simply add the line 105.

```
105 VY=VY+G
```

Experiment with the constants on line 70. Change the values of acceleration, gravity and rebound to your own fancy. Make sure acceleration is greater than gravity. Otherwise you will not be able to lift the Image.

THE CUSTOMISING SECTION WITH POKE TABLES OF ADDRESSES AND VALUES.

COLLISION COLOUR	LOW BYTE	HIGH BYTE
G R E E N	(31536)	(31537)
Y E L LO W	(31538)	(31539)
B L U E	(31540)	(31541)
R E D	(31542)	(31543)

When PEEKing these addresses, they will give the number of corresponding coloured pixels that have been overlayed. A value of zero means that that colour has not been superimposed upon by the sprite. The low byte is the only byte you have to worry about PEEKing. There are two modes of collision available. The first and standard one is the instantaneous collision mode. This is normally active and will give a count of each pixel currently being overlayed upon. The second mode is the cumulative mode. Here the count is left to add upon itself and is not continually cleared. To activate this mode, you must add four to the status register (31631). When initialising you would, for example, use the following line:

30 POKE 31631,3: D=USR(-1) : POKE 31631,4

Note: - the last POKE used the value of 4 instead of the usual value of 0. If the normal value were 2, to set cumulative mode, you would use the value 6.

The above example line would Initialise the collision registers and start the cumulative mode. To now read the collision registers you must let
N = PEEK(HIGH BYTE) * 256 + PEEK (LOW BYTE). The variable N will now give you the amount of pixels of a certain colour that the sprite has glided over since Initialisation. This may be useful if you want to know when you have gone over 80 blue dots, for example, in a game like Pacman. You could also use this as a primitive time limit or as a count of total points gained.

MOTION REGISTER:- (31628)
MOTION MASK:- (31635)

DECIMAL VALUE	32	16	8	4	2	1
THE DIRECTION	ARM	FIRE	RIGHT	LEFT	DOWN	UP
BIT POSITION	5	4	3	2	1	0

The motion register will keep the result of the joystick and keyboard status in the format shown in the above table. For every activated direction, the decimal value is added to the register. For example, if FIRE and LEFT were activated, the motion register would hold the value of 16 + 4 or 20. To test for a specific direction use the basic AND command. The following two lines of basic programming will test for left movement. If active it will G0SUB 1000.
The example will not be effected by any other directions activated at the same time.

```
M=PEEK (31628)
IF (M AND 4)=4 THEN G0SUB 1000
```

If you had used the test ' IF M = 4 THEN' Instead, only the left direction may have been activated for a true condition (any other direction activated simultaneously would give M a value other then 4. It might be equal to 20). To disable a DIRECTION, ARM or FIRE button, you must use the motion mask address. To disable the direction, simply poke a value of 255 minus the particular direction's decimal value. For example, to disable the right direction simply poke a value of 255-8.

POKE 31635,247

This facility might be useful in a game that has limited ammunition. When ammunition runs out, simply disable the FIRE key with the POKE 31635,239. You don't have to test if the ammunition has run out every time the FIRE button is pressed, because the FIRE button will not register. More than one direction may be disabled simply by subtracting all the directions disabled from 255.

MAXIMUM and MINIMUM LIMITS:

DESCRIPTION	ACTION	CONDITIONS	NORMAL VALUES
MAX X-RIGHT	POKE 31740,X	0 =< X =< 120	120
MIN X-LEFT	POKE 31741,X	0 =< X =< 120	0
MAX Y-BOTTOM	POKE 31742,Y	0 =< Y =< 56	56
MAX Y-TOP	POKE 31743,Y	0 =< Y =< 56	0

The reason these extreme limits of 120 and 56 are 8 less than the normal co-ordinate limits in hi-res. is because the image is an 8 by 8 sized object.

For example to set a bottom limit of about the middle of the screen, use the following statement :-

POKE 31742,27

To set a margin for the sprite of 10 pixels in from each edge, use this line :

POKE 31741,10 : POKE 31740,110

These limits take effect over the sprite in all conditions. If in automatic mode, the sprite will not wander past these boarders. If in co-ordinate positioning mode, and given co-ordinates that are outside the set boarders, the image will be brought back into the set range of the utility.

If at any stage you wish to know the value of X or Y co-ordinate (you may be in automatic motion mode and have no idea of the position) all you have to do is PEEK the X and Y memory locations:

X value = PEEK (31633)
Y value = PEEK (31634)

These co-ordinates represent the top left-hand pixel of the sprite.

When in automatic motion mode you will find the next two locations fairly useful. They are the velocity X and velocity Y locations. These represent the minimum step in pixels of each move in automatic motion mode. The larger the number, the faster but coarser the motion.

VELOCITY X	POKE 31629,V
VELOCITY Y	POKE 31630,V

If V equals zero no motion will occur. If V equals 1 this is normal, smooth motion. If V is greater than one, then the minimum step of each move is equal to V. this gives the effect of speed. The following example will triple the apparent speed.

POKE 31629,3 : POKE 31630,3

To return to normal, just poke the addresses back with 1.

IMAGE NUMBER

To swap the Image to and fro from basic use the following address and two values.

POKE 32320,152 this will enable the first image (sprite #1).
POKE 32320,184 this will enable the second image (sprite #2).

INPUT DEVICE.

To disable any input from the joysticks, use the POKE 31828,0.
To enable the joysticks back after being disabled, use the POKE 31828,181

IMPORTANT STATUS INFORMATION.

The status register located at (31631) controls the vital modes of the sprite. By POKEing several different control numbers Into the status, you change the overall performance of the utility. These values can't just be poked anywhere in the program. They must be executed while either Initialising the sprite, changing its characteristics, disabling the sprite or when starting cumulative collision mode. By obeying this simple rule you will save getting 'garbage', or unwanted pixels set on the screen.

POKE 31631,N

N=3 :- Disabled totally. Will not display sprite or restore background. Will keep record of position, even update position and background so that the sprite may be enabled at any time immediately after this stage.

N=2 :- This mode will only display the sprite and not restore background. It will keep track of background but just not restore it after the sprite has moved. This is great for either leaving a trail or pattern. Also during initialisation, no background should be restored.

N=1 :- This will restore the background and not redisplay the sprite. This is ideal for disabling a sprite from normal mode. When wishing to clear the sprite from the screen, simply poke a value of 1 and call the routine. This will effectively clear the image and not redisplay.

N=0 :- This Is the normal mode and you will find the sprite is usually in this state. In this mode both the background is restored and the sprite is redisplayed at its new relevant position. In this mode the sprite will only be redisplayed if you have moved from the previous position or if you are using co-ordinate mode.

N=N+4 :- If at any stage you use cumulative collision, simply add 4 to the supposed value as described above. To clear the cumulative registers either poke then with zero or call the utility when the status register is below 4, then add 4 back Into the status. At no stage should the status register be above 7. Its value will be between 0 and 3 for instantaneous collision and between 4 and 7 for cumulative collision.

At any t time when the text uses the phase that you must call the routine or utility this simply means you should execute a USR statement.

SPECIALISED MOTION CUSTOMISING.

JOYSTICKS:

The utility as it comes is set up for input from both joysticks at the same time. Both the right and left joysticks will register as the same. AIso there are two buttons on each joystick. Technically one is an ARM and the other is a FIRE button. This is how the utility will read the joystick. The fifth bit is set on FIRE and the sixth bit is set on ARM in the motion memory location. You can customise the utility so that it will read both buttons on the joystick as fire buttons. You can also specify that you only want it to read the left or right joystick but not both at the same time. The programming may switch from reading one joystick to the other fairly simply the following locations and values are given to do this. Just poke addresses with corresponding correct values. These are as follows:

JOYSTICK	1-FIRE	2-F 1 R E	ARM
BOTH	POKE 31803,42	POKE 31803,32	POKE 31812,37
RIGHT	POKE 31803,46	POKE 31803,44	POKE 31812,45
LEFT	POKE 31803,43	POKE 31803,35	POKE 31812,39

For example, if you wanted only to register the right hand joystick and use both ARM and FIRE, you would POKE 31803,46 : POKE 31812,45.

These POKEs only need be done once and only by loading in the original or POKEing back old values will you return to normal joystick status.

To disable the joysticks POKE 31828,0
To enable the joysticks POKE 31828,181

Sometimes, on some computers you will find the sprite image drifts around the screen without any help. This is due to the fact that sometimes the joystick ports give Incorrect data. False data can be minimised by reading the port several times, and mixing the values together to get a more accurate reading. There is a location that holds the number of joystick readings. For one reading of the joysticks POKE 31801,1.
This will be fine for most computers. If you find your computer gives false readings either increase this value like so**:-** POKE 31801 100. This will read the port 100 times. The maximum number of reads is 255. If you find you are still getting unreliable readings it is best if you disable the joysticks as mentioned above.

KEYBOARD:

The keyboard is set up quite straight-forwardly. Due to old SYSTEM-80 formats and ease of use <Q> and <A> have been allocated up and down respectively. <M> and <**,**> have been used for left and right, because the corresponding arrows are positioned above them on the keyboard. The ARM uses <**.**> and the FIRE uses the <SPACE> button. As with the joysticks, any characteristics may be changed. You may change any or all keyboard equivalents using the following two tables. Use the first table to find the values needed for each different keyboard button. Every key has its own set of values. Then find the addresses in the second table for the direction to be modified. Poke the corresponding addresses with values. This only needs to be done once. Once changed, it is stored in the machine Language routine. Not even NEWing the computer will bring back old values. You will have to poke the original keys back, if they are ever needed.

TABLE NUMBER ONE

	VA L U E # 2						
		110	102	94	86	78	70
V	254	R	Q	E		W	T
A	253	F	A	D	CNTL	S	G
L	251	V	Z	C	SHFT	X	B
U	247	4	1	3		2	5
E	239	M	SPC	,		**.**	N
	223	7	0	8	-	9	6
#	191	U	P	I	RETN	0	Y
1	127	J	;	K	:	L	H

TABLE NUMBER TWO

DIRECTION	ADDRESS #1	ADDRESS #2
UP	POKE 31751	POKE 31752
DOWN	POKE 31759	POKE 31760
LEFT	POKE 31767	POKE 31768
RIGHT	POKE 31775	POKE 31776
ARM	POKE 31783	POKE 31784

FIRE	POKE 31791	POKE 31792

For example, if you wanted the fire button to be the <Z> button, you would look up both tables and write down their different numbers.
Always remember you POKE the ADDRESS with the VALUES.
Now address #1 for the fire feature is 31791 and the value #1 for <Z> is 251.
Corresponding numbers for the second are 31792 and 102.

Now to change the utility, simple POKE the corresponding numbers like so

POKE 31791,251 : POKE 31792,102

S U MM A R Y

BASIC PROCEDURES.

1 - CRUN first demo program.
2 - CRUN the following editor program.
3 - Edit the sprite image and reset the computer.
4 - Start programming. Make sure you follow all necessary steps given.
5 - Poke memory down low ready to save both basic and utility.

The following line is correct for this - POKE 30884, 233 : POKE 30885,122

6 - CSAVE program onto tape.
7 - RUN and debug program slowly and carefully.

EDI TOR PROCEDURES.

1 - CRUN the second program on the tape. It is called 'SPRITE PACKAGE'.
2 - Use the joystick to edit both images.
3 - Use directions to move cursor and fire and arm buttons to set and reset pixels.   Change colours as necessary.
4 - Assemble to memory with the <Z> command.
5 - Test in hi-res. with the <X> command.
6 - Go back to step two until you are happy with both images.
7 - Reset the computer with SHIFT <8> command.

PROGRAMMING PROCEDURES.

1 - Follow the editor procedures and reset the computer.
2 - Copy the line 'POKE 30862,0 : POKE 30863,124' as the first fine.
3 - Select MODE (1) Lay any background.
4 - Initialise the sprite with 'POKE 31631,V1: D=USR(-1) : POKE 31631,V2   ' Substitute V1 and V2 for valid values.
5 - Start major loop with the USR statement.
6 - Carry out any testing here (i.e. collision, Input, position or firing).
7 - Calculate any co-ordinate Information you might need.
8 - End the major loop and return to start of loop with a GOTO.
9 - Complete any subroutines referenced within your loop.