HOUR 12
2D Game Tools

What You'll Learn in This Hour:

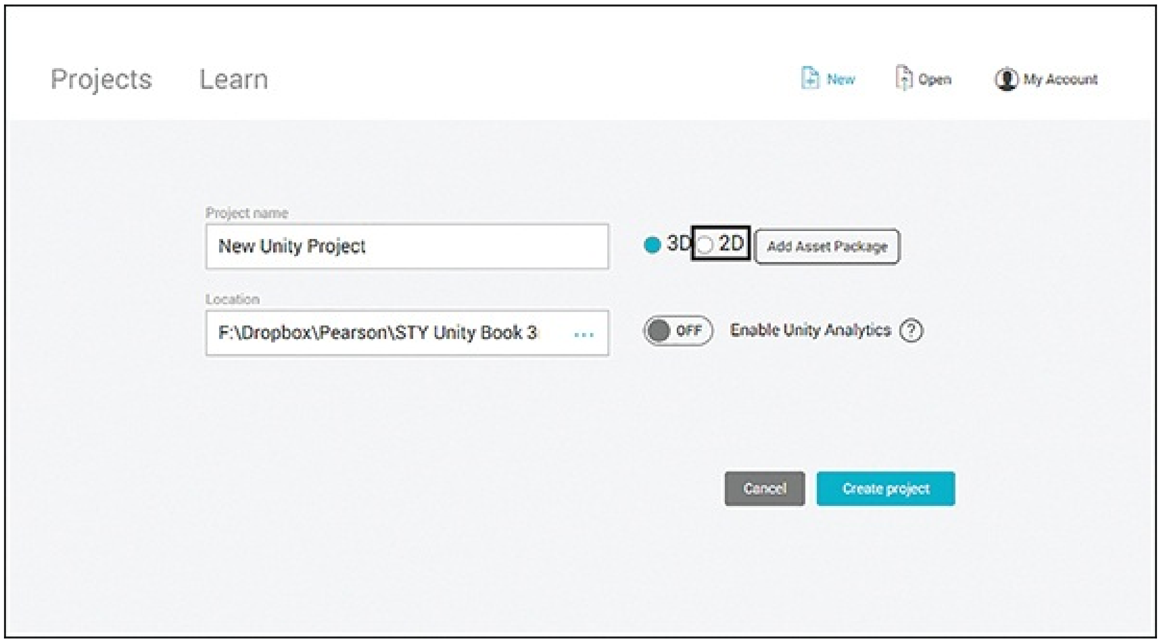
- How orthographic cameras work
- How to position sprites in 3D space
- How to move and collide sprites

Unity is a powerhouse at creating both 2D and 3D games. A pure 2D game is one in which all the assets are simple flat images called *sprites*. In a 3D game, the assets are 3D models with 2D textures applied to them. Typically, in a 2D game, the player can move in only two dimensions (for example, left, right, up, down). This hour is devoted to helping you understand the basics of creating 2D games in Unity.

The Basics of 2D Games

The principles of design for a 2D game are the same as for a 3D game. You still need to consider the concepts, rules, and requirements of the game. There are pros and cons to making a game 2D. On one hand, a 2D game can be simple and much cheaper to produce. On the other hand, the limitations of 2D can make some types of games impossible. 2D games are built from images called sprites. These are a bit like cardboard cutouts in a kids' stage show. You can move them around, place them in front of each other, and cause them to interact to create a rich environment.

When you create a new Unity project, you have the choice of 2D or 3D (see Figure 12.1). Selecting 2D defaults the Scene view to 2D mode and makes the camera orthographic (as discussed later this hour). Another thing you will notice about a 2D project is that there's no directional light or skybox in the scene. In fact, 2D games are generally unlit because sprites are drawn by a simple type of renderer called a "sprite renderer." Unlike a texture, lighting does not generally affect the way a sprite is drawn.



Project name: New Unity Project

Location: F:\Dropbox\Pearson\STY Unity Book 3

● 3D ○ 2D

OFF Enable Unity Analytics

FIGURE 12.1
Setting a project to 2D.

TIP

2D Game Challenges

2D games bring unique design challenges. Examples include the following:

- The lack of depth makes immersion harder to achieve.
- A lot of game types don't translate well to 2D.
- 2D games are generally unlit, so the sprites have to be carefully drawn and arranged.

The 2D Scene View

A project with 2D defaults will already be in the 2D Scene view. To change between 2D and 3D view, click the 2D button at the top of the Scene view (see Figure 12.2). The 3D gizmo disappears when you enter 2D mode.

You can move around in 2D mode by right-clicking (or middle-clicking) and dragging in the scene. You can use the mouse wheel to zoom, or you can use the scroll gesture on a trackpad. You can use the background grid to orient yourself. You no longer have the scene gizmo, but you don't need it. Your scene orientation in 2D mode doesn't change.

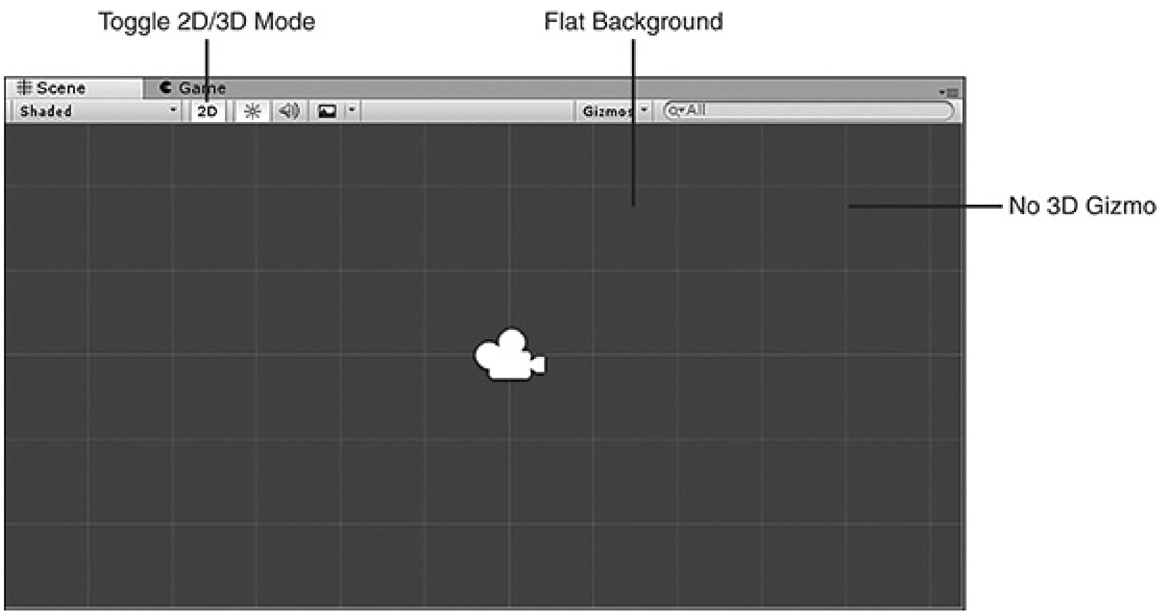


FIGURE 12.2
The 2D Scene view.

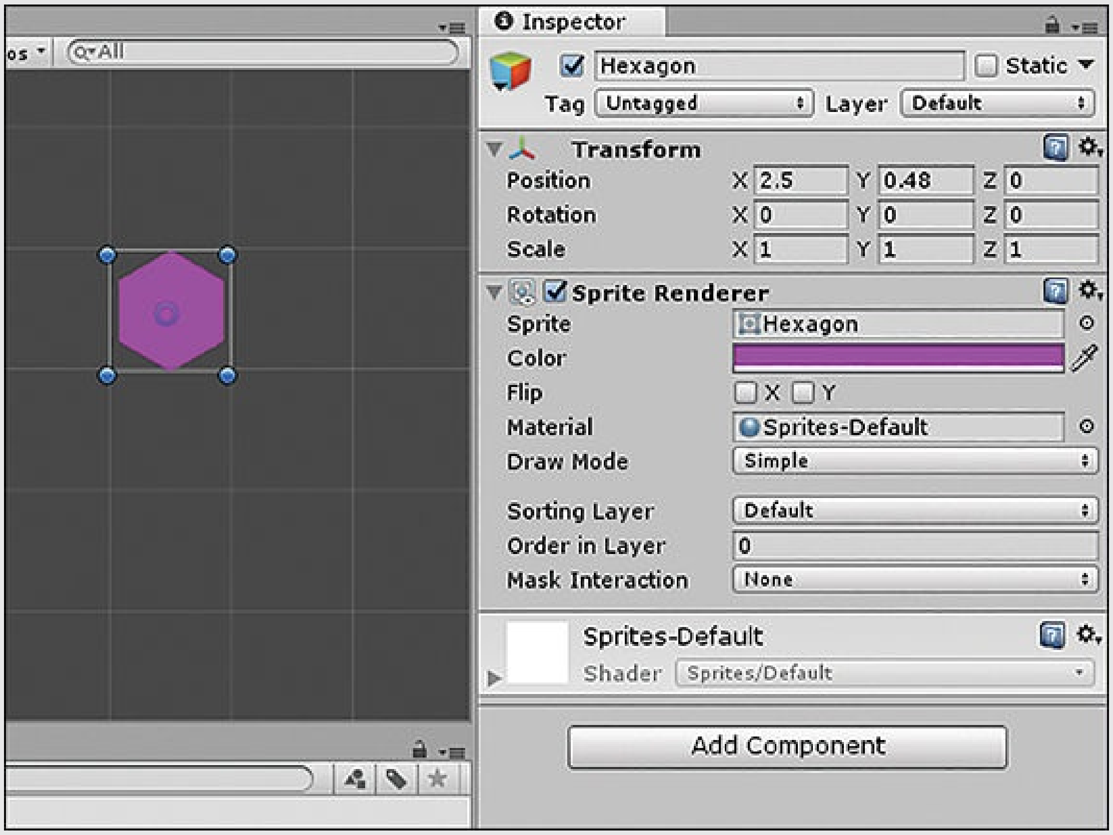
▼ TRY IT YOURSELF

Creating and Placing a Sprite

You'll look at importing and using sprites later. For now, you will just create a simple sprite and add it to the scene:

1. Create a new project, this time selecting 2D (refer to Figure 12.1). The project defaults are now set up for a 2D game. (Note that there is no directional light in a 2D game.)
2. Add a new folder to the project and name it **Sprites**. Right-click in the Project view and select **Create > Sprites > Hexagon**.
3. Drag the newly created Hexagon sprite into the Scene view. Notice the Sprite Renderer component in the Inspector.
4. While the sprite asset is white, you can change its color by using the Sprite Renderer component, so choose a different color for the sprite

(see Figure 12.3).


Inspector

Hexagon — Static
Tag: Untagged — Layer: Default

Transform
Position X 2.5 Y 0.48 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Sprite Renderer
Sprite: Hexagon
Color
Flip: X Y
Material: Sprites-Default
Draw Mode: Simple
Sorting Layer: Default
Order in Layer: 0
Mask Interaction: None

Sprites-Default
Shader: Sprites/Default

FIGURE 12.3
The Hexagon sprite.

TIP

The Rect Tool

As explained in Hour 2, “Game Objects,” the Rect transform tool is ideal for manipulating rect(angular) sprites. You can access this tool at the top left of the Unity editor or by pressing the **T** key. Try playing with it while you have a sprite onscreen.

Orthographic Cameras

Because you set up your project as a 2D project, the camera defaults to the orthographic type. This means the camera doesn’t show perspective distortion; rather, all objects appear the same size and shape, regardless of distance. This type of camera is suitable for 2D games, where you control the apparent depth of type of camera is suitable for 2D games, where you control the apparent depth of sprites by their size and sorting layer.

A *sorting layer* allows you to determine which sprites and groups of sprites draw in front of each other. Imagine that you are setting up a stage. You would want the background props behind the foreground props. If you sort the sprites from front to back properly and choose appropriate sizes, you can give the impression of depth.

You can see the camera type for yourself by clicking **Main Camera** and noticing that Projection is set to **Orthographic** in the Inspector (see Figure 12.4). (The rest of the camera settings are covered in Hour 5, “Lights and Cameras.”)

TIP

The Size of an Orthographic Camera

Often you want to change the relative sizes of sprites in your scene all at once, without resizing them. If the camera doesn’t take depth into account, you may be wondering how you do this, as you can’t just move them closer to the camera.

In this case, you need to use the Size property in the Inspector (see Figure 12.4), which appears only for orthographic projection cameras. This is the number of world units from the center to the top of the camera’s view. That might seem odd, but it has to do with measurements such as aspect ratio. In essence, though, bigger numbers “zoom out,” while smaller numbers “zoom in.”

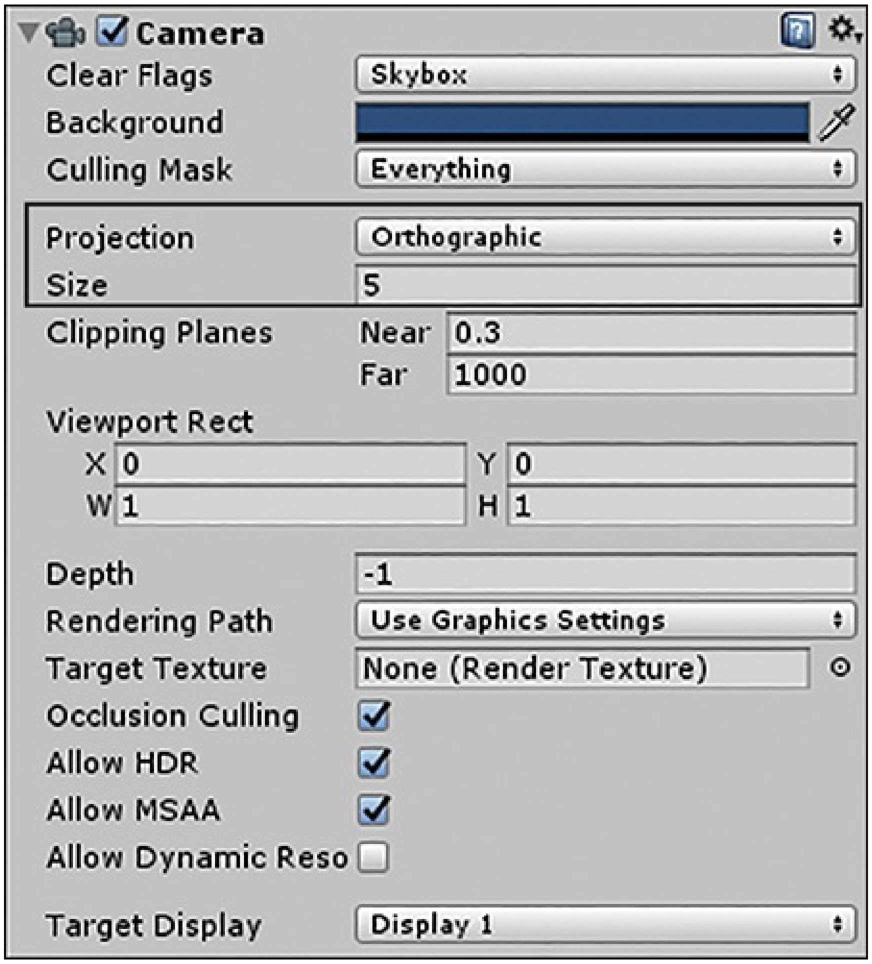

Camera
Clear Flags: Skybox
Background
Culling Mask: Everything
Projection: Orthographic
Size: 5
Clipping Planes: Near 0.3, Far 1000
Viewport Rect: X 0, Y 0, W 1, H 1
Depth: -1
Rendering Path: Use Graphics Settings
Target Texture: None (Render Texture)
Occlusion Culling: ✓
Allow HDR: ✓
Allow MSAA: ✓
Allow Dynamic Reso: ☐
Target Display: Display 1

FIGURE 12.4
Setting the size of an orthographic camera.

Adding Sprites

Adding a sprite to a scene is a very easy process. Once a sprite image is imported into a project (again, super simple), you simply need to drag it into your scene.

Importing Sprites

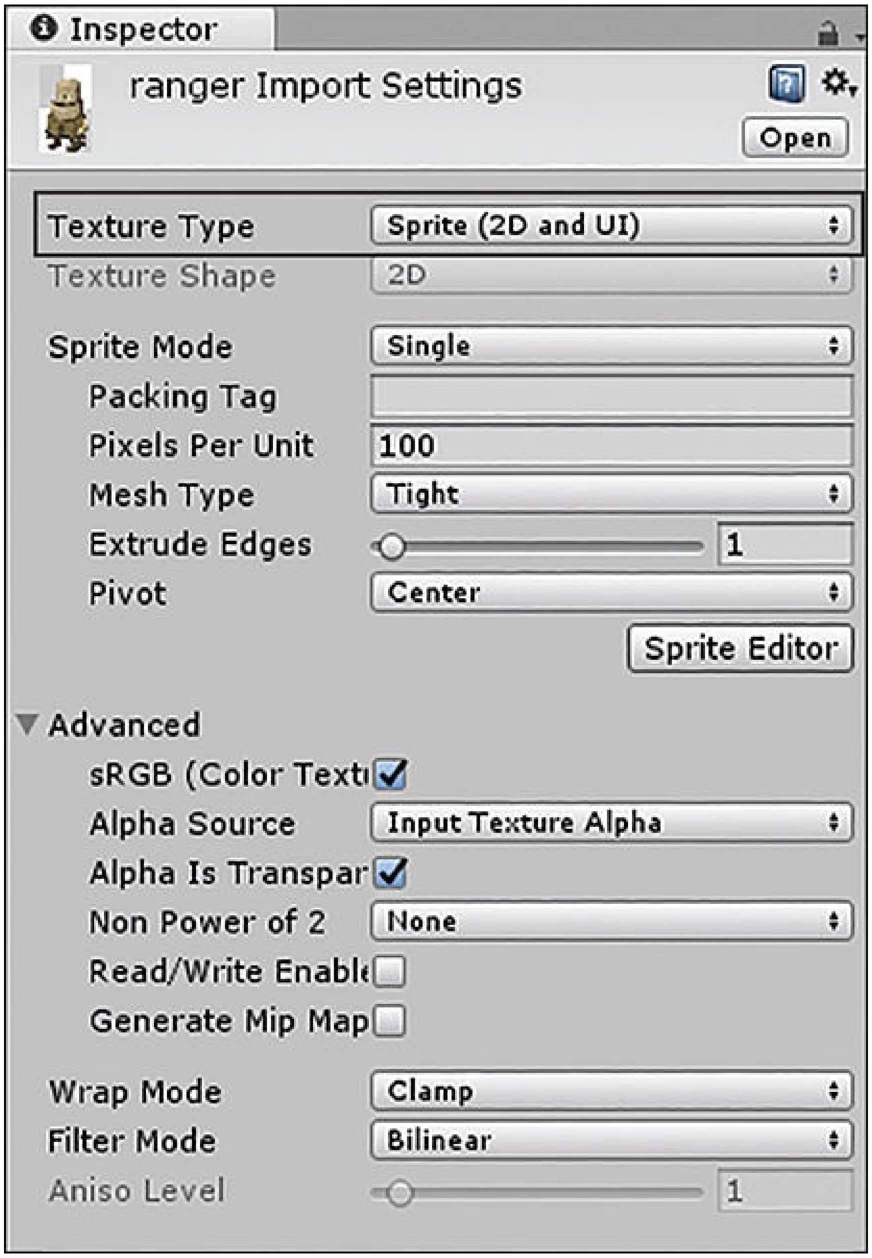
If you wish to use your own image as a sprite, you need to make sure Unity knows that it is a sprite. Here's how you import a sprite:

1. Locate the ranger.png file in the book assets (or use your own).

2. Drag the image into the Project view in Unity. If you created a 2D project, it imports as a sprite. If you made a 3D project, you need to tell

Unity to use it as a sprite by setting Texture Type to **Sprite** (see Figure 12.5).

3. Simply drag the sprite into the Scene view. It’s that simple! Note that if your Scene view is not in 2D mode, the sprite that is created won’t necessarily be at a z axis position of 0 (like the rest of the sprites).



Inspector

ranger Import Settings

Open

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Single
Packing Tag:
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Pivot: Center

Sprite Editor

Advanced
sRGB (Color Texti: ✓
Alpha Source: Input Texture Alpha
Alpha Is Transpar: ✓
Non Power of 2: None
Read/Write Enabl:
Generate Mip Map:
Wrap Mode: Clamp
Filter Mode: Bilinear
Aniso Level: 1

FIGURE 12.5
Making an image texture into a sprite.

Sprite Mode

Single sprites are fine, but the fun really starts when you start animating. Unity provides some powerful tools for using sprite sheets. A *sprite sheet* is an image with multiple frames of an animation laid out in a grid.

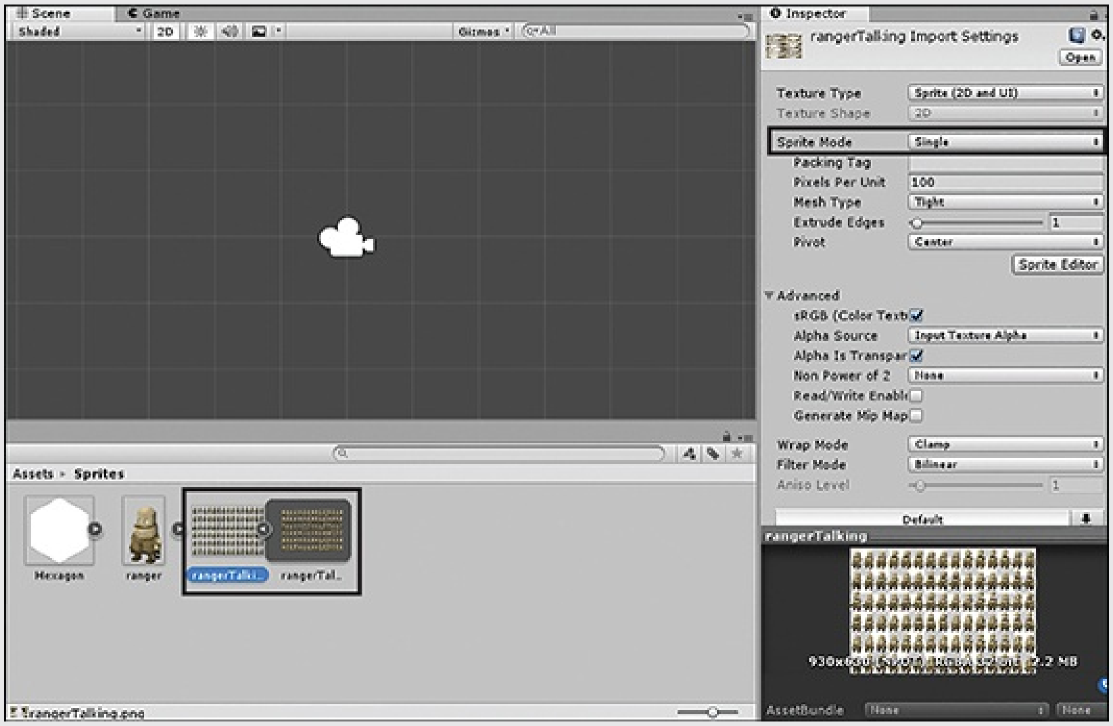
The Sprite Editor (which you will explore in a moment) helps you automatically extract tens or hundreds of different animation frames from a single image.

▼ TRY IT YOURSELF

Exploring Sprite Modes

Follow these steps to explore the difference between the Single and Multiple sprite modes:

1. Create a new 2D project or a new scene in your existing 2D project.
2. Import rangerTalking.png from the book asset files or source your own sprite sheet.
3. Ensure that Sprite Mode is set to **Single** and expand the sprite tray in the Project view to make sure Unity is treating this image as a single sprite (see Figure 12.6).



rangerTalking Import Settings

Texture Type: Sprite (2D and UI)
Texture Shape: 2D
Sprite Mode: Single
Packing Tag:
Pixels Per Unit: 100
Mesh Type: Tight
Extrude Edges: 1
Pivot: Center

▼ Advanced
sRGB (Color Textu ✓
Alpha Source: Input Texture Alpha
Alpha Is Transpar ✓
Non Power of 2: None
Read/Write Enabl ☐
Generate Mip Map ☐

Wrap Mode: Clamp
Filter Mode: Bilinear
Aniso Level: 1

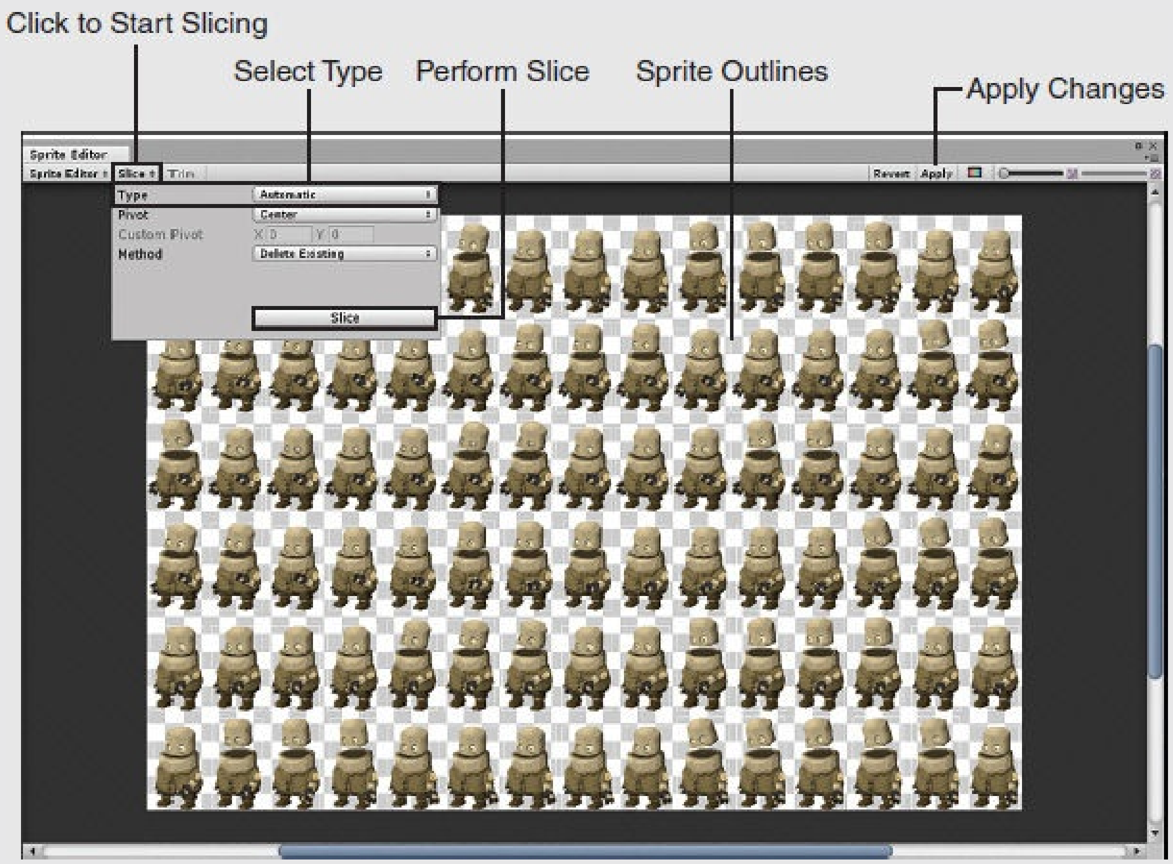
930x630 (NPOT) RGBA 32 bit 2.2 MB
AssetBundle: None None

FIGURE 12.6

Importing a sprite in Single mode

importing a sprite in Single mode.

4. **4.** Now change the Sprite Mode to **Multiple** and click **Apply** at the bottom of the Inspector. Note that there is temporarily no tray to expand.
5. **5.** Click the **Sprite Editor** button in the Inspector; a window pops up. Click the **Slice** drop-down, set Type to **Automatic**, and click **Slice** (see Figure 12.7). Notice how the outlines are automatically detected but are different for each frame.
6. **6.** Set Type to **Grid by Cell Size** and adjust the grid to fit your sprite. For the supplied image, the grid values are x = 62 and y = 105. Leave the other settings as they are and click **Slice**. Notice that the borders are now more even.
7. **7.** Click **Apply** to save the changes and close the Sprite Editor.
8. **8.** Look at your sprite in the Project view and note that the tray now contains all the individual frames, ready for animation.



Type Automatic
Pivot Center
Custom Pivot X 0 Y 0
Method Delete Existing

FIGURE 12.7
The Sprite Editor window.

The Sprite Editor window.

NOTE

Sprite Sheet Animation

The Try It Yourself “Exploring Sprite Modes” shows how to import and configure a sprite sheet. As mentioned earlier, sprite sheets are often used for 2D animations. You will learn about sprite-based animations in Hour 17, “Animations.”

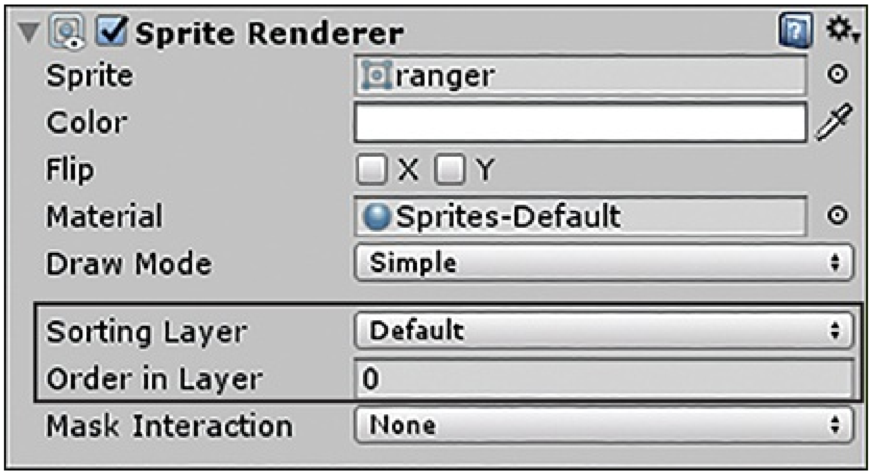
Imported Sprite Sizes

If you need to scale sprite images so that their sizes match, you have a couple options. The best way to fix sprite sizing issues is to open the sprites in image editing software such as Photoshop and make the corrections. You may not always have that capability (or time), though. Another option is to use the scale tool in the scene. Doing that, however, adds inefficiencies and potentially “odd” scaling behavior in the future.

If you need to consistently scale a sprite—for example, SpriteA always needs to be half as big, consider instead using the Pixels per Unit import setting. This setting determines how many world units a sprite of a given resolution occupies. For example, a 640 × 480 image imported with a Pixels per Unit setting of **100** would occupy 6.4 × 4.8 world units.

Draw Order

To determine what sprites draw in front of each other, Unity has a sorting layer system. It is controlled by two settings on the Sprite Renderer component: Sorting Layer and Order in Layer (see Figure 12.8).



Sprite Renderer	
Sprite	ranger
Color	
Flip	☐ X ☐ Y
Material	Sprites-Default
Draw Mode	Simple
Sorting Layer	Default
Order in Layer	0
Mask Interaction	None

FIGURE 12.8
The default sprite layer properties.

Sorting Layers

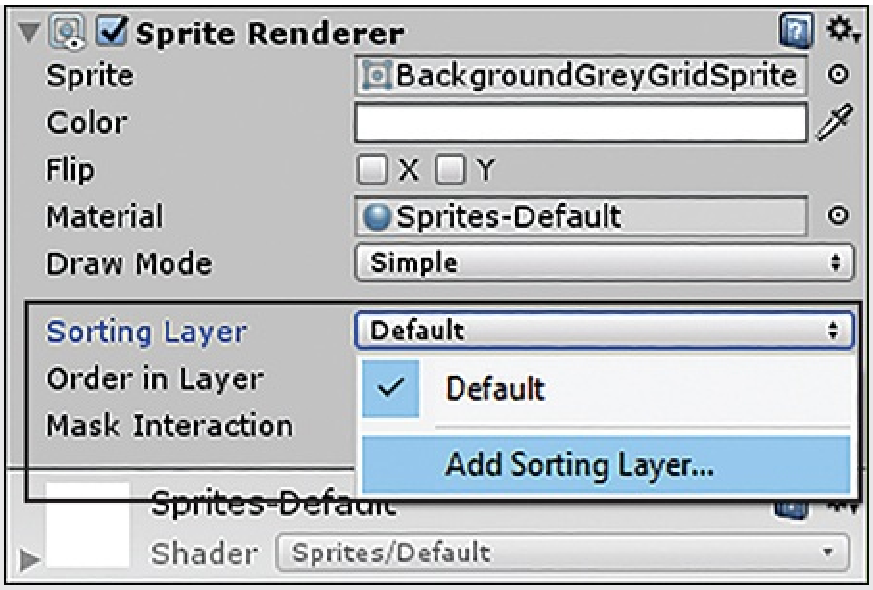
You can use sorting layers to group major categories of sprites by depth. When you first create a new project, there is only one sorting layer, called Default (refer to Figure 12.8). Imagine that you have a simple 2D platform game with a background consisting of hills, trees, and clouds. In this case, you would want to create a sorting layer called Background.

▼ TRY IT YOURSELF

Creating Sorting Layers

Follow these steps to create and assign a new sorting layer to a sprite:

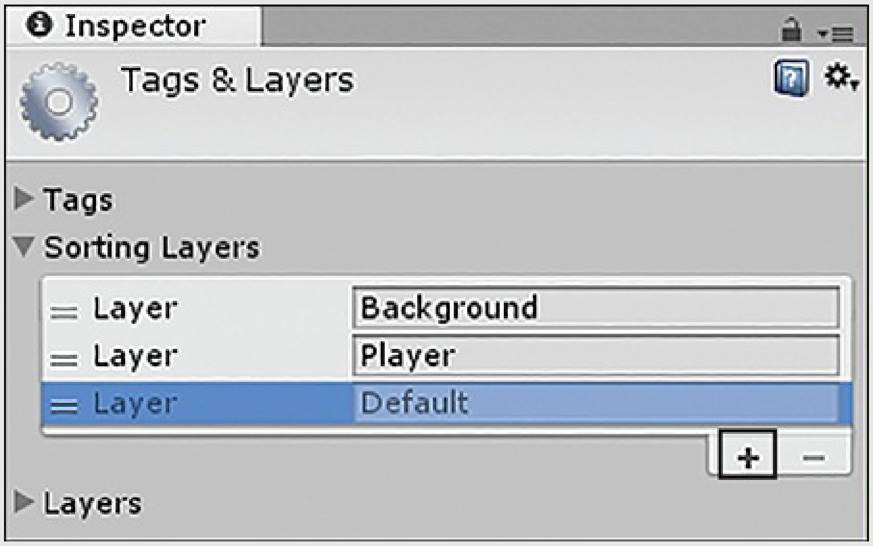
1. Create a new project and import the entire 2D asset pack (by selecting **Assets > Import Package > 2D**).
2. Locate the sprite BackgroundGreyGridSprite in the folder Standard Assets\2D\Sprites and drag it into the scene. Set its position to (–2.5, –2.5, 0) and scale it to (2, 2, 1).
3. Add a new sorting layer by clicking the **Sorting Layer** drop-down of the Sprite Renderer component and then clicking **Add Sorting Layer** (see Figure 12.9).



Sprite Renderer
Sprite: BackgroundGreyGridSprite
Color:
Flip: X Y
Material: Sprites-Default
Draw Mode: Simple
Sorting Layer: Default
Order in Layer
Mask Interaction

FIGURE 12.9
Adding new sorting layers.

4. Add two new layers called Background and Player. Do this by clicking the + button under Sorting Layers (see Figure 12.10). Note that the lowest layer in the list, currently Player, will be drawn last and will thus appear in front of all others in the game. It is a good idea to move the default layer to the bottom so that new items are drawn on top of other layers.



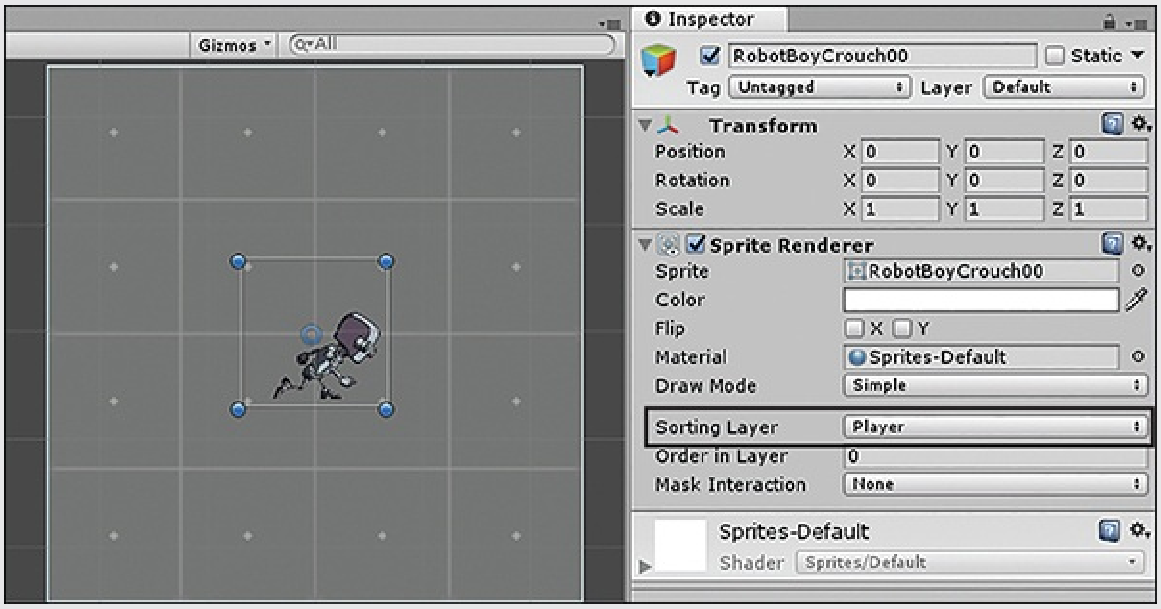
Inspector
Tags & Layers
Tags
Sorting Layers
Layer: Background
Layer: Player
Layer: Default

FIGURE 12.10
Managing sorting layers.

5. Re-select the BackgroundGreyGridSprite game object and set its sorting layer to the newly created **Background**.

6. In the Project view, search for the sprite RobotBoyCrouch00 and drag it into the scene. Position it at (0, 0, 0).

7. Set Sorting Layer for this sprite to **Player** (see Figure 12.11). Note that the player is drawn on top of the background. Now try going back into the Tags & Layers Manager (by selecting **Edit > Project Settings > Tags and Layers**) and rearranging the layers so that the background is drawn on top of the player.



RobotBoyCrouch00

Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Sprite RobotBoyCrouch00
Color
Flip X Y
Material Sprites-Default
Draw Mode Simple
Sorting Layer Player
Order in Layer 0
Mask Interaction None

FIGURE 12.11
Setting a sprite's sorting layer.

You would likely want another layer for power-ups, bullets that may spawn, and so on. The ground or platform may be on another layer. Finally, if you want to add more sense of depth with subtle foreground sprites, they could go in a Foreground layer.

Order in Layer

Once you have your major layers defined and assigned to your sprites, you can fine-tune the draw order with the Order in Layer setting. This is a simple priority system, with higher numbers drawing on top of lower numbers.

WARNING

Sprites Going Missing

When you are starting out, it is quite common for sprites to go missing altogether. This is usually either because a sprite is behind a larger element or behind the camera.

You may also find that if you forget to set the Sorting Layer and Order in Layer properties, the z depth can affect the draw order. This is why it is important to always set the layer for every sprite.

2D Physics

Now that you understand how static sprites work in your game, it’s time to add some spice by making them move. Unity has some fantastic tools to help with this. There are two main ways of animating sprites in Unity, as detailed in the following sections. Let’s dive in and explore them. Unity has a powerful 2D physics system that integrates a system called Box2D. Just as in 3D games, you can use rigidbodies with gravity, various colliders, and physics specific to 2D platform and driving games.

Rigidbody 2D

Unity has a different type of rigidbody for 2D purposes. This physics component shares many of the same properties with its 3D counterpart, which you have already met. It is a common mistake to choose the wrong type of rigidbody or collider for a game, so be careful to ensure that you are using items from **Add Component > Physics 2D**.

TIP

Mixing Physics Types

2D and 3D physics can exist in the same scene together, but they will not interact with each other. Also, you can put 2D physics on 3D objects and vice versa!

2D Colliders

Just like 3D colliders, 2D colliders allow game objects to interact with one another when they touch. Unity comes with a range of 2D colliders (see Table 12.1).

TABLE 12.1 2D Colliders

Collider	Description
Circle Collider 2D	A circle with a fixed radius and an offset.
Box Collider 2D	A rectangle with an adjustable width, height, and offset.
Edge Collider 2D	A segmented line that does not need to be closed.
Capsule Collider 2D	A capsule shape with offset, size, and direction.
Composite Collider 2D	A special collider that is a composite of several primitive colliders, turning several into one.
Polygon Collider 2D	A closed polygon with three or more sides.

▼ TRY IT YOURSELF

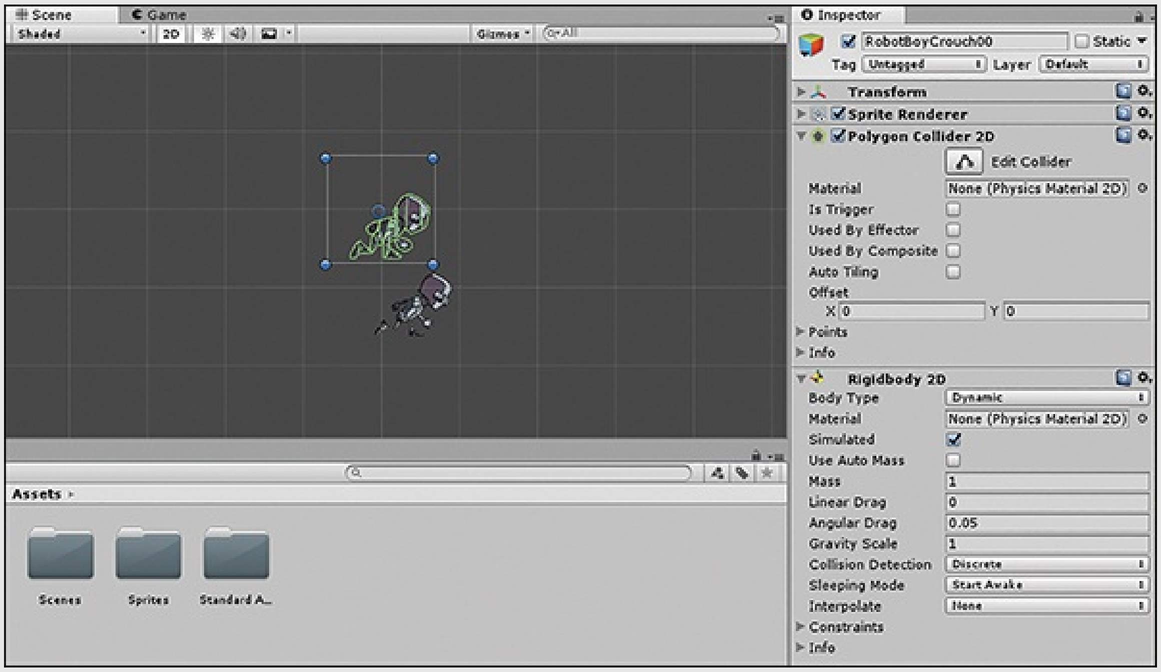
Making Two Sprites Collide

Follow these steps to see the effects of 2D colliders by playing with some 2D squares:

1. Create a new 2D scene and ensure that the 2D package is imported.
2. Find the sprite RobotBoyCrouch00 in the folder Assets\Standard Assets\2D\Sprites and drag it into your Hierarchy view. Ensure that the sprite is at (0, 0, 0).
3. Add a polygon collider (by selecting **Add Component** > **Physics 2D** > **Polygon Collider 2D**). Note that this collider roughly fits the outline of the sprite.
4. Duplicate this sprite and move the duplicate down and to the right to (.3, –1, 0). This gives the top sprite something to fall onto.

5. So that the top sprite responds to gravity, select it and add a rigidbody 2D (by selecting **Add Component > Physics 2D > Rigidbody 2D**). See Figure 12.12 for the final setup.

6. Play the scene and notice the behavior. See? Just like 3D physics.



RobotBoyCrouch00 ☐ Static
Tag Untagged Layer Default
Transform
Sprite Renderer
Polygon Collider 2D
Edit Collider
Material None (Physics Material 2D)
Is Trigger ☐
Used By Effector ☐
Used By Composite ☐
Auto Tiling ☐
Offset X 0 Y 0
Points
Info
Rigidbody 2D
Body Type Dynamic
Material None (Physics Material 2D)
Simulated ✓
Use Auto Mass ☐
Mass 1
Linear Drag 0
Angular Drag 0.05
Gravity Scale 1
Collision Detection Discrete
Sleeping Mode Start Awake
Interpolate None
Constraints
Info

FIGURE 12.12
The finished setup with the top robot selected.

TIP

2D Colliders and Depth

One thing you may find a little odd about 2D colliders, if you look at them in a 3D Scene view, is that they don't need to be at the same z depth in order to collide. They work only on the x and y positions of the collider.

Summary

In this hour, you have learned about the basics of 2D games in Unity. You started by increasing your understanding of orthographic cameras and how depth works in 2D. You went on to make a simple 2D object move and collide, which is the basis of many 2D games.

Q&A

Q. Is Unity a good choice for creating 2D games?

A. Yes. Unity has a fantastic set of tools for creating 2D games.

Q. Can Unity deploy 2D games to mobile and other platforms?

A. Absolutely. One of Unity's core strengths is its capability to deploy to many platforms with relative ease. 2D games are no exception to this, and many very successful 2D games have been made with Unity.

Workshop

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

Quiz

1. What camera projection renders all objects without perspective distortion?
2. What does the size setting of an orthographic camera relate to?
3. Do two 2D sprites need to be at the same z depth in order to collide?
4. Will sprites render if they are behind the camera?

Answers

1. Orthographic
2. The size setting specifies half of the vertical height that the camera covers, specified in world units.
3. No. 2D collisions take account of only x and y positions.
4. No. This is a common cause of lost sprites when making 2D games.

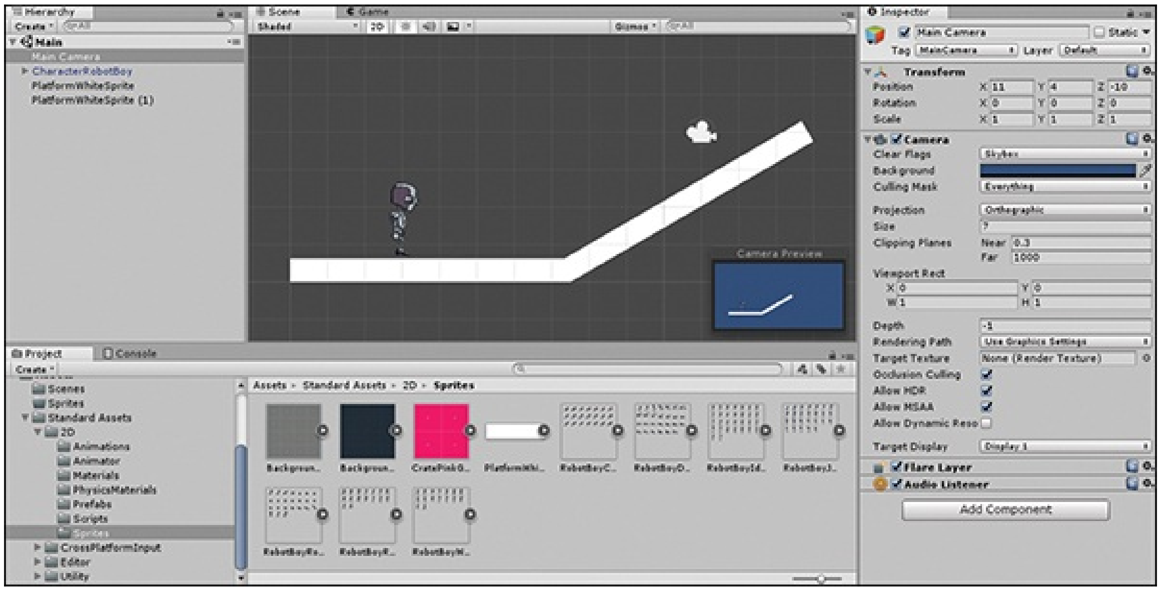
Exercise

In this exercise, you are going to use some Unity standard assets so that you can see what can be achieved when you combine sprites with a little animation, some character control script, and some colliders.

1. Create a new 2D project or a new scene in an existing project.
2. Import the 2D asset pack (by selecting **Assets > Import Package > 2D**).

Leave everything selected.

3. Find CharacterRobotBoy in Assets\Standard Assets\2D\Prefabs and drag it into the Hierarchy view. Set the position to (3, 1.8, 0). Note that this prefab comes with many components in the Inspector.
4. Find PlatformWhiteSprite in Assets\Standard Assets\2D\Sprites and drag it into the Hierarchy view. Set the position to (0, 0, 0) and scale to (3, 1, 1). Add a Box Collider 2D so that your player doesn't fall through the floor!
5. Duplicate this platform game object. Position the duplicate at (7.5, 0, 0), rotate it to (0, 0, 30) to make a ramp, and scale it to (3, 1, 1).
6. Move the Main Camera to (11, 4, –10) and adjust its size to 7. See Figure 12.13 for the final setup.
7. Click **Play**. Use the arrow keys to move and the spacebar to jump.



Main Camera
Static
Tag MainCamera Layer Default

Transform
Position X 11 Y 4 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Camera
Clear Flags Skybox
Background
Culling Mask Everything
Projection Orthographic
Size 7
Clipping Planes Near 0.3 Far 1000
Viewport Rect X 0 Y 0 W 1 H 1
Depth -1
Rendering Path Use Graphics Settings
Target Texture None (Render Texture)
Occlusion Culling ✓
Allow HDR ✓
Allow MSAA ✓
Allow Dynamic Reso ☐
Target Display Display 1

Flare Layer
Audio Listener
Add Component

FIGURE 12.13
The final exercise setup.