shortcut

Your Short Cut to Knowledge

Learn to Program with Phrogram™!

A Guide to Learning Through Game Programming Using the Latest Version of Kids Programming Language

Jon Schwartz
Walt Morrison
David Witus

Addison-Wesley
Pearson Education

www.awprofessional.com

http://phrogram.com/

Visit us on the Web: www.awprofessional.com

Pearson Education, Inc.
Rights and Contracts Department
75 Arlington St., Suite 300
Boston, MA 02116
Fax: (617) 848-7047

ISBN 13: 978-0-321-49683-6
ISBN 10: 0-321-49683-3

First release, February 2007

click the **File** menu and then select **Save As New Program.** When the dialog comes up, save this new program with the filename UFOMovingWithSound. Once you save it, you should see the new file appear in Phrogram's File Explorer.

Now, let's recap briefly what we learned in Section 2, looking only at those instructions, which are also included in the new example we're doing in this section:

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

We told Phrogram to load the image for a sprite named `myUFO` from the file `"ufo.gif"` with this instruction: `myUFO.Load( "ufo.gif" )`. We told Phrogram where to place the sprite on the screen with the instruction `myUFO.MoveTo( 50, 50 )`. Notice here that there are quotation marks around the words `"ufo.gif"`, but not around the numeric values 50 and 50. In general, Phrogram requires quotation marks around values that are words (also known as **string** values). Phrogram does not require quotation marks around values that are numbers (also known as **integer** and **decimal** values). Other programming languages use the same or a similar convention.

Now let's look at the new instructions we are adding:

```
PlaySound("eerie.wav")
Define ufoY As Integer
For ufoY = 51 To 150
     Delay(49)
     myUFO.MoveTo(50, ufoY)
Next
```

The first instruction is the easiest, and it is pretty obvious: `PlaySound("eerie.wav")` tells Phrogram to play the sound that is stored in the sound file `"eerie.wav"` through the computer's speakers. Note that the same rule applies for this filename: It must have double quotes around it, as shown. As the name of the sound file suggests, it is indeed an eerie kind of sound, like from a scary, old, science fiction movie. Many additional sounds are available in Phrogram, under the Media Files folder in the Sounds subfolder. Also, hundreds of additional sounds are available for you to download for free, from the Phrogram community Web site. You can also record your own sounds and add them to the same Sounds subfolder. But this eerie one seemed very appropriate as our UFO flies down for a landing.

Variables Let You Control Your Game Over Time

Now we want our UFO to "land" by moving down to the bottom of the screen. To do this, we define and use a **variable,** which is one of the most important concepts for a beginning programmer to learn.

As the name suggests, they are called variables because they define a value that can change (or vary!) over time. Changing over time is in fact what makes variables powerful and useful, as we will see in this example. Variables should be defined with a name that makes sense given how they will be used. In this case, our variable will be used to change the y-axis location of our UFO, in order to make the UFO move down the screen, so let's call it `ufoY`. Here is the line of code that defines our variable:

```
Define ufoY As Integer
```

`Define` is a keyword you have seen before, but this time, instead of using it to define a sprite, you are using it to let Phrogram know that `ufoY` is an integer. As you may recall from your math class,

an integer is any whole number (one that does not have a fraction or decimal). We have not yet told Phrogram which whole number value to use for `ufoY`, but because we defined it as `Integer`, Phrogram knows that its value will be a whole number when we give it one.

An `Integer` variable, as a fundamental building block of programming in Phrogram, is simpler than a sprite, which we know is a system class. A variable defined as a sprite—in our case, `myUFO`—can be manipulated with commands such as `MoveTo()` and `Show()`. But an integer is simply a number that you can use for just about any purpose for which whole numbers are useful. In our example, we will use the integer value of `ufoY` to move the `myUFO` sprite down the screen.

Loops Tell Phrogram to Do Something Over and Over Again

We know that `myUFO` starts at location (50, 50) on the screen, because that is where we told Phrogram to put it. To move it down the screen, we increase its y-axis location—to (50, 51), then (50, 52), then (50, 53), then (50, 54), and so on. It's up to us how far we want it to go, so let's decide that the UFO will stop when it gets to (50, 150).

Do you see the pattern there? The x value of the sprite's location is always 50 and the y value is going up one pixel at a time, from 51 to 150. Here is the Phrogram code that tells Phrogram to increase the value of the variable `ufoY` from `1` to `150`, one step at a time:

```
For ufoY = 51 To 150
     myUFO.MoveTo(50, ufoY)
Next
```

This is your first **loop.** Loops are another important programming concept that may stretch your brain a bit, but once you understand them, loops are easy—and very powerful for making games and other kinds of programs.

Our loop starts with the value of `ufoY = 51`, and because we also said `To 150`, we know the loop will stop after it gets to the value of `ufoY = 150`. Notice the keyword `Next`, which defines the end of the loop. When Phrogram gets to the keyword `Next`, Phrogram knows that it is time to go back up to the "top" of the loop and increase the value of `ufoY`—from 51 to 52, or from 52 to 53, or from 53 to 54, and so on, all the way to 150. Basically, our `For` loop tells Phrogram to count from 51 to 150, and Phrogram uses the variable `ufoY` to keep track of this value as it counts.

And what do we want Phrogram to do while it is counting? We want it to move the UFO down the screen, right? That's what `myUFO.MoveTo( 50, ufoY )` tells Phrogram to do. Because this instruction is "inside" the `For` loop, Phrogram performs this instruction *each time it counts* from 51 to 150. This is the real power of a loop! Just counting from 51 to 150 is not that interesting, but if the program is actually *doing something useful each time it counts,* all kinds of cool things can happen, including the movement of our UFO down the screen.

Let's recap what is happening:

```
For ufoY = 51 To 150
     myUFO.MoveTo( 50, ufoY )
Next
```

We have already seen how `MoveTo()` works: Phrogram moves `myUFO` to the (x, y) location we give it. What's different this time? Before, we moved the UFO to (50, 50)—two specific numbers. What happens when we do this in the loop, and we instead use our variable `ufoY`? Remember that the

first time through the loop, the value of `ufoY = 51`, and the second time `ufoY = 52`, then `ufoY = 53`, then `ufoY = 54`, and so on, all the way to `ufoY = 150`. So, the first time through the loop, Phrogram uses the value of `ufoY = 51` to move the sprite so that the instruction Phrogram performs is effectively:

```
myUFO.MoveTo( 50, 51 )
```

And the next time through the loop, `ufoY = 52`, so Phrogram performs:

```
myUFO.MoveTo( 50, 52 )
```

And so on, like this, until Phrogram has used the loop to count all the way to 150:

```
myUFO.MoveTo( 50, 53 )
myUFO.MoveTo( 50, 54 )
myUFO.MoveTo( 50, 55 )
...
myUFO.MoveTo( 50, 150 )
```

Each time through the `For` loop, `MoveTo` is moving `myUFO` one pixel down the screen—just like we want it to! And because our loop is `To 150`, Phrogram will stop counting and looping after it reaches 150.

So now, we have studied variables and loops for the first time, and how useful it can be to perform instructions inside a loop—these are all *very important* programming concepts. Can you think of any real-world analogies that you use like a loop? Perhaps you know exactly how many steps there are from the first to the second floor in your house. Or perhaps you count the number of times you stir the ingredients in your favorite recipe.

If the idea of a loop is not clear enough yet, please go back to the beginning of this section and reread it. If after doing that something still isn't clear, remember that you can ask questions on the discussion forums of the Phrogram Web site.

Delay

Let's explain one final detail to finish our program: the instruction `Delay(49)` (see Figure 3.3).

FIGURE 3.3
Using Delay() to control the movement of our UFO