HOUR 22

Mobile Development

What You'll Learn in This Hour:

- How to prepare for mobile development
- How to use a device's accelerometer
- How to use a device's touch display

Mobile devices such as phones and tablets are becoming common gaming devices. In this hour, you'll learn about mobile development with Unity for Android and iOS devices. You'll begin by looking at the requirements for mobile development. From there, you'll learn how to accept special inputs from a device's accelerometer. Finally, you'll learn about touch interface input.

NOTE

Requirements

This hour covers development for mobile devices specifically. So, if you do not have a mobile device (iOS or Android), you will not be able to follow along with any of the hands-on exercises in this hour. Don't worry, though: What you read here should still make sense, and you will still be able to make games for mobile devices but just won't be able to play them.

Preparing for Mobile

Unity makes developing games for mobile devices easy. You will also be happy to know that developing for mobile platforms is almost identical to developing for other platforms. The biggest difference to account for is that mobile

for other platforms. The biggest difference to account for is that mobile platforms have different input (no keyboard or mouse, usually). If you approach your development with this in mind, however, you can build a game once and deploy it everywhere. There is no longer any reason you can't build your games for every major platform. This level of cross-platform capability is unprecedented. Before you can begin working with mobile devices in Unity, however, you need to get your computer set up and configured to do it.

NOTE

Multitudes of Devices

There are many different types of mobile devices. At the time this book was published, Apple had two classes of game-ready mobile devices: iPad and iPhone/iPod. Android had an untold number of phones and tablets, and many Windows mobile devices were available as well. Each of these types of devices has slightly different hardware and requires slightly different configuration steps. Therefore, this lesson simply attempts to guide you through the installation process. It would be impossible to write an exact guide that would work for everyone. In fact, several guides by Unity, Apple, and Android (Google) already exist that explain the process better than this text could. You are referred to them when needed.

Setting Up Your Environment

Before even opening Unity to make a game, you need to set up your development environment. The specifics of this differ depending on your target device and what you are trying to do, but the general steps are as follows:

1. Install the software development kit (SDK) of the device you are targeting.
2. Ensure that your computer recognizes and can work with that device (though this is important only if you want to test on the device).
3. For Android only, tell Unity where to find the SDK.

If these steps seem a bit cryptic to you, don't worry. Plenty of resources are available to assist you with these steps. The best place to start is with Unity's own documentation, which is available at http://docs.unity3d.com. This site contains living documentation about all facets of Unity.

As you can see in Figure 22.1, the Unity documentation has guides to assist you

in setting up both the iOS and the Android environments. These documents are updated as the steps to set up the environments change. If you're not planning on following along with a device, continue on to the next section. If you are planning on following along with a device, complete the steps in the Unity documentation to configure your development environment before continuing on to the next section.

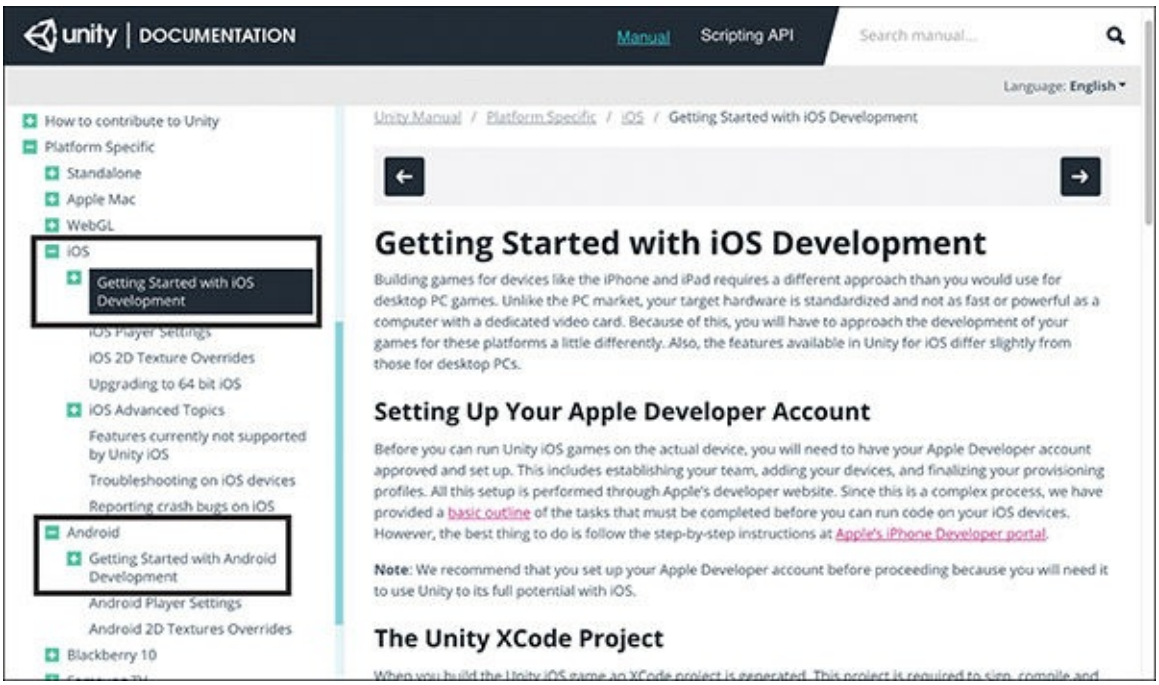

FIGURE 22.1
Platform-specific documentation.

Unity Remote

The most basic way to test your games on a device is to build your projects, put the resulting files on the device, and then run your game. This can be a cumbersome system and one you're sure to tire of quickly. Another way to test your games is to build a project and then run it through an iOS or Android emulator. Again, this requires quite a few steps and involves configuring and running an emulator. These systems can be useful if you are doing extensive testing on performance, rendering, and other advanced processes. For basic testing, though, there is a much better way: Use Unity Remote.

Unity Remote is an app you can obtain from your mobile device's application store that enables you to test your projects on your mobile device while it is running in the Unity editor. In a nutshell, this means you can experience your

game running on a device in real time, alongside development, and use the device to send device inputs back to your game. You can find more information about Unity Remote at https://docs.unity3d.com/Manual/UnityRemote5.html.

To find the Unity Remote application, search for the term *Unity Remote* in your device's application store. From there, you can download and install it just as you would any other application (see Figure 22.2).

Once installed, Unity Remote acts as both a display for your game and a controller. You will be able to use it to send click information, accelerometer information, and multi-touch input back to Unity. It is especially effective because it gives you the ability to test your game (minimally) without installing any mobile SDKs or specialized developer accounts.

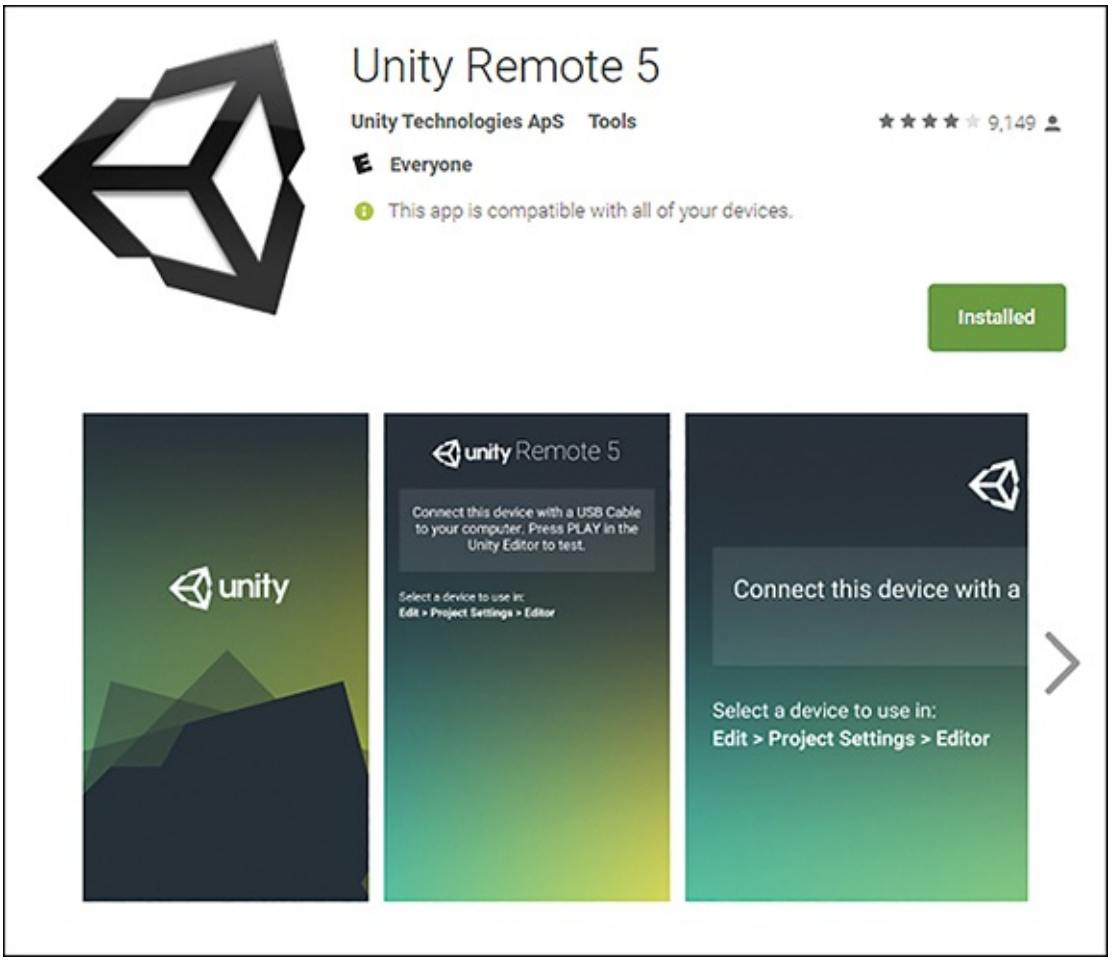

FIGURE 22.2
Unity Remote on the Google Play store.

▼ TRY IT YOURSELF

Testing Device Setup

This exercise gives you an opportunity to ensure that your mobile development environment is set up correctly. In this exercise, you'll use Unity Remote from your device to interact with a scene in Unity. If you don't have a device set up, you won't be able to perform all these steps, but you can still get an idea of what's happening by reading along. If this process doesn't work for you, it means that something with your environment is not set up correctly. Follow these steps:

1. Create a new project or scene and add a UI button to the center of the screen.
2. Set the button's Pressed Color to **Red** in the Inspector.
3. Run the scene and ensure that clicking the button changes its color. Stop the scene.
4. Attach your mobile device to your computer with a USB cable. When the computer recognizes your device, open the **Unity Remote** app on your device.
5. In Unity, select **Edit > Project Settings > Editor** and choose your device type in the Inspector under **Unity Remote > Device**. Note that if you don't have a mobile platform installed for Unity (Android or iOS), the option for it will not appear in the Editor settings.
6. Run the scene again. After a second, you should see the button appear on your mobile device. You should now be able to tap the button on your device's screen to change its color.

Accelerometers

Most modern mobile devices come with built-in accelerometers. An *accelerometer* relays information about the physical orientation of the device. It can tell whether the device is moving, tilted, or flat. It can also detect these things in all three axes. Figure 22.3 shows a mobile device's accelerometer axes and how they are oriented in *portrait orientation*.

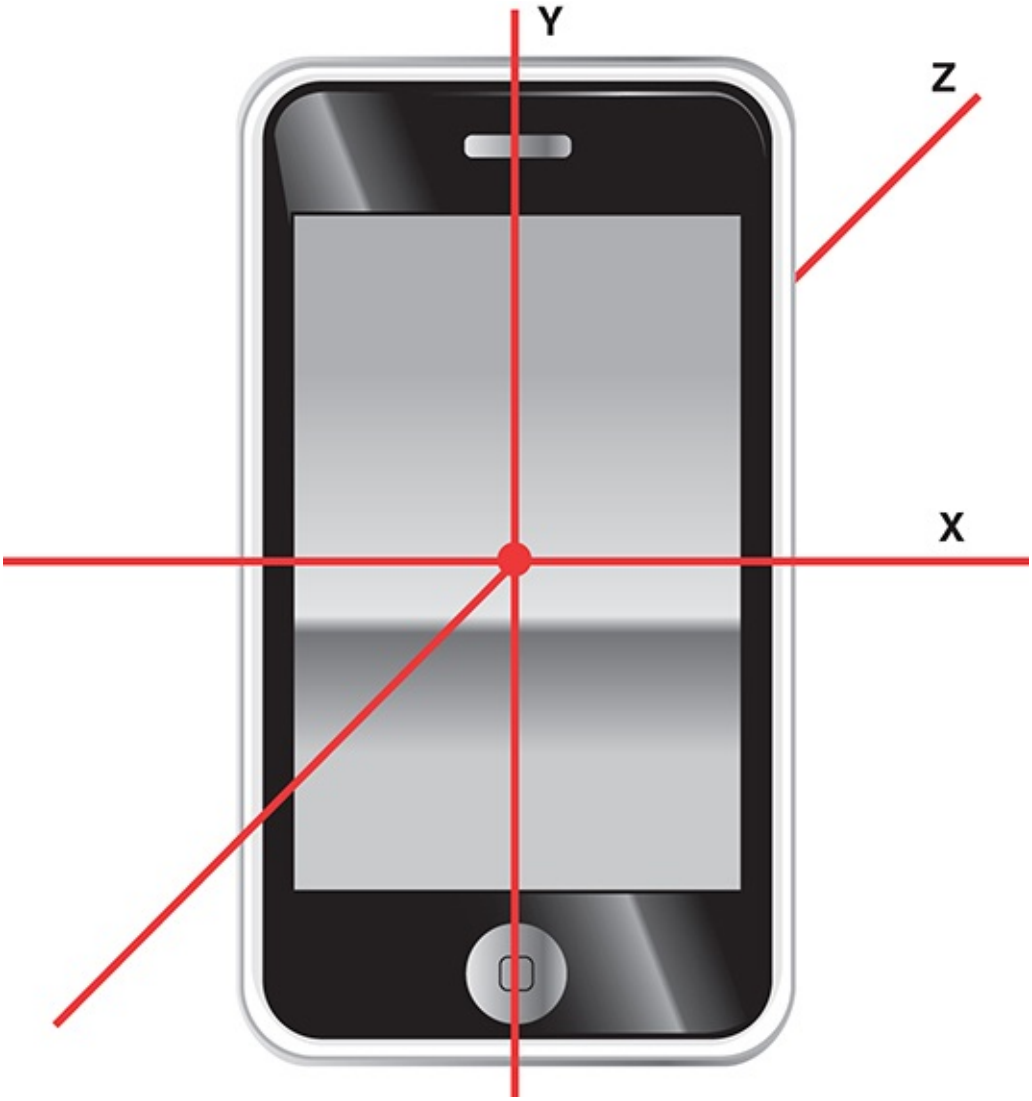

FIGURE 22.3
Accelerometer axes.

As you can see in Figure 22.3, the default axes of a device align with the 3D axes in Unity while the device is being held upright in portrait orientation directly in front of you. If you turn the device to use it in a different orientation, you need to convert the accelerometer data to the correct axis. For example, if you were using the phone pictured in Figure 22.3 in landscape orientation (on its side), you would use the X axis from the phone's accelerometer to represent the Y axis in Unity.

Designing for the Accelerometer

You need to keep in mind a few things when designing a game to use a mobile

device's accelerometer. The first is that you can only ever reliably use two of the accelerometer's axes at any given time. The reason for this is that no matter the orientation of the device, one axis will always be actively engaged by gravity. Consider the orientation of the device in Figure 22.3. You can see that while the x and z axes can be manipulated by tilting the device, the y axis is currently reading negative values, as gravity is pulling it down. If you were to turn the phone so that it rested flat on a surface, face up, you would only be able to use the x and y axes. In that case, the z axis would be actively engaged.

Another thing to consider when designing for an accelerometer is that the input is not extremely accurate. Mobile devices do not read from their accelerometers at a set interval, and they often have to approximate values. As a result, the inputs read from an accelerometer can be jerky and uneven. A common practice, therefore, is to smoothly move objects with accelerometer inputs or to take an average of the inputs over time. Also, accelerometers give input values from -1 to +1, with full 180-degree rotation of the device. No one plays games while fully tilting a devices, though, so input values are generally less than their keyboard counterparts (for example, -.5 to .5).

Using the Accelerometer

Reading accelerometer input is done just like reading any other form of user input: via scripts. All you need to do is read from the Vector3 variable named `acceleration`, which is a part of the `Input` object. Therefore, you could access the x, y, and z axis data by writing the following:

Click here to view code image

```
Input.acceleration.x;
Input.acceleration.y;
Input.acceleration.z;
```

Using these values, you can manipulate your game objects accordingly.

NOTE

Axis Mismatch

When using accelerometer information in conjunction with Unity Remote, you might notice that the axes aren't lining up as described earlier in this hour. This is because Unity Remote bases the game's orientation on the aspect ratio chosen. This means that Unity Remote automatically displays in landscape orientation (holding your device sideways so that the longer edge

is parallel to the ground) and translates the axes for you. Therefore, when you are using Unity Remote, the x axis runs along the long edge of your device, and the y axis runs along the short edge. It might seem strange, but chances are you were going to use your device like that anyway, so this saves you a step.

▼ TRY IT YOURSELF

Moving a Cube with the Power of Your Mind…or Your Phone

In this exercise, you'll use a mobile device's accelerometer to move a cube around a scene. Obviously, to complete this exercise, you need a configured and attached mobile device with an accelerometer. Follow these steps:

1. Create a new project or scene. (If you create a new project, remember to modify the editor settings as described in the previous section.) Add a cube to the scene and position it at (0, 0, 0).

2. Create a new script called AccelerometerScript and attach it to the cube. Put the following code in the `Update()` method of the script:

Click here to view code image

```
float x = Input.acceleration.x  Time.deltaTime;
float z = -Input.acceleration.z  Time.deltaTime;
transform.Translate(x, 0f, z);
```

3. Ensure that your mobile device is plugged in to your computer. Hold the device in landscape orientation and run Unity Remote. Run the scene. Notice that you can move the cube by tilting your phone. Notice which axes of the phone move the cube along the x and z axes.

Multi-Touch Input

Mobile devices tend to be controlled largely through the use of their touch-capacitive screens. These screens can detect when and where a user touches them. They usually can track multiple touches at a time. The exact number of touches varies based on the device.

Touching the screen doesn't just give the device a simple touch location. In fact, there is quite a bit of information stored about each individual touch. In Unity,

each screen touch is stored in a `Touch` variable. This means that every time you touch a screen, a `Touch` variable is generated. That `Touch` variable will exist as long as your finger remains on the screen. If you drag your finger along the screen, the `Touch` variable tracks that. The `Touch` variables are stored together in a collection called `touches`, which is part of the `Input` object. If there is currently nothing touching the screen, then this collection of touches is empty. To access this collection, you could enter the following:

```
Input.touches;
```

By using the `touches` collection, you could iterate through each `Touch` variable to process its data. Doing so would look something like this:

Click here to view code image

```
foreach(Touch touch in Input.touches)
{
   // Do something
}
```

As mentioned earlier in this hour, each touch contains more information than the simple screen data where the touch occurred. Table 22.1 lists all the properties of the `Touch` variable type.

TABLE 22.1 Touch Variable Properties

Property	Description
`deltaPosition`	The change in touch position since the last update. This is useful for detecting finger drags.
`deltaTime`	The amount of time that has passed since the last change to the touch.
`fingerId`	The unique index for the touch. For example, this would range from `0` to `4` on devices that allow five touches at a time.
`phase`	The current phase of the touch: `Began`, `Moved`, `Stationary`, `Ended`, or `Canceled`.
`position`	The 2D position of the touch on the screen.
`tapCount`	The number of taps the touch has performed on the screen.

These properties are useful for managing complex interactions between the user and game objects.

TRY IT YOURSELF

Tracking Touches

In this exercise, you'll track finger touches and output their data to the screen. Obviously, to complete this exercise, you need a configured and attached mobile device with multi-touch support. Follow these steps:

1. Create a new project or scene.
2. Create a new script called TouchScript and attach it to the Main Camera. Put the following code in the script:

Click here to view code image

```
void OnGUI()
{
    foreach (Touch touch in Input.touches)
    {
        string message = "";
        message += "ID: " +touch.fingerId + "\n";
        message += "Phase: " +touch.phase.ToString() + "\n";
        message += "TapCount: " +touch.tapCount + "\n";
        message += "Pos X: " +touch.position.x + "\n";
        message += "Pos Y: " +touch.position.y + "\n";

        int num = touch.fingerId;
        GUI.Label(new Rect(0 + 130 * num, 0, 120, 100), message);
    }
}
```

3. Ensure that your mobile device is plugged in to your computer. Run the scene. Touch the screen with your finger and notice the information that appears (see Figure 22.4). Move your finger and see how the data changes. Now touch with more fingers simultaneously. Move them about and take them off the screen randomly. See how it tracks each touch independently? How many touches can you get on your screen at a time?

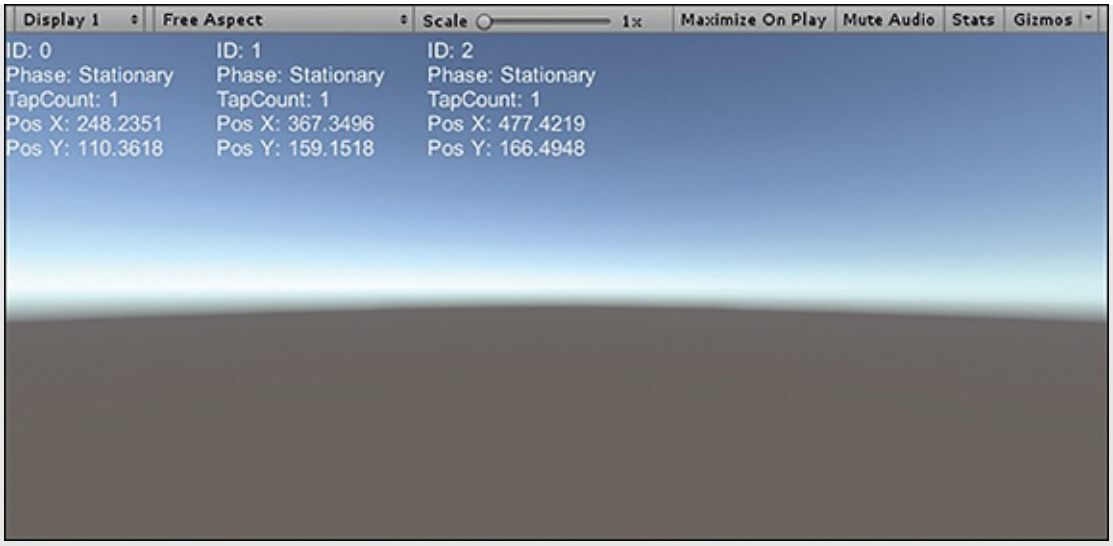

FIGURE 22.4
Touch output on the screen.

CAUTION

Do Because I Say, Not Because I Do!

In the Try It Yourself "Tracking Touches," you created an `OnGUI()` method that collects information about the various touches on the screen. The part of the code where the string `message` is being built with the touch data is a *big* no-no. Performing too much processing in an `OnGUI()` method can greatly reduce the efficiency of a project, and you should try to avoid this. I used this method because it was an easy way to build the example without unneeded complexity and for demonstration purposes only. Always keep update code where it belongs: in `Update()`. In addition, you should really use the new UI because it is much faster.

Summary

In this hour, you've learned about using Unity to develop games for mobile devices. You started by learning how to configure your development environment to work with Android and iOS. From there, you learned how to work with a device's accelerometer. You finished up the hour by experimenting with Unity's touch-tracking system.

Q&A

Q&A

Q. Can I really build a game once and deploy it to all major platforms, mobile included?

A. Absolutely! The only thing to consider is that mobile devices generally don't have as much processing power as desktops. Therefore, mobile device users might experience some performance issues if your game has a lot of heavy processing or effects. You will need to ensure that your game is running efficiently if you plan to also deploy it on mobile platforms.

Q. What are the differences between iOS and Android devices?

A. From a Unity point of view, there isn't much difference between these two operating systems. They are both treated as mobile devices. Be aware, though, that there are some device differences (such as processing power, battery life, and phone OS) that can affect your games.

Workshop

Take some time to work through the questions here to ensure that you have a firm grasp of the material.

Quiz

1. What tool allows you to send live device input data to Unity while it is running a scene?
2. How many axes on the accelerometer can you realistically use at a time?
3. How many touches can a device have at once?

Answers

1. The Unity Remote app
2. Two axes. The third is always engaged by gravity, depending on how you are holding the device.
3. It depends entirely on the device. The last time I tested an iOS device, I was able to track 21 touches. That's enough for all your fingers and toes, plus 1 from a friend!

Exercise

In this exercise, you'll move objects about a scene based on touch input from a mobile device. Obviously, to complete this exercise, you need a configured and attached mobile device with multi-touch support. If you do not have that, you can still read along to get the basic idea.

1. Create a new project or scene. Select **Edit > Project Settings > Editor** and set the Device property to recognize your Unity Remote app.
2. Add three cubes to the scene and name them **Cube1**, **Cube2**, and **Cube3**. Position them at (-3, 1, -5), (0, 1, -5), and (3, 1, -5), respectively.
3. Create a new folder named Scripts. Create a new script called InputScript in the Scripts folder and attach it to the three cubes.
4. Add the following code to the `Update()` method of the script:

Click here to view code image

```
foreach (Touch touch in Input.touches)
{
    float xMove = touch.deltaPosition.x * 0.05f;
    float yMove = touch.deltaPosition.y * 0.05f;

    if (touch.fingerId == 0 && gameObject.name == "Cube1")
        transform.Translate(xMove, yMove, 0F);

    if (touch.fingerId == 1 && gameObject.name == "Cube2")
        transform.Translate(xMove, yMove, 0F);

    if (touch.fingerId == 2 && gameObject.name == "Cube3")
        transform.Translate(xMove, yMove, 0F);
}
```

5. Run the scene and touch the screen with up to three fingers. Notice that you can move the three cubes independently. Also notice that lifting one finger does not cause the other fingers to lose their cubes or their place.