THE

SPREDITOR

USER'S MANUAL

A CheetahSoft Product

By Dennis M. Osborn

THE

SPREDITOR

USER'S MANUAL

A CheetahSoft Product

By Dennis M. Osborn

Visit WWW.VISIONBASIC.NET for more great software!

INTRODUCTION

I've come across many a sprite editor, and I've concocted a few editors of my own, but no sprite editor I have ever used has made the sprite-making task easy for me. A couple of years ago I was working on a game that would employ quite a number of sprite shapes. But I didn't just want an editor that would ease the pain – I wanted my sprites to come to life. I wanted "dynamic sprites," sprites that would jump off the screen, and sprites that would animate gracefully, despite the crude graphical limits set deep within the heart of an eight-bit Commodore machine. I put my game on hold for just a little while – long enough to complete my sprite-editing masterpiece: The Spreditor.

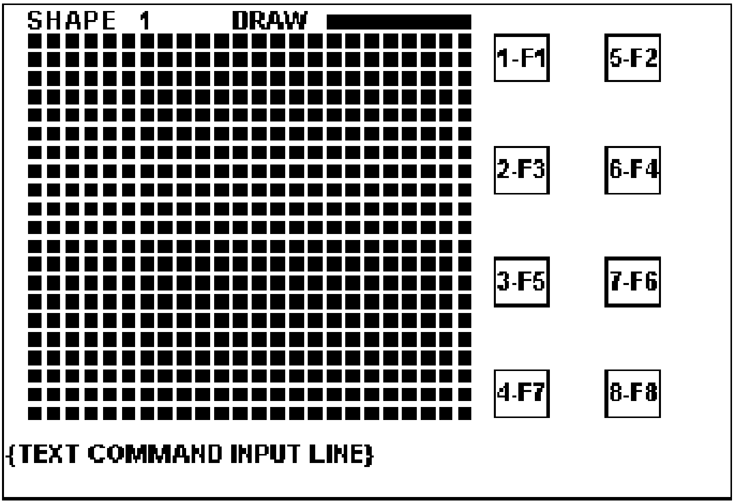
The name sounds intimidating doesn't it? I simply formed the name by tacking the first three letters of "sprite" to the word "editor" – it's a name that suits the purpose, and yet it's really much more than just an editor. For your average "Joe," the main editor is more than adequate, incorporating more editing functions than you could possibly ever need. But for those of you who are wizards at math, prepare yourself for a unique editing experience.

Okay, enough blabbering! I will attempt to explain to you, as best I can, everything there is to know on how to use The Spreditor. I suggest reading the manual from front to back and experimenting with The Spreditor along the way.

I wrote the main Spreditor program in BASIC, with added machine language subroutines, and then compiled the program with the Blitz compiler to speed it up considerably. Because of the massive size of the program, I created the hi-res editor and the multicolor editor as separate programs. There are also two versions of The Spreditor on disk that are more suited for ninety-degree sprite rotations. To load any version of The Spreditor, simply turn on your Commodore 64 computer system, insert The Spreditor diskette into your number eight disk drive, and type in the usual LOAD“*”,8,1 and press **RETURN**. If you decide to load The Spreditor without turning your computer off and then on again, you may have to type LOAD“BOOT”,8,1 instead. In a short time you will see a menu pop up, listing the different versions of The Spreditor next to the keys you will need to press to access each version (for now I recommend selecting one of the non-rotational versions). After you have made your selection, the version you have chosen will be loaded into memory. Soon The Spreditor’s main shape-editing screen will pop into view.

THE EDITOR

As with most sprite editors, a large grid will appear on which all sprite editing takes place. The number of the shape currently being edited is displayed just above the grid, along with the current drawing color. These drawing colors are just for reference and may not match the colors used on the actual sprites themselves.



To the right of the grid are two columns of sprites – all eight of them – some of which may not appear visible due to content or coloring. All multicolored sprites share two plotting colors, but each sprite may be displayed in its own unique color, resolution, X and Y expansion, and sprite shape.

When The Spreditor is first run, each sprite to the right of the grid is assigned a sprite shape from the 128 total shapes available. The initial shapes chosen are the first eight (1-8) in memory. When you are editing a sprite shape, any sprites designated to that shape will change instantly as editing takes place.

The bottom row of the screen is a line reserved for user text commands. Most editing will be done by single key presses, but sometimes you will need to type in a command for a particular function. Let's start with the key presses.

On the grid you will see a flashing cursor. To move it, simply use the cursor keys as they are normally used or use a joystick plugged into port #2. If you find that the cursor moves too fast for you, you may press * to slow it down more and more, and shifting this key will bring the speed back up to normal. In order to draw using the current plotting color, you must press the joystick button as you move around – drawing "free style" can only be done with the joystick. To change the plotting color, simply use the + and – keys to cycle through the color choices.

You may have up to 128 sprite shapes in memory at any given time. To cycle through the different shapes – on the editing grid – press the space bar to advance and shift the space bar to go back. To clear a shape that you wish to start working on, press the CLR key (the home portion of this key brings the cursor back to the home position, in the upper left-hand corner).

If there's one thing that makes sprite shape creation a royal pain in the derrière (excuse my French) is when you have to fill a "large" area of a sprite shape solid. To make this job so much easier, there is a "fill" function that you can access at any time by pressing the ← key. The filling will begin at the cursor, and the surrounding pixels (that are the same color as the pixel underneath the cursor) will be filled in with the current drawing color. The fill will even wander through a maze of pixels until the job is complete.

Should you need to shift the image around on the grid, you may do so with the numeric keys 1 through 4. Any pixels pushed off the edge of the shape will wrap around to the other side. Use 1 to shift up, 2 to shift down, 3 to shift left, and 4 to shift right. Pressing 5 will flip the shape vertically, pressing 6 will flip the shape horizontally, and pressing 7 will rotate the top left portion of the sprite shape clockwise at ninety-degree intervals. In the multicolor editor, the top twelve by twelve pixels will be rotated (the resulting image will appear somewhat distorted for multicolor rotations). In the hi-res editor, a 21 by 21 pixel area will be rotated, which is most of the sprite shape. If you wish to do extensive rotation work on your sprite shapes, you may want to load up one of the rotational editors, which reduce the grid size to make rotation work simpler.

The Spreditor has a shape buffer that can come in handy from time to time. To store a sprite shape to the buffer press the 8 key. To retrieve a shape being held in the buffer, press the 9 key. A buffered shape can be easily copied to other shape slots with only a few key presses. For example, if you wish to copy the current shape to the next seven shape slots, simply press 8 and then press both the space bar and the 9 key seven times each and alternating. Pressing the 0 key will animate sprite #1 using the range of shapes chosen during RUN parameter entry (the RUN command will be explained later). For now, leave the 0 key alone.

If you press the INST key (shifted DEL), The Spreditor will wait for you to press yet another key before taking action. If you press X, all pixels from the cursor to the right edge of the sprite grid will be pushed horizontally to the right by one pixel. All pixels running vertically along the right edge of the sprite grid will disappear. The vertical “space” that is opened up will not be cleared, but instead will mimic the vertical column of pixels just to the right of it. If you press Y, all pixels from the cursor to the bottom of the sprite grid will be pushed down by one horizontal line of pixels. Think of using X for horizontal "pushing" and of using Y for vertical pushing. The DEL key works much the same way, but "pulls" instead of pushes. Pixels at the cursor will be deleted, and a space will open up at the edge of the sprite grid. Try testing these key presses on a test shape to better understand how they work. Since these functions first buffer the current shape before altering it, you may "undo" a single insertion or deletion by pressing the 9 key.

If you want to reverse your shape, press the £ key. To reverse it back, press £ again. This works best in the hi-res editor. For the multicolor editor, colors 0 and 3 (binary 00 and 11) will be swapped and colors 1 and 2 (binary 01 and 10) will be swapped.

If you press keys A through Z, you will be plotting reference points on your grid. Several user-typed text commands make reference to these points (mostly the program commands). Let me give you a little taste of how these reference points can be used. Move the cursor to the home position (upper left-hand corner) and press the A key. You will see the letter "A" appear. Now go from corner to adjacent corner and place "B", "C", and "D" in the remaining corners, in order. Next, press RETURN (this places you in "text command mode") and type "LINE A-B-C-D-A" and press RETURN again. If you placed the letters in the right corners you should see a box form before your eyes – make sure the shape grid has been cleared and that the drawing color is anything but the "background" color.

You can move any of the points by simply placing the cursor to a new location and pressing the letter of the reference point you wish to relocate. If you wish to remove a letter from the grid, move the cursor on top of that letter and type the same letter on the keyboard. Letters can overlap each other, but only one will be visible. All 26 letters may be used, but I don't think you'll need them all.

As you now know, pressing **RETURN** brings you into text command mode, and the text commands will be explained shortly, but if you press **SHIFT** - **RETURN**, the last text command used will reappear. You can then make any desired changes to the command and press **RETURN** again for execution.

If you wish to leave The Spreditor at any time, just press the **SHIFT** and **RUN/STOP** keys. The main Spreditor version choice menu will load up, as long as The Spreditor disk is still in the drive – if not, an error message will appear and control will go back to the BASIC editor. Your shapes will remain undisturbed in memory as you hop from version to version of The Spreditor.

Finally, pressing any function key allows you to enter "sprite control mode." Each function key corresponds to a different sprite. While you are in sprite control mode, pressing a *different* function key allows you to shift control over to a different sprite.

Sprite control mode has its own set of key presses. As mentioned above, each function key will give you control over a different sprite – there are eight function key presses and also eight sprites, giving you control over all eight sprites. You can move your selected sprite around by pressing the cursor keys or by moving the joystick. The current sprite being manipulated is indicated at the top of the sprite manipulation area, as well as its shape. To change the sprite shape to any one of the 128 shapes in memory, just press the space bar to advance, and shift the space bar to go back. The asterisk key works the same as it does for editing by changing the speed of movement.

Now for the following key presses, any changes you make to a sprite will be *adhered* to the sprite shape as well, thus changing the appearance of *all* sprites set to that particular shape. Pressing **X** turns horizontal expansion on and off, pressing **Y** turns vertical expansion on and off, and pressing **M** turns multicolor mode on and off – for the sprite in question. The **+** and **–** keys change the sprite's main color. The word "SHAPE" at the top of the screen is displayed in this color, so if your sprite does not contain pixels in the main color, you will still know what the main color is. The last key in the list is the **RETURN** key, which returns you back to editor control.

The benefit of being able to move the sprites around is that you can group them together to form "larger" sprites, or you can overlap them for more colorful sprite combos (this is especially useful for hi-res sprites, which have only one plotting color).

That's basically it for Spreditor key presses. Listings in the back of this manual provide a quick reference should you forget. There aren't too many keys left on the keyboard that actually don't do anything, so the reference charts should prove handy when you work with The Spreditor.

NOTE: Although The Spreditor was written without prior knowledge of Creative Micro Design's SuperCPU™, I have added a key to toggle between the SuperCPU's 1 MHz and 20 MHz modes – press the **@** *key to toggle.*

TEXT COMMANDS

When you press **RETURN** in editing mode, you will open up a whole slew of extra commands at your disposal. You can type each command as is, or you can even type a single letter instead for several commands. For many of the commands, you may omit one or more parameters. The Spreditor simply selects a default value for parameters that have been omitted (make sure, though, to still type in any indicated punctuation). For example, “SET 8-128**:**4,M” can be shortened to “SET 8-**:**4,M”. The dash and the comma in this example *can* be left out safely, but for the sake of safety, I recommend leaving the punctuation in place for *all* of the commands.

BUFFER, BUF, or B

This command buffers the current shape and works the same as pressing the **8** key (use whichever is most convenient for you).

FETCH, FET, or F

This command fetches the shape currently being held in the buffer and works the same as pressing the **9** key.

CLR

FORMAT: CLR
or
FORMAT: CLR *start-end,byte*

This command “clears” the sprites from *start* to *end* using the specified *byte.* Entering no parameters at all clears all sprite shapes from the current shape to the last shape in memory (128). This is useful when you have loaded up some shapes into memory and wish to add to them – you can quickly clear all shapes at the end of memory by choosing the current shape just after the last loaded shape and then by entering “CLR” at the bottom of the screen. Leaving out the *start* parameter sets the starting point to the current shape. Leaving out the *end* parameter sets the ending point to the last shape in memory. Leaving the *byte* parameter out sets the byte value to zero.

COPY or COP

FORMAT: COPY *start,new,count*

This command copies the shapes starting from *start* to the area of shapes starting at *new.* The amount of shapes copied is specified by the *count.* Only copy overlapping areas if you are copying from a higher shape number to a lower shape number. Any parameters left out will be set to a value of 1.

LINE or L

FORMAT: LINE *letter-letter...*

This command draws lines between lettered reference points using the current plotting color. You can follow this command with many reference points if you wish, separating the letters with dashes. Here are some examples of how to use the command:

```
LINE A-B
LINE A-B-C
LINE A-L  B-M
```

The first example connects points A and B with a line. The second example connects points A and B with a line and connects points B and C with a line. The last example connects points A and L with a line and connects points B and M with a line (points L and B are *not* connected).

SAVE

FORMAT: SAVE *start-end,filename{,address}*
or
FORMAT: SAVE *shape,filename{,address}*
or
FORMAT: SAVE *filename{,address}*

This command saves the specified range of sprite shapes to disk. Leaving out the *start* parameter defaults to the first shape. Leaving out the *end* parameter defaults to the last shape. Leaving out the dash (and thus the *end* parameter) causes only the one specified shape to be saved to disk. Leaving out ***both*** the *start* and *end* parameters causes all 128 shapes to be saved to disk. The *filename* can contain letters and numbers only and should begin with a letter – spaces are deleted. If you specify a loading *address* (the address you want the shapes to load at for your own programs), the filename will be tagged with a "MOB" extension. A filename of "CRASH" would appear as "CRASH.MOB" in your disk directory. **Do not type quotes around your filename!** If a loading address is not specified, then the filename will be tagged with a "DAT" extension instead. The resulting file cannot be loaded into memory with the BASIC "LOAD" command from your programs since a loading address is not saved with the file. Any program that accesses files such as these must load them into memory byte by byte.

When your shape data is first saved to disk, it is saved under the filename of "FILE.MOB". When the save is completed, your old file – if there is a previous version – will be deleted and your new file will be renamed to your chosen filename (from the temporary name "FILE.MOB"). This protects your old file data until the new data has finished saving. Just make sure there is always ample room on disk before you start to save any new data.

A feature of The Spreditor not found in other sprite programs is the saving of sprite properties along with the shape data. In sprite control mode you can change the sprite X and Y expansion, color, and color mode (multicolor or hi-res). These properties are saved along with sprite shape data in a manner that is virtually invisible. Each sprite shape occupies 63 bytes of memory, and each shape is separated by an extra byte in between. The Spreditor uses this extra byte to store property information for the shape just before the byte. The two colors shared by all multicolor sprites are tagged onto the very end, so keep this in mind when you load the data into memory from your own programs. The property data should be virtually invisible and should not pose any problems when you load sprite shapes for your own use. When you load sprite shapes created by The Spreditor back into The Spreditor, these properties will reinstate themselves. This is a tremendous benefit to the designing process and can also benefit those to whom you give your shapes to use as well.

For both the SAVE and LOAD commands, the screen will temporarily blank during all disk access. It appears that leaving the screen display visible sometimes interferes with drive communications.

LOAD

FORMAT: LOAD *start,filename**
or
FORMAT: LOAD *filename**

This command loads a variety of sprite files from disk into memory. Specifying a *start*ing shape is optional – if no *start*ing shape is specified, then the shapes will load in starting at the current shape being edited. If you are loading any files other than those ending with the "DAT" extension, add an asterisk (*) to the end of the filename. Files not tagged with "DAT" will be assumed to have a loading address (which will be ignored and tossed away when loaded). Spreditor files normally are tagged with the "MOB" extension. Other sprite shape files will probably be tagged with "SPR", some other extension, or none at all. When you type the asterisk after a filename, The Spreditor actually uses the asterisk when it loads your file, so be careful not to use similar filenames (ex.: "FROG.MOB" and "FROG.SPR").

Since non-Spreditor files will not contain sprite property information, the shapes loaded will probably appear invisible against the background because they will probably be black – you can use the SET command to make them visible by changing their color. You may also have to rename non-Spreditor sprite files in order for them to load properly – for example, you might have two files on disk titled "BLUE CARS.SPR" and "BLUE TRUCKS.SPR". You can not access *both* of these files as named.

DIR or D

This command allows access to the directory listing of files on disk all tagged with "DAT," "MOB," or "SPR" extensions.

DEV

FORMAT: DEV *number*

If you want to use a disk drive with a number other than eight, you can do so by implementing this command. The device number range is 8 to 11.

SET

FORMAT: SET *start-end***:***col,MXY*

This command sets the sprite properties for the given range of shapes. This command is especially useful right after you load shape files not created using The Spreditor. Following the colon, you first must specify the sprite color (0-15). If none is given, then a color used on the editing grid will be chosen (the grid color that represents the main sprite color). Following the color, you can select multicolor, X-expansion, and Y-expansion by simply using the letters "M," "X," and "Y" respectively. **Do not use commas to separate the letters.** Using no letters at all defaults the sprite properties to the current editor's own resolution with no expansions. Here's an example that "sets" shapes 1 through 10 to the color blue, multicolor resolution, and vertical expansion:

```
SET 1-10:6,MY
```

Using a letter more than once will reverse the condition of the property associated with that letter. For instance, to set sprites to hi-res you *could* use "MM." If you are going to start creating sprite shapes from scratch, then I suggest using this command before you begin creating any shapes.

ANIM or A

FORMAT: ANIM *num,start-end*
or
FORMAT: ANIM *num,start-end,start-end,...*

EXAMPLES: ANIM 1,16-24 and ANIM 4,1-31,32-2

This command allows you to animate up to eight sprites using a selected series of shapes. The first parameter specifies the *num*ber of sprites you wish to animate. If you select "1," then only the first sprite will animate. If you select “2,” then the first two sprites will animate. If you select “8,” then *all* eight sprites will animate. The same series of shapes will be used for *all* sprites chosen for animation.

Your animation series may contain up to 128 shapes, but you are not limited to using each shape only one time. In the first example, the first sprite will animate through shapes sixteen to twenty-four over and over again, but will only cycle forward. The second example will animate the first four sprites with shapes one through thirty-one, cycling forward, and shapes thirty-two through two, cycling backwards. Your only limitations are a series of 128 shapes and however many ranges of shapes you can fit on a single line of text input. For example, a command like

```
ANIM 8,1-16,15-2,1-32,31-2,64-79,84-99
```

is perfectly legal. The animation will cycle endlessly through these shapes until **RETURN** or the **RUN/STOP** key is pressed. Use the asterisk key, as mentioned earlier, as a means of speed control. All *start*ing and *end*ing parameters default to “1” when omitted.

This command will come in handy when you wish to view an animation of shapes that you have “programmed” yourself. By the way, pressing the **0** key in editing mode actually causes the ANIM command to execute using parameters set during a PROGram RUN (these two commands will be explained shortly).

COL or C

FORMAT: COL *share1,share2*

This command allows you to set the two colors that are shared by *all* multicolored sprite shapes. If no values follow the command, then grid colors are used as defaults.

GRID or G

FORMAT: GRID *clear,share1,main,share2,cursor*
or
FORMAT: GRID *clear,main,cursor* *{for hi-res editors}*

This command allows you to change the editing grid colors. All ranges are 0-15, and all defaults go to black, except for the cursor, which defaults to white, should you omit any parameters. Here is a description of each parameter:

clear – This color represents clear pixel data.
share1 – Shared color 1 for multicolor sprites.
main – The main sprite color.
share2 – Shared color 2 for multicolor sprites.
cursor – The cursor color.

SCR or S

FORMAT: SCR *border,background*

This command allows you to change the screen colors. All defaults are black when no parameters are specified. **Note: Be careful – there is no way to change the text colors.**

PROG and RUN

These commands will be explained in the following sections.

PROGRAMMING MODE

PROG or P

This command requires no parameters – it enters you into “shape programming mode.” One of the most outstanding capabilities of The Spreditor is that it allows you to enter (and run) programs that aid greatly to the sprite shape creation process. A great deal of the time sprite shapes are designed to animate, and sprites will cycle through those shapes to give the illusion of movement (the pixels themselves do not actually move). Designing a series of animating shapes can be an arduous task and the end result can often be disappointing to say the least. Programs written for use with The Spreditor use math in the aid of movement, and so movement within the sprite becomes more “exact.” Sometimes the only difference between two shapes may rest in a single pixel, but such a seemingly insignificant change may be crucial to an impressive end result. Writing these programs will give you access to almost unlimited possibilities. Your sprites will animate more smoothly than ever possible with your own hand designs. Remember those reference points we talked about earlier? Here is where their importance is made utterly clear.

In shape programming mode you will see four areas of text entry – each area has the word “program” written above it followed by a number. For shape generation, you can write up to four programs that will be run in order to basically achieve *one* result. You may type in text on any of the sixteen lines available and you may cursor around if you like. Each program can be up to four lines of text in length and you don't use line numbers as you would in BASIC – execution order is simply left to right and then top to bottom.

Commands in shape programming mode start as a single character (a symbol, letter, or number) and are often followed by one or more parameters. Two or more command statements on the same line must be separated with colons – just like in BASIC. The following is a list of the commands available and how they are used:

Period Command (.)

FORMAT: **.**{letter}{letter}{letter...}
or
FORMAT: **.**{letter}**-**{letter}**-**{letter...}

EXAMPLES: **.AFZK** and **.R-S-T**

The "period command" allows you to plot dots or draw lines on a sprite shape using reference letters placed on the sprite grid. The first example given plots dots at points A, F, Z, and K. The second example draws a line between R and S, and between S and T. You may use as many letters as space permits on a line.

Number Sign Command (#)

FORMATS: # or #{number} or {number}

EXAMPLES: **#** and **#2** and **3**

The number sign command allows you to change the current plotting color. If the symbol stands alone, then the color is advanced to the next one in sequence. If a number follows or if a number stands alone with no preceding symbol, then the plotting color is changed to the indicated value (0-3).

At Command (@)

FORMAT: @{letter}-{instruction & value...}

EXAMPLE: **@K-PU3R2M0D1L1M1P**

This command allows you to draw a complex design at the specified point. The example above basically reads: At point K, plot a dot, move up three pixels and plotting, move right two pixels and plotting, turn off plotting mode, move down one pixel *without* plotting, move left one pixel *without* plotting, turn plotting mode back on, and plot a pixel. The letters function as follows:

P	Plots dot at current location. No following parameter.
M#	Changes plotting mode on or off. A “zero” parameter turns off plotting mode so that moves can be made without plotting. A “one” parameter turns on plotting so that a pixel will be lit *after* a single advance.
U#	Advances the location up specified number of pixels. If plotting mode is on, pixels will be lit *after* each advance.
D#	Like U#, but movement is downward.
L#	Like U#, but movement is left.
R#	Like U#, but movement is right.
C#	Changes the plotting color. By itself, it changes the plotting color to the next color in sequence. If followed by a value in the range of zero to three, the plotting color will be changed to the specified value.
F	Begins a fill at the current location with the current color.

Any omitted values default to a “1”. Also, for this command you *may* use a comma in place of the hyphen if you wish.

Slash Command (/)

FORMAT: /{angle}

EXAMPLES: **/30** and **/-45**

When you RUN a shape-generating program with The Spreditor, you are asked at some point (before execution) to specify a starting and ending angle, which are used primarily for animations involving rotations of some sort. Now let's say you have a ten-shape animation you are working on and you wish to have something rotating from shape to shape (or frame to frame if you will). For instance, it could be an object with a propeller of some kind. You decide to set your starting angle to zero and your ending angle to 180 degrees (since a propeller usually has *two* blades, we only need to rotate the blade half of the way to give an illusion of a total 360-degree rotation). The angle for your first shape will, of course, start at zero. The angle for your second shape will be eighteen. The angle for your third shape will be thirty-six, and eighteen will be added again and again for each shape in succession. A variable keeps track of changes in the angle value from frame to frame of animation. Well, within your shape-generating program, you can add to or subtract from this variable at any time by using the "slash command" followed by an angular value, positive or negative.

This command is best utilized for generating "polar" coordinates. I *could* write an entire supplemental manual explaining polar coordinates to detail, but I could also write manuals for every little detail not fully understood by every single user – basically, if you have an idea of what polar coordinates are, then you can do more with The Spreditor than everyone else.

A Polar coordinate consists of an angle and a radius measurement. To convert a polar coordinate to an X and Y coordinate, merely reposition the angle-tracking variable to the desired slope (by using the slash command) and then use *format#2* letter commands to perform the transformation from polar to X,Y. You will probably want to bring things "full circle" by returning the angle-tracking variable back to its initial position when you are done. I have included program examples in the back of the manual that demonstrate the use of the slash command.

Letter Commands (A-Z)

FORMAT #1: {letter}{X or Y}{± value}

EXAMPLE: **AX+1**

There are several formats for "letter commands." This first format allows you to move a coordinate horizontally or vertically. The above example causes reference point A to be moved to the right by a single pixel. It could be read as:

$A_x = A_x + 1$.

Positive values generate movement in a right or downward direction. Negative values generate movement in a left or upward direction. Using letter commands causes permanent changes to the positioning of reference points – that is, while a program is being RUN. The value you use can be positive or negative, and does not have to be a whole number. Use "X" for horizontal movement and use "Y" for vertical movement. For diagonal movement you will have to use two separate commands – one for the horizontal movement of a reference point and one for the vertical movement of that same reference point. Reference points *can* be moved beyond the limits of the grid without causing an error.

FORMAT #2: {letter}{X or Y}{letter}{S or C}{± value}

EXAMPLES: **AXBC10** and **BYCS-5**

This format moves a reference point by a certain distance, specified by a trigonometric function, away from another reference point. The trigonometric functions use the angle-tracking variable's current value for the shape in question. Our first example above can be expanded, for our own personal understanding, to:

$A_x = B_x + \cos(\textit{current angle}) \times 1\text{Ø}.$

Our second example can be expanded to:

$B_y = C_y - \sin(\textit{current angle}) \times 5.$

If you're not familiar with trigonometry, you might just want to use the examples in the back of the book to get by on.

The positioning of each letter defines what it represents to the command. The first letter *must* be a reference point, the second letter *must* be "X" or "Y", the third letter *must* be a "referential" reference point, and the last letter *must* be "C" or "S" (for cosine and sine) followed by a multiplicative value. The plus sign may be omitted. This positioning sticks for even confusing examples like XYYC4 or SXCC9. For your own sanity, try to avoid this kind of confusion by not using X, Y, C, or S for reference points.

This format works great for drawing circles, moving pixels in a circular fashion, and for creating "polar" coordinates.

FORMAT #3: {letter}={letter}{X or Y}{± value}

EXAMPLE: **R=TX-2**

This format works a little like the first format. It passes the X *and* Y coordinates of the reference point following the equal sign to the reference point preceding the equal sign, and then a value is added to either the X *or* the Y coordinate of the reference point preceding the equal sign. The example given can be expanded (for better understanding) to:

$R_y = T_y$ and $R_x = T_x - 2.$

If you use a reference point in the beginning of a "letter command" that has not been placed yet on the grid, don't worry – in the last two formats a "new" reference point is created using an already-existing reference point. When your program is finished running, all reference points will be moved back to their initial positions.

The Question Mark (?)

FORMAT: {...number}**?**

EXAMPLE: **DX+4?**

The question mark is not actually a command, but in The Spreditor it *does* have a very important function. If you type a question mark after any numerical parameter, The Spreditor will generate a random number from zero to that number and use the random number instead. In the above example, a random value from zero to four will be used in place of the original "four" value.

The NEW command

You may type this command on the text command input line to quickly erase all four programs in memory. **There is no way to undo this command once implemented!**

RUNNING A PROGRAM

RUN or R

Once you have created the program that will generate detail on your sprite shapes, enter text command entry mode (by pressing **RETURN** in the shape editor) and type in “RUN” (or “R”) and press **RETURN**. You will be prompted to enter some parameters before your program is executed.

First, you will be asked to specify a “**SOURCE SHAPE**.” You may cycle through the four choices available by pressing the spacebar. The first option is “clear,” which causes each shape to be totally erased before any changes to the shape can be made. The next option is “same,” which allows you to “add to” the existing shape. This works great for making changes to your shapes in *layers* (I recommend saving your shapes regularly to disk, though, before implementing new changes or additions). The third option is “previous,” which uses a previous shape in your sequence of shapes as the shape to begin with and add on to. Each shape in succession builds on a previous shape. The final option is “buffer,” which uses the contents of the sprite shape buffer as the basis to start each shape out with. When your choice is made, press **RETURN**.

Next you will be prompted to specify a “**LOW LIMIT**” for your sequence of shapes. If you wish to use your program to generate detail for shapes *five* through *ten*, the low limit must be set to “5.” Use the **+** and **–** keys to change this value (and to change the rest of the parameters to come). Again, press **RETURN** when done.

The "**HIGH LIMIT**" is next, which would be shape number "10" if you use the example above (Note: These "limit" parameters are used when you press the 0 key in the shape editor).

The fourth prompt asks you to specify a "**STARTING SHAPE**" within the limits you have previously set. If you are layering on changes one "RUN" at a time, you may need to change the starting shape with each RUN of your program. This starting shape will be the first shape in which any changes or additions will be made.

The next prompt asks for the "**DIRECTION**" your program will sequence through your chosen range of shapes. Select "1" for forward advancement, select "-1" for backward advancement, and select "0" for no advancement at all. If you choose zero as your "direction," your program will consistently make changes or additions to only the "starting shape" you have chosen. As an example for selecting zero, your program may only be designed to generate a circle for just one of your shapes, so in this case zero would be the setting you must choose. And when the shape is advanced beyond the chosen limits, the program begins next with the shape on the opposite side of your limit range (basically a wraparound).

For the next four prompts, you must enter the number of cycles you want each of your programs to run. For most applications, you will probably be creating (and running) one program at a time to generate detail for a selected range of shapes. So for "**PROG1 CYCLES**" you will probably be entering a value equal to the number of shapes you have limited yourself to earlier in the game. The value would then be calculated by:

PROG1 CYCLES = HIGH LIMIT - LOW LIMIT + 1

For a shape range of one to thirty-two, you would need to run your program a total of thirty-two cycles to generate detail for all thirty-two shapes in your selected range.

Sometimes you may wish to run more than one program. An example for this might include animating a small ball of pixels from corner to corner in a sequence of sprite shapes. Well, your first program can run the ball from the upper left-hand corner to the upper right-hand corner; your second program can run the ball from the upper right-hand corner to the lower right-hand corner; your third program can run the ball from the lower right-hand corner to the lower left-hand corner; and your fourth and final program can run the ball from the lower left-hand corner to the upper left-hand corner. You would then need to know how many total shapes your animation will require, and how many cycles to run each program for.

After the program cycle settings are out of the way, you will be asked to specify a "**STARTING ANGLE**" and "**ENDING ANGLE**" as mentioned in earlier text. You can leave these alone if your program isn't using any letter commands that involve trigonometric functions. As each program cycle is ended, the variable that keeps track of the angle for the trigonometric functions is advanced closer and closer to the ending angle – the amount of advancement for each cycle will depend on the *total* number of cycles selected for all four programs. If you are rotating an object (in a complete circle) within your selected range of shapes, you would probably select zero for your starting angle and 360 for your ending angle. The range for these angles can be anywhere from zero to 720 degrees, and the ending angle does not have to be greater than the starting angle. The zero degree starts on the rightmost "side" of an imaginary circular protractor and the degrees increment in a clockwise fashion.

Finally, you will be prompted to select the color that will start things off. The range for this parameter is 0-3, with zero being the background color.

Now your program will begin to actually RUN, and all of your hard work will come to fruition. Detail will be generated one shape at a time. Each program line will be displayed at the bottom of the screen as it is being executed, and all changes to each shape will occur before your very eyes. Once the process is completed you will see the flashing cursor reappear. To instantly view the results of an animation sequence you have just created, press **0**. To change the speed of the animation (which will appear on sprite #1) use the ***** key, shifted or non-shifted. To stop the animation, press the **RETURN** or **RUN/STOP** keys. If your project needs more work, go back and make the necessary changes and RUN your program(s) again. You won't need to reset your parameters – they are "remembered" as you last entered them. You can quickly skip through all parameter entries by keeping the **RETURN** key pressed.

To halt the RUNning process at any given time, press **SHIFT** - **RETURN** or press the **RUN/STOP** key. And as a note, your program(s) will *not* run unless you specify *at least one cycle* for one of the programs.

APPENDIX A

The Spreditor Editing Keys

CLR/HOME	Clear the current shape / Home the cursor.
SPACEBAR	Advance to next shape (shift to go back).
CRSR	Move cursor on shape grid.
+ and **–**	Change plotting color.
1 **2** **3** **4**	Shift shape up, down, left, and right.
5	Flip shape vertically.
6	Flip shape horizontally.
7	Rotate shape clockwise 90°.
8	Store shape to buffer.
9	Fetch shape from buffer.
0	Animate sprite #1 using the range set during RUN parameter entry.
*****	Slow down the cursor movement (shift to speed it up).
@	Toggle between 1Mhz and 20Mhz modes (if available).
£	Reverse shape.
←	Fill area with current plotting color.
INST/DEL	Insert/delete line (press “X” for horizontal, and press “Y” for vertical).
RETURN	Text command entry mode (shift for last cmd.)

A thru **Z** Plot a reference point (or “un-plot” if on letter).

F1 thru **F8** Enter Sprite Control Mode.

SHIFT - **RUN/STOP** Exit the current editor.

In addition to the cursor keys, the joystick in port #2 also moves the cursor around the grid and the button plots pixels.

APPENDIX B

Sprite Control Mode: Key Presses

F1 thru **F8** Select a sprite to control.

CRSR Move the selected sprite.

SPACEBAR Advance to next shape (shift to go back).

+ and **–** Change selected sprite’s color.

***** Slow down sprite movement (shift to speed it up).

X X-expand on/off for selected sprite and shape.

Y Y-expand on/off for selected sprite and shape.

M Multicolor mode on/off for selected sprite and shape.

RETURN Exit Sprite Control Mode.

In addition to the cursor keys, the joystick in port #2 also moves the selected sprite around the screen.

APPENDIX C

Shape Programming Examples

I have provided the following "shape programs" as examples to help you better understand the process of programming shapes. Type in any program you wish, enter the given parameters when you are ready to RUN, and watch the results on your monitor. When coordinates are specified, you must place them on the grid yourself – the upper left-hand corner is coordinate (0,0). Unless indicated otherwise, use the hi-res editor for these examples.

TWINKLING STAR

This example uses simple arithmetic to create an impressive shape animation. You will need to plot coordinates A(0,0), B(11,10), C(23,0), D(12,10), E(23,20), F(12,10), G(0,20) and H(11,10). Notice that B and H, and F and D will overlap.

```
PROGRAM 1
.A-B-C-D-E-F-G-H-A
AX.5:AY.4:BY-.4:CX-.5:CY.4:DX.5
EX-.5:EY-.4:FY.4:GX.5:GY-.4:HX-.5

SOURCE SHAPE:  CLEAR
LOW LIMIT:  1
HIGH LIMIT:  24
STARTING SHAPE:  1
DIRECTION: 1
PROG1 CYCLES:  24
STARTING ANGLE:  0
ENDING ANGLE:  0
COLOR:  1
```

Use **ANIM 1,1-24,23-2** to animate when finished.

MESH MAKER

This program demonstrates the use of multiple programs to generate a series of sprite shapes. It also demonstrates the use of the "**PREVIOUS**" source shape setting. Clear the first sprite shape and place A(0,0) and B(20,0) on the grid before RUNning. Pay close attention to the differences between the four programs when typing them in.

PROGRAM 1
```
.A-B:AX+4:BY+4
```

PROGRAM 2
```
.A-B:AY+4:BX-4
```

PROGRAM 3
```
.A-B:AX-4:BY-4
```

PROGRAM 4
```
.A-B:AY-4:BX+4
```

```
SOURCE SHAPE:  PREVIOUS
LOW LIMIT:  1
HIGH LIMIT:  20
STARTING SHAPE:  1
DIRECTION: 1
PROG1 CYCLES:  5
PROG2 CYCLES:  5
PROG3 CYCLES:  5
PROG4 CYCLES:  5
STARTING ANGLE:  0
ENDING ANGLE:  0
COLOR:  1
```

Press **0** to animate when finished.

BOXED IN

This program demonstrates the use of random numbers in the generation of sprite shapes. Although the need for generating random numbers is not altogether frequent, it's nice to know that the capacity is there. You will need to clear the first sprite shape and place A(0,0) on the grid. **Use the multicolor editor for this example!**

```
PROGRAM 1
@A-M0R9?D17?M1C1PD3RU3RD3
@A-M0R9?D17?M1C2PD3RU3RD3
@A-M0R9?D17?M1C3PD3RU3RD3
```

SOURCE SHAPE: PREVIOUS
LOW LIMIT: 1
HIGH LIMIT: 16
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 16
STARTING ANGLE: 0
ENDING ANGLE: 0
COLOR: 1

Use **ANIM 1,1-16,15-2** to animate when finished

CIRCLE FORMULA

You can use the following program to draw a variety of circles and ellipses on your sprite shapes when such shapes are needed. In the program, the value of eleven is the horizontal radius, and the value of ten is the vertical radius. For this example, you will need to place D(11,10) on the grid and you might want to clear the first sprite shape.

```
PROGRAM 1
AXDC11:AYDS10:/10
BXDC11:BYDS10:.A-B
```

SOURCE SHAPE: SAME
LOW LIMIT: 1
HIGH LIMIT: 1
STARTING SHAPE: 1
DIRECTION: 0
PROG1 CYCLES: 36
STARTING ANGLE: 0
ENDING ANGLE: 0
COLOR: 1

COPTER BLADES, TOP VIEW

Here we are going to create some spinning copter blades as seen from a top view. Since our copter will have two blades, we will only need to spin them a total of 180 degrees. Place D(11,10) on the grid.

PROGRAM 1

```
AXDC11:AYDS10:BXDC-11:BYDS-10:/6:.A-B
@D-U1R1D2L2U2R1
```

SOURCE SHAPE: CLEAR
LOW LIMIT: 1
HIGH LIMIT: 30
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 30
STARTING ANGLE: 0
ENDING ANGLE: 0
COLOR: 1

Press 0 to animate when finished.

COPTER BLADES, SIDE VIEW

Here we are going to create the copter blades as seen from a side view. In some of the examples, you may notice that some of the generated shapes are identical to one another. Don't be too concerned with saving memory unless you absolutely have to – keep things simple by leaving the identical shapes in their place. Place D(9,1) on the grid.

```
PROGRAM 1
A=DY:B=DY:AXDC9:BXDC-9:/6:.A-B
@D-U1D1R1D2L1D5U5L1U2R1
```

SOURCE SHAPE: CLEAR
LOW LIMIT: 1
HIGH LIMIT: 30
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 30
STARTING ANGLE: 0
ENDING ANGLE: 0
COLOR: 1

If you would like to generate the entire helicopter in one sweep, place the copter body into the buffer before RUNning, and then select "BUFFER" for your SOURCE SHAPE. If you want an added element of realism, try making a copter that tilts to and fro. Press 0 to animate when finished.

ROTATING TRIANGLE

In this example, we are creating a rotating triangle that only needs to rotate 120 degrees. A full rotation is not necessary since the triangle is symmetrical in thirds. Place D(11,10) on the grid.

```
PROGRAM 1
AXDC11:AYDS10:/120
BXDC11:BYDS10:/120
CXDC11:CYDS10:/120
.A-B-C-A
```

```
SOURCE SHAPE:  CLEAR
LOW LIMIT:  1
HIGH LIMIT:  32
STARTING SHAPE:  1
DIRECTION: 1
PROG1 CYCLES:  32
STARTING ANGLE:  0
ENDING ANGLE:  120
COLOR:  1
```

Press 0 to animate when finished.

SPINNING ASTEROID

In this example, we are using “polar” coordinates to generate the shape of an asteroid. Polar coordinates provide a rigidness for generating shapes that you wish to rotate. In this example we need to use two separate programs due to the numerous instructions needed to generate the asteroid. Before RUNning the example you will need to clear shapes one through thirty-two. Place A(11,10) on the grid.

```
PROGRAM 1
/20:BXAC10:BYAS10:/70:CXAC10:CYAS10
/60:DXAC10:DYAS10:/30:FXAC4:FYAS4
/20:EXAC9.5:EYAS9.5:/30:GXAC8:GYAS8
/40:HXAC9.5:HYAS9.5:/90:.B-C-D-E-F-G-H
```

```
PROGRAM 2
/20:BXAC10:BYAS10:/250:HXAC9.5:HYAS9.5
/50:IXAC9.5:IYAS9.5:/10:JXAC7:JYAS7
/30:.H-I-J-B
```

SOURCE SHAPE: SAME
LOW LIMIT: 1
HIGH LIMIT: 32
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 32
PROG2 CYCLES: 32
STARTING ANGLE: 0
ENDING ANGLE: 720
COLOR: 1

Press 0 to animate when finished.

TEETER TOTTER

This example uses many different commands to produce two people on a moving teeter totter. Here we are creating a pretty complex animation with only a few lines of program text. What could take hours of trial and error by hand can be accomplished by The Spreditor in a matter of minutes. You will need to place A(11,16), B(9,20), C(14,20), and D(12,16) on the grid before running the program.

```
PROGRAM 1
EXAC-10:EYAS-10:FXDC10:FYDS10
@E-RU3RU2RDLD2RLD2RD3URU3
@F-LU3LU2LDRD2LRD2LD3ULU3
EX-1:FX1:.E-F  A-B-C-A-D
```

SOURCE SHAPE: CLEAR
LOW LIMIT: 1
HIGH LIMIT: 20
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 20
STARTING ANGLE: 340
ENDING ANGLE: 380
COLOR: 1

Use **ANIM 1,1-20,19-2** to animate when finished.

3D SPINNING CUBE

This program creates a spinning 3D "wireframe" cube. Although creating such 3D graphics takes a lot of preplanning, The Spreditor makes the task a lot easier. This example only lacks true 3D depth, which is difficult to establish. You will need to place A(11,10), B(11,13), C(11,3), D(11,17), E(11,7), F(12,7), G(12,17), H(12,3), and I(12,13) on the grid before running the program.

```
PROGRAM 1
/37:BXAC11:/25:CXAC7:/56:DXAC7:/25
EXAC11:/74:FXAC11:/25:GXAC7:/56:HXAC7
/25:IXAC11:/37:.B-C-E-D-B-I-H-F-G-I
.C-H  E-F  D-G
```

SOURCE SHAPE: CLEAR
LOW LIMIT: 1
HIGH LIMIT: 40
STARTING SHAPE: 1
DIRECTION: 1
PROG1 CYCLES: 40
STARTING ANGLE: 0
ENDING ANGLE: 360
COLOR: 1

Press **0** to animate when finished.

APPENDIX D

Help Routines

SPRITE INITIALIZATION ROUTINE

Here's a routine that can be of great assistance to you when you are using shapes generated by The Spreditor in your own programs:

```
10 DIMB(8):FORD=0TO7:B(D+1)=2↑D:NEXT

5000  POKE2039+SN,(SL/64+SH-1)AND255
5010  PK=PEEK(SL+SH*64-1):POKE53286+SN,PK:
MA=255-B(SN)
5020 POKE53276,(PEEK(53276)ANDMA)ORB(SN*
SGN(PKAND16))
5030 POKE53271,(PEEK(53271)ANDMA)ORB(SN*
SGN(PKAND64))
5040 POKE53277,(PEEK(53277)ANDMA)ORB(SN*
SGN(PKAND32))
5050 POKE53269,(PEEK(53269)ANDMA)ORB(SN*
SO):RETURN
```

You should include the first line somewhere at the beginning of your programs (you may renumber these lines, of course). Before entering the routine at 5000, you need to set some variables:

SL = Shape Location, the address where your sprite data begins.

SN = Sprite Number, the number of the sprite you wish to initialize (range 1-8).

SH = SHape, the shape you wish to initialize your sprite with (range 1-128).

SO = Sprite On/off. Use “1” if you want your sprite turned on, or use “0” if you are not ready to display your sprite.

This routine makes it incredibly easy to initialize sprites that are created with The Spreditor. You won’t have to pump out a long series of confusing pokes when setting up your sprites. The only pokes you need to concern yourself with are those that deal with moving your sprites into position, and the two colors shared by all multicolor sprites. And don’t let the variables confuse you – **SL** should be set to the location specified when you saved your sprite shapes to disk (if you are loading them in at that location), and **SH** should correspond to the shape numbers as you remember them from the editor. If you move the visible text screen to a different location in memory, then you will have to change the value of 2039 in the first line of the routine to the appropriate new value. To calculate the new value, take the location to where you’ve moved the display screen to and add 1015.

DISK DATA LOADING ROUTINE

Why waste program space when you can load your sprite shapes and ML routines directly into memory with a simple routine? Using DATA statements just wastes considerable memory and can also take up more time than a simple LOAD when you are dealing with large amounts of data. The following routine will load your data into memory, and can also be used to load up files created by the users of your program. The routine cleverly places the filename into locations 2024-2039 before executing the SETNAM routine. I chose this area of memory because it is definitely not a common place for data storage. It is also exactly the length we need it to be – sixteen bytes long. Here’s the routine:

```
6000 POKE780,3:POKE781,DV:POKE782,SE:SYS
```

```
65466:L=LEN(F$):POKE780,L
6010 FORD=1TOL:POKE2023+D,ASC(MID$(F$,D)
):NEXT:POKE781,232:POKE782,7:SYS65469
6020 POKE780,0:POKE782,LA/256:POKE781,LA
–PEEK(782)*256:SYS65493:RETURN
```

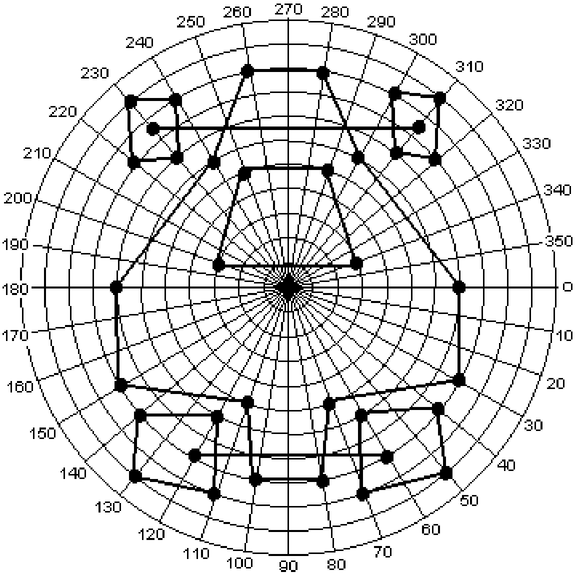
Of course you may renumber the lines without any problems, if you need to. Before entering the routine at 6000, you need to set some variables:

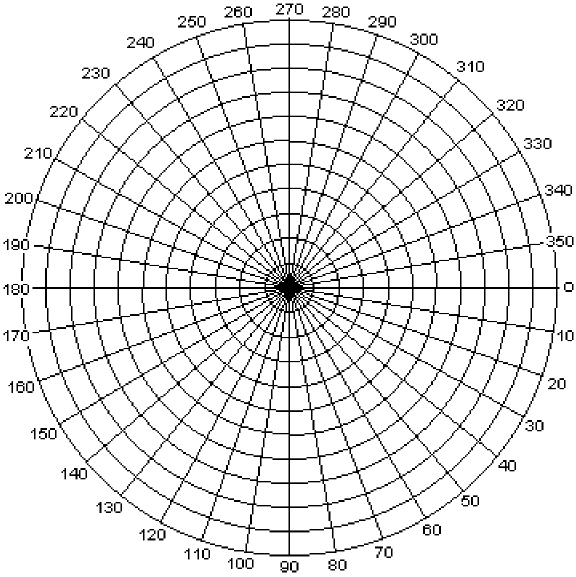
F$ = The Filename of the program you wish to load.
DV = The DeVice number of the disk drive (or other device) you wish to load from.
SE = SEcondary address. A “1” causes the program to load in at its own loading address. A zero causes the program to load in at the location *you* choose.
LA = Loading Address, the address you wish the program to start loading in at (you must set the variable **SE** to zero or this address will be ignored).

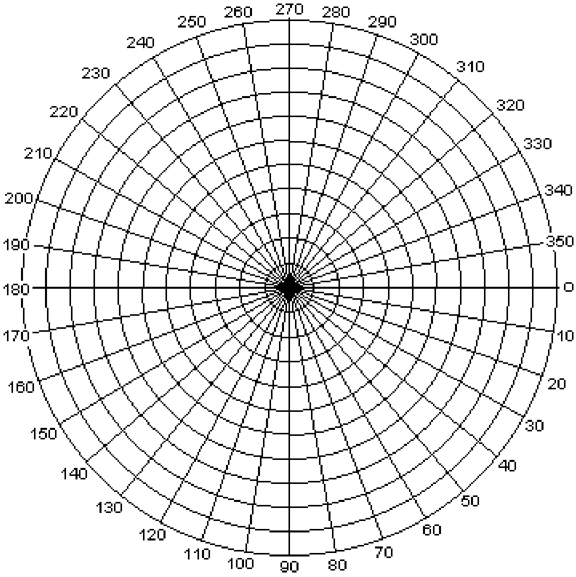
APPENDIX E

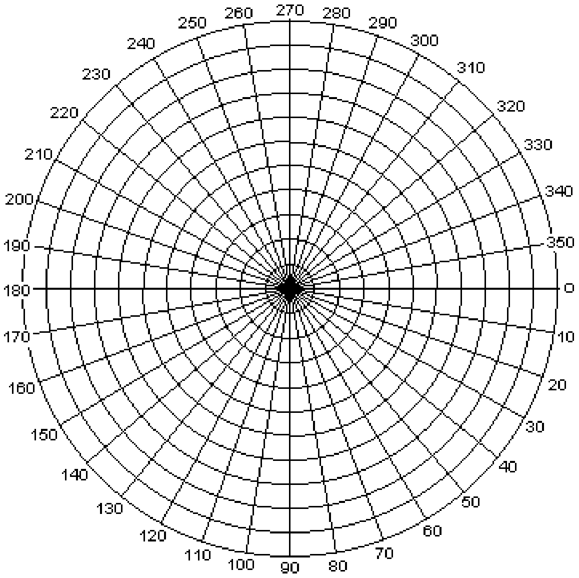
Polar Coordinate Graph Circles

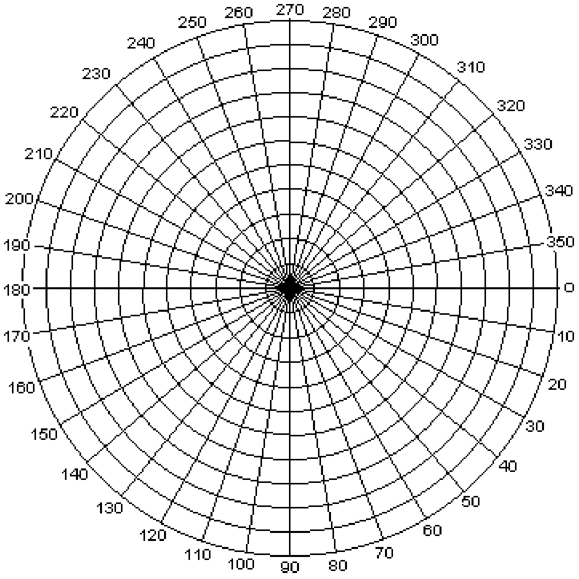
The graphing circles on the following pages can be photocopied for designing objects that use polar coordinates as reference points for object rotations. The circle below illustrates how they may be used. I wish I had created some of these much earlier for my own personal use. I hope you enjoy using The Spreditor for all of your sprite-making needs.

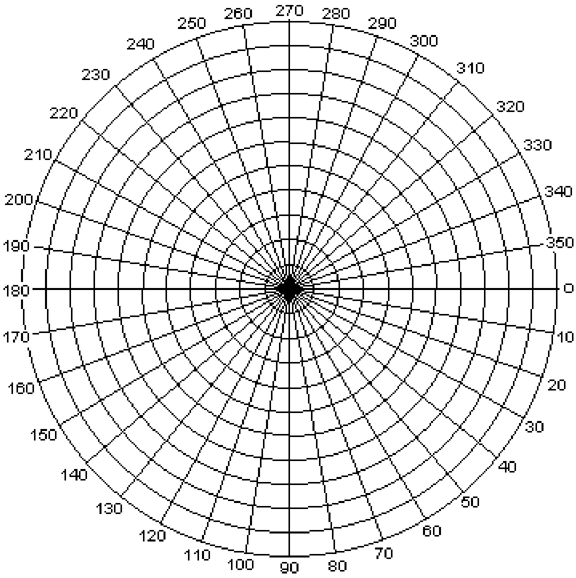


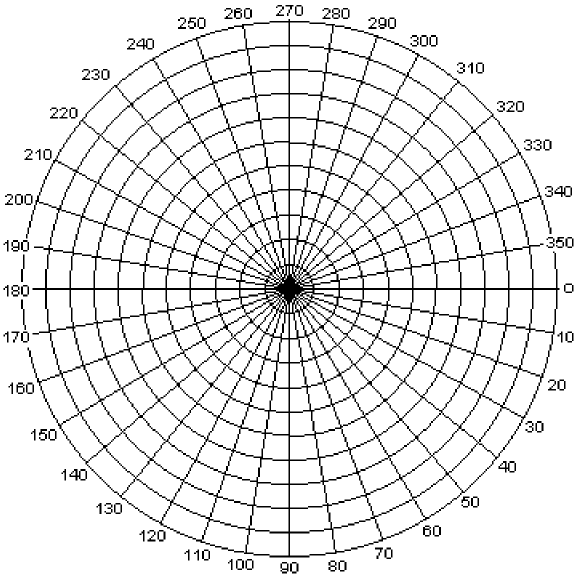

270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260


270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260


270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260


270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260


270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260


270
280
290
300
310
320
330
340
350
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
210
220
230
240
250
260