THE NEW BASIC KEYWORDS AND SPECIAL CHARACTERS

Sprite Commands

ANIMATEn, (d1,d2)
AUTOREV ON/OFF
CHASE ON/OFF
DGETn,(xdim,ydim)
FLEE ON/OFF
FLETCH(n)
MAZE ON/OFF
MOVEn
MOVEn1,n2
MOVEM
NODE(n)=x,y,dir
SDIMS=(x1,y1)–(x2,y2)
SGETn, (drawing, attributes)
SMODEn,p
SMODEQ
SPEEDn
SPUTn, (x,y)

Sprite Functions

X=ATTR(n)
X=COX(n)
X=COY(n)
X=DIR(n)
X=DRWG(n)
X=HIT
X=REPORT
X=REPORT(n)

ATTR(n)=X
DIR(n)=X
DRWG(n)=X

Text Commands

@POS(column,line)
CHR(n)=r0 . . . . . r7
COLOURf,b
HOLDn
PAGE ON/OFF
SCORE=X

Special Characters

CHR$ (8) Cursor Left
CHR$ (9) Cursor Right
CHR$ (10) Cursor Down
CHR$ (11) Cursor Up
CHR$ (12) Cls and Cursor Home
CHR$ (31) Clear to End of Screen

Joystick/Keyboard Commands

ANALOG ON/OFF
STIX(n) ON/OFF, ON/OFF
STIX(n) ON/OFF, (x,y)

Joystick/Keyboard Functions

X=BUTTON
X=INKEY
X=KEY

General Commands

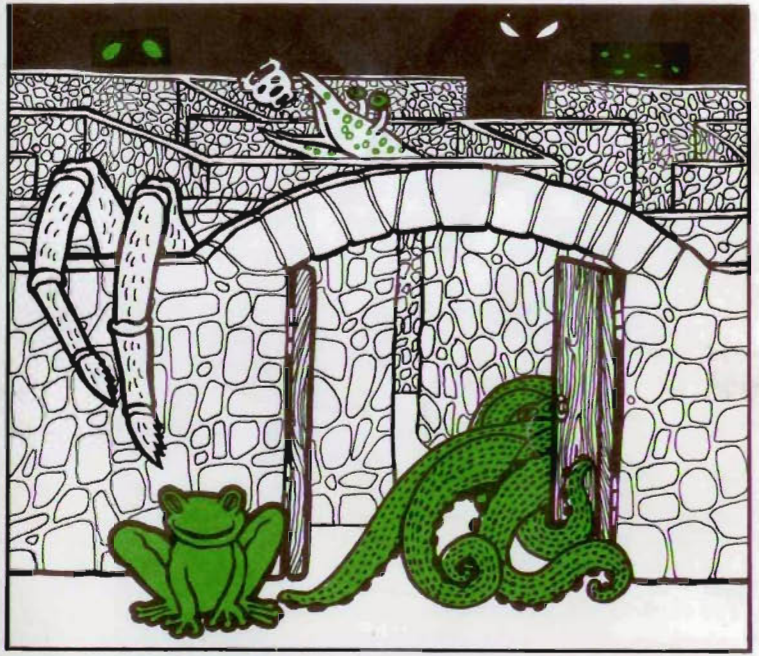
BREAK ON/OFF
KEEP ON/OFF
RESERVEn

Sound Commands

AUTOBEEP(n)=b
BEEPn
BEEP(period,proc,vol,vroc,cycles,type)

© Copyright Knight Software 1984
Designed and Printed by Express Print, Stockton, Cleveland.

DRAGON 32
"SPRITE MAGIC"



- Up to 128 non-destructive sprites
- Size single pixel to 1600 pixels
- Automatic maze-running mode
- Collision detection
- Text in all modes, true lower case ASCII.
- Re-definable character set.
- Auto-repeat keyboard
- Enhanced sound facilities
- Dozens of powerful new commands
- Six free demonstration programs

KNIGHT SOFTWARE

SPRITE MAGIC

Loading: First of all, reserve some space at the top of memory for the data which is about to be loaded, load the program and initialise the system:–

CLEAR200,&H6000:CLOADM:EXEC

Once the data has loaded you will see a copyright message and the familiar OK prompt. Sprite Magic is now at your disposal and you may load and run the Basic programs which follow on the Sprite Magic Cassette or you may commence developing your own software.

DEMONSTRATION PROGRAMS

All the programs are named "DEMO n" where n is the program number (1–7). CLOAD "DEMO 1", for example, will load the first program. . .

DEMO 1 gets individual characters and strings of characters as sprites and moves them around the screen in various ways.

DEMO 2 displays the graphics font and allows you to re-define the character set and save it to tape.

DEMO 3 outputs Sprite Magic's pre-programmed sounds and shows how to generate new ones.

DEMO 4 draws a chessboard and pieces and plays a "mate in two".

DEMO 5 is a fairground shooting gallery simulation and requires the use of a joystick/fire button.

DEMO 6 is "Breakout", the old classic where you control a paddle to keep a ball bouncing round the screen while you demolish a wall. The reward for this another wall. . . .

DEMO 7 is a recording of the program listing discussed below.

HOW IT WORKS

Having loaded and initialised Sprite Magic, you may use the high resolution screen for program development . . . enter the following line of Basic:–

SMODE4,1:SCREEN1,1:COLOUR1,0

This enables graphics mode 4 but instead of returning immediately to the green text screen, you will see the OK message and the new (static) text cursor at screen top left, awaiting further input. You should now proceed just as you would with the old text screen, except that all of Sprite Magic's new facilities are available to you.

The initialisation procedures for generating sprites are quite straightforward. First of all, the programmer prepares a number of drawings at screen top left and these are copied by Sprite Magic's DGET command, in much the same way as the GET command copies drawings. Next, the drawings are assigned to a number of sprites along with instructions about how the sprites are to be moved. Then the sprites are displayed, using the new SPUT command and set in motion with one of the MOVE commands. Once the sprites are mobilised, most of the hard work is over because so much is handled automatically by Sprite Magic. Often, all that is necessary is for the program to take appropriate action following collisions or after the expiry of a time limit.

The seventh demonstration program shows how to initialise sprites, how to use Sprite Magic's Maze Mode, how to control sprites from the keyboard or joysticks and how to animate sprites. The program is listed in full on page 3.

LINE 20 restricts Basic to the first &H6000 bytes of memory, leaving the remaining 8K for the Sprite Magic program and drawing data. The RESERVE command keeps some space free from drawings for data which will be generated later.

LINE 30 uses a FOR . . . NEXT loop to read the data for the maze nodes. A node is simply a point in the maze where two or more tracks intersect.

LINE 40 implements graphics mode 4, using the new forms of PMODE and COLOR.

LINE 60 draws a box at screen top left and uses the new command DGET to save a copy of the drawing for future use.

LINE 70 takes copies of three of Sprite Magic's graphics characters.

LINES 80 to 100 use a couple of loops to DGET four more drawings. This time, drawing data is POKEd into the video RAM starting at location &H600, which is screen top left. This method is sometimes easier for certain types of drawing and is also used on the chess demonstration program.

LINE 110 DGETS character 150, a ball.

LINES 130 to 150 initialise the sprites, using the new SGET command. The first value after SGET is the sprite number and the values in parenthesis are the number of the drawing to be used and the movement attributes. These are explained in detail in the next section.

LINE 180 calls the subroutine at Line 300 to draw the maze and the SPUT command displays sprite 0 at co-ordinates 132,152. This sprite has been placed into a horizontal track in the maze and the MDIR command restricts sprite 0's possible directions to left or right and prevents vertical motion. Unlike the others, sprite 0 will be controlled by the user and MDIR is required to prevent the sprite being guided in "impossible" directions when it first sets off. Subsequently, sprite 0 will be restricted automatically to permitted changes of direction.

LINE 190 puts three more sprites into the maze.

LINE 210 places Sprite Magic in Maze Mode.

LINE 220 sets the sprites in motion with the MOVEM command. The FLETCH command scans the keyboard and adjusts sprite 0's direction in response to the cursor keys. As you can see, Line 220 forms a closed loop so the rest of the program will not execute . . . yet.

LINE 240 displays another four sprites and tells Sprite Magic to animate them, using drawings 4 through 7. This means that every time one of the sprites is moved, a different drawing will be used.

LINE 250 displays the ball.

LINE 260 sets the screen dimensions to be observed when the ball bounces.

LINE 280 uses various forms of the MOVE command, first of all to move ALL the sprites and then to move only sprites 0, 1, 2, 3 and 8 (three times). The effect is to slow down sprites 4 to 7 making the animation more realistic.

Run the program and you will see three of the sprites moving around the maze. The other sprite (sprite 0) will respond to the cursor keys . . . try pressing them and observe how the other three sprites give chase. Now press **BREAK** and add the following line to the program:—

215 STIX (0) ON,ON:ANALOGON

Connect a joystick to the right hand port, run the program again and sprite 0 will now respond to the joystick. Note that the STIX command has precedence over FLETCH. Press Break once more, change Line 220 into a REM statement and run the program again. This time you will see the remaining sprites displayed.

Press Break again and experiment further with the new features described in the following section.

```
10 '       ** DEMO 7 **
20 CLEAR200,&H6000:RESERVE96
30 FORI=0TO31:READX,Y,D:NODE(I)=X,Y,D:NEXT
40 SMODE4,1:SCREEN1,1:COLOUR0,1
50 '   ** GET THE DRAWINGS **
60 PCLS:LINE(0,0)-(6,6),PSET,B:DGET0,(6,6)
70 FORI=1TO3:PRINTCHR$(12)CHR$(143+I):DGETI,(6,6):NEXT
80 P=&H600:FORI=4TO7:PCLS
90 FORJ=0TO12:READA:POKEP+J*32,255-A:NEXTJ
100 DGETI,(7,12):NEXTI
110 PRINTCHR$(12)CHR$(150):DGET8,(7,7)
120 '   ** GET THE SPRITES **
130 FORI=0TO3:SGETI,(1,2):NEXT
140 FORI=4TO7:SGETI,(1,6):NEXT
150 SGET8,(8,33)
160 PCLS:@POS(10,0):PRINT"** DEMO 7 **
170 '* DRAW MAZE/PUT SPRITES *
180 GOSUB290:SPUT0,(132,152):MDIR(0)=10
190 FORI=1TO3:SPUTI,(64+I*18,32):NEXT
200 '    ** RUN THE MAZE **
210 MAZEON
220 FLETCH(0):MOVEM:GOTO220
230 ' * SPUT RUNNERS AND BALL *
240 FORI=4TO7:SPUTI,((I-4)*64,174):ANIMATEI,(4,7):NEXT
250 SPUT8,(99,89)
260 SDIMS=(1,1)-(247,183)
270 ' ** SET 'EM ALL GOING **
280 MOVEM:MOVE0,3:MOVE8:MOVE0,3:MOVE8:MOVE0,3:MOVE8:GOTO280
290 '    ** DRAW THE MAZE **
300 X=63:Y=31:XM=124:YM=92
310 FORI=0TO32STEP8:XL=X+I:XR=191-I:YT=Y+I:YB=159-I
320 LINE(XL,YT)-(XR,YB),PSET,B
330 IFI=0ORI=32THEN380
340 LINE(XL,YM)-(XL,YM+6),PRESET
350 LINE(XR,YM)-(XR,YM+6),PRESET
360 LINE(XM,YT)-(XM+6,YT),PRESET
370 LINE(XM,YB)-(XM+6,YB),PRESET
380 NEXT:RETURN
390 '      ** NODES DATA **
400 DATA64,32,6,64,92,7,64,152,3,72,40,6,72,92,15,72,144,3
410 DATA80,48,6,80,92,15,80,136,3,88,56,6,88,92,13,88,128,3
420 DATA124,32,14,124,40,15,124,48,15,124,56,11,124,128,14
430 DATA124,136,15,124,144,15,124,152,11
440 DATA160,56,12,160,92,7,160,128,9,168,48,12,168,92,15,168,136,9
450 DATA176,40,12,176,92,15,176,144,9,184,32,12,184,92,13,184,152,9
460 '     ** RUNNER DATA **
470 DATA24,24,8,28,90,60,24,24,40,72,68,66,196
480 DATA24,24,8,188,90,24,24,24,24,24,20,8
490 DATA24,24,8,28,154,121,26,24,24,31,17,16,48
500 DATA24,24,8,60,24,24,24,120,72,40,24,24,8
```

THE NEW BASIC COMMANDS

In the following description of Sprite Magic's commands and functions, please note that :—

1) All screen co-ordinates, sprite dimensions and the like are expressed in terms of the 256 x 192 grid that Basic uses to reference the graphics display.

2) Where we give a range of values, say 0 to 31, the effective range includes the two limits.

3) We have observed Microsoft Basic's syntax for the new commands. Where a command requires parameters to be passed these are always in parenthesis or after the equals directive. System variables, such as sprite numbers, sound effect numbers and the like are outside the parentheses.

4) Some of the examples given below use Basic's bit testing facilities. If you are unfamiliar with bit testing, please refer to Appendix A.

@POS

The @POS command moves Sprite Magic's print cursor to a new position specified by column and line number and it is therefore a substitute for PRINT@.

@POS (x,y)

x is the column number where output is to commence, range 0 to 15 (SMODES 0 to 3) or range 0 to 31 (SMODE 4).

y is the line number where output is to commence, range 0 to 11 (SMODES 0 and 1) or range 0 to 23 (SMODES 2 to 4).

Examples:— @POS(13,0):?"TITLE" @POS(13,L):?"TEXT" @POS(3,4):?USING"###";X

ANALOG

The ANALOG command governs how Sprite Magic handles joystick response.
With ANALOGOFF, the joysticks are handled as switches and the sprites will either remain stationary or move at full speed in one of the eight directions. With ANALOGON, sprite speed and direction will be proportional to the position of the joysticks, making control much easier.

ANIMATE

The ANIMATE command permits sprites 0 to 7 to be animated automatically by the MOVE commands. If you specify the sprite number and the block of drawings to be used, Sprite Magic will step through the drawings in sequence each time one of the MOVE commands is executed. On reaching the end of the specified range, Sprite Magic will begin again with the lowest numbered drawing.

ANIMATEn, (start,finish)

n is the number of the sprite to be animated, range 0 to 7.

start and finish specify the consecutively numbered block of drawings that are to be used. When start and finish have the same value animation is cancelled.
For example, if you initialise sprite 4 using drawing 12 and then execute ANIMATE4, (10,14) and then set the sprite(s) in motion with MOVE, Sprite Magic will move sprite 4 using drawing 12, then 13, then 14, then 10, then 11... 12... 13.. and so on. The drawings should all have the same dimensions.

AUTOBEEP

The AUTOBEEP command permits automatic output of Sprite Magic's sound effects.

AUTOBEEP (n)=m

n is a number in the range 0 to 2 and m is Sprite Magic's BEEP number, range 0 to 15. When m is greater than 15, AUTOBEEP is cancelled.

n=0 outputs BEEPm when a sprite reaches the edge of the screen.

n=1 outputs BEEPm when a sprite is fired in response to one of the joystick buttons.

n=2 outputs BEEPm when a sprite collides with some other displayed object.

AUTOREV

The AUTOREV command causes sprites to reverse direction following a collision. The feature is effected as part of the REPORT function and applies only to sprites programmed for collision detection. **AUTOREVON** turns on the facility and **AUTOREVOFF** cancels it.

BEEP

The BEEP command outputs one of a range of 16 pre-programmed sound effects or allows the programmer to devise new ones. New sound effects are generated with a single command and although six parameters are required, a wide range of effects is possible.

BEEPn

n is the sound effect number in the range 0 to 15. Greater values will be ignored.

BEEP (period, period ROC, start volume, volume ROC, cycles, type)

period is the duration of one half-cycle of the sound and is therefore inversely proportional to frequency. The permitted range is 0 to 2047.

period rate of change is the adjustment to be made to the period after each full cycle. The permitted range of values is —128 to +127. Note that BEEP terminates sound output if period goes negative or if period exceeds a value of 4095. However, a special effect is that if period attains a value of exactly zero then BEEP inverts period rate of change and sound output continues.

start volume has an effective range of 4 (quietest) to 255 (loudest).

volume rate of change is the adjustment to be made to the volume after each full cycle. For the perceived volume actually to change, the volume parameter must be adjusted by at least 4, which is the smallest single effective step size. After adjustment for rate of change, volume is calculated modulo 256 so that large values for volume rate of change permit the availabie range of sound levels to be implemented more than once during a single execution of BEEP. A special effect is that if volume attains a value of exactly zero then BEEP inverts volume rate of change and sound output continues.

cycles is the number of sound cycles to be output, range 0 to 32767.

type governs the wave shape of the sound. With a value of 0 for this parameter, BEEP generates a square wave whilst type 1 produces pseudo white noise. Our thanks to Mr Brian Cadge for the basic idea of using the Dragon's ROM in this way.

BREAK

The BREAK command allows the user to disable the Break Key. This is useful for programs that call for frequent use of keys situated near the machine's Break key, the arrow keys for example. **BREAKOFF** disables the Break key and **BREAKON** restores normal operation.

CHASE

The CHASE command forces all the sprites on screen to chase sprite zero. If sprite 0 is not currently on screen the sprites will head towards the co-ordinates where sprite 0 last appeared, else towards co-ordinates 0,0. The feature is turned on with **CHASEON** and **CHASEOFF** restores normal operation. This command has no effect when the MAZE feature is ON.

CHR

The CHR command allows Sprite Magic's characters to be redefined. Each character is formed on an 8 x 8 grid, as illustrated in the second demonstration program.

CHR (n) = r^0, r^1, r^2, r^3, r^4, r^5, r^6, r^7

n is the character number and corresponds to Basic's CHR$(n), although the range here is restricted to Sprite Magic's character set, characters 32 to 159.

r^0 to r^7 are the eight rows of the character. Each row value is the sum of the bits that are to be turned on when the character appears on screen. For example, CHR(95) may be defined as a large "L" shape with the following statement:– CHR (95)= 128,128,128,128,128,128,128,255.

COLOUR

The COLOUR command (note the anglicised spelling) is a direct replacement for Basic's COLOR command. It is in all respects similar to COLOR except that it permits Sprite Magic to initialise its variables without constant and repeated reference to Basic's records to find out whether the colour has been changed. Use COLOUR just as you would use COLOR.

DGET

The DGET command may only be used in high resolution graphics modes. DGET copies the graphics contents of the top left corner of the screen into the space below Sprite Magic. Space must previously have been reserved for drawings with the **CLEAR** command. Although DGET performs a similar function to Basic's GET, note that only only one pair of co-ordinates is required, for the bottom right corner of the drawing. The top left corner of the drawing is assumed to be at co-ordinates 0,0.

DGETn,(x,y)

n is the drawing number, in the range 0 to 31.

x,y are the lower right co-ordinates of the drawing.

FLEE

The FLEE command forces all the sprites on screen to run away from sprite zero. If sprite 0 is not currently on screen the sprites will avoid the co-ordinates where sprite 0 last appeared, else they will head towards co-ordinates 255,191. The feature is turned on with **FLEEON** and **FLEEOFF** restores normal operation.

FLETCH(n)

The FLETCH command places sprite number n under keyboard control and should be executed before moving the sprite with one of the range of MOVE commands.
FLETCH scans the keyboard arrow keys and adjusts the sprite's direction according to the key(s) which are pressed. If none of the relevant keys is pressed then the sprite's direction is cleared to zero. When sprites 2 or 3 have been programmed as missiles by the STIX command they will also be fired by keys 1 or 2 (sprite 2) and keys 8 or 9 (sprite 3).

HOLDn

The HOLD command prevents the top n lines of the display from scrolling when the cursor gets to the bottom of the screen. Its use is for "fixing" program titles and the like and for retaining a block of code on screen when listing and de-bugging programs. It may also be used to restrict text output to the lower part of the screen whilst the top is used for graphics. The bottom two lines of the screen may not be held so the range for this command is 0 to number of screen lines minus two.

KEEP

The KEEP command preserves Sprite Magic's drawing data and tables which are normally cleared by the RUN command. It is therefore possible to generate a number of drawings and sprites, delete the development program and then use the drawing and sprite data on a different program. **KEEPON** prevents RUN from clearing the tables and **KEEPOFF** reinstates normal operation.

MAZE

In Maze Mode, all the other sprites chase sprite zero, which will normally be subject to keyboard or joystick control. However, this is not the same as setting CHASEON as the sprites observe a number of rules.

1. A sprite's preferred direction is towards sprite zero.
2. Whenever available, sprites will always choose a preferred direction at a node, but they will not go into reverse, unless the node is a "dead end".
3. Sprites will only exit a node in one of the directions permitted by the NODE command.
4. When there is a choice consistent with the other rules, whether they turn left or right is selected randomly.
5. Sprites move orthogonally, never on the diagonals.
6. Sprites observe the FLEE command only when it is consistent with the preceding rules except that the preferred direction will be away from sprite 0.

MAZEON turns on the feature and **MAZEOFF** restores normal operation. The CHASE command has no significance in Maze Mode.

MDIR

When sprites are first placed into a maze, it is important that the direction specified for SGET should be a direction which is actually available, given the construct of the maze. In other words, do not place a sprite programmed to move vertically into a track that only runs left to

right. With regard to sprite zero, more than one direction will usually be available and the MDIR command initialises this sprite's possible starting directions.

MDIR(0)=direction bits

direction bits is calculated as the sum of the possible directions which sprite 0 may take from its initial maze location. Thus, if the starting position is on a left/right track, directions will have a value of 10 (right=2 + left=8), as detailed in the description of the ATTR function. The command is redundant if sprite 0 is placed initially at a node.

MOVE

The MOVE command sets the sprites in motion. Various forms of the MOVE command permit movement of a single sprite, a block of sprites or all of the sprites that are on the screen. The distance is determined by the SPEED command which sets the number of units on the 256 x 192 grid that sprites are to travel during one MOVE.

MOVEn

n is the number of the sprite to be moved, range 0 to 127, and should refer to a sprite which is presently on display.

MOVEn1,n2

n1,n2 are the start and end numbers of a block of sprites that are to be moved. Sprites outside the defined range or sprites that are not on display are ignored.

MOVEM moves all the sprites that are presently on screen.

The MOVE command updates its record of sprite co-ordinates and directions and adjusts various flags which are all accessible with Sprite Magic's function calls. Sprites that are programmed to disappear or rebound at the edge of the screen will do so and sprites programmed to respond to the joysticks, joystick buttons or keyboard will act accordingly.

NODE

The NODE command defines the co-ordinates and possible exit directions for maze junctions. Each node requires 3 bytes of memory (above MEMTOP and below the drawing data) and space must previously have been reserved with the RESERVE command.

NODE(n)=x,y,d

n is the node number, range 0 to 255.

x,y are the node's screen co-ordinates on the 256 x 192 grid.

d is the sum of the possible exit direction from the node, where up=1, right=2, down=4 and left=8.

PAGE

The PAGE command introduces a Paging Mode for text on the high resolution screen. In Paging Mode, the machine will wait for a keypress before continuing output when the cursor gets to the bottom of the screen. Press the space-bar to output a further single line of text and press the Clear key to output another screenful. **PAGEON** enables Paging Mode and **PAGEOFF** turns it off. Pressing the Break key also turns off Paging Mode.

RESERVE

When Sprite Magic copies drawings, the data is normally stored above MEMTOP and below Sprite Magic, in the space reserved by the CLEAR command. However, using the RESERVE command it is possible to keep some memory free for user routines, just above MEMTOP.

RESERVEn

n is the number of bytes to be kept free. Note that the NODE command also uses this space so when Sprite Magic is in Maze Mode, your routines should reside in memory at MEMTOP, plus number of nodes x 3.

EXAMPLE MEMORY MAP WITH RESERVED SPACE

Basic Program:— 10 CLEAR200,&H6000
20 RESERVE400
30 NODE0... NODE20

Basic uses the first &H6000 bytes of memory	from &H0000 to &H5FFF
Nodes use the next 63 bytes of memory	from &H6000 to &H603E
Free for user routines	from &H603F to &H618F
Drawing data is stored after the reserved space	from &H6190 to &H6F7F
Sprite Magic uses	from &H6F80 to &H7FFF

SCORE

The SCORE command prints a number at the current text cursor location and resets the cursor back to its starting position, ready for the next number to be output. The number is printed in a field width of five charactors and is equivalent to PRINTUSING "#####";X except that leading spaces are "stepped over", rather than printed. It is used, as the name suggests, for outputting games scores quickly and without the need to reposition the cursor repeatedly.

SCORE=X

X is the value to be displayed, range 0 to 32767.

SDIMS

The SDIMS command is used to set the screen limits observed by the MOVE command when "wiping" or "bouncing" sprites.

SDIMS=(x1,y1) — (x2,y2)

x1,y1 are the co-ordinates of the top left hand corner of the screen.

x2,y2 are the co-ordinates of the bottom right hand corner of the screen.

When initialising sprites and setting the screen dimensions, note that MOVE takes the necessary action when the top left hand corner of a sprite coincides with one of the edges of the screen. The important point here is that one of the sprite's co-ordinates must be the same as (not greater or less than) one or more of the screen edges. This permits flexibility with regard to which screen edges will affect a sprite's screen-edge behaviour.
For example, suppose that a sprite is SPUT at co-ordinates (33,40) and the SDIMS are (0,0) — (248,184) and the SPEED is set to 2 and the sprite is programmed to rebound from the edge of the screen. Under these circumstances, the sprite will rebound from the top and bottom edges of the screen but it will "wrap round" left to right because the sprite's X co-ordinate will always

have an odd value and consequently will never coincide with the even values set for the left and right hand screen edges.

SGET

The SGET command initialises a sprite by defining its drawing number and movement parameters.

SGETn,(drawing, direction + attributes)

n is the sprite number in the range 0 to 127.

drawing is the number of the drawing which will be used for the sprite.

direction is the direction the sprite is to take when it is set in motion. The range of directions starts with direction 1 = "north east" and continues clock-wise around the compass in 45° steps, ending with direction 8 = "north". A zero value may be used for a stationary sprite.

attributes is optional. It is a multi-purpose value which is added to the direction number and it specifies whether the sprite is to be examined for collisions and how the sprite will behave at the edge of the screen. Comprising four "flags", attributes is calculated as follows:—

wipe has a value of 16 and adding this value to the direction number causes the sprite to disappear when it reaches the edge of the screen.

bounce has a value of 32 and adding this value to the direction number causes the sprite to rebound from the edge of the screen.

collision detect has a value of 64 and adding this value to the direction number tells Sprite Magic that the sprite is to be examined for collisions. Collision detection is initiated by the REPORT function.

status has a value of 128 and indicates that the sprite is currently displayed. This flag is different from the others in that it is Sprite Magic's (not the programmer's) record of the sprite's status. When initialising sprites, the status element of attributes will normally be zero and Sprite Magic will adjust it automatically when the sprite is SPUT.

Any or all of SGET's parameters may be adjusted later in the program, either automatically as a result of a SPUT or MOVE command, or by use of the various functions that are available for the purpose.

SMODE

The SMODE command is a direct replacement for Basic's PMODE command. It is in all respects similar to PMODE except that it permits Sprite Magic to initialise its own variables without constant and repeated reference to Basic's records to find out whether the mode has been changed.
Also, it enables Sprite Magic's text and keyboard handling routines and subsequent text will be output on the high resolution screen, rather than Basic's text screen. Please note that in immediate mode (without program lines), SMODE always clears the selected high resolution screen and homes the print cursor to screen top left. The screen is not cleared in program mode and it is therefore possible to change SMODEs without losing graphics displays.
The keyboard handling routines offer auto repeat, so that output continues if a key is held down. The initial delay, before auto repeat commences, may be tailored to suit your typing ability by adjustment of memory location &H6F86 (28550$_{10}$). To disable auto-repeat entirely POKE&H6F86,0 and to enable auto-repeat POKE&H6F86, delay where delay is a value between 1 and 255. You will probably find that a delay value of about 60 is about right. Use SMODE just as you would use PMODE and use the high resolution text facilities just as you would use Basic's text screen.

SMODEQ

The SMODEQ command disables SMODE and returns control to the old screen and keyboard handling routines.

SPEEDn

The SPEED command is the master control for sprite movement and it sets the number of pixels (on the 256 x 192 grid) that a sprite will travel during one execution of MOVE. The permitted range is 1 to 15 and the default is SPEED2.

SPUT

The SPUT command is used to display a sprite, previously initialised with the DGET and SGET commands. SPUT should be used in the same mode that was used to DGET the drawing, otherwise the results may be unpredictable.

SPUTn,(x,y)

n is the sprite number and should previously have been initialised by SGET.

x,y, are the screen co-ordinates where the sprite is to be displayed. The co-ordinates refer to the screen position of the top left hand corner of the sprite.

SPUTn

n is the sprite number, as defined in SGET. The command is used to display a sprite for which co-ordinates have already been defined. Since SPUT has a "toggling action" the effect of repeated execution is to turn the sprite on and off. That is, a sprite which is "on" will be removed from display and a sprite which is "off" will be returned to the screen.

STIX

The STIX command specifies how sprites 0 to 3 are to respond to joystick control Sprites 0 and 1 may be programmed to respond to the right and left hand joysticks, either wholly or in a single axis.

STIX(n) h,v

n is the sprite number, either 0 or 1.

h and **v** refer to control in the horizontal and vertical axes and will be either ON or OFF.

Sprites 2 and 3 may be programmed as "missiles", fired from sprites 0 and 1 in response to the right and left hand joystick buttons. When the missile sprites are fired they will start from a position relative to their "mother" sprites, wherever the mother sprites happen to be on screen. The STIX command allows the programmer to define the offset to be applied.

STIX(n)s,(x,y)

n is the sprite number, either 2 or 3.

s enables or disables fire-button control and will be either ON or OFF.

x and y are the offsets to be applied to sprite (n−2)'s position for calculating the start position of sprite n. The effective range of values is −128 to +127. These parameters are only required when fire-button control is ON.

Examples:−

STIX(0)ON,OFF turns on joystick control for sprite 0, in the horizontal axis only.

STIX(1)ON,ON turns on joystick control for sprite 1, in both axes.

STIX(2)ON,(4,−10) programs sprite 2 as a missile fired in response to the right hand joystick button. It will appear at a position 4 pixels to the right and 10 pixels above sprite 0.
STIX(2)OFF turns off the facility.

FUNCTIONS AND SYSTEM VARIABLES

ATTR(n) Returns the sum of the direction, screen edge, collision detect and status (on or off screen) flags for sprite n. The direction value is not the same as that reported by the DIR function. In order to allow bit testing, the ATTR direction value is the sum of up (1), right (2), down (4) and left (8). Thus, if a sprite's current direction is towards the top left corner of the screen, the direction value included in ATTR will be 9 (up=1 + left=8).
ATTR will include a value of 16 for a sprite that has been programmed to disappear at the edge of the screen and 32 for a sprite programmed to rebound from the edge of the screen. The collision detect flag has a value of 64 and the final parameter, status, has a value of 128, for sprites that are presently on screen.
Examples:- X=ATTR(N) U=ATTR(N)AND1 B=ATTR(8)AND32
ATTR(1)=3 ST=ATTR(1)AND128 ATTR(5)=ATTR(5)OR64

BUTTON Returns a value of 1 if the right hand button is pressed, 2 if the left hand button is pressed and 3 if both are pressed.
Examples:- X=BUTTON R=BUTTONAND1 L=BUTTONAND2

COX(n) Returns the current screen X co-ordinate of sprite n.
Examples:- X=COX(N) X1=COX(4)−COX(7) IFCOX(3)<10THENSPUT3

COY(n) Returns the current screen Y co-ordinate of sprite n.
Examples:- Y=COY(N) Y1=COY(H)−COY(0) IFCOY(3)=170THENEND

DIR(n) Returns sprite n's direction and allows it to be adjusted. The direction will have a value in the range 0 to 8 and is as defined in SGET. It is not the same as the direction value returned by the ATTR function.
When adjusting a sprite's direction, note that Sprite Magic will "correct" user-assigned directions by deducting 8 from any values greater than 8. This permits statements such as DIR(n)=DIR(n)+6 where sprites n's original DIR of 5, say, would be corrected to 3 (5 + 6=11−8 =3).
Examples:- D=DIR(4) DIR(1)=6 DIR(N)=DIR(N)+4

DRWG(n) Returns the current drawing number being used for sprite n and allows it to be altered. When a sprite's drawing is changed, Sprite Magic first removes the sprite from display and then replaces it at the same co-ordinates but using the new drawing. Animation is thus possible by assigning a sequence of slightly different drawings to a moving or stationary sprite.
Examples:- E=DRWG(N) DRWG(N)=4 DRWG(N)=DRWG(N)+1

HIT Returns the sprite numbers for the collisions detected by the REPORT function. If REPORT returns a value of 3, for example, then calling HIT three times will furnish the numbers of the three crashed sprites. Starting with the lowest sprite number, HIT reports one sprite number each time it is called and promptly forgets it! It is important to note that having reported a sprite number, HIT no longer acknowledges the fact that the sprite has collided with something. A value of −1 is returned if no further collisions are available to be reported.
Examples:- X=HIT SPUTHIT IFHIT=1THEN140
FORI=1TOX:C(I)=HIT:NEXT:REM save collisions

INKEY Polls the keyboard and returns the ASCII code for any key pressed, else 0.
Examples:- X=INKEY 100 IFINKEY=89THEN300ELSE330

KEY Polls the keyboard and waits for a key to be pressed before returning with the ASCII code for the key.
Examples:- X=KEY X=KEY−48 IFKEY=32THEN100

REPORT Examines the displayed sprites and reports how many of them, if any, are in collision. It does so only for those sprites with the collision detect flag set and others are ignored. The actual sprite numbers are returned by the HIT function.
Examples:-X=REPORT IFREPORT=0THEN90 IFREPORT=1THENX=HIT

REPORT(n) Examines sprite n and returns a value of 1 (else 0) if the sprite is in collision. The sprite is tested irrespective of the condition of its collision detect flag. A zero value is returned if the sprite is not currently displayed.
Examples:- X=REPORT(N) IFREPORT(N)THENSPUTN:GOTO100

SCORE Returns the current value of SCORE.
Examples:- X=SCORE X=SCORE+100 IFSCORE>2000THEN350

SPECIAL CHARACTERS

As well as @POS, Sprite Magic offers six control characters which may be included in PRINT lists to move the cursor around under program control and to clear the screen, wholly or in part.

CHR$(8) Back-spaces the cursor one position, moving up a line if necessary.

CHR$(9) Forward-spaces the cursor, moving down a line if necessary.

CHR$(10) Moves the cursor down a line, scrolling if necessary.

CHR$(11) Moves the cursor up a line.

CHR$(12) Clears the screen and homes the cursor to screen top left. The CLEAR key performs this operation in immediate mode.

CHR$(31) Clears the screen from the current cursor position to the end of the screen. The cursor position is unaffected.

ABOUT COLOURS AND COLLISION DETECTION

When using Sprite Magic in the Dragon's four-colour modes there are certain rules to be observed:—

1. Always use the new COLOUR command, instead of COLOR.

2. Although the background colour may be any of those available, do not change it after the drawings have been generated otherwise collision detection will not work as described.

3. Always SPUT sprites at co-ordinates with an even value for the X axis. Because of the way the machine handles its video RAM, colours will not be those expected if sprites are screened at odd-numbered X co-ordinates.

4. For similar reasons, SPEED should always have an even value.

5. For maximum flexibility, it is desirable to be able to ignore collisions between certain colour combinations and Sprite Magic provides this facility. The rules here are quite straightforward and are dependent on the background colour selected:—

a) With a background colour of 1 or 4, collisions between colours 2 and 3 are ignored.

b) With a background colour of 2 or 3, collisions between colours 1 and 4 are ignored.

c) Add 4 to these values for the SCREENn,1 colour set.

The following routine demonstrates collision detection. Sprite 0, a ball, is controlled from the keyboard and a BEEP is output whenever the sprite collides with the rectangle drawn in the middle of the screen.

```
10 CLEAR200,&H6000:RESERVE0:SMODE4,1:SCREEN1,1:COLOUR0,1
20 PRINTCHR$(12)CHR$(150):DGET0,(7,7):SGET0,(0,64)
30 PCLS:LINE(50,50)-(206,142),PSET,B:SPUT0,(10,10)
40 FLETCH(0):MOVE0:IFREPORTTHENBEEP14
50 GOTO40
```

THE RUN COMMAND

Although it is possible to turn on many of Sprite Magic's facilities when working in immediate mode, everything will be reset by the RUN command. ANALOG, AUTO. . . , CHASE, FLEE, HOLD, MAZE, PAGE, SCORE and STIX will all be turned off. Additionally, SPEED will be set to its default value of two and all drawing and sprite data will be cleared.

ABOUT MEMTOP

We use the term MEMTOP to mean the address of the last byte available to Basic. Sprite Magic's usage starts at MEMTOP+1 and you will appreciate that the value assigned to MEMTOP is vital if the system is to work correctly. When using the CLEAR command to adjust MEMTOP, please note that:—

1. MEMTOP should always be less than &H6CE0.

2. Any drawings already generated will probably be lost after executing CLEAR.

3. In program mode, CLEAR should always be followed immediately by a RESERVE command, even if this means using RESERVE0. This is not necessary in immediate mode.

HOW TO SAVE THE DRAWINGS AND SPRITES

As indicated earlier, it is possible to transfer Sprite Magic's data between programs. This may be desirable either to save memory or for using the same data in a number of different programs. The procedure is as follows:—

1. SPUT any displayed sprites so that they are no longer on screen.

2. Execute the KEEPON command.

3. CSAVEM"DATA", MEMTOP,&H6F83,&HB44F. MEMTOP is as described above and &HB44F is simply a convenient re-entry to Basic which will not cause any problems in the event of an EXEC call.

4. To re-use the data in a different program, CLOADM"DATA" and KEEPON. To ensure correct operation in subsequent program runs, it may be necessary to clear the status bit of ATTR by ANDing with 127, before the sprites are displayed. The old program and the new program should both use the same value to MEMTOP.

APPENDIX A

Bit Testing. To make best use of Sprite Magic's facilities it is advantageous to understand the concept of bit testing. If you are already familiar with the binary and hexadecimal number systems, skip to AND and OR below.

Number Systems:– Consider first how the relative positions of the digits in a decimal (base 10) number such as 198 indicate their value in terms of powers of ten. Thus 198 means $1*10^2+9*10^1+8*10^0$

In binary, 198$_{10}$ would be expressed as 1010 0110 which is $1*2^7+0*2^6+1*2^5+0*2^4+0*2^3+0*2^2+0*2^1+0*2^0$. Note how the digits are arranged, highest to lowest power of two from left to right.

The hexadecimal (base 16) number system obviously requires more characters than are available in the decimal set. Rather than invent new ones, it is convenient to use the letters A through F for the additional six so that numbering continues .... 7,8,9,A,B,C,D,E,F.

Converted to hexadecimal, the number 198$_{10}$ therefore becomes C6 which is $12*16^1+6*16^0$.

Why hexadecimal ? Refer again to the binary representation of the number and note how the first and second group of four bits are represented by just a single character each in the hex version: hexadecimal is a more convenient and less error-prone way of handling binary's long strings of ones and zeros.

AND and OR operators in Microsoft Basic are very versatile. Besides their logic capabilities they also provide bitwise tests on expressions, with values ranging from -2^{15} to $2^{15}-1$.

Bitwise AND compares all bits in one number with their corresponding bits in another number. If BOTH bits are set then the corresponding bit in the answer is set. For example 37 AND 23 is 5 since in binary:–

```
37 = 0000 0000 0010 0101
23 = 0000 0000 0001 0111
 5 = 0000 0000 0000 0101
```

Bitwise OR compares bits in a similar fashion but it sets the bit in the answer if EITHER of the corresponding bits was set in the two numbers. Therefore, 37 OR 23 is 55 since in binary:–

```
37 = 0000 0000 0010 0101
23 = 0000 0000 0001 0111
55 = 0000 0000 0011 0111
```

A note here about negative values. The bitwise AND and OR operators adopt what is known as twos complement notation for negative numbers. This entails adding one to the inversion (ones complement) of a positive number. For example, the inverted binary of 37$_{10}$ is 1111 1111 1101 1010 and the twos complement is therefore 1111 1111 1101 1011. Try adding this to the binary expression of 37$_{10}$ and the result is the desired zero, plus a "carry" into the "non-existent" bit 16.

TRUE OR FALSE Clearly, a condition will be either TRUE or FALSE and Microsoft Basic assigns to expressions a value of –1 (all bits set in binary), if true and 0, if false. So, PRINTA=7 when A is some other value will return 0. If A does have a value of 7 then Basic will return –1 in the above example. Thus, the way Basic handles a line such as IFA=3ANDB=4THEN550 is to evaluate the two equations, ascertain whether they are true (–1) or false (0), perform a bitwise AND and GOTO 550, or not GOTO 550, accordingly. Suppose that A does equal 3 and that B does equal 4 then you would expect control to pass to line 550 and it does, because –1 AND –1 is –1 which is TRUE, so the test succeeds.

Whilst the proper value of TRUE is –1, Basic will also accept as TRUE any non-zero value. This permits tests such as IFN THEN. . . . where the condition will be met when N has any non-zero value. Similarly, IFN AND4THEN. . . . succeeds if binary N has bit 2 set.

The fact that Basic assigns a value of –1 to TRUE expressions also permits routines such as 10 I=I+1+7*(I=7):PRINTI;:GOTO10 where the equation inside parentheses will be evaluated to –1 when I=7 and to zero at all other times. Running this one line program will result in output of 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 . . . .

Examples of the use of these facilities are included in the description of Sprite Magic's commands and functions and further examples follow.

U=ATTR(N)AND1 sets U=1 if bit 0 of ATTR is set, else U=0. Therefore U now contains the information about sprite n's vertical motion.

R=BUTTONAND1 "sets" R if the right hand joystick button is pressed, else R=0.

ATTR(5)=ATTR(5)OR64 turns on collision detect for sprite 5, without changing any of the other elements of ATTR(5).

ATTR(G)=ATTR(G)AND191 turns off collision detect for sprite G. Binary 191 is 10111111 so ANDing with this value will clear only bit 6, leaving the others unchanged.

IFREPORT(8)THEN sprite 8 has collided with something. Remember that Basic interprets any non-zero value as TRUE.

IFATTR(1)AND128THEN sprite 1 is currently displayed.

IFREPORTTHENSPUTHIT removes the lowest-numbered of the crashed sprites from display.

DIR(3)=DIR(3)–4*(COX(3)< 20)–2*(COY(3) >171) reverses sprite 3's direction if it got to within 20 of the left hand edge of the screen, by adding 4 to DIR(3). Alternatively, or additionally, it adds 2 to DIR(3) if the sprite got to within 20 of the bottom edge of the screen. (The expressions in parentheses will be evaluated as –1 if TRUE, else 0.) This example is equivalent to two IF . . . . THEN statements.

APPENDIX B

Sprite Magic is thoroughly de-bugged and should never crash but remember that it is stored in RAM so be careful not to POKE into the reserved area.

Errors detected by Sprite Magic will be reported using the unique syntax "SMn ERROR IN LINE NNN" where n is a number in the range 1 to 7 and NNN is the current line number. Program execution will halt and the machine will return to command mode. The seven possible error codes are as follows :–

ERROR 1 Sprite Magic has been asked to handle a drawing outside the permitted range of 0 to 31. Check the drawing number used for DGET and SGET statements.

ERROR 2 A sprite is too big to be handled by DGET. Check that you have used sensible dimensions. The size of the drawing must not exceed 256 bytes of information although what this means in terms of actual size on screen depends on the graphics mode employed. To

calculate the byte size of a drawing, first of all divide the X dimension by 8. If in SMODE 0 or 2, divide again by two, but in either case add 1 for any resultant fraction. Add 1 again and multiply this by the Y dimension. This is the byte size for SMODEs 2, 3 and 4. Halve the value for SMODEs 0 and 1.
A 40 x 40 drawing in SMODE 4, for example, needs 240 bytes while a similar drawing in SMODE0 needs 80 bytes. Sprite Magic handles bigger (and more) sprites than any other system we know, but in the unlikely event that a drawing is too big, the remedy is to redesign a smaller drawing, use a lower resolution mode, or DGET the drawing in two halves. In the last case, the two halves may be assigned to two separate sprites subsequently manipulated by the program as "Siamese Twins".

ERROR 3 Sprite Magic has run out of space for drawings. The solution is to make more space available with the CLEAR command, or re-design the program to work with fewer or smaller drawings, or use a lower resolution SMODE.

ERROR 4 A non-existent drawing has been assigned to a sprite.

ERROR 5 Sprite Magic has been asked to handle a sprite outside the permitted range of 0 to 127.

ERROR 6 The start and end numbers are improper in a SMOVEn,n statement.

ERROR 7 A parameter, other than a sprite or drawing number, is outside its permitted range. Examples:– SPEED100, CHR(200)=. . . ., STIX(5)=. . . . etc.

To assist you with de-bugging we describe below a number of error symptoms and their possible cause :–

Sprites appear inside a coloured box. This may be the result of DGETing a drawing on a background colour that was not the background colour specified by the COLOUR command. Alternatively, COLOR has been used inadvertently instead of the new COLOUR command.

Sprites have "flickering bits" either side. Check the dimensions used and that the area to the right of the drawing is clear when executing DGET.

No text output on screen. If in a four colour mode, the foreground and background colours may be set to the same value – enter an appropriate COLOUR command or execute a mode change to force the default colours. The condition may also arise after a partial mode change. It is advisable always to execute the SCREEN command as well as the SMODE command, otherwise text may be output to an area of video memory that is not being displayed. Note that keyboard input is still valid, despite the absence of screen output.

A function call returns "FC ERROR". This is probably the result of the argument being outside Basic's permitted range of -2^{15} to $2^{15}-1$ (–32768 to 32767).

QUICK REFERENCE GUIDE

Sprite Statements

ANIMATEn, (drawing1,drawing2) Specifies the range of drawings to be used for sprite n on successive MOVEs.
AUTOREVON Reverses sprite directions automatically following collisions. AUTOREVOFF.
CHASEON Forces all displayed sprites to pursue sprite 0. CHASEOFF.
DGETn,(x,y) Copies drawing n, range 0–31. Drawing must be at screen top left and (x,y) define the bottom right co-ordinates of rectangle whose top left co-ordinates are (0,0).
FLEEON Forces all displayed sprites to run away from sprite 0. FLEEOFF.
FLETCH(n) Polls the cursor keys and adjusts sprite n's direction according to key(s) pressed. Should be executed immediately prior to a MOVE statement.
MAZEON Enables Maze Mode. MAZEOFF.
MOVEn Moves sprite n in accordance with attributes. If after FLETCH(n), then sprite responds to keyboard. STIX has precedence over FLETCH.
MOVEn1,n2 Moves block of sprites from n1 to n2 inclusive. Only displayed sprites are moved and sprites not on screen are ignored. Other parameters as for MOVEn.
MOVEM Moves all the sprites, as above.
NODE(n)=x,y,directions Defines maze node n at screen co-ordinates x,y. The direction(s) a sprite may take on arriving at a node are specified by the directions parameter where 1=up, 2=right, 4=down, 8=left.
SDIMS=(x1,y1)–(x2,y2) Sets screen limits to be observed by sprites programmed to disappear or rebound at screen edge.
SGETn, (d,a) Initialises sprite n, range 0–127. The sprite will use drawing d and movement attributes are specifed by a, comprising direction (0–8), wipe at screen edge (16), bounce at screen edge (32), collision detection (64) and display status (128). Direction numbers start with 1=north-east and continue clockwise round the compass in 45° steps ending with direction 8=north.
SMODEn,p Direct replacement for PMODE command where n and p select the graphics mode and memory page.
SMODEQ Exits graphics mode and returns control to standard text screen and keyboard handling routines.
SPEEDn Master control for sprite speed, range 0–15. Sets number of pixels to be moved on 256 x 192 grid.
SPUTn,(x,y) Displays sprite n at co-ordinates (x,y). SPUT has a toggling action so repeated execution turns sprite on and off.

Text Statements

@POS(column, line) Moves the print cursor to absolute position specified by column and line.
CHR$(n)=r0. . . r7 Defines the eight rows of character n, range 32–159. Each row comprises eight pixels and setting a bit in the row value sets the corresponding pixel.
COLOURforeground,background Direct replacement for COLOR. As well as controlling graphics colours, also determines text foreground and background colours.
HOLDn Fixes top n lines of the screen so that they do not scroll. HOLD0.
PAGEON Enables Paging Mode. Machine will wait for a keypress when the cursor gets to the bottom of the screen. Space-bar outputs another line, CLEAR outputs another screenful, BREAK disables. PAGEOFF.
SCOREv Outputs v in field-width of 5 characters and resets cursor to starting position. X=SCORE returns current value of SCORE.

Special Characters

CHR$(8)=Cursor left. CHR$(9)=Cursor right. CHR$(10)=Cursor down. CHR$(11)=Cursor up.
CHR$(12 Clears the screen and homes the cursor to screen top left.
CHR$(31) Clears the screen from current cursor location to end of screen.

Joystick Statements

ANALOGON Sets sprite movement proportional to joysticks for sprites 0 and 1, else handled as switches. ANALOGOFF.

STIX(n)h,v Turns on joystick control for sprites 0 or 1 in horizontal and vertical axis. STIX(0)ON,ON enables joystick control of sprite 0 in both axes. STIX(0) OFF, OFF.

STIX(n)ON,(x,y) Programs sprites 2 or 3 as missiles fired from sprites 0 and 1, with starting position offset by (x,y). STIX(n)OFF.

Sound Statements

AUTOBEEP(n)=b Sets automatic output of three of Sprite Magic's BEEPs. n=0 outputs BEEPb when sprite reaches edge of screen. n=1 outputs BEEPb when sprite 2 or 3 fired by joystick button. n=2 outputs BEEPb when sprites collide.

BEEPn Outputs preprogrammed sound effect n, range 0–15.

BEEP(period,proc,vol,vroc,cycles,type) Outputs a user-defined sound effect. Period is duration of one half-cycle, range 0–2047. Proc is period rate of change, range –128 to +127. Vol specifies start volume, range 0–255. Vroc is volume rate of change, range –128 to +127. Cycles is the number of cycles to be output, range 0–32767. Type 0 generate a square wave, type 1 generates psuedo white noise.

General Statements

BREAKOFF Disables the Break key. BREAKON.

KEEPON Stops the RUN command clearing sprite and drawing data. KEEPOFF.

RESERVEn Reserves n bytes of memory just above MEMTOP for nodes and user routines.

Sprite Functions

X=ATTR(n) Returns sprite n's attributes, sum of 1=up, 2=right, 4=down, 8=left, 16=wipe at screen edge, 32=rebound at screen edge, 64=collision detect on, 128=sprite currently displayed. ATTR(n)=X allows attributes to be adjusted.

X=COX(n) Returns sprite n's X co-ordinate.

X=COY(n) Returns sprite n's Y co-ordinate.

X=DIR(n) Returns sprite n's direction. DIR(n)=X allows sprite directions to be adjusted.

X=DRWG(n) Returns sprite n's drawing number. DRWG(n)=X replaces existing drawing with Drawing X.

X=HIT Returns the lowest crashed sprite number, else –1. May only be used after calling the REPORT function.

X=REPORT Returns the number of crashed sprites, if any.

X=REPORT(n) Returns 1 if sprite n in collision, else 0.

Joystick/Keyboard Functions

X=BUTTON Returns 1 if right hand joystick button is pressed, 2 if left hand button pressed and 3 if both pressed.

X=INKEY Returns ASCII code for keypress, else 0.

X=KEY Waits for a keypress before returning with ASCII code.

"Sprite Magic" Copyright, Knight Software 1984.
All rights reserved worldwide on "Sprite Magic".
Its name and all associated hardware, software, code, listings, audio and visual effects, graphics text and illustrations are the exclusive property of Knight Software and may not be copied, reproduced, transmitted, transfered, hired, lent, distributed, stored or modified in full or in part, in any form without the express written permission of Knight Software.