COMP1150/MMCC1011 Week 4 Prac

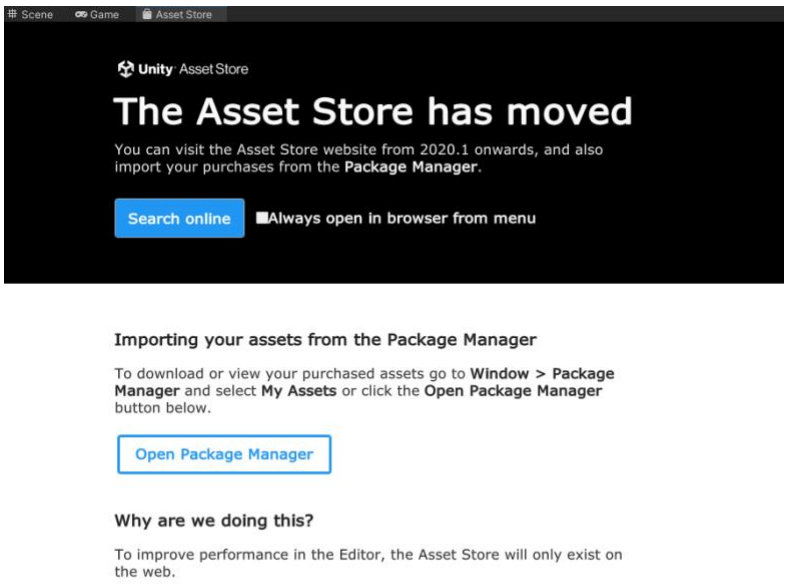
Topics covered:

- The Unity Asset Store
- Sounds
- Editing a prefab
- Sprite Animation
- Movement Animation
- Animation Curves
- Animation State Machines

This practical will focus on making your game a bit more vibrant and exciting. We will be adding sound effects and music, as well as creating animated objects in your world, allowing things more things to move and making your scene come alive. Please continue to build out your platformer scene from the same '2d-prac' Unity project (originally cloned from GitHub Classroom in prac 2) that you have used over the last couple of weeks.

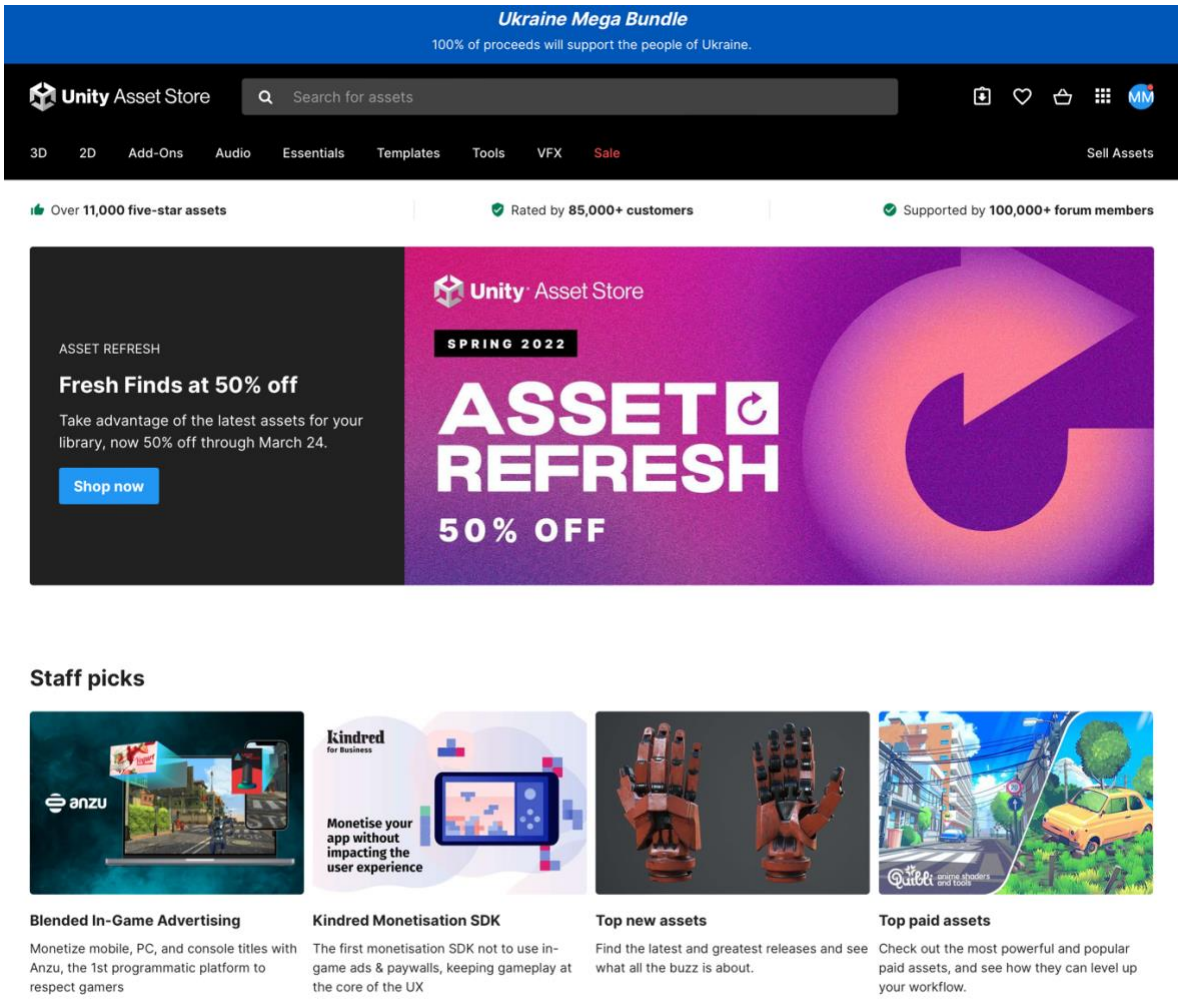
The Unity Asset Store

We're going to add a sound effect to indicate when the player picks up a coin, but first we'll need an appropriate sound. A good place to get assets is the official Unity Asset Store, which you can access in your browser at https://assetstore.unity.com/ (it's also linked from within the Unity Editor via **Window > Asset Store**, which will open a separate panel explaining the asset store has moved and to click the **Search online** button to access the store).



For overseas students: Firewall settings may prevent some students from accessing the Unity Asset store. You may have to toggle your VPN on or off, or find a sound effect from a different source such as https://freesound.org/ Keep in mind though, when using a third-party asset (e.g. a sound effect taken from a website) you must have a valid license to use it!

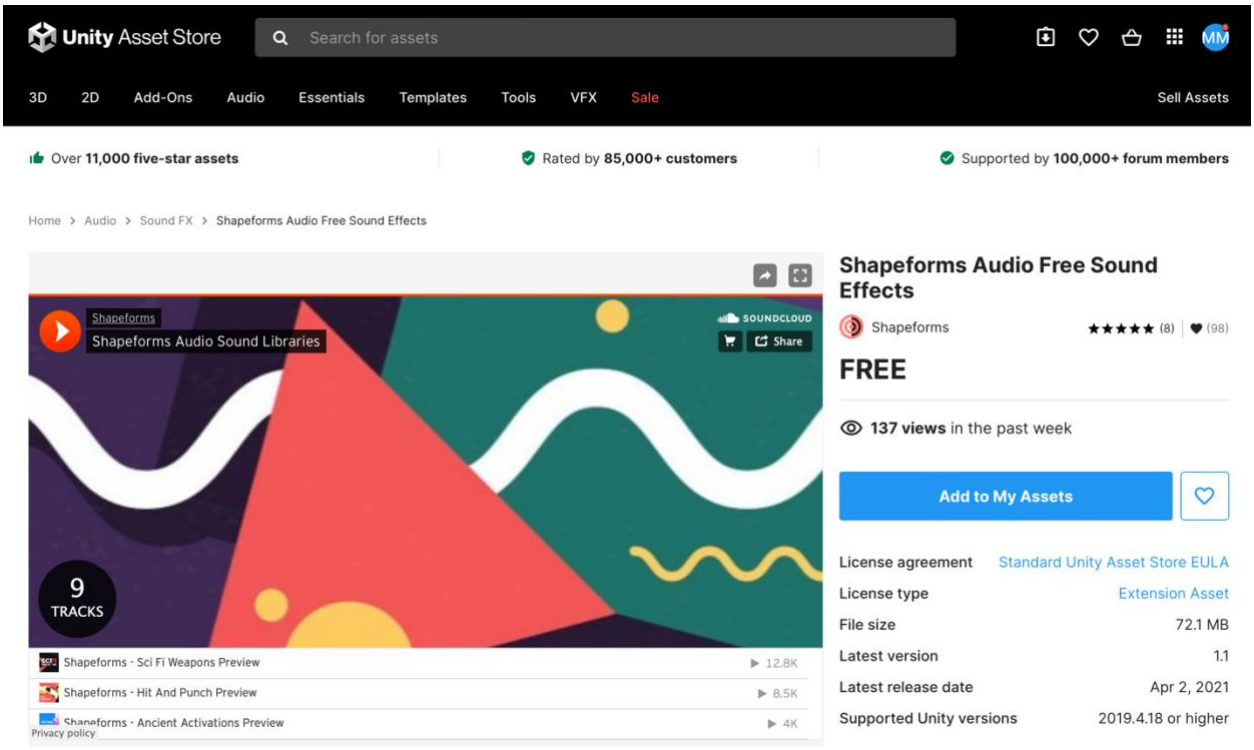
Once you arrive to the Unity Asset Store homepage, first make sure you are signed into the same Unity account as Unity Hub (via the profile icon in the top right corner of the Asset Store).



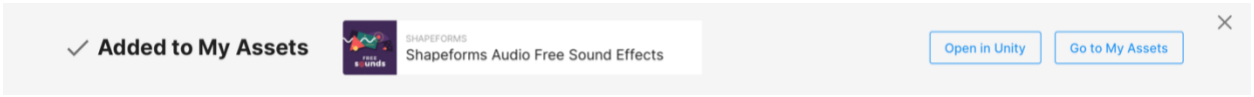
Next, search for “coin sound” via the search bar at the top of the screen. Refine the results by clicking the **Free Assets** checkbox under the ‘Pricing’ filter on the right of the page.

View the page for the asset called ‘Shapeforms Audio Free Sound Effects’, which should appear in the search results. Select the blue **Add to My Assets** button, and review and **Accept** the ‘Asset Store Terms of Service and EULA’ (or end user license agreement). This is similar to many other required legal agreements for use of products and services (e.g. apps on the App Store or Play Store), however also includes additional agreements for ensuring you comply with the licenses for acquired/purchased assets.

Advice: The Asset Store is a great resource for Unity developers, and there are a lot of free assets available for developers to use and play around with. There are also several dedicated developers and companies that make money by creating their own assets and selling them on the Unity Store, so it's certain that you can find some quality products on here.

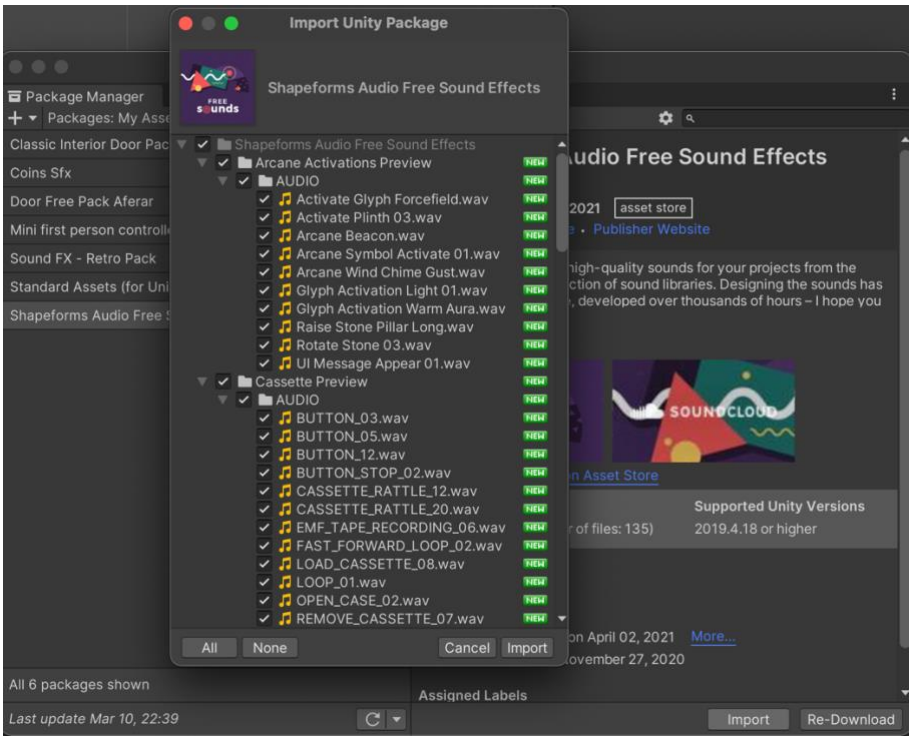


A popup will appear at the top of the page, confirming the asset has been added to the list of assets associated with your account, and giving you the options to view that list or open it.



Click **Open in Unity** on that popup or via the changed blue button on the refreshed asset's page to open the asset in the 'Package Manager' in Unity. You can access all of your assets 'purchased' from the Unity Asset Store in the Package Manager in the Unity Editor (also accessible via **Window > Package Manager**).

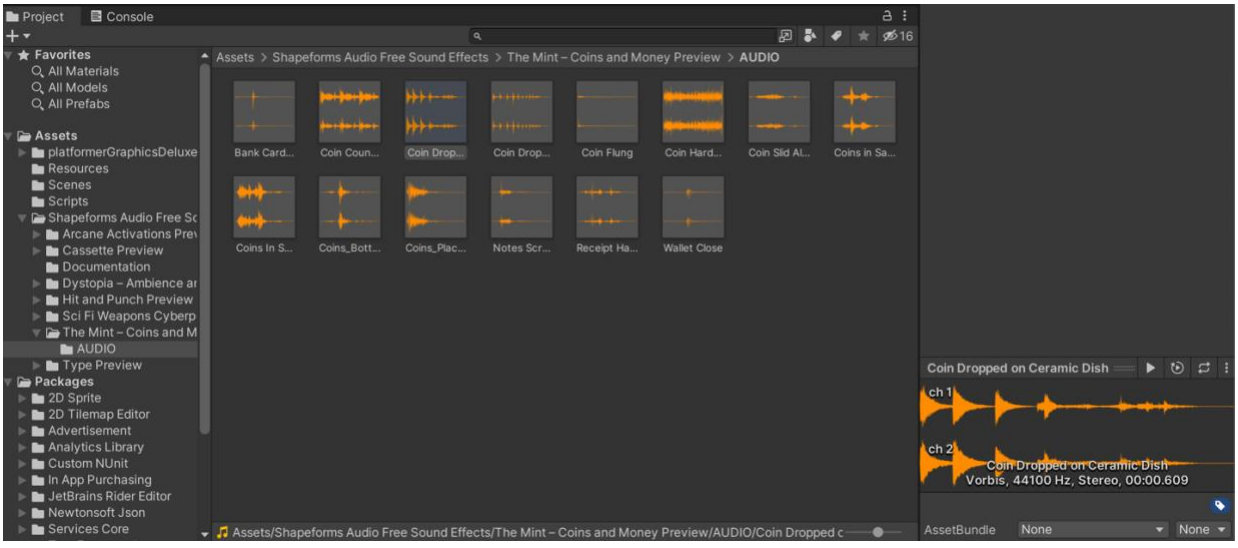
In the Package Manager window in Unity, click the **Download** button to download the 'Shapeforms Audio Free Sound Effects' package, and then once complete click the **Import** button that appears in its place.



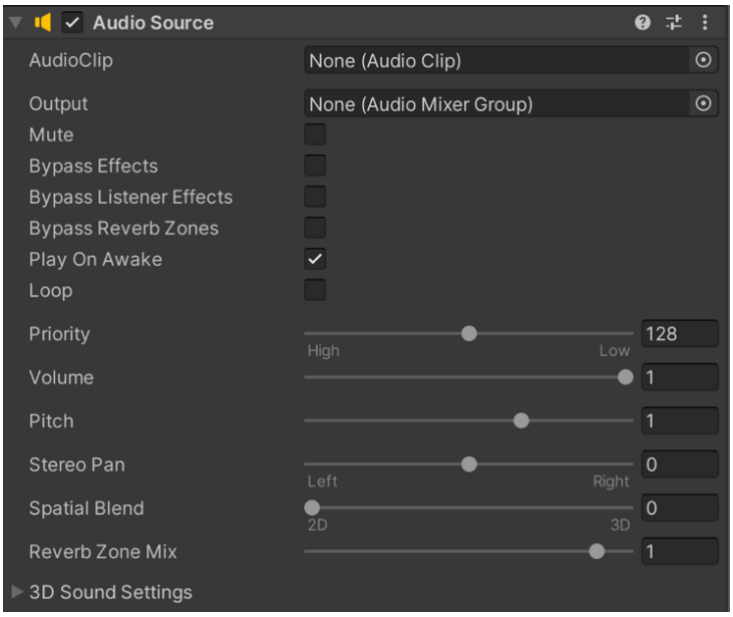
The now familiar package import popup window should appear, with all the files automatically ticked. Press **Import** again to add them into your project, which will place them in a new folder within the Assets folder (in the Project panel).

Adding Sound

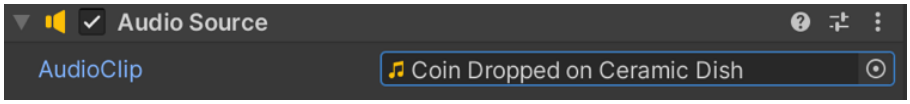
Note: Please use headphones for this if you are in the lab, to avoid disturbing other students. Find the 'AUDIO' subfolder within the 'The Mint – Coins and Money Preview' subfolder and try out some of the included coin sounds by clicking on them and then pressing play on the audio wave view that appears at the bottom of the inspector.



To add one of the included coin sound effects to your coin, you need an AudioSource component. Select the coin prefab in your project files and press **Add Component > Audio > AudioSource**. It should appear in the Inspector as:



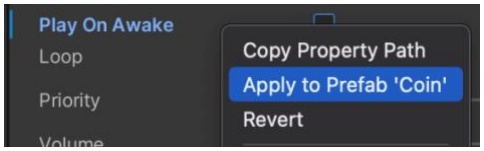
There are a lot of settings here, most of which do not interest us. The important one is **AudioClip**. Click the little circle and enter “coin” in the search bar to filter the assets to the coin sounds you recently added to the project via your new package. **Double click** one of the coin sounds to select it.



Now that you have done these steps with the prefab, you will see that all the coins in your scene now have an audio source attached to them, with the correct sound effect. The CoinPickup script looks for an AudioSource on its object and plays it when the coin is picked up. Play the game and test that it works.

Let’s try another way to change a prefab. Click on one of the individual coin objects in your scene, and untick the **Play On Awake** checkbox in the Audio Source component. This will stop the coin noise from playing as soon as the game starts.

You will now see that the text for “Play On Awake” text is blue and has a blue mark next to it, which is also shown on the instance of the coin you changed in your scene’s hierarchy panel. This indicates that the properties for this individual object have been updated and are now different to the value saved in the source prefab. To sync the prefab with the object, **right-click** on “Play On Awake”, and select **Apply to Prefab**. Now the prefab will be updated so that Play On Awake is turned off for all copies of the coin.



This method of updating a prefab is useful if you make changes to an individual instance of it in your scene that you would then like to reflect in all copies of the prefab in the scene.

We can also use AudioSources to add music to our scene. Go to the Sample Swap website and find a track which suits your game (perhaps from Royalty Free Electro?). When you find the right track, click on the Download MP3 button to go its page where you can download the MP3 (lower Kbps should be fine). Once downloaded, create a new 'Sounds' folder in the Assets folder of your project (**Right-click** and **Create > Folder**), and drag the mp3 into that folder. You can also move the imported 'Shapeform' package folder into it to keep your Project tidy.

Now create an empty object (**GameObject > Create Empty**) somewhere in your scene (it doesn't matter where). Rename it as "Music" and add an AudioSource to it. Set the AudioClip to the music file you downloaded. This time, make sure **Play On Awake** is turned on, and also turn on **Loop**, so that the music starts when you enter the game and loops when it is finished. Play your game to test out your new audio. You can adjust the 'Volume' setting in the Audio Source inspector if any particular sound is too loud.

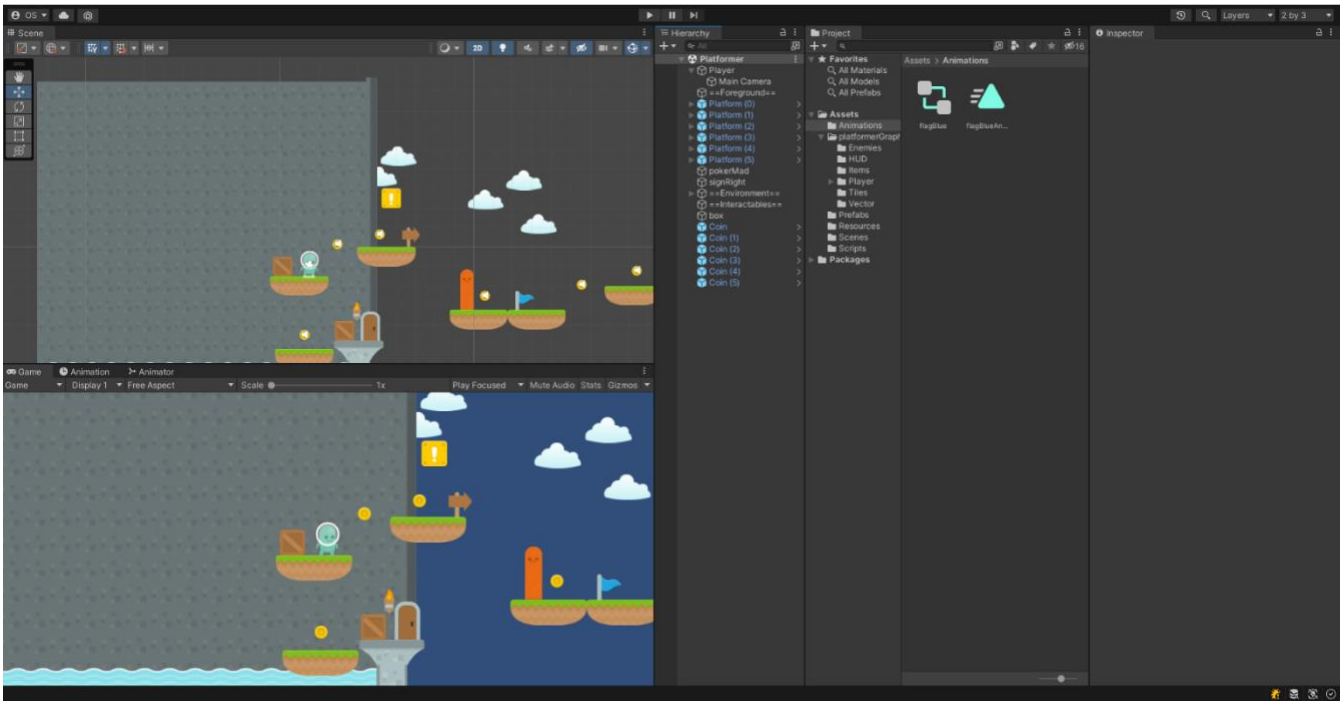
The audio system in Unity can do much more, but we will leave it at that for now.

Reminder: Save, Commit and Push to GitHub

Window Layouts

Before we move on to animation, it is worth knowing that there are two new panels we are going to be working with: the **Animation** panel and the **Animator** panel (yes, the names are confusing). These two panels are not open by default. You can find them in the **Window** dropdown menu – **Window > Animation > (Animation/Animator**).

Working with animations in the default window layout can be awkward. If you open the menu **Window > Layouts** you will see a number of alternative layouts for the standard panels. We recommend the **2 by 3** layout, as shown below and in the remaining examples.



You can resize panels within the layout by dragging their edges, or move them to different locations by dragging their tab header.

Using the Window menu, open the Animation and Animator panels (**Window > Animation > Animation** and **Window > Animation > Animator**) and dock them alongside the Game panel like so:

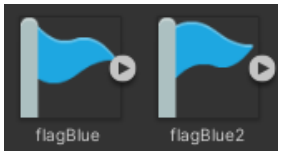
This will make the panels easier to access when we want to edit our animations later.

Animating Sprites

One common way to do 2D animation is to flip through multiple sprites with small changes between them. Consider the simplest case: flipping back and forth between two sprites. We can do this with the flag sprites that can be found in the 'Assets/platformerGraphicsDeluxe/Items' folder.

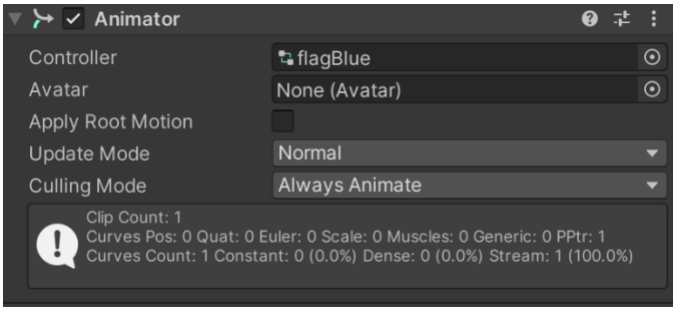
We can make the flag appear to flutter in the wind by flipping between these two sprites.

In the ‘Project’ window, select **both** flag sprites (by holding Ctrl on PC or Command on Mac as you select them) and drag them together into the scene. Unity will then ask you to save the new animation. Make sure to save it in the ‘Assets’ folder, within a newly created ‘Animations’ folder**,** and give it a meaningful name such as “flagBlueAnim”.

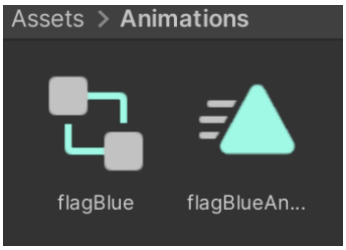


Even though we dragged in two sprites, only a single sprite is added to the scene. In the Hierarchy you should see a single object called flagBlue. What happened to flagBlue2?

If you click on the **flagBlue** object and look in the Inspector you will see that this isn’t *just* a sprite. It has an ‘Animator’ component attached to it. Unity has created a single animated sprite out of the two.



In the ‘Project’ window you will see two new assets have been created within your ‘Animations’ folder. An **Animation** file (the dashing triangle icon) and an **Animator** file (the icon with the boxes and lines).

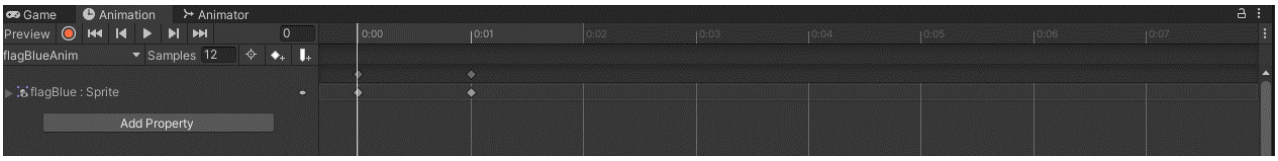


> An **Animation** is a sequence of frames which change various properties of an object. In this case, it changes the sprite used to render the flag.
>
> An **Animator** is a controller for assigning Animations to objects and for determining how to transition or blend between multiple animations.

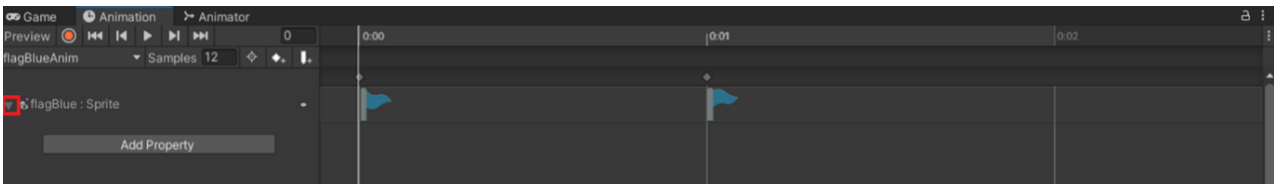
For example, a player avatar might have multiple Animations for running, jumping, etc., and a single Animator to handle when and how to switch between them.

The Animation Panel

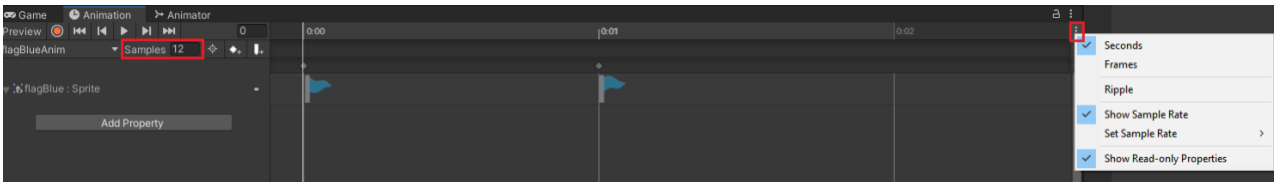
Select the **flagBlue** object in your Hierarchy and open the Animation panel. You should see something like this:



At the top left are the controls for recording and playing back animations. Underneath these is a list of the properties that are being animated. In this case it is just the Sprite. Click on the dropdown triangle to the left of **flagBlue : Sprite** to see this in the timeline on the right:



This animation has two frames, one for each sprite. Click the three dots on the right of the Animation and select **Show Sample Rate**. This will reveal a samples field view.

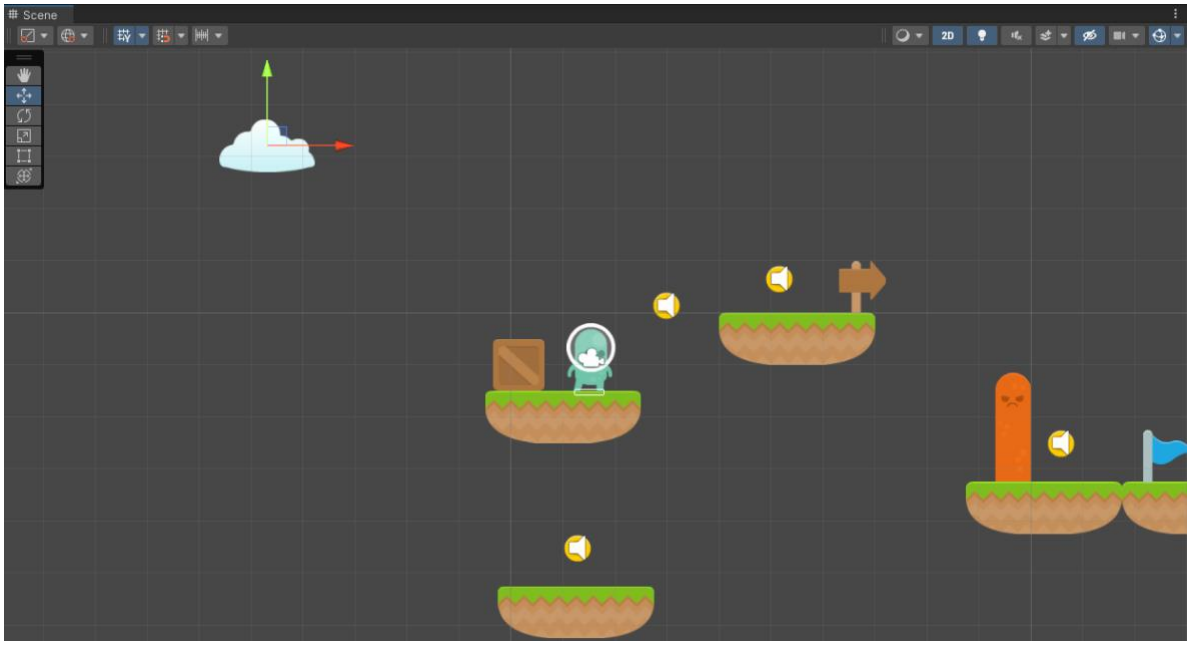


The **Samples** setting determines how often the sprite will change each second; in this case 12 frames per second. Press the **play** animation clip button, the play button found in the 'Animation' panel above the 'Samples' field (i.e. not the Play Game button), to preview the animation within the 'Scene' view. You can also see a line in the timeline view on the right as it flips through the frames. You can speed it up by increasing the number of Samples per second or slow it down by decreasing them. The timescale at the top of the timeline should scale appropriately. Select a sample rate that looks good for your flag.

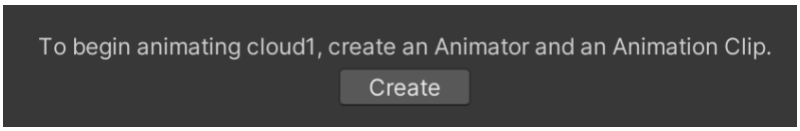
Animating movement

Animations are not just for changing sprites. They can be used to animate all manner of object properties. For example, we can use an animation to move an object around the scene.

Drag a cloud sprite into the top left area of your scene as per the example below.

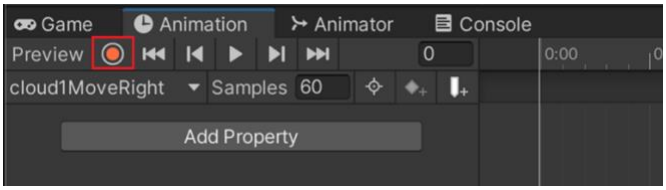


Now we want to animate the cloud so that it automatically drifts along a selected path in the scene. Open the **Animation** window and then select the cloud sprite in the Hierarchy. In the Animation window you should see:



Press **Create**, give your new animation a (good) name and save it in the project's Animations folder you created earlier.

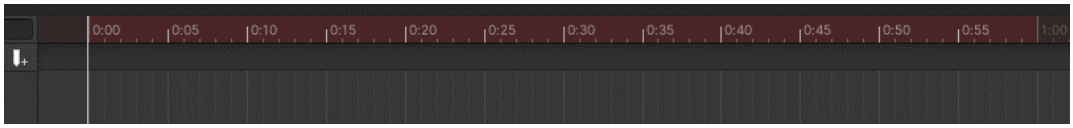
An empty animation has been created. We need to add keyframes to the timeline to show Unity how we want the cloud to animate. We can do this by using the red record button in the play animation controls in the top left corner of the panel. Click this **record button** now to activate it.



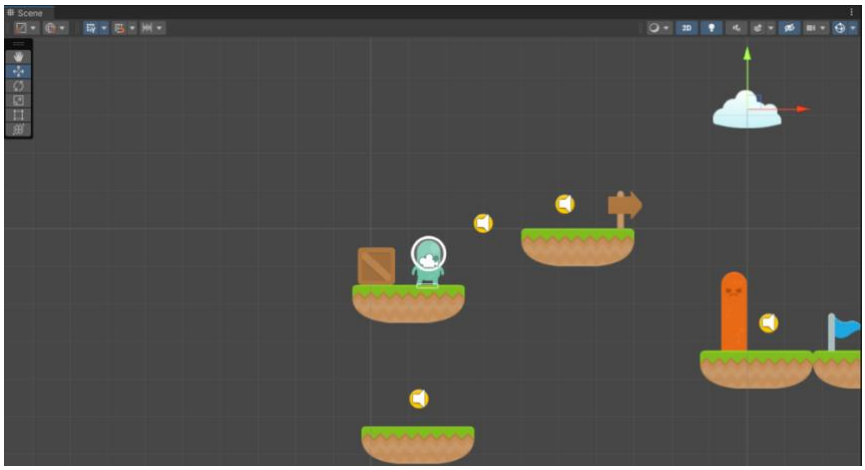
When active, the Animator will record any changes we make to the cloud object.

Animations are created by recording a sequence of 'keyframes' – snapshots of the object at specific points in time. Unity then animates the object by blending between the keyframe settings. The first keyframe for the cloud is automatically set to its initial position. Let's create another keyframe to move the cloud across the screen.

First select the time you want the new keyframe to happen, by clicking in the timeline view across to the 1:00s mark.

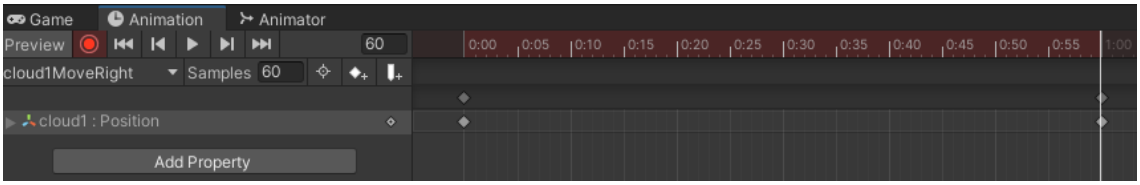


Now drag the cloud within the scene to a new position on the other side of your camera's view:

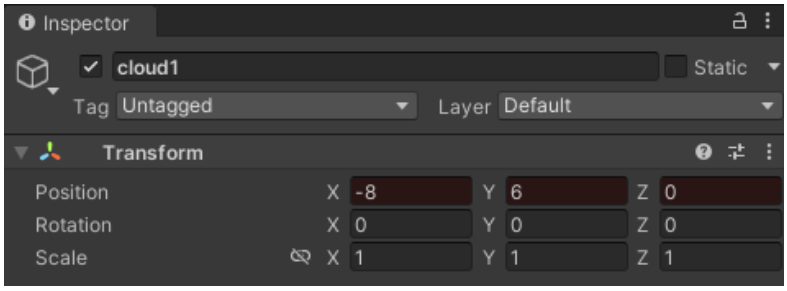


You should notice a couple of things that have changed:

- The 'Animation' panel now shows **cloud1 : Position** in the list of properties being animated.
- The timeline now shows the two keyframes as diamonds at 0:00 and 1:00.



- In the Inspector, the Transform for the cloud shows the position in red, to indicate that it is being animated.



If you press the **play** animation clip button, you will see how Unity blends between the two keyframes to move the cloud across the screen. At the moment the cloud moves in a straight line between the two keyframe positions.

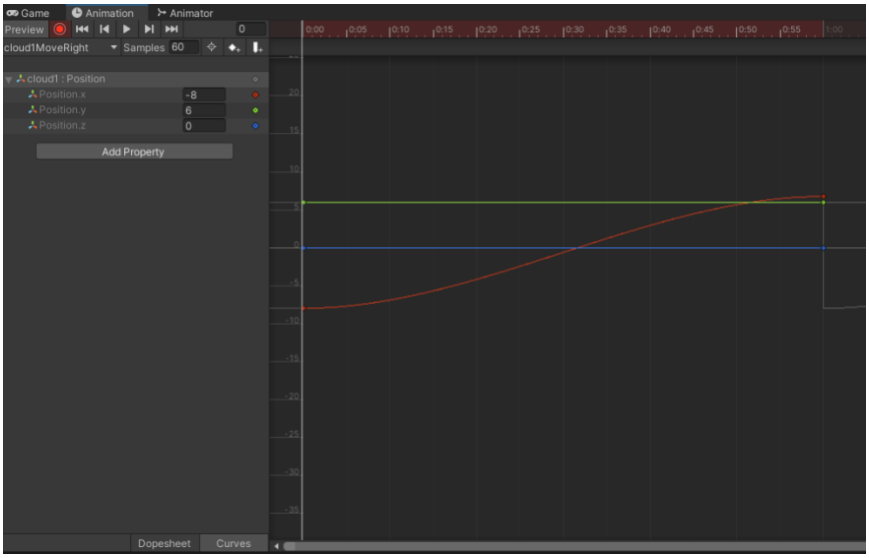
Select **0:30** in the timeline and **Right-click > Add Key** to create a third keyframe. **Play** the animation clip again and you should see the cloud move smoothly between the three positions.

Animation Curves

For a closer view of how this blending happens, select the **Curves** tab at the bottom of the Animation panel:

This opens a view which shows how the intermediate points between keyframes are calculated.

> **Note**: You can press the **dropdown triangle** to expand the cloud1 : Position property to see the colours used to represent each axis in the Curves view, and can also select an axis (e.g. Position.x) to view and modify the curve for that axis only.
>
> To move around the Curves view you can use the scroll wheel on the mouse to zoom in and out and click with the scroll wheel button to pan around the view (as in the scene view).



Try dragging around the relevant keyframe points and see how the curves are affected. If you right-click on a keyframe a popup menu will appear, which will allow you to adjust the smoothing at each point – note that manually adjusting the tangent handles (the grey lines with diamonds that appear when selecting a keyframe point) will set the keyframe to **Free Smooth**.

You can select other properties to animate using the Add Property button. Try adding **Transform > Rotation** and then changing the clouds Z rotation at different keyframes.

Now try duplicating the cloud in your Hierarchy window. Select the new cloud and make some changes to its animation. Now play the game (not just the animation). Where did the second cloud go?

If you look in the Scene window you should see that the two clouds are both there, but are on top of each other, in exactly the same position. They share a single Animation file (and a single Animator). When you copy and paste a cloud, you **do not** duplicate the animation. So when you changed the new cloud's animation, you were actually affecting the original cloud as well.

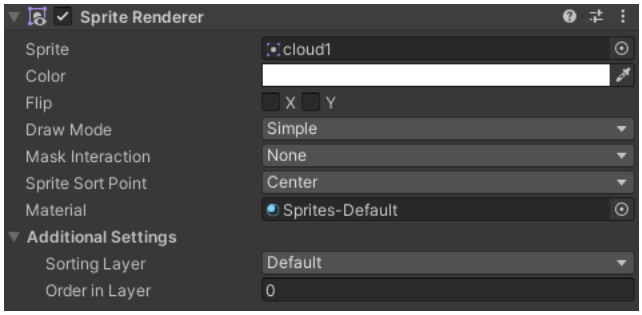
If you want the clouds to have different animations (that are not identical), you would generally want to create them separately so that each one has its own Animation and Animator. Another approach would be to make the clouds children of different parents, so that they inherit the parent's properties when animated (i.e. their positions), but that's a bit too advanced for this prac.

Reminder: Save, Commit and Push to GitHub

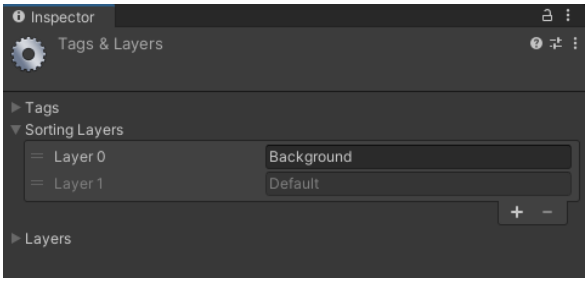
Sprite Sorting Layers

Before we get to the Animator you may have noticed another problem with the clouds. They move in front of the other sprites rather than behind. There are a variety of ways to fix this (such as using **Order in Layer**, which we covered in week 1) but the best fix for our clouds is to put them in a different **Sorting Layer** (not related to the layers we detailed last week – i.e. the Layer called "Player" used for jumping).

Select the cloud sprite and look in the Inspector. In the **Sprite Renderer** component you should see a field called **Sorting Layer** which is set to Default.



Click on this and select **Add Sorting Layer…** The Tags & Layers view will open. If you don't already see a layer called 'Background' then press the + button to create one.



You can click the handle on the left (the = grip icon) and drag the layer above Default to ensure that any objects assigned that sorting layer will be drawn first, behind the Default layer.

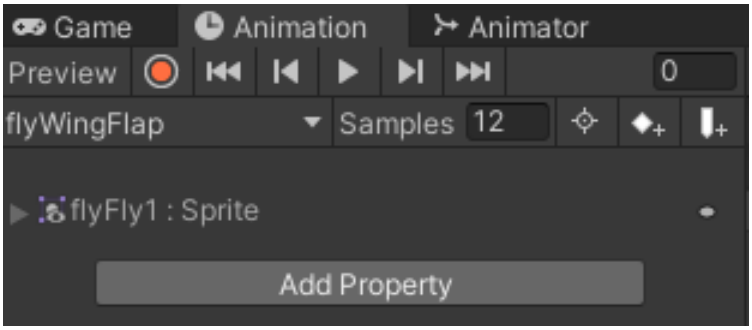
Go back to the scene, select all your clouds, and change their Sorting Layer to Background. They should now render behind the other sprites.

Layering animations

Let's add another animated sprite to our scene, an angry fly. In the Enemies folder, grab the two fly sprites and drag them together into the scene to create an animated sprite. Again, you will be prompted to create a new animation file. Give it an appropriate name.

Play the animation and the fly should buzz, menacingly. Now suppose we want the fly to be an enemy for the player to avoid. We would want it to fly up and down so that the player has to time their run underneath it. To implement this, we could add a movement component to the existing animation, but the length of the wing-flapping animation and the length of the movement are quite different and unrelated. It would be better to create a separate animation.

Select the fly and open the Animation panel. Under the play record button should be a box marked with the name of your animation. This is the current animation (automatically created). Click on this box to reveal a menu:

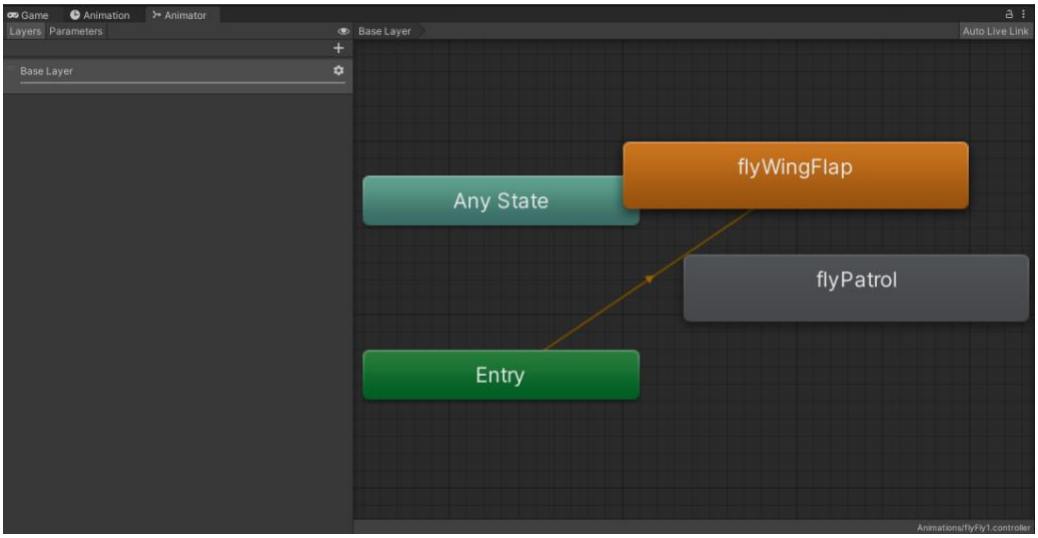


Select **Create New Clip…** from this menu. This creates a new animation. Call it **flyPatrol**.

In the new Animation clip, set up a patrol using keyframes like you did for the cloud. You will need to make it loop (min. three keyframes total), with the final keyframe being a copy of the first (use Copy/Paste), or else the fly will teleport back to the beginning when it reaches the end.

You now have two separate animations. The original flyWingFlap animation, which animates the sprite and the flyPatrol animation which moves the sprite around. If you play the game now, only the first animation has effect. You need to use the 'Animator' to tell Unity to use both at once.

Open the **Animator** panel. You should see something like the below image.



Note: Move around this view using scroll zooming and panning, as with the Curves view.

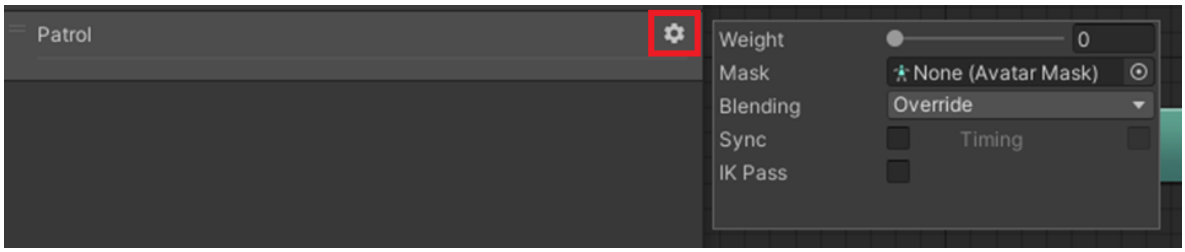
The two animations, flyWingFlap and flyPatrol, are shown as boxes. There is also a box marked 'Any State' (we won't worry about this box for now) and 'Entry', with an arrow from 'Entry' to 'flyWingFlap'. That arrow essentially means "When the game is played, activate the flyWingFlap animation".

This is a 'state-machine'. It describes how we turn animations on and off at different times. Normally only a single animation can be active at any time, but if you look over the left-hand side of the panel, you'll see a list of animation Layers (again, different Layers to the other two kinds we've already seen). We can have multiple animation layers playing at the same time.

Click the + button at the top of the list to create a new layer and call it "Patrol".

The state-machine on the right should show just ‘Any State’ and ‘Entry’. Find the Patrol animation in your Project panel (it should in the Animations folder) and drag it into this Animator window. It should appear as an orange box. The first animation you add to a layer is automatically linked to ‘Entry’, so this animation will play when the game begins.

Now, click the little cog to the right of Patrol. A menu will appear:



The **Weight** slider determines how much effect this layer has. It is initially zero. Turn it up to 1 so that the layer has complete control over the fly’s movement. Now **Play** your game. The fly should flap its wings *and* fly around at the same time.

In this example, the two animations we added affected two different properties of the object. If we had two animations which both affected the same properties, then we would use this menu to determine how they combine. The **Blending** mode is currently set to ‘Override’, which means later animations override earlier ones in the list. If you change this to Additive, then the later animations will add to the earlier ones. The weights can be used to control how much each animation contributes to the final result.

Moving Platforms

So it may have already occurred to you that we could use the animation system to make moving platforms in our game. Unfortunately this isn’t quite as easy as it seems. Try it and see what happens. Add an Animation to a platform to make it move, then put a box with a rigidbody on it.

You’ll notice a few problems:

1. When the platform moves up, the box sinks into the surface.
2. When the platform moves down, the box bounces
3. When the platform moves sideways, the box slides around.

These problems are strongly embedded in the way the Unity physics engine works. The model of physics we expect in platformers is unrealistic in a number of ways. Ultimately it requires a lot of extra programming to do convincingly in a realistic manner, which unfortunately falls outside the scope of this unit (though will come up again in COMP2160).

Reminder: Save, Commit and Push to GitHub

Animating the Avatar

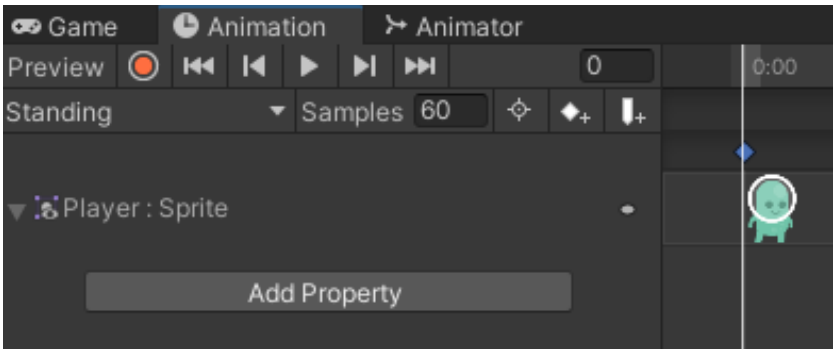
The last thing we're going to do is animate the player avatar. This is a little tricky, as we need the animation to respond to the avatar's current situation. We're going to set up three different states:

1. **Standing**: When the avatar is on the ground and not moving.
2. **Walking**: When the avatar is on the ground and moving.
3. **Falling**: When the avatar is not on the ground.

First of all, we'll need a different player controller script, to recognise when these different states change. Find the **PlatformerMoveAnimated.cs** script in the Scripts folder of your project. Attach this script to your player character and do the following:

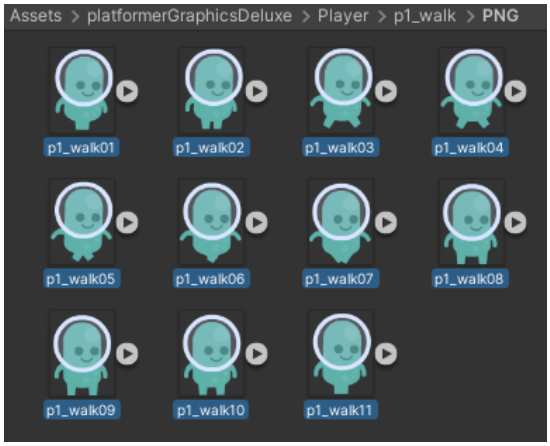
- Adjust your **Walk Speed**, **Jump Speed** and **Walk On Air** to match the settings in the old PlatformerMove component if necessary
- Remove **PlatformerMove** (the old script attached to the player)
- Make sure the **Ground Layer Mask** has the 'Player' Layer unticked.

Now we want to create three different animations, one for each state. Select the player object and open the Animation window. Click the **Create** button to create a new animation called **Standing** (save it to the 'Animations' folder). Drag the standing sprite which matches your player character from the platformerGraphicsDeluxe/Player folder and drop it onto the timeline. Since the standing animation just has one frame, we only need the one sprite.



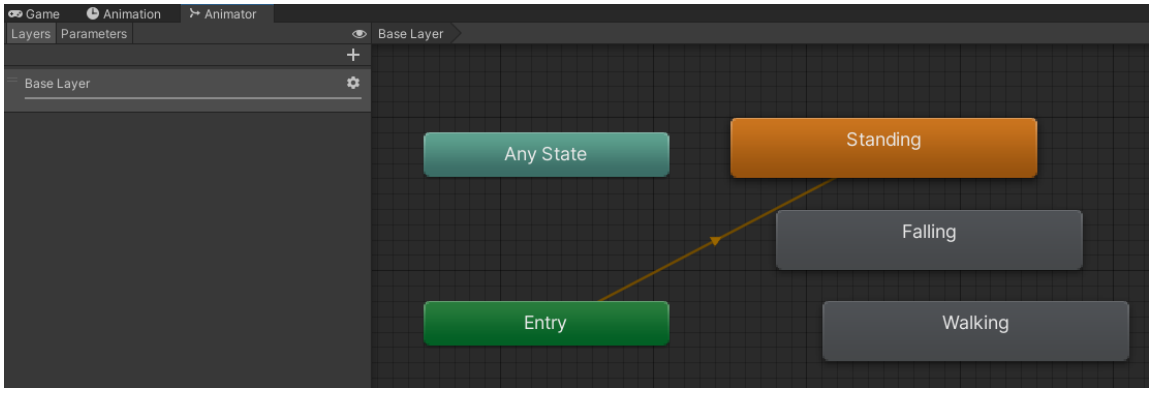
Now click on the word **Standing** in the top left of the panel and select **Create New Clip…**. Call the new clip **Falling** and drag an appropriate sprite onto the timeline (hint: jump or hurt?).

Finally, create a third animation clip called **Walking**. Drag *all the sprites* from the platformerGraphicsDeluxe/Player/p#_walk/PNG folder onto the timeline (matching # to the 'p' number of your player character sprite, i.e. 1 for the green character, 2 for the blue character, and 3 for the pink character).



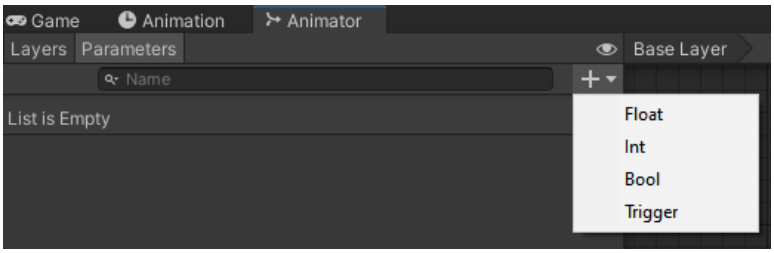
This creates an animation of the player walking. Play this animation and tweak the **Samples per second** until it is running at a reasonable speed.

Now we need to connect these three animations together. We do this in the 'Animator' panel. In this panel you should see:



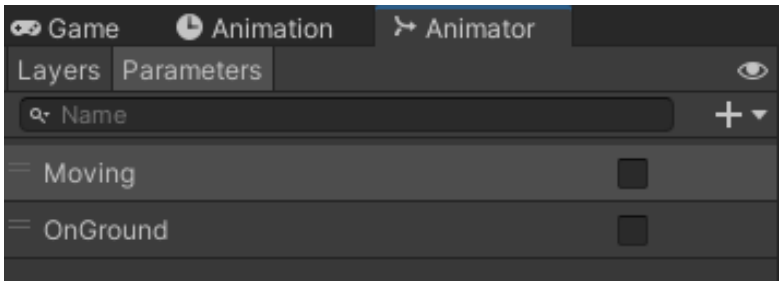
'Standing' is the default animation (in orange) because we created it first. If this isn't the default for you, right-click on 'Standing' and select **Set as Layer Default State** to fix this.

The 'PlatformerMoveAnimated' script checks for two parameters to control the animation, called 'Moving' and 'OnGround', however we need to tell the animator that these exist. On the top left of the panel are two tabs, 'Layers' and 'Parameters'. Select the 'Parameters' tab, and press press the **+** button next to the search box.

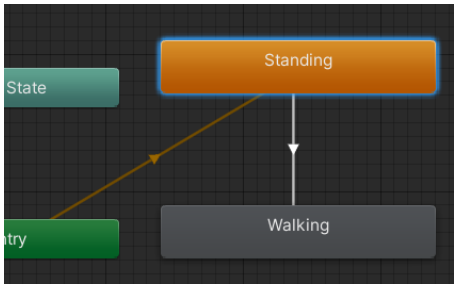


You will see a little menu with the options ‘Float’, ‘Int’, ‘Bool’ and ‘Trigger’. Our parameters, ‘Moving’ and ‘OnGround’ are both Booleans (true/false values) so select **Bool**, creating a new parameter. Call this parameter **Moving**. Create a second Boolean parameter called **OnGround.**

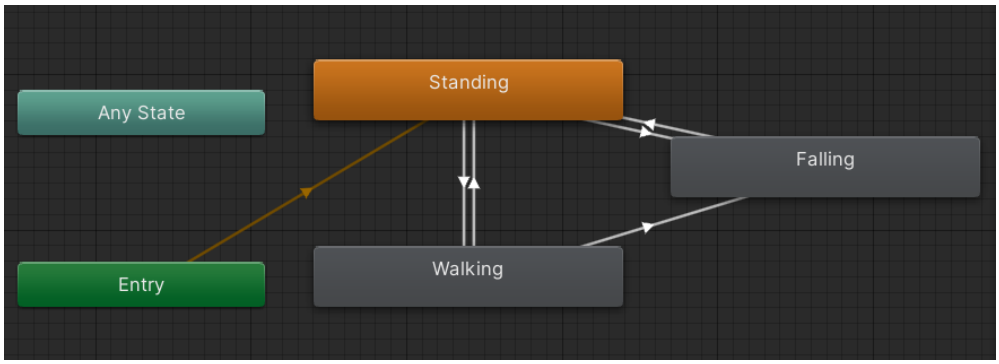
Important: These names are case sensitive and must have no spaces, as they are variables set by the ‘PlayerMoveAnimated’ script. Type them out **exactly** as directed, like pictured below.



Next, we want to create transitions between the different animation states. Right-click on the **Standing** state and select **Make Transition**. Then click on **Walking** to create a transition between them as shown below.



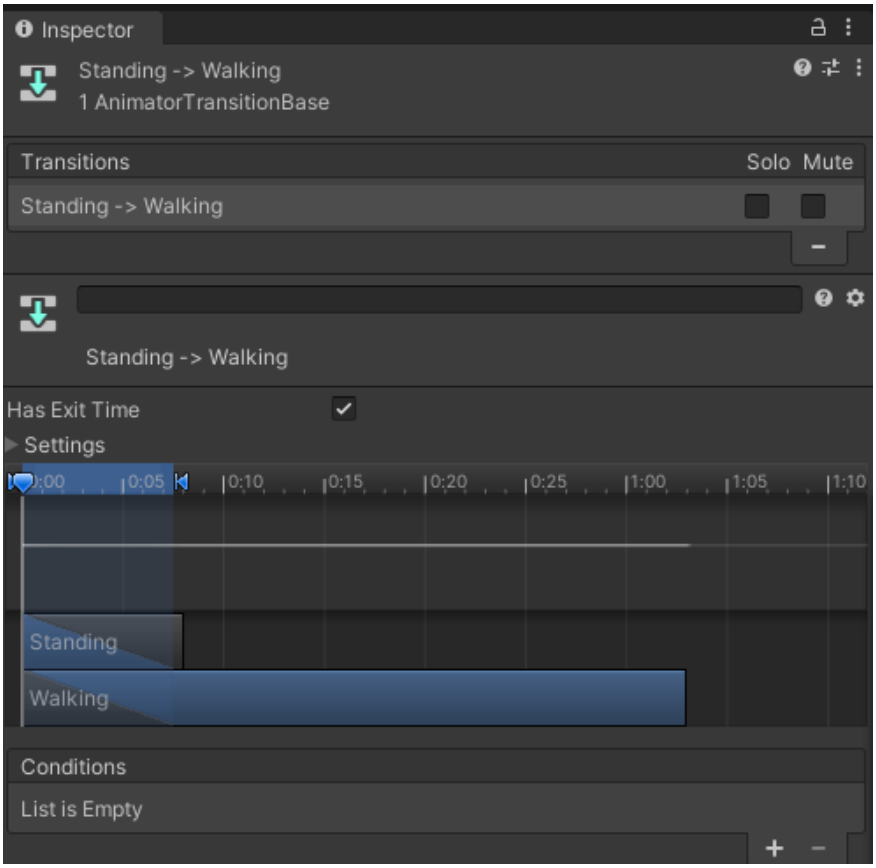
Repeat this process to make another transition from **Walking** back to **Standing**. Then do it again to make two transitions back and forth between **Standing** and **Falling**. Finally make a single transition from **Walking** to **Falling**. In the end, your state machine should look like the one show below.



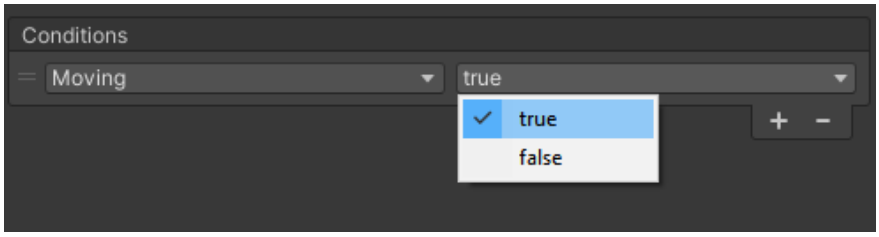
These are the rules we want for our state machine:

1. If the player is Standing and starts moving, move to the Walking state.
2. If the player is Walking and stops moving, move to the Standing state.
3. If the player is Standing and is no longer on the ground, move to the Falling state.
4. If the player is Falling and touches the ground, move to the Standing state.
5. If the player is Walking and is no longer on the ground, move to the Falling state.

To set up these rules we need to connect the transitions above to the ‘Moving’ and ‘OnGround’ parameters. Click the **Transition** (the connecting arrow) for **Standing** → **Walking** and look in the Inspector. You should see something like the picture below.



We need to add a condition to the list at the bottom. Press the **+** button in the bottom right corner and select the parameter **Moving** and set the value to **true**.



Now when **Moving** becomes **true**, we will transition from the 'Standing' state to the 'Walking' state. Repeat this process with the other transitions to create the rules above.

1. Standing → Walking when Moving is true
2. Walking → Standing when Moving is false
3. Standing → Falling when OnGround is false
4. Falling → Standing when OnGround is true
5. Walking → Falling when OnGround is false

Play this and test that it works. You may notice one problem; it takes a while for the Walking animation to stop playing when transitioning to Standing or Falling. This is because the Walking animation is finishing its cycle before the next animation can start. You can change this by turning off **Has Exit Time** in the Walking → Standing and Walking → Falling transitions.

If you're not seeing the animations correctly:
If you are finding that your character is displaying the wrong animation or seems to be stuck on an animation, double check that:

1. the boolean conditions are set correctly in each transition.
2. in the Animator panel, the parameters **Moving** and **OnGround** are spelt and capitalised correctly, and do not have spaces in their name.
3. you have unticked the **Has Exit Time** checkbox in the inspector for your transition if you find any of your animation transitions are lagging.

Reminder: Save, Commit and Push to GitHub

Show Your Demonstrator

To demonstrate your understanding of this week's content to your prac demonstrator you might show:

- Your awesome level with background music.
- The sounds from your player character collecting coins.
- Your animated player character, flag, and fly.
- How to create an animation out of multiple sprites.
- How to adjust the speed of the animation.
- How to create a movement animation.
- How to adjust the animation using curves.
- How to layer two animations using the Animator.
- How to transition between multiple animations using the Animator.