HOUR 18
Animators

What You'll Learn in This Hour:

- The basics of animators
- How to use an animator's state machines
- How to control animations from script via parameters
- An introduction to blend trees

In this hour, you'll take what you've already learned about animations and put it to use with Unity's Mecanim animation system and animators. You'll start by learning about animators and how they work. From there, you'll look at how to rig or change the rigging of models in Unity. After that, you'll create an animator and configure it. Finally, you will see how animations are blended to produce amazingly realistic results.

NOTE

Warning: Get Your Hands Dirty!

This section is like one big Try It Yourself. Make sure you save your project in between the practical exercises, as each builds on the one before. You will certainly want to be in front of your computer while you read this section. The topic is best learned by doing!

Animator Basics

All animation in Unity starts with an Animator component. In Hour 17, while

you were creating and learning about animations, you were using animators without really knowing it. At its heart, Unity's animation system (Mecanim) comprises three pieces: the animation clip, the Animator controller, and the Animator component. These three pieces all exist to make your characters come to life.

Figure 18.1, which is taken from Unity's online documentation about the Animator component (https://docs.unity3d.com/Manual/class-Animator.html), shows how these parts relate to one another.

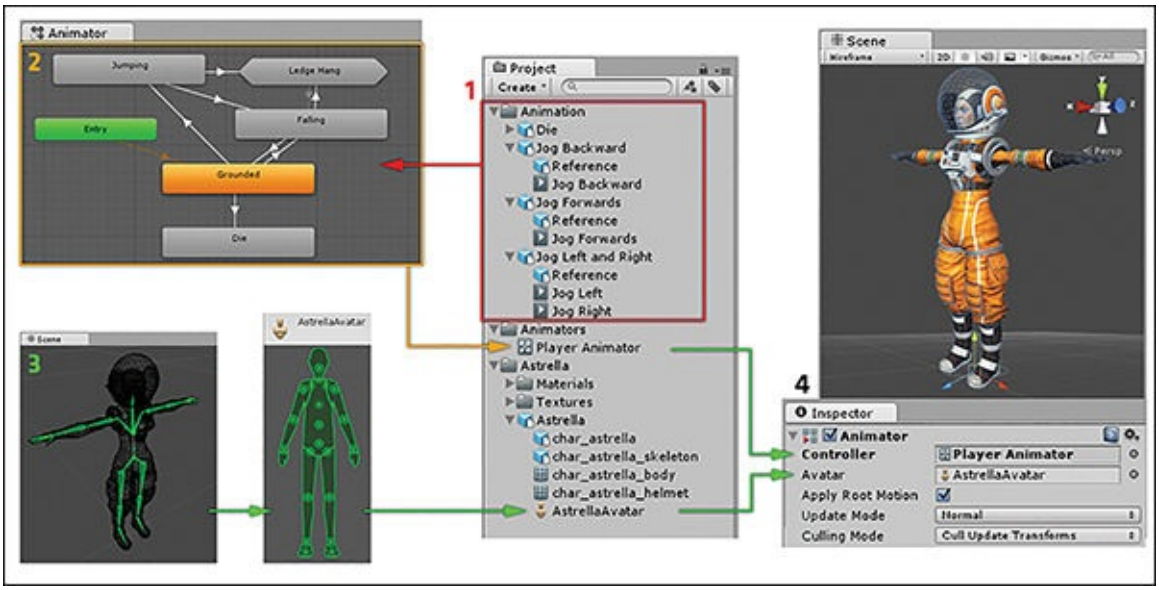

FIGURE 18.1
How the parts of a humanoid animation relate to each other.

The animation clips (see #1 in Figure 18.1) are the various motions that you either import or create in Unity. The Animator controller (#2) contains your animations and determines which clips should play at any given moment. Models have something called an avatar (#3) that acts as the "translator" between the Animator controller and the rigging of the model. You can generally ignore the avatar because it is set up and used automatically. Finally, both the Animator controller (which you can just call the "controller") and the avatar are put together on the model using an Animator component (#4). Does it seem like a lot to remember? Don't worry. Most of this stuff is either intuitive or automatic.

One of the best things about Unity's animation system is that you can use it to "retarget" an animation onto other game objects. If you animate a cube, you can also apply that animation to a sphere. If you animate a character, you can apply the animation to another character with the same rigging (or different rigging, as

the animation to another character with the same rigging (or different rigging, as you will soon see). This means you can, for example, have an orc and a man doing the same happy dance together.

NOTE

Analyzing a Specific Use Case

In order to get the most out of this hour, you will work with a very specific use case: 3D animations on a humanoid model (a very common use case to be sure). This will allow you to learn about 3D animations, importing models and animations, working with rigs, and using Unity's awesome humanoid retargeting system. Just remember that, with the exception of humanoid retargeting, everything covered in this hour applies completely to any other type of animation. So, if you are building a multipart 2D animation system, all the knowledge learned here still matters.

Rigging Revisited

In order to begin building a complex animation system, you first need to ensure that your model's rigging is prepared. Recall from Hour 17, "Animations," that models and animations have to be rigged exactly the same way in order to function. This means that it can be very difficult to get animations made for one model to work on a different model. Therefore, animations and models are generally made specifically to work together.

If you are using a humanoid model, though (two arms, two legs, head, and torso), you have the ability to tap into Mecanim's animation retargeting tools. With the Mecanim system, humanoids can have their rigging remapped in the editor without using any 3D modeling tools. The result is that any animation made for a humanoid model can work with any other humanoid you have. This means animators (the people, not the Unity asset) can produce large libraries of animations that can be applied to a large range of models using many different rigs.

Importing a Model

For this hour, you will use Ethan, a model from the Characters standard asset pack. This model comes with a lot of different items, and you will go through each piece to ensure that it is configured properly. To import the model, select **Assets > Import Package > Characters**. Leave everything checked and click

Import.

Now go ahead and find Ethan in your Project tab under Assets\Standard Assets\Characters \ThirdPersonCharacter\Models (see Figure 18.2).

If you click the little arrow to the right of the Ethan file, you can expand the model to see all the constituent parts (see Figure 18.2). How these parts are structured depends on how the model was exported from the 3D application used to make it.

These components are, from left to right, Ethan's body with texture, the textured glasses, the definition of the skeleton, the raw EthanBody mesh, the raw EthanGlasses mesh, and finally a definition of Ethan's avatar (which is used for rigging).

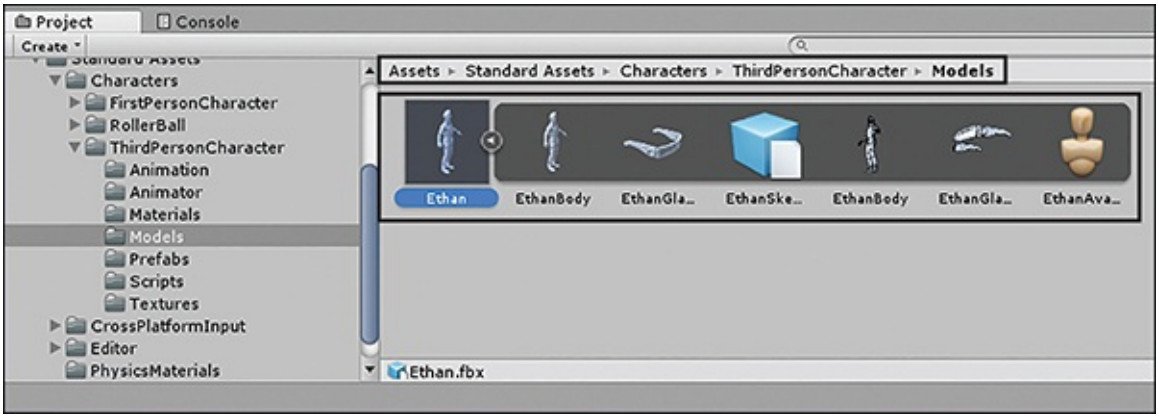

FIGURE 18.2
Finding the Ethan model.

NOTE

Previewing Meshes

If you click either the Ethan or the Glasses model in the tray, you should notice a small preview window at the bottom of the Inspector. (If not, drag it up to show it.) Here you can rotate that submodel around to take a look at it from all angles (see the bottom of Figure 18.3).

When you're done looking at the components, collapse the Ethan.fbx tray by clicking the arrow to the right of the asset.

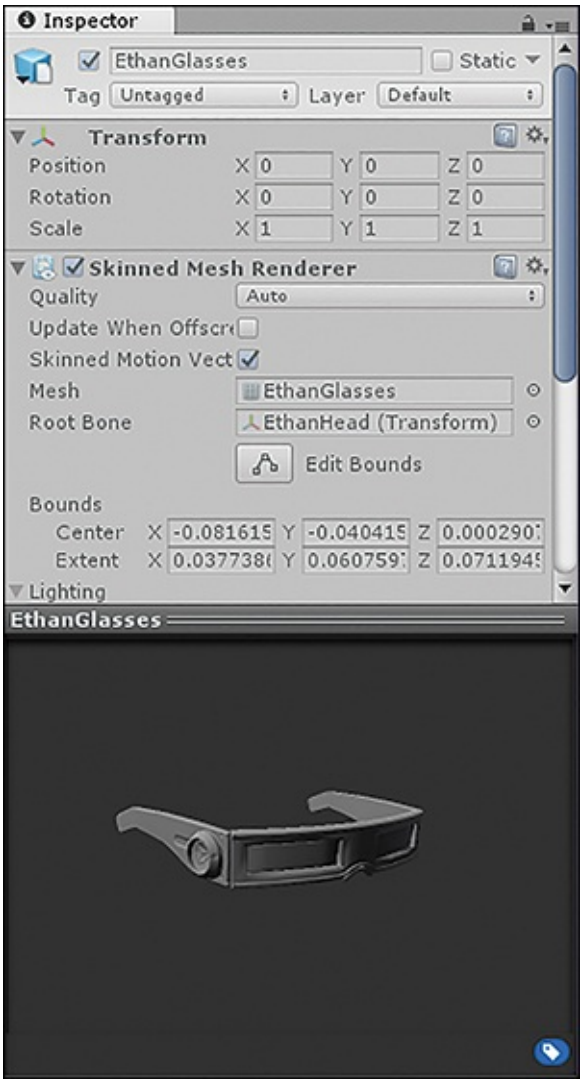

FIGURE 18.3
The model's Inspector view.

Configuring Your Assets

Now that you have imported a model and animations (which came with the rest of the assets), you need to configure them. The process for configuring animations is fairly identical to configuring models.

With a model selected, you can see the import settings listed in the Inspector view. The Model tab is home to all the settings that dictate how the model is imported into Unity. These items can be safely ignored for the purposes of this hour. The tab you are concerned with for now is the Rig tab. (The Animations tab is covered a little later in this hour.)

Rig Preparation

You configure a model's rig in the import settings, under the Rig tab in the Inspector view. The property you are most concerned with here is Animation Type (see Figure 18.4). There are currently four types available in the drop-down: None, Legacy, Generic, and Humanoid. Setting this property to None causes Unity to ignore this model's rig. Legacy is for Unity's old animation system and shouldn't be used. Generic is for all nonhumanoid models (simple models, vehicles, building, animals, and so on), and all models imported into Unity default to this animation type. Finally, Humanoid (which is the one you will be using) is for all humanoid characters. This setting allows Unity to retarget animations for you.

As you can see, Ethan is already set up properly as a humanoid. When you set a model as humanoid, Unity automatically goes through the process of mapping the rig for you. If you'd like to see how easy this is, you can simply change Animation Type to Generic, click Apply, and then change it back (which is exactly how this model was set up for you originally; no extra work was hidden). To see the work that Unity does for you, you can enter the rigging tool by clicking the Configure button (see Figure 18.4).

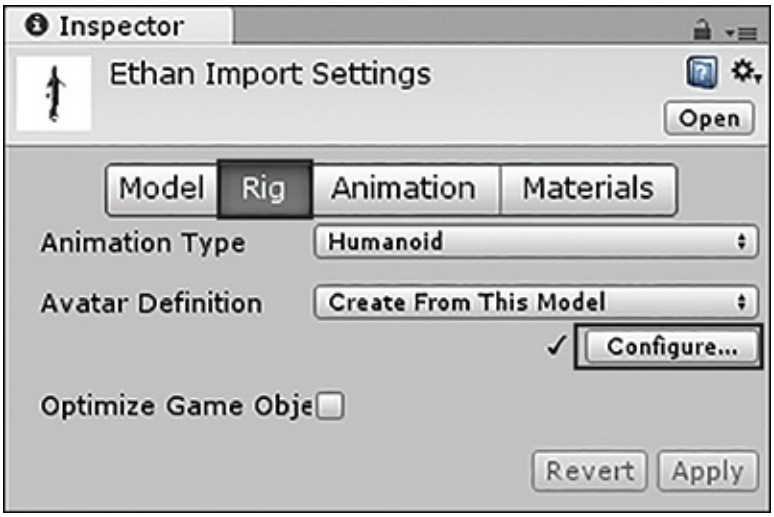

FIGURE 18.4
The rig settings.

TRY IT YOURSELF

Exploring How Ethan Is Rigged

In this exercise, you'll take a look at how Ethan is rigged. This will give

you a much more practical idea of how a rigged model is assembled. Follow these steps:

1. If you haven't already done so, create a new project and import the character assets from Standard Assets. Locate the Ethan.fbx asset and select it to display its import settings in the Inspector view, as described earlier in this hour.
2. Click **Configure** on the Rig tab. Doing so launches you into a new scene, so save your old one if prompted.
3. Rearrange your interface so that you can see mainly the Hierarchy and Inspector views. (Hour 1, "Introduction to Unity," covers how to close and move tabs.) You may want to save this layout. You can always go back to Default later.
4. With the Mapping tab selected, click on various green circles (see Figure 18.5). Notice how this highlights the corresponding child of EthanSkeleton in the Hierarchy view and puts a blue circle around the corresponding skeleton point below the outline. Notice all the extra points of the rig in the Hierarchy view. Those pieces aren't important for humanoids and thus aren't retargeted. Don't worry, though: They still play a part in ensuring that the model looks correct when moving.

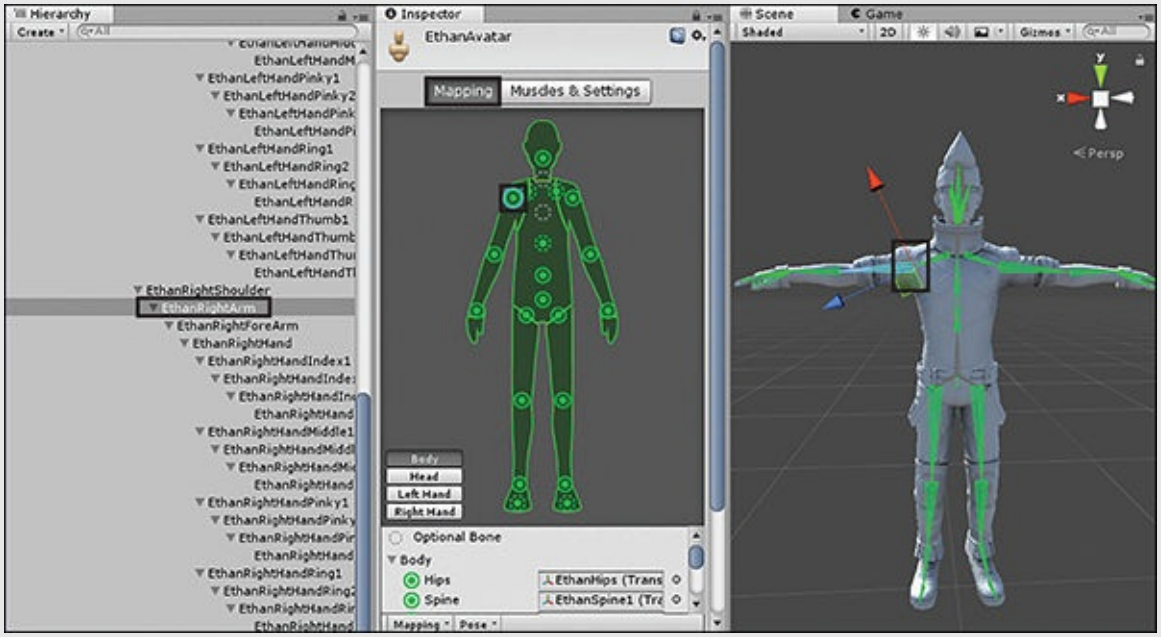

FIGURE 18.5
The Rigging view with the right arm selected.

5. Continue to explore other body parts by clicking **Body**, **Head**, **Left Hand**, and so on. These are all joints, which can be completely retargeted for any humanoid.

6. Click **Done** when you are finished. Notice that the temporary Ethan(Clone) disappears from the Hierarchy view.

At this point, you have previewed Ethan and seen how his skeleton is arranged. He is now ready to go!

Animation Preparation

For this hour, you could use the animations that came with Ethan, but that would be boring and wouldn't illustrate the flexibility of the Mecanim system. Instead, you are going to use some other animations provided in the book files for Hour 18. Each animation has options that control how the animation behaves that must be specifically configured the way you want them. For example, you need to ensure that the walking animation loops appropriately so that transitions don't have any obvious seams. This section walks you through preparing each animation.

Start by dragging the Animations folder from the book assets into the Unity editor. You will be working with four animations: Idle, WalkForwardStraight, WalkForwardTurnRight, and WalkForwardTurnLeft (though the Animations folder only contains three files; more on that soon). Each of these animations needs to be set up uniquely. If you look in the Animations folder, you will see that the animations are actually .fbx files. This is because the animations themselves are located inside their default models. Don't worry, though: You will be able to modify and extract them inside Unity.

Idle Animation

To set up the Idle animation, follow these steps (see Table 18.1 for an explanation of the settings):

1. Select the **Idles.fbx** file in the Animations folder. In the Inspector, select the **Rig** tab. Change the animation type to **Humanoid** and click **Apply**. This tells Unity that the animation is for a humanoid.

2. When the rig is configured, click the **Animations** tab in the Inspector. Set the starting frame to 128 and check the boxes next to **Loop Time** and

Loop Pose. In addition, check the box **Bake into Pose** for all the Root Transform properties. Ensure that your settings match the ones in Figure 18.6 and then click **Apply**.

3. To ensure that the animation is now properly configured, expand the **Idles.fbx** file (see Figure 18.7). Be sure to remember how to access that animation. (The model is irrelevant. It's the animation you want.)

TABLE 18.1 Important Animation Settings

Setting	Description
Loop Time	Indicates whether the animation loops.
Root Transform	Controls whether the animation is allowed to change an object's rotation, vertical position (y axis), and horizontal position (x/z plane).
Bake into Pose	Indicates whether the animation is allowed to actually move an object. If you check this box, the animation will not actually change the object but will just appear to.
Offset	Specifies some amount by which to modify the original position of the animation. For instance, by modifying the offset under Root Transform Rotation, you make small rotation changes to the model along the y axis. This is useful for correcting any motion error in the animation.

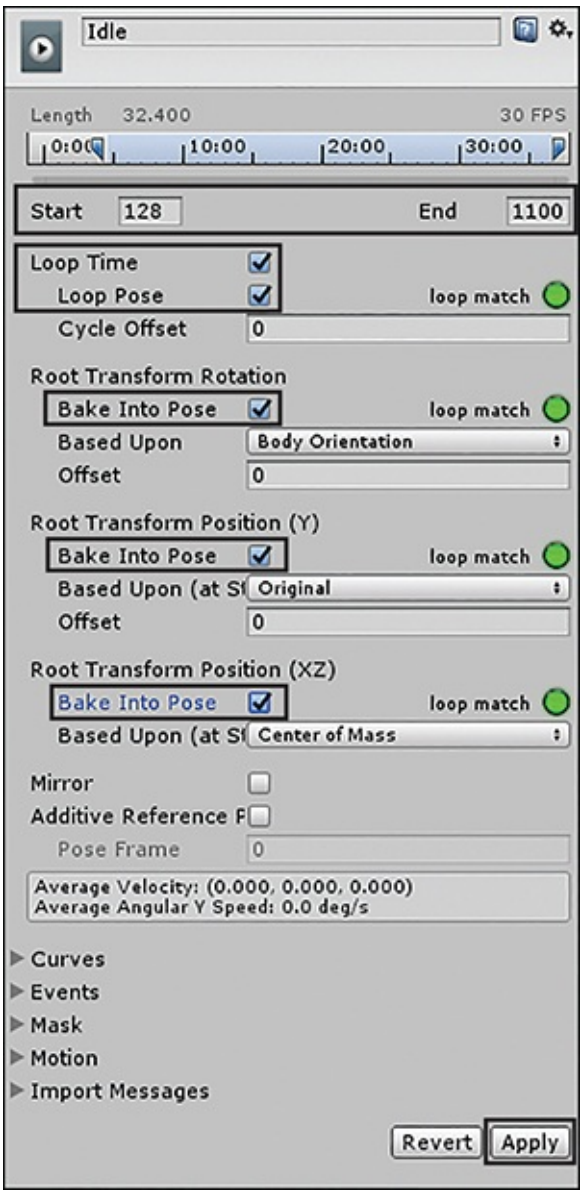

FIGURE 18.6
The Idle animation.

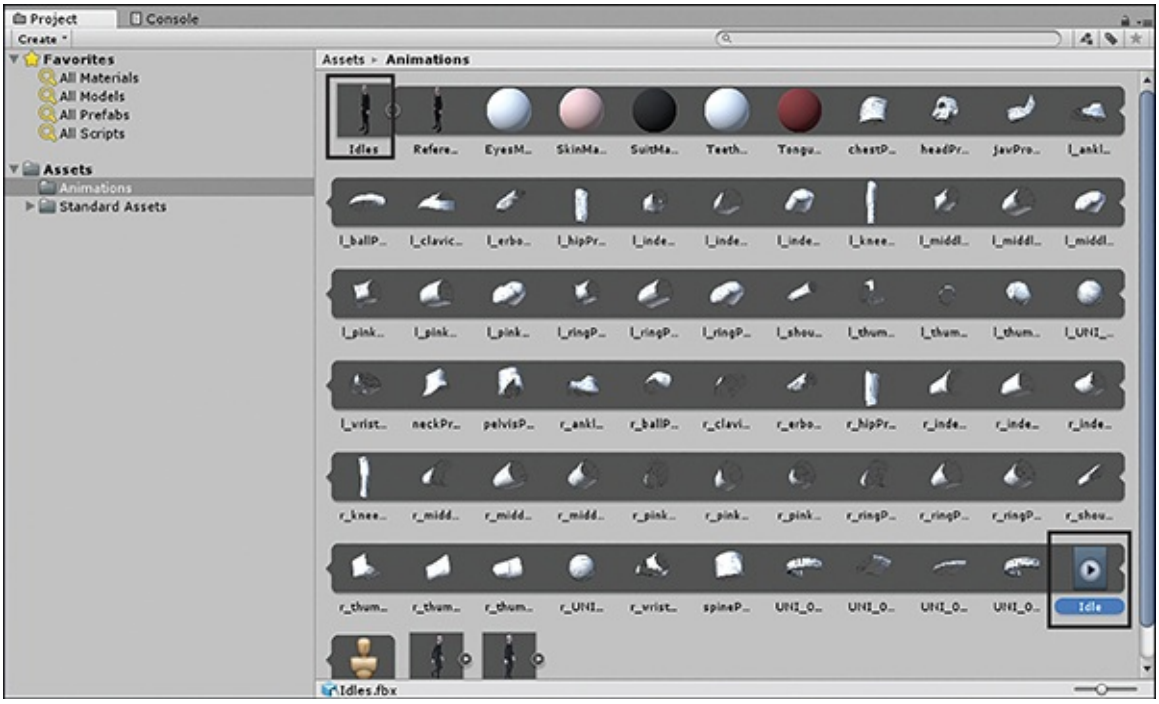

FIGURE 18.7
Finding the animation clip.

NOTE

Red Light, Green Light

You might have noticed the green circles present in the animation settings (refer to Figure 18.6). Those are nifty little tools you can use to designate whether your animations are lined up. The fact that the circles are green means that they will loop seamlessly. A circle that is yellow indicates that the animation comes close to looping seamlessly, but there is a minor difference that will create a bit of a seam. A red circle indicates that the beginning and end of the animation don't line up at all, and a seam will be very apparent. If you have an animation that doesn't line up, you can change the Start and End properties to find a segment of the animation that does.

WalkForwardStraight Animation

To set up the WalkForwardStraight animation, follow these steps:

1. Select the **WalkForward.fbx** file in the Animations folder and complete the rigging the same way you did for the Idle animation.
2. Change the clip's currently very generic name (Take 001) to

WalkForwardStraight by clicking the name and changing it (see Figure 18.8).

3. On the Animations tab, ensure that your settings are the same as the ones shown in Figure 18.8. You should note two things. First, Root Transform Position (XZ) has a red circle next to it. This is good; it means that at the end of the animation, the model is in a different x and z axis position. Because this is a walking animation, that is the behavior you want. The other thing you should notice is the Average Velocity indicator. You should notice a nonzero velocity in the x axis and z axis. The z axis velocity is good because you want the model moving forward, but the x axis velocity is a problem because it will cause the model to drift sideways while walking. You will adjust this setting in step 4.

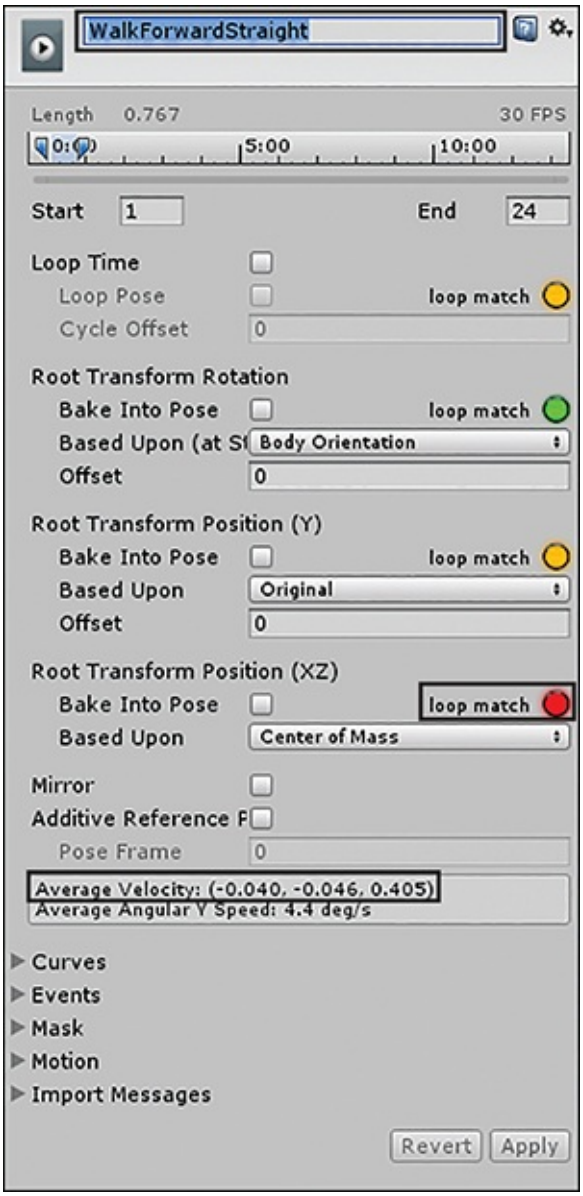

FIGURE 18.8
The WalkForwardStraight animation settings.

4. To adjust the x axis velocity, check the **Bake into Pose** check box for both the Root Transform Rotation and the Root Transform Position (Y) properties. Also change Root Transform Rotation Offset so that the x axis value of Average Velocity becomes **0**.

5. Set the End frame to 244.9 and the Start frame to 215.2 (in that order) so the animation only contains the walking frames.

6. Finally, check the **Loop Time** and **Loop Pose** check boxes.

7. Ensure that your settings match the final settings shown in Figure 18.9 and click the **Apply** button.

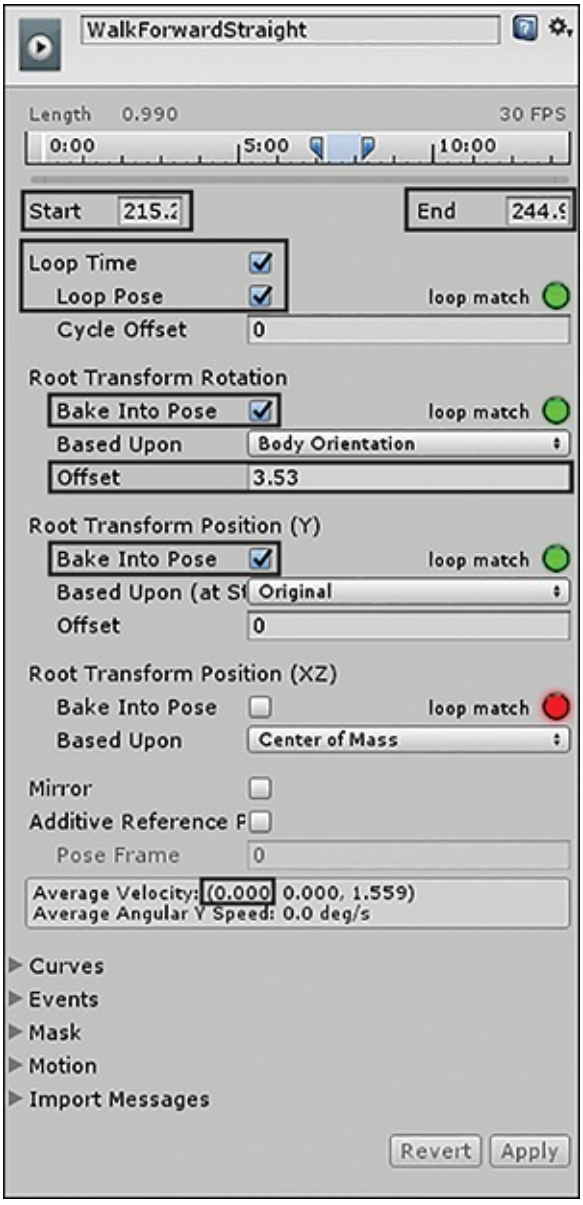

FIGURE 18.9
The fixed WalkForwardStraight animation settings.

WalkForwardTurnRight Animation

The WalkForwardTurnRight animation allows the model to smoothly change direction while walking forward. This one differs a little from the two you have already created in that you need to make two animations out of a single animation recording. This sounds trickier than it really is. Follow these steps:

1. Select the **WalkForwardTurns.fbx** file in the Animations folder and complete the rigging the same way you did for the Idle animation.
2. By default, there will be a long animation with the name *7*a_U1_M_P_WalkForwardTurnRight. Rename it by typing **WalkForwardTurnRight** into the Clip Name text field and pressing the **Enter** key.
3. With the WalkForwardTurnRight clip selected, set the properties to match those shown in Figure 18.10. The shorter Start and End times will cut the clip down and ensure that it only contains the model moving in a rightward circle. (Be sure to preview it to see what it looks like.) After you have done this, click **Apply**.

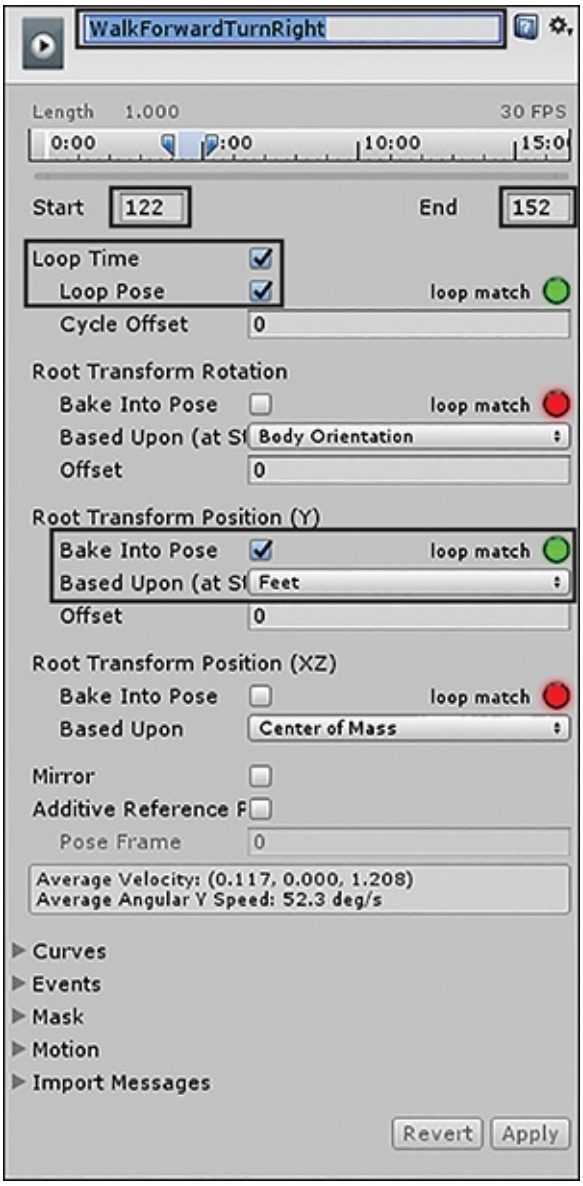

FIGURE 18.10
The WalkForwardTurnRight settings.

4. Create a WalkForwardTurnLeft animation clip by clicking the +icon in the Clips list (see Figure 18.11). The properties for the WalkForwardTurnLeft clip will be exactly the same as the WalkForwardTurnRight clip except that you need to put a check in the **Mirror** property (see Figure 18.11). Remember to click **Apply** when you're done with the settings.

At this point, all the animations are set up and ready to go. Now all that's left to do is build the animator.

FIGURE 18.11
Mirroring the animation.

Creating an Animator

Animators in Unity are assets. This means they are a part of a project and exist outside any one scene. This is nice because it allows for easy reuse over and over again. To add an animator to your project, in Project view you simply right-click a folder and select **Create > Animator Controller** (but don't do that just yet).

▼ TRY IT YOURSELF

Setting Up the Scene

In this exercise, you'll set up a scene and prepare for the rest of this hour. Be sure to save the scene created here because you'll need it later. Follow these steps:

1. If you have not done so already, create a new project and complete the model and animation preparation steps in the previous section.

2. Drag the Ethan model into your scene (from Assets\Standard Assets\Characters \ThirdPersonCharacter\Models) and give it the position (0, 0, -5).

3. Nest the Main Camera under Ethan (by dragging the Main Camera onto the Ethan game object in the Hierarchy view) and position the camera at (0, 1.5, -1.5) with a rotation of (20, 0, 0).

4. In your Project view, create a new folder named Animators. Right-click the new folder and select **Create > Animator Controller**. Name the animator **PlayerAnimator**. With Ethan selected in the scene, drag the animator onto the Controller property of the Animator component in the Inspector (see Figure 18.12).

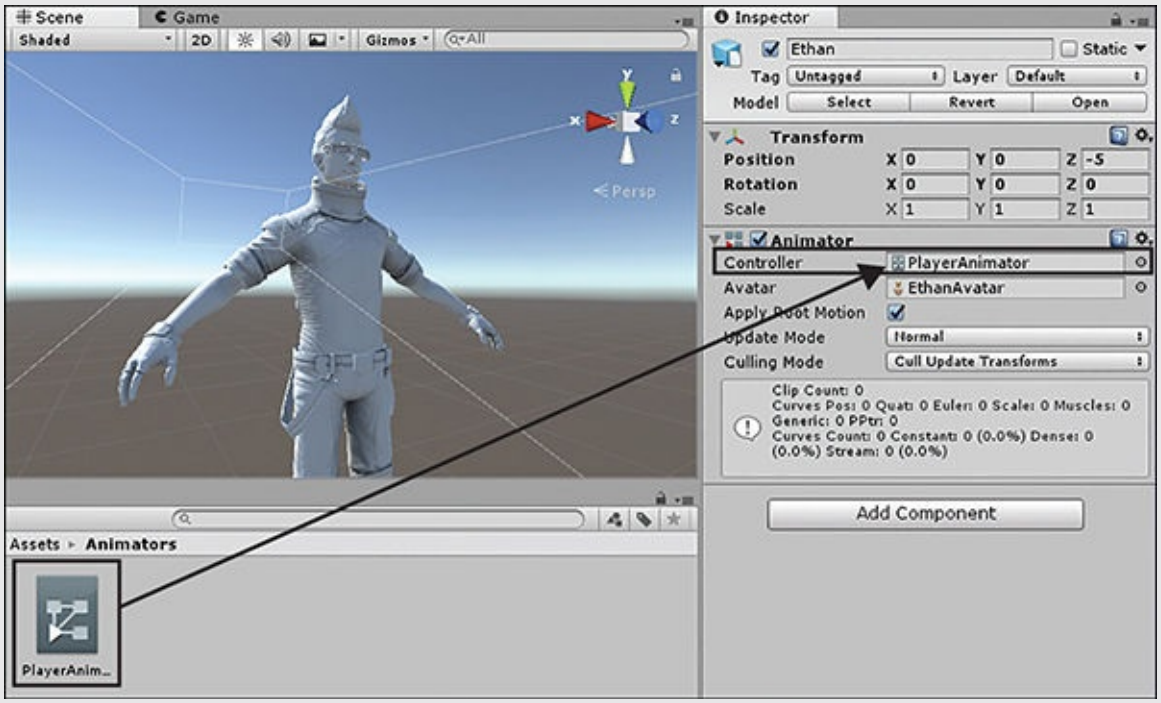

FIGURE 18.12
Adding the animator to the model.

5. Add a plane to your scene. Position the plane at (0, 0, -5) with a scale of (10, 1, 10).

6. Locate the file Checker.tga in the book assets for Hour 18 and import

it into your project. Create a new material named Checker and set **Checker.tga** as the albedo for the material (see Figure 18.13).

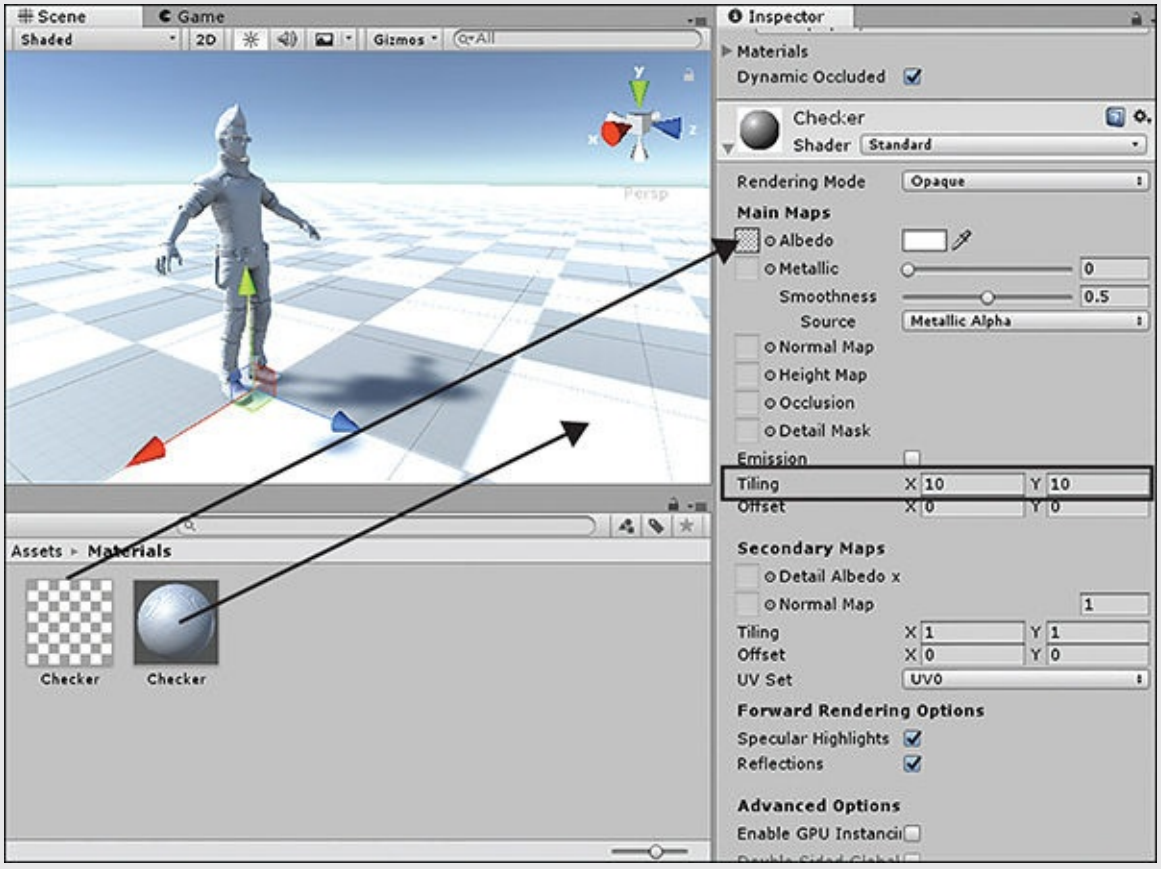

FIGURE 18.13
Setting the tiling of the checker texture.

7. Set both the X and the Y Tiling properties to **10** and apply the material to the plane. (The plane isn't super useful right now, but it will become important later in this hour.)

The Animator View

Double-clicking an animator brings up the Animator view (which you can also open by selecting **Window** > **Animator**). This view functions like a flow graph, allowing you to visually create animation paths and blending. This is the real power of the Mecanim system.

Figure 18.14 shows the basic Animator view. You can move around the Animator view by dragging with the middle mouse button held down, and you

can zoom with the scroll wheel. This new animator is very plain: It has only a base layer, no parameters, Entry and Exit nodes, and an Any State node. (These components are discussed in more detail later in this hour.)

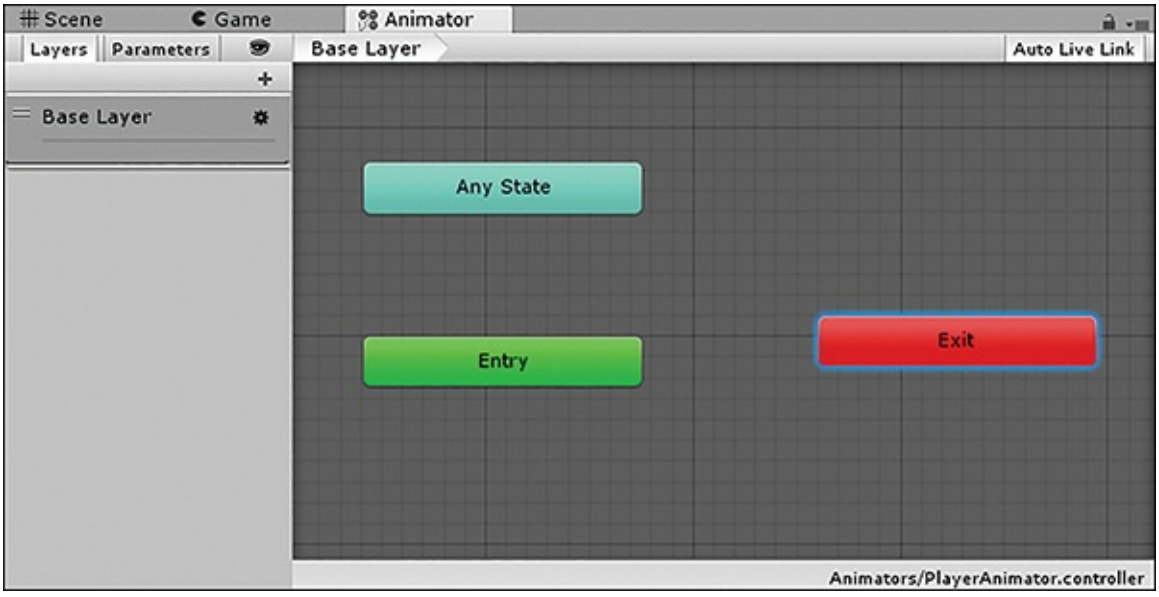

FIGURE 18.14
The Animator view.

The Idle Animation

The first animation you want to apply to Ethan is the Idle animation. You completed the long setup process earlier, and now, adding this animation is simple. You need to locate the Idle animation clip, which is stored inside the Idles.fbx file (refer to Figure 18.7, earlier in the hour), and drag it onto the animator in the Animator view (see Figure 18.15).

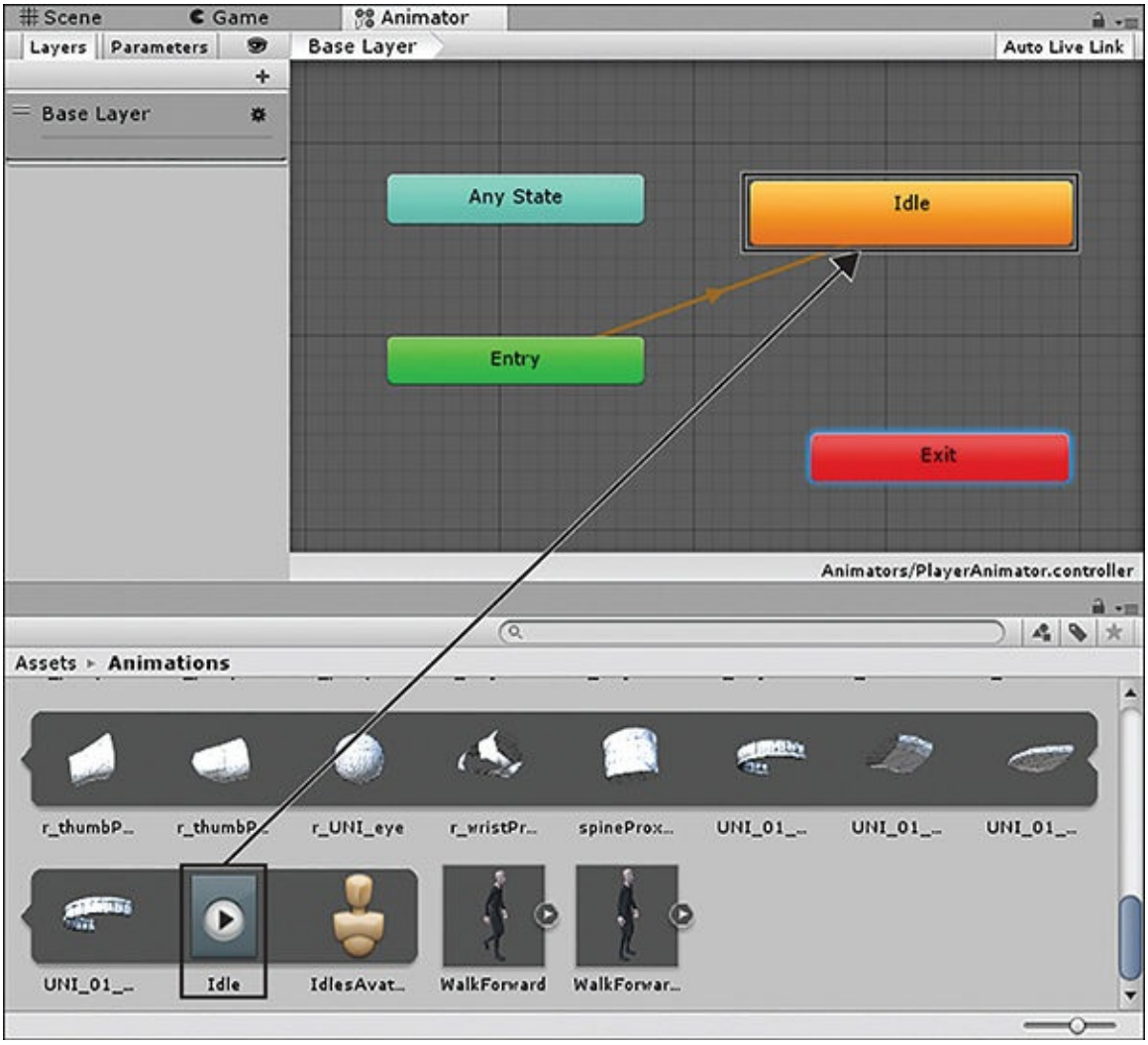

FIGURE 18.15
Applying the Idle animation.

You should now be able to run your scene and see the Ethan model looping through the Idle animation.

TIP

Slip and Slide

When you run the scene to see the Ethan model playing the idle animation, you may notice the model's feet sliding on the ground. This is due to how this particular animation was authored. In this animation, the character is determining its motion based on its hips as opposed to its feet (causing it to rotate a little bit like a pinwheel). You can fix this in the Animator view. Simply select the **Idle** animation state, and in the Inspector view, select the

Foot IK check box (see Figure 18.16). The model now attempts to track its feet to the ground. This causes the character to animate correctly, with its feet firmly planted. By the way, IK stands for Inverse Kinematics; the details of such are mostly out of the scope of this text.

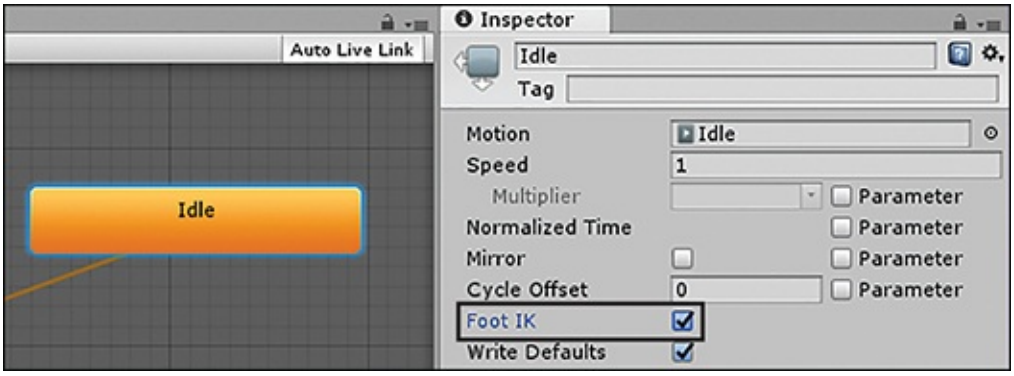

FIGURE 18.16
Selecting **Foot IK**.

Parameters

Parameters are like variables for an animator. You set them up in the Animator view and then manipulate them with scripts. These parameters control when animations are transitioned and blended. To create a parameter, simply click the + in the Parameters tab in the Animator view.

▼ TRY IT YOURSELF

Adding Parameters

In this exercise, you'll add two parameters. This exercise builds on the project and scene you have been working on thus far this hour. Follow these steps:

1. Make sure you've completed all the steps up to this point.
2. In the Animator view, click the **Parameters** tab on the left and then click the + to create a new parameter. Choose a Float parameter and name it **Speed** (see Figure 18.17).

FIGURE 18.17
Adding parameters.

3. Repeat step 2 to create a float parameter named Direction.

States and Blend Trees

Your next step is to create a new state. A state is essentially a status that the model is currently in that defines what animation is playing. You created a state earlier, when you added the Idle animation to the Animator controller. The model Ethan will have two states: Idle and Walking. Idle is already in place. Because the Walking state can be any of three animations, you want to create a state that uses a *blend tree*, which seamlessly blends one or more animations, based on some parameter. To create a new state, follow these steps:

1. Right-click a blank spot in the Animator view and select **Create State > From New Blend Tree**. In the Inspector view, name the new state **Walking** (see Figure 18.18).

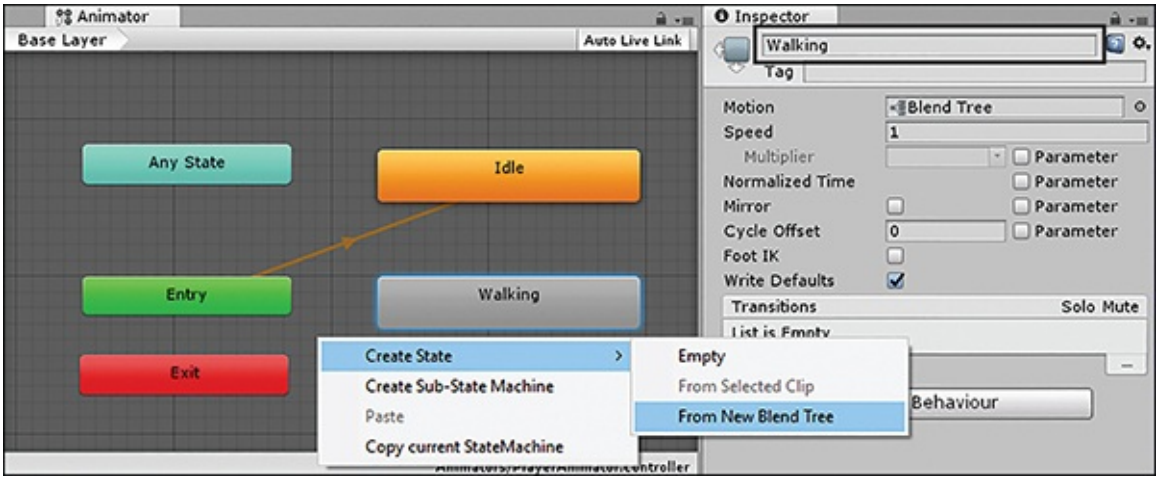

FIGURE 18.18
Creating and naming a new state.

2. Double-click the new state to expand it and select the new **Blend Tree** state. In the Inspector, click the Parameter property drop-down and change it to **Direction**. Then add three motions by clicking the + under the motions and selecting **Add Motion Field**. Under the graph, set the minimum value to **-1** and the maximum value to **1** (see Figure 18.19).

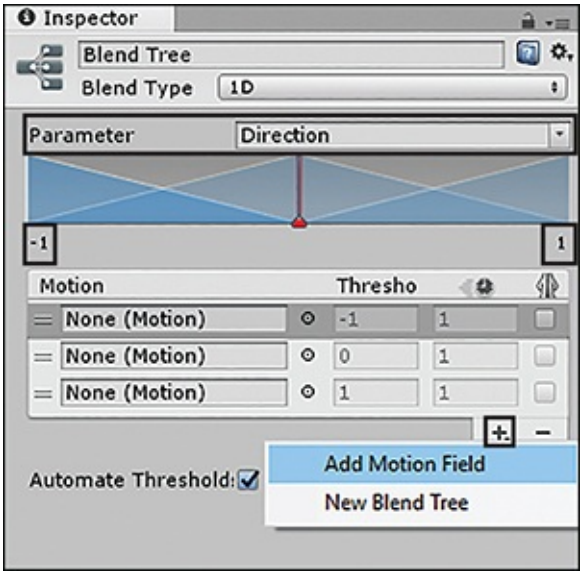

FIGURE 18.19
Adding motion fields.

3. Drag each of the three walking animations into one of the three motion fields, in this order: WalkForwardTurnLeft, WalkForwardStraight, WalkForwardTurnRight (see Figure 18.20). Remember that the turning animation clips are located under WalkForwardTurns.fbx, and the straight walking animation is under WalkForward.fbx.

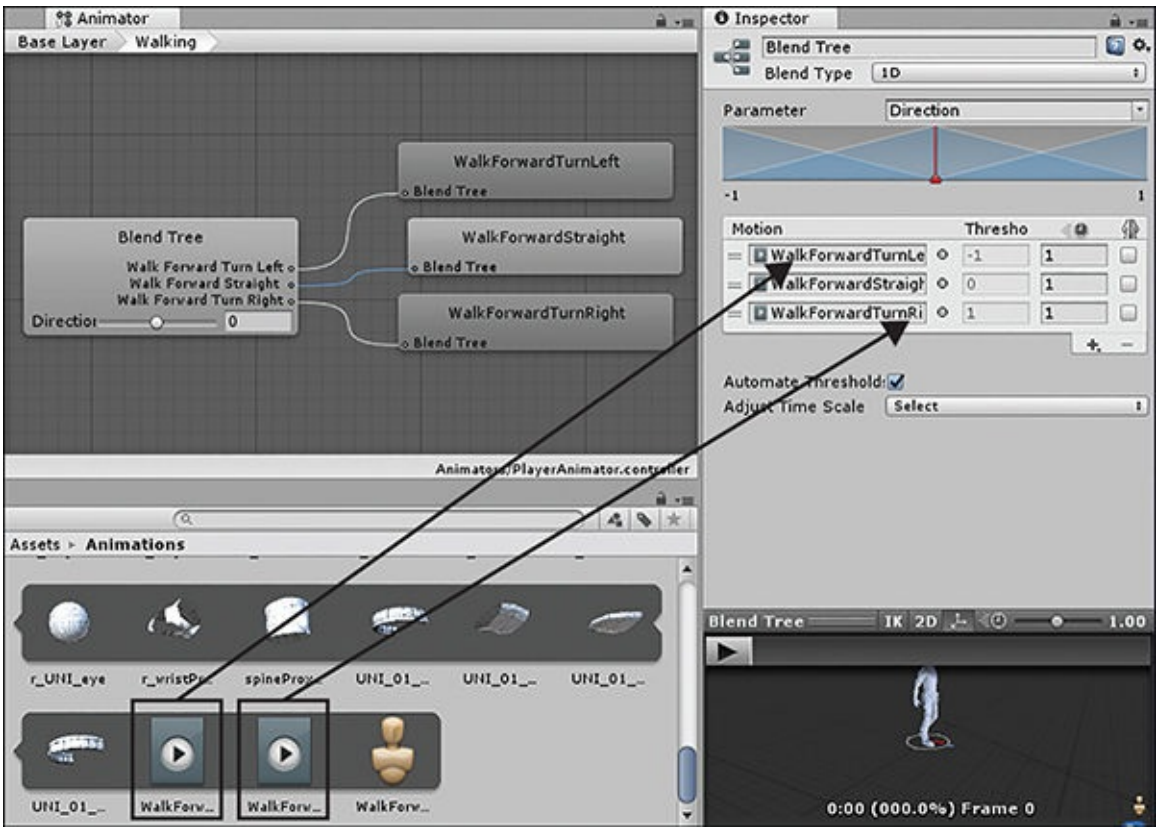

FIGURE 18.20
Changing minimum values and adding animations to a blend tree.

Your walking animation is now ready to blend, based on the Direction parameter. Basically, you told the blend state to evaluate the parameter Direction. Based on the value of that parameter, the blend tree will choose some percentage amount of each animation to blend to give you the final unique animation. For instance, if Direction equals -1, the blend tree plays 100% of the WalkForwardTurnLeft animation. If Direction equals .5, the blend tree plays 50% of the WalkForwardStraight animation blended with 50% of the WalkForwardTurnRight animation. You can easily see how powerful blend trees can be! In order to get out of the expanded view, click the **Base Layer** breadcrumb at the top of the Animator view (see Figure 18.21).

FIGURE 18.21
Navigating the Animator view.

Transitions

The last thing you need to do to ensure that your animator is finished is to tell the animator how to transition between the Idle and Walking animations. You need to set up two transitions. One of them transitions the animator from idle to walking, and the other transitions back. To create a transition, follow these steps:

1. Right-click the Idle state and select **Make Transition** to create a white line that follows your mouse. Click the **Walking** state to connect it to the Idle state.
2. Repeat step 1, except this time connect the Walking state to the Idle state.
3. Edit the Idle to Walking transition by clicking the white arrow on it. Add a condition and set it to be **Speed Greater** than the value **.1** (see Figure 18.22). Do the same for the Walking to Idle transition, except set the condition to **Speed Less Than** the value **.1**.
4. Uncheck the **Has Exit Time** box to allow the Idle animation to be interrupted when the walk key is pressed.

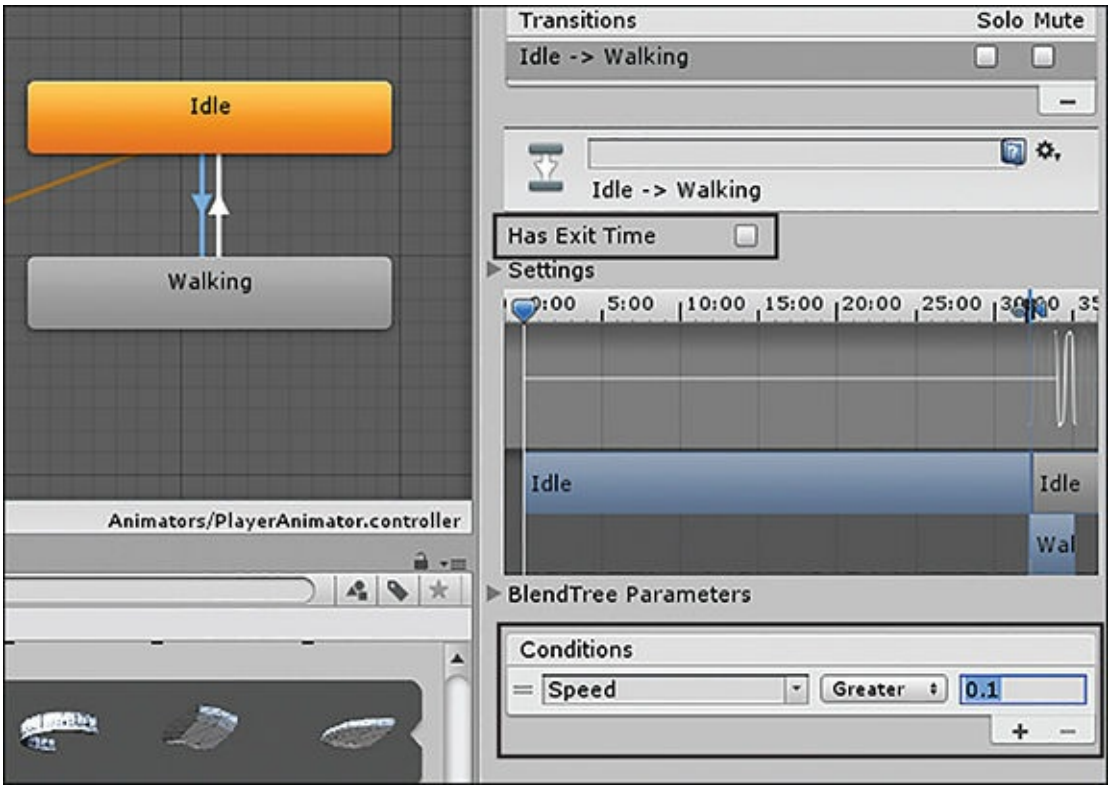

FIGURE 18.22
Modifying transitions

Modifying transitions.

The animator is finished. You might notice that when you run the scene, there aren't any working movement animations. This is because the speed and direction parameters are never changed. In the next section, you'll learn how to change these through scripting.

Scripting Animators

Now that everything has been set up with the model, the rigging, the animations, the animator, the transitions, and the blend tree, it is finally time to make the whole thing interactive. Luckily, the actual scripting components are simple. Most of the hard work was already done in the editor. At this point, all you need to do is manipulate the parameters you created in the animator to get Ethan up and running. Because the parameters you set up were of type `float`, you need to call the animator method:

```
SetFloat (<name> , <value>);
```

TRY IT YOURSELF

Adding the Final Touches

In the following steps, you will add a scripted component to the project you have been working on during this hour to make it all work:

1. Create a new folder called Scripts and add a new script to it. Name the script **AnimationControl**. Attach the script to the Ethan model in the scene. (This step is important!)
2. Add the following code to the AnimationControl script:

Click here to view code image

```
Animator anim;
void Start ()
{
    // Get a reference to the animator
    anim = GetComponent<Animator> ();
}
void Update ()
{
    anim.SetFloat ("Speed", Input.GetAxis ("Vertical"));
    anim.SetFloat ("Direction", Input.GetAxis("Horizontal"));
}
```

3. Run the scene and notice that the animations are controlled with the vertical and horizontal input axes. (If you've forgotten, the horizontal and vertical input axes are the WASD keys and the arrow keys.)

That's it! If you run your scene after adding this script, you might notice something strange. Not only does Ethan animate through idle, walking, and turning, but the model also moves. This is due to two factors.

The first factor is that the animations chosen have built-in movement to them. This was done by the animators outside Unity. If this hadn't been done, you would have to program the movement yourself.

The second factor is that by default, the animator allows the animation to move the model. This can be changed by selecting the Apply Root Motion property of the Animator component (see Figure 18.23), but in this case it will cause some strange effects!

The final project file is included in the book files in case you want to compare notes.

FIGURE 18.23
Root motion animator property.

Summary

You started this hour by building a very simple animation from scratch. You started with a cube and added an Animator component. You then created an Animator controller and linked it to the animator. Next, you created animation states and corresponding motion clips. Finally, you learned how to make the states blend.

Q&A

Q. Can you do key frame animations on humanoids in Unity?

A. Unity doesn't allow key frame animation on humanoids, and although you

may find workarounds on the Internet, you are better off creating humanoid animations in a dedicated 3D package and importing them into Unity.

Q. Can an object be moved by both the animator and the physics engine?

A. Although this is possible with some care, generally you want to avoid trying to mix the two. At least at any one time, you want to be clear on whether the animator or the physics engine is controlling a game object.

Workshop

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

Quiz

1. To what must an Animator component have a reference in order to work?
2. What color is the default animation state in the Animator tab?
3. How many motion clips (motions) can an animation state have?
4. What do you use to trigger animation transitions from script?

Answers

1. An Animator controller must be created and connected to the Animator component. In the case of humanoid characters, an avatar is also required.
2. Orange
3. It depends; an animation state can be a single clip, a blend tree, or another state machine.
4. Animator parameters

Exercise

A lot of information is required to produce a robust and high-quality animation system. In this hour, you got to see one way and one group of settings to achieve this. Plenty of other assets are available, however, and learning is paramount to success.

Your exercise for this hour is to continue studying the Mecanim system. Be sure

to start by browsing Unity's documentation on the system. You can find it on Unity's website at https://docs.unity3d.com/Manual/AnimationSection.html.

You can also explore the fully animated Ethan prefab, found at Assets\Standard Assets \Characters\ThirdPersonCharacter\Prefabs\ThirdPersonController.prefab. This prefab has a much more complex animator than the Ethan you worked with in this hour, with three blend trees for Airborne, Grounded, and Crouching states.