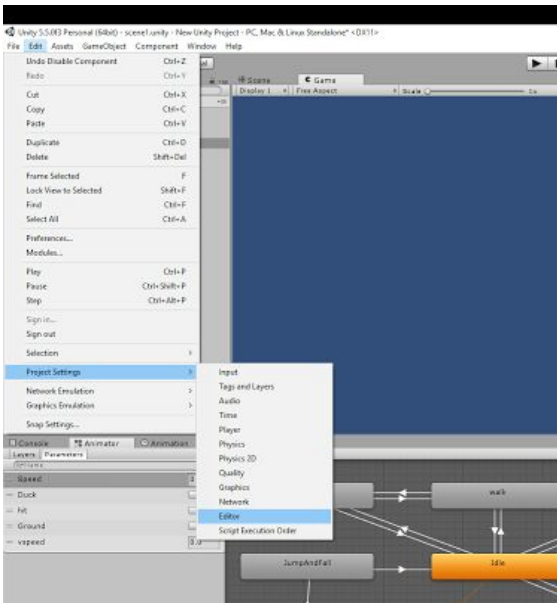
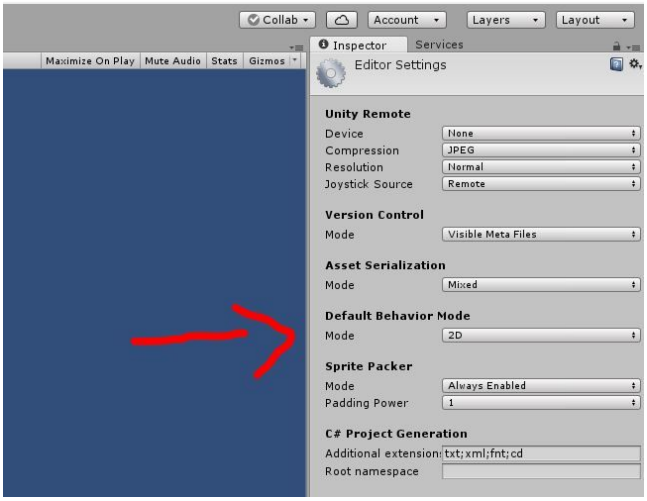
Setting up your file.

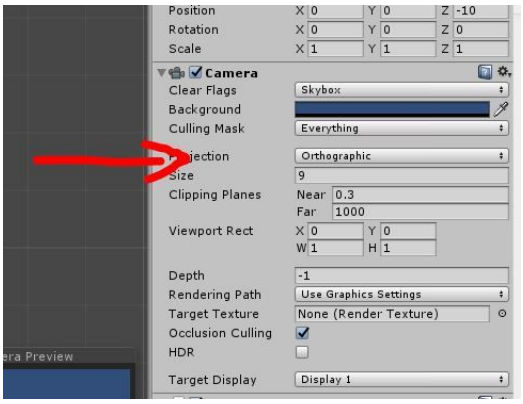
New Unity 2D project.

Edit - project settings - editor

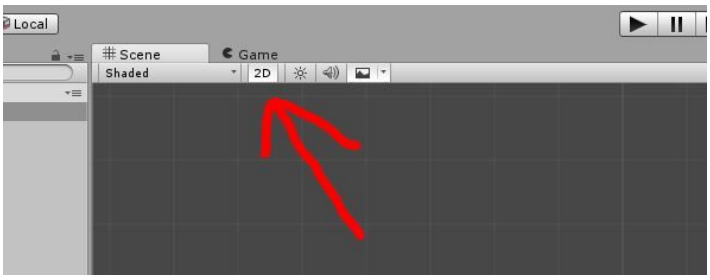




Default Behaviour mode should be 2D

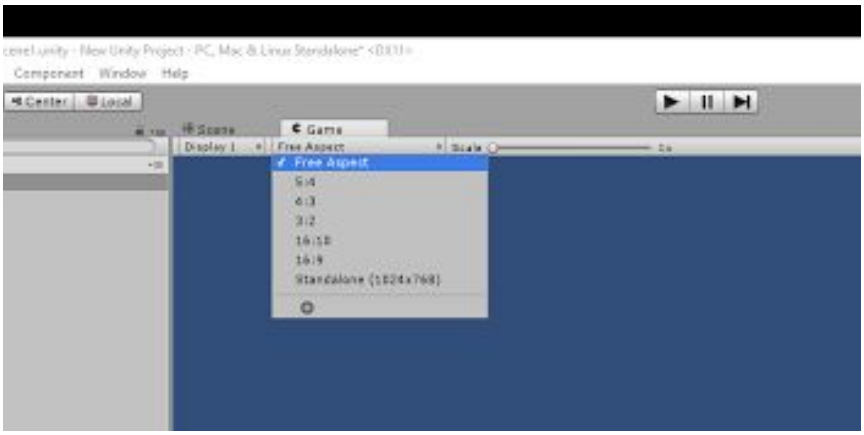


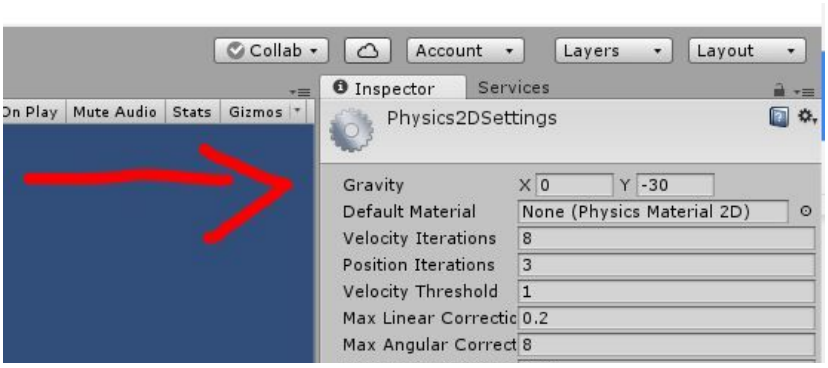
Cameras should be Orthographic



Scene view set to 2D mode.

You can set the screen output on the top of the game view



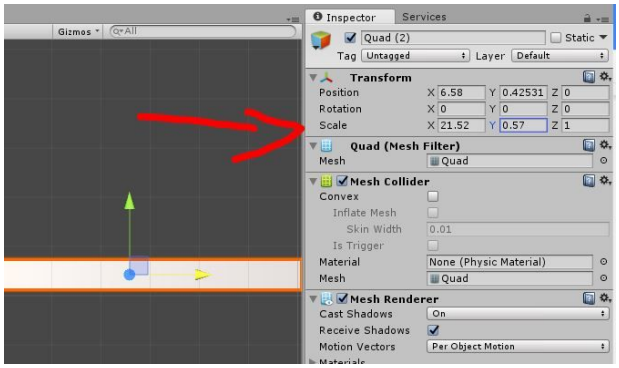


Set gravity in
edit -project -settings - 2D physics
from -9.81 too -30
(more of a platformer feel)

Set up an environment

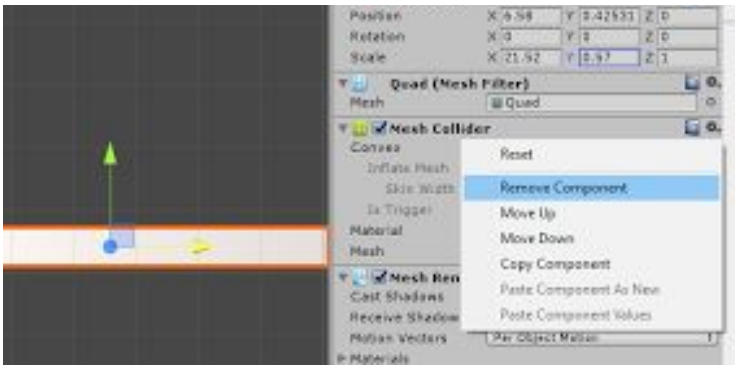
create some quads for the ground,

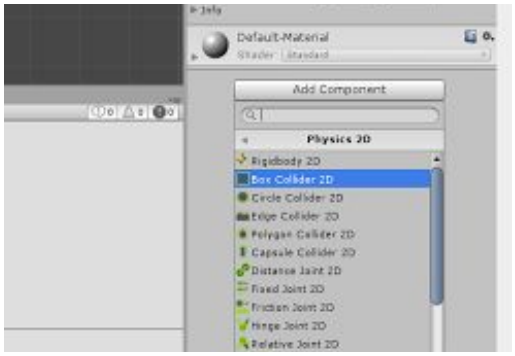
GameObject - 3D Object - Quad



Set them to a scale that suits your needs

remove (3d) Mesh collider component

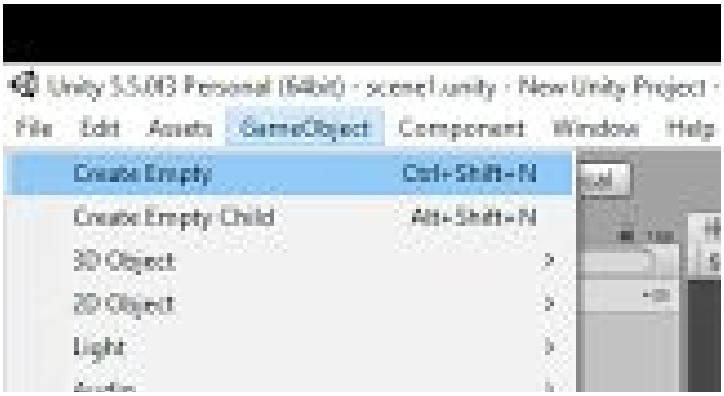




& add a 2d box collider
Add Component - Physics 2D - Box Collider 2D

Create an empty game object named *character*.

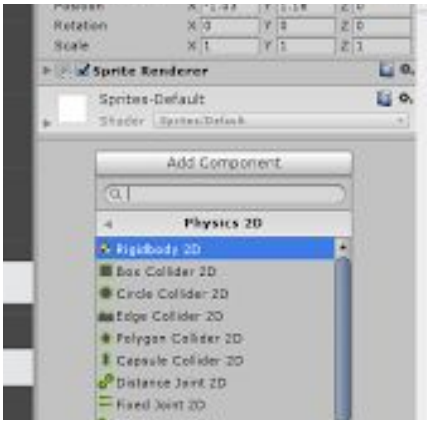
Game Object - Create Empty

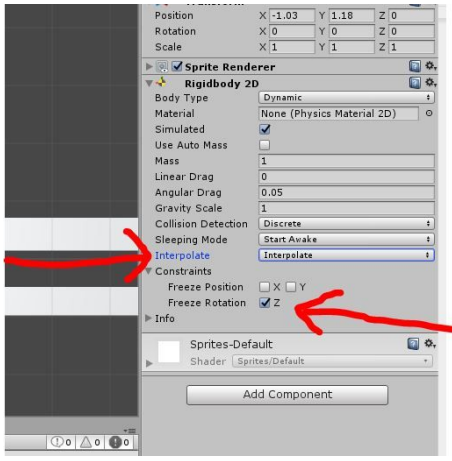


With the **character** GameObject selected.

Add a renderer to it.
Add component - rendering - sprite renderer,
(all textures will come in as sprites in 2D mode).

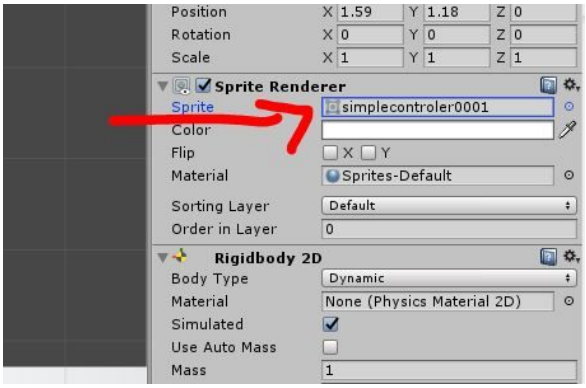
Add a Rigidbody to it.
Add component - Physics 2D - RigidBody 2d





In the Rigidbody 2D settings,
tick constrain z axis
& set interpolation to interpolate

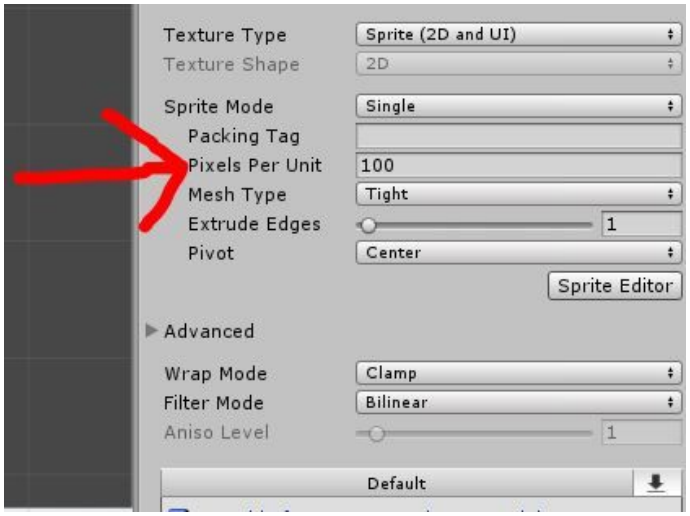
Copy your animation sprites into your Unity project folder.

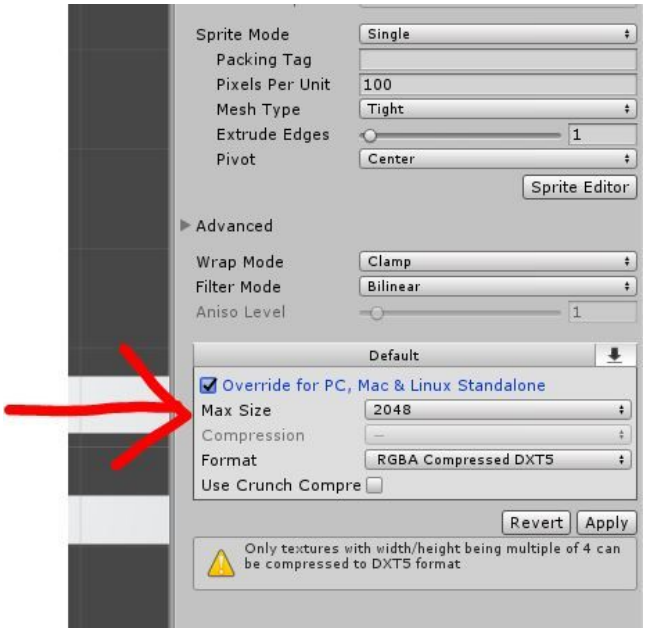


& from the **Project** window drag a sprite (one of your idle frames) into the renderer,

If your sprite is blurry, you may need to check the size your brining it in at,

you can set the pixels to unity unit ratio by selecting a sprite in the project window and looking in the inspector here.





You may also need to adjust the max size unity is rendering at and tick

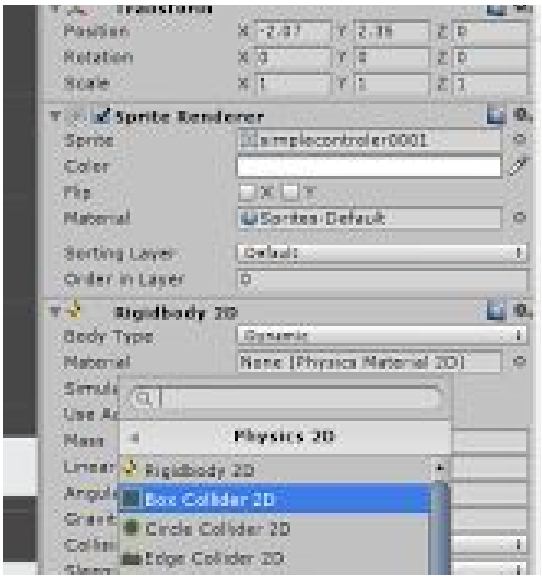
override for pc, Mac & Linux Standalone

to adjust the Max Size limit to prevent unity from compressing your images

(although they should not really be that big and can be resized from large working files before import).

Once you're happy with your sprite, select all your sprites together and make the same changes to them all (if you had any)

Set your colliders on your character gameobject.

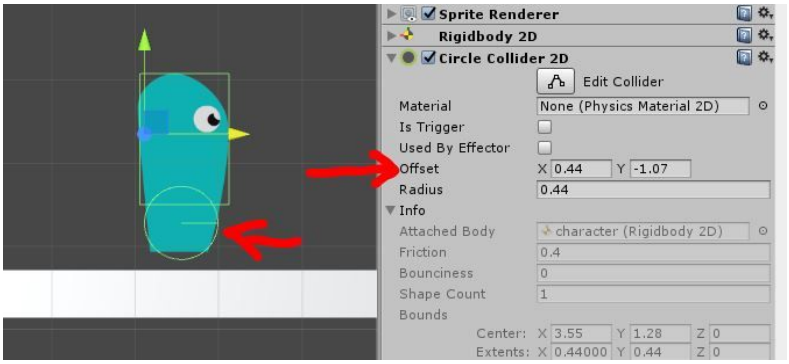


Depending on the shape of your character you may need to use more than one collider.

I'm using one circle and one box collider for this long bodied character.

ADD Component - Physics 2D -Box Collider 2D

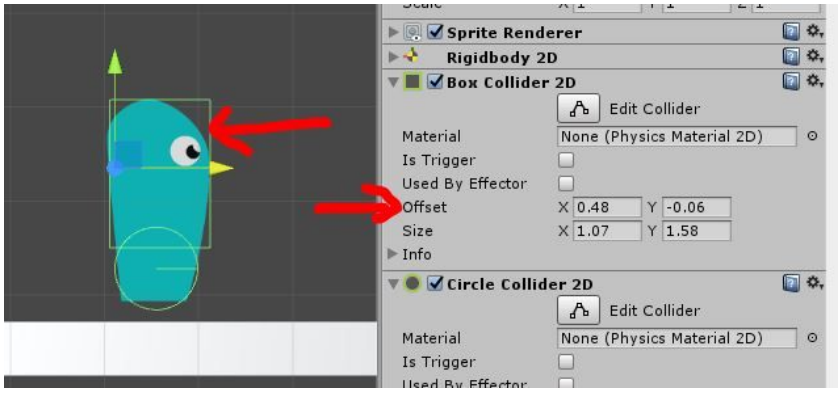
ADD Component - Physics 2D -Circle Collider 2D



Use the **offset** and **radius** settings in the circle collider settings in the inspector to adjust the green circle to neatly fit at the bottom of your character.

The circle shapes helps with edges ie the player becoming stuck at uneven surfaces in the level.

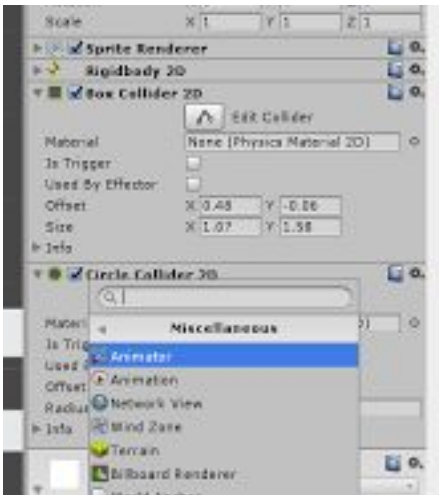
Use the **offset** and **size** settings in the circle collider settings in the inspector to adjust the green square to neatly fit the rest of your character.

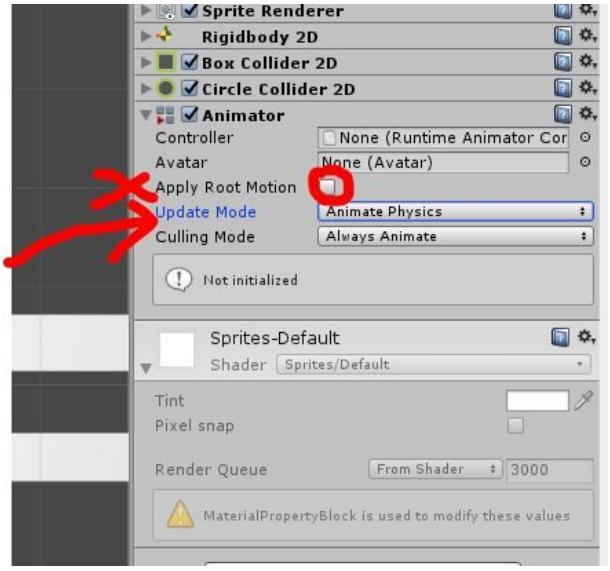


At this point you can play the game a see the character fall and lad on the floor.

We need to add the **Animator** component to the character gameobject.

Add component - Miscellaneous - Animator,





In the Animator component in the inspector, uncheck **apply root motion** if its checked

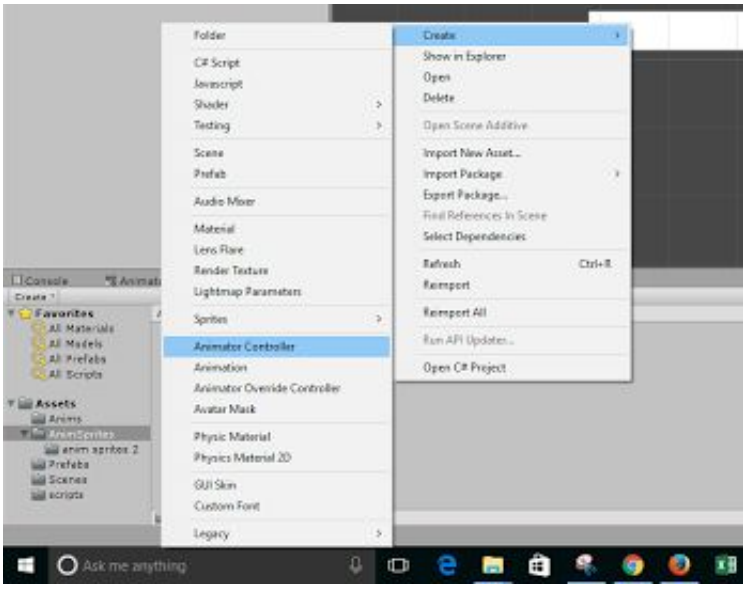
Choose **Animate Psychics** from the **Update Mode** drop down menu.

Now we need to create the animation controller.

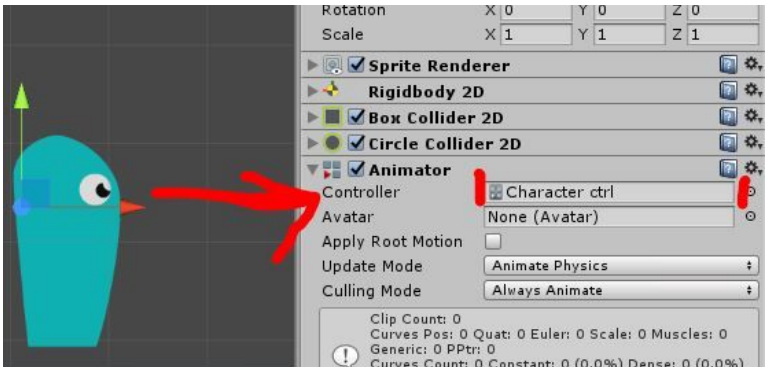
In project window, *(you can put it in a new folder called Anims with your animations if u wish)* right click and choose

Create - Animator Controller,

and name it ie *character ctrl*.



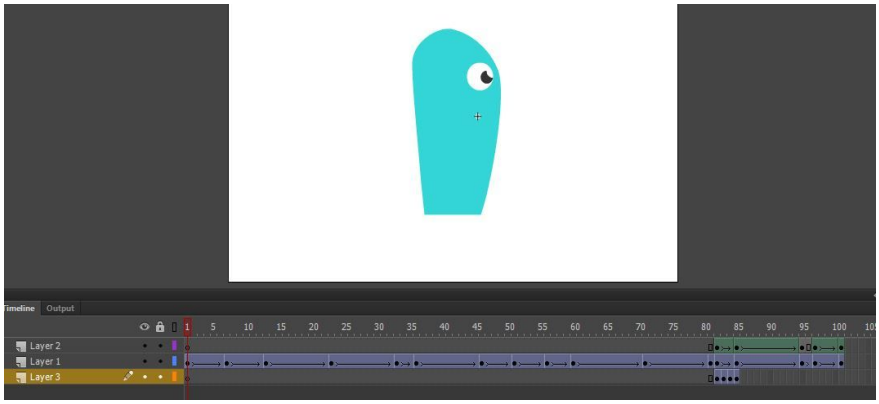
Then drag and drop it in the character game objects **Animator Component**.

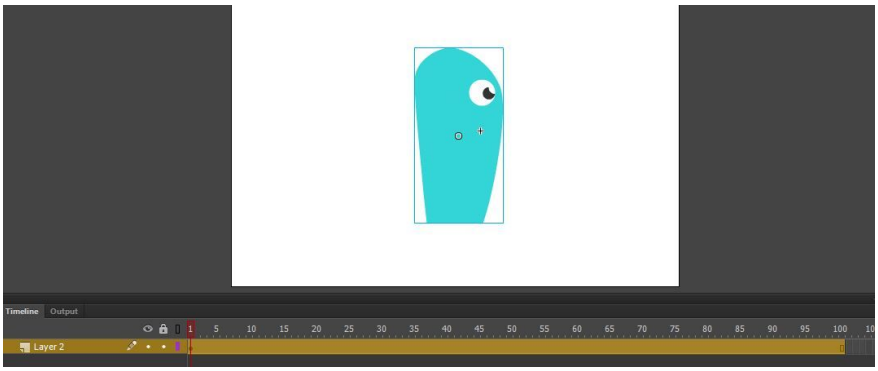


We need to bring in and create our animations if we haven't already done so, in this case I have

Idle, walk, duck, duck idle, duckWalk, hit, & jump.

I created all these anims in flash,& pasted them all along one timeline

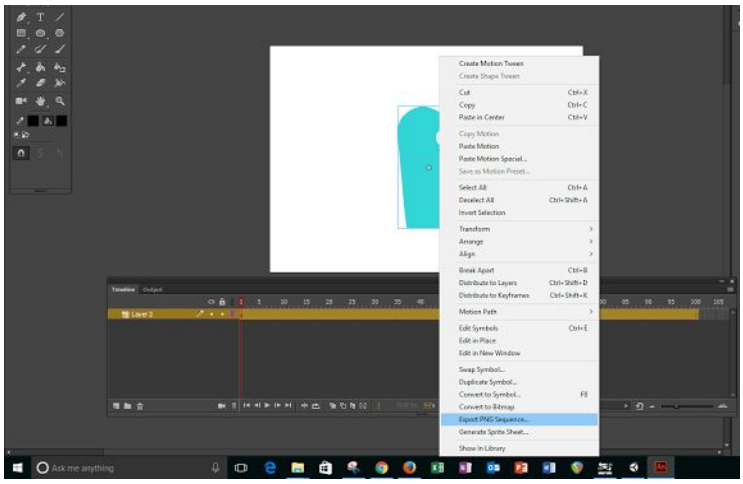




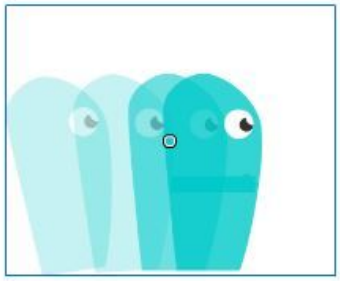
Then popped them all into a movie clip symbol.

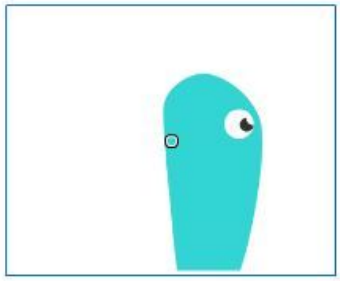
Then I could right click the symbol and choose

Export PNG Sequence



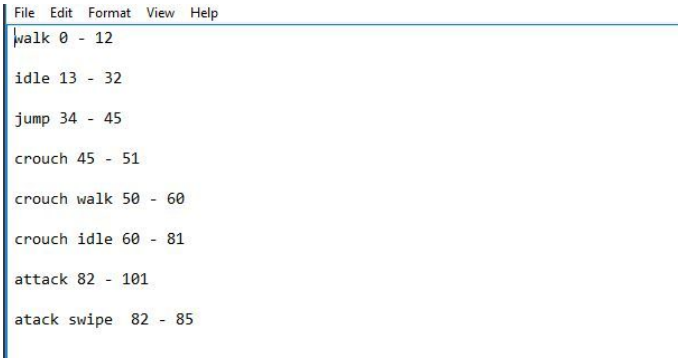
Using this method *(exporting all my sprites from one movie clip*) meant all my sprites are registered from the same spot,





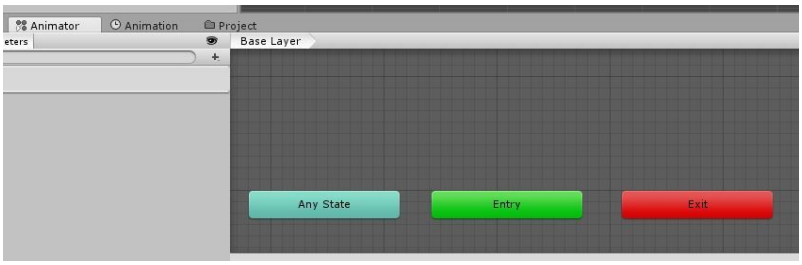
also they are all exported at the size of the largest sprite in the sequence, this prevents registration and popping issues when playing different animations in Unity later.

You may need to tweak your sprites in flash to get your registration point dead center for flipping later depending on your needs..



I also made a note in notepad of the length and position of all my frames along the timeline to easily slot them into separate animations later.

Creating an animation in Unity.

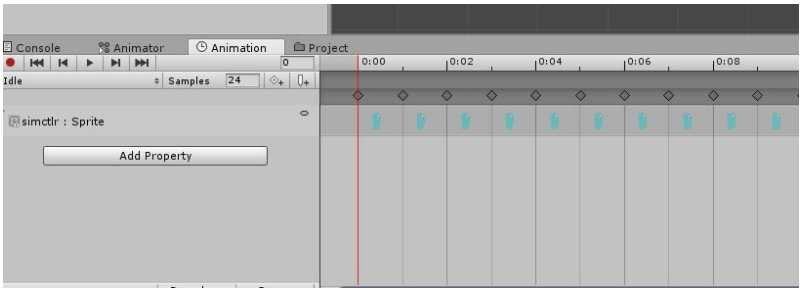


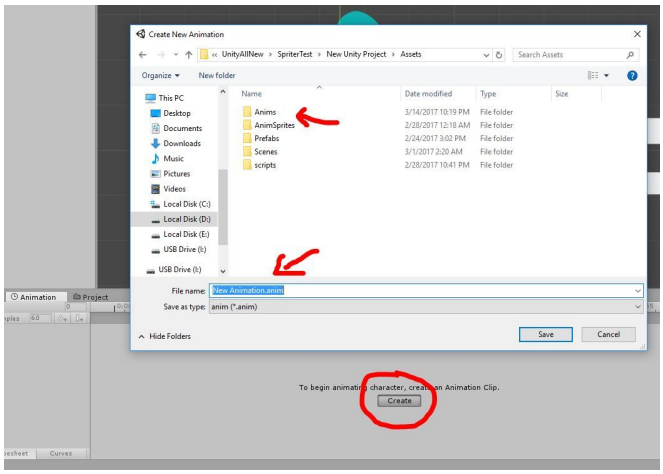
We need to create our animations in unity in the **Animation** window as opposed to the **Animator** window where we will set up our logic and anim trees later.

If you can't see these windows go

Window - Animation &
Window - Animator

in the top toolbar and dock them somewhere convenient for you in the interface.



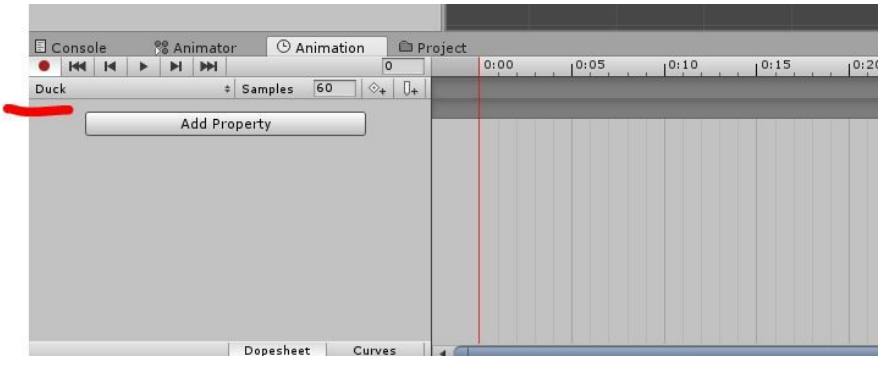


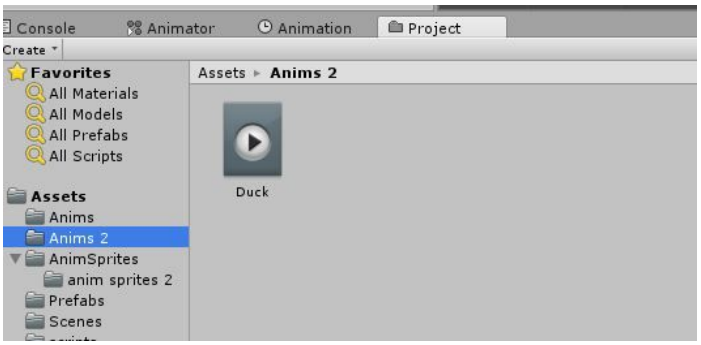
To create an animation, go to the animation window while your character gameobject is selected and choose create.

Name the animation you want to create and save it in your Anims folder in your Unity project.

Now you can see the new animation in the Animation window, in this case i've created the Duck anim.

Here I can see the name, the Samples which is the frame rate I can set, icons for adding key frames and events along the timeline and a button for adding new properties.

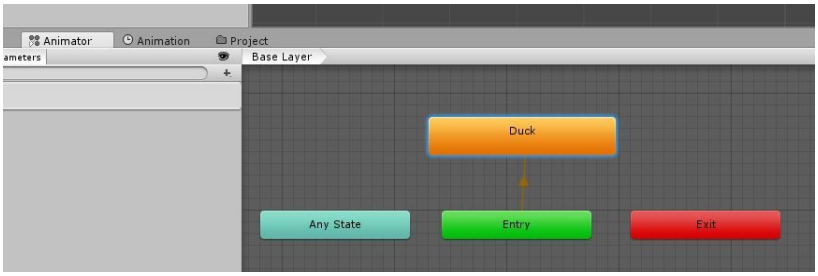


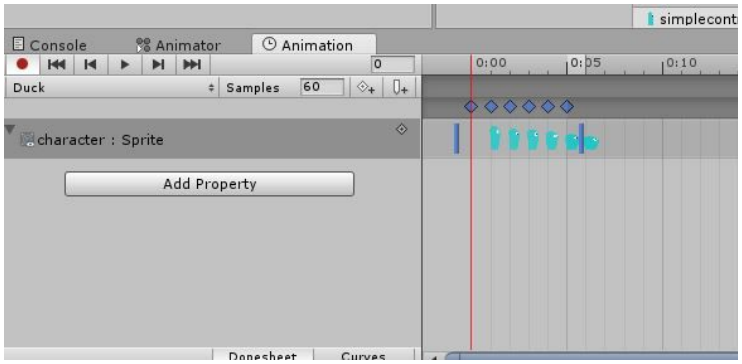


I can also see the new Duck animation in the project view in the anims folder

And it's automatically been added to my Character ctrl Animator window.

So long as my Character ctrl Animator is attached to the Animator component, that's attached to the character gameobject.

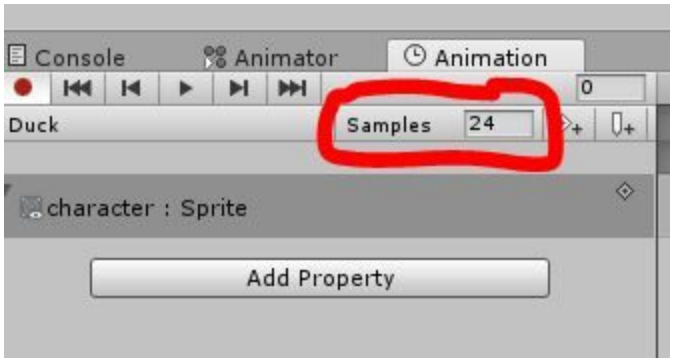


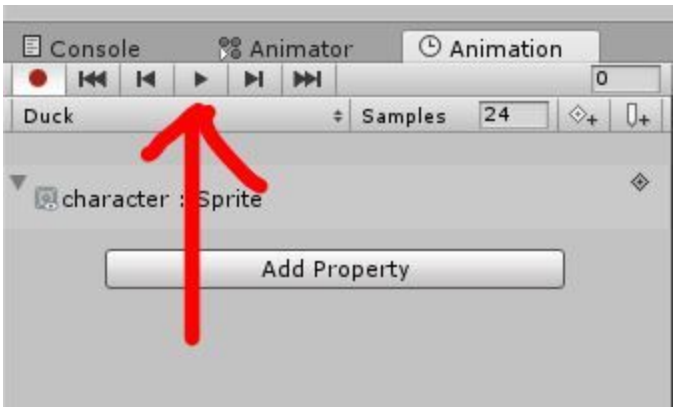


Now we want to add the sprites and set the frame rate.

Navigate to the folder containing your character sprites, check the notepad doc to see what numbers are contained within the Duck animation, select them all and drop them into the timeline in Duck Animation in the Animation window.

My animations were created at 24 fps so ill set my samples in the Animation Window to 24





If you press the play button in the animation Window you can see the animation play in the scene view,

Make sure it's doing what you expect it to do and that no extra or empty frames have made it onto your timeline.

When you're happy your animation is working correctly, you can create your next animation the same way by clicking the drop down menu where the animations are named in the Animation window, and selecting

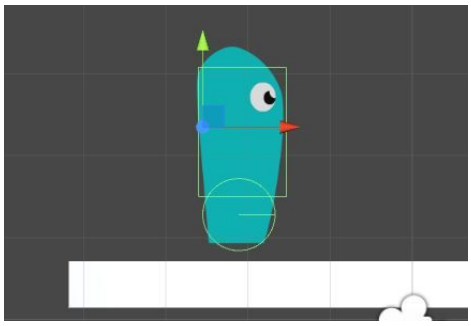
Create New Clip

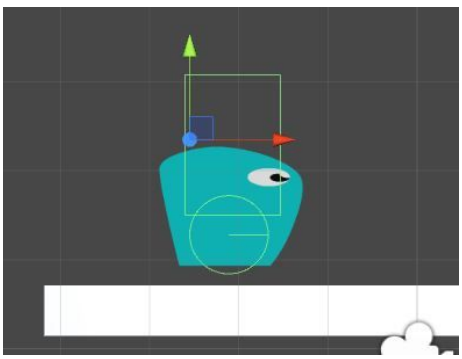
Animating Colliders.

In some cases, like this Duck animation, you may want the colliders we set up for the character Gameobject to change as the animations play and the gameobject takes on a different form.

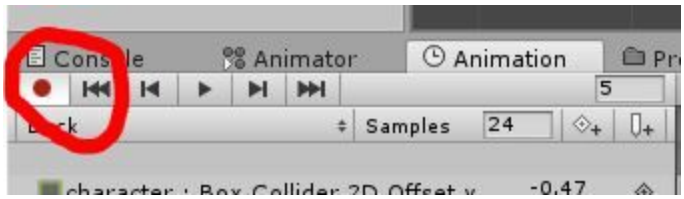
Here we can see the Duck animation at the first frame and again at the last, the green colliders no longer fit the gameobject on the last frame.

In this case he gets smaller. We can animate the colliders to match the new size of our game object in the animation view too.

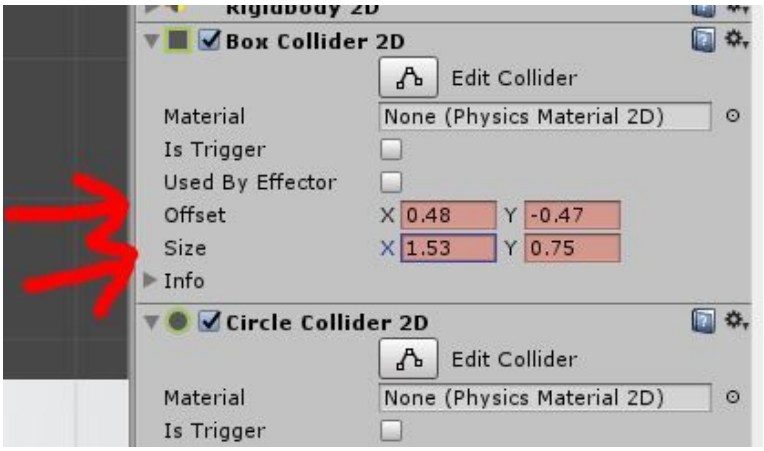




When we create a new animation, the Record function in the Animation Window is on by default, you can tell as it's red and also the play buttons at the top of the interface used to go to game mode are also bright red.



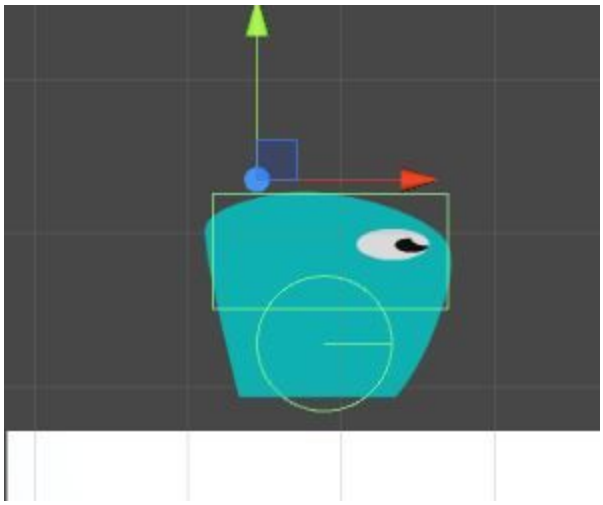
This means Unity is recording whatever you do in the editor and will repeat it as part of the animation you created.



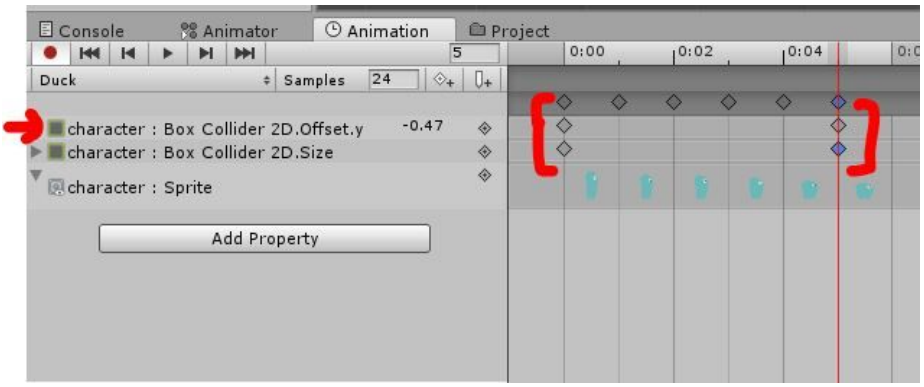
We can use this to animate our game objects Box collider, **make sure the timeline slider in your Animation window timeline is on the last frame of the animation** & hit record.

Now adjust the Offset and Size settings in the Box Collider 2D to fit the character's sprite at this frame.

Tweak the collider until you get a position that suits your gameobjects new sprite. Then turn off the record button.



When you look at the Animation window with the Duck animation selected now, you can see there are two new Properties for the box collider.

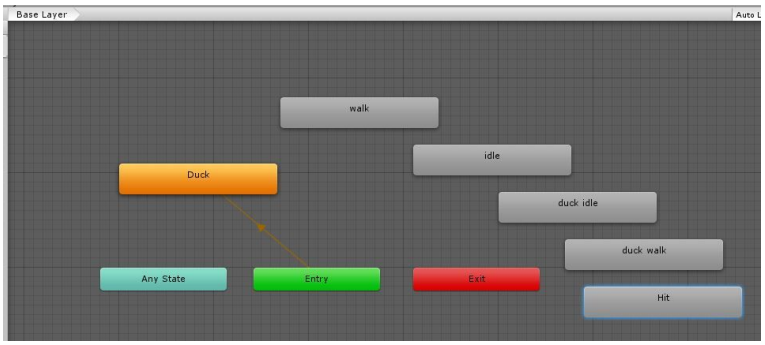


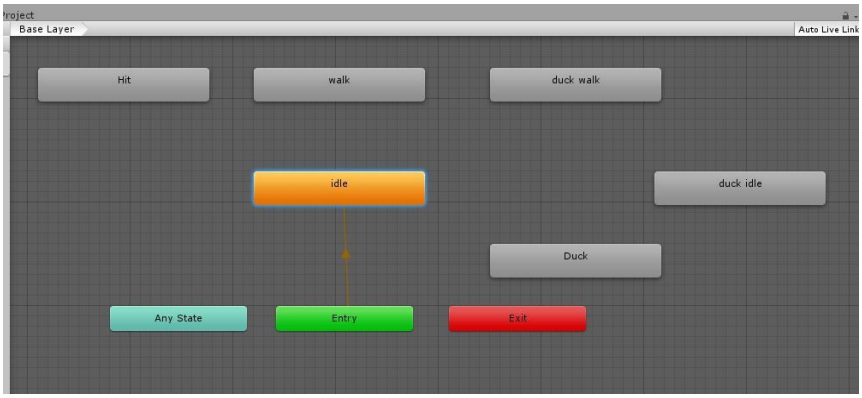
There are also new keys in the timeline for them, one at the first frame then it tweens to the new position we created at the end as the animation plays.

If you play the animation now you will see the collider animate with the sprite.

Setting up the Animation States

Having created all my animation *(except jump which we will come back to later)* we can now go into the Animator window and view all the states that have been created. As we created the Duck animation first it's orange and connected to the entry state by default.



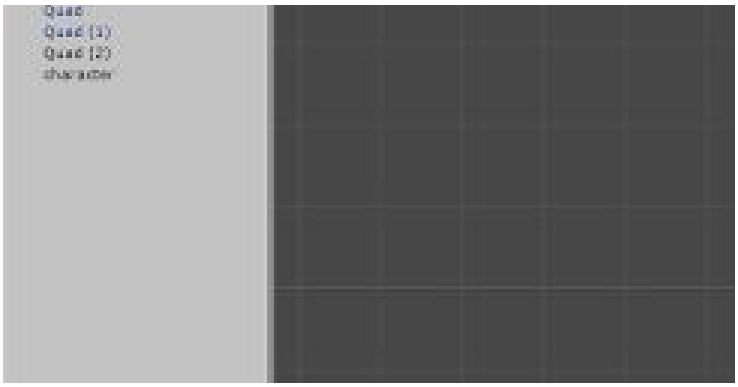


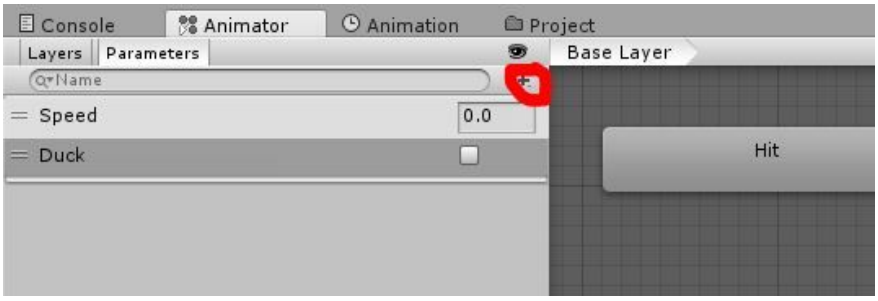
I want my default state to be Idle so i can just right click the idle state and choose **Set as Layer Default State**

Now we're ready to plan out the logic for the State

Machine, but first we need a couple of parameters.

There are 4 types of parameters in the Animator we can use to trigger our animations.
Float (a number with a decimal point)
Int (a movement of 1 whole number)
Bool (returns true or false)
Trigger (Returns true and goes back to false straight away)



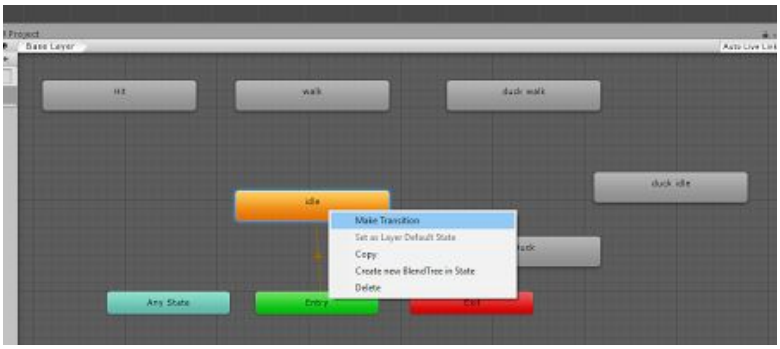


We can **Set** and **Get** these parameters in the animation controller script later, for now we will just create two to enable us setup our logic in the state machine.

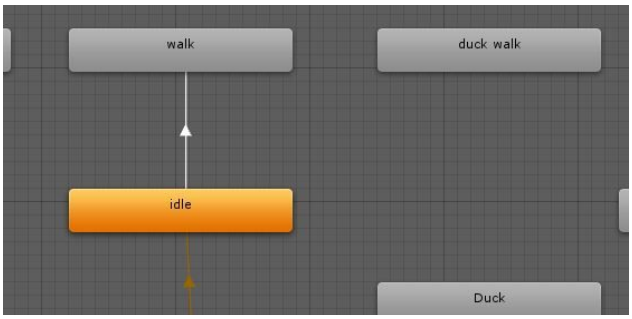
We need one Bool for our duck animations named "**Duck**" and one float for our horizontal movement (walk) animations named "**Speed**" we can create new parameters in the Animator window by clicking the small + icon in the toolbar *see above*.

We can now set up our Logic, we need to think about how we want to move between the different animations and then set up the transitions. We can then place our parameters on those transitions to make sure we only move from one animation to another animation when the conditions we set are correct.

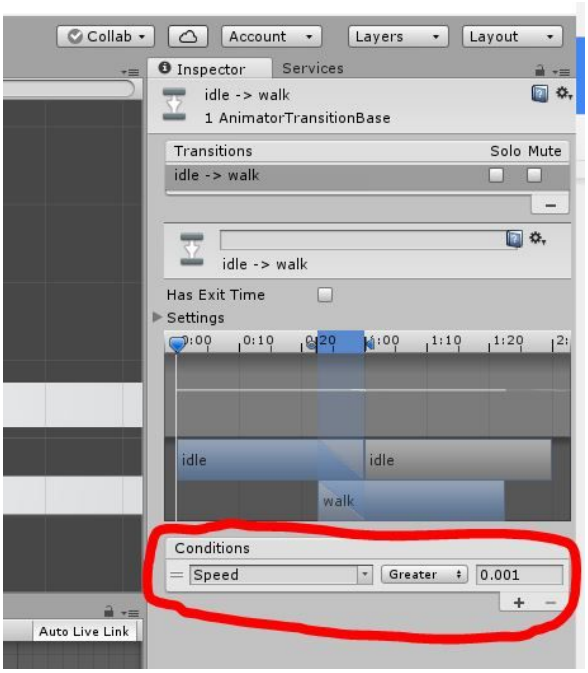
I know I want to transition from idle to walk so i'll make a connection between the two states by right clicking on idle and choosing **Make Transition**



Then dragging the transition line across to the walk State.



Now we need to attach some parameters to the transition, we can do this by clicking on the white arrow and going to the inspector.



In the transition settings in the inspector, we can uncheck **Has Exit Time** as we are going to use one of our parameters that we set up to dictate when we transition from idle to walk, not a set time.

At the bottom of the window where it says **conditions** click the + icon and it will allow you chose between the Duck and Speed parameters we created.

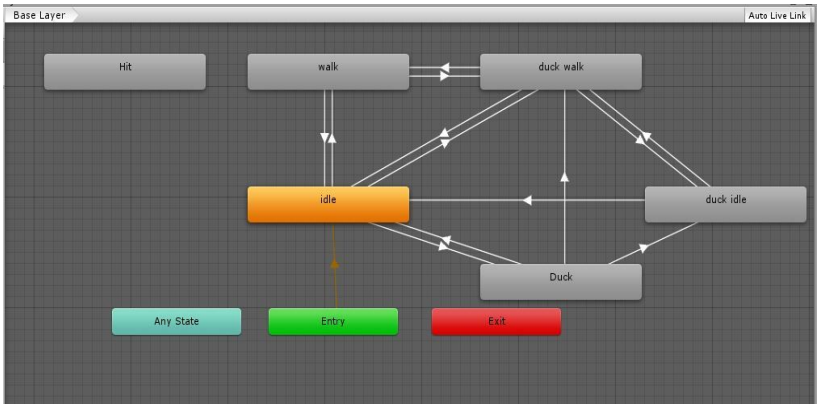
As this is moving from a stationary idle animation to a walk choose the **Speed** parameter, as its a float set it to **greater than .001**.

Now we can set up the rest logic in the animator for our Duck, Idle and walk states

Transitions	**Parameters**
Idle to walk	Speed greater than .001
Walk to idle	Speed less than .001
Idle to duck	Duck = true
Duck to idle	Duck = false
Idle to DuckWalk	Speed greater than .001 & Duck = true

Duck walk to idle	Speed less than .001 & Duck = false
Duck to duck idle	Duck = true
Duck idle to idle	Duck = false
Duck idle to duck walk	Speed greater than .001 & Duck = true
Duck walk to duck idle	Speed less than .001 & Duck = true
Duck walk to walk	Speed greater than .001 & Duck = false
Walk to Duck Walk	Speed greater than .001 & Duck = true
Duck to Duck Walk	Speed greater than .001 & Duck = true

So we end up with this.



Most of the states have two way transitions with each other except Duck and Duck idle & Duck and duck walk.

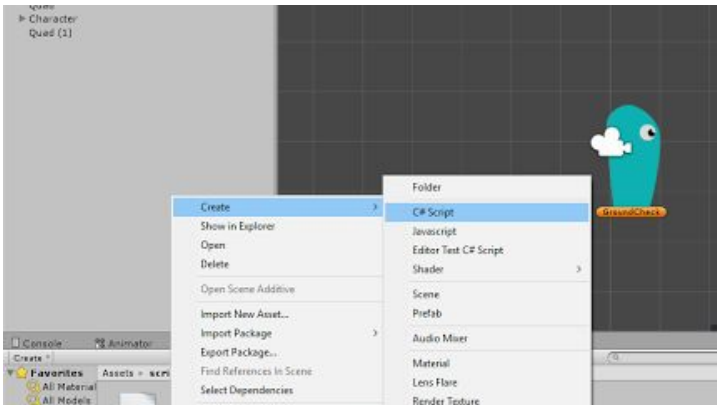
this is because Duck is an “inbetween” animation that moves from the idle position to the duck position,

It doesn't work going the other way.

If the animation doesn't make it the whole way through the state and get to duck idle before the Bool becomes untrue, then it will pop back to idle from duck (otherwise it would have nowhere to go), but once it gets to duck idle I don't want to travel back through it again on the way back to the idle state.

To get this Animator working on our Character Gameobject we need to set our player

input, movement/speed &
set the parameters in our Animator
so we need to add a script.



Create a scripts folder in your project window then right click and
Create - C#Script
Double click on it and it should open in monodevelop.

Script

First we will set up moving left and right, the speed and set up the “Duck” animations to work. We need some variables and references.

//this is setting the maximum speed for the character, you will see it in the editor and can adjust it there if u wish.
public float maxSpeed = 10f;

//setting a float for our horizontal movement input.
float move;

// Setting a reference for our Animator
Animator anim;

// Setting a reference for our Rigidbody
Rigidbody2D rb;

//creating a bool to set input for our duck animation
bool Dodge;

(I name the bool for the **duck** animation as **Dodge**, it's good practise to do this as i may want another animation in my animator to play **from the same input** that i'm using for the duck animations, ie a dashing animation while in the air, so i would need another parameter to set this up in my animator but could use the **Dodge** bool and input to trigger it.)

```
// initialising my references to the animator and to the  rigidbody 2D.
void Awake () {

        anim = GetComponent<Animator> ();
        rb = GetComponent<Rigidbody2D> ();
    }
```

//Using fixed update as i'm scripting things using the physics engine

```
void FixedUpdate () {
/*
```

setting the input to the "move" float i created in my variables. Using the axis as opposed to the GetKey which is more restrictive, to view the pre existing axis in unity go to Edit-Project Settings-Input and look at the inspector to get the names to use in the script.

```
*/

move = Input.GetAxis ("Horizontal");

/*
```

If the "joystick" was moving left on the horizontal axis then my move variable would be a number between 0.0 and -1.0 i.e. a negative number. If it was moving right it would be between 0.0 and 1.0 i.e. a positive number.

When I call the function Mathf.Abs(move) below it returns the value of move as a positive number if it was negative (otherwise it leaves it alone). So if move was -0.5 then Mathf.Abs(move) would give me back 0.5.

I always need it to be positive so that I can correctly trigger state transitions.

```
*/


        anim.SetFloat ("Speed", Mathf.Abs (move));
```

/*
I need to set the velocity on the rigidbody. The velocity has an x part and a y part which indicate the end point of the vector2, I am going to set the x part (i.e. the x speed) to maxSpeed * move. I have set maxSpeed to be 10 and move can have a value between -1.0 and 1.0 so the x speed will be somewhere between -10 and 10 depending on the value of move (which in turn is dependant on the "joystick" position"). The y part of the vector (i.e. the y speed) I set to whatever it currently is i.e. I don't change it.
*/

```
rb.velocity = new Vector2 (move * maxSpeed, rb.velocity.y);
```

- Now the character game object should move left and right based on the input linked to the horizontal axis (arrow keys and game controller stick)

Setting the Duck animations to work based on the Dodge bool set in the variables

//setting the value of Dodge, using the axis "fire1" from the input manager.use getbutton as opposed to getkey to use an axis from the input manager. Didge will be true of false.

```
Dodge = Input.GetButton ("Fire1");
```

/*
write an **if statement**, if Dodge is true, ie if there **is** input on the appointed button "fire1" set the bool "duck" in our animator to true.
OR
if Dodge is false (there is no input) set the bool "duck" in our animator to false.
*/

```
        if (Dodge == true) {
            anim.SetBool ("Duck", true);


        } else if (Dodge == false) {
            anim.SetBool ("Duck", false);
        }
}
```

Save the file, go back to the editor and attach the script to out character gameobject by dragging it from the scripts folder and placing on top of the character in the Hierarchy window.

Everything works, the character should move left and right, play the walk and idle anims, duck, duck idle and walk idle anims based on input.

Flipping the Character Gameobject

We need a new variable

```
//We are automatically facing right!!
bool facingRight = true;

// create a New Function, (outside fixed update or update). This will be called when we are doing
// a flip
void Flip () {
        // set facingRight to be equal to not facingRight (we have just toggled the facingRight
        // bool)
                facingRight = !facingRight;
        // Get the local scale
                Vector3 theScale = transform.localScale;
        // change the sign of the x axis be multiplying it be -1. Anynumber multiplied by -1
       // changes its sign (i.e. whether its negative or positive)
                theScale.x *= -1;
        //apply it back to the local scale
                transform.localScale = theScale;

        //all together we're taking the gameobject and flipping it on the y axis
            }
```

Now link it in our Fixed update after we move!!

```
        // if move is greater than 0 and we are not facing right flip.
            if (move > 0 && !facingRight)
                Flip ();

        // else if move is less than 0 (moving in the negative direction - left) and we ARE facing
        right flip.
            else if (move < 0 && facingRight)
                Flip ();
```

Not on the ground Blend Tree animation (and jump)

Start in the script!

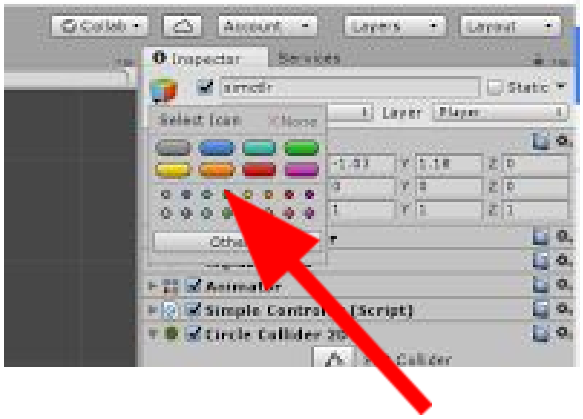
```
//Bool to tell us if he's on the ground (mine starts in the air so it's false now)
    bool grounded = false;

//Transform to tell us where the ground is.
    public Transform groundcheck;

//Same as a collider, and how big it's going to be.
    float groundRadious = 0.2f;

//Telling us what is considered ground through the layers system.
    public LayerMask whatIsGround;

//Strength of the jump
    public float jumpforce = 700f;
```

Inside fixedupdate, constantly checking if we are on the ground or not.

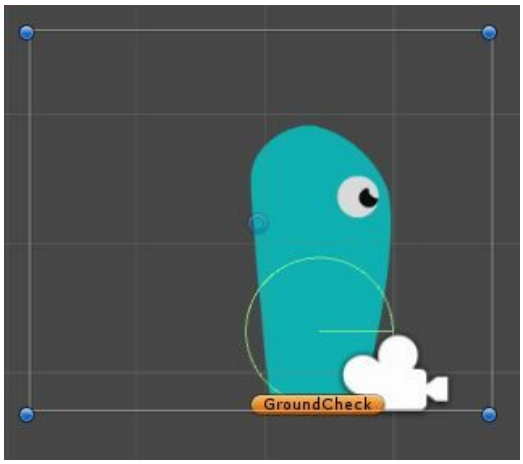
```
//Our grounded variable (bool true or false) =

//Phisics2D.OverlapCircle (part of unity api that simulates a collider)

//groundcheck = are transform and we are finding its position,

//setting the size of the circle,

//and telling it what is ground through the layers system (layer Mask), ie what is ground.

    grounded = Physics2D.OverlapCircle (groundcheck.position,
    groundRadious, whatIsGround);

//Setting our parameter in the animator (we haven't created yet)
    anim.SetBool ("Ground", grounded);
```

Save and go to the editor

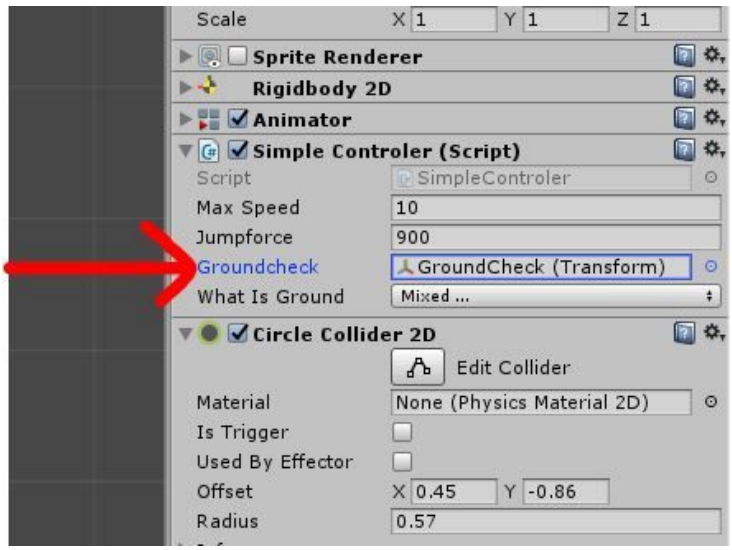
Create an empty game object,
Name it groundcheck,
Nest it into your character game objects hierarchy.



Give it an Icon so we can see it in the editor,

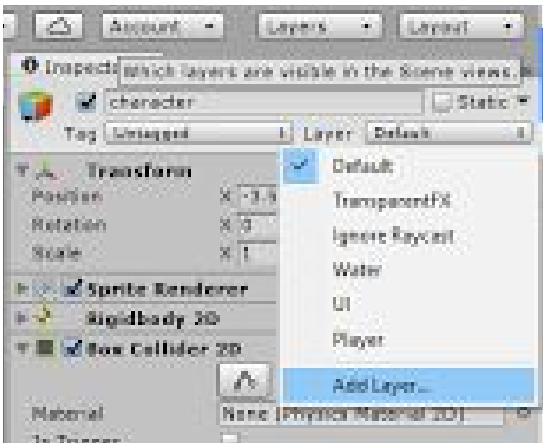


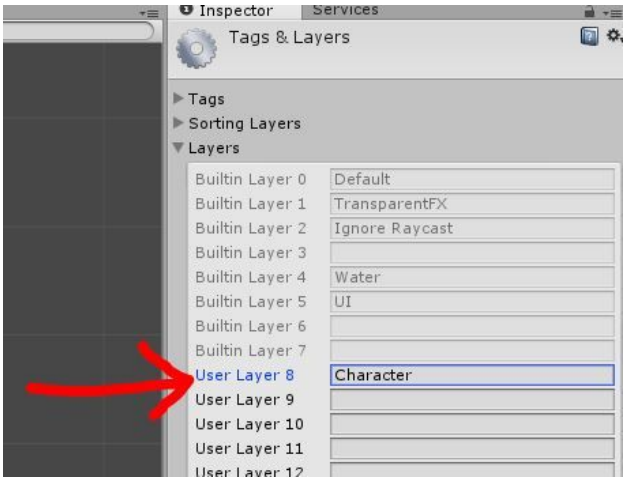
Place under our character gameobject where you want it to contact the ground,



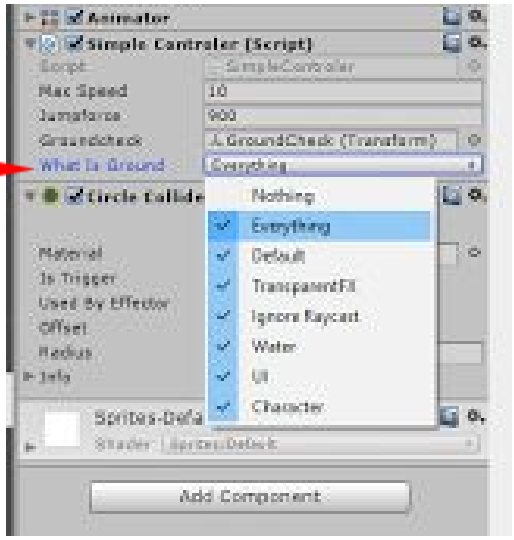
Then grab the the groundcheck gameobject and drop it into the Groundcheck transform slot that's now in the character gameobject's script.

We need to set the character gameobject on its own layer to set up the layer mask.
With the character selected go to the layers drop down and choose Add layer if you want to create a new character layer.



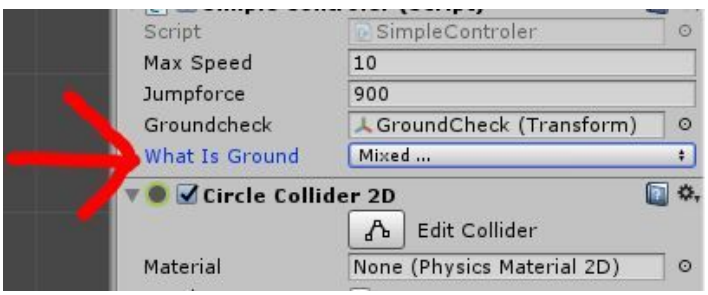


Create a new layer for your character in the Tags and Layers window that opens up.



Then in the “What is Ground” public layermask that's now in the characters script, click the drop down and select “**Everything”**

Then click the drop down menu again and **CTRL CLICK** on the Character layer to remove it from the list.

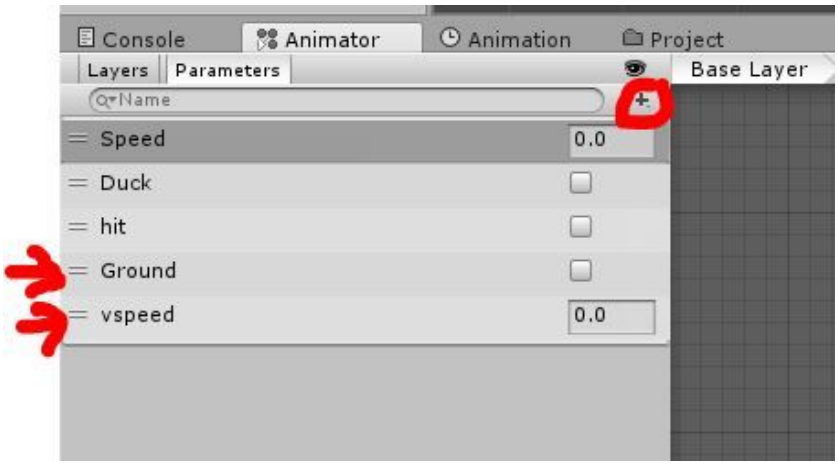


The “**what is ground**” Layer Mask should now say it’s “**mixed**”

Now we need to set our new “**Ground**” bool parameter in the Animator

We can also set a new float parameter called “**vspeed**”

We will use this to link our jump blend tree to the vertical speed on our Rigidbody2D.



Back in the Script!!!

In our fixed update, under our anim. Setbool (“Ground”, grounded); line of code.

//setting the float in the animator “vspeed” to the rigidbody 2D’s velocity in its Y axis.

anim.SetFloat ("vspeed", rb.velocity.y);

Outside fixed update, Start a new Update (So the input is more accurate!).

```
void Update () {
 // Set the input with an if statement,
// if grounded is true and the input is pressed (using the jump axis in the input manager)
                if (grounded && Input.GetButtonDown ("Jump"))
                        {
//Set bool parameter “Ground” in the animator to false.
                                anim.SetBool ("Ground", false);
//Add force to the rigidbody 2D,  vector 2 as its 2D,
// 0 for the X axis and using the  variable we set “jumpforce” on the Y axis.
                                rb.AddForce(new Vector2(0, jumpforce));
                        }
                }
```

Save and check the character gameobject is now jumping in the editor,

space is the default key in the “jump” axis in the input manager.

The “Ground bool in the animator should also be checking on when the character is on the ground and off when it is in the air,

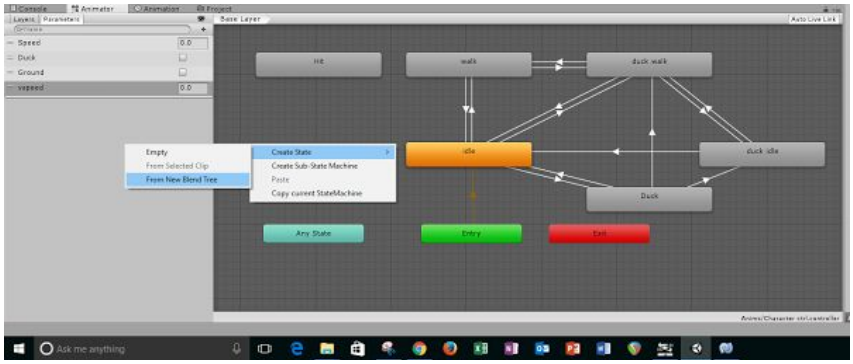
so if we walk off a ledge without jumping we know we are in the air and we are recording the vertical speed in the “vspeed” float in the parameter when we are not grounded.

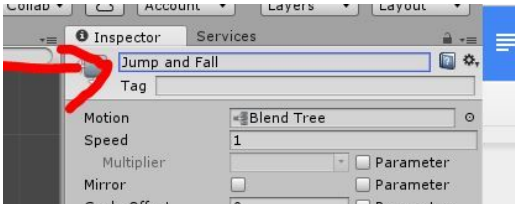
Back in the Editor

Setting up the jump and fall blend tree.

We need hook up the jump blend tree in the animator, right click in the window and choose

Create State - (hover over)
From new Blend Tree

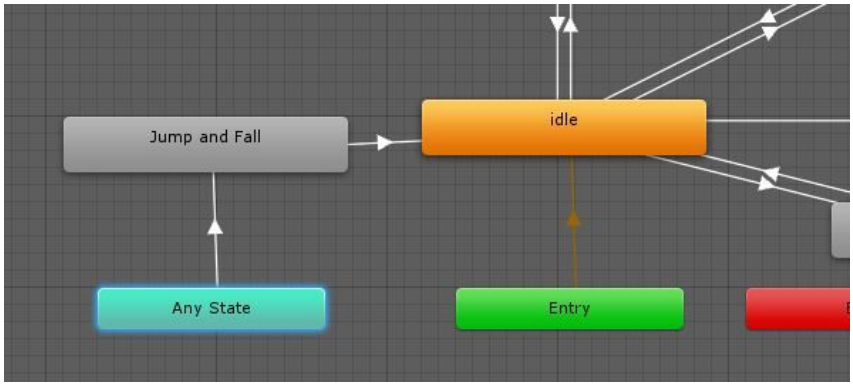




Name it Jump and Fall in the inspector.

Set up transitions from the **Anystate** to the **jump and fall** state.

And from the **jump and fall** state to the **Idle** state.



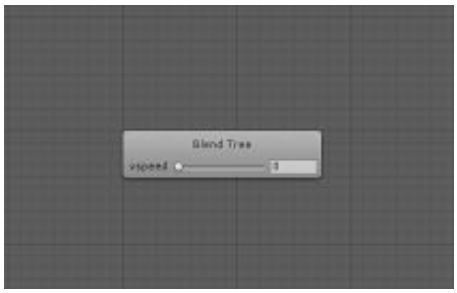
Transitions	Parameters
Any State to Jump and Fall	Ground = false
Jump and Fall to Idle	Ground = true

We can go into **jump and fall** from any other state (once we are not on the ground)

And we go from **Jump and Fall** to **idle** once we become grounded again,

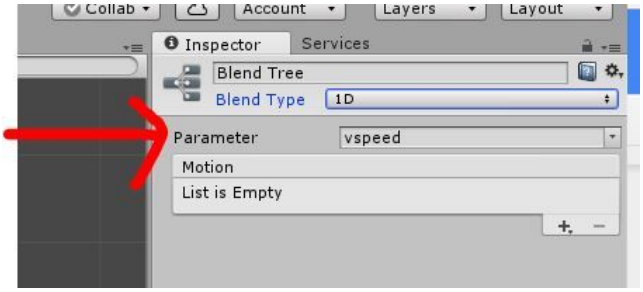
From **idle** we than have access to all the other logic already set up.

Double click into the Jump and Fall state to see the Blend Tree.



You can tell you inside another state by looking at the top of the window, like the **nesting** symbol system in flash, we can see here we are inside the Jump and fall state and that there is a Base layer up on level.

Set the blend tree as **1 D** *dimensional* in the inspector, as we are only blending from one parameter **vspeed**.



Set the parameter to **vspeed**.

In the Animation Window

In this case I'm using a jump animation with 11 frames in it,

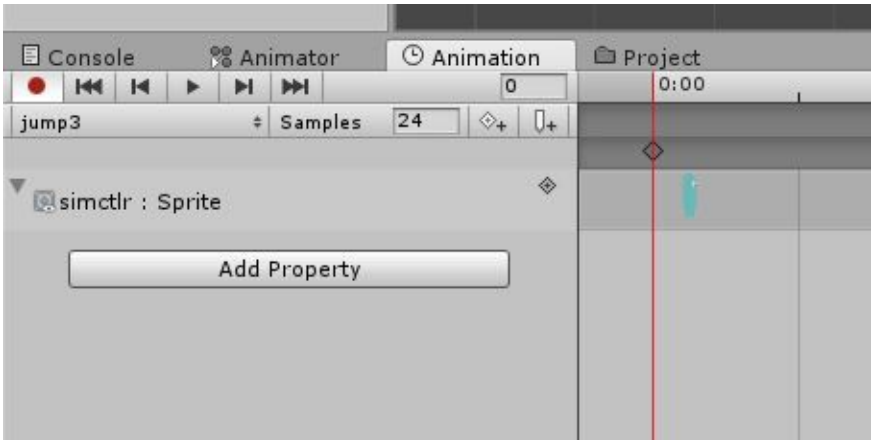
If you want to set your animation up with more or less that's fine, you just need to test it to figure out what numbers to assign the frames in the tree.

The minimum would be two frames, one up, one down, so greater than .001 and less than .001 in the vspeed float parameter.

The same way we created the animations before,

Create 11 animations, one sprite per anim.

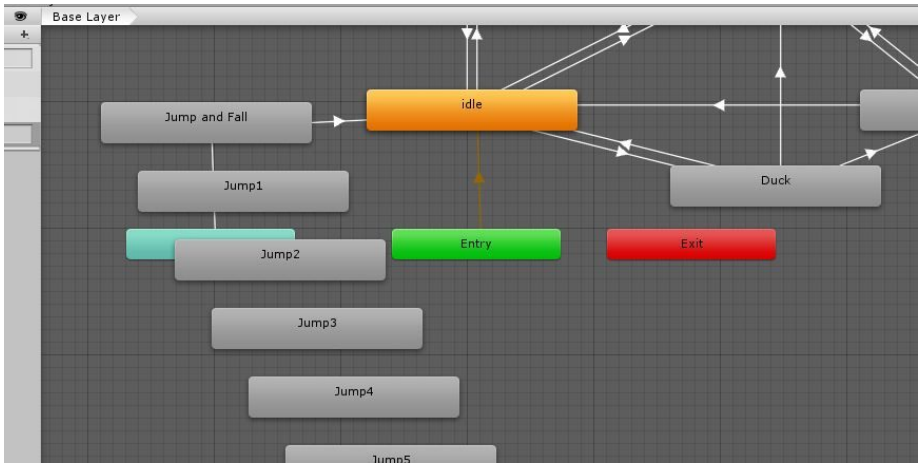
Name them **Jump1** - **Jump2** etc

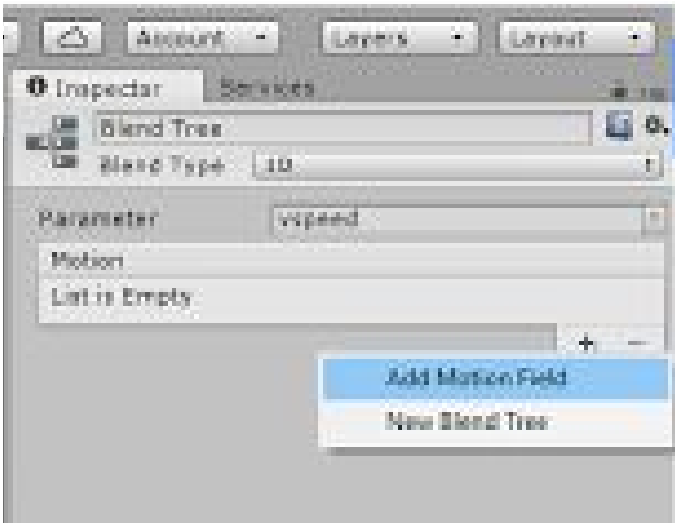


Save the jump anims into your anim folder in the unity project with the rest of the animations and the Character controller.

The anims will also automatically go into the Animator window as states.

We don't need them in here so just go in and delete them.





Click into **Jump and Fall** to get into the blend tree,

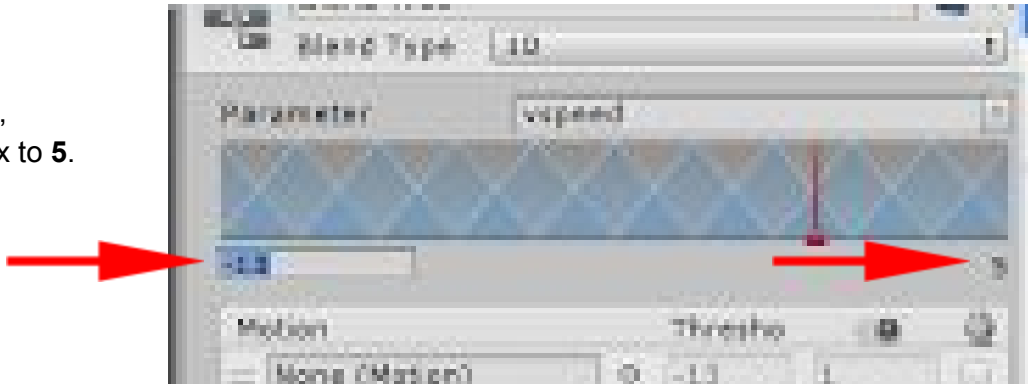
In the inspector (with the blend tree selected)

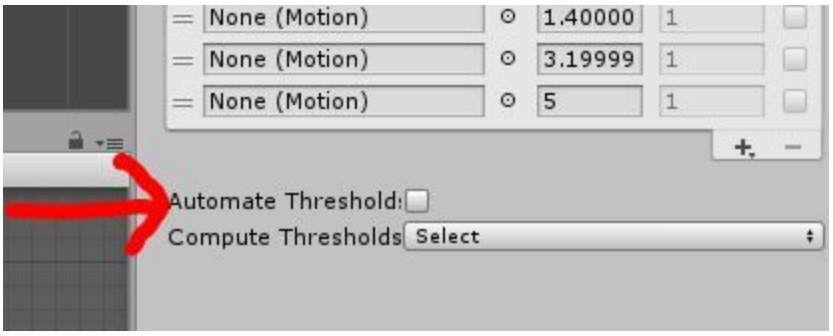
under the **Motion** tab, click the little plus icon.

And choose **Add Motion Field.**

Create (in this case) 11 motion fields.

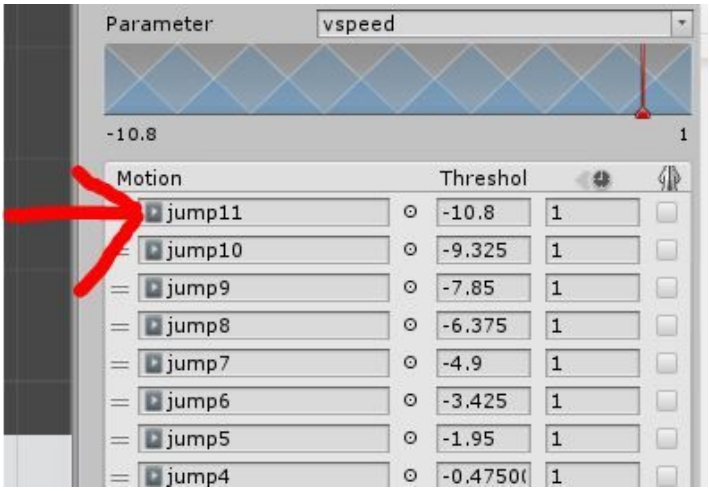
Under the motion field,
set min to **-13** and max to **5**.

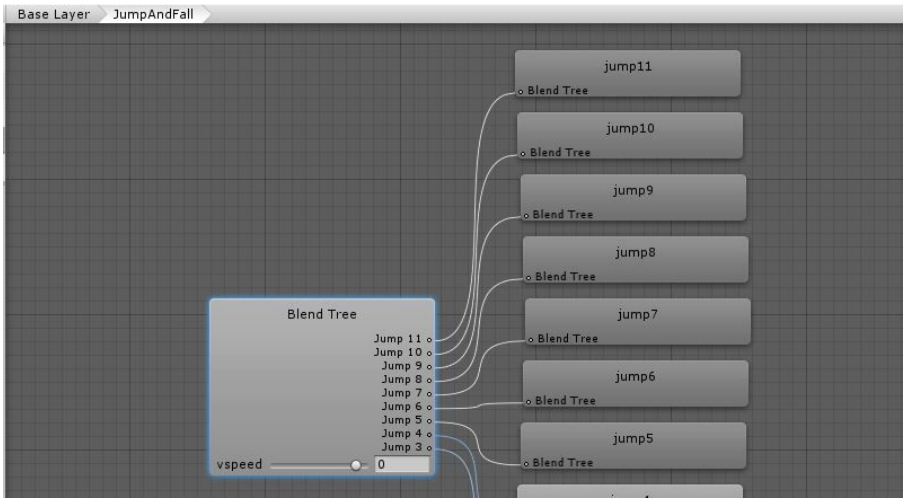




Uncheck
Automate Threshold
at the bottom below the new motion field slots.
(Unless you want to let unity make a guess.)

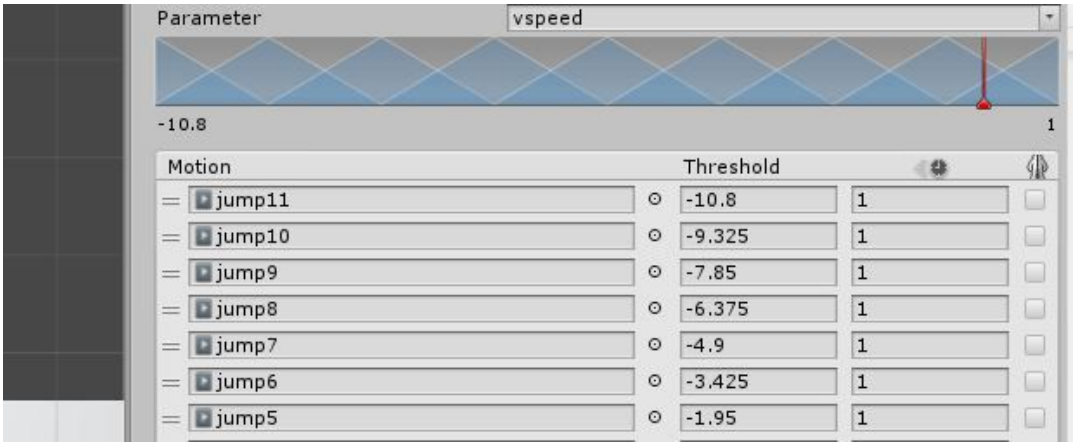
Click and drag the 11 animations from the project window anims folder, up into the motion field slots.





We can see the animations have been added to the blend tree now in the Animator Window.

Now we need to add in the numbers into the Threshold boxes for each motion field, ie at what vertical speeds do u want the particular anims to blend at.

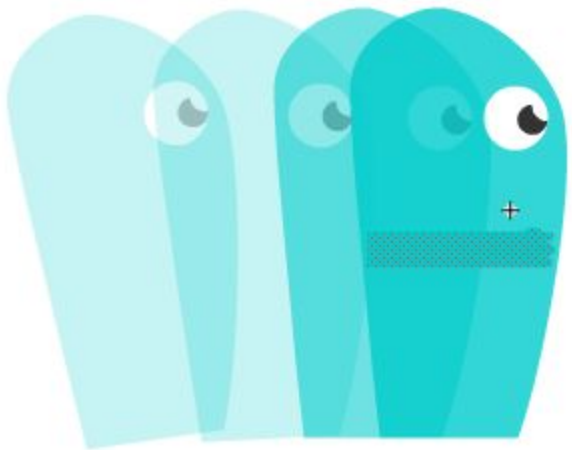


This is a testing thing that you will need to work out for your own character based on jump strength, gravity, amount of frames etc.

After following a tute and some tweaking and messing mine ended up something like this, but it still needs work.

-10.8 -8.11 -7.3 -6.2 -5.4 -4.6 -3.7 -2.9 -1.7 -0.03 2.16

Setting up the Hit Animations.



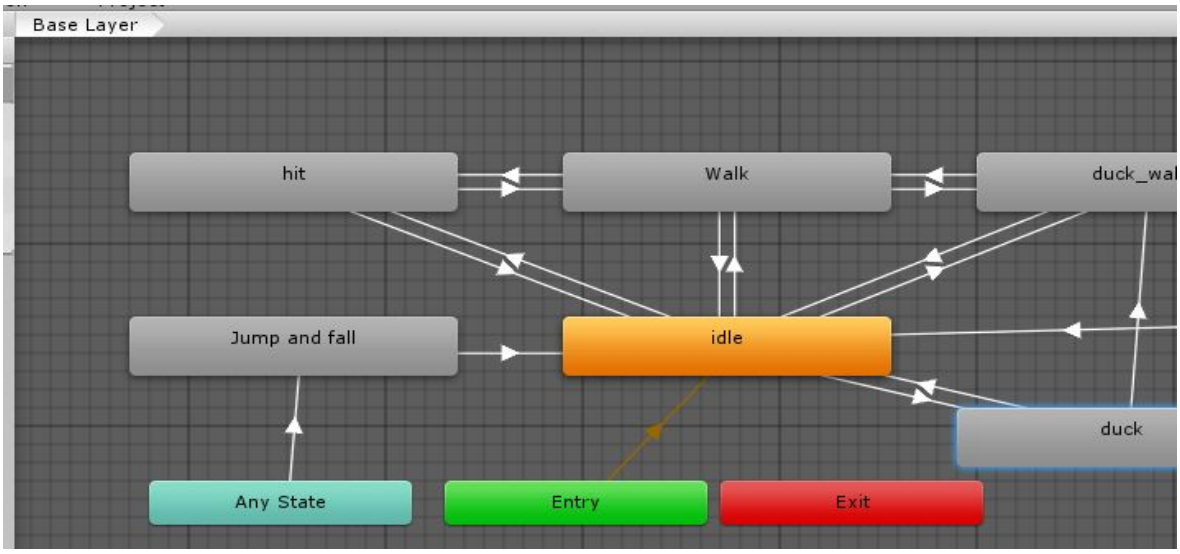
The Last animation we need to set up in the logic and the script is the hit animation.

The character has one hit animation, it's about 20 frames long at 24fps

Its animated on the spot and I faked a motion blur on the first few frames.

In the Animator

I've also applied a **Force** (similar to a jump) to the hit or attack input, and I've increased the force if the character is moving forward when the attack input is triggered.



Also I've set it up so you can only attack when in **idle** or **walk** states, **not** from any Duck states or if the character is off the Ground.

So I only need transitions from my hit state two and from the **idle** state and the **walk** state.

I've also set up a new bool parameter in the animator named "hit"

Transitions	Parameters
Idle to hit	Speed = Less .001 & hit = true
Hit to idle	Speed = Less .001 & hit = false
Walk to hit	Speed = Greater .001 & hit = true
Hit to walk	Speed = Greater .001 & hit = false

In the Script

I've used an if statement to set this up, it got pretty long and you would probably use the level manager to set this up in a more efficient way going forward, in the context of this tutorial and being clear about the logic this is fine for now.

If the first branch of the **if statement** in the list comes back true it won't read the rest, or it will keep going through the rest of the **else if branches** until it hits one that is true, so the order of these is important.

```
//Create a bool variable called Attack
        bool attack;
```

In fixedupdate

```
// setting up the input using GetButtonDown (true then false again as opposed to get
// button which would be true continuously true once pressed)  and using the "fire2" axis
// from the input manager (Alt key by default)
        attack = Input.GetButtonDown ("Fire2");

// If attack (input) is true & Dodge (our duck variable) is false & we are on the ground and not
// moving & we are facing right.
        if (attack == true && Dodge == false && grounded == true && move == 0 &&
facingRight == true) {

//add force to our rigid body 2d, in a vector 2 (2d) 100 strength in our x axis and 0 in the y, and
//the type of force is impulse (all at once as opposed to gradually over time)
                rb.AddForce (new Vector2 (100, 0), ForceMode2D.Impulse);

// reference the animator and set out "hit" parameter to true.
                anim.SetBool ("hit", true);

// Else if everyting is the same but we are NOT (!) facing right.
        } else if (attack == true && Dodge == false && grounded == true && move ==
0 && !facingRight == true) {

// add the force in the negative direction (left) - 100 in the x axis, 0 in the y.
                rb.AddForce (new Vector2 (-100, 0), ForceMode2D.Impulse);

// set "hit to true
                anim.SetBool ("hit", true);
```

// Else if everyting is the same, we are facing right again but move is greater (>) than 0 (so we are moving right).

} else if (attack == true && Dodge == false && grounded == true && move > 0 && facingRight == true) {

// add the force at the higher value of 300 in the x axis, 0 in the y.
rb.AddForce (new Vector2 (300,0), ForceMode2D.Impulse);
// set "hit" to true
anim.SetBool ("hit", true);

// Else if everyting is the same, but we are NOT (!) facing right & move is greater (>) than 0 (so we are moving left).

} else if (attack == true && Dodge == false && grounded == true && move < 0 && !facingRight == true) {

// add the force at the higher value of negative - 300 (left) in the x axis, 0 in the y.
rb.AddForce (new Vector2 (-300, 0), ForceMode2D.Impulse);

// set "hit" to true
anim.SetBool ("hit", true);

// Last if the input is not there
} else if (attack == false) {

// add no force
rb.AddForce (new Vector2 (0, 0), ForceMode2D.Impulse);

// and set "hit" to false
anim.SetBool ("hit", false);
}