Chapter Seven: Sprites

In this chapter we will discuss one of the most important building blocks of any game: sprites. We will also begin building a new game to put our sprites to good use!

Lesson One: Sprite Concepts

What is a *sprite*? In game programming, a sprite is a graphical object on the screen that we can move and manage with our code. Sprites can be simple, fixed objects or complicated, animated, movable objects that can collide with one another. A single game can have any number of sprites. Sprites can represent your good guys, bad guys, ships, missiles, walls, and so on.

In game programming, a sprite is not a refreshing drink! A sprite can be any graphical object on the screen. A spaceship, ping-pong ball, and a rock are all possible examples of game Sprites.

Although your sprites may each look very different, you will find that writing the code behind them requires the same basic set of tasks each time. Any sprite may need to move, speed up, slow down, bounce off a wall, or display an animated image.

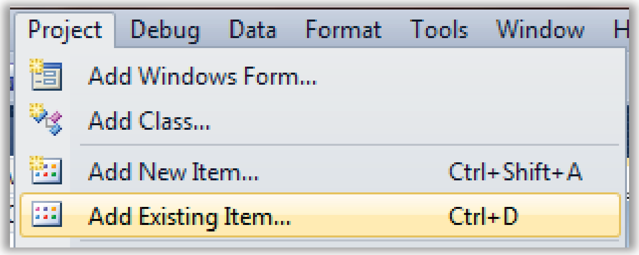
Instead of writing lots of the same code to handle each sprite individually, most game programs use a sprite library as part of their game logic. For this course we provide a sprite library for you! The sprite library defines a new **Sprite** data type you can declare as a variable. The **Sprite** contains properties to represent the object's size, position, speed, image, and other parameters. The **Sprite** also contains functions that will make the sprite move, collide, rotate, and other useful features. By using a sprite library you will be able to write programs more quickly and easily instead of re-inventing the wheel within each program!

In the next two chapters we will describe several important sprite concepts and show how those concepts are supported by our **Sprite** library. Along the way you will be developing a fully functional "Bubble Blaster" game with the techniques you have learned!

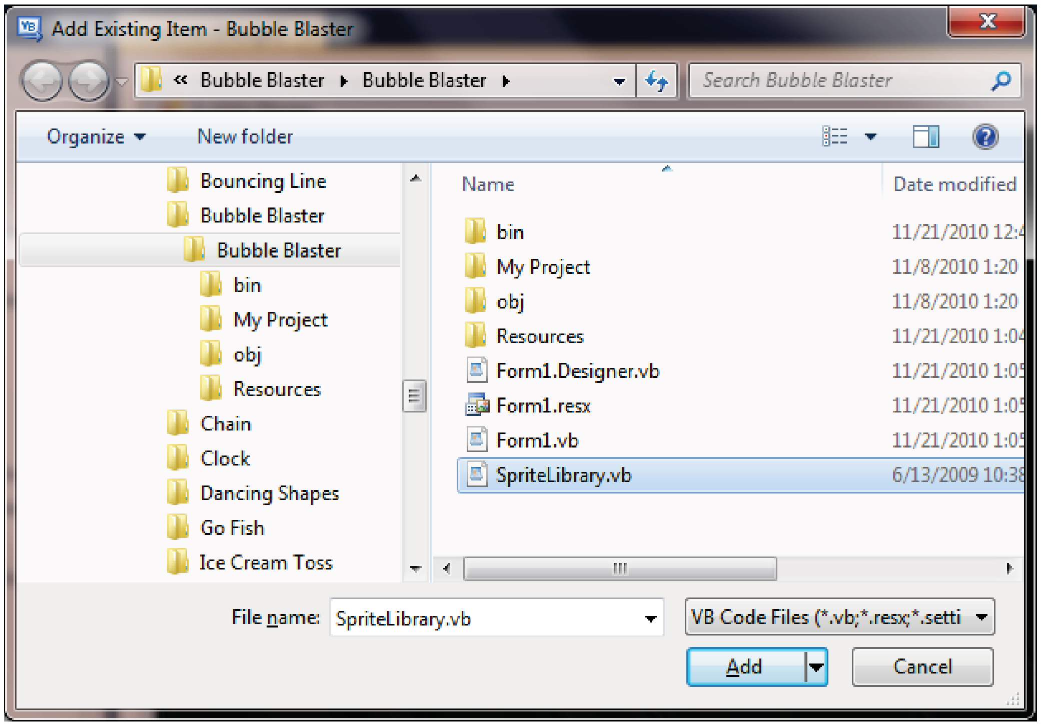
The Sprite Library

Our **Sprite** library is contained in a code file called "SpriteLibrary.vb". This file will be present in each of your Activity Starter projects. During the next few chapters we will show you how this library works. Once we are done you will have a fully-featured **Sprite** library that can be used in any game program!

To use the **Sprite** library, first you need to add the file "SpriteLibrary.vb" to your project. This is done by clicking on the Project menu and then choosing "Add Existing Item".



Then you just need to find the directory with the "SpriteLibrary.vb" file and add that file to the project.



This step is already done for you in all of the Activity Starter projects, but when you write your own games you will have to do it yourself!

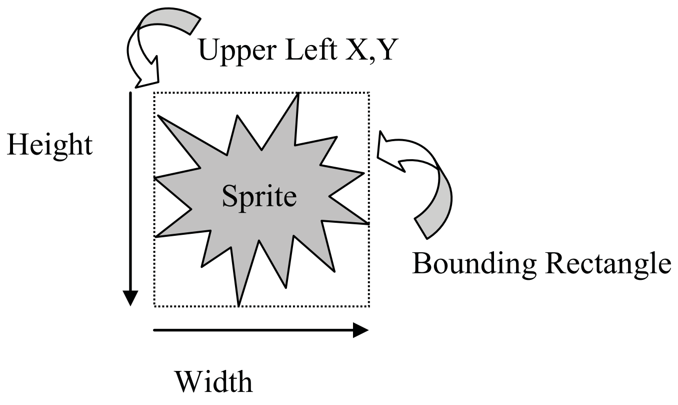
Once you have the **Sprite** library file added to the project, you can then create as many variables as you like where the data type is **Sprite**:

```
    Dim mySprite As Sprite = New Sprite()
```

Notice that you have to use the **New** keyword to create a new copy of the **Sprite** to store in the variable!

Sprite Positioning and Size

Sprites are visual objects you see on the screen. Sprites therefore must have some properties describing their position and size. The position is represented by the same X and Y coordinates we have worked with in previous chapters. Since we are representing the position of a sprite with a single pair of coordinates, we need to know which part of the sprite those coordinates represent. Sprites are usually positioned using the coordinates of the upper-left corner of the sprite. If the sprite is not a rectangle, you can mentally draw a rectangle around the outer edges of the sprite shape. This rectangle is called a bounding rectangle. The sprite is then located using the upper-left corner of the bounding rectangle.



The **Sprite** library contains a function called **GetBoundingRectangle**(), which will return a **Rectangle** with the **Sprite's** location coordinates, width, and height.

```
    Public Function GetBoundingRectangle() As Rectangle
```

Using the **GetBoundingRectangle**() function is pretty simple. For example:

```
    Dim boundingRectangle As Rectangle
    boundingRectangle = mySprite.GetBoundingRectangle()
```

This will provide you with a rectangle that represents the sprite's boundaries on the screen. Since this information is often useful in your game logic, you will find that the **GetBoundingRectangle**() function comes in real handy!

So far you have learned about two properties of a sprite: *position* and *size*. These properties are part of the **Sprite** data type as follows:

```
    Public UpperLeft As Point
    Public Size As Point
```

Both properties are **Points**, which means they have an X and Y component. The **UpperLeft** Point represents the screen coordinates of the upper left corner. The **Size** Point represents the width (X) and height (Y) of the sprite.

When you want to get or set the **Sprite** size or position just read or set the **UpperLeft** and **Size** properties with new values. In this example we set the position to (50, 100), the width to 20, and the height to 30.

```
      ' set the sprite position to (50,100)
      mySprite.UpperLeft.X = 50
      mySprite.UpperLeft.Y = 100

      ' set the sprite size to width = 20, height = 30
      mySprite.Size.X = 20
      mySprite.Size.Y = 30
```

Sometimes you may find it more convenient to get or set the sprite position in terms of the sprite's center instead of the upper-left corner. Our **Sprite** library has functions that will do that too!

```
      Public Sub SetCenter(ByVal center As Point)
      Public Function GetCenter() As Point
```

You can call those methods on your sprite objects as follows:

```
      ' get the sprite's current center location
      Dim myCenter = mySprite.GetCenter()

      ' set the sprite's new location using the center point at (200,150)
      myCenter.X = 200
      myCenter.Y = 150
      mySprite.SetCenter(myCenter)
```

Setting Sprite Images

Sprites are almost always represented as an image on the screen. This means that we must be able to set and display the Sprite's image in a game program.

The Sprite Library contains two different methods for setting the Sprite's image: **SetImage**() and **SetImageResource**(). The method that you choose to use depends on how you will be adding the images to your game program.

If you simply want to load the image from a file on your hard disk, you can use the **SetImage**() method. This method looks like this:

```
    SetImage(filename As String)
```

To use this method, you can simply pass in the name of an image on your computer, like this:

```
    mySprite.SetImage("c:\myimage.jpg")
```

The example above assumes that the "myimage.jpg" file is in the root (or main) directory of the C drive. You would have to type the correct location for any image file that you want to load. This method is similar to the **Image.FromFile**() method mentioned in the previous chapter.

If you have chosen to add your images to your project as resources, you can use the **SetImageResource**() method to set your Sprite's image. This method looks like this:

```
    SetImageResource(resourceName As System.Drawing.Image)
```

To use this method, you will pass in the name of your image resource:

```
    mySprite.SetImageResource(My.Resources.myImage)
```

This will load a resource called "myImage" into your Sprite. In general, using resources is better than using filenames for your game images. Resource files will actually become part of the game program when it is compiled and built in the IDE. This means that you do not have to worry about copying images in order to play the game on different computers.

Painting Sprite Images

In order to paint a sprite's image on the screen, you can use the **Sprite**.**PaintImage**() function. This function takes two parameters: a **Graphics** object, which represents the paintable area of the game screen and a **Boolean** value which represents the transparency of the image. If the value is **True**, the image is drawn with transparency. If the value is **False**, the image is not drawn with any transparency.

```
    PaintImage(myGraphics As System.Drawing.Graphics, transparent As Boolean)
```

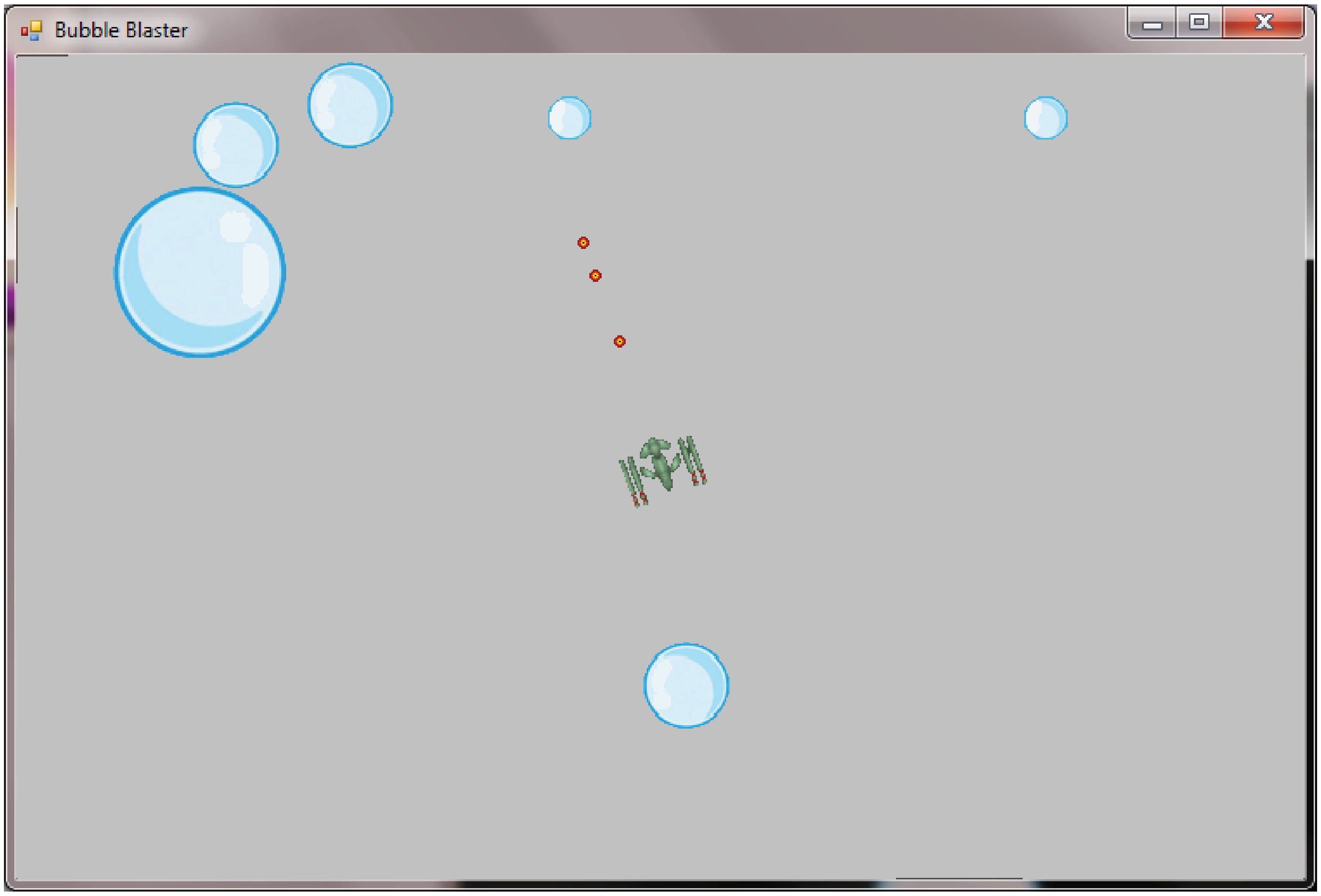
To use this method, you could use a line of code like the following one:

```
    mySprite.PaintImage(myGraphics, True)
```

This would paint the Sprite's image, with transparency, to a Graphic object called **myGraphics**. Remember that in order to use transparency, an image must be of type GIF or PNG!

Lesson Two: Introducing Bubble Blaster

Over the next few chapters you will be writing a complete game -- Bubble Blaster! This game will show off many important sprite concepts and is fun to play! Here is an example screenshot of what the finished game will look like:



Game Play

Let's first take a look at what the game of Bubble Blaster is all about.