

Controlling a Sprite's Movement

Creating a Sprite

Creating a Sprite with a Transparent Background

Giving a Sprite a Velocity

Mirroring and Flipping a Sprite

Positioning a Sprite

Rotating a Sprite

Setting a Sprite's Origin

Sprite Rebound

Creating and Moving Sprites

Introduction

A **sprite** is an object designed to contain an image. We've already seen that an image can be displayed and manipulated in DarkBASIC Pro using the BITMAP or IMAGE commands. However, if we want to use an image such as a spaceship or a ball as part of a game, it may be more useful to load that image as a sprite.

Programming commands are available to allow manipulation of a sprite in ways that are not possible with other types of screen images.

Loading a Sprite Image

We cannot load an image file directly into a sprite. Instead we must begin by loading the image file into an image object. For example, we can load the *arrow.bmp* file into image 1 using the statement:

```
LOAD IMAGE "arrow.bmp", 1
```

This assumes that the file *arrow.bmp* is in the current folder on your disk.

From here the image can then be loaded into a sprite.

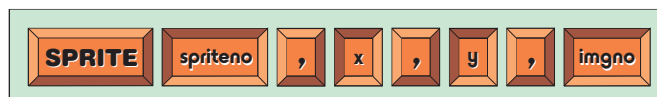
The SPRITE Statement

The SPRITE statement copies an image into a sprite component. The sprite is then displayed at a specified position on the screen. This statement has the format shown in FIG-21.1.

In the diagram:

FIG-21.1

The SPRITE Statement



spriteno is the integer value to be assigned to the sprite.

x,y is a pair of integer values specifying the position on the screen at which the sprite is to be placed. The position is given in pixels.

imgno is the integer value specifying the image to be loaded into the sprite. A picture should already have been loaded into this image.

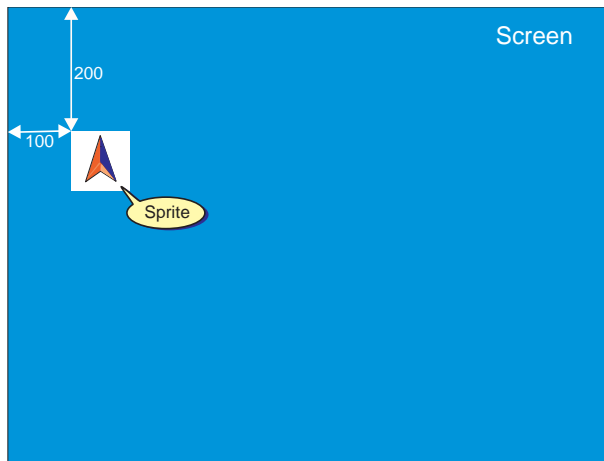
To assign image object 1 to sprite number 1 and display it at position (100,200) on the screen we would use the line:

```
SPRITE 1,100,200,1
```

The resulting display is shown in FIG-21.2.

FIG-21.2

Creating and Positioning a Sprite



Of course, this will only work if the `LOAD IMAGE` statement given earlier has already been executed by the program.

Activity 21.1

What statement would be required to load image 1 to sprite 5 and display the sprite at position (150,50)?

The program in LISTING-21.1 demonstrates the use of the `SPRITE` statement.

LISTING-21.1

Displaying a Sprite

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** End program ***
WAIT KEY
END
```

Activity 21.2

Type in and test the program given above (*sprites01.dbpro*).

The arrow sprite is 32 pixels wide by 32 pixels high, so when we request the sprite to be placed a position (100,200), just exactly which part of the sprite will be placed at that position? From FIG-21.2 above we can see that it is the top-left corner of the sprite that is placed at (100,200). To understand why this is we need to know a little more about how the coordinate system works.

We already know that the screen has its own coordinate system with the origin (0,0) at the top-left of the screen, but each sprite we place on that screen has its own local coordinate system with the sprite's origin at the top-left corner of the sprite (see FIG-21.3).

So the statement

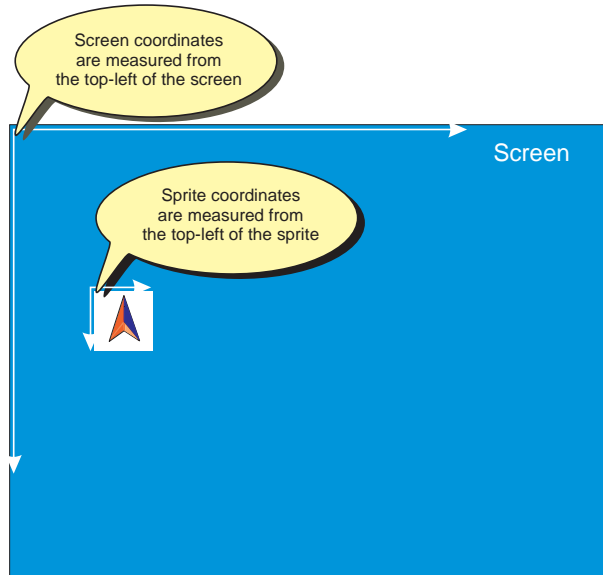
```
SPRITE 1,100,200,1
```

should be read as

Transfer image 1 to sprite 1 and draw sprite 1 on the screen with the sprite's origin at screen coordinates (100,200).

FIG-21.3

Screen Coordinates and Sprite Coordinates



Translating a Sprite

It is possible to move the sprite to a new position by simply using the `SPRITE` statement again, but this time with different coordinates. For example, the line

```
SPRITE 1,150,20,1
```

will move the arrow from its original position to the location (150,20).

Activity 21.3

In your previous program, add the following lines

```
REM *** Move arrow ***  
WAIT KEY  
SPRITE 1,150,20,1
```

immediately before the `REM *** End program ***` line.

Test your program and check that the arrow moves after you press a key.

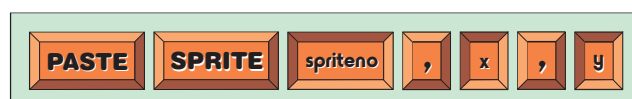
This is probably the commonest method used for moving a sprite, but other methods are available.

The PASTE SPRITE Statement

A second way to move a sprite is to use the `PASTE SPRITE` which allows an existing sprite to be moved to any position on the screen. The statement has the format shown in FIG-21.4.

FIG-21.4

The `PASTE SPRITE` Statement



In the diagram:

spriteno is the integer number of the sprite to be moved.

x,y specifies the position on the screen for the top left corner of the sprite.

For example, we can move sprite 1 to the location (450,230) using the statement

```
PASTE SPRITE 1, 420, 230
```

The program in LISTING-21.2 moves a sprite to 10 random positions with a one second delay between each move.

LISTING-21.2

Repositioning a Sprite

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Seed randomiser ***
RANDOMIZE TIMER()

REM*** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** Reposition sprite randomly ***
FOR c = 1 TO 10
  PASTE SPRITE 1, RND(600),RND(500)
  WAIT 1000
NEXT c

REM *** End program ***
WAIT KEY
END
```

Activity 21.4

Type in and test the program given above (*sprites02.dbpro*).

Modify the program so that the sprite is moved 20 times with a half second delay between each move.

The MOVE SPRITE Statement

A final way to move a sprite is to use the MOVE SPRITE statement which takes the format shown in FIG-21.5.

FIG-21.5

The MOVE SPRITE Statement



In the diagram:

spriteno is the integer number of the sprite to be moved.

distance is a real number representing the distance the sprite is to be moved. The distance is given in pixels, so if the value used contains a fraction, automatic rounding to the nearest integer will occur.

Movement can only be in an up/down direction - no sideways movement is possible. By assigning a positive value to *distance*, movement is in an upward direction; negative values cause a downward movement.

For example, the statement

```
MOVE SPRITE 1, 50.3
```

will move sprite 1 up 50 pixels (being rounded to the nearest integer) while the statement

```
MOVE SPRITE 1, -25.0
```

will move the sprite down 25 pixels.

The program in LISTING-21.3 moves the arrow sprite up 30 pixels, and then, after a key press, down 15 pixels.

LISTING-21.3

Moving a Sprite

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** Move sprite up ***
WAIT KEY
MOVE SPRITE 1, 30
REM *** Move sprite down ***
WAIT KEY
MOVE SPRITE 1, -15

REM *** End program ***
WAIT KEY
END
```

Activity 21.5

Type in and test the program in LISTING-21.3 (*sprites03.dbpro*).

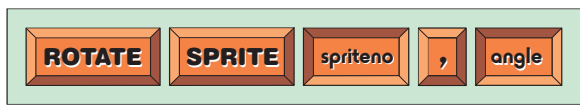
Modify the program so that the arrow is first moved up by 100 units and then down by 250 units.

The ROTATE SPRITE Statement

A sprite may be rotated to a specified angle using the ROTATE SPRITE statement. This statement has the format shown in FIG-21.6.

FIG-21.6

The ROTATE SPRITE Statement



In the diagram:

spriteno is the integer number of the sprite to be rotated.

angle is a real number representing the absolute angle (in degrees) to which the sprite is to be turned. The angle is measured from the positive x-axis in a clockwise direction.

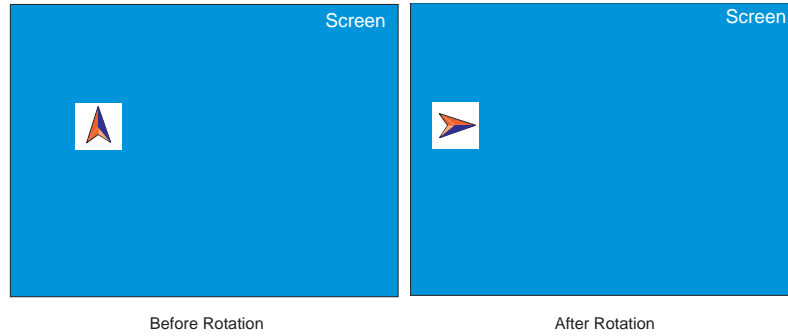
We could rotate sprite 1 to 90° using the statement

```
ROTATE SPRITE 1, 90.0
```

FIG-21.7 shows the effect of rotating the arrow sprite 90° starting from the position of (100,200).

FIG-21.7

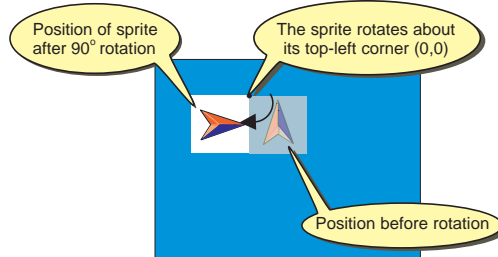
Rotating a Sprite



A close examination of the before and after screen dumps given in FIG-21.7 shows that not only has the sprite been rotated, but it also appears to have moved position on the screen. This is because the sprite rotates about its own origin (the top-left corner of the sprite). The effect of this is shown in FIG-21.8.

FIG-21.8

How a Sprite Rotates



We can see exactly how this works by running the next program (LISTING-21.4) which rotates the arrow sprite through a full 360°, one degree at a time.

LISTING-21.4

Rotating a Sprite Incrementally

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** Rotate sprite in increments ***
FOR angle = 0 TO 360
  ROTATE SPRITE 1, angle
  WAIT 10
NEXT angle

REM *** End program ***
WAIT KEY
END
```

Activity 21.6

Type in and test the program in LISTING-21.3 (*sprites04.dbpro*).

Rotating a sprite has a knock-on effect on the MOVE SPRITE statement. Once rotated, the direction in which the sprite travels in response to a MOVE SPRITE command changes. After a 90° rotation, instead of moving up or down, the sprite will now move to left or right. The program in LISTING-21.5 demonstrates this by first moving a sprite up and down and then, after rotating the sprite, moving it right and left.

LISTING-21.5

Moving a Sprite that has been Rotated

```

REM *** Set screen resolution ***
SET DISPLAY MODE 1280, 1024, 32
REM *** Load image & transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** Move sprite up and down ***
WAIT KEY
MOVE SPRITE 1, 21
WAIT KEY
MOVE SPRITE 1, -21

REM *** Rotate sprite by 90 degrees ***
WAIT KEY
ROTATE SPRITE 1, 90

REM *** Move sprite right and left ***
WAIT KEY
MOVE SPRITE 1, 40
WAIT KEY
MOVE SPRITE 1, -40

REM *** End program ***
WAIT KEY
END

```

Activity 21.7

Type in and test the program given in LISTING-21.5 (*sprites05.dbpro*).

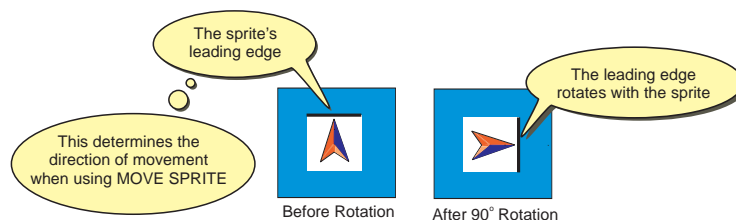
Change the rotation of the sprite from 90° to 45°. How does this effect the movement of the sprite?

How MOVE SPRITE Operates

At first glance, the effect of the MOVE SPRITE command may seem strange, but, in fact, it is quite consistent. If we consider a sprite to have a leading edge which, when initially created, is at the top of the sprite, rotating the sprite moves that leading edge (see FIG-21.9).

FIG-21.9

How Rotating a Sprite Affects the Direction in which it Moves



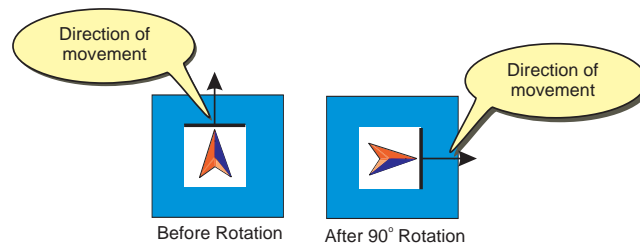
The sprite always moves perpendicular to the leading edge, hence, the line

```
MOVE SPRITE 1, 40.0
```

moves the unrotated sprite in a different direction from the rotated one (see FIG-21.10).

FIG-21.10

The Direction of Movement



Using a negative value for the distance moved, as in the line

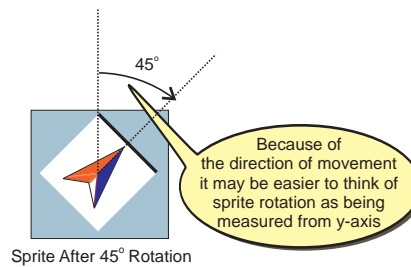
```
MOVE SPRITE 1, -40.0
```

will cause the sprite to move in the opposite direction.

Notice that the direction of travel is 270° ahead (or 90° behind) the angle of rotation of the sprite. That is to say, that when a sprite is first placed on the screen, its direction of movement is at an angle of 270° from the positive x-axis, but having rotated the sprite by 90°, the direction of movement is then 0° (90 + 270). Because of this, it is sometimes easier to think of a sprite's rotation angle as being measured from the negative y-axis (see FIG-21.11).

FIG-21.11

Measure a Sprite's Rotation from the Negative y-axis



Moving a Sprite's Origin

The OFFSET SPRITE Statement

Although a sprite's origin is initially taken as the top-left corner, that origin can be moved to a different position using the OFFSET SPRITE statement which has the format shown in FIG-21.12.

FIG-21.12

The OFFSET SPRITE Statement



In the diagram:

- spriteno* is the integer number of the sprite whose origin is to be moved.
- xoffset* is an integer value specifying how far the origin is to be moved in the x direction. The value is specified in pixels.
- yoffset* is an integer value specifying how far the origin is to be moved in the y direction. The value is specified in pixels.

In the case of the arrow sprite used earlier, this 64 by 64 pixel sprite could have its origin moved to the centre of the image using the statement

```
OFFSET SPRITE 1,32,32
```

The effect of this statement is shown visually in FIG-21.13.

FIG-21.13

Move a Sprite's Origin



Moving the origin of a sprite affects the positioning of the sprite on screen and the results produced by other operations on the sprite. The program in LISTING-21.6 creates a sprite, waits for a key press and then moves the sprite's origin.

LISTING-21.6

Changing a Sprite's Origin

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32
REM *** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1
REM *** Wait for key press then move origin ***
WAIT KEY
OFFSET SPRITE 1,32,32
REM *** End program ***
WAIT KEY
END
```

Activity 21.8

Type in and test the program given in LISTING-21.6 (*sprites06.dbpro*).

What affect does changing the origin have on the position of the sprite on the screen.

From the above Activity we saw that moving the sprite's origin caused the position of the sprite on the screen to change too. Remember that the line

```
SPRITE 1,100,200,1
```

is a request to place the sprite's origin at screen position (100,200). Since the sprite's origin has moved when the OFFSET SPRITE statement was executed, the program redraws the sprite so that its new origin is placed at (100,200). And, since a sprite rotates about its origin, the OFFSET SPRITE statement also modifies the effect of ROTATE SPRITE.

Activity 21.9

Modify the program *sprites05.dbpro* so that the sprite's origin is offset by 32 pixels along both the x and y axes before the FOR loop is executed. How is the rotation affected by this change?

A sprite's origin can be offset by amounts larger than the size of the sprite itself. In LISTING-21.7 we position the arrow sprite in the centre of the screen, offset its origin by 300 pixels in the x-axis and 200 in the y-axis and then rotate the sprite about its new origin.

LISTING-21.7

Rotating an Offset Sprite

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image and transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,640,512,1
REM *** Wait for key press then move origin ***
WAIT KEY
OFFSET SPRITE 1,300,200

REM *** Rotate sprite ***
FOR angle = 0 TO 360
  ROTATE SPRITE 1, angle
  WAIT 10
NEXT angle

REM *** End program ***
WAIT KEY
END
```

Activity 21.10

Type in and test the program given in LISTING-21.7 (*sprites07.dbpro*).

Sprite Reflection

The MIRROR SPRITE Statement

A sprite can be reflected about an imaginary vertical line drawn through the centre of the sprite using the MIRROR SPRITE statement. This statement takes the format shown in FIG-21.14.

FIG-21.14

The MIRROR SPRITE Statement



In the diagram:

spriteno

is the integer number of the sprite to be reflected about its central y-axis.

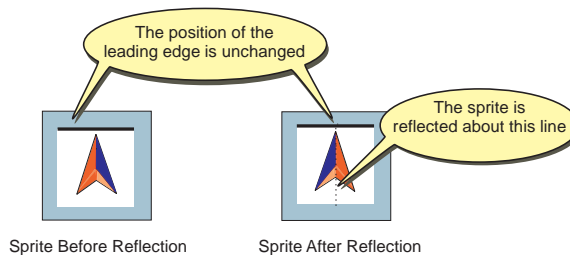
A typical statement would be

```
MIRROR SPRITE 1
```

The visual effect of such a statement is shown in FIG-21.15.

FIG-21.15

The Effect of Using MIRROR SPRITE



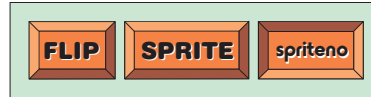
Notice that the position of the leading edge is unchanged, so any subsequent call to MOVE SPRITE will still move the sprite in an up/down direction. Also the position of the sprite is unaffected by this statement.

The FLIP SPRITE Statement

A sprite can also be reflected about an imaginary horizontal line using the statement FLIP SPRITE which has the format shown in FIG-21.16.

FIG-21.16

The FLIP SPRITE Statement



In the diagram:

spriteno

is the integer value of the sprite to be reflected about its central x-axis.

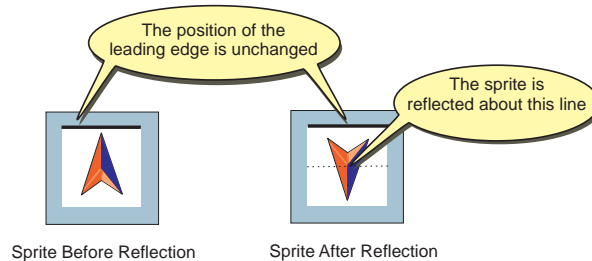
A typical statement would be

```
FLIP SPRITE 1
```

The visual effect of such a statement is shown in FIG-21.17.

FIG-21.17

The Effect of Using FLIP SPRITE



Again, the position of the leading edge is unaffected by this command.

The program in LISTING-21.8 demonstrates the use of the MIRROR SPRITE and FLIP SPRITE statements.

LISTING-21.8

Using MIRROR SPRITE and FLIP SPRITE

```
REM *** Set screen resolution ***
SET DISPLAY MODE 1280,1024,32

REM *** Load image & transfer to sprite ***
LOAD IMAGE "arrow.bmp", 1
SPRITE 1,100,200,1

REM *** Mirror sprite ***
WAIT KEY
MIRROR SPRITE 1

REM *** Move sprite ***
WAIT KEY
MOVE SPRITE 1,60

REM *** Flip sprite ***
WAIT KEY
FLIP SPRITE 1

REM *** Move sprite ***
WAIT KEY
MOVE SPRITE 1,60

REM *** End program ***
WAIT KEY
END
```