Animation

In order to make your experience come to life we need to add some motion. We will do this by adding animations! Unity allows us to animate in a couple different ways.

Animation Through Scripting:

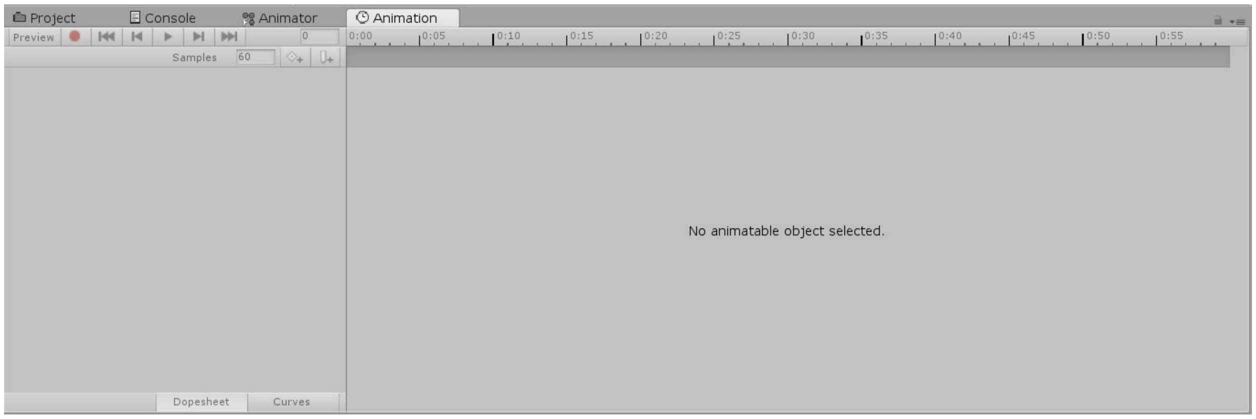
Guess what? You already know how to do this! In the previous assignment, you learned how to control the position, rotation, and scale of an object through scripts to open the door. However if you are not privy to the idea of doing vector math, there is a much simpler, more visually intuitive way to do it.

Unity's Animation System

In Unity, every object is a animatable, so long as it has an **Animator Controller** component on it. The Animator Controller interacts with an **Animator** which keeps track of the various individual animations or **Animation Clips** associated with a particular object. Each animatable object can have many clips and they may interact with each other using

Using the Animation Window:

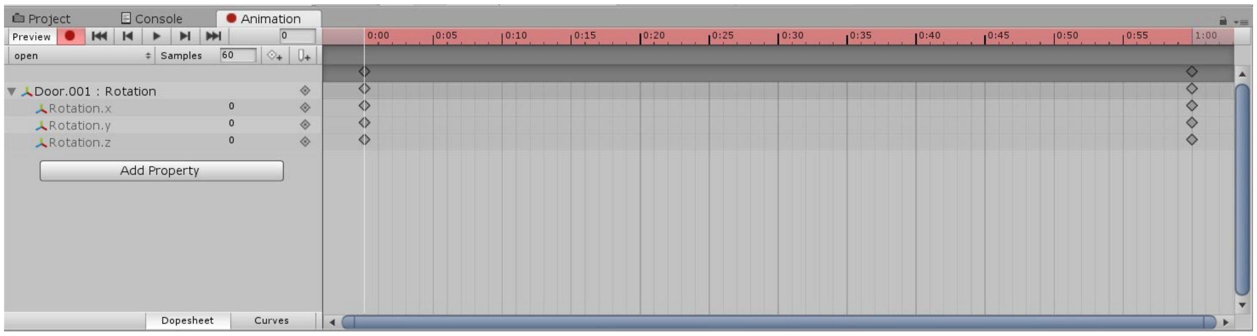
You can access the Animation window by using Window -> Animation -> Animation. It should look something like this:



We want to animate the door, but right now there is no animator component on it. You can create one by selecting the door and clicking **Create** in the **Animation Window.** This will also prompt you to choose a name for your animation clip, which in our case you should name it **open.anim**. Alternatively you can create one by selecting the door, going to the inspector and hitting **Add Component** and search for **Animator Controller**. Then you can go in the **Project -> Create -> Animator** and give it a descriptive name. Then you must drag the Animator into the slot on the Animator Controller.

In Unity, we can use the Animation Editor to key properties of any component on a GameObject. In order to move an object, we can key the position values of the object, just like you might key them in an animation program like Maya or Blender.

We can add properties that we want to key by hitting the **Add Property** button**.** This will bring up a list of our keyable components. To open a door, it rotates on its hinges, therefore we will chose **Transform -> Rotation**. You will notice that we can individually key the x, y and z rotation. Because Unity uses a **Y up** coordinate system, we will key the **Y rotation**.



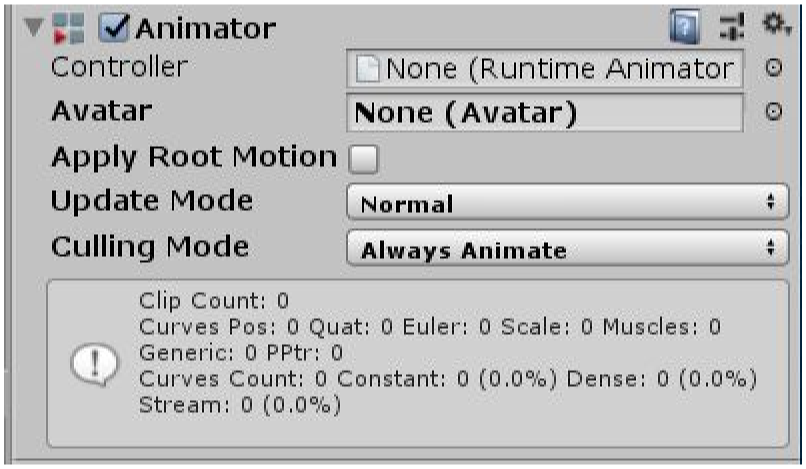
We can turn on automatic keyframing by hitting the little red **Record** icon above the timeline. This will automatically create a keyframe for the door in its neutral position. To set the next keyframe, lets **drag the time slider** to the end of our clip. Once we are there, we can adjust the rotation value for the door, by either manipulating it with the **rotation tool b** or by **setting the value in the inspector**. Let's set it to around **90 degrees**. Once you set it, turn off automatic keyframing. You can now see your animation by hitting play.

Yay! Our door opens, but you might notice that it snaps back as soon as the animation clip ends. We can fix this. Go into your **project** tab and find the **open** animation clip. If you select it and look in the inspector, you will see that the **Loop Time** checkbox is selected. This will continuously loop the animation, which we don't want, so we can deselect it. Now if you play again, it will stay open.

Congrats! You have just animated your first object in Unity! Unity allows you to do a lot more with animations as well. If you want to get

Character Animation

Our game would be kind of boring without characters, so were going to show you how to add them. First we'll go to the **Supercyan Character Pack Free Sample -> Prefabs -> buddy**. Select and drag that into your scene. Now we need to add an **Animator Controller** component by going to the inspector and clicking **Add component** and searching **Animator**. You should see something like this.



Controller: None (Runtime Animator
Avatar: None (Avatar)
Apply Root Motion
Update Mode: Normal
Culling Mode: Always Animate

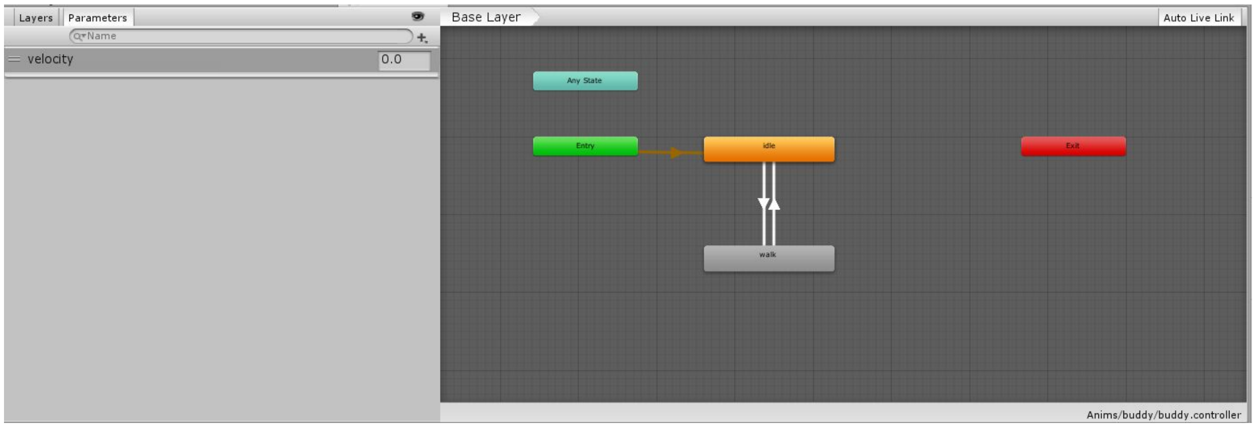
```
Clip Count: 0
Curves Pos: 0 Quat: 0 Euler: 0 Scale: 0 Muscles: 0
Generic: 0 PPtr: 0
Curves Count: 0 Constant: 0 (0.0%) Dense: 0 (0.0%)
Stream: 0 (0.0%)
```

The **Avatar** and **Animator** variables are currently set to none. **Avatars** are definitions for how animations affect the transforms of a model. Let's assign it an avatar. Because we have animations, one is pre defined for us so we can select that by pressing the circle icon next to the **Avatar** box and selecting the **common_people@mposeAvatar.** Next let's create an animator in your **project** tab by hitting **Create -> Animator Controller,** and name it something like **main character**. To edit this we need to open the **Animator Editor** by going to **Window -> Animation -> Animator.** When it opens you should see something like this. This window allows you to build a **State Machine** to control all of your animations.

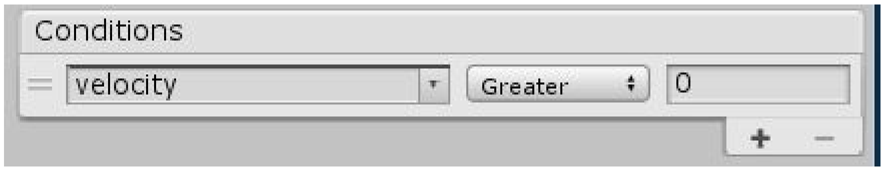
A **State Machine** is a graph of different **states** that are connected to each other by **conditions**. In our case a state is a particular **animation** or **blend tree** in the graph, and our edges are boolean conditions that test the **parameters** that we define in our state machine. We are going to use the parameters with our movement script from the previous assignment to make the character walk around.

You can create a state by dragging an **animation clip** into the graph. Let's start by dragging our **idle** animation into the window. You will notice that it turns orange. This is because Unity automatically sets this to be our **default state**. This is what we want, because idle means our character is not doing anything, and we don't want the character to do anything unless we tell it to. If you ever want to change the default state, we can do this by **right-clicking** on the node and selecting **Set as default state**.

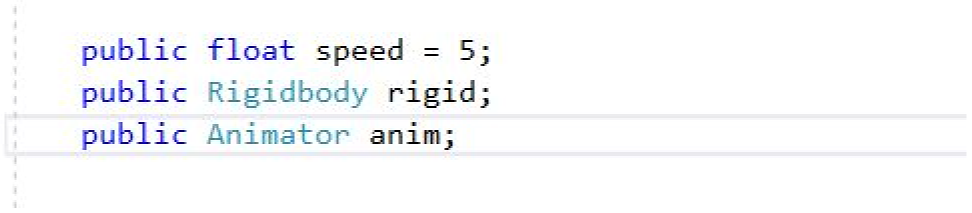
Next we will add the **walk** animation to the graph. Because we want our player to actually walk while they're moving forward, we need to add a parameter. Go to the **Parameter** tab on the Animator Window and set **Add Float** and name it **velocity**. We can now use this parameter to make transition logic. Back in the graph, **right click** the **idle** node and choose **Make Transition**. This will make a little arrow, which you should drag to and click on the **walk** animation.



If you select the arrow, this should bring up a menu in the inspector. Uncheck the box marked **Has exit time**. Look for the **Conditions** list and add a new condition by hitting the **+** button. Because velocity is our only parameter it will have already setup the transition for you. It should look something like this.

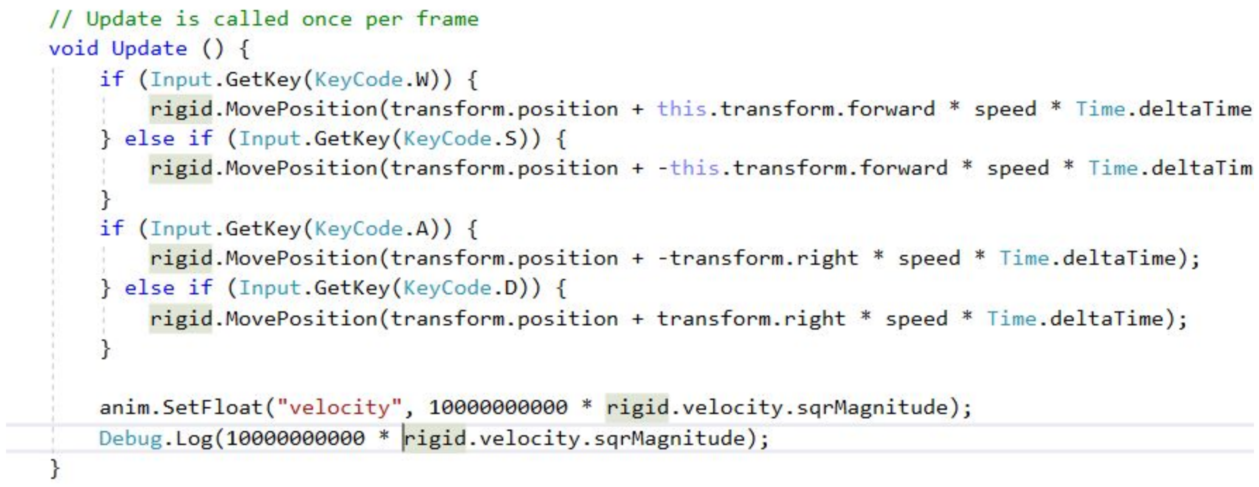


What this is saying is that we will transition from the idle to the walk animation when velocity is greater than 0. We will need to **change** the value of **0** to **0.1**. This is to account for floating point math errors. In order to make this actually update, we need to set the parameter in our script. If you open up your PlayerController Script from the last assignment, we will be making a few simple changes. First we need to add a reference to our **Animator** component like below.



```
public float speed = 5;
public Rigidbody rigid;
public Animator anim;
```

Next we will just add one line at the end of our update method. We use the setFloat method from our animator and we are giving it the name of the parameter and our z velocity. It is important to spell the parameter name exactly as you have written it in the editor, otherwise it will not work.



```
// Update is called once per frame
void Update () {
    if (Input.GetKey(KeyCode.W)) {
        rigid.MovePosition(transform.position + this.transform.forward * speed * Time.deltaTime
    } else if (Input.GetKey(KeyCode.S)) {
        rigid.MovePosition(transform.position + -this.transform.forward * speed * Time.deltaTim
    }
    if (Input.GetKey(KeyCode.A)) {
        rigid.MovePosition(transform.position + -transform.right * speed * Time.deltaTime);
    } else if (Input.GetKey(KeyCode.D)) {
        rigid.MovePosition(transform.position + transform.right * speed * Time.deltaTime);
    }

    anim.SetFloat("velocity", 10000000000 * rigid.velocity.sqrMagnitude);
    Debug.Log(10000000000 * rigid.velocity.sqrMagnitude);
}
```

Now you should add the **PlayerController** script to your character, and assign the **rigidbody** and **animator** components in the inspector. You can do this by dragging your buddy into both of the slots, and the inspector will find the right component on them. Now if you play and move forward, your buddy should start walking. However you may notice that he keeps walking and doesn't stop. This is because we don't have a way to get out of the **walk** state once we get there. To do this we can add another transition from **walk** to **idle**. For this set the condition to be **velocity less 0.1,** and make sure to again turn off **Has exit time**. Now you should be ready to go!

Congrats!! Your character has come to life! Some of his movements are still a little awkward. As a **Bonus Challenge** use the **backwards-walk** animation and try to get your character to walk backwards as well as forwards.

If you want to look at a more complex version, you can check out the prefab pack that is included with the character.