Unity Animation Essentials

Bring your characters to life with the latest features of Unity and Mecanim

Alan Thorn
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 'Sprite Animation'
- A synopsis of the book’s content
- More information on Unity Animation Essentials
About the Author

Alan Thorn is a freelance game developer and author, with over 13 years of industry experience. He founded Wax Lyrical Games and is the creator of the award-winning game Baron Wittard: Nemesis of Ragnarok. He is also the author of over 10 video training courses and 15 books on game development, including Mastering Unity Scripting, How to Cheat in Unity 5, and UDK Game Development. Besides this, he is a visiting lecturer for the master's degree course in game design and development at the National Film and Television School, London.

Alan has worked as a freelancer on over 500 projects, including games, simulators, kiosks, serious games, and augmented-reality software for game studios, museums, and theme parks worldwide. He is currently working on two game projects. He also enjoys graphics, philosophy, yoga, and hiking. More information about him can be found at http://www.alanthorn.net.
Preface

There's no getting around it! Animation plays a crucial role almost everywhere in games, from a simple scenario, such as a moving spaceship, to a complex scenario, such as facial animation. You simply cannot build a complete game and a truly interactive experience without some degree of animation. In this book, we'll take a step-by-step journey through the extensive feature set offered by Unity for creating real-time animations. We'll approach the subject from the ground up. This book assumes no prior knowledge about animation or experience in animation in Unity. However, it does assume basic familiarity with the Unity engine in terms of level design, basic interface usage, and some fundamental knowledge of C# coding. Using only this as the starting point, it proceeds to break down the bewildering range of animation features and considers their application in real-world scenarios, helping you achieve tangible results quickly and easily. Now, let's see what's ahead of us...

What this book covers

Chapter 1, Animation Fundamentals, begins by considering animation formally as a concept—animation at its core. From there, this chapter considers miscellaneous and very practical cases of animation in Unity that specifically involve scripting. This includes working with deltaTime and animation curves and even animating Mesh UVs. Overall, this chapter marks the beginning of our journey through animation and features everything you need to know to move ahead.

Chapter 2, Sprite Animation, explores the world of 2D animation, examining the sprite feature set used to create 2D games, flip-book textures, and planar animation. We consider the Unity Sprite editor, animation frames, frame rates, and how to fix common problems.
Chapter 3, Native Animation, enables us to see animation more generally in Unity, for both 2D and 3D, using both the Animation window and Particle Systems. We look at two very practical and useful projects. First, we create a camera fly-through across a level. Then, we create a dust/firefly particle system, which is very commonly found in fantasy games.

Chapter 4, Noncharacter Animation with Mecanim, teaches you about a flagship feature of Unity, namely Mecanim. Mecanim refers to a collection of features that together make advanced and smooth animations possible, especially for characters. In this chapter, we see the more unconventional uses of Mecanim as we animate doors and other interactive elements in our levels.

Chapter 5, Character Animation Fundamentals, begins with an analysis of rigged character animation. In this chapter, we see how to animate humanoid skeletons and characters for real-time animation. We also consider how to import rigged characters and configure them optimally, ready for animation.

Chapter 6, Advanced Character Animation, follows logically from the previous chapter. Here, we take an imported and configured character in Unity, and then we animate it as a player-controlled character. This character will support idle and forward movement, for both walking and running, as well as turning animations for moving left and right.

Chapter 7, Blend Shapes, IK, and Movie Textures, marks the end of this book. This chapter covers the following topics: Blend Shapes, used to create facial animations and other morph-like motions; inverse kinematics, used to accurately position a character's feet and hands in real time; and movie textures, used to play movie files as a texture projected onto a geometry.
In this chapter, we enter the world of 2D game animation in Unity—specifically, sprite animation and its associated feature set, which is extensive. If you’re planning on making 2D games, such as side-scrolling platformers or casual puzzle games for mobiles, or if you’re creating animated GUIs and menu systems, then 2D animation will probably be highly important for your project.

Before getting started, however, let me elaborate further on the meaning of 2D, as opposed to 3D animation, to clarify the scope of this chapter and the kind of animation we are concerned with here.
Sprite Animation

By 2D (short for two-dimensional), I mean only a specific kind of presentation, that is, any scene presented using either an orthographic camera or a camera showing objects in a planar or flat way, not allowing the user to rotate their view and observe events from other angles. In this sense, even a 3D scene may be part of a 2D game because 2D relates only to a scene's mode of display through cameras and not to any intrinsic properties of the scene. Even so, most of our 2D work will be related to sprite objects, a special 2D game object type supported by Unity. So, let's get started. Our aim in this chapter will be to create an animated sprite character that runs through the level, playing a run animation as he moves.

Sprites – importing and configuration

A **sprite** is a dedicated 2D object type in Unity. It lets you import a flat, regular texture for displaying a scene as if it were a cardboard prop or billboard. Sprites are useful for animated characters and props, such as the playable character in a side-scrolling platformer. 2D sprite animation works much like traditional **flip-book** animation, where key frame sketches are drawn on the pages of a book, one frame per page, and are played back to create animation by quickly turning the pages and observing the effect. There are two methods of importing and configuring animated sprites for your game, and this section covers both of them. The sprite objects used here are included in this book's companion files in the Chapter02/assets folder. So, open the assets and follow along with me. Let's start by considering two methods of importing sprites: importing individual sprites and importing a sprite atlas.

More information about sprite objects can be found in the online Unity documentation at [http://docs.unity3d.com/ScriptReference/Sprite.html](http://docs.unity3d.com/ScriptReference/Sprite.html).

**Individual sprites**

If you've created an animated sprite using separate image files for each frame of animation, you can simply drag and drop all the frames at once into the Unity Project panel for importing.
When you do this, each file is imported as a regular texture.
Sprite Animation

To convert them into sprites, select all the imported textures in the Project panel. From the Object Inspector, choose the Sprite option for the Texture Type setting. Make sure that Sprite Mode is set to Single, as each separate image represents a unique frame for the same sprite overall (that is, the same character). In addition, the Generate Mip Maps setting should be disabled so as to improve texture quality. Finally, the Pivot option for the sprite should be set to Bottom, the location of the feet of most character sprites. This is because the feet are where the character meets the ground plane and is effectively positioned within the scene. Afterwards, click on Apply to confirm.

Remember that you don't have to apply sprite settings to each texture individually. That would be tedious! Instead, you can select multiple textures in the Project panel and apply settings to all of them as a batch.

Configuring imported sprite textures
And that's it! The imported frames are now configured and ready for sprite animation in the scene. We'll see how to do that specifically later in this chapter. Next, let's see how to import a slightly different kind of sprite.

**The sprite atlas**

Sometimes, your sprites may already be compiled together into a single sprite sheet, called a **texture atlas**. In other words, all the frames in an animation may be bundled together into regular rows and columns inside a single texture file, as opposed to many separate files. These kinds of sprite sheets are also supported by Unity, but they require some additional setup and tweaking at the import stage.

A sprite atlas texture
**Sprite Animation**

To import the sprite atlas, simply drag and drop the texture into the Unity Project panel. It'll be imported as a regular texture file. This will be shown inside the Project panel.

By default, sprite atlas textures are imported as regular textures.

After the import, you'll need to configure the texture properly to work as a sprite. To do this, select the texture in the Project panel. From the Object Inspector, choose **Sprite** for the **Texture Type** option. Then choose **Multiple** for the **Sprite Mode** option. This is because the texture file contains multiple frames, not just one. Disable the **Generate Mip Maps** checkbox for optimal texture quality and then click on **Apply**.

Preparing the atlas texture in the Object Inspector
Choosing **Multiple** for the **Sprite Mode** option signals that multiple frames exist within the texture, but Unity still doesn't know precisely where they are in the texture. To define this, the **Sprite Editor** can be used. This tool is accessed by clicking on the **Sprite Editor** button from the Object Inspector when the texture is selected in the Project panel. From the **Sprite Editor** dialog, you can manually draw the boundaries for each sprite by simply clicking and dragging a box around each one in the image. Alternatively, you can automatically generate sprite slices from a tiling pattern. For the robot texture provided with this book's companion files, each sprite is in a 512 x 512 pixel grid. To generate slices for this, click on the **Slice** button in the top-left corner of the **Sprite Editor** window.

![Sprite Editor](image.png)

Click on the slice button in the Sprite Editor to generate slices from the atlas.
After clicking on the **Slice** button, a configuration popup appears. Fill in the slice parameters to generate slices for the frames in the atlas. For the robot texture provided here, the **Slice** type should be set to **Grid** (instead of **Automatic**), since the sprites are arranged in a grid inside the image. **Pixel Size** should be **512 x 512**, and **Pivot** should be set to **Bottom** to position the pivot at the character's feet.

![Sprite Editor](image)

Generating slices from a grid

After clicking on the **Slice** button from the **Slice** popup, Unity will divide the texture atlas into separate sprites, each of which is surrounded by a selectable border. If required, you can select each slice and change its settings. However, for the robot texture, the default arrangement can be accepted, and the **Apply** button can be clicked on from the top toolbar in the **Sprite Editor** window.

![Sprite Editor](image)

The slices are now created
Clicking on **Apply** from the **Sprite Editor** window generates a sequence of sprite objects, which appear in the Project panel as independent entities and are now ready for animation.

**Animation with sprites**

After you’ve imported a set of sprite objects, either as individual images or as atlas textures, you’re ready to start animating with them using a single technique. Achieving this is really simple, at least initially. From the Unity Project panel, just select all sprites that go together as a complete animation sequence, and then drag and drop them as one into the Scene Hierarchy panel. Dropping them into the scene or game tabs won’t work; it must be inside the hierarchy panel. For example, drag and drop the player run animation sequence (from this book’s files) into the scene. When you do this, Unity displays a **Save** dialog, prompting you to create a new animation sequence asset (`.anim` is the file extension).
Assign the animation a name (such as PlayerRun.anim) and then click on Save.

A sprite object is created in the scene, and it should be visible in the Scene and Game tabs. If it's not, make sure that the object is positioned within the view of the camera. And that's all there is to creating an initial animation sequence. If you test your scene, your sprite character will now be animated, running through all the images in the sequence. This is achieved because Unity has performed a series of configuration steps behind the scenes. Specifically, Unity created a new Animation Clip (.anim) asset that defines the key frame sequence. Secondly, Unity has created a Mecanim controller to initiate an animation playback as the level begins, and to control the playback speed. Thirdly, Unity has added an animator component to the sprite object in the scene to link the object to its animation data. By default, however, the animation created may not appear exactly as you want. So, in the subsequent sections, we'll see how we can tweak the animation.
Sprite animation is too slow or too fast

If your sprite animation is playing too fast or too slow, then you'll need to edit the sprite Mecanim graph (Mecanim is covered in depth in later chapters). The animation speed for sprites can be changed quickly and easily. To achieve this, select the sprite object in the scene. From the Object Inspector, double-click on the Animation Controller asset inside the Controller slot of the Animator component.
Double-clicking on the sprite Animation Controller will open the Mecanim graph for the sprite. Here, the animation speed can be controlled. This graph contains several nodes connected together.

From the graph, click on the PlayerRun (default) node, which represents the sprite animation, to select it and view its properties in the Object Inspector. For the player sprite graphic (included in this book's companion files), this node represents a full walk animation. The animation speed is controlled using the Speed setting. In it, 0 means stopped, 1 means the default speed, 0.5 means half speed, 2 means double speed, and so on. If your animation is too slow, then increase the Speed value, and if it is too fast, then reduce the Speed value. For my player character animation, I set the speed to 2 double speed. Once you're done, just replay your game to see the effect.
Chapter 2

Changing the animation speed

**Animation shouldn't be looping**
By default, sprite animations play on a loop cycle; that is, they play again and again without an end. When the animation playback completes a cycle, it simply returns to the beginning and plays again. This may be exactly what you need in some cases, but not always. Sometimes, you may want the animation to play only once and then stop. If so, you’ll need to access the animation data (inside the `.anim` asset) and adjust its properties. To do this, select the sprite animation asset in the Project panel. Animation assets feature a **Play** icon and match the name you gave the animation during its creation.
After the asset is selected, remove the checkmark from the **Loop Time** checkbox inside the Object Inspector. Now replay your game and the animation will play only once.

Disabling the loop time for play-once animations

**Frames play in the wrong order**

If your sprite animation features many frames, it's possible that Unity arranged them in the wrong order while generating the animation, resulting in some frames appearing before or after they should. If this happens, you'll need to edit the animation data itself, through the **Animation** window. This window can be accessed by going to **Window** | **Animation** from the application menu.
With the **Animation** window visible in the editor, select the sprite object in the scene, and its animation data will show up automatically in the timeline. The timeline represents the total duration of the animation, from start to end. The diamond-shaped symbols dotted evenly along the timeline represent key frames, in which the sprite image changes. You can select a specific key frame in the timeline by clicking on it. Then, you can also view its properties for the sprite in the Object Inspector. For example, with the player character animation, at time equal to **0:06**, the `player_run_00013` sprite is shown. The **Sprite** field in the Object Inspector is highlighted in red to show that the field is animated and changes over time.

![Viewing key frames for sprite animation](image)
When you select a key frame from the timeline in which the wrong image is shown for the sprite at that time, you can easily correct it by just clicking on the **Sprite** field in the Object Inspector and picking a new sprite object from the sprite browser. Unity will automatically record, or understand, that the newly selected sprite should be displayed for that key frame instead.

![Select Sprite](image)

Using the sprite browser to correct animation frames

---

**Summary**

This chapter covered Unity's extensive 2D feature set used to create animated sprites. Sprites can be imported on a frame-by-frame basis from separate files, or as a texture atlas containing many frames together. In either case, frames can easily be assembled into an animated sequence, and Unity's animation tools (such as Mecanim and the **Animation** window) can be used to repair the most common issues, such as those concerning animation speed, looping, and misplaced frames.

Using the assets provided with this chapter (including the player character asset), you should now be able to quickly and effectively create an animated character playing a run animation. In addition, you can use the object Mover script created in the previous chapter to move the animated sprite object along an axis in the scene, while also playing its run animation to simulate a complete run cycle. A completed project that does this is available in the companion files in the `Chapter02\AnimatedChar` folder. In the next chapter, the Unity **Animation** window, as well as other tools, will be explored in more detail.
Where to buy this book
You can buy Unity Animation Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.