Chapter 3

Scene Navigation and Physics

At this point, you have become familiar with creating, positioning, and manipulating various assets in a scene. The next logical step is to be able to experience your environment as a player. This is how you will be able to test the flow of your scene design and, as your game progresses, test functionality and game play.

Scene Navigation

Obviously, not all games will feature first-person or third-person navigation, but they are both a staple of the game industry and are important to be familiar with. In this chapter, you will be introduced to the First Person Controller and a lot of the concepts related to character navigation in general.

First Person Controller

The quickest way to get up and running in a scene is to create a First Person Controller. When you created the project, you imported the Character Controllers package. It contains scripts and assets for both a First Person Controller and a Third Person Controller. The Third Person Controller is somewhat outdated since the inclusion of the Mecanim system, but the First Person Controller is one of the most valuable assets immediately available. Let's get started.

1. Open the UnityTest project you started in the last chapter.
2. Open the Terrain Test scene, and position the Scene view to somewhere along the path you created in the second chapter.
3. Select the All Prefabs filter in the Project view.
4. Select the First Person Controller, and drag it into the scene view.
5. Drag it around, and then position it on the path.

The First Person Controller automatically is placed on the terrain. Its pivot point intersects the ground.

6. Move it up so the bottom of its capsule is just above the ground.

If it is too low, it will fall through the ground when you press Play. If it is a little high, it will fall down to ground level.

To see what the First Person Controller sees, you will have to switch to the Game view. Now would be a good time to change the UI layout so you can see both at once.

1. From the Layout drop-down menu in the top right corner, click the down arrow and select the 2 x 3 layout.

The Scene and Game views are both visible at the same time (Figure 3-1).

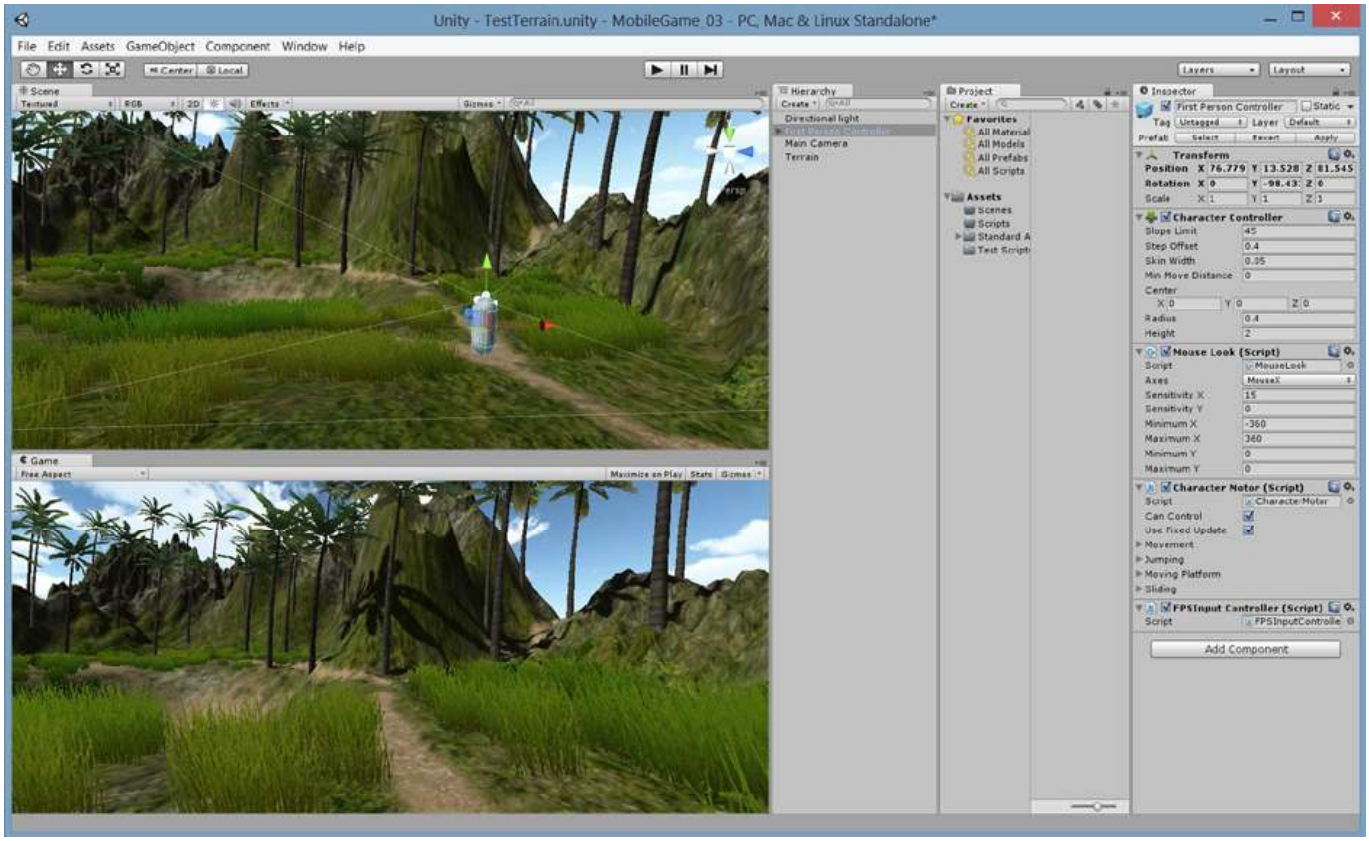

Figure 3-1. *The Scene and Game views visible at the same time*

The Project view will be easier to manage with the more compact One Column view, especially as you will have little need of the Project view for this chapter.

2. Right-click over the Project tab, and switch to the One Column Layout.
3. Select the First Person Controller in the Hierarchy view, and double-click or use Alt+F to frame the view to it.
4. Set the coordinate system to Local.

You can now see the view through the camera in the Game view The First Person Controller group contains a camera, so you will be able to see which way the First Person Controller is facing. With the coordinate system set to Local, you can also see that the First Person Controller faces in the

positive Z direction, Unity's "forward" direction. With the second camera, you now also have two Audio Listeners according to the message on the status line at the bottom of the Game view and must get rid of one of them.

5. Delete the Main Camera from the Hierarchy view by selecting it and pressing the Delete key or selecting Delete from the right-click menu.

6. Orbit the Scene view so that you can see the First Person Controller from something near a top vantage point (Figure 3-2).

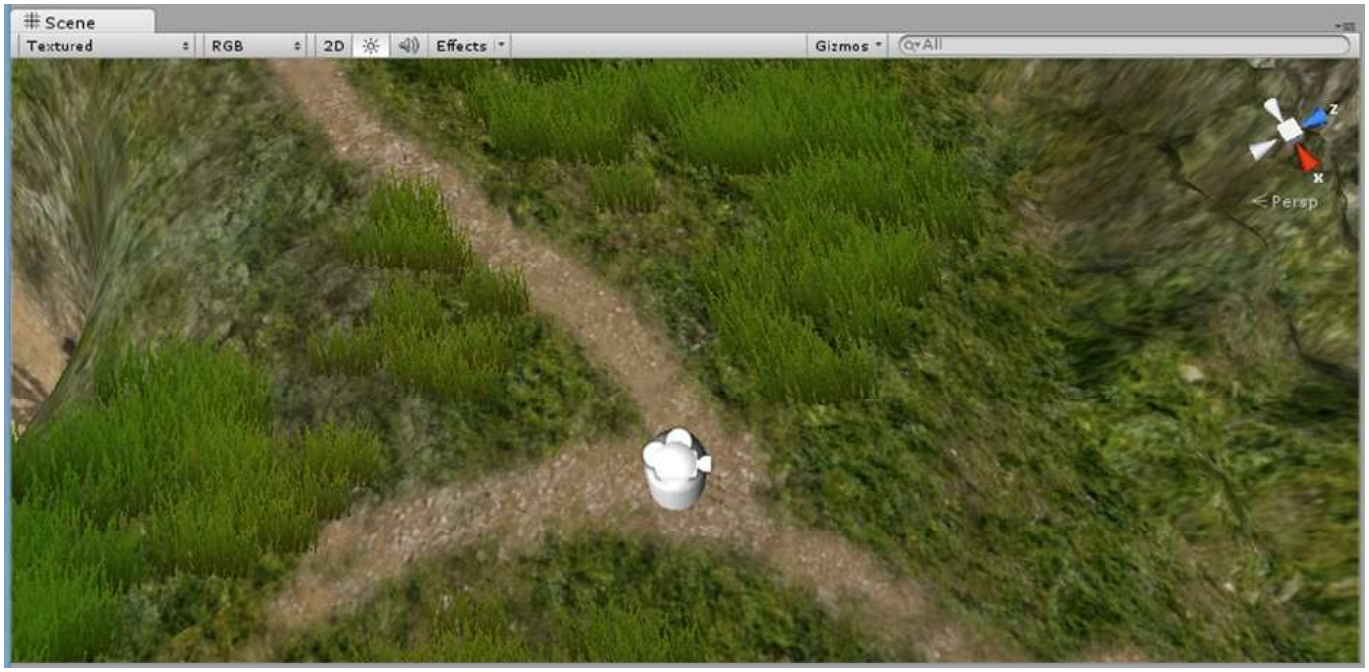

Figure 3-2. *The First Person Controller seen from a near top view*

7. Click Play, and drive the First Person Controller along your path using W for forward, S for backward, and A and D for strafe left and right, respectively.

To turn and look up and down, move the mouse around. Left/right turns the First Person Controller, and up/down movement rotates its camera up and down. Clicking the spacebar makes the First Person Controller jump. When you click Play, the focus is automatically set to the Game window.

If you click outside of the Game view, focus is moved back to the editor. This will allow you to tweak settings during game play and move objects around in the Scene view. Most of these adjustments will be lost when you stop Play mode, but it gives you a handy way to test things. Click back in the Game window.

1. Right-click in the Scene view to activate that view.

2. Select the First Person Controller from the Hierarchy view (to make sure you have selected the parent object).

When you make selections in the Scene view, Unity will first select the parent-most object; additional clicks in the same spot will select its children.

3. Using the left mouse button, grab the transform gizmo in the yellow square that bridges the X and Z arrows, and drag the First Person Controller around the scene without releasing the button.

4. Stop Play mode.

The First Person Controller attempts to stay grounded. If you move it over a pit, it will drop until it hits the ground again. If you move it quickly into a mountain and let go of it, it will start falling. Move it back where it is over the terrain again, and it will once again try to position itself at the terrain height.

Virtual Keys: The InputManager

Now is a good time for a first look at Unity's key mapping system. Key mapping allows the user to select their favorite keys for the various input needed throughout the game. Typically, it is for controlling a character, but it is by no means limited to that. By creating virtual keys, the scripts that make the action happen can remain generic as far as user input goes.

The First Person Controller has several alternate keys that can be used to control it. Let's take a look at the InputManager to see what they are.

1. From the Edit menu, Project Settings, select Input.

2. In the Inspector, click the Axes arrow to expand the list.

You will see 15 preset virtual keys or inputs (Figure 3-3).

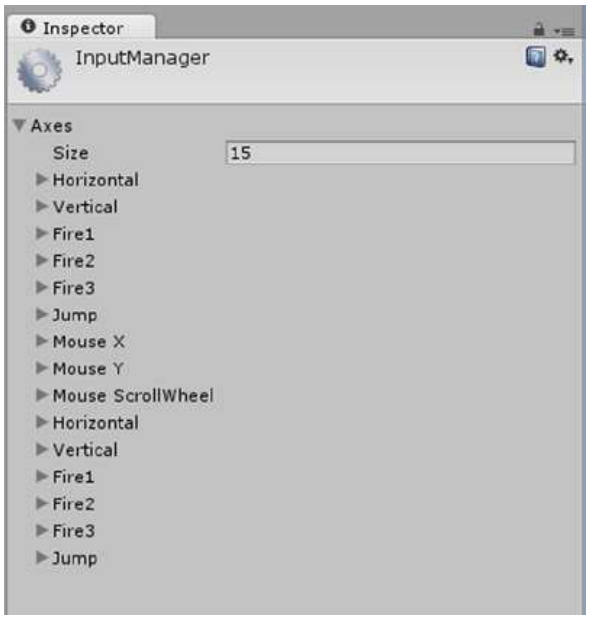

Figure 3-3. The 15 preset inputs in the InputManager

Unity uses “Horizontal” and “Vertical” as the two main directions for character control. They can be confusing unless you understand their origin. Picture a simple 2D game played in a top-down view. The character is moved forward in the up direction of your monitor and backward in the down direction, the vertical directions. Strafing, or sideways movement, is left or right, the horizontal directions. Now picture tipping the monitor down so that you are in a 3D world. “Vertical” movement is forward or backward. “Horizontal” movement is left or right.

3. Click to open the Vertical and Horizontal inputs at the top of the list (Figure 3-4).

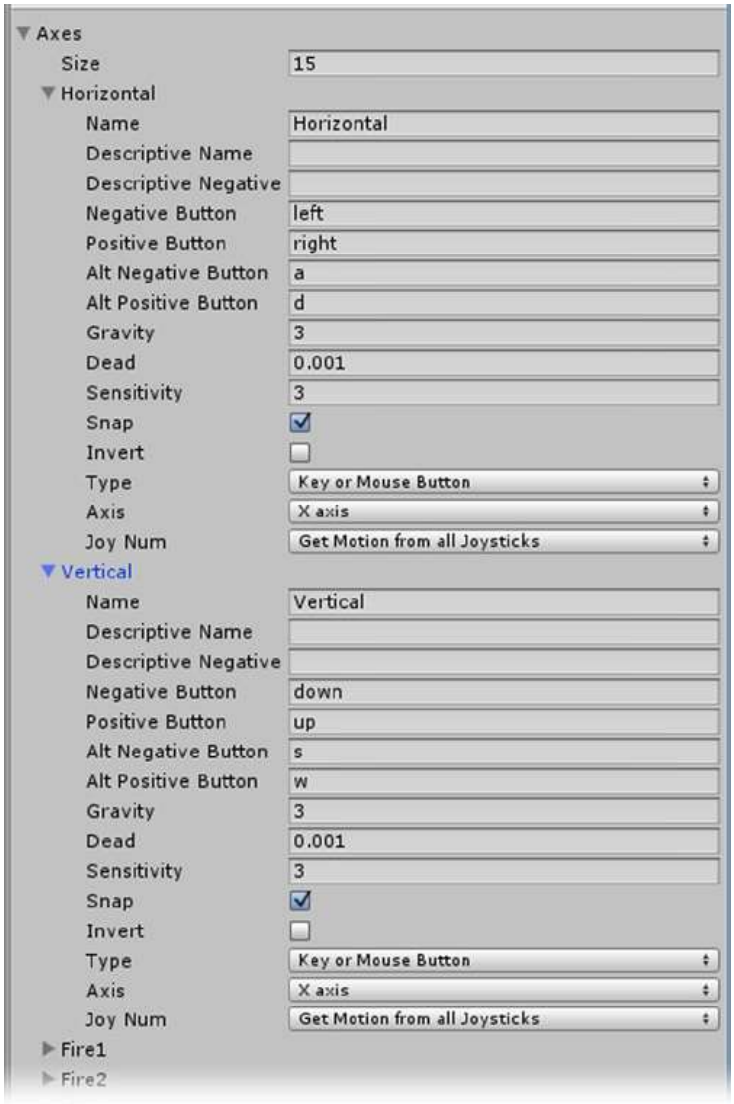

Figure 3-4. *The expanded Horizontal and Vertical inputs*

The Name field determines how the input is accessed through scripting. The Descriptive Name and Descriptive Negative fields will appear in the built game’s configuration dialog at startup where the player can remap the keys. In case you are wondering what *Negative* refers to, take a look at the next four parameters. Instead of using separate inputs for, say, forward and reverse, Unity considers them the same input. Internally, a positive number is generated when the W key is pressed, and a

negative number is generated when the S key is pressed. The second set of parameters begin with Alt, for *alternative*. The first set uses the left and right arrow keys, and the alternates use the W and S keys. Note that all key assignments are typed as lowercase.

Several key names use abbreviations. A search of the Unity Manual for "Input" will explain the Input in depth and also lists the key naming conventions.

The names of keys follow this convention:

- Normal keys: "a", "b", "c" ...
- Number keys: "1", "2", "3", ...
- Arrow keys: "up", "down", "left", "right"
- Keypad keys: "[1]", "[2]", "[3]", "[+]", "[equals]"
- Modifier keys: "right shift", "left shift", "right ctrl", "left ctrl", "right alt", "left alt", "right cmd", "left cmd"
- Mouse Buttons: "mouse 0", "mouse 1", "mouse 2", ...
- Joystick Buttons (from any joystick): "joystick button 0", "joystick button 1", "joystick button 2", ...
- Joystick Buttons (from a specific joystick): "joystick 1 button 0", "joystick 1 button 1", "joystick 2 button 0", ...
- Special keys: "backspace", "tab", "return", "escape", "space", "delete", "enter", "insert", "home", "end", "page up", "page down"
- Function keys: "f1", "f2", "f3", ...

To get some firsthand experience with Unity's "virtual keys," you will be taking the First Person Controller for a spin (or a jump, or a run...).

1. Click Play, and test the alternate forward key, the up arrow.
2. While in the Game view, press the spacebar to see the First Person Controller jump.
3. Stop Play mode.
4. Close the Horizontal and Vertical inputs, and expand the first Jump input.
5. Change its Positive Button setting from *space* to *escape* (Figure 3-5).

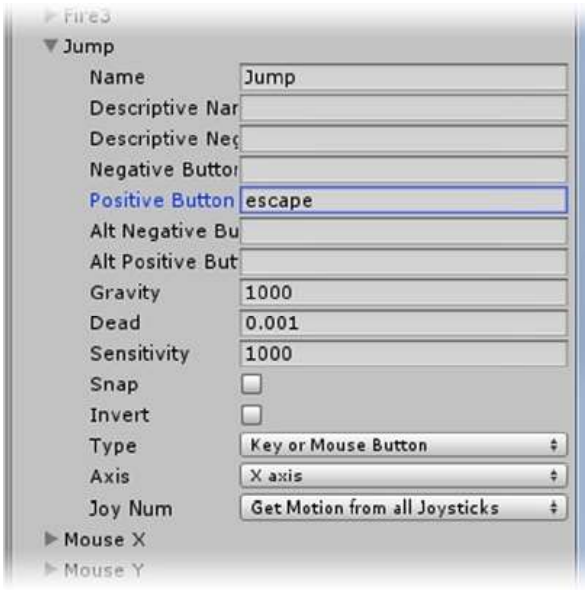

Figure 3-5. *The Jump input's Positive Button mapped to the escape key*

6. Click Play, and test the new jump mapping by pressing the escape key.
7. Stop Play mode, and change the Positive Button setting back to *space*.

While you were in the InputManager, you probably noticed a set of duplicate inputs for several of the keys. Two alternatives are built into the input template. If you require more alternatives for the same input, you can simply create another of the same name. The current preset duplicates add input for joysticks. To create your own input definition, you have to increase the Size value of the Axes list.

1. At the top of the InputManager in the Inspector, change Size to **16**.

A new input is added by copying the last one in the list.

> **Tip** You can also duplicate specific array elements by right-clicking on the element parent and selecting Duplicate Array Element. This gives you a quick way to add multiple input options or to set up new inputs by starting with something close to your desired setup.

2. Expand it, and change its Name field to *Vertical*.
3. Set its Negative Button to *k* and its Positive Button to *j*.
4. Click Play, and test the new alternative "Vertical" input keys.
5. Test the original W and S keys and the up and down arrows to assure yourself they still work.
6. Stop Play mode.
7. In the InputManager, set the Axes array Size back to **15** to remove your extra Vertical input.

Components

Now that you've had a chance to experiment with the First Person Controller's basic functionality, it's time to inspect the First Person Controller itself.

1. Select the First Person Controller in the Hierarchy view.
2. Click its down arrow to expand its hierarchy (Figure 3-6).

Figure 3-6. The expanded First Person Controller

You will find it has two children, Graphics and Main Camera.

3. Select Graphics.

It has only two components besides the requisite Transform component. The Mesh Filter holds an internal mesh that you do not have access to, though for all intents and purposes, it is a capsule. In the Mesh Renderer component, it is worth noting that Cast Shadows and Receive Shadows are both turned off. Shadow calculation is expensive, and this object is more of a visual placeholder, so you neither want to waste resources nor have it cast shadows into the scene.

4. Select Main Camera.

Besides the components you have already seen on the original scene camera, this has a Mouse Look script. If you are starting to catch on to the components idea, you may have wondered if one couldn't just add a Camera component to the First Person Controller directly. Let's try it and see what happens.

5. First, deactivate the Main Camera gameObject by unchecking the check box next to its name at the top of the Inspector.

With no camera in the scene, the Game view goes blank.

6. Select the First Person Controller.
7. From the Component menu, Rendering, add Camera.

The Game view is once again rendered.

8. Click Play, and drive around the scene.

The first problem is that the component functions from near the First Person Controller's bellybutton, the location of its transform pivot point (Figure 3-7). The other problem is that you've lost the ability to look up and down. If you inspect the First Person Controller's Mouse Look component, you will see that its Axes parameter is set to use Mouse X. This time, it is referring to the mouse movement. Traditionally (think graph paper), the X axis is on the horizontal and the Y axis is on the vertical. With the mouse, this means X or horizontal mouse movement. In the script, it controls rotation on the First Person Controller's Y axis, or look left/right. It turns out that the Mouse Look component *does* have the option to use both axes, but other components on the First Person Controller will keep it from tipping up or down. The bottom line here is that although you can combine specialty components, there are often good reasons to put them on separate gameObjects.

Figure 3-7. *The camera component on the First Person Controller gameObject*

9. Right-click on the First Person Controller's Camera component, and select Remove Component (Figure 3-8).

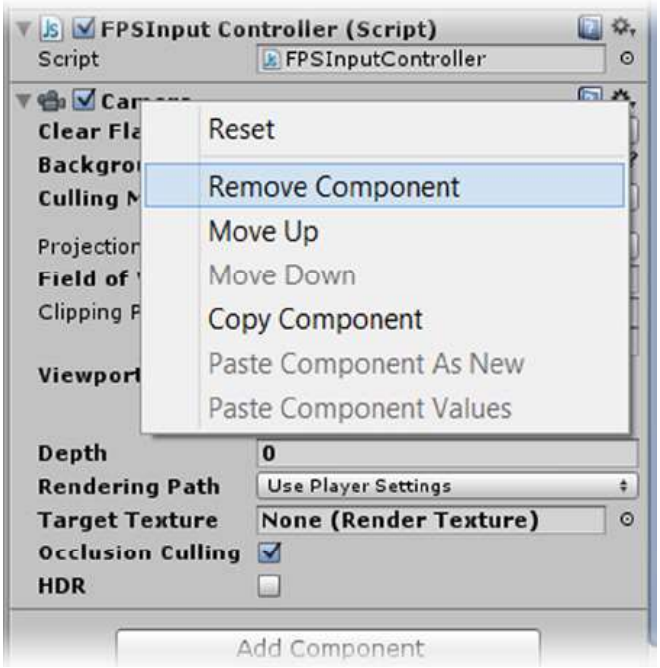

Figure 3-8. The component right-click menu, selecting Remove Component

10. Select the Main Camera, and activate it by clicking the check box next to its name at the top of the Inspector.

The Game view returns to its previous view. "Activate" is the term used to control a gameObjects's state and "Enable" is the term used for components. Later, when you manipulate those parameters from scripts, those terms will be used.

Let's take a look at the rest of the First Person Controller's components.

The Character Controller Component

The Character Controller is a specialty component that serves, more or less, to combine the functionality of the collider and Rigidbody components. A rigidbody component is required for using physics to drive interactions in the scene, but it is too expensive to use for the First Person Controller, so Unity adds the most important functionality to it through the Character Controller. This, not physics gravity, is what causes the First Person Controller to fall when you drag it or drive it off of cliffs. It has a few parameters that can be adjusted to help or hinder your First Person Controller as it moves around the scene.

1. Select the First Person Controller.
2. Click Play, and try to drive the First Person Controller into a deep pit. Then try to get out or go up the side of a steep slope.

At some point, you can go no higher without resorting to jumping as you go up at an angle, and even that may not be enough.

3. In the Character Controller component, set the Slope Limit to **90** degrees.
4. Attempt the steep slope again.

This time you should be able to get out of the pit or up to the top of the most unlikely mountain peak (Figure 3-9).

Figure 3-9. *A deep, steep-sided pit, no longer a trap for the First Person Controller*

5. Set the Slope Limit back to **45** degrees.
6. Stop Play mode.

Step Offset determines how high an object is when the slope is steeper than the limit. This will allow the First Person Controller to go up steps without being blocked by the Slope Limit.

1. Focus in on the First Person Controller in the Scene view.
2. From the GameObject menu, Create Other, create a Cube.
3. In its Transform component, set its Scale parameters to **3** x **0.4** x **3** to create a nice platform.
4. Position it somewhere in front of the First Person Controller, slightly intersecting the ground on the approach side.

5. With Local coordinates set, rotate the First Person Controller on the Y axis so that it faces the cube.

6. Click Play, and drive the First Person Controller up onto the platform (Figure 3-10).

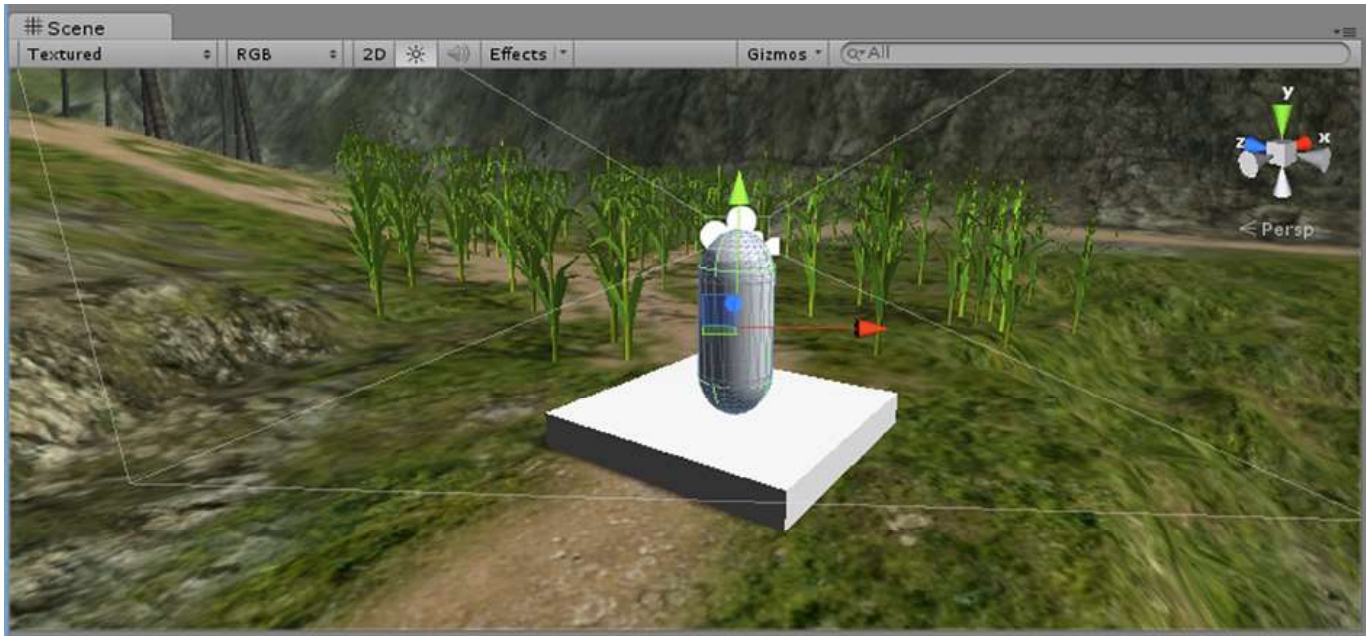

Figure 3-10. *The First Person Controller is able to drive up onto a low platform*

7. Increase the cube's height to **0.6**, and lift it so it is no longer intersecting with the ground on the side that the First Person Controller will approach it.

8. Now try to drive the First Person Controller up onto it again.

This time the First Person Controller cannot drive up onto the platform. An inspection of its Step Offset parameter shows that it will not be able to "glide" up anything higher than 0.4 meters (Unity's default units) in front of it.

9. Select the First Person Controller, and change its Step Offset to **1.0**.

Once again, the First Person Controller can glide over the platform with ease.

10. Stop Play mode.

The cube returns to its original height, and the Step Offset is returned to its original amount.

The Slope Limit and Step Offset will go a long way toward establishing the difficulty and feel of the scene navigation in a first-person game, so set them up accordingly.

The Character Motor Component

The Character Motor is the script that controls the rest of the First Person Controller's functionality.

1. Select the First Person Controller again.
2. In the Character Motor script, open the Movement, Jumping, Moving Platform, and Sliding sections (Figure 3-11).

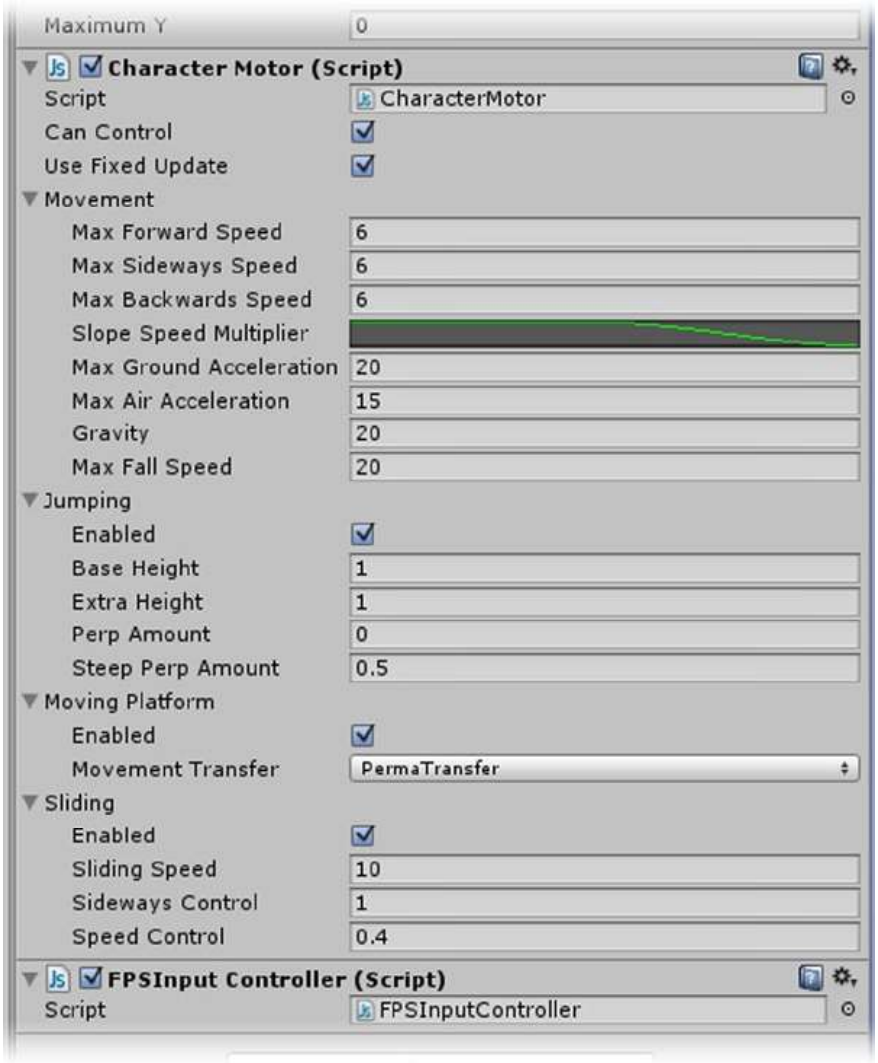

Figure 3-11. The Character Motor component, expanded

As you can see, besides the various parameters, Jumping, Moving Platform, and Sliding can be turned off individually. All of these parameters, by the way, could be exposed to the player with a GUI and some scripting. Let's test a few of them in Play mode so you won't have to remember their original values.

3. Click Play.
4. Set the Max Forward Speed to **15,** and click in the Game view to change the focus back to it.

When you first click on Play, the application focus is changed to the game window, so your input (mouse movement, keyboard entries, etc.) is applied to that view. As soon as you click *outside* of the Game view, your input no longer applies to the running game. This allows you to make adjustments in the Scene view, the Hierarchy view, and the Inspector during run-time. To "get back" to the game, you must return focus to it by clicking in the view before your input will be applied to the game. If you click outside of the Unity editor, the running game is paused.

5. Drive around the terrain.

The First Person Controller zips speedily around the environment, but it slides badly around turns. To adjust for the extra speed, you could increase the Mouse Look component's Sensitivity X if you wish.

Let's play with the Gravity parameter next. This will not affect the scene gravity, as it is controlled by the Physics settings.

6. Set the Max Forward Speed back to about **8**.
7. Set Gravity to **1**.
8. Drive the First Person Controller off the side of a pit or mountain peak, and enjoy the long scenic trip to the ground.

Jumping offers some interesting possibilities.

9. Set the Base Height to **10**, and try a few jumps (Figure 3-12).

Figure 3-12. View from the tree tops

For aggressive players, you could increase the Extra Height setting. This will add more height if the player holds down the spacebar. On the off chance your character will have a jet pack, you are out of luck with this script. It doesn't allow the player to jump again until after the First Person Controller has landed.

To test, you will import a little script to set the platform to move back and forth. This script moves the object you put it on in the z direction.

1. Right-click in the Project view, and from the right-click menu, choose Create, Folder, or click the Create down arrow at the top of the Project view and select Folder from there.
2. Name it **Test Scripts**.
3. Select it and from the right-click menu, and then select Import New Asset.
4. Locate the Hz_PositionCycler.cs script in the Chapter 3 Assets folder, and import it into your scene.
5. Drag the new script from the Test Scripts folder in the Project view to the Cube object in the Hierarchy view.
6. Click Play.

The platform moves slowly back and forth in the global z direction.

7. Select the Cube, and inspect the new component's parameters.
8. Try adjusting the Max Range to **8**.
9. Back in the Game view, drive the First Person Controller up onto the platform when it comes within range.

The First Person Controller goes for a ride on the platform.

10. Drive the First Person Controller off of the platform.

The First Person Controller can move freely around the platform because the Character Motor is adjusting its position relative to the platform. If you move the First Person Controller into the path of the platform, you will probably find that platform goes through it. If the platform is low enough, it may push the First Person Controller away.

11. Stop Play mode.
12. If the platform goes through the First Person Controller, move it lower into the ground; if it pushes the First Person Controller away, move it up higher.
13. Click Play and see if your previous results have changed.

The other possible outcome is that the platform bumps the First Person Controller and picks it up. Either way, the results are not dependable enough to use without being able to control the mounting circumstances. If you disable the Moving Platform in the Character Motor, you will see different results.

14. Select the First Person Controller.
15. Uncheck Enabled in the Moving Platform.
16. Drive the First Person Controller into the path of the platform.

The First Person Controller will be bumped up when the platform is low enough, but its position is no longer matched. There are no more free rides for the First Person Controller.

17. Select the First Person Controller.

The Sliding parameter may have suffered from version changes. Sliding, where the First Person Controller slides back down steep slopes, shows no change when enabled or disabled but is affected more by the Slope Limit. Unity has introduced a new real character controller from the new Unity Sample Assets (https://www.assetstore.unity3d.com/#/content/14474) which, unlike the old Character Controller, uses a Rigidbody to properly interact with the rest of the world. It can be downloaded from the Asset Store, but it has not yet been incorporated into the Standard Assets that ship with Unity. Feel free to give it a try.

Colliders

You've now had some firsthand experience with colliders. The platform, because it is a Cube, comes with a Collider component. Before adding the animation to the cube, the First Person Controller either was stopped by it or moved up on top of it. Once objects are animating, the rules change, but for static (or nonmoving) objects, a collider component will act as a wall, floor, or combination of both, depending on where the intersection takes place.

1. Stop Play mode.
2. Select the Cube object, and set its Y Scale to **5**.
3. Disable its Hz_Position Cycler component by unchecking the box next to the component's name.
4. Click Play, and try to drive into the side of the Cube.
5. The collider stops it.
6. Disable the Cube's Box Collider.
7. Try driving into the side of the Cube again.

With no collider, the First Person Controller goes right through the Cube. Next you will test the Is Trigger parameter.

1. Enable the Cube's Box Collider.
2. Turn on the collider component's Is Trigger parameter.
3. Drive into the side of the Cube again.

This time the Collider is on, but it no longer blocks the First Person Controller. It does, however, register the intersection, as you will find when you start scripting.

There's one other thing you can discover about colliders as long as you are in the terrain environment.

1. Drive the First Person Controller around the terrain, and drive at a few of the Palm trees.
2. Repeat the experiment with a BigTree.

The First Person Controller goes right through both types of trees. With a small-trunked tree, you might decide that stopping the First Person Controller would interfere with the game flow. With something as substantial as the BigTree, being able to drive through is not something that can be ignored.

To remedy the problem, you will start by examining the original prefab.

3. With the Project view currently in Single Column Layout, type **BigTree** in the search field at the top of the panel.
4. To quickly identify the correct asset, you can select the Prefab filter from "Search by Type," the first icon to the right of the search field.

You can also slide the thumbnail slider at the bottom of the view to the far right to see the asset icons. At the far left, you can recognize prefabs by their blue cube icons.

5. Once the BigTree prefab is selected, take a look at its components in the Inspector.
6. Collapse the Tree component so you can see the components more easily (Figure 3-13).

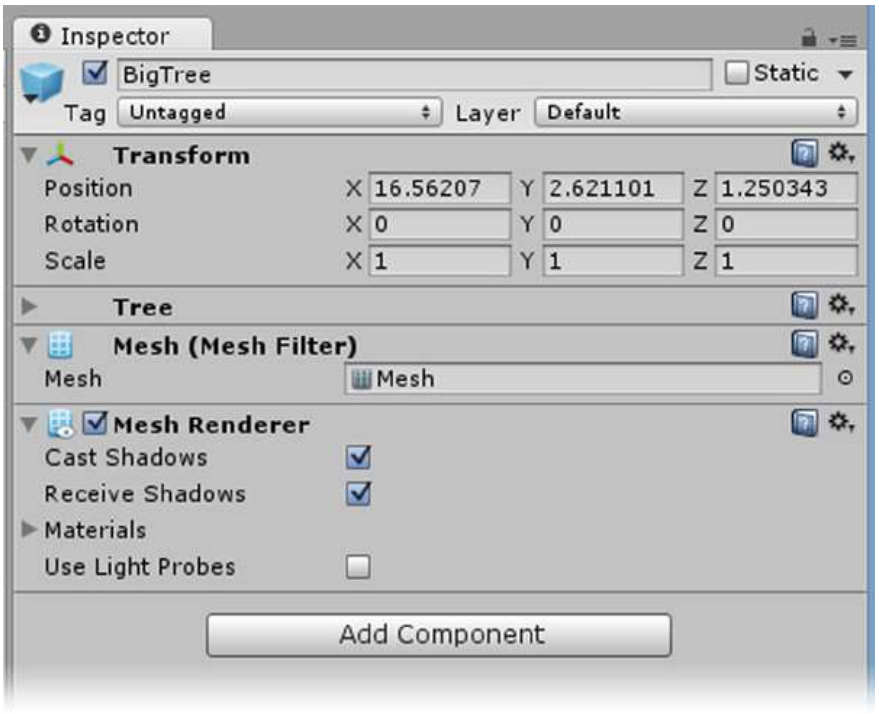

Figure 3-13. *The BigTree's components in the Inspector*

As you can see, there is no collider component of any type. When Colliders are added to gameObjects, they are scaled according to the object's bounding box. For a tree, you will want to adjust the collider's size to fit the trunk. The easiest way to do the adjustment is to do it in the Scene view. Let's begin by adding a collider component. Besides the Component menu, you can add components using the Add Component button at the bottom of the current components. Unlike you did in its menu counterpart, you will have to click to open the submenus.

7. Drag the BigTree prefab into the scene.
8. Click the Add Component button just below the Mesh Renderer component.
9. Click to open the Physics submenu, and select Capsule collider.

The Capsule collider is scaled to fit around the entire tree (Figure 3-14).

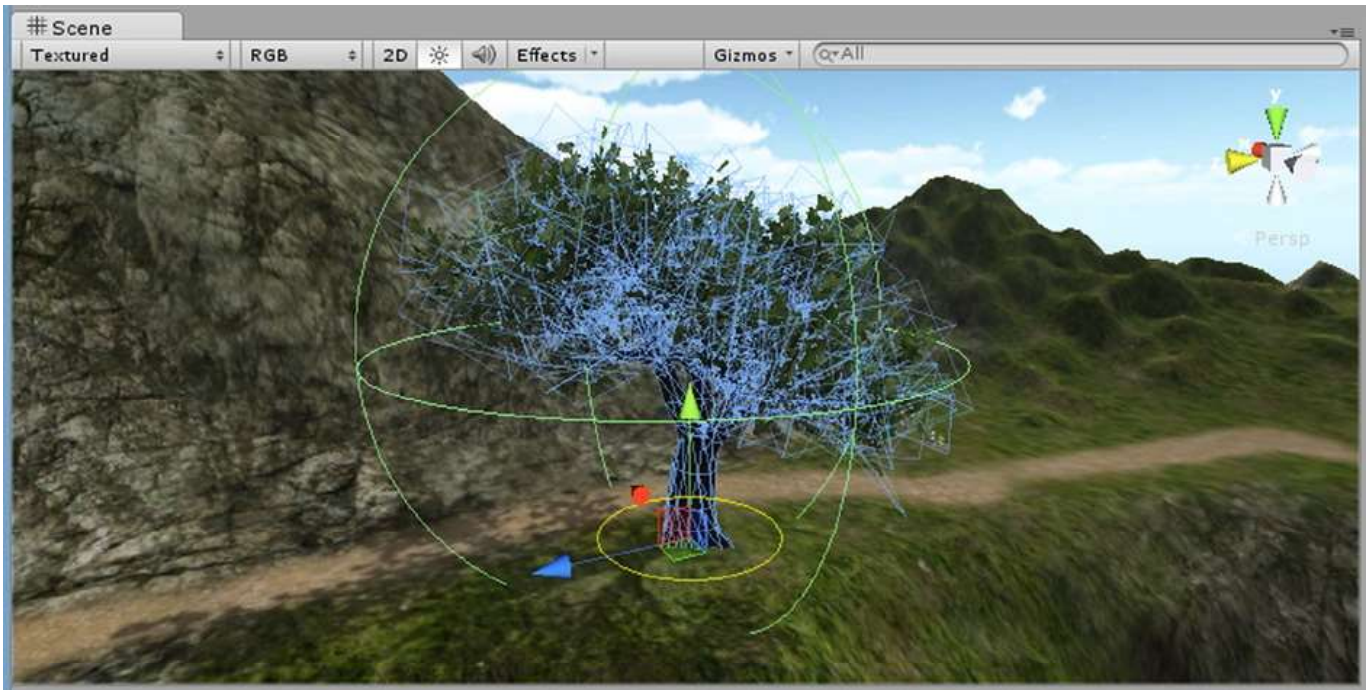

Figure 3-14. The capsule collider added to the BigTree prefab

10. Set the Capsule Collider's Radius value to **0.7**.

11. Adjust the Height and Center values until the collider fits the tree trunk (Figure 3-15).

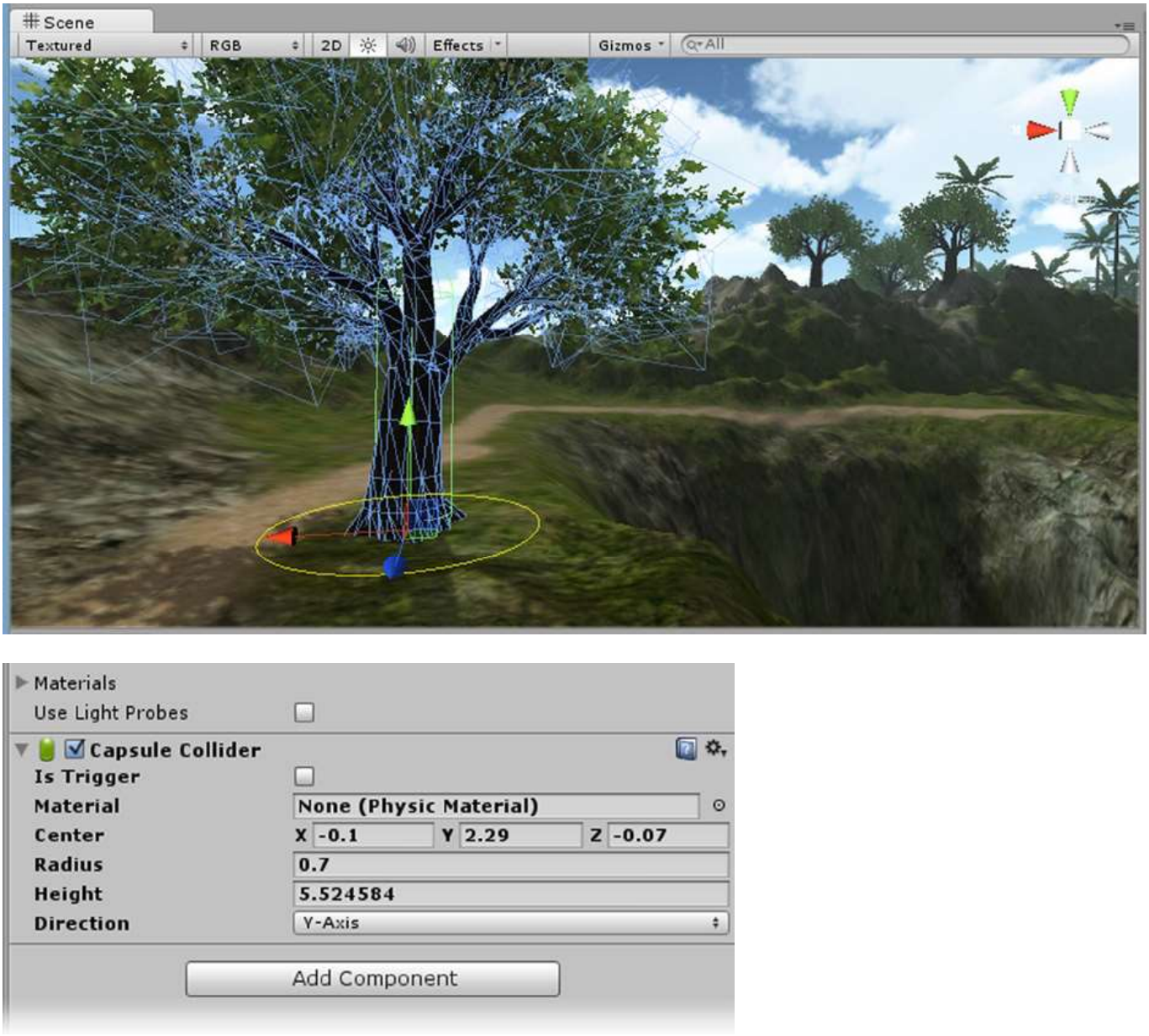

Figure 3-15. *The capsule fit to the tree trunk (top), and the collider's values (bottom)*

12. Click Play, and drive the First Person Controller into the tree.

13. Stop Play mode.

Now that you have made sure the tree will stop the First Person Controller, you have to update the prefab to include the new component before refreshing the terrain's trees.

1. With the BigTree selected, at the top of the Inspector, click the Apply button at the right of the Prefab line (Figure 3-16).

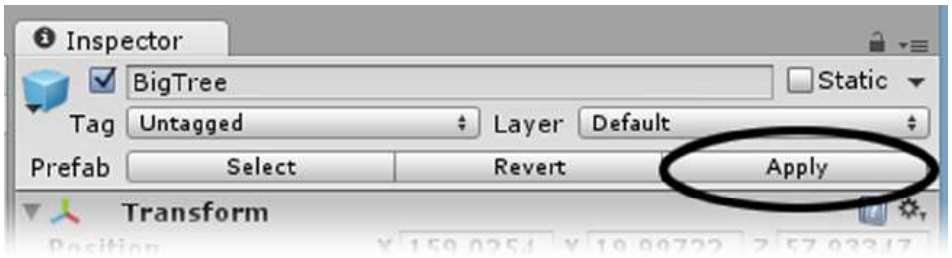

Figure 3-16. Updating the Prefab setting

Once you have updated the prefab, it is no longer required in the scene.

2. Delete the BigTree prefab from the scene.
3. Select the Terrain object in the Hierarchy view.
4. In the Paint Trees section, click the Refresh button to the right of the Edit Trees button.
5. Click Play, and test the rest of the BigTrees by trying to drive through them.
6. Stop Play mode.

Physics

As computers have gotten faster, it has become feasible to use physics to control a lot of game action. Accurate algorithms are still too slow for real-time, however, so game physics are still only rough approximations. Because most game physics are used for mayhem and destruction, this is perfectly acceptable.

The big gain by applying physics to animation projectiles, collisions, and other actions has been to significantly reduce the need for traditional key-frame animation. It also has a side effect of automatically adding a good measure of randomness to actions. If you do need a specific result, you will be better off using a pre-made animation, though even that could be inserted into a string of physics-driven actions to preserve the illusion of randomness.

Game physics are not a silver bullet (pun intended). Physics-driven objects will not yet perform true to life. As far as frame rate goes, it continues to be too costly to do so. The most typical example of this is shooting a projectile at a wall. If the frame rate is too slow, the projectile could be in front of the wall at one frame when it is checked and beyond the wall in the next frame. Since an intersection was never detected, no reaction ever occurs. Physics intersections may be checked more often than every frame, but the concept remains the same.

Rigidbody

The two most important ingredients for using in-game physics are the collider and Rigidbody components. The collider defines the physical boundaries of the object, and the Rigidbody component handles the physics that do the work. Scenes already have default “gravity,” but you can also add forces and torque to physically move or rotate objects. To make things more interesting,

you can even add Physic Materials to further define the way an object reacts to the environment. Unity also provides a nice assortment of "joints" to help you combine multiple objects involved in physics-based encounters.

Let's start the investigation with a simple Cube. You will be working and observing in the Scene view, so feel free to deactivate the First Person Controller in the Hierarchy view for now and increase the Scene view size in the editor (Figure 3-17).

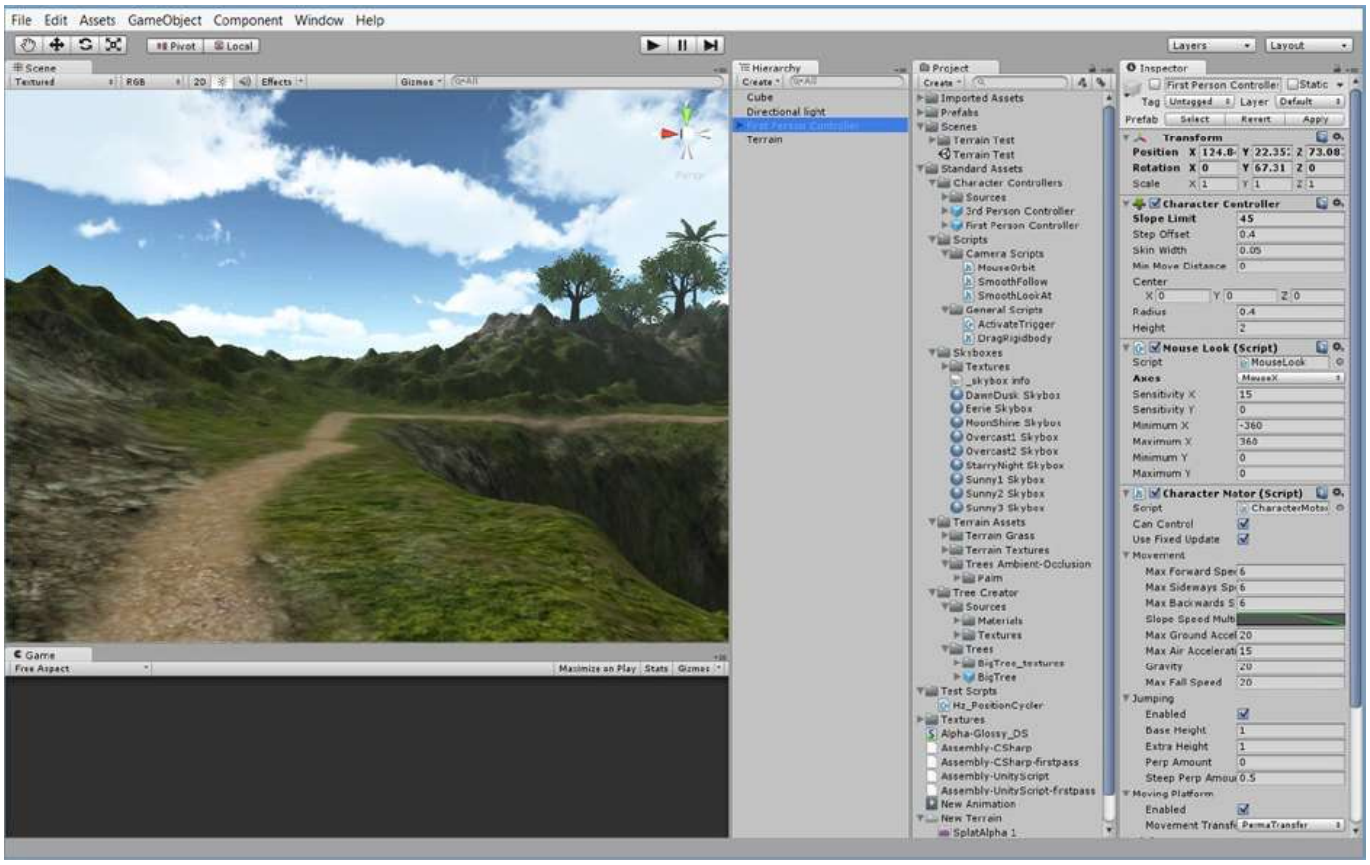

Figure 3-17. Rearranging the UI for the Physics tests

1. Select the original Cube in the Hierarchy view, and rename it **Cube Platform**.
2. Create a new Cube, and position it a couple of meters off of the ground (Figure 3-18).

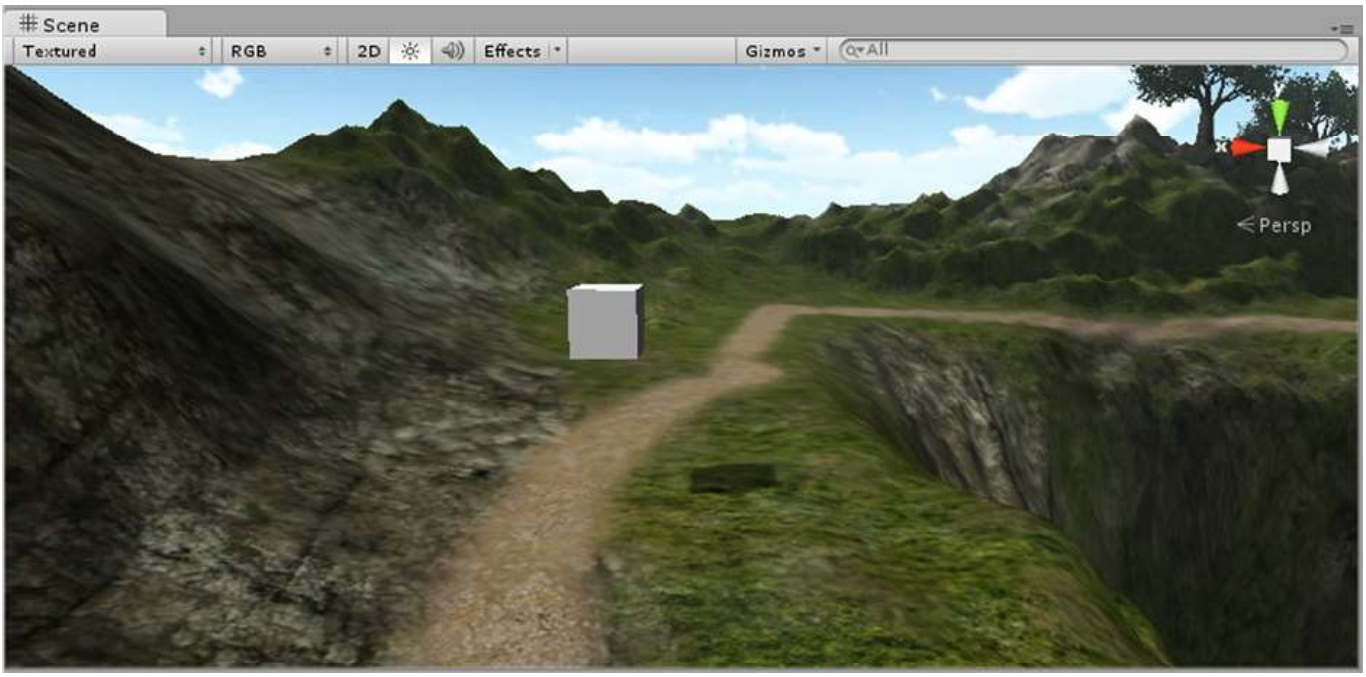

Figure 3-18. The new Cube in mid-air

3. Click Play.

The Cube does nothing.

4. Stop Play Mode.
5. From the Component menu, Physics, add a Rigidbody component (Figure 3-19).

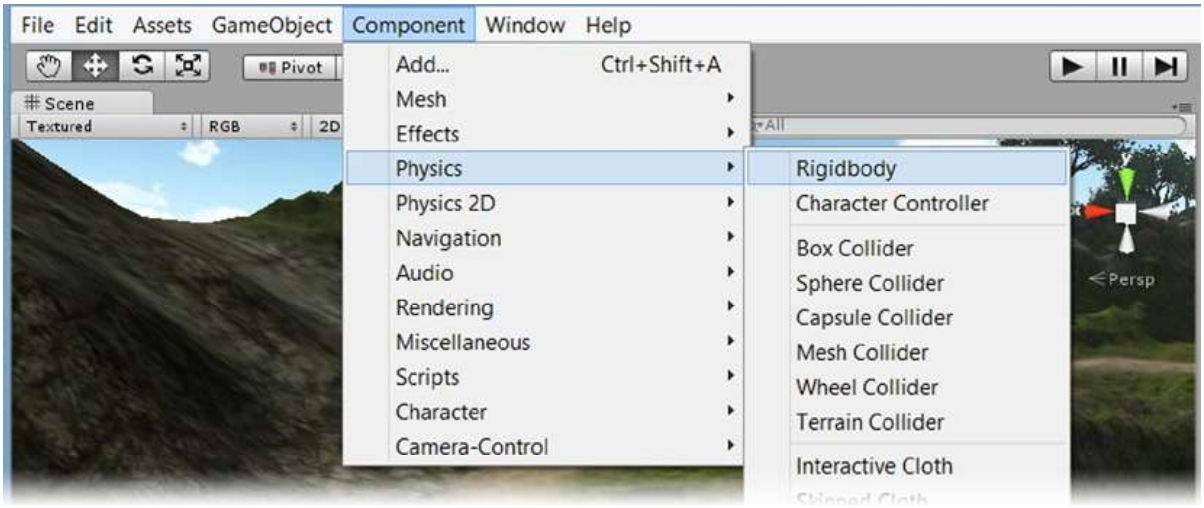

Figure 3-19. Adding a Rigidbody component

6. Click Play.

This time the Cube drops to the ground. If the ground is not level, it may even move a bit after it lands.

7. Drag the cube around, dropping it occasionally, to see how it reacts with the terrain.
8. Stop Play mode.
9. Use Ctrl+D to duplicate the first cube a couple of times, and then create a three-cube stack, leaving a little space between each (Figure 3-20).

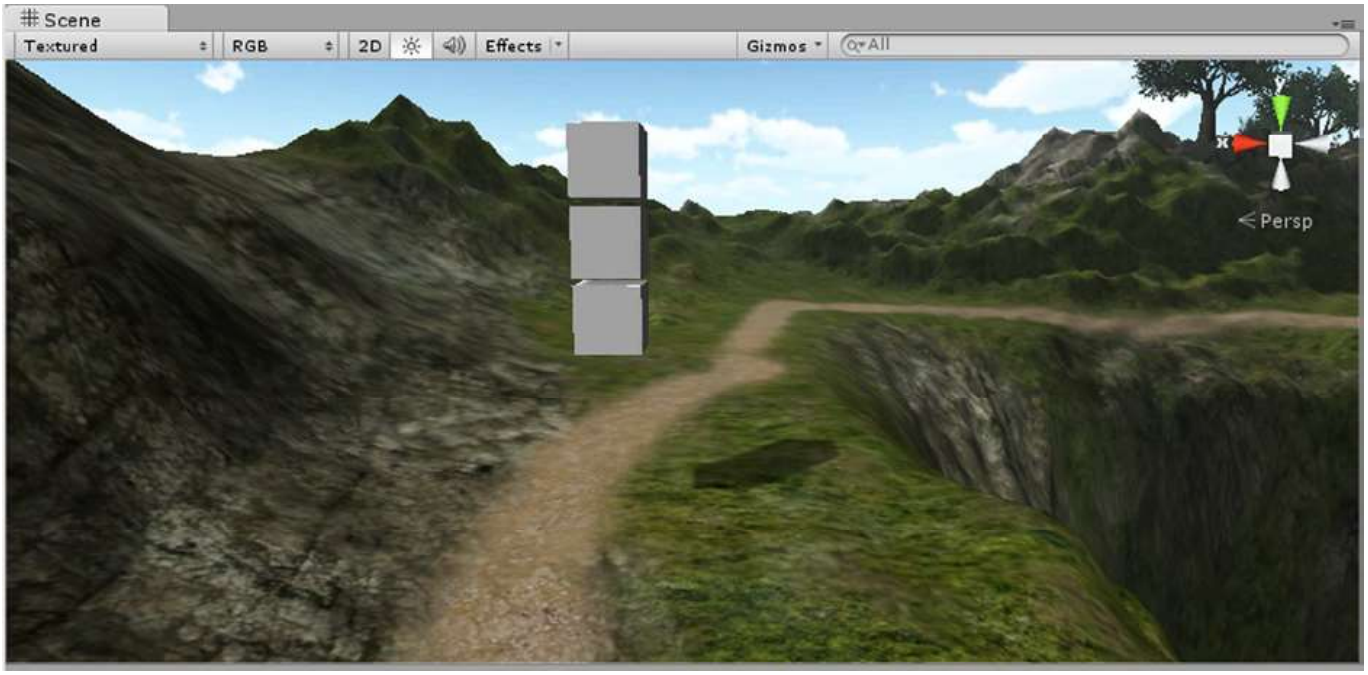

Figure 3-20. *The three-cube stack*

10. Click Play.

If the terrain is fairly level, they drop and stay stacked after a bit of shifting around. If you offset them, they will tumble as they fall, making the action more interesting.

1. Stop Play mode.
2. Move the middle cube half way out from its current position, and rotate it on the Y axis (Figure 3-21, left).
3. Click Play.

This time the cubes drop and tumble (Figure 3-21, right).

Figure 3-21. *The offset cube stack (left), and the results of the adjustment in Play mode (right)*

The results are more interesting, but there's something missing. They react as if they were made of concrete.

Physic Materials

The Rigidbody component controls a good part of the object's physical interaction with the scene, but not all. To have the objects interact as if they were made of different materials, you will have to assign a Physic Material. This property is assigned through the collider components.

1. Stop Play mode.
2. Right-click in the Project view, and from the Create submenu, select Physic Material.

A new Physic Material is created (Figure 3-22). It has a number of parameters that are not necessarily meaningful unless you've had a Physics class. You could, of course, do a lot of testing, but there's another option. There is a Unity package with several preset materials that you can use as is or tweak to refine.

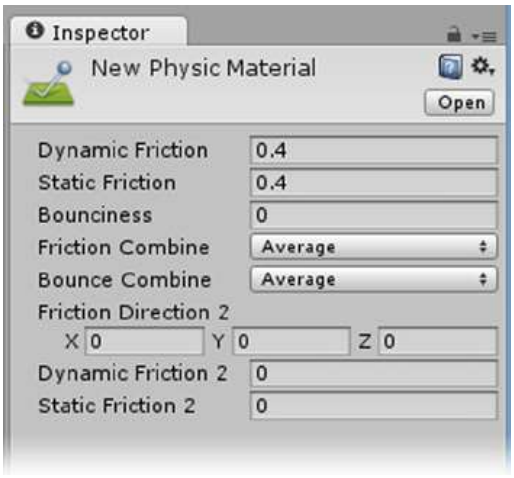

Figure 3-22. *The new Physic material*

3. Once again, open the right-click menu in the Project view, or open the Assets menu and select Import Package.
4. Choose Physic Materials.
5. In the Import dialog, note the folder location, note the Physic Materials folder in the Standard Assets folder, and click Import.
6. Open the folder, and click on each Physic material to see its settings in the Inspector.

Fortunately, the materials have been given nice descriptive names for those of us to whom physics remain a mystery.

7. Drag and drop a different material directly on each cube in the Hierarchy view. Put Bouncy on the top cube, Wood on the middle cube, and Ice on the bottom cube.
8. Select each object, and examine its collider's Material field.

The materials are added to the proper parameter automatically.

9. Click Play.

This time the cubes react according to their individual physic materials, Bouncy bounces quite nicely. Ice, having very little Static Friction, slides off as it lands, and Wood bounces lightly before settling.

This is a good time to experiment with the Rigidbody a bit more.

1. Stop Play mode.
2. Select the cube with Bouncy.
3. Set its Mass parameter to **0.1**.
4. Click Play.

With less mass (weight/volume), you get a lot more bounce with the reaction.

5. Stop Play mode.
6. For the middle cube, turn off Use Gravity.
7. Click Play.

Having been clipped by the top cube, the Bouncy cube tumbles slowly downward, floating upward when it collides with the bottom cube.

The Is Kinematic Parameter

There are a few different ways to freeze an object with a Rigidbody component so it won't be affected by physics in the scene or will have limited freedom to react. The first method is using the Is Kinematic flag. This is recommended when you have objects that are not being animated by physics. In your test scene, the moving platform would qualify because a script is setting its location dynamically during runtime. Objects with key-frame animation are another use case. If the animated object has a collider and can interact with other dynamic objects in the scene, collision-detection calculations are far more efficient with a Rigidbody component set to Is Kinematic.

To develop good working habits, you can go ahead and add a Rigidbody component to the Cube Platform.

1. Select the Cube Platform, set its Y Scale back to **0.25,** and adjust its Y location so the First Person Controller will be able to get onto it again.
2. Enable its Hz_PositionCycler script component and its Box Collider.
3. Add a Rigidbody component, and turn its Is Kinematic property on.

You could activate the First Person Controller and test the platform after enabling the Moving Platform property in its Character Motor component, but you will see no difference in functionality. Instead, let's look at another means of restricting physics reactions.

1. Select the middle cube, and turn its Use Gravity parameter back on.
2. Select the bottom cube.
3. Check Is Kinematic, and click Play.

The bottom cube remains in place as the other two cubes interact with it. Because it is not doing anything, it would probably be better to remove the Rigidbody entirely. In this case, however, there are more experiments to run.

4. Stop Play mode.
5. Uncheck its Is Kinematic property.
6. Under Constraints, check the X,Y, and Z Freeze Position parameters.
7. Click Play.

This time the cube retains its position but spins wildly when the other cubes collide with it. You will see the Rigidbody constraints used in 3D-platform, jumper-type games where the objects may only move in the Y or Z directions and rotate on the X axis. This serves to keep them on the ramps and platforms without using invisible collider walls.

Forces

A less passive means of affecting objects with Rigidbody components is by using a physics Force. Through scripting, you will be able to add a one-time force for objects such as projectiles. The component force, Constant Force, is constant or, more accurately, continuous.

1. Select the middle cube.
2. From the Component menu, Physics, select Constant Force.
3. Give it a Z force of **10** (or **-10** if you wish) to send it off over the hills.
4. Zoom out a bit in the Scene view.
5. Click Play.

The cube goes rolling off over the terrain until it falls in a hole or goes off the edge of the terrain. Adding a local or Relative force to the bouncy cube will send it off.

1. Stop Play mode.
2. Add a Constant Force component to the top cube.
3. Set its Relative Force, Y to **-2**.
4. Click Play.

The initial downward force causes the cube to bounce more violently, causing it to spin. The force, being local, causes the cube to go off in unexpected directions.

While Force is a linear motion, Torque causes rotation.

1. Stop Play mode, and select the bottom cube.
2. In the Rigidbody Constraints, activate its X and Z Freeze Rotation parameters.
3. Add a Constant Force component to it.
4. Set its Torque, Y to **1**.
5. Click Play.

This time the cube is rotating before the other two interact with it.

6. Stop Play mode.
7. At the top of the Inspector, deactivate the top and middle cubes.
8. Click Play.

The remaining cube happily ramps up and then spins merrily around on its own.

9. Stop Play mode.

Joints

As mentioned earlier, Unity has a several Joints to allow you to combine physics objects (gameObjects with Rigidbody components). You will be doing a few tests with the Hinge Joint, the most familiar of the joint types.

1. Select the active cube, and focus the viewport to it.
2. Set its Y Torque to **30**.
3. Create a new Cube, and name it **Lid**.
4. Scale its X to 0.01, and move the Lid to the side of the Cube.
5. From the Component menu, Physics, select Hinge Joint (Figure 3-23).

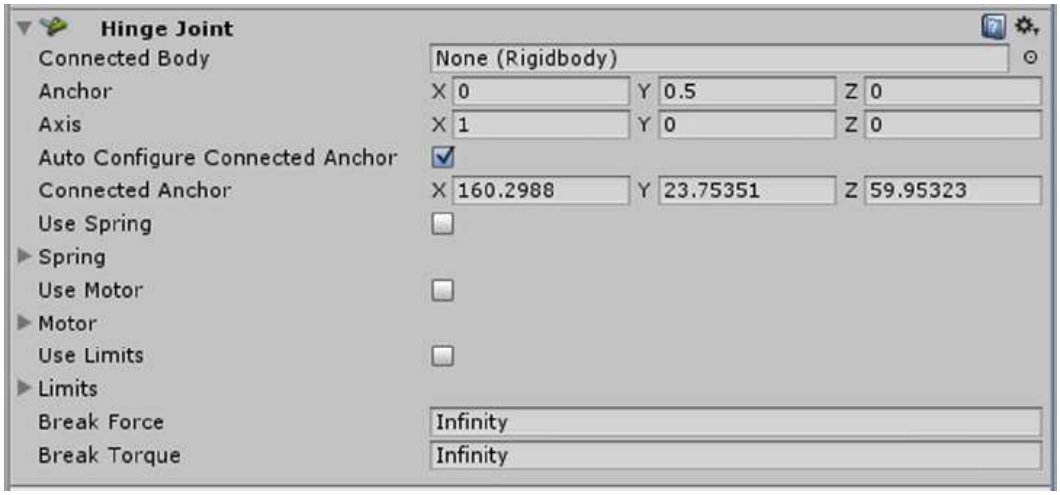

***Figure 3-23.** The Hinge Joint component*

When the Hinge Joint is added to an object that does not already have one, a Rigidbody component will be added automatically.

As a default, the Hinge Joint's axis of rotation is set to X. You can see the small axis at the top of the Lid, pointing across (or away from, depending on the side you chose) the top face of the cube (Figure 3-24).

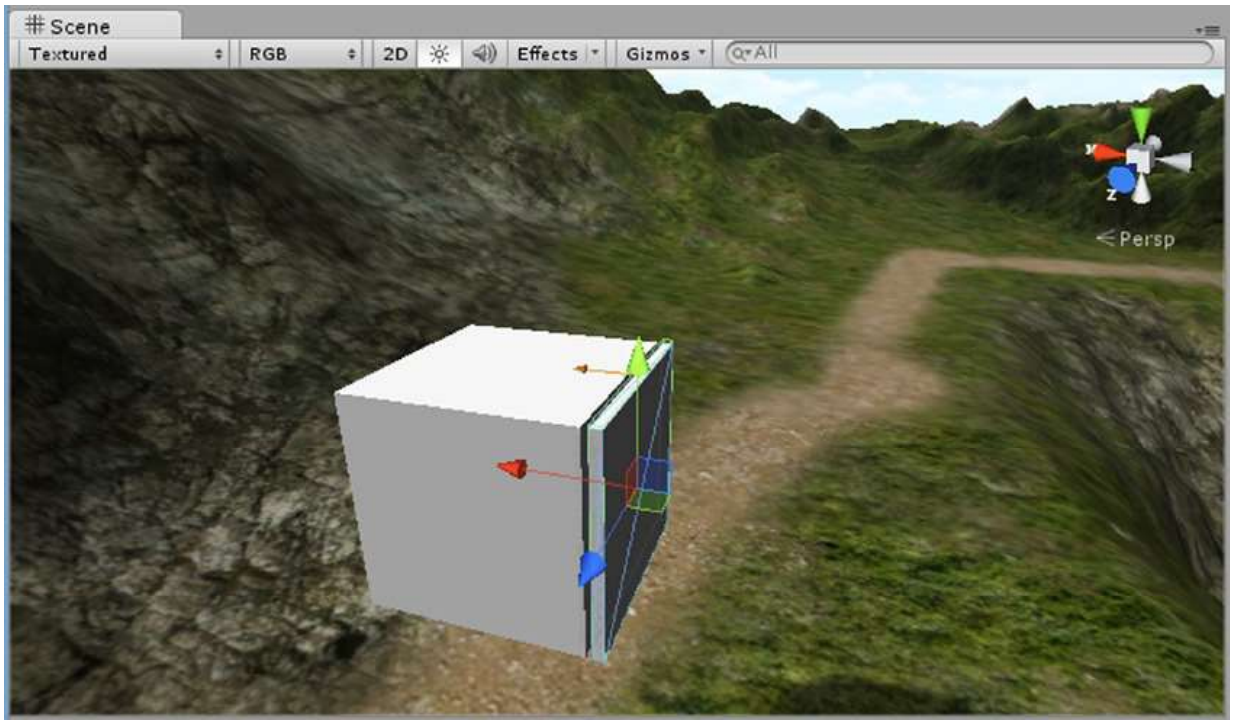

Figure 3-24. The Lid's X axis at its anchor point

6. Set the Axis X to **0** and the Axis Z to **1**.

Now the axis/anchor point aligns with the edge of the Lid object (Figure 3-25).

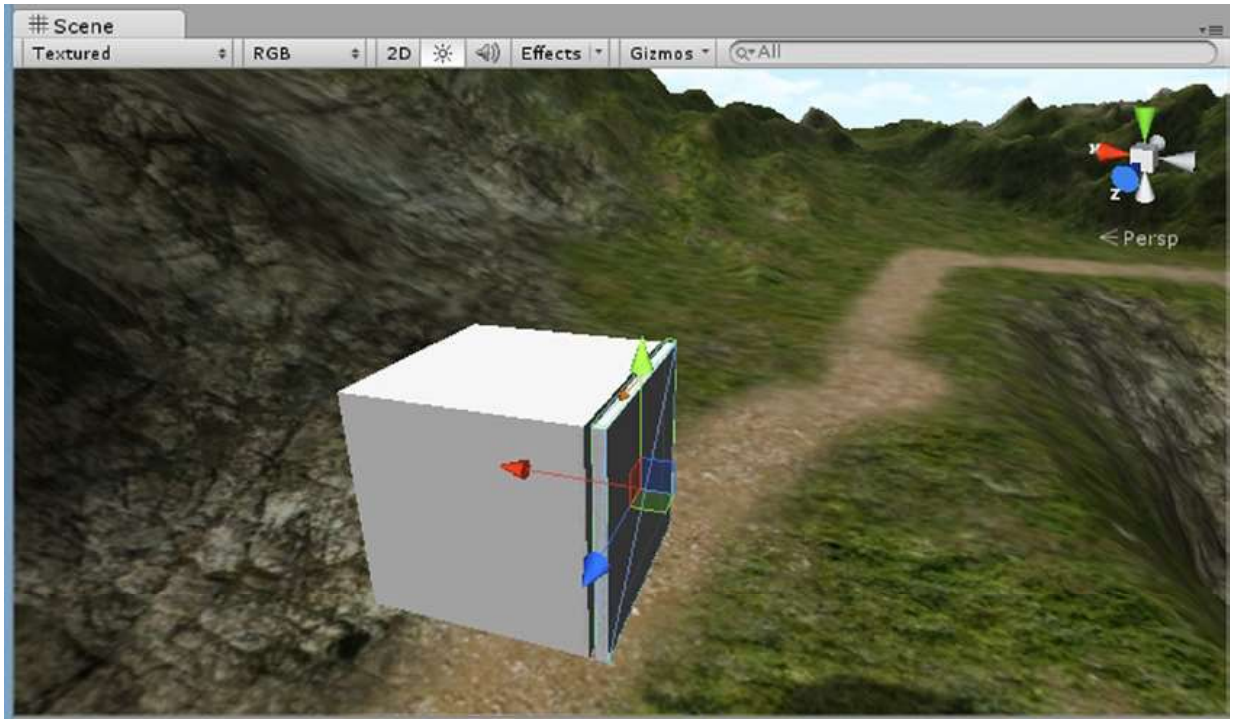

Figure 3-25. The Lid's Z axis as the hinge

7. Click Play.

The Lid is bumped by the spinning Cube (Figure 3-26).

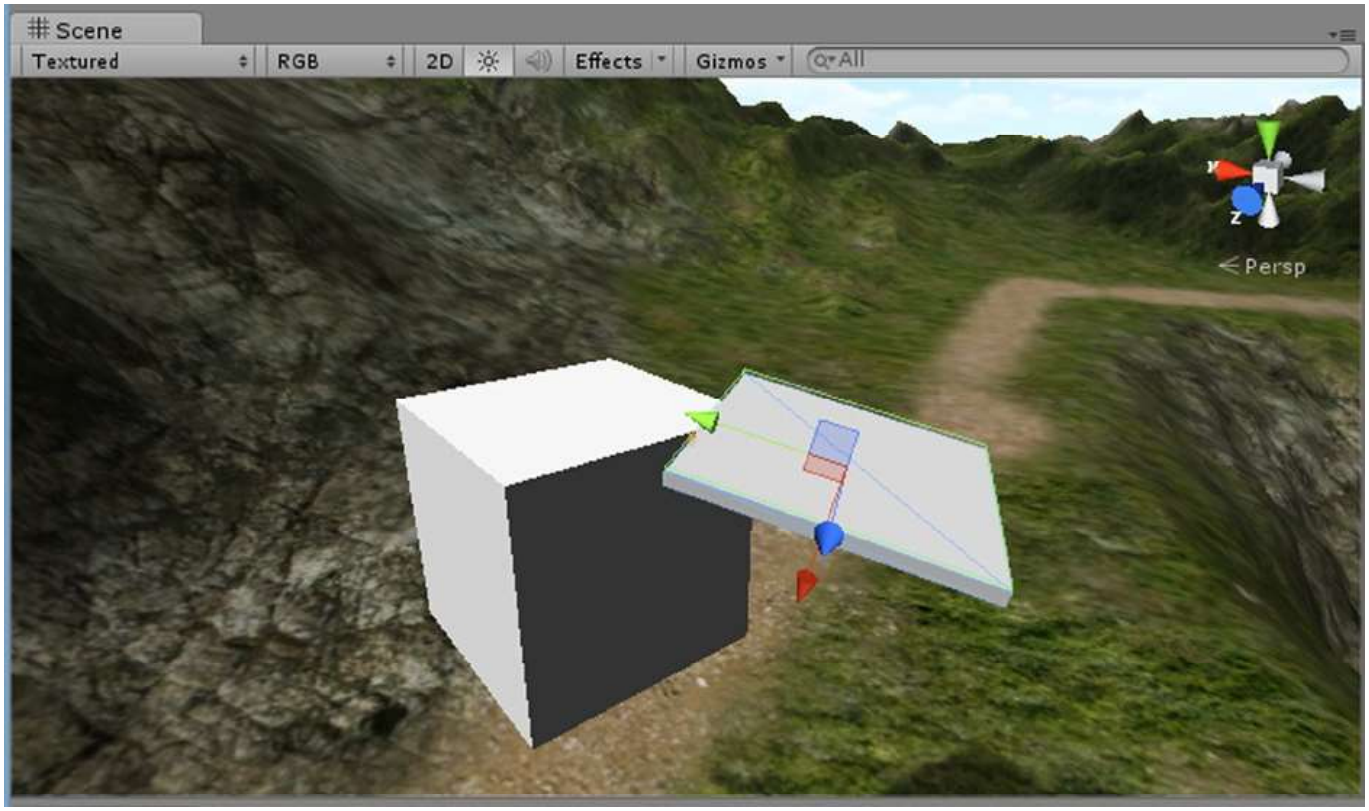

Figure 3-26. The Lid bumped as the Cube spins

The most logical thing to try next is to link the lid to the Cube.

8. Stop Play mode.
9. At the top of the Hinge Joint component, drag the Cube into the Connected Body field.
10. Click Play.

This time the Lid flaps open from the Cube as it spins along with it (Figure 3-27).

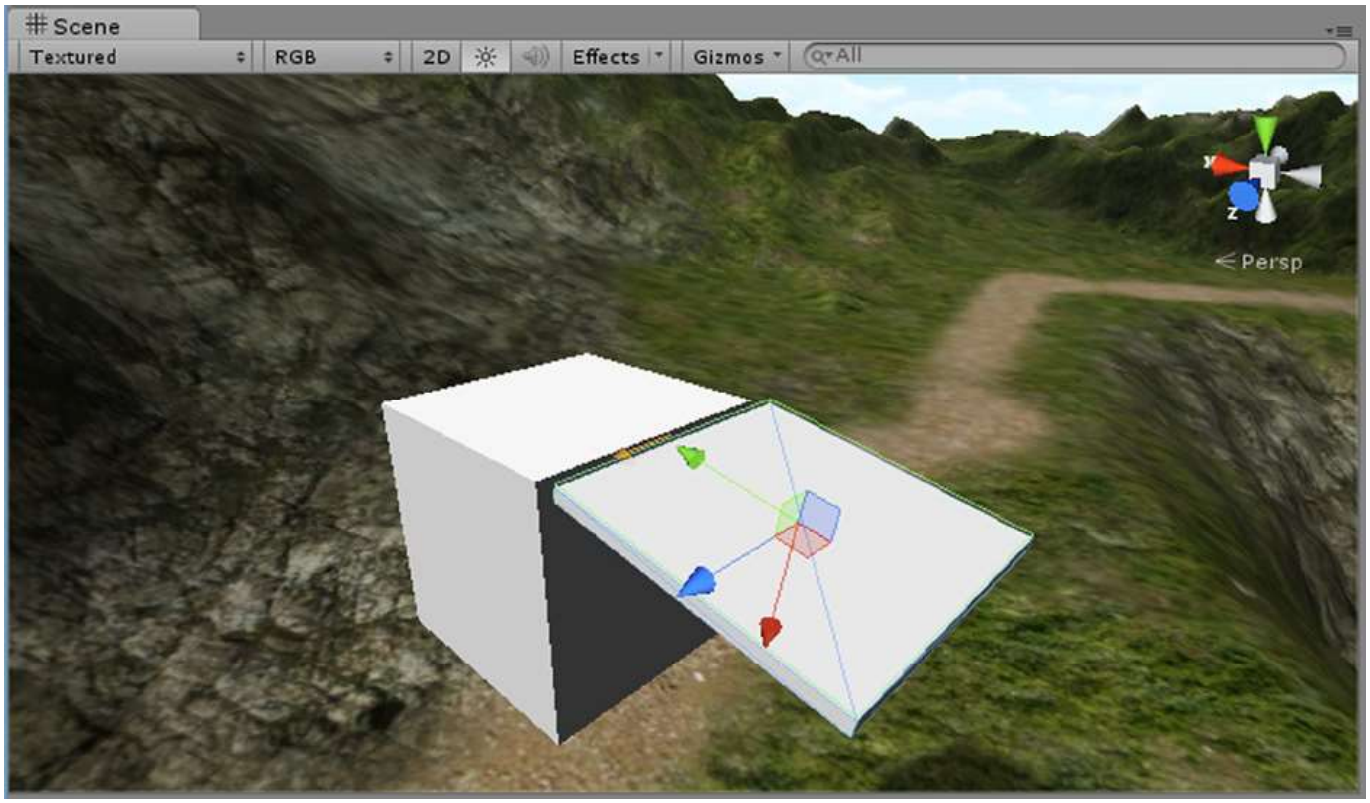

Figure 3-27. The Lid hinging correctly to the Cube as it spins

The flap is connected to the box at the same place as its anchor point, but that can be adjusted, giving you plenty of options for nonstandard objects. To change the anchor point, you would first have to uncheck Auto Configure Connected Anchor.

The Hinge Joint offers more options with its Spring and Motor parameters. With the right settings, the object will swing back and forth and attempt to get back to its original orientation when hit by an outside object. The flap on a Wheel of Fortune set up would make use of the spring settings as it is hit by the pegs.

The Motor lets you spin the object about its anchor.

1. Duplicate the Lid object, and rename it **Flap**.
2. Deactivate the Cube and the Lid.
3. Select the Flap, and in its Hinge Joint, click the Browse button to open the browser.
4. Select None to clear the Cube as the Connected Body (Figure 3-28).

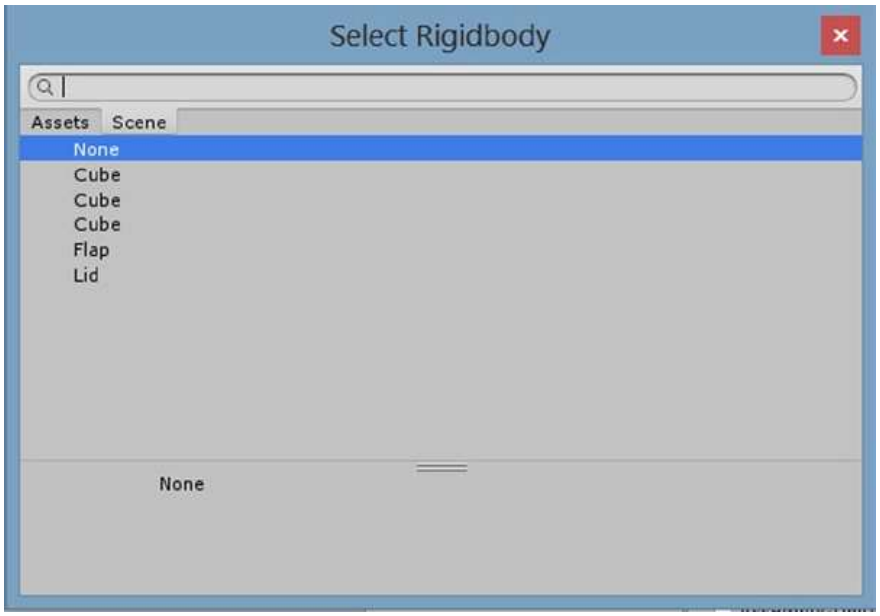

Figure 3-28. Clearing a parameter's field in the browser

5. Set the Flap's Mass to **1** in its Rigidbody component.
6. In the Hinge Joint, turn on Use Motor and open the Motor drop-down menu.
7. Set the Target Velocity to **100**, set the Force to **1**, and turn on Free Spin (Figure 3-29).

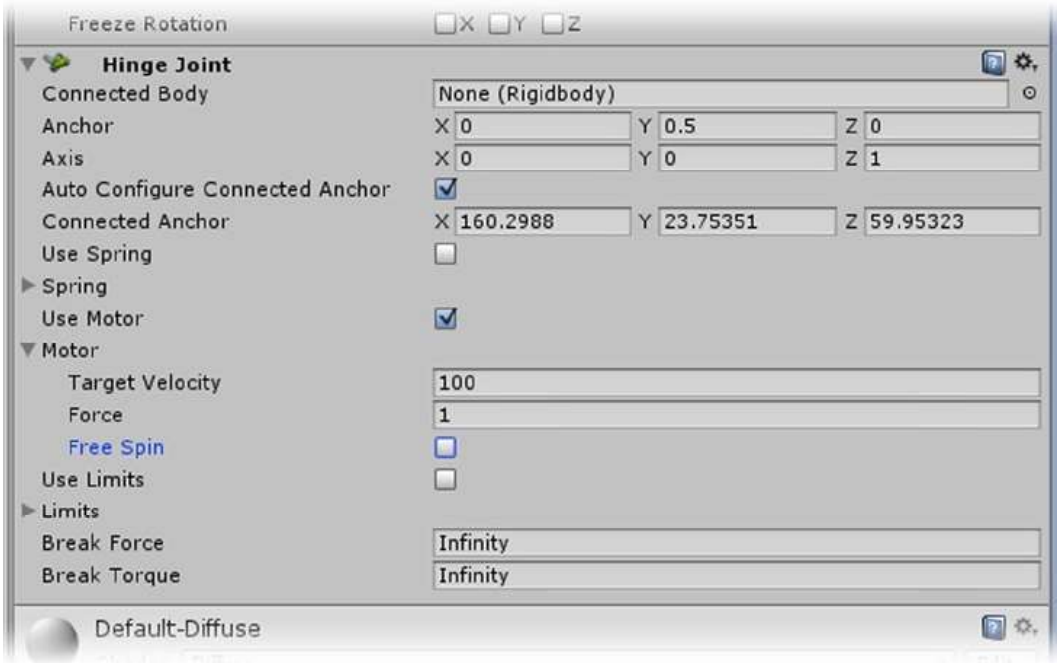

Figure 3-29. The Flap's Hinge settings

8. Click Play.

The flap spins slowly around.

9. Stop Play Mode.
10. Check Free Spin.
11. Click Play.

The Flap spins, but it is now affected by gravity and the Mass of the flap.

12. While still in Play mode, uncheck Use Motor.

The flap swings down and slowly comes to rest. To speed up the settling process, you could increase the Angular Drag in the Rigidbody component.

13. Turn the Use Motor setting back on.

The Flap starts to spin again.

14. Stop Play mode.
15. Turn off Use Motor.

Wind

You are probably wondering how wind interacts with the physics objects. Let's try it and see what happens.

1. Focus in on Flap in the viewport.
2. From the GameObject menu, Create Other, select Wind Zone.
3. From the GameObject menu, use "Move to View" to bring the Wind Zone into view.
4. Rotate the Wind Zone so that it points at the Flap (Figure 3-30).

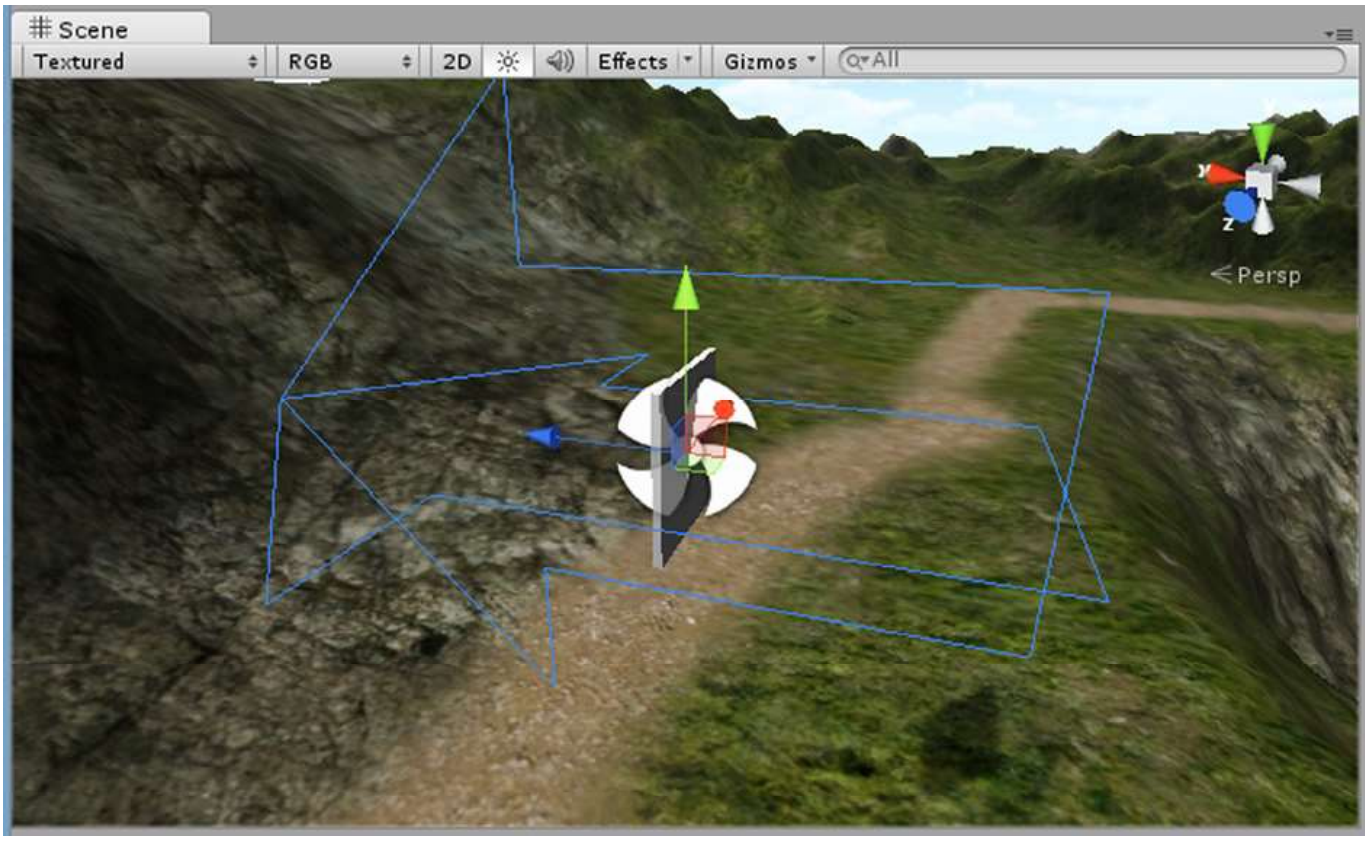

Figure 3-30. The Wind Zone pointing at the Flap

5. Click Play.

Nothing happens. The biggest hint that this is the expected behavior is that the Wind Zone is *not* found under the Physics submenu. So the takeaway here is that to get physics objects to "react" from "scene" wind, you will have to add a Constant Force component.

6. Stop Play mode.
7. Deactivate the WindZone.

Cloth

The next logical feature to take a look at is cloth. As you might guess, cloth can be costly, so use it sparingly. As a default, it uses a plane with 400 faces, but you can also load your own mesh into the Cloth component.

1. Focus in on the Flap.
2. From the GameObject menu, Create Other, select Cloth.
3. Rename it **Cloth**.
4. Rotate the Cloth 90 degrees so you can see it in the Scene view.

Default Cloth objects have no thickness, so they are only rendered on one side unless you use a two-sided shader in their material.

5. Scale the Cloth to X, **0.2**, Y, **1**, and Z, **0.12**.

6. Position it under the Flap (Figure 3-31).

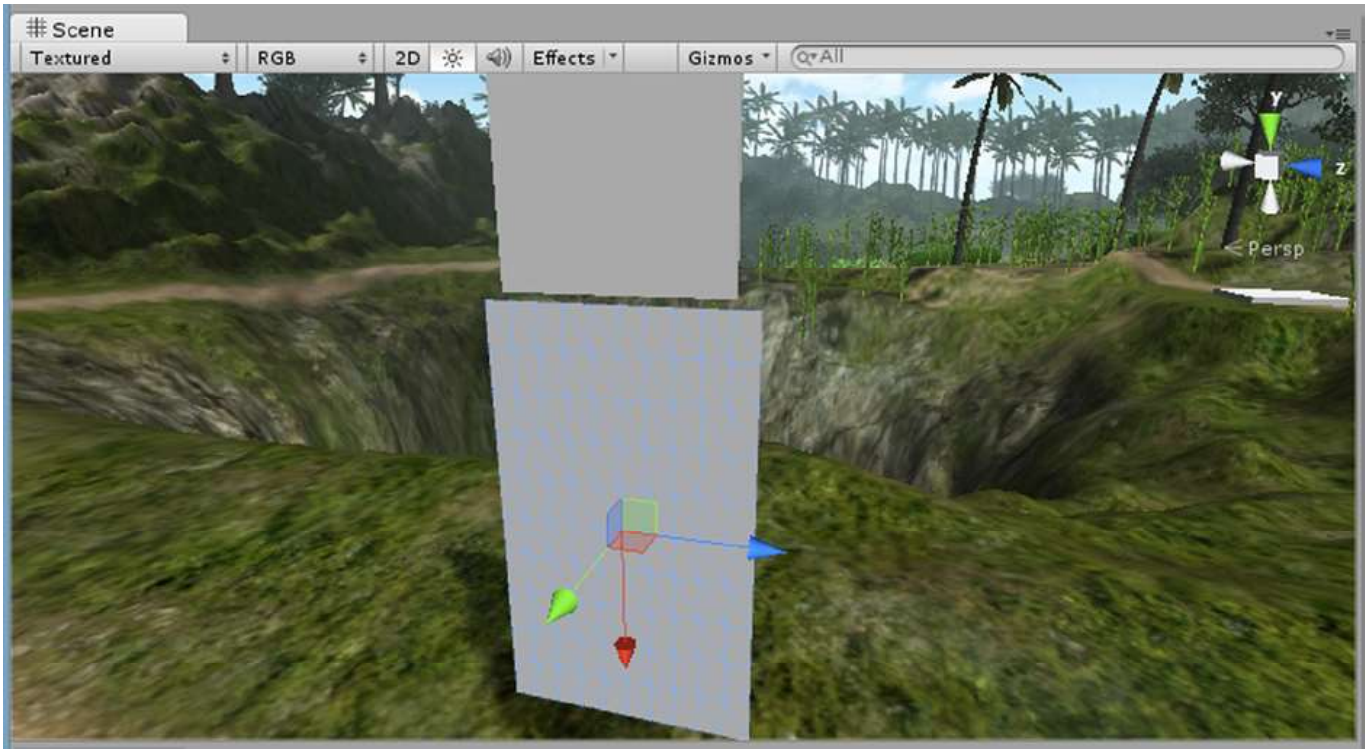

Figure 3-31. *The Cloth and Flap*

7. Click Play.

The Cloth object crumples to the ground, but its transform does not move (Figure 3-32).

Figure 3-32. The cloth crumpled on the ground; note the location of its transform

To attach an Interactive Cloth object to another gameObject or merely prevent it from falling, you must use Attached Colliders as the connectors. The Flap object has a collider, so you can begin by using it.

1. Select the Cloth Object.
2. At the bottom of the Cloth component, click to open the Attached Colliders array and set the Size to **1**.
3. Open the new Element 0 (Figure 3-33).

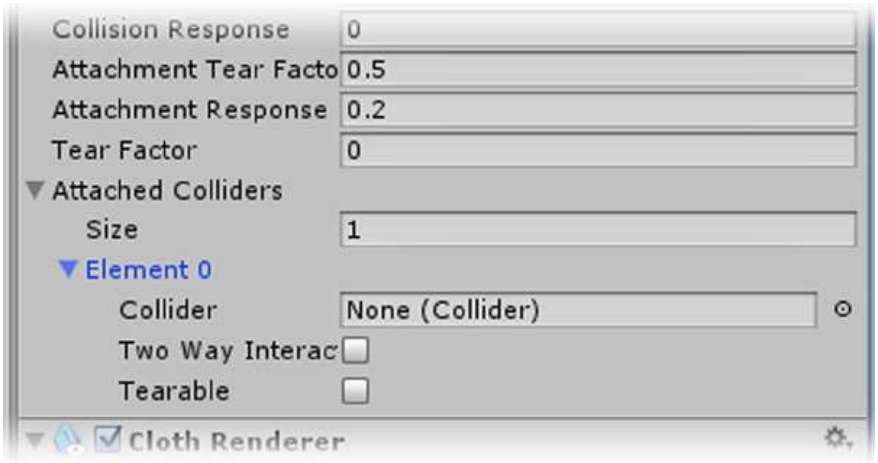

Figure 3-33. The Cloth component's Attached Colliders parameters

4. Drag the Flap object into the Element 0 Collider field from the Hierarchy view.
5. Click Play.

If the cloth is close enough to the Flap, it hangs quite nicely.

6. Stop Play mode.
7. Try moving the cloth farther below the Flap to see how close it must be to be held during run time.
8. Stop Play mode.

For the Attached colliders to work, the cloth's vertices must be within range of the colliders.

1. Create a new Cube, and name it **Hanger**.
2. Scale it down, and position it at one end of the flap.
3. Turn off its Mesh Renderer, and drag and drop it onto the Flap in the Hierarchy view.
4. Duplicate it, and rename it **Hanger2**.
5. Drag the new Hanger to the other side (Figure 3-34).

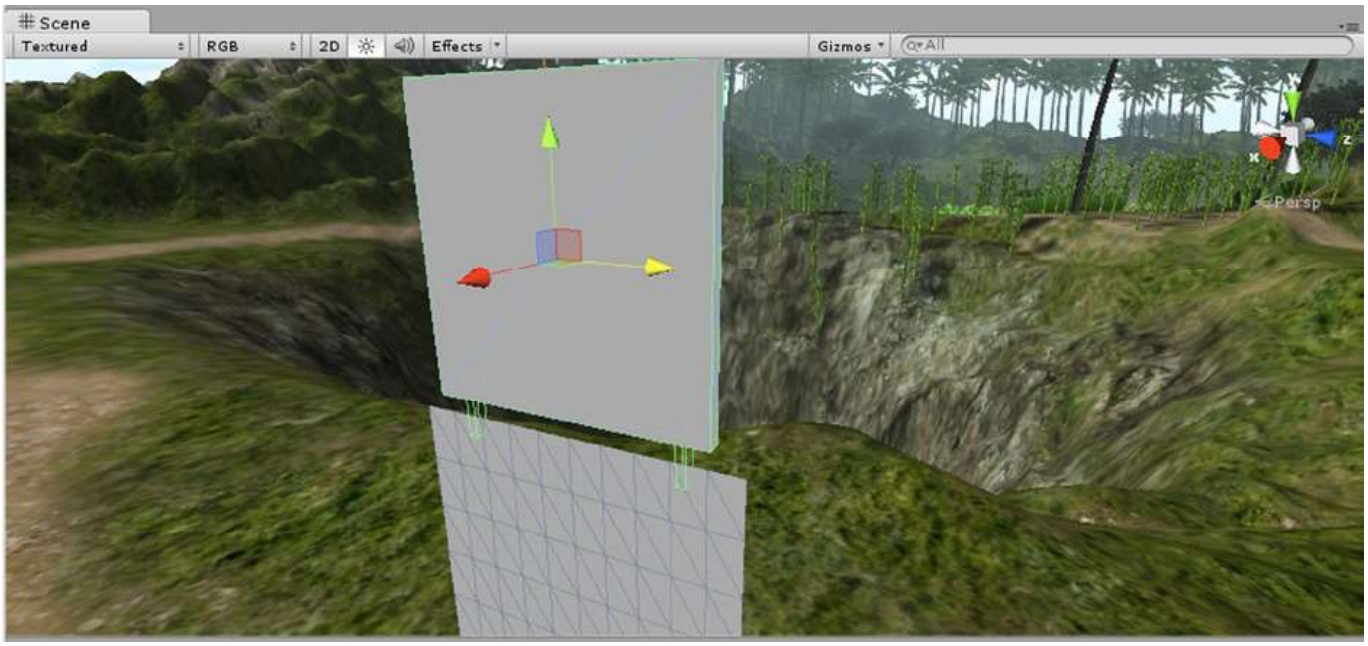

Figure 3-34. *The new Hanger objects in position*

6. Select the Cloth, and change the Attached Collider array Size to **2**.
7. Drag the two Hangers into the two elements.

8. Select the Cloth, and set the Interactive Cloth component's Stretching Stiffness parameter to **0.5**.

9. Click Play.

The cloth, attached only on each end, sags in the middle (Figure 3-35). The faces intersected by the two attached colliders retain their original positions.

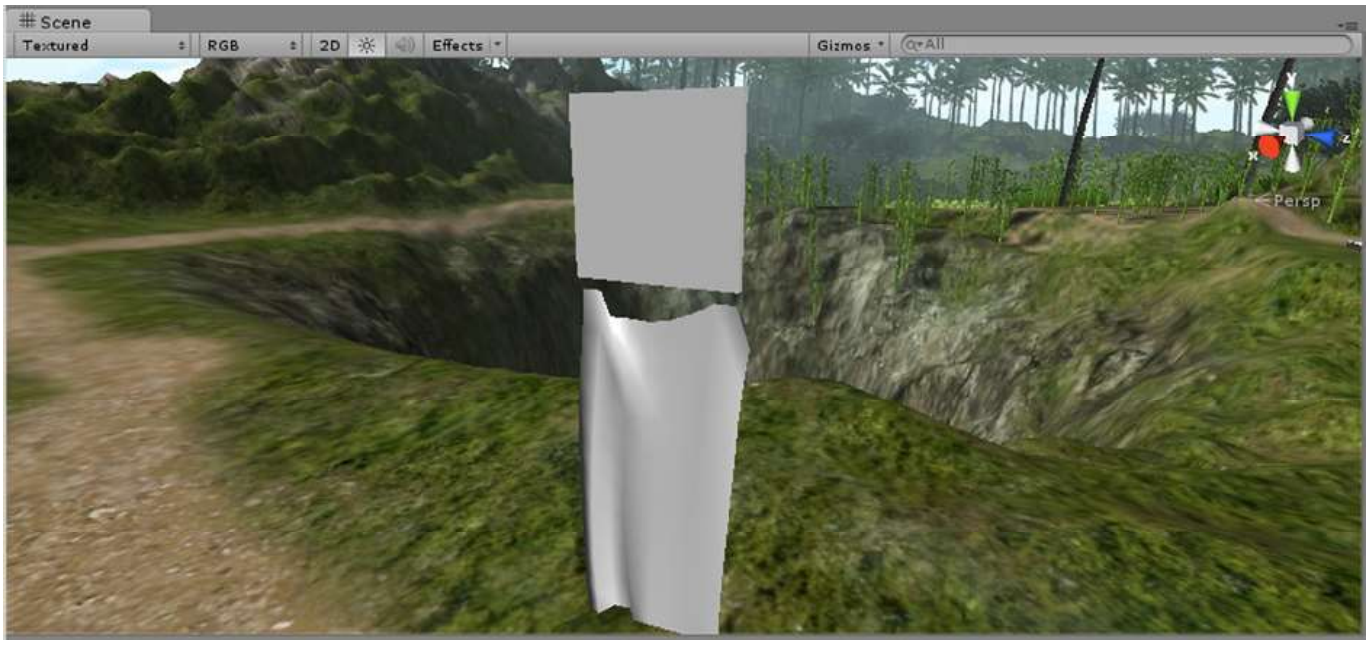

Figure 3-35. *The cloth held by the two Hanger objects in Play mode*

The cloth, like the Rigidbody objects, will not respond to Unity's Wind Zone. Like the Rigidbody component, Interactive cloth has its own version in the External Acceleration and Random Acceleration parameters. Both are world or global directions.

10. While in Play mode, try adjusting the two acceleration parameters. (Figure 3-36).

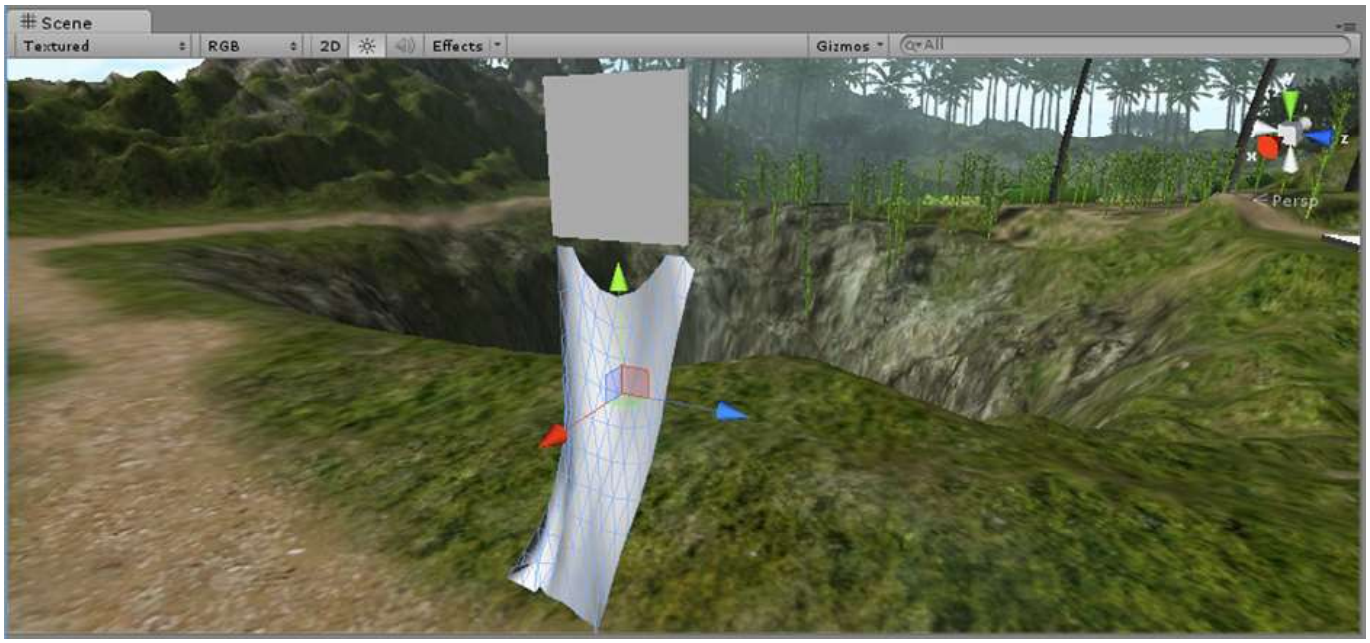

Figure 3-36. The cloth with a Random X Acceleration of 18

And finally, let's see what happens when the flap is set to spinning.

1. Stop Play mode, and make sure the Accelerations have returned to 0.
2. Select the Flap, and change its Hinge Joint's Axis Y to **1** and X and Z to **0**.
3. Turn on Use Motor in the Hinge Joint component.
4. Click Play, and watch as the cloth reacts to the torque (Figure 3-37).

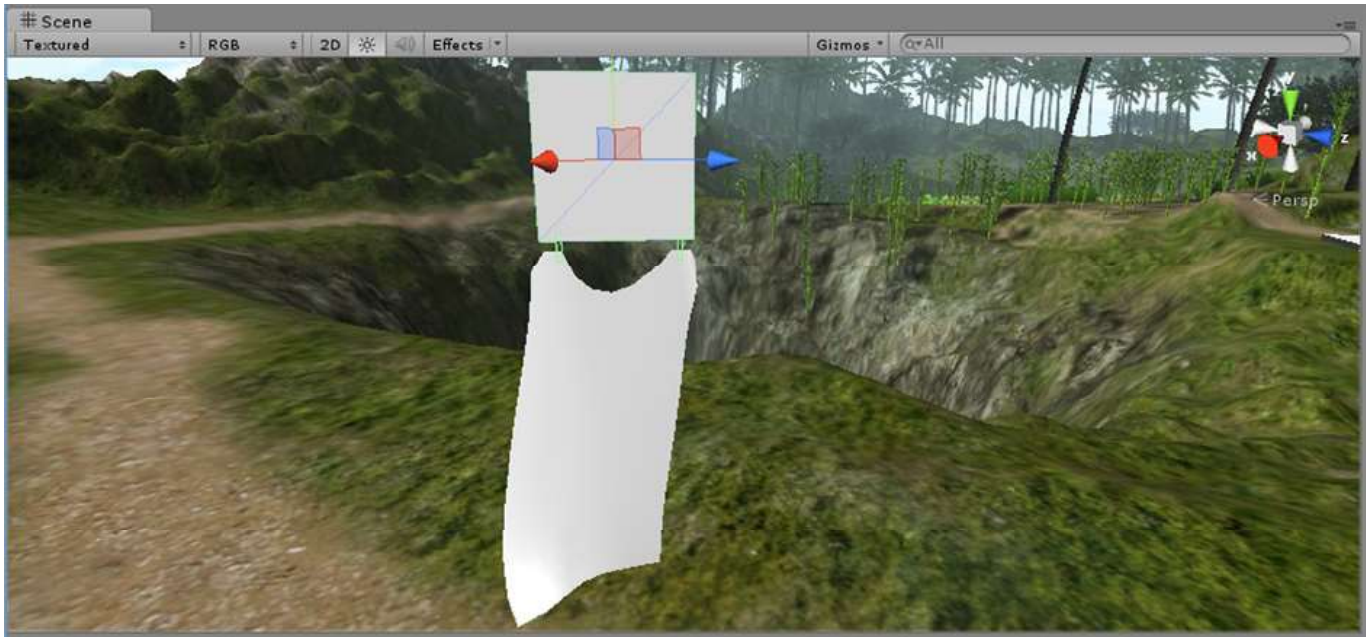

Figure 3-37. The cloth reacting as the Flap spins

Because the two Attached Colliders have the Flap as their parent, the Cloth follows right along. The most notable issue you will notice is that the cloth is rendered only on one side. To use it out in the open, you would have to use a material with a two-sided shader. You could also use another cloth for the back side, but that would use twice as many resources. If you were using a custom mesh for the cloth, you could use the Self Collision check, but that also is expensive.

5. Stop Play mode.

Cloth has many parameters that will let you fine-tune it to resemble the behavior of velvet to canvas to silk. Bending Stiffness, Thickness, and Friction are a few of the parameters that are a good starting point for customization.

Interacting with the First Person Controller

So far, your tests have been centered around objects with physics-based components. After seeing the lack of reaction with the Wind Zone, you are probably wondering how the First Person Controller will interact with the various objects. If you remember, the First Person Controller does not have a true Rigidbody component, but it does have a collider.

1. Click Play.
2. Drive the First Person Controller over to the Flap (Figure 3-38).

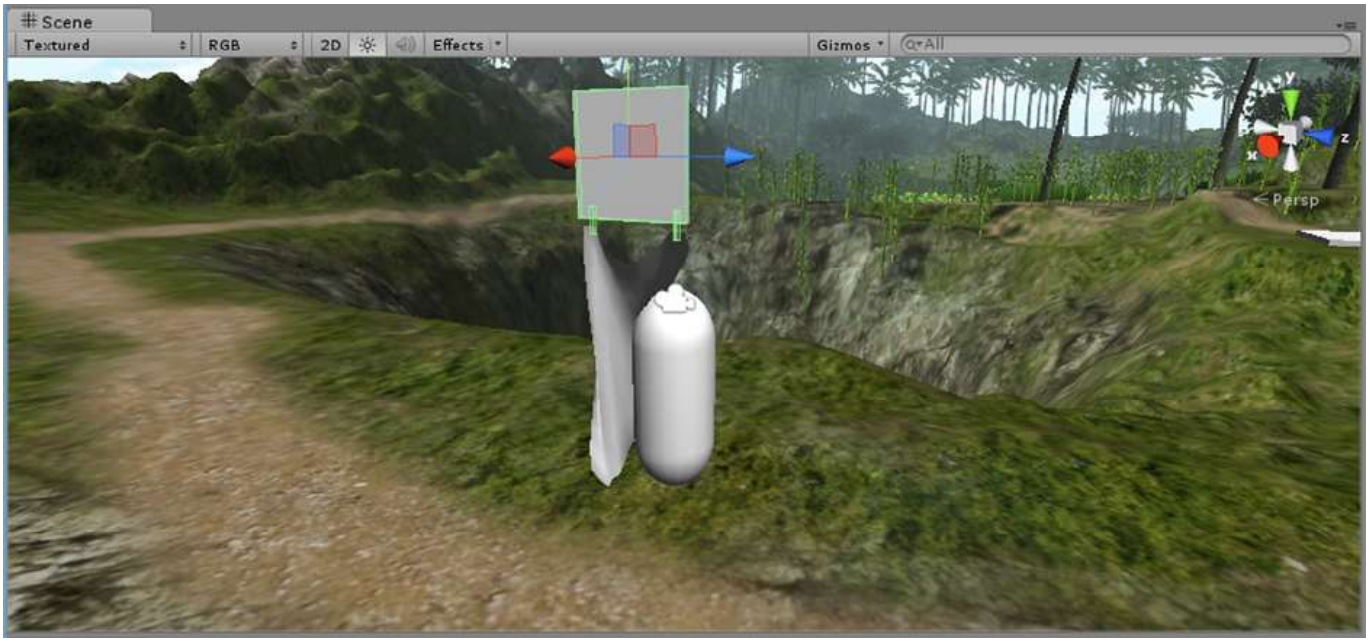

Figure 3-38. *The cloth reacting as the First Person Controller moves into it*

The Cloth clearly reacts to the First Person Controller's Capsule collider. You will find, however, that you may have to increase the Thickness and reduce the Friction parametersto prevent it from passing though the First Person Controller's collider.

With the Rigidbodies, you will want to try two experiments. The first is to see if moving Rigidbodies can affect the First Person Controller, and the second is to see if the First Person Controller can affect Rigidbodies.

1. Stop Play mode.
2. Deactivate the Cloth and Flap objects.
3. Duplicate the middle cube.
4. Activate the new cube, and move it so that its Constant Force will push it into the First Person Controller.
5. Click Play, and watch the result.

The cube either stops at the First Person Controller or twists and continues along its way.

6. Increase the cube's Force value to **1000** (or **-1000** if you are using a negative number).
7. In the Rigidbody component, increase the Cube's Mass to **50**.
8. Click Play.

The result is the same: the cube is unable to move the First Person Controller.

Next you will test the opposite scenario by driving the First Person Controller at the box.

1. Stop Play mode.
2. Set the Cube's Mass back to **1** and its Force back to **10** (or **-10**).
3. Click Play, and drive the First Person Controller at the moving cube.

The First Person Controller is able to push the Cube away.

4. Stop Play mode.
5. Set the Cube's Force to **0**.
6. Click Play, and drive the First Person Controller into the Cube.

When the cube is not being moved by physics, the First Person Controller is not able to influence its movement. To directly affect an object with a Rigidbody component, you would have to apply a physics force through scripting using the First Person Controller's direction to calculate the direction of a one-off force.

7. Stop Play mode.

By now, you probably realize why game physics are not comparable to real-world physics. Not only are the algorithms stripped down for speed, but the functionality itself is severely limited to control CPU usage. The important thing to remember when designing your game's functionality is not to take any physics-based functionality for granted. Some desired behavior can be enabled though the Rigidbody and other physics-related component's parameters, but other behaviors, if even feasible, may require specialized scripting.

First Build

With the addition of the First Person Controller, you were able to move around your scene and interact with a few objects. The accomplishment may be a long way from a creating a AAA title, but it is enough to let you try your first "build." Unity makes it very easy to "build," or output, your application as an executable. From Mac or Windows, you can make a game that can run either as a desktop application or in a web browser. Outputting to mobile requires extra steps, a mobile device or emulator and, in some cases, a specific operating system. It also has several more stringent guidelines and requirements for content and scripting. You will be creating a desktop application.

Creating a build takes just a few mouse clicks if you use the defaults. Before you proceed with your first build, there are a few settings worth looking at.

1. From the Edit menu, Project Settings, select Player.
2. In the Inspector, examine the top section.

You can put your name in the company Name field. The Product Name is taken from the Project name, but can be changed.

3. Change the Product Name to **MyFirstUnityApp**.
4. For the Default Icon, click the Select button and choose the Corn Tassel (Figure 3-39).

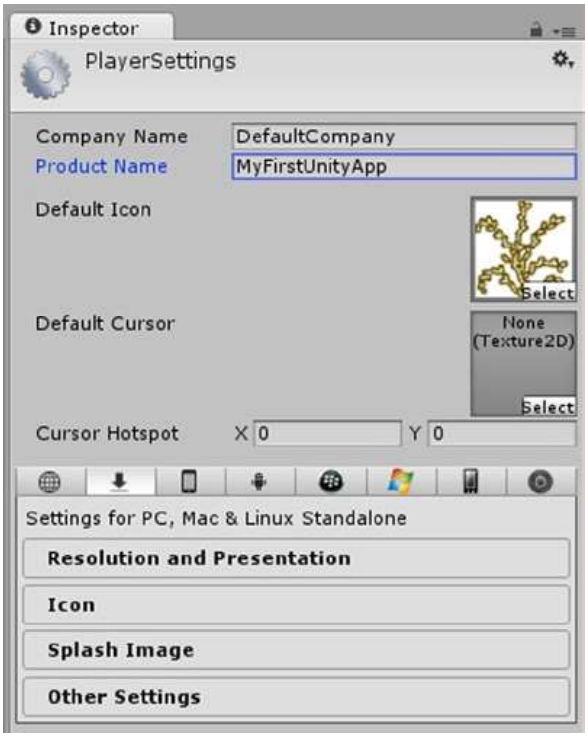

Figure 3-39. The Player Settings, top section

With Default Cursor, you have the option of using a custom hardware cursor. This means you can use an image of your choice that is directly handled by the machine's operating system. The cursor will always be drawn on top, but it has a limit of 128 x 128 pixels in size. To use a particular image, it must be marked as Cursor type in its import settings. If the Operating system can't support custom cursors, it will drop back to its own default.

1. Select the Corn Tassel image in the Project view.
2. Duplicate it, and name it **CornTasselCursor**.
3. In the Inspector, change its Texture Type to Cursor and click Apply (Figure 3-40).

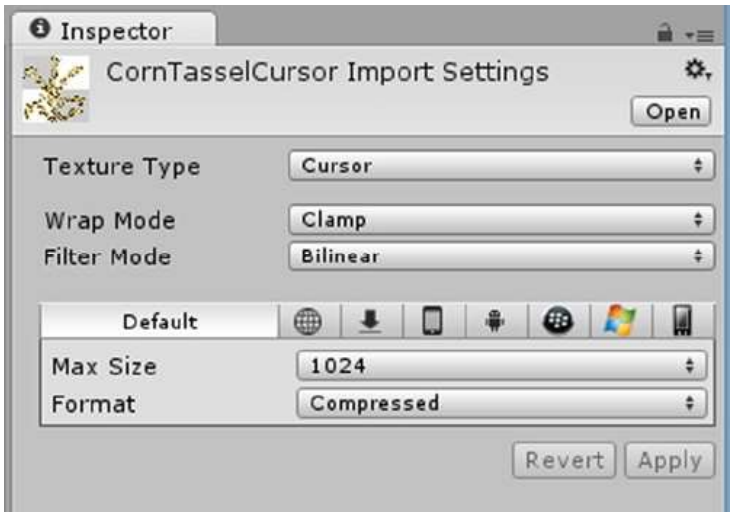

Figure 3-40. *The CornTasselCursor images set to Cursor Texture Type*

4. Return to the Player from Project Settings.
5. Drag the CornTasselCursor image onto the Default Cursor thumbnail.
6. Move the cursor over to the Game view to see your new cursor (Figure 3-41).

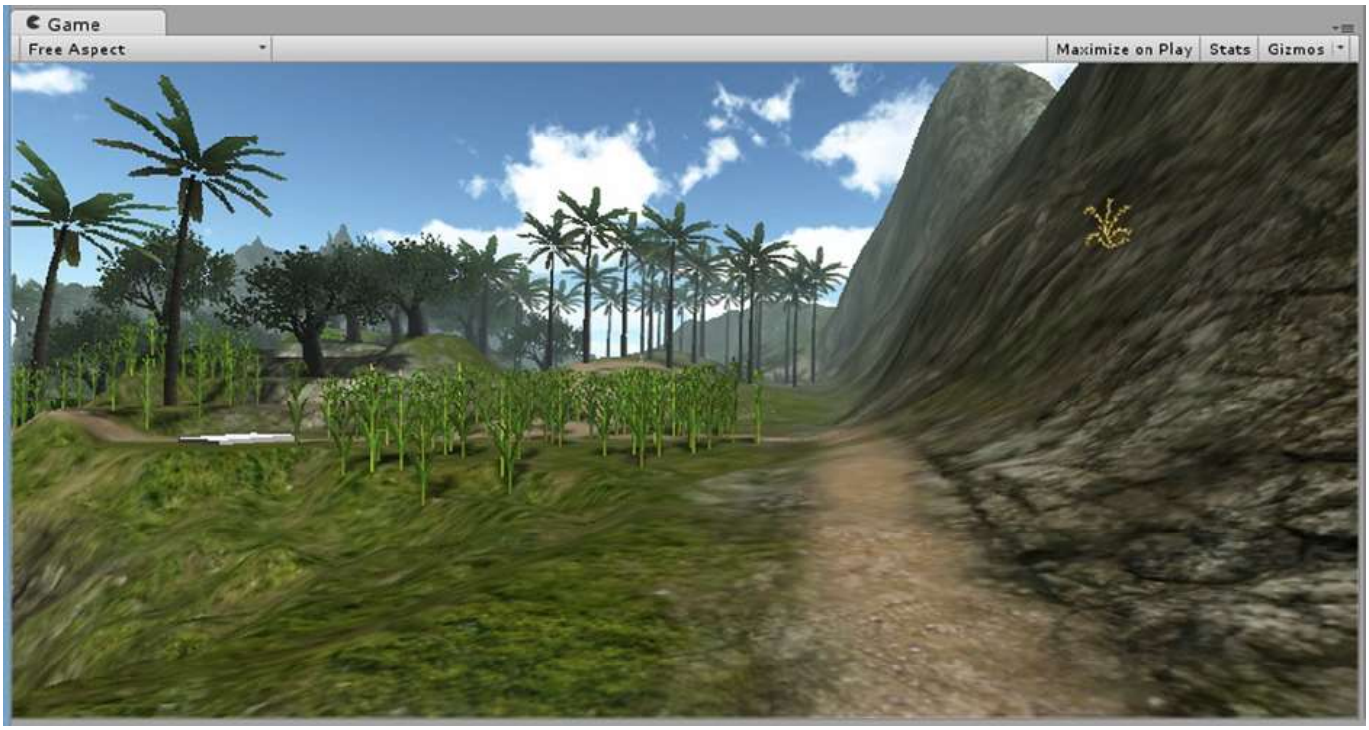

Figure 3-41. The new hardware cursor in the game view

The next section of the Player Settings allows you to customize settings according to target platform (Figure 3-42).

Figure 3-42. Platform settings

1. Take some time to check out each platform's settings, and then return to "Settings for PC, Mac & Linux Standalone."
2. Uncheck Default Is Full Screen.
3. Check Resizable Window.

Display Resolution Dialog, also known as the Config Dialog, is the screen that comes up on startup, allowing the user to change resolution, quality settings, and map keys. As you can see by its choices, you can block user access to this dialog.

In the Icon section, you will find that the CornTasselCursor icon has been automatically loaded for you.

The Splash Image section is available for Pro users only. There, they can set a custom image for the Config Dialog banner.

Other Settings is where you set the Render Path, among other things. If you have Pro, you have to change to Deferred Lighting to make use of dynamic shadows *and* baked lighting in your scene.

Having tweaked a few of the Player settings, you are ready to try your first build. The first thing you must do is save your work.

1. Save the scene, and save the project.
2. From the File menu, select Build Settings.
3. Click Add Current to add the currently active scene into the build (Figure 3-43).

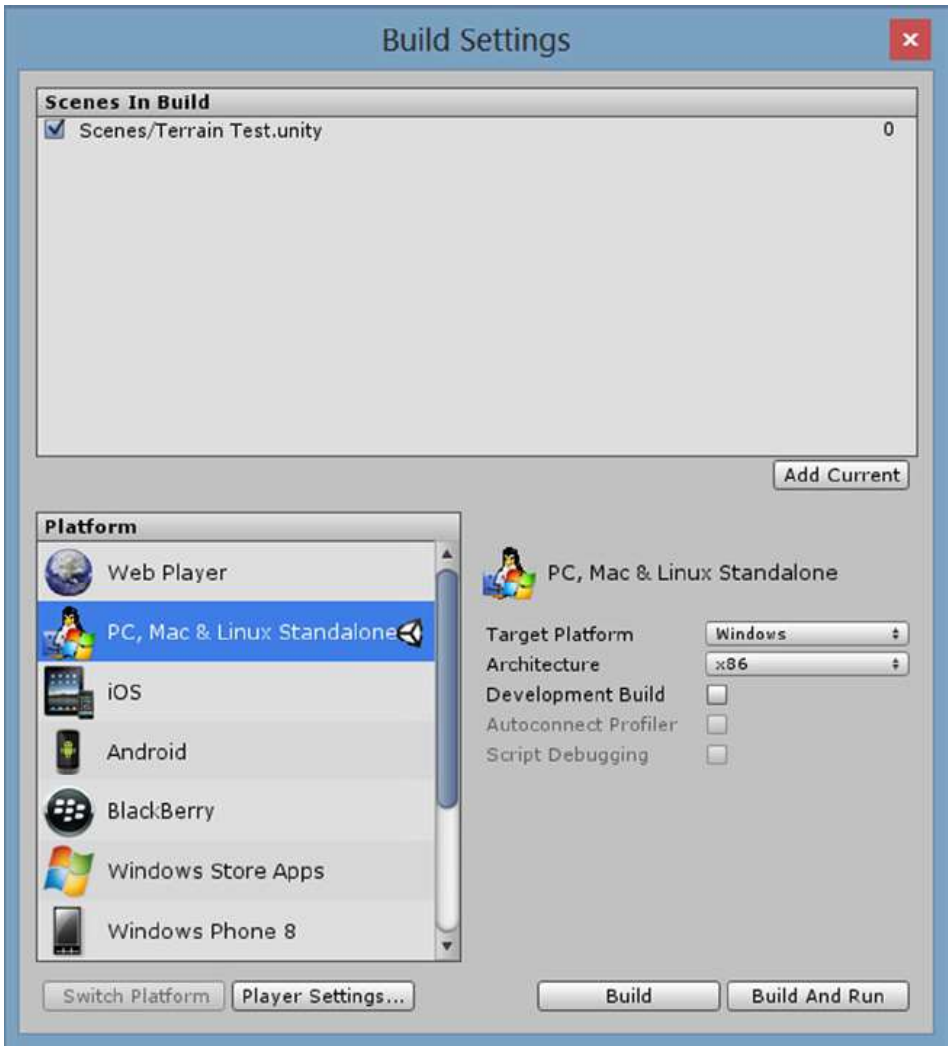

Figure 3-43. Adding the current scene to Build Settings

4. Make sure the Target Platform is correct. (It should automatically be set to your machine.)
5. Click Build And Run.
6. Save the output somewhere other than in your project folder, and name it **TestBuild**.

The project is built, and the Config Dialog pops up when it is ready to play.

7. Make sure Windowed is checked, and change any other settings as you care to (Figure 3-44).

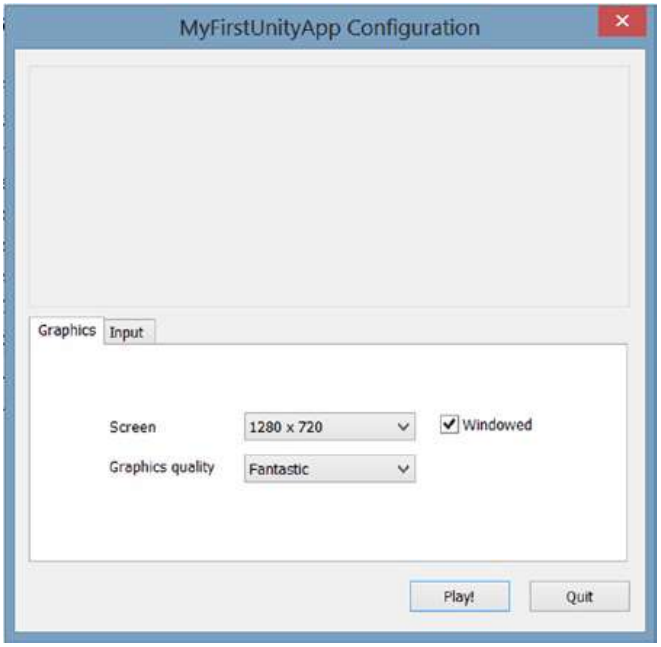

Figure 3-44. The Config Dialog

8. Click Play.

Your first Unity scene pops up, and you can drive around the scene and interact with anything you left active (Figure 3-45).

Figure 3-45. The new game running on the desktop

9. Take a minute to inspect the results.

The application name is TestBuild, but the title (as seen in the windowed title bar) is MyFirstUnityApp, which you set in the Player Settings.

10. Look in Windows Explorer (if you are on a PC) for the TestBuild application.

You will find TestBuild.exe and a folder called TestBuild_Data. When you distribute your games for the PC desktop deployment, the .exe and the data folder must be in the same location.

The content for this book was created using Windows. Other than a few keyboard differences, Unity should look and act the same on whatever platform you are authoring on. It is easy to build for other platforms.

1. Close the running game, and return to the Unity editor.
2. In the Build Settings dialog, change the Target Platform to Mac OS X (or Windows if you are already on a Mac).
3. After the changeover is complete, click Build.
4. Save the new build in the same place as the first, with the same name.

The extension type will be different (.app), so there will be no conflicts. The Mac application is contained in a single folder named TestBuild.app.

5. If you have access to the other platform, feel free to test the new version.
6. Back in the Unity editor, close the Build Settings dialog.

Summary

In this chapter, you added a First Person Controller into your terrain scene and were able to travel around in it. You discovered that several script components that made up the First Person Controller contained a lot of settings you could tweak to change the First Person Controller's behavior. You found that various functionality, such as jumping and the ability to ride on moving platforms, was optional. Moving platforms offered some interesting problems with regard to the height of the platform and how the platform could interact with the First Person Controller.

During your experiments with the First Person Controller, you had an introduction to Unity's InputManager. You were able to not only reassign input keys, but to create your own virtual keys to expand an application's versatility. At the very end of the chapter, you got to see where the player was allowed to change the key "mapping" before playing the game.

Having a means of traversing the scene, you got some firsthand experience with colliders. At some point, you realized you were able to go through the terrain trees, and then you learned how to apply colliders to assets used as terrain trees.

Moving on to physics, you experimented with the Rigidbody component and saw how it works with the colliders to provide lots of non-key-framed animation in your scene. You found that any of its position or rotation axes could be "frozen" to restrict the physics-based reactions. The Physic Material in the collider component gave you a means of tailoring the object's reactions according to a virtual material assignment. You found that you could add a Constant Force component to move

or rotate (torque) your physics object. With the Hinge Joint, you discovered a means of combining physics objects. That component, you discovered, has built-in force and torque settings. One important thing you learned was that adding a Rigidbody component to objects that were animated by means other than physics was more efficient. To do so, you had to check Is Kinematic.

After a quick look at cloth, you tested your First Person Controller against the objects with Rigidbody components and the cloth object to see what reactions occurred. In the end, you found that game physics were not always predictable due to the need to keep the frame rate at acceptable levels.

Finally, you made your first build. Before doing so, you tweaked a few of the Player settings just to get a feel for what could be manipulated.