fix any basic problems with your terrain.

TIP

Falling Off the World

Generally, game levels have walls or some other obstacle in place to prevent the player from exiting the developed area. If a game employs gravity, the player may fall off the side of the world. You always want to create some way to prevent players from going somewhere they shouldn’t. This game project uses a tall berm to keep the players in the play area. The heightmap provided to you in the book’s assets for Hour 6 intentionally has a few places where the player can climb out. See if you can find and correct them. You can also set a slope limit for FPSController in the Inspector, as explained earlier in this hour.

Gamification

You now have a world in which your game can take place. You can run around and experience the world to an extent. The piece that is missing is the game itself. Right now, what you have is considered a toy. It is something that you can play with. What you want is a game, which is a toy that has rules and a goal. The process of turning something into a game is called *gamification*, and that’s what this section is all about. If you followed the previous steps, your game project should now look something like Figure 6.2 (though your choices for fog, skybox, and vegetation may create some differences). The next few steps are to add game control objects for interaction, apply game scripts to those objects, and connect them to each other.

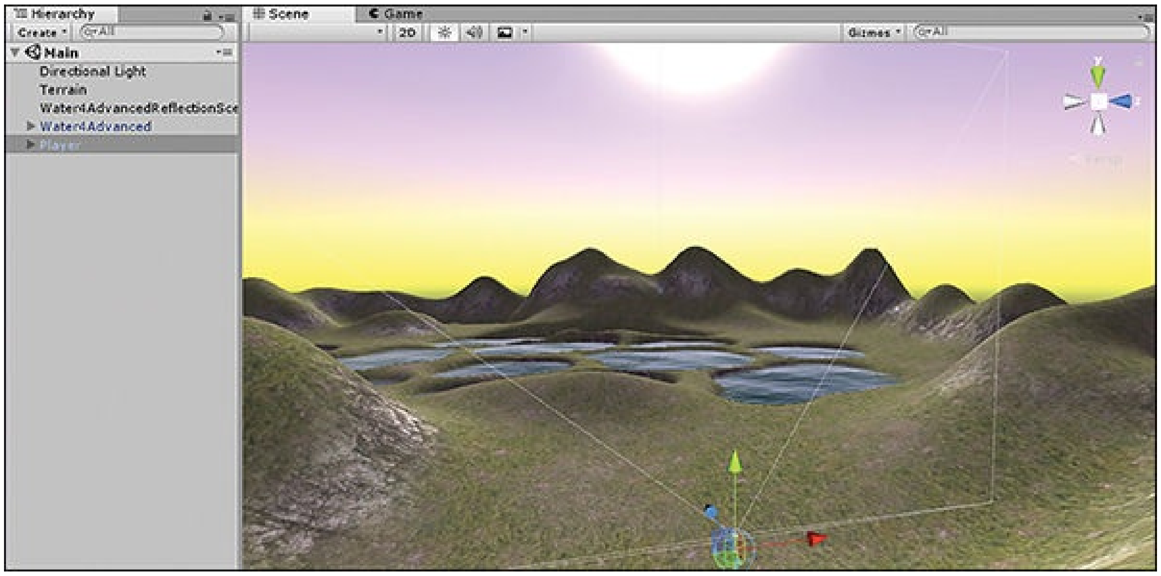


FIGURE 6.2
The current state of the *Amazing Racer* game.

NOTE

Scripts

Scripts are pieces of code that define behaviors for game objects. You have not yet learned about scripting in Unity. To make an interactive game, however, scripts are a must. With this in mind, the scripts needed to make this game have been provided for you. These scripts are as minimal as possible so that you can understand most of this project. Feel free to open the scripts in a text editor and read them to see what they are doing. Scripts are covered in greater detail in Hour 7, “Scripting, Part 1,” and Hour 8, “Scripting, Part 2.”

Adding Game Control Objects

As defined earlier in this hour, in the section “The Requirements,” you need four specific game control objects. The first object is a spawn point—a simple game object that exists solely to tell the game where to spawn the player. To create the spawn point, follow these steps:

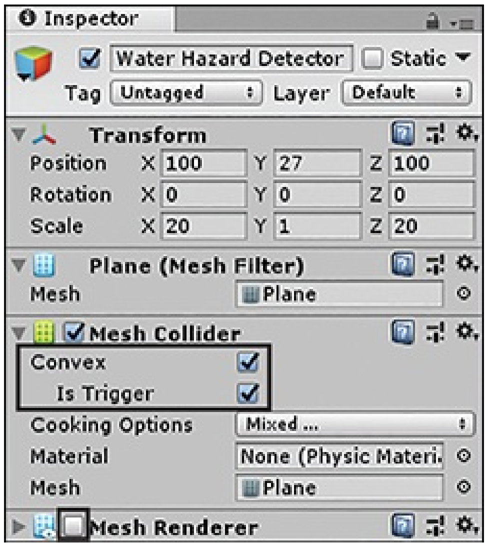
1. Add an empty game object to the scene (by selecting **GameObject > Create Empty**).
2. Position the game object at (165, 32, 125) and give it a rotation of (0,

260, 0).

3. Rename the empty object **Spawn Point** in the Hierarchy view.

Next, you need to create the water hazard detector. This will be a simple plane that will sit just below the water. The plane will have a trigger collider (as covered in more detail in Hour 9, “Collision”), which will detect when a player has fallen in the water. To create the detector, follow these steps:

1. Add a plane to the scene (by selecting **GameObject > 3D Object > Plane**) and position it at (100, 27, 100). Scale the plane to (20, 1, 20).
2. Rename the plane **Water Hazard Detector** in the Hierarchy view.
3. Check the **Convex** and **Is Trigger** check boxes on the Mesh Collider component in the Inspector view (see Figure 6.3).
4. Make the object invisible by disabling its Mesh Renderer component. Do this by unchecking the box next to the Mesh Renderer component’s name in the Inspector (see Figure 6.3).



✓ Water Hazard Detector ☐ Static
Tag Untagged Layer Default

Transform
Position X 100 Y 27 Z 100
Rotation X 0 Y 0 Z 0
Scale X 20 Y 1 Z 20

Plane (Mesh Filter)
Mesh Plane

✓ Mesh Collider
Convex ✓
Is Trigger ✓
Cooking Options Mixed ...
Material None (Physic Materi.
Mesh Plane

☐ Mesh Renderer

FIGURE 6.3
The Inspector view of the Water Hazard Detector object.

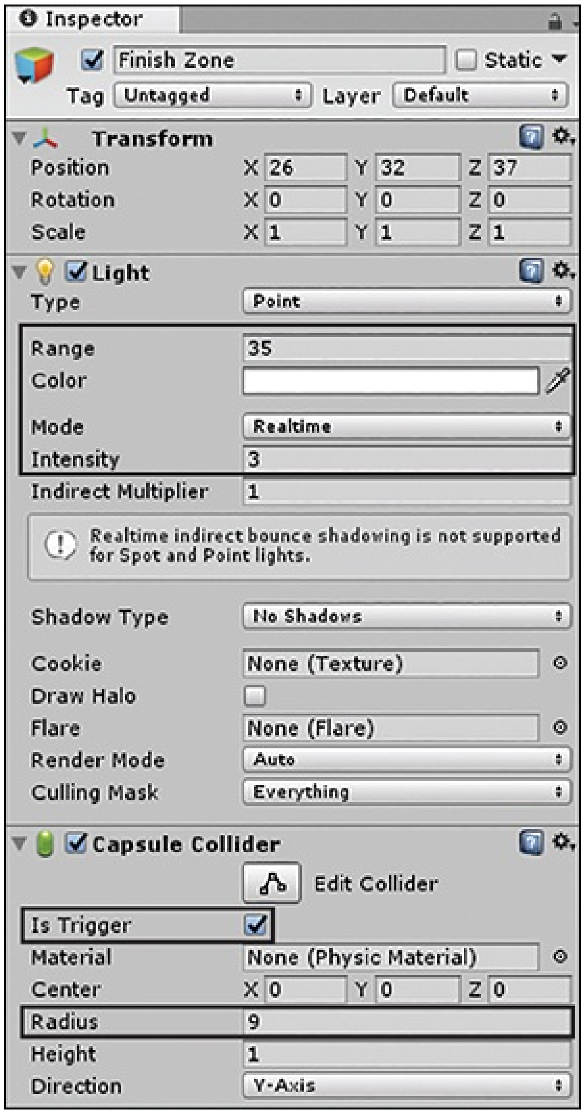
Next, you need to add the finish zone to your game. This zone will be a simple object with a point light on it so that the player knows where to go. The object will have a capsule collider attached to it so that it will know when a player enters the zone. To add the **Finish Zone** object, follow these steps:

1. Add an empty game object to the scene and position it at (26, 32, 37).

2. Rename the object **Finish Zone** in the Hierarchy view.

3. Add a light component to the Finish Zone object. (With the object selected, click **Component > Rendering > Light**.) Change the type to **Point** if it isn't already set to this and set the range to **35** and intensity to **3**.

4. **Add** a capsule collider to the Finish Zone object by selecting the object and clicking **Component > Physics > Capsule Collider**. Check the **Is Trigger** check box and change the Radius property to **9** in the Inspector view (see Figure 6.4).



```
Inspector
[✓] Finish Zone                      [ ] Static
Tag Untagged          Layer Default

Transform
Position      X 26   Y 32   Z 37
Rotation      X 0    Y 0    Z 0
Scale         X 1    Y 1    Z 1

[✓] Light
Type                 Point
Range                35
Color
Mode                 Realtime
Intensity            3
Indirect Multiplier  1
  Realtime indirect bounce shadowing is not supported
  for Spot and Point lights.
Shadow Type          No Shadows
Cookie               None (Texture)
Draw Halo            [ ]
Flare                None (Flare)
Render Mode          Auto
Culling Mask         Everything

[✓] Capsule Collider
                     Edit Collider
Is Trigger           [✓]
Material             None (Physic Material)
Center               X 0   Y 0   Z 0
Radius               9
Height               1
Direction            Y-Axis
```

FIGURE 6.4
The Inspector view of the Finish Zone object.

The final object you need to create is the Game Manager object. This object doesn’t technically need to exist. You could instead just apply its properties to some other persistent object in the game world, such as the Main Camera. You generally create a dedicated game manager object to prevent any accidental deletion, though. During this phase of development, the game manager object is very basic. It will be used more later on. To create the Game Manager object, follow these steps:

1. Add an empty game object to the scene.
2. Rename the game object **Game Manager** in the Hierarchy view.

Adding Scripts

As mentioned earlier, scripts specify behaviors for game objects. In this section, you’ll apply scripts to your game objects. At this point, it is not important for you to understand what these scripts do. You can add scripts to your project in one of two ways:

- Drag existing scripts into the Project view of your project.
- Create new scripts in your project by right-clicking in the Project view and selecting **Create > C# Script**.

Once scripts are in a project, applying them is easy. To apply a script, simply drag it from the Project view onto whatever object you want to apply it to in either the Hierarchy view or the Inspector view (see Figure 6.5).

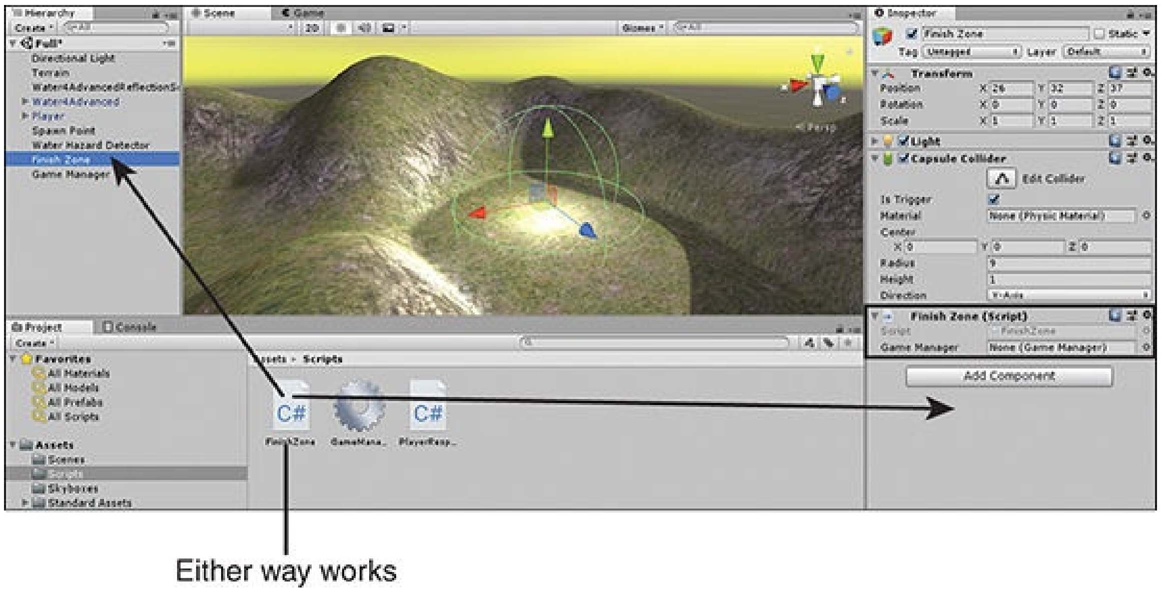


FIGURE 6.5

Applying scripts by dragging them onto game objects.

You could also apply a script by dragging it onto an object in the Scene view, but if you do, you run the risk of missing and accidently putting the script on some other object. Therefore, applying scripts through the Scene view is not advisable.

TIP

The Special Script Icon

You may have noticed that the GameManager script has a different, gear-shaped, icon in the Project view. This is due to the script having a name that Unity automatically recognizes: GameManager. There are a few specific names that, when used for scripts, change the icon and make it easier to identify.

▼ TRY IT YOURSELF

Importing and Attaching Scripts

Follow these steps to import the scripts from the book files and attach them to the correct objects:

1. Create a new folder in your Project view and name it **Scripts**. Locate the three scripts in the Hour 6 book files—FinishZone.cs, GameManager.cs, and PlayerRespawn.cs—and drag them into your newly created Scripts folder.
2. Drag the FinishZone.cs script from the Project view onto the Finish Zone game object in the Hierarchy view.
3. Select the Game Manager object in the hierarchy. In the Inspector, select **Add Component > Scripts > GameManager**. (This is an alternative way of adding a script component to a game object.)
4. Drag the PlayerRespawn.cs script from the Project view onto the Water Hazard Detector game object in the hierarchy.

Connecting the Scripts

If you read through the scripts earlier, you may have noticed that they all have placeholders for other objects. These placeholders allow one script to talk to another script. For every placeholder in these scripts, there is a property in the component for that script in the Inspector view. Just as with scripts, you apply the objects to the placeholders by clicking and dragging (see Figure 6.6).

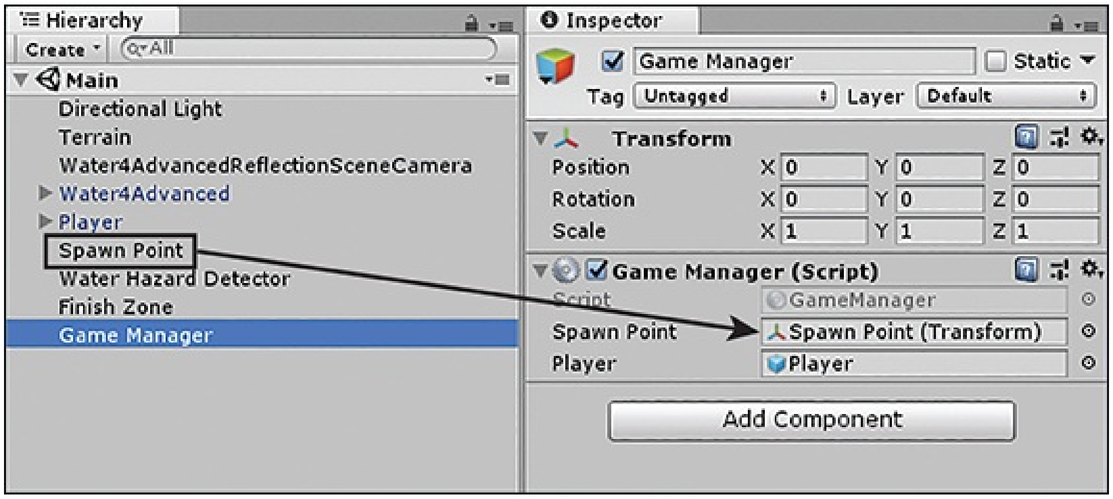


FIGURE 6.6
Moving game objects onto placeholders.

▼ TRY IT YOURSELF

Connecting Scripts to Scripts

Follow these steps to give the scripts the game objects they need to function correctly:

1. Select the **Water Hazard Detector** object in the Hierarchy view. Notice that the Player Respawn component has a Game Manager property. This property is a placeholder for the Game Manager object you made previously.

2. Click and drag the **Game Manager** object from the Hierarchy view onto the Game Manager property of the Player Respawn (Script) component. Now, whenever players fall into the water hazard, the water hazard will let the Game Manager object know, and the player will get moved back to the spawn point at the beginning of the level.

3. Select the **Finish Zone** game object. Click and drag the **Game Manager** object from the Hierarchy view onto the Game Manager

property of the Finish Zone (Script) component in the Inspector view. Now, whenever the player enters the finish zone, the game control will be notified.

4. Select the **Game Manager** object. Click and drag the **Spawn Point** object onto the Spawn Point property of the Game Manager (Script) component.
5. Click and drag the **Player** object (this is the character controller) onto the Player property of the Game Manager object.

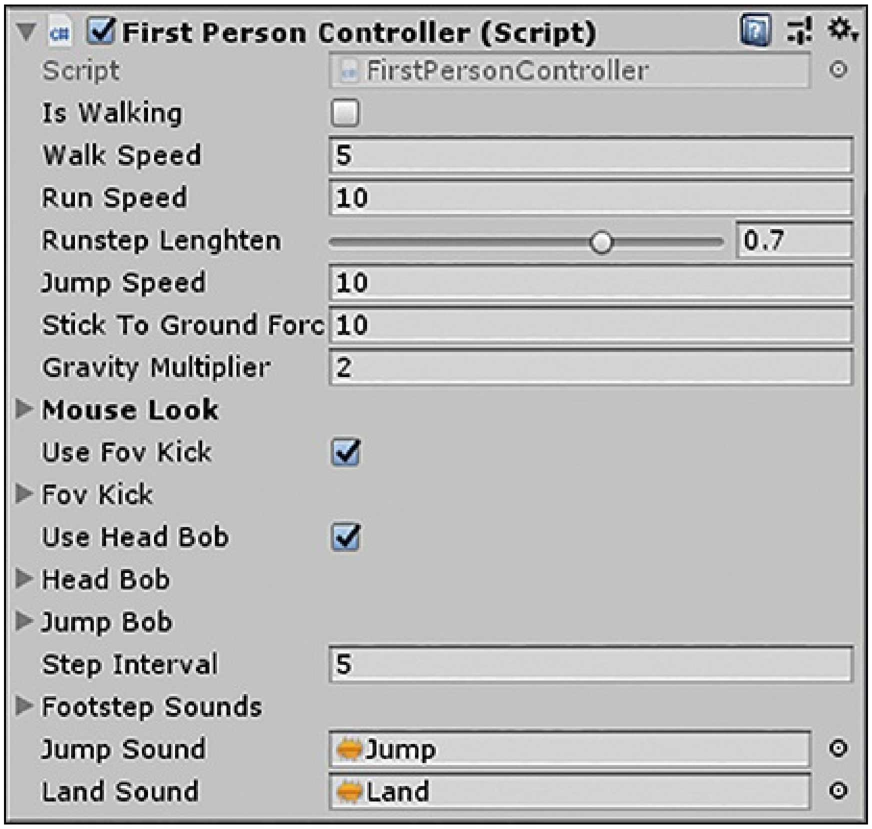
That's all there is to connecting the game objects. Your game is now completely playable! Some of this might not make sense right now, but the more you study it and work with it, the more intuitive it becomes.

Playtesting

Your game is now done, but it is not time to rest just yet. Now you have to begin the process of playtesting. Playtesting involves playing a game with the intention of finding errors or things that just aren't as fun as you thought they would be. A lot of times, it can be beneficial to have other people playtest your games so that they can tell you what makes sense to them and what they found enjoyable.

If you followed all the steps previously described, there shouldn't be any errors (commonly called *bugs*) for you to find—at least, I hope so. The process of determining what parts are fun, however, is completely at the discretion of the person making the game. Therefore, this part is left up to you. Play the game and see what you don't like. Take notes of the things that aren't enjoyable to you. Don't just focus on the negative, though. Also find the things that you like. Your ability to change these things may be limited at the moment, so write them down. Plan on how you would change the game if you had the opportunity.

One simple thing you can tweak right now to make the game more enjoyable is the player's speed. If you have played the game a couple of times, you might have noticed that the character moves rather slowly, and that can make the game feel very long and drawn out. To make the character fast, you need to modify the First Person Controller (Script) component on the Player object. Expand the **Movement** property in the Inspector view and change the run speed (see Figure 6.7). The sample project has this set at 10. Try faster or slower speeds and pick one you enjoy. (You did notice that holding down **Shift** as you play makes you run, right?)



First Person Controller (Script)

Script	FirstPersonController
Is Walking	☐
Walk Speed	5
Run Speed	10
Runstep Lenghten	0.7
Jump Speed	10
Stick To Ground Forc	10
Gravity Multiplier	2
Mouse Look	
Use Fov Kick	✓
Fov Kick	
Use Head Bob	✓
Head Bob	
Jump Bob	
Step Interval	5
Footstep Sounds	
Jump Sound	Jump
Land Sound	Land

FIGURE 6.7
Changing the player’s walk and run speeds.

TIP

Where’s My Mouse?

To avoid having an annoying mouse cursor flying around while you are trying to play the game, the First Person Character Controller (the FPSController game object) hides and “locks” it. This is great when playing, except when you need your mouse to click a button or stop playing. If you need your mouse cursor back, you can simply press the **Escape** key to unlock and show it. Alternatively, if you need to leave play mode, you can do so by pressing **Ctrl+P** (**Command+P** on a Mac).

Summary

In this hour, you made your first game in Unity. You started by looking at the various aspects of the game’s concept, rules, and requirements. From there, you built the game world and added environment effects. Then you added the game objects required for interactivity. You applied scripts to those game objects and connected them. Finally, you playtested your game and noted the things you liked and didn't like.

Q&A

Q. This lesson seems over my head. Am I doing something wrong?

A. Not at all! This process can feel very alien to someone who is not used to it. Keep reading and studying the materials, and it will all begin to come together. The best thing you can do is pay attention to how the objects connect through the scripts.

Q. You didn't cover how to build and deploy the game. Why not?

A. Building and deployment are covered later in this book, in Hour 23, "Polish and Deploy." There are many things to consider when building a game, and at this point, you should just focus on the concepts required to develop it.

Q. Why couldn't I make a game without scripts?

A. As mentioned earlier, scripts define the behavior of objects. It is very difficult to have a coherent game without some form of interactive behavior. The only reason you are building a game in Hour 6 before learning scripting in Hours 8 and 9 is to reinforce the topics you have already learned before moving on to something different.

Workshop

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

Quiz

1. What are a game's requirements?
2. What is the win condition of the *Amazing Racer* game?
3. Which object is responsible for controlling the flow of the game?
4. Why is it important to playtest a game?

Answers

1. The requirements are the list of assets that will need to be created to make the game.
2. Trick question! There is no explicit win condition for this game. It is assumed that the player wins when he or she gets a better time than previous attempts. This is not built into the game in any way, though.
3. The game manager, which in the *Amazing Racer* game is called Game Manager
4. To discover bugs and determine what parts of the game work the way you want them to

Exercise

The best part about making games is that you get to make them the way you want. Following a guide can be a good learning experience, but you don’t get the satisfaction of making a custom game. For this exercise, modify the *Amazing Racer* game a little to make it more unique. Exactly how you change the game is up to you. Some suggestions are listed here:

- Try to add multiple finish zones. See whether you can place them in a way that offers the players more choices.
- Modify the terrain to have more or different hazards. As long as the hazards are built like the water hazard (including the script), they will work just fine.
- Try having multiple spawn locations. Make it so some of the hazards move the player to a different spawn point.
- Modify the sky and textures to create an alien world. Make the world experience unique.