SECTION 2

# How Your First Program Works

## How Programs Are Structured

In Section 1, you typed in and ran your first program—way to go! From now on, our job with this short cut is to explain it all to you so that you understand how and why your program works the way it does, and how you can start creating your own programs. Of course, by the end of this short cut, you will have created a complete game of your own, not just the simple example shown in Figure 2.1.

Let's note a few key things about how the instructions (or **code**) are presented in Phrogram. First, Phrogram doesn't actually use the line numbers to "run" your program. The line numbers shown at the left end of each instruction are shown there for your own reference as you work within the program. Computer programs are normally processed one instruction at a time, starting at the top and moving downward. We will discuss later how there are important exceptions to this principle in Phrogram (just as there are in other languages).

Another important point to note is that you must place each instruction on a separate line in your program. For instance, the following



**Section 2:**
**How Your First Program Works**

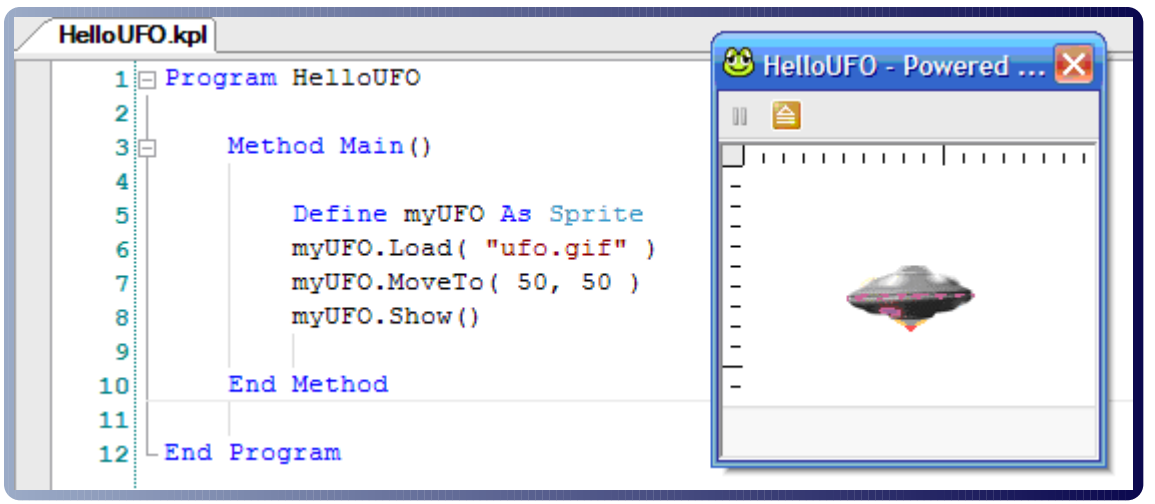



**FIGURE 2.1**
Your first Phrogram program

program contains the same instructions as the one you typed, but this one will *not* work, because each instruction is not on its own line:

```
Program HelloUFO      Method Main()      Define myUFO As Sprite
myUFO.Load( "ufo.gif" )      myUFO.MoveTo( 50, 50 )
myUFO.Show()   End Method   End Program
```

All programming languages have rules such as these that make it possible for the computer to understand your instructions. Person-to-person communication has lots of rules, too; it's just that we are so used to them that we follow them without even thinking about them. For example, we don't say "Goodbye" at the beginning of a telephone call; instead, we say "Hello"!

Just as with our telephone calls, there are specific cues that Phrogram recognizes for the start and end of a program. Thus, all programs must begin with a line such as `Program HelloUFO`, as shown

in line 1, and they all must end with `End Program`, as shown in line 12. That first line—`Program HelloUFO`—indicates that the *name* of this program is HelloUFO. You can name a program anything you want, but it's a good idea to name it something that describes what the program does.

## Method Main ( )

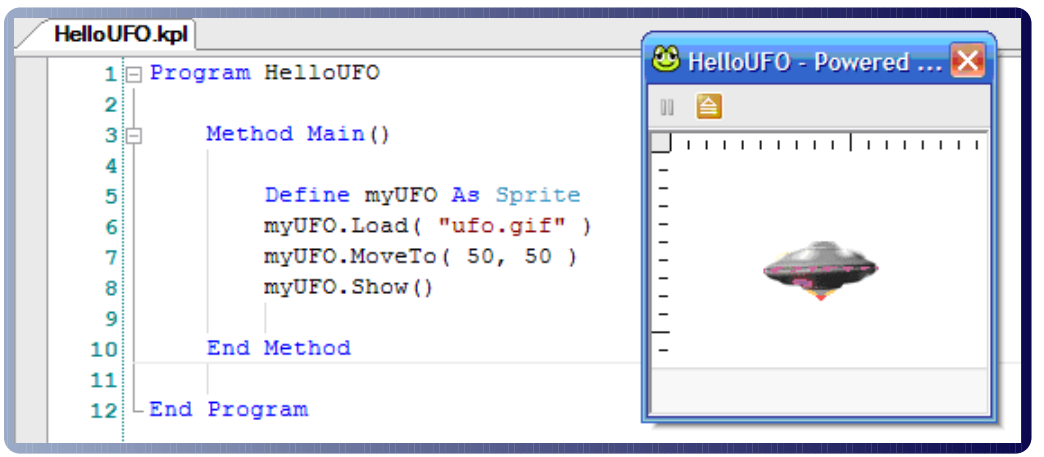
The first instruction that tells Phrogram to do something specific is *always* the first line in `Method Main()`. `Method Main()` may seem like an arbitrary way to define where the program begins processing instructions, but this is based on how all other modern programming languages work. Just as every program (or phone call!) requires a start and end point, so does every **method** inside the program, which is why you see `End Method` on line 10.

We will explain methods further in this short cut, but for now, just take it as a given when you look at the following code that these four instructions "**in**" `Method Main` tell Phrogram to display the UFO:

```
Method Main()
     Define myUFO As Sprite
     myUFO.Load( "ufo.gif" )
     myUFO.MoveTo( 50, 50 )
     myUFO.Show()
End Method
```

You've probably wondered about the blank lines that are mixed in with the instructions. Blank lines are not actual instructions—Phrogram just ignores them. You can just as easily remove blank lines if you prefer, but we don't recommend that. We think you will find that using blank lines and indentation, as we show them in Figure 2.2, will help you and other people read and understand your program more easily.

**FIGURE 2.2**
Blank lines can group instructions logically, and make them easier to read




Let us summarize, in English, what the instructions on lines 5 through 8 tell Phrogram to do, and then we will examine the instructions in detail:

- `myUFO` is a graphical **object**.
- Load the graphics picture for `myUFO` from the image file `"ufo.gif"`.
- Move `myUFO` to the screen location (50, 50).
- Show `myUFO` on the screen.

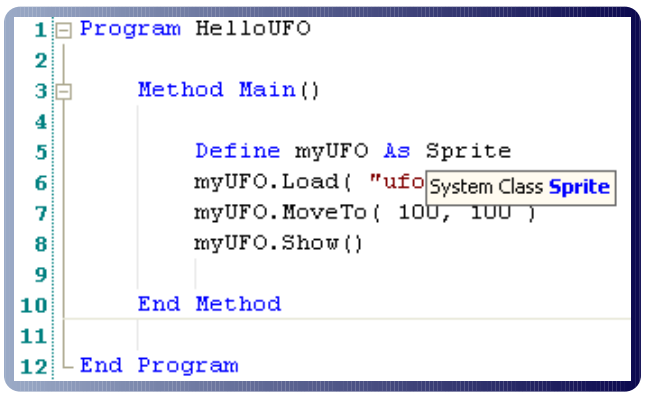
# Using Sprites to Display Graphics for Games

```
Define myUFO As Sprite
```

As we noted, the preceding instruction simply tells Phrogram that `myUFO` is a graphical object. Phrogram needs to know what `myUFO` is so that it knows the kinds of things it can do with `myUFO`.

But, you may ask, what is a **sprite?** Drag your mouse over the word *Sprite* in your program and you will see a descriptive tool tip pop up: System Class Sprite (see Figure 2.3). Try dragging your mouse over other parts of your program, and you will see a few other tool tips pop up. We will explain those soon, but for now, let's explain the System Class Sprite tool tip.

**FIGURE 2.3**
A tool tip identifying System Class Sprite




A system class is a kind of object that Phrogram provides to you, and that makes it easy for your programs to do certain things. Different system classes are useful for different things. Sprites are particularly useful for displaying graphical objects on the screen, such as UFOs or kittens or cars or anything else you might like to include in your game. Another system class that will we show you how to use soon is the sound system class. We are sure you can guess the kinds of things the sound system class lets your program do!

The next instruction, on line 6, tells Phrogram to fetch a file called `"ufo.gif"` and use the picture in that file to display `myUFO`:

```
myUFO.Load( "UFO.GIF" )
```

Where does Phrogram find this file? Phrogram automatically finds files in the Media Files subfolders under the My Phrogram Files folder. There are separate subfolders there for images and for sounds. If you want to use your own images or sounds, just drop them into the appropriate folder, and then you can use them by name just like you used `"ufo.gif"`. Phrogram will also automatically find a media file if it is stored in the same folder where you save your program.

The rules about using `Load` are not difficult, but they are exact! `Load` requires only one "value" for it to work: It requires the name of the image file Phrogram will use for this sprite—in our case, `"ufo.gif"`. This value must be surrounded by double quotation marks, as shown. Here are two examples that show incorrect ways of telling Phrogram to load this sprite's image. Let us repeat: These two instructions will not work because they do not follow Phrogram's rules about using `Load()`. Can you figure out what to change to make these work?

```
myUFO.Load( 'UFO.GIF' )
myUFO.Load( UFO.GIF )
```

In order to explain the next instruction in our program, on line 7, we will bring up a very important new topic, your computer's coordinate system:

```
myUFO.MoveTo( 50, 50 )
```

## Coordinate System

Think of your computer screen as though it is a "playing field," like for soccer or football. If you think about the way computer games move objects around on the screen, a playing field is a very good analogy. You are going to move your game objects around the screen just like players move around on a playing field. Maybe you'll even make your own football or soccer game in Phrogram!

In computing, the playing field that is your computer screen is known as a **coordinate system.** You have probably already studied coordinate systems, when you studied graphs in algebra. But a computer's coordinate system is different from the algebraic coordinate system we learn about in school, in two important ways. First, your computer screen is fairly small and has definite edges, so it does not pretend to cover a potentially "infinite" number of points along an x- (horizontal) and y- (vertical) axis. Instead, it is the finite set of locations that are viewable on your screen. Second, instead of moving upward, as in algebraic graphing, the y-axis on computer screens moves downward. This difference makes computer and algebraic coordinate systems inconsistent, but it is based on how TVs and computer monitors have been built for the past 60 years, so it's customary to use our coordinate system this way. It's not hard once you get used to it.

To be specific, your computer uses an (x, y) coordinate system for locations on the screen in which the left edge of your screen defines x = 0, and the top of your screen defines y = 0. This means that the **origin,** where x = 0 and y = 0, is the upper-left corner of your screen. Moving to the **right** across the screen increases the x value, and moving **down** the screen increases the y value.

If you are not used to thinking about coordinate systems, all of this may seem pretty complicated, and it might be easier to grasp with a picture. Figure 2.4 shows where a Phrogram program displays several (x, y) locations.

The first value of each pair is the **x value** of that location on the screen—and as you can see, the x value increases as you move from left to right. The second value is the y value of the location on the screen, and that value increases as you move from top to bottom.

Now, let's take a closer look at the instructions in our `myUFO` program to better understand how the coordinate system works. Notice that in line 7, the instruction reads `myUFO.MoveTo( 50, 50 )`. The (x, y) location of (50, 50) places the sprite 50 pixels to the right along the x-axis, and 50 pixels

downward along the y-axis. As you can see, that is not very far from the upper-left corner of the screen. We don't often think about individual pixels on our computer screens, but in fact, everything displayed there is composed of many small **pixels**, or points of color. The window that Phrogram launches when we run a program is, as you can see from Figure 2.4, more than 500 pixels wide and more than 400 pixels high.

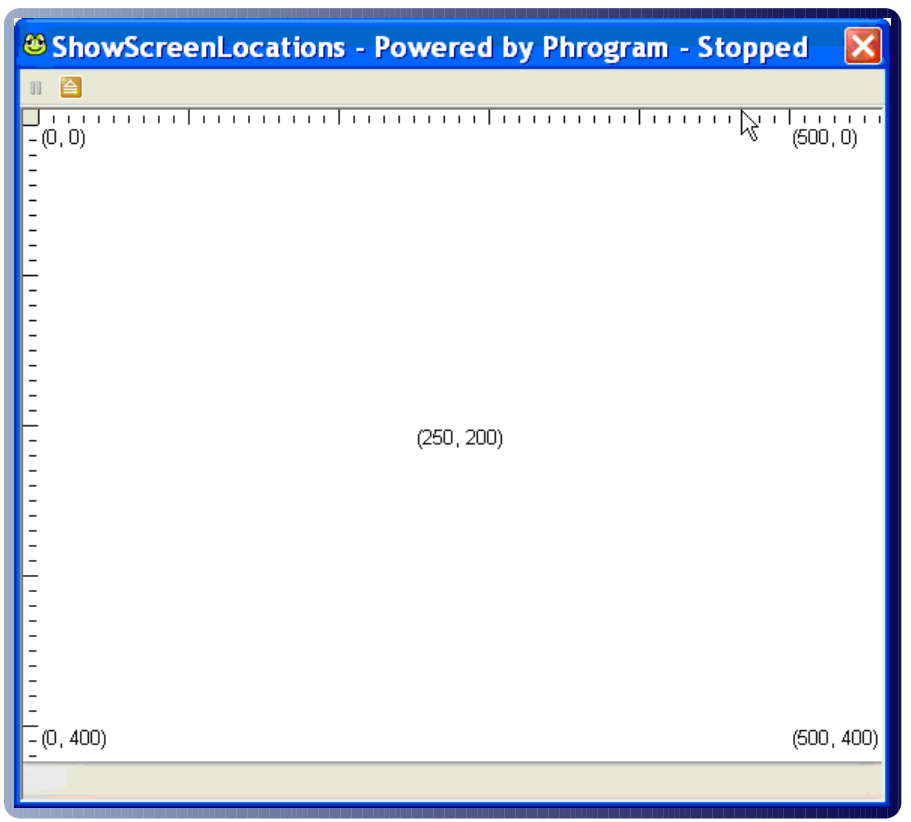



**FIGURE 2.4**
Phrogram example showing several (x, y) screen locations

It may help you understand this if you experiment with changing the location of your UFO and running the program again. Try that a few times, by clicking and changing (50, 50) to other values, such as (300, 50), and then running your program again. The way the coordinate system works is very important—and this applies directly to many programs you will create in the future, even if you use languages other than Phrogram. So, you should experiment with this, and reread this section, until it makes sense.

The last of our instructions is the simplest. After we have told Phrogram about `myUFO`, where to get its picture, and where to display it on the screen, we simply need to tell Phrogram to show it on the screen. That's easy:

```
myUFO.Show()
```

## Media Files

Let's pause here to consider that two primary types of media files are used in programming games: image files and sound files. Phrogram also uses 3D objects, but those are the subject of the next short cut, not this one—so, let's just think of images and sounds. You're already very familiar with how things move on the screen when you play a game, and of course, how music and sound effects happen when you play a game. Now that you're beginning to program games *yourself,* you get to see how this actually works from the computer's point of view. A game programmer makes a game work over time by telling the computer where to display images on the screen, and when to play sounds. There is a lot of interesting logic that you will learn as you do this, but it's good to keep in mind that what we're doing is fairly simple: We are telling the computer where and how to display game images, and we will soon also tell the computer when to play game sounds.