HOUR 6

Game 1: *Amazing Racer*

What You'll Learn in This Hour:

- How to design a basic game
- How to apply your knowledge of terrains to build a game-specific world
- How to add objects to a game to provide interactivity
- How to playtest and tweak a finished game

In this hour, you'll use what you have learned so far to build your first Unity game. This lesson starts by covering the basic design elements of the game. From there, you will build the world in which the game will take place. Then you'll add some interactivity objects to make the game playable. You'll finish by playing the game and making any necessary tweaks to improve the experience.

TIP

Completed Project

Be sure to follow along in this hour to build the complete game project. If you get stuck, look for the completed copy of the game in the book assets for Hour 6. Take a look at it if you need help or inspiration!

Design

The design portion of game development is where you plan ahead of time all the major features and components of a game. You can think of it as laying down the blueprint so that the actual construction process is smoother. When making a

game, a lot of time is normally spent working through the design. Because the game you are making in this hour is fairly basic, the design phase will go quickly. You need to focus on three areas of planning to make this game: the concept, the rules, and the requirements.

The Concept

The idea behind this game is simple: You start at one end of an area and run quickly to the other side. There are hills, trees, and obstacles in your path. Your goal is to see how fast you can make it to the finish zone. This game concept was chosen for your first game because it highlights everything you have worked on so far in this book. Also, because you have not learned scripting in Unity yet, you cannot add very elaborate interactions. Games you make in later hours will be more complex.

The Rules

Every game must have a set of rules. The rules serve two purposes. First, they say how the player will actually play the game. Second, because software is a process of permission (see the "Process of Permission" note), the rules dictate the actions available to the players to overcome challenges. The rules for *Amazing Racer* are as follows:

- There is no win or loss condition, only a completed condition. The game is completed when the player enters the finish zone.
- The player always spawns in the same spot. The finish zone is always in the same spot.
- There are water hazards, and whenever the player falls into one of them, the player is moved back to the spawn point.
- The objective of the game is to try to get the fastest time possible. This is an implicit rule and is not specifically built into the game. Instead, cues are built into the game as hints to the player that this is the goal. The idea is that the players will intuit the desire for a faster time based on the signals given to them.

NOTE

Process of Permission

Something to always remember when making a game is that software is a process of permission. What this means is that unless you specifically allow

process of permission. What this means is that unless you specifically allow something, it is unavailable to the player. For instance, if the player wants to climb a tree, but you have not created any way for the player to climb a tree, that action is not permitted. If you do not give players the ability to jump, they can't jump. Everything that you want the player to be able to do must be explicitly built in. Remember that you cannot assume any action and must plan for everything! Also remember that players can combine actions in inventive ways—for example, stacking blocks and then jumping from the top block—if you make them possible.

NOTE

Terminology

Some new terms are used in this hour:

- **Spawn:** Spawning is the process by which a player or an entity enters a game.
- **Spawn point:** A spawn point is the place where a player or an entity spawns. There can be one or many of these. They can be stationary or moving around.
- **Condition:** A condition is a form of a trigger. A win condition is the event that causes the player to win the game (such as accumulating enough points). A loss condition is an event that causes the player to lose the game (such as losing all health points).
- **Game manager:** The game manager dictates the rules and flow of a game. It is responsible for knowing when the game is won or lost (or just over). Any object can be designated as the game manager, as long as it is always in the scene. Often, an empty object or the Main Camera is designated as the game manager.
- **Playtesting: Playtesting** is the process of having actual players play a game that is still in development to see how they react to the game so it can be improved accordingly.

The Requirements

An important step in the design process is determining which assets will be required for the game. Generally speaking, a game development team is made up of several individuals. Some of them handle designing, others work with

programming, and others make art. Every member of the team needs something to do to be productive during every step of the development process. If everyone waited until something was needed to begin working, there would be a lot of starting and stopping. Instead, you determine your assets ahead of time so that things can be created before they are needed. Here is a list of all the requirements for *Amazing Racer*:

- A piece of rectangular terrain. The terrain needs to be big enough to present a challenging race. The terrain should have obstacles built in as well as a designated spawn and finish point (see Figure 6.1).
- Textures and environment effects for the terrain. These are provided in the Unity standard assets.
- A spawn point object, a finish zone object, and a water hazard object. These will be generated in Unity.
- A character controller. This is provided by the Unity standard assets.
- A graphical user interface (GUI). This will be provided for you in the book assets. Note that this hour uses the old-style GUI, which works purely from script, for simplicity. In your projects, use the new UI system introduced in Hour 14, “User Interfaces.”
- A game manager. This will be created in Unity.

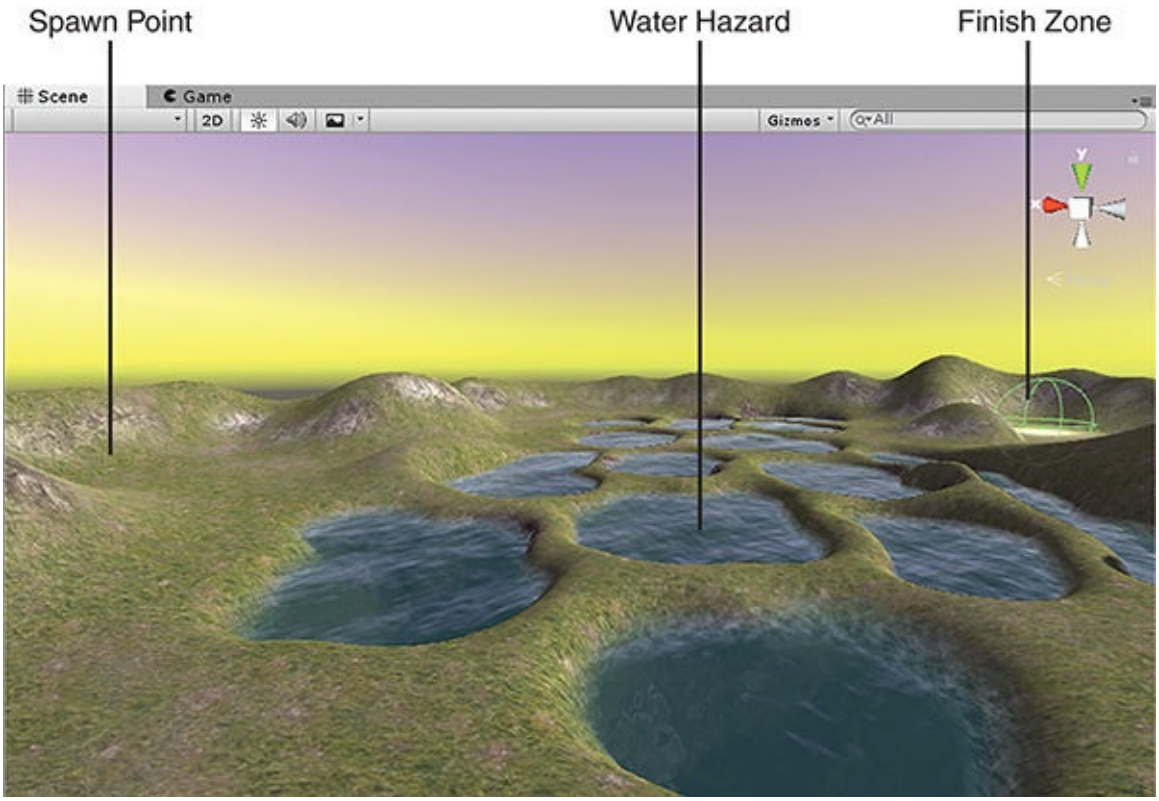

FIGURE 6.1
The general terrain layout for the game *Amazing Racer*.

Creating the Game World

Now that you have a basic idea of the game, it is time to start building it. There are many places to begin building a game. For this project, you'll begin with the world. Because this is a linear racing game, the world will be longer than it is wide (or wider than it is long, depending on how you look at it). You will use many of the Unity standard assets to rapidly create the game.

Sculpting the World

There are many ways you can create the terrain for *Amazing Racer*. Everyone will probably have a different vision for it in his or her head. To streamline the process and ensure that everyone will have the same experiences during this hour, a heightmap has been provided for you. To sculpt the terrain, follow these steps:

1. Create a new project and name it **Amazing Racer**. Add a terrain to the project and position its transform at (0, 0, 0) in the Inspector.

2. Locate the file TerrainHeightmap.raw in the book assets for Hour 6. Import the TerrainHeightmap.raw file as a heightmap for the terrain (by clicking **Import Raw** in the Heightmap section under Terrain Settings in the Inspector).

3. Leave the Depth, Width, and Height settings as they are. Change Byte Order to **Mac** and set Terrain Size to **200** wide by **100** long by **100** tall.

4. Create a **Scenes** folder under Assets and save the current scene as **Main**.

The terrain should now be sculpted to match the world in the book. Feel free to make minor tweaks and changes to your liking.

CAUTION

Building Your Own Terrain

In this hour, you are building a game based on a heightmap that has been provided for you. The heightmap has been prepared for you so that you can quickly get through the process of game development. You may, however, choose to build your own custom world to make this game truly unique and yours. If you do that, however, be warned that some of the coordinates and rotations provided for you in this hour might not match up. If you want to build your own world, pay attention to the intended placement of objects and position them in your world accordingly.

Adding the Environment

At this point, you can begin texturing and adding the environment effects to your terrain. You need to import the Environment package (by selecting **Assets > Import Package > Environment**).

You now have a bit of freedom to decorate the world however you would like. The suggestions in the following steps are guidelines. Feel free to do things in a manner that looks good to you:

1. Rotate the directional light to suit your preference.

2. Texture the terrain. The sample project uses the following textures: **GrassHillAlbedo** for flat parts, **CliffAlbedoSpecular** for steep parts, **GrassRockyAlbedo** for the areas in between, and **MudRockyAlbedoSpecular** for inside the pits.

3. Add trees to your terrain. Trees should be placed sparsely and mostly on flat surfaces.

4. Add some water to your scene from the Environment assets. Locate the **Water4Advanced** prefab in the folder Assets\Standard Assets\Environment\Water\Water4\Prefabs and drag it into your scene. (You'll learn more info about prefabs in Hour 11, "Prefabs.") Position the water at (100, 29, 100) and scale it to (2, 1, 2).

The terrain should now be prepared and ready to go. Be sure to spend a good amount of time on texturing to make sure you have a good blend and a realistic look.

Fog

In Unity, you can add fog to a scene to simulate many different natural occurrences, such as haze, actual fog, or the fading of objects over great distances. You can also use fog to give new and alien appearances to your world. In the *Amazing Racer* game, you can use fog to obscure the distant parts of the terrain and add an element of exploration.

Adding fog is quite simple:

1. Select **Window > Lighting > Settings**. The Lighting window opens.

2. Turn on fog by checking the **Fog** check box under Other Settings.

3. Change the color to **white** and set the density to **0.005**. (*Note:* These are arbitrary values and can be changed (or omitted) based on your preferences.)

4. Experiment with the different fog densities and colors. Table 6.1 describes the various fog properties.

Several properties impact how fog looks in a scene. Table 6.1 describes these properties.

TABLE 6.1 Fog Properties

Setting	Description
Fog Color	Specifies the color of the fog effect.
Fog	Controls how the fog is calculated. The three modes are Linear,

Mode	Exponential, and Exponential Squared. For mobile, Linear works best.
Density	Determines how strong the fog effect is. This property is used only if the fog mode is set to Exponential or Exponential Squared.
Start and End	Control how close to the camera the fog starts and how far from the camera it ends. These properties are used only in Linear mode.

Skyboxes

You can add some punch to a game by adding a skybox. A skybox is a large box that goes around a world. Even though it is a cube consisting of six flat sides, it has inward-facing textures to make it look round and infinite. You can create your own skyboxes or use Unity's standard skybox, which comes enabled in every 3D scene. For the most part, in this book, you will use the built-in ones.

The standard skybox is called a "procedural skybox." This means the color is not fixed but is instead calculated and can change. You can see this by rotating your scene's directional light. Notice how the color of the sky, and even the simulated sun, changes as the light is rotated. Procedural skyboxes key off a scene's main directional light by default.

Creating and applying your own custom procedural skybox is quite simple:

1. Right-click the Project view and select **Create > Material**. (Skyboxes are really just materials applied to a "giant box in the sky.")
2. In the Inspector for this new material, click the Shader drop-down and select **Skybox > Procedural**. Note that this is where you can also choose to create 6 Sided, Cubemap, or Panoramic skyboxes.
3. Apply the skybox to the scene through the Lighting settings window (which you open by selecting **Window > Lighting > Settings**). Alternatively, you can just drag your skybox material into any empty space in the scene view. When you apply the skybox to your scene, you don't immediately see any change. The skybox you just created has the same properties as the default skybox and thus will look exactly the same.
4. Experiment with the different skybox properties. You can modify how the sun looks, experiment with how light scatters in the atmosphere, and

change the sky color and exposure. Feel free to make something truly alien and unique for your game.

Skyboxes don’t have to be procedural. They can use six textures to create highly detailed skies (often called *cubemaps*). They could even contain HDR or panorama images. All these settings are available depending on the type of skybox shader you choose. For the most part, though, you will be working with procedural skyboxes in this book because they are easy to get up and running quickly.

The Character Controller

At this stage of development, add a character controller to your terrain:

1. Import the standard character controllers by selecting **Assets > Import Package > Characters**.
2. Locate the **FPSController** asset in the folder Assets\Standard Assets\Characters\FirstPersonCharacter\Prefabs and drag it into your scene.
3. Position the controller (which is named FPSController and is blue in the Hierarchy view) at (165, 32, 125). If the controller doesn’t seem to be positioned correctly on the terrain, ensure that the terrain is positioned at (0, 0, 0), as per the previous exercise. Now rotate the controller to 260 on the y axis so that it faces the correct direction. Rename the controller object **Player**.
4. Experiment with the First Person Controller and Character Controller components on the Player game object. These two components control much of how the player will be able to behave in your game. For instance, if your character is able to climb over hills that you want to be impassable, you could lower the **Slope Limit** property on the Character Controller component.
5. Because the Player controller has its own camera, delete the Main Camera from the scene.

Once the character controller is in your scene and positioned, play the scene. Be sure to move around and look for any areas that need to be fixed or smoothed. Pay attention to the borders. Look for any areas where you are able to escape the world. Those places need to be raised, or the controller needs to be modified, so that the player cannot fall off the map. This is the stage at which you generally

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.