HOUR 15
Game 3: *Captain Blaster*

What You'll Learn in This Hour:

- How to design the game *Captain Blaster*
- How to build the *Captain Blaster* world
- How to build the *Captain Blaster* entities
- How to build the *Captain Blaster* controls
- How to further improve *Captain Blaster*

Let's make a game! In this hour, you'll make a 2D scrolling shooter game titled *Captain Blaster*. You'll start by designing the various elements of the game. From there, you'll begin building the scrolling background. Once the idea of motion is established, you'll begin building the various game entities. When the entities are done, you'll construct the controls and gamify the project. You'll finish this hour by analyzing the game and identifying ways to improve it.

TIP

The Completed Project

Be sure to follow along in this hour to build the complete game project. In case you get stuck, you can find a completed copy of the game in the book assets for Hour 15. Take a look at it if you need help or inspiration.

Design

You learned about design elements in Hour 6, "Game 1: *Amazing Racer*." This hour, you’ll dive right into them.

The Concept

As mentioned earlier, *Captain Blaster* is a 2D scrolling shooter-style game. The premise is that the player will be flying around a level, destroying meteors, and trying to stay alive. The neat thing about 2D scrolling games is that the players themselves don’t actually have to move at all. The scrolling background simulates the idea that the player is going forward. This reduces the required player skill and allows you to create more challenges in the form of enemies.

The Rules

The rules of this game state how to play and allude to some of the properties of the objects. The rules for *Captain Blaster* are as follows:

- The player plays until being hit by a meteor. There is no win condition.
- The player can fire bullets to destroy the meteors. The player earns 1 point per meteor destroyed.
- The player can fire two bullets per second.
- The player is bounded by the sides of the screen.
- Meteors come continuously until the player loses.

The Requirements

The requirements for this game are simple:

- A background sprite to be outer space.
- A ship sprite.
- A meteor sprite.
- A game manager. This will be created in Unity.
- Interactive scripts. These will be written in MonoDevelop or Visual Studio as usual.

The World

Because this game takes place in space, the world will be fairly simple to implement. The game will be 2D, and the background will move vertically behind the player to make it seem like the player is moving forward. In actuality, the player will be stationary. Before you get the scrolling in place, though, you need to set up your project. Start with these steps:

1. Create a new 2D project named Captain Blaster.
2. Create a folder named Scenes and save your scene as Main.
3. In the Game view, change the aspect ratio to **5:4** (see Figure 15.1).

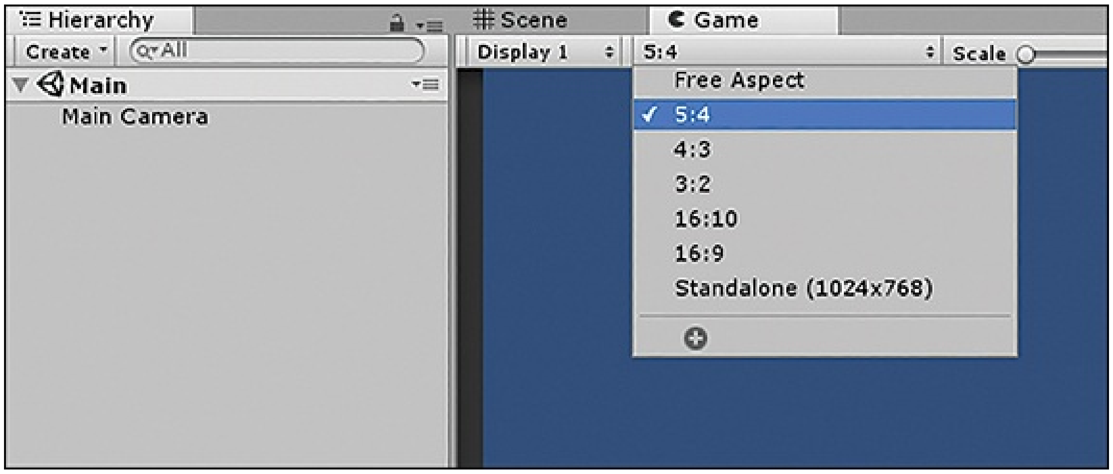
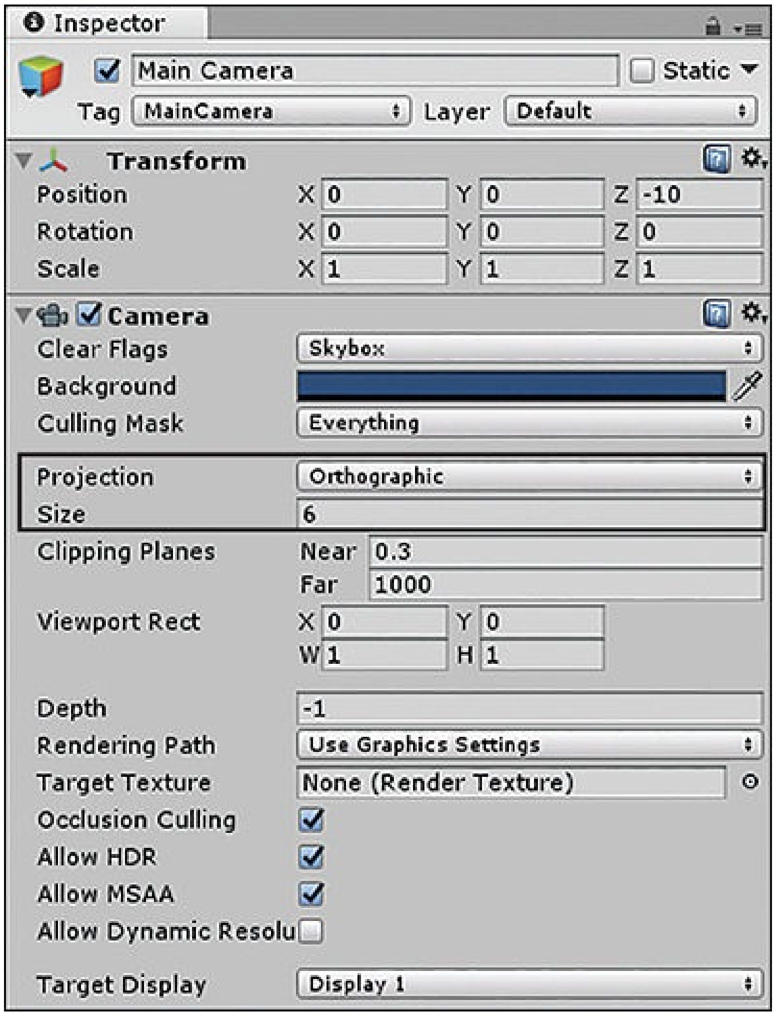


FIGURE 15.1
Setting the game's aspect ratio.

The Camera

Now that the scene is set up properly, it is time to work on the camera. Because this is a 2D project, you have an orthographic camera, which lacks depth perspective and is great for making 2D games. To set up the Main Camera, simply set the Size property to **6**. (See Figure 15.2 for a list of the camera's properties.)


Inspector

Main Camera ☑ ☐ Static

Tag MainCamera Layer Default

Transform

Position	X 0	Y 0	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	6
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 Resolu	☐
Target Display	Display 1

FIGURE 15.2
The Main Camera properties.

The Background

The scrolling background can be a little tricky to set up correctly. Basically, you need to have two background objects moving down the screen. As soon as the bottom object goes offscreen, you place it above the screen. You keep flipping back and forth between them, and the player doesn't know this is what's happening. To create the scrolling background, follow these steps:

1. Create a new folder named Background. Locate the Star_Sky.png image from the book files and import it into Unity by dragging it into the Background folder you just created. Remember that because you made a 2D project, this image automatically imports as a sprite.

2. Select the newly imported sprite in the Project view and change its

Pixels per Unit property in the Inspector view to **50**. Drag the Star_Sky sprite into the scene and ensure that it is positioned at (0, 0, 0).

3. Create a new script in your Background folder named ScrollBackground and drag it onto the background sprite in the scene. Put the following code in the script:

Click here to view code image

```
public float speed = -2f;
public float lowerYValue = -20f;
public float upperYValue = 40;

void Update()
{
    transform.Translate(0f, speed * Time.deltaTime, 0f);
    if (transform.position.y <= lowerYValue)
    {
        transform.Translate(0f, upperYValue, 0f);
    }
}
```

4. Duplicate the background sprite and place it at (0, 20, 0). Run the scene. You should see the background seamlessly stream by.

NOTE

Alternate Organization

Up until now, you have used a fairly straightforward system for organization. Assets went into corresponding folders: sprites in a sprites folder, scripts in a scripts folder, and so on. In this hour, however, you are going to do something new. This time, you are going to group assets based on their "entity": all background files together, ship assets together, and so on. This system works well for finding all related assets quickly. Don't worry if you like the other system, though. You can still search and sort asset names and types by using the search bar and filter properties located at the top of the Project view. As Ben Tristem (Howdy, Ben!) recently told me, "There's more than one way to do things!"

NOTE

Seamless Scrolling

You might notice a small line in the scrolling background you just created.

This is due to the fact that the image used for the background wasn't made specifically to tile together. Generally, it isn't very noticeable, and the actions of the game will more than cover it up. If you want a more seamless background in the future, however, you should use an image made to tile together.

Game Entities

In this game, you need to make three primary entities: the player, the meteor, and the bullet. The interaction between these items is very simple. The player fires bullets. Bullets destroy meteors. Meteors destroy the player. Because the player can technically fire a large number of bullets, and because a large number of meteors can spawn, you need a way to clean them up. Therefore, you also need to make triggers that destroy bullets and meteors that enter them.

The Player

Your player will be a spaceship. The sprites for both the spaceship and the meteors can be found in the book assets for Hour 15. (Thanks to Krasi Wasilev, at http://freegameassets.blogspot.com.) To create the player, follow these steps:

1. Create a new folder in your Project view called Spaceship and import spaceship.png from the book files into this folder. Note that the spaceship sprite is currently facing downward. This is okay.

2. Select the spaceship sprite and in the Inspector set Sprite Mode to **Multiple** and click **Apply**. Then click **Sprite Editor** to start slicing your sprite sheet. (If you've forgotten about slicing, look back at Hour 12, "2D Game Tools.")

3. Click **Slice** in the upper left of the Sprite Editor window and set Type to **Grid By Cell Size**. Set x to **116** and y to **140** (see Figure 15.3). Click **Slice** and notice the outlines around the ships. Click **Apply** and close the Sprite Editor window.

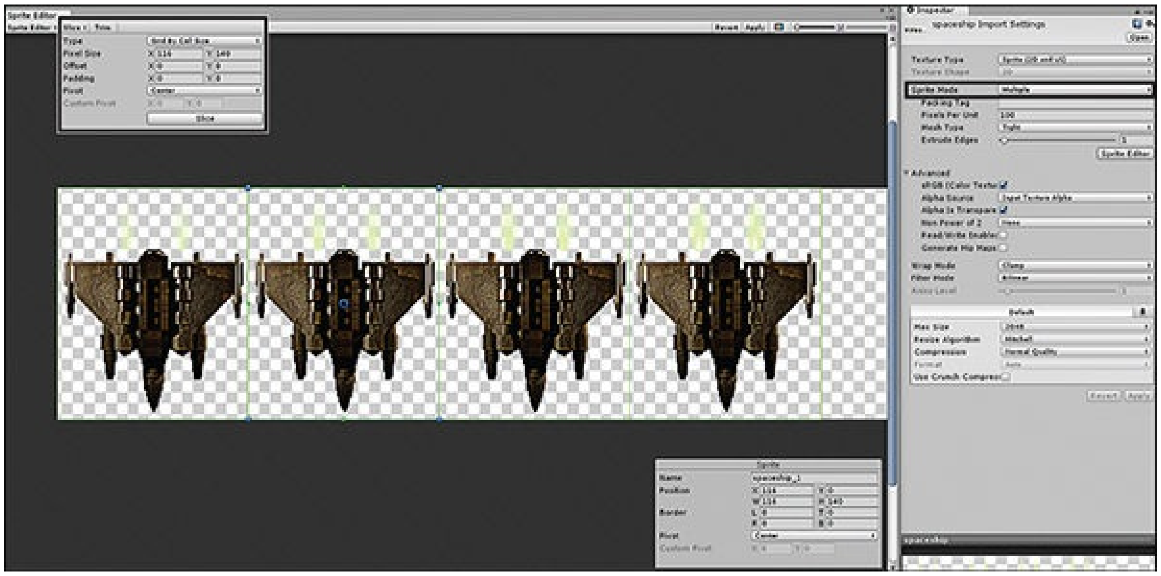


FIGURE 15.3
Slicing the spaceship sprite sheet.

4. Open the tray of the spaceship and select all the frames. You can do this by clicking the first frame, holding down **Shift**, and then clicking the last frame.

5. Drag the sprite frames to the Hierarchy view or Scene view. This causes a Create New Animation dialog to appear, which will save your new animation as an .anim file. Name this animation **Ship**. When you are done, an animated sprite will be added to the scene, and an animator controller and animation clip asset will be added to the Project view as in Figure 15.4. (You will learn more about animation in Hour 17, “Animations,” and Hour 18, “Animators.”)

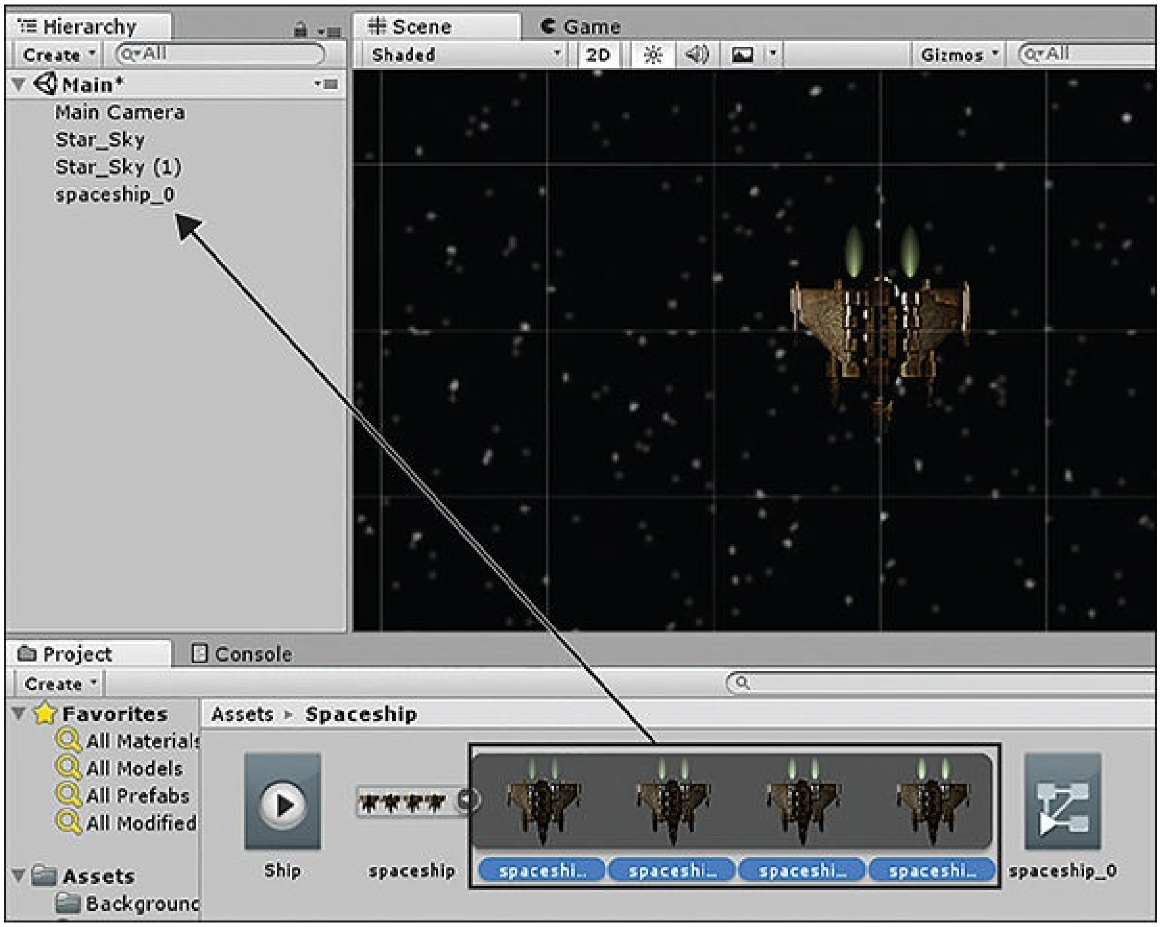


FIGURE 15.4
The finished spaceship sprite.

6. Set the ship's position to (0, -5, 0) and scale it to (1, -1, 1). Note that scaling to -1 in the y axis turns the ship to face upward.

7. On the ship's Sprite Renderer component, set the Order in Layer property to **1**. This ensures that the ship always appears in front of the background.

8. Add a Polygon Collider component to the ship (by selecting **Add Component > Physics2D > Polygon Collider**). This collider will automatically surround your ship for very decent collision detection accuracy. Be sure to check the **Is Trigger** property in the Inspector to ensure that it is a trigger collider.

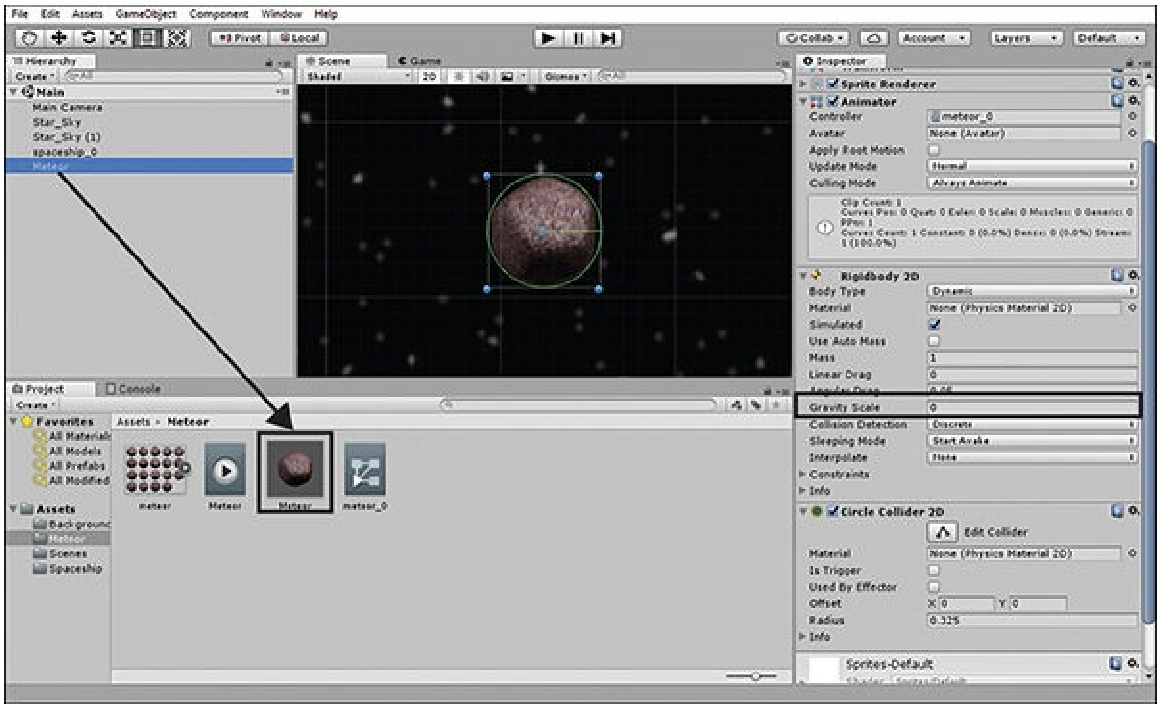
9. Play the game and notice the subtle animation of the ship's engines.

You should now have a nice, animated upward-facing spaceship ready to destroy some meteors.

The Meteors

The steps for creating the meteors are similar to those for creating the spaceship. The only difference is that the meteors will end up in a prefab for later use. Follow these steps:

1. Create a new folder called Meteor and import meteor.png into it. This is a sprite sheet that contains 19 frames of animation.
2. Set Sprite Mode to **Multiple** and then enter the Sprite Editor as before.
3. Set Slice Type to **Automatic** and leave the rest of the settings at their defaults. Click **Apply** to apply your changes and close the Sprite Editor window.
4. Expand the tray of the meteor sprite and select all 19 frames of the meteor sprite sheet. Drag these frames into the Hierarchy view and, when prompted, name the animation **Meteor**. Unity creates another animated sprite with the necessary animation components for you. Handy!
5. Select the meteor_0 game object in the Hierarchy view and add a Circle Collider 2D component (by selecting **Add Component > Physics2D > Circle Collider 2D**). Note that the green outline roughly fits the outline of the sprite. This is good enough to work in the *Captain Blaster* game. A polygon collider would be less efficient and would not give you a noticeable improvement in accuracy.
6. On the meteor’s Sprite Renderer component, set the Order in Layer property to **1** to ensure that the meteor always appears in front of the background.
7. Add a Rigidbody2D component to the meteor (by selecting **Add Component > Physics2D > Rigidbody2D**). Set the Gravity Scale property to **0**.
8. Rename the meteor_0 game object **Meteor** and then drag it from your Hierarchy view into your Meteor folder in the Project view (see Figure 15.5). Unity creates a prefab of the meteor that you will use later.
9. Now that you have captured the meteor setup in a prefab, delete the instance in your Hierarchy view. You now have a reusable meteor just waiting to cause havoc.



Sprite Renderer

Animator

Controller	meteor_0
Avatar	None (Avatar)
Apply Root Motion	☐
Update Mode	Normal
Culling Mode	Always Animate

Clip Count: 1
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 1
Curves Count: 1 Constant: 0 (0.0%) Dense: 0 (0.0%) Stream: 1 (100.0%)

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	0
Collision Detection	Discrete
Sleeping Mode	Start Awake
Interpolate	None

Constraints
Info

Circle Collider 2D

Material	None (Physics Material 2D)
Is Trigger	☐
Used By Effector	☐
Offset	X 0 Y 0
Radius	0.325

Info

Sprites-Default

FIGURE 15.5
Creating the meteor prefab.

The Bullets

Setting up bullets in this game is simple. Because they will be moving very quickly, they don't need any detail. To create the bullet, follow these steps:

1. Create a folder named Bullet and import bullet.png into it. With the new bullet sprite selected in the Project view, set the Pixels per Unit property in the Inspector to **400**.
2. Drag a bullet sprite into your scene. Using the Color property of the Sprite Renderer component, give the bullet a strong green color.
3. On the Sprite Renderer component of the bullets, set the Order in Layer property to **1** to ensure that the bullet always appears in front of the background.
4. Add a Circle Collider 2D component to the bullet. Also add a Rigidbody2D component to the bullet (by selecting **Add Component** > **Physics2D** > **Rigidbody2D**) and set the Gravity Scale property to **0**.
5. To keep with convention, rename the bullet game object **Bullet**. Drag the bullet from the Hierarchy view into your Bullet folder to turn it into a

prefab. Delete the Bullet object from your scene.

That’s the last of the primary entities. The only thing left to make is the triggers that will prevent the bullets and meteor from traveling forever.

The Triggers

The triggers (which in this game will be called “shredders”) are simply two colliders, one of which will sit above the screen and the other below it. Their job is to catch any errant bullets and meteors and “shred” them. Follow these steps to create the shredders:

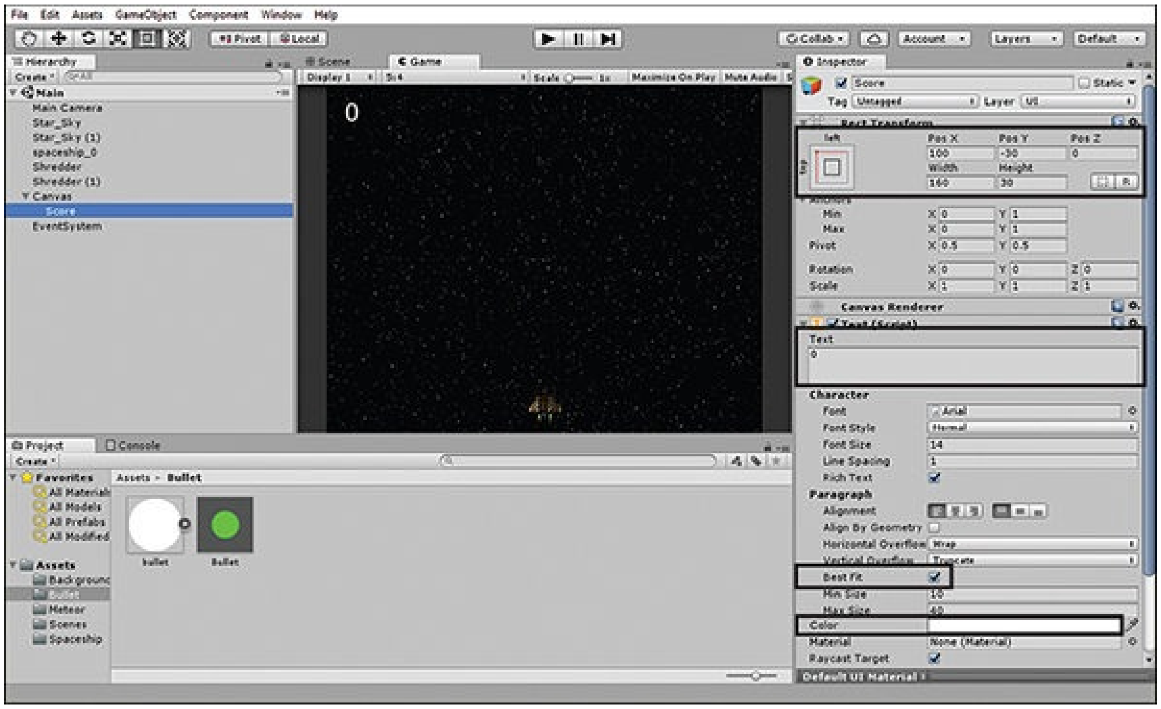
1. Add an empty game object to the scene (by selecting **GameObject > Create Empty**) and name it **Shredder**. Position it at (0, -10, 0).
2. Add a Box Collider 2D component to the shredder object (by selecting **Add Component > Physics2D > Box Collider 2D**). In the Inspector view, be sure to put a check next to the **Is Trigger** property of the Box Collider 2D component and set its size to (16, 1).
3. Duplicate the shredder and place the new one at (0, 10, 0).

Later these triggers will be used to destroy any objects that hit them, such as stray meteors or bullets.

The UI

Finally, you need to add a simple user interface to display the player’s current score and to say “Game Over” when the player dies. Follow these steps:

1. Add a UI Text element to the scene (by selecting **GameObject > UI > Text**) and rename it **Score**.
2. Position the score text’s anchor in the upper-left corner of the canvas and set its position to (100, -30, 0) (see Figure 15.6).



Score Static
Tag Untagged Layer UI

left	Pos X 100	Pos Y -30	Pos Z 0
	Width 160	Height 30	
Anchors Min	X 0	Y 1	
Max	X 0	Y 1	
Pivot	X 0.5	Y 0.5	
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Text
0

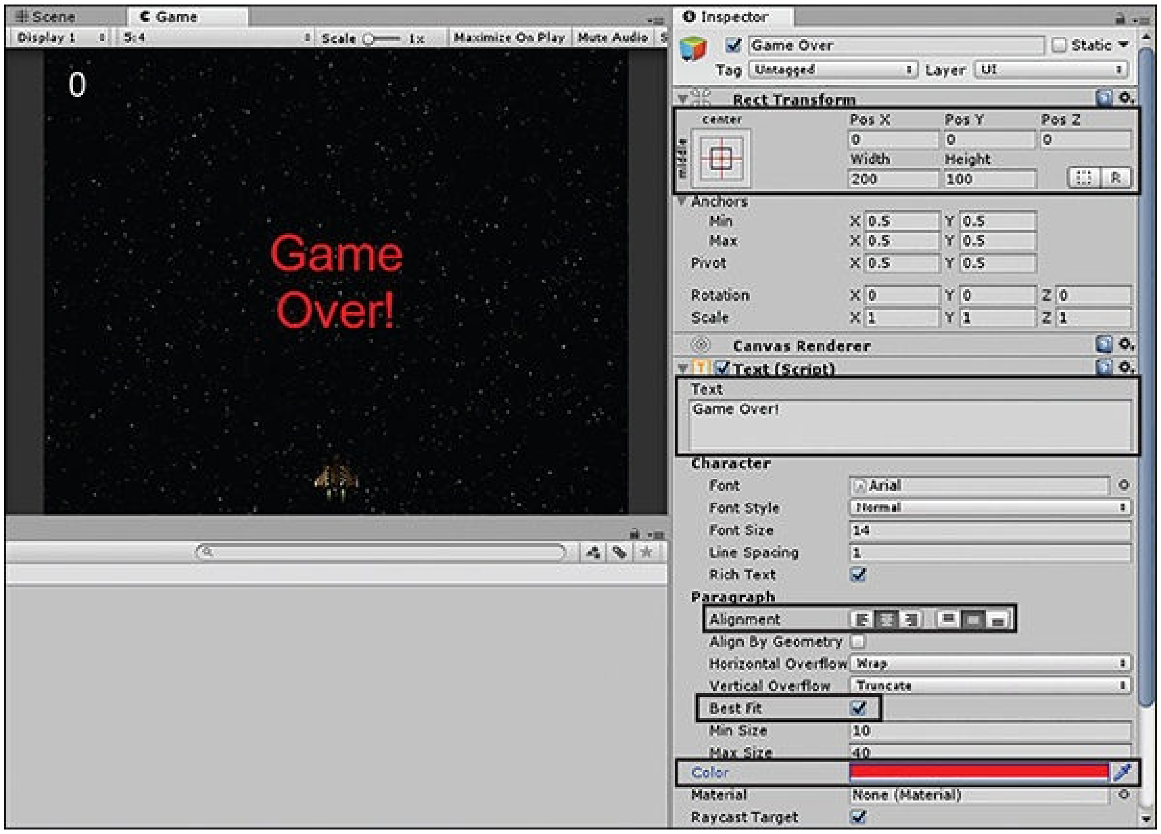
Font	Arial
Font Style	Normal
Font Size	14
Line Spacing	1
Rich Text	✓
Align By Geometry	
Horizontal Overflow	Wrap
Vertical Overflow	Truncate
Best Fit	✓
Min Size	10
Max Size	40
Color	
Material	None (Material)
Raycast Target	✓

FIGURE 15.6
The player score settings.

3. Set the Text property of the Score object to **0** (which is the initial score), check **Best Fit**, and set Color to white.

Now you can add the Game Over text that will appear when the player loses:

1. Add another UI Text element to the scene and rename it **Game Over**. Leave the anchor in the center and set its position to (0, 0, 0).
2. Set the width of the Game Over text object to **200** and set the height to **100**.
3. Change the Text to **Game Over!**, check the **Best Fit** property, set the paragraph alignment to centered, and change the color to red.
4. Finally, uncheck the box next to the Text (Script) component's name to disable the text until it is needed (see Figure 15.7). Note that Figure 15.7 illustrates the text before being disabled so you can see what it looks like.



Game Over Static
Tag Untagged Layer UI

Rect Transform
center middle
Pos X 0 Pos Y 0 Pos Z 0
Width 200 Height 100
Anchors
Min X 0.5 Y 0.5
Max X 0.5 Y 0.5
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Canvas Renderer

Text (Script)
Text
Game Over!
Character
Font Arial
Font Style Normal
Font Size 14
Line Spacing 1
Rich Text ✓
Paragraph
Alignment
Align By Geometry
Horizontal Overflow Wrap
Vertical Overflow Truncate
Best Fit ✓
Min Size 10
Max Size 40
Color 🟥
Material None (Material)
Raycast Target ✓

FIGURE 15.7
The Game Over! sign settings.

Later you will connect this score display to your GameManager script so that it can be updated. Now all your entities are in place, and it is time to begin turning this scene into a game.

Controls

Various script components need to be assembled to make this game work. The player needs to be able to move the ship and shoot bullets. The bullets and meteors need to be able to move automatically. A meteor spawn object must keep the meteors flowing. The shredders need to be able to clean up objects, and a manager needs to keep track of all the action.

The Game Manager

The game manager is basic in this game, and you can add it first. To create the game manager, follow these steps:

1. Create an empty game object and name it **GameManager**.