Mike Geig

In Full Color

Fourth Edition

Sams Teach Yourself

Unity® Game Development

in 24 Hours

Mike Geig

Sams **Teach Yourself**

Unity Game Development in 24 Hours

SAMS

CAUTION

New to Programming

If you have never programmed before, this lesson might seem strange and confusing. As you work through this hour, try your best to focus on how things are structured and why they are structured that way. Remember that programming is purely logical. If a program is not doing something you want it to, it is because you have not told it how to do it correctly. Sometimes it is up to you to change the way you think. Take this hour slowly and be sure to practice.

Scripts

As mentioned earlier in this hour, using scripts is a way to define behavior. Scripts attach to objects in Unity just like other components and give them interactivity. There are generally three steps involved in working with scripts in Unity:

1. Create the script.
2. Attach the script to one or more game objects.
3. If the script requires it, populate any properties with values or other game objects.

The rest of this lesson discusses these steps.

Creating Scripts

Before creating scripts, it is a good idea to create a Scripts folder under the Assets folder in the Project view. Once you have a folder to contain all your scripts, simply right-click the folder and select **Create** > **C# Script**. Then give your script a name before continuing.

NOTE

Scripting Language

Unity allows you to write scripts in C#. In the past, however, JavaScript and a language called Boo were supported. These two other languages were phased out over the years in order to focus on a common language and common functionality. Don't worry, though: C# is an incredibly powerful and flexible language.

Once a script is created, you can view and modify it. Clicking a script in the Project view enables you to see the contents of the script in the Inspector view (see Figure 7.1). Double-clicking the script in the Project view opens your default editor, where you can add code to the script. Assuming that you have installed the default components and haven't changed anything, double-clicking a file opens the Visual Studio development environment (see Figure 7.2).

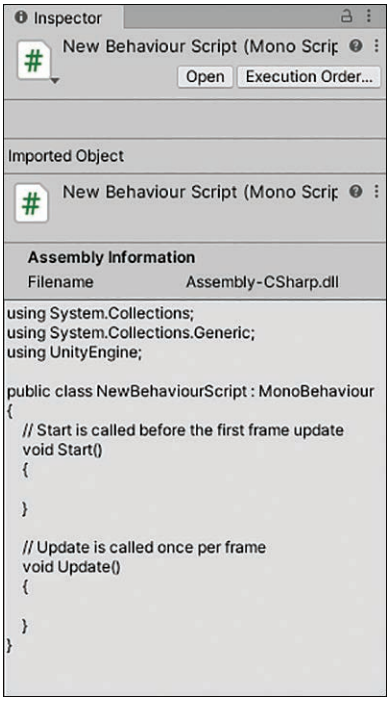

FIGURE 7.1
A script in the Inspector view.

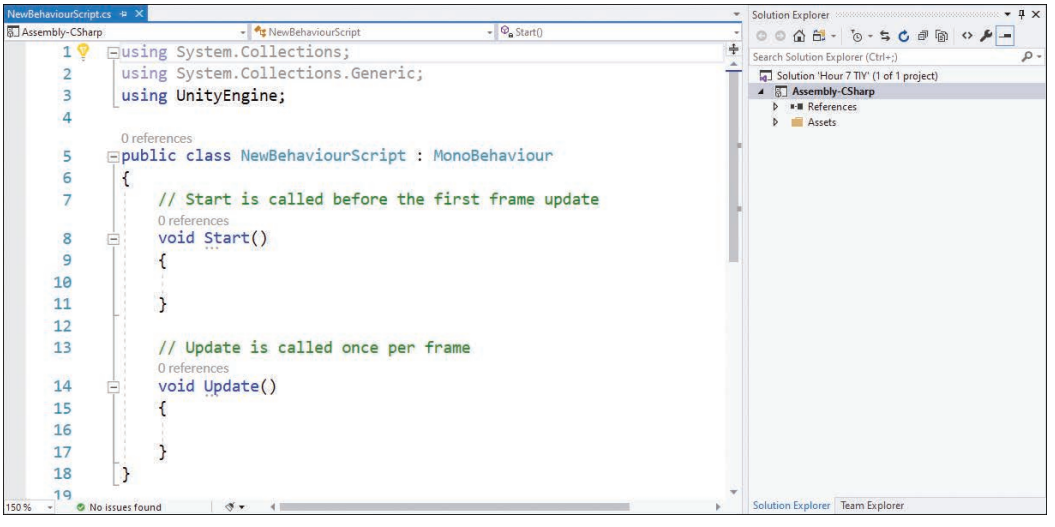

FIGURE 7.2
The Visual Studio software with the editor window showing. (Image is a copyright of Microsoft Corporation.)

NOTE

IDEs

Visual Studio is a robust and complex piece of software that comes bundled with Unity. Editors like this, known as IDEs (integrated development environments), assist you with writing the code for games. Since IDEs are not actually part of Unity, this book does not cover them in any depth. The only part of Visual Studio you need to be familiar with right now is the editor window. If there is anything else you need to know about IDEs, you will learn about it in the hour where you need it. (Note: Prior to Unity 2018.1, an IDE called MonoDevelop also came packaged with Unity. You can still acquire and use this software separately, but MonoDevelop is no longer shipped with the engine.)

▼ TRY IT YOURSELF

Creating a Script

Follow these steps to create a script for use in this section:

1. Create a new project or scene and add a **Scripts** folder to the Project view.
2. Right-click the **Scripts** folder and choose **Create > C# Script**. Name the script **HelloWorldScript**.
3. Double-click the new script file and wait for Visual Studio to open. In the editor window of Visual Studio (refer to Figure 7.2), erase all the text and replace it with the following code:

```
using UnityEngine;
public class HelloWorldScript : MonoBehaviour
{
    // Use this for initialization
    void Start ()
    {
        print ("Hello World");
     }

     // Update is called once per frame
     void Update ()
     {

      }
}
```

4. Save your script by selecting **File > Save** or by pressing **Ctrl+S** (**Command+S** on a Mac). Back in Unity, confirm in the Inspector view that the script has been changed and run the scene. Notice that nothing happens. The script was created, but it does not work until it is attached to an object, as discussed in the next section.

NOTE

Script Names

You just created a script named HelloWorldScript; the name of the actual script file is important. In Unity and C#, the name of the file must match the name of the class that is inside it. Classes are discussed later in this hour, but for now, suffice it to say that if you have a script containing a class named MyAwesomeClass, the file that contains it will be named MyAwesomeClass.cs. It is also worth noting that a class name, and therefore a script filename, cannot contain spaces or begin with a number or special character.

TIP

Easy Script Names

As mentioned earlier, the name of the script file must match the name of the class contained in the script. This means that if you create a script named MyAwesomeScript, you will need to open the script and change the name of the class to match (as discussed in more detail later in this hour). There is a simpler way to handle this, however. When you first create a new script in Unity, the file is ready to be renamed. If you immediately type the name of the script without clicking away from the script name, Unity automatically renames the class inside the script to match. This can save a bit of time and a bunch of headache!

Attaching a Script

To attach a script to a game object, just click the script in the Project view and drag it onto the object (see Figure 7.3). You can drag the script onto the object in the Hierarchy view, the Scene view, or the Inspector view (assuming that the object is selected). Alternatively, you can select the object you want to attach a script to and click **Add Component** > **Scripts** and select the desired script from the list of available scripts. Once attached to an object, the script becomes a component of that object and is visible in the Inspector view.

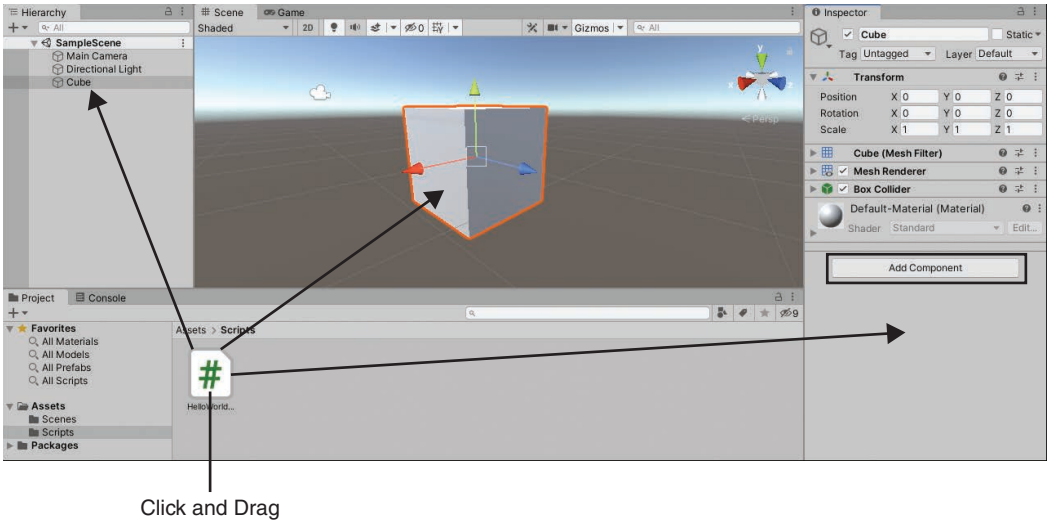

FIGURE 7.3
Clicking and dragging the script onto the desired object.

To see this in action, attach the script HelloWorldScript that you created earlier to the Main Camera. You should now see a component named Hello World Script (Script) in the Inspector view. If you run the scene, you see Hello World appear at the bottom of the editor, underneath the Project view (see Figure 7.4).

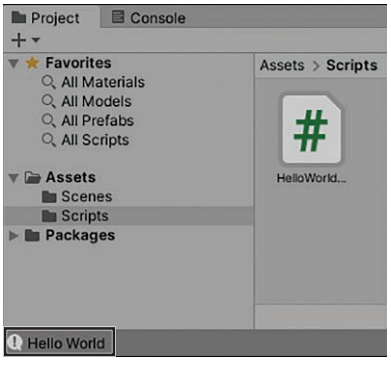

FIGURE 7.4
The words Hello World output when you run the scene.

Anatomy of a Basic Script

In the preceding section, you modified a script to output some text to the screen, but the contents of the script were not explained. In this section, you'll look at the default template that is applied to every new C# script. Listing 7.1 contains the full code that is generated for you by Unity when you make a new script named HelloWorldScript.

LISTING 7.1 Default Script Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class HelloWorldScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

This code can be broken down into three parts: the `using` section, the class declaration section, and the class contents.

The `using` Section

The first part of the script lists the libraries (that is, collections of other behind-the-scenes code) that the script will be using. It looks like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

You don't change this section too often and should just leave it alone for the time being. These lines are usually added for you when you create a script in Unity. The System.Collections and System.Collections.Generic libraries are optional and are often omitted if the script doesn't use any functionality from them.