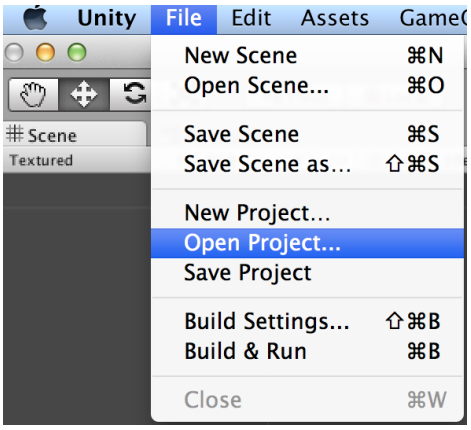
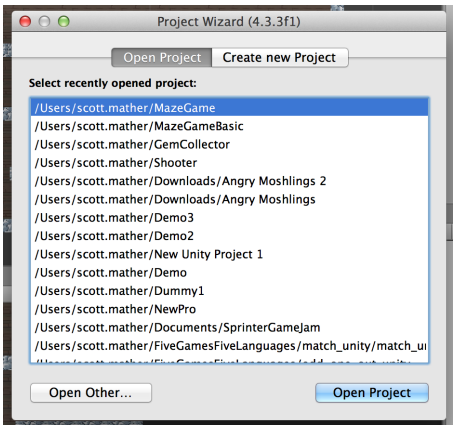
Maze Game

Before you start please make sure you download the necessary project files from the Yammer group (MazeGameBasic.zip)

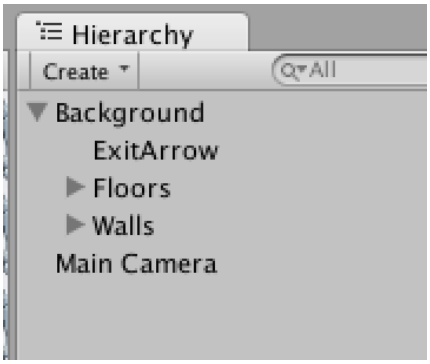
1. Open Unity and select ‘File > Open Project...’ then Select ‘Open Other...’ (bottom left) and select the folder MazeGameBasic.



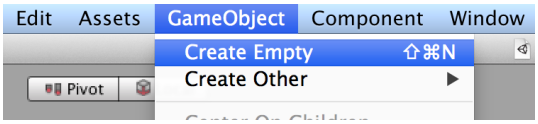


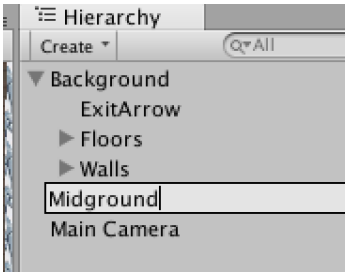
2. If the scene is empty, double click ‘GameScene’ under ‘Assets > Scenes’. You will see that some of the scene has already been set up for you this time. This is to save time so we can concentrate on new features. Currently it’s just basic sprites, the wall sprites have Box Collider 2D’s attached to them.

3. If you look in the Hierarchy panel you will notice that there are two objects: Background and Main Camera. You will also notice that the Background object has a small arrow to the left of it. Click the small arrow to reveal a list of ‘nested’ sprites underneath it. Any GameObject can be nested/made a child of another GameObject by dragging it onto the other object. It is a very useful way to organise the objects in your Hierarchy into folders. With the bonus that if you move/scale/rotate the parent then all of the nested clips will do the same.

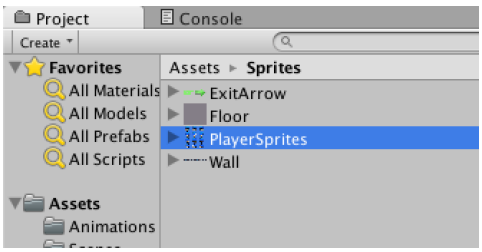


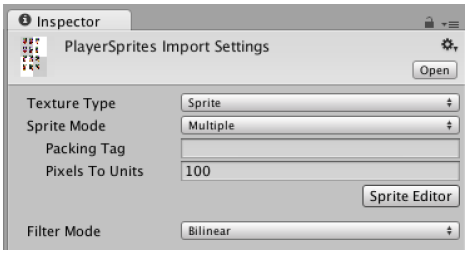
4. We will start by creating a new Empty GameObject that we will use as a ‘Midground’ folder. To do this select GameObject from the Menu Bar and then Create Empty. You will want to rename the object to Midground in the hierarchy so we know what it is.

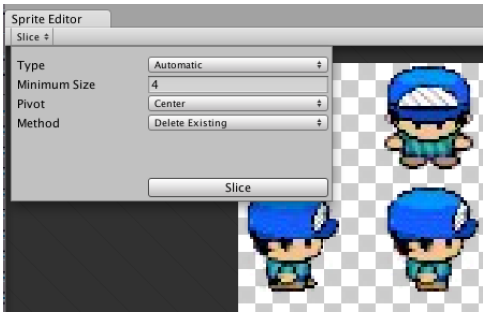


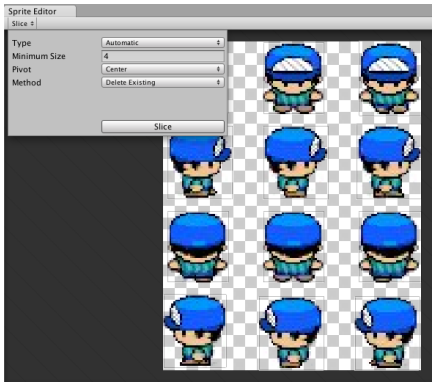


5. Now we will add the player character to the scene. To do this, select ‘Assets>Sprites>PlayerSprites’ and in the Inspector panel change Sprite Mode from ‘Single’ to ‘Multiple’ and click the ‘Sprite Editor’ button. In the Sprite editor click ‘Slice’ in the top left of the panel and then click ‘Slice’ again in the new pop up. (9 rectangles should have appeared around the characters in the image). Then click ‘Apply’ in the Sprite Editor window to save your changes. You can now close the Sprite Editor window.

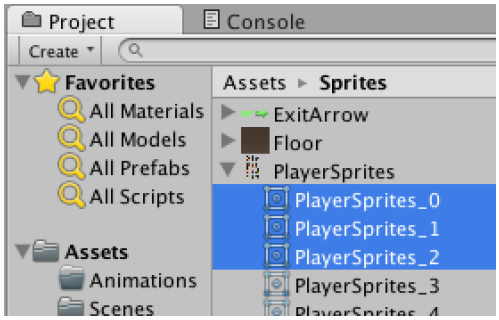


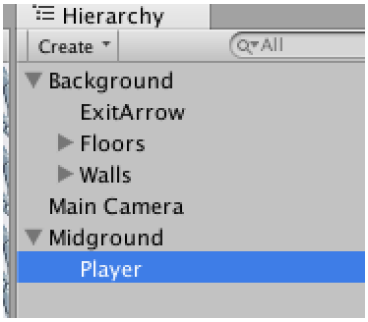




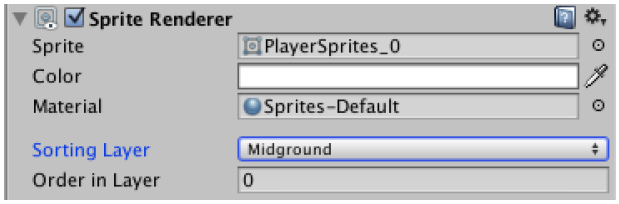


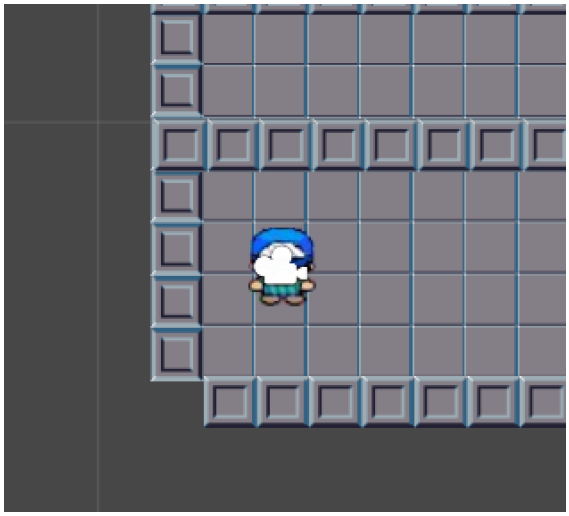
6. Click the arrow to the left of 'PlayerSprites' in the Assets window and you should see 12 sub sprites now. We will just create 1 animation for now, so select the top 3 frames and drag them into the hierarchy. A popup window will appear asking you to name the animation, name it 'Down' and click ok. You will also want to rename the PlayerSprites_0 object in the hierarchy to just 'Player'. And then drag this player object onto Midground so it becomes nested underneath it. You will see that the sprite was added but we are unable to see it in the scene view....





7. In order to view the sprite we need to make it use the new layering System of Unity 2D. With the sprite selected look in the Inspector panel and you will see a field named "Sorting Layer" and it's set to default. If you click the button you will be presented with "Background", "Midground" and "Foreground". For now lets select Midground so that we can see the player in our scene. It's possible to add as many layers to your games as you like and if you have more than 1 sprite in a layer you can control this using the "Order In Layer" field. Note that for the Order In Layer higher numbers are rendered in front of lower numbers. Now that you can see the player you will want to move her so she is positioned where the camera is, as shown here:

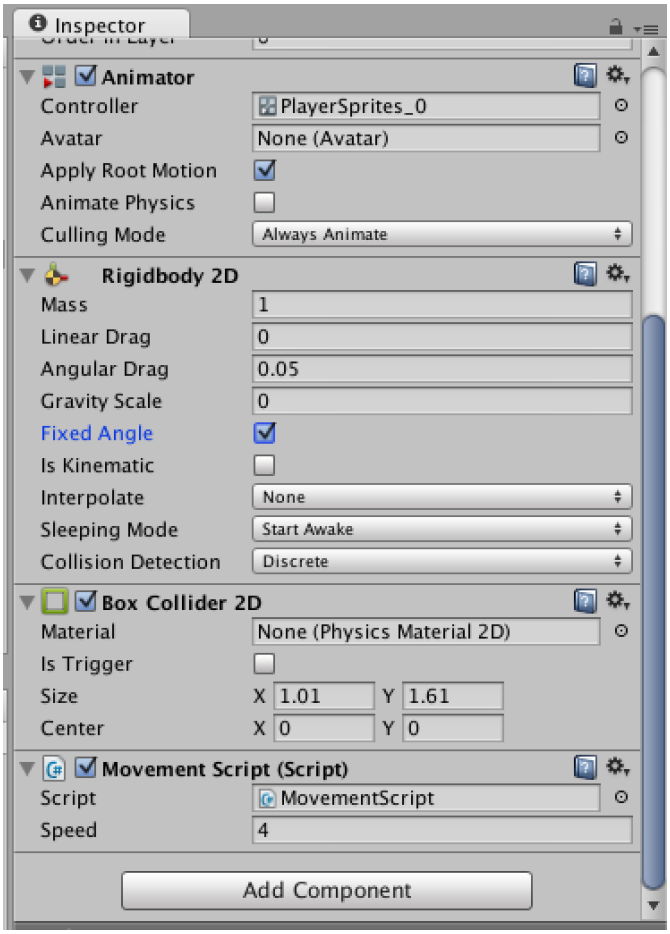




8. It's time to get the player moving so first things first, let's add a 'RigidBody 2D' component to the player. Select the player sprite and click "Add Component" In the Inspector. Search for and add the 'Rigidbody 2D' component. You will want to set Gravity Scale to 0 to prevent the player from moving. You will also want to check the Fixed Angle button to prevent the character rotating when they collide with other objects.

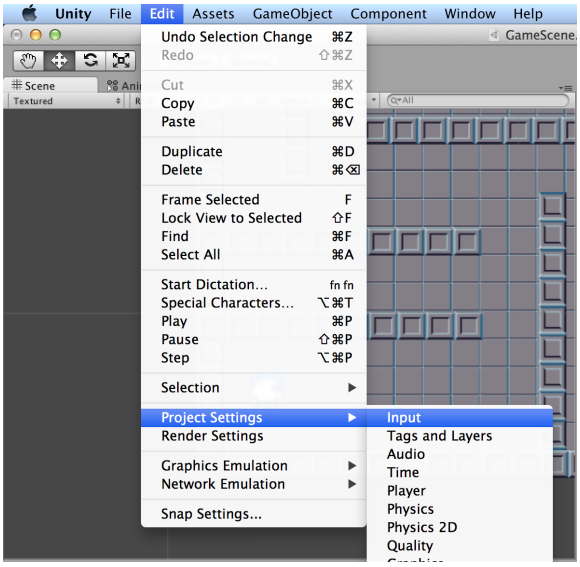
9. While we are here add a 'BoxCollider 2D' component to the Player as this will be used to detect when we collide with those walls in the scene.

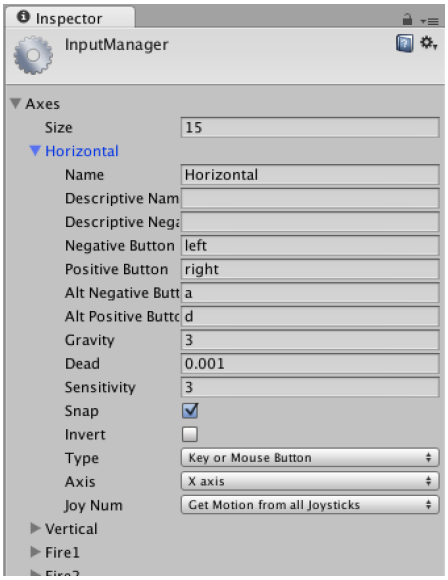
10. Finally you will want to drag the Movement Script onto the player. The script is located in the Assets> Scripts folder. Now click play and you will see that you can move the player around using both the directional keys and the A, W, S, D keys.



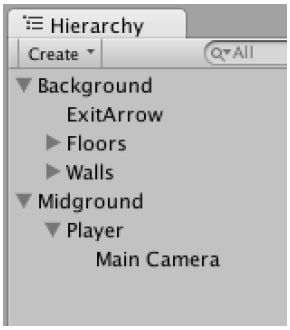
11. If you take a look at the script by double clicking it you will see that there is very little code here. The main parts are getting the values of Horizontal and Vertical from the Input class. These values range from -1 to +1. For Horizontal -1 represents Left and +1 Right and for Vertical -1 represents Down and +1 represents Up. These values are then multiplied by our speed variable and stored in a Vector2 object. Then we use Unitys FixedUpdate function to set the velocity of our player to the movement Vector we just created. The reason we use FixedUpdate here is because we are dealing with Rigid Bodys and Physics and FixedUpdate is always called at the same frame rate compared to Update whose frame rate can fluctuate depending on the complexity of what's going on.

12. You can configure custom Input settings in Unity. Which is how Unity knows to use both the directional keys and A, W, S, D for "Horizontal", and "Vertical" movement. On the Main Menu select 'Edit > Project Settings > Input' - You will see the Input panel appear in place of the Inspector. Here you will see all of the button mappings that you can use. We won't go into too much depth about these settings here but feel free to have a play with them.



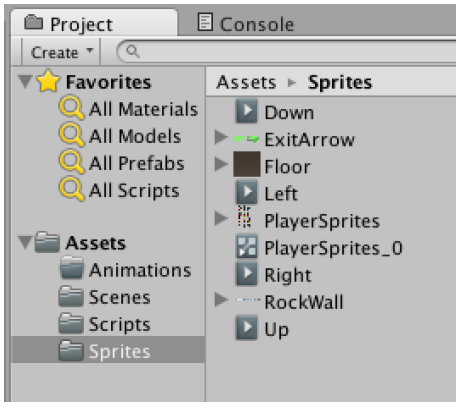


13. Now you will have noticed the camera view is very small compared to the size of the map in our scene view. We are going to make the camera follow the player as they move around the maze. This is incredibly simple. Select the Main Camera and drag it onto the player sprite so that it becomes nested under it. Then hit Play and watch the camera track your character as you move.

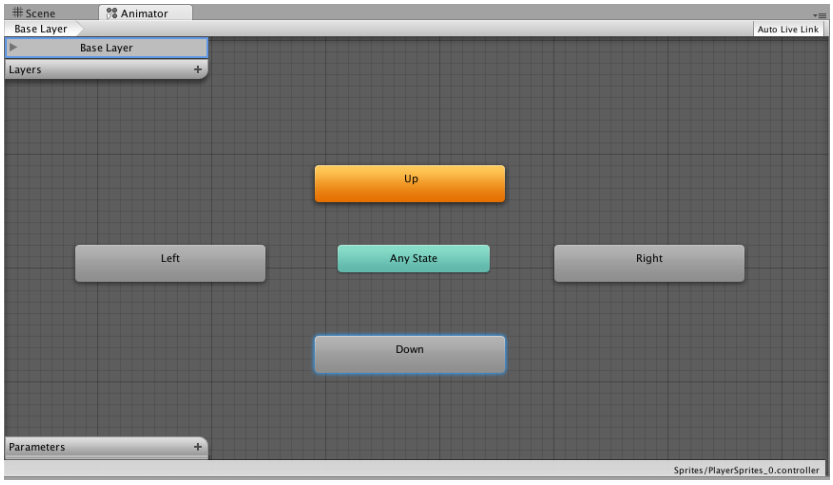


14. Great! Now it's time to get the rest of the animations working! Go back to the 'PlayerCharacter' sprites we created earlier and multi select 3 sprites at a time and drag them onto the Hierarchy and call them as follows (3,4,5 - Name "Up") , (6,7,8 - Name "Left") and (9,10,11 - Name "Right"). Then delete the three you just added from the Hierarchy view as we are not going to use them in this way. (Your hierarchy view should look the same as above when you're done, but the animations will still exist in your assets folder).

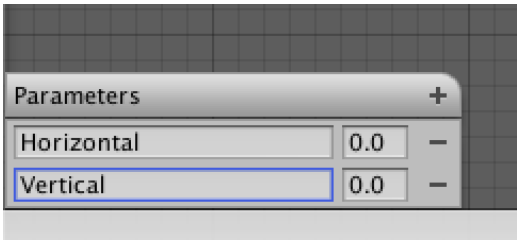
15. You will notice in the assets folder you now have 8 animation files. Down, Up, Left and Right are the animations, represented with a 'Play' icon. PlayerSprites_0 ~ PlayerSprites_9 are Animation Controllers, represented by a little flow chart icon. We don't need 4 animation controllers so delete PlayerSprites_3,PlayerSprites_6 and PlayerSprites_9. Your asset folder should now look like this:



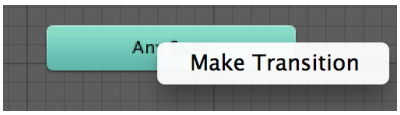
16. Double click the remaining animation controller (PlayerSprites_0) to open up the Animator view. In the Animator view you will see two blocks one should say “Any state” and the other “Down”. Drag the remaining 3 animations into this view and space them out so they are not overlapping.

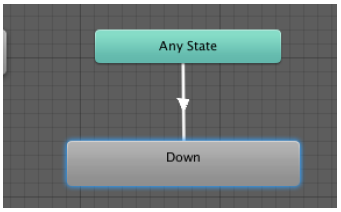


17. The Animator is used to specify conditions that make certain animations play. To do this we’re going to link up the “Horizontal” and “Vertical” values we used to move the player earlier. At the bottom of the Animator window you can see an element called ‘Parameter’ with a + symbol next to it. Click the + and select ‘Float’ from the drop down menu. Name the New Float “Horizontal” and then create a second parameter called “Vertical”.



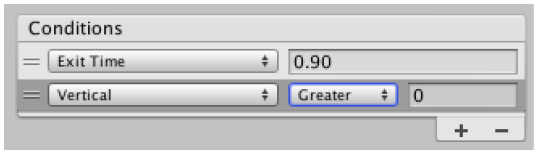
18. Now we're going to create transitions to the animations we do this by Right clicking one of the blocks, selecting 'Make Transition' and then clicking the animation you want to make a transition to. We are going to want to create 4 transitions from "Any State" to each of the four animation blocks.



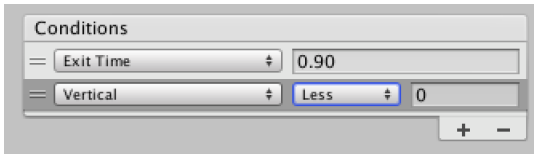


19. We now want to set the individual conditions for when each animation should play. To set conditions click on the white arrow of the transition you wish to edit and in the Inspector you will see a small block called Conditions with one value 'Exit Time' (Don't delete this or your animations will not play) . Click the + button to add another Condition. Set up each of the four transitions as follows:

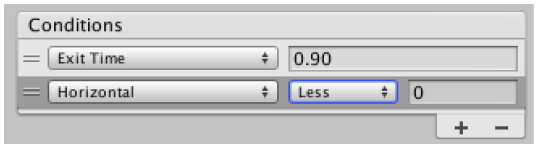
Up condition :



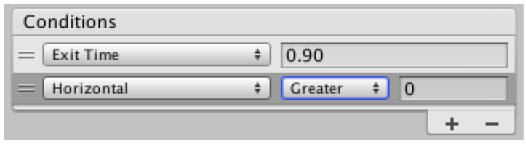
Down condition :

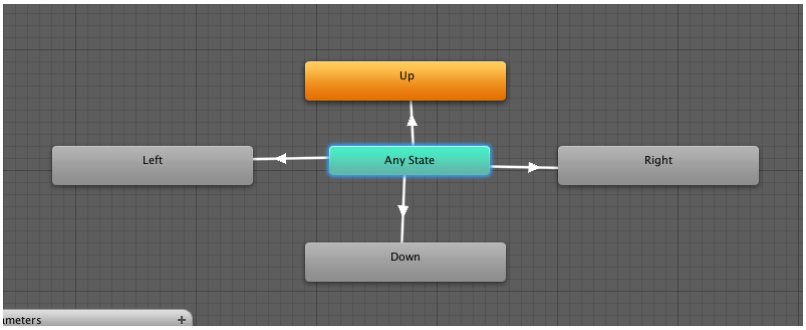


Left Condition :



Right Condition :





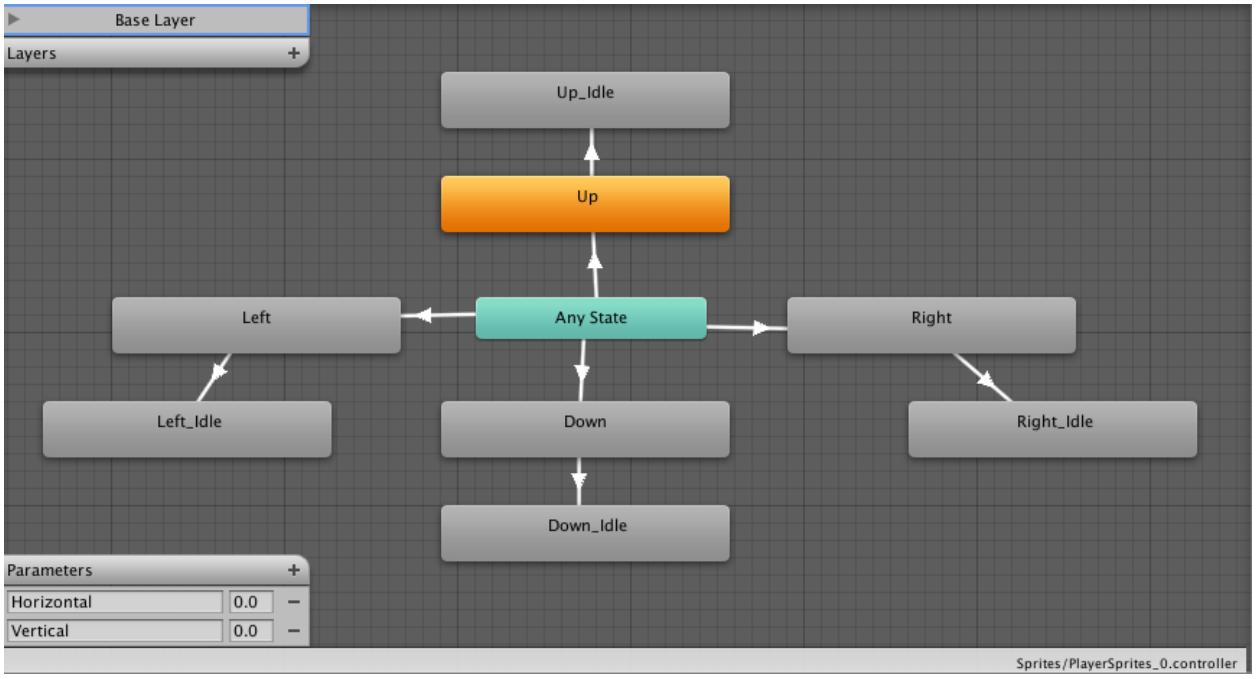
20. Now lets go back to our Movement Script. It's time to uncomment the code in the MovementScript file so that the values for Input are passed to our animator. Now hit play and watch the magic!

```
using UnityEngine;
using System.Collections;

public class MovementScript : MonoBehaviour {

    public float speed = 4;

    private Vector2 movement;
    private Animator animator;

    void Start()
    {
        animator = GetComponent<Animator>();
    }

    void Update()
    {
        float inputX = Input.GetAxis("Horizontal");
        float inputY = Input.GetAxis("Vertical");

        movement = new Vector2(
            speed * inputX,
            speed * inputY);

    }

    void FixedUpdate()
    {
        rigidbody2D.velocity = movement;
        animator.SetFloat("Horizontal",movement.x);
        animator.SetFloat("Vertical",movement.y);
    }
}

```

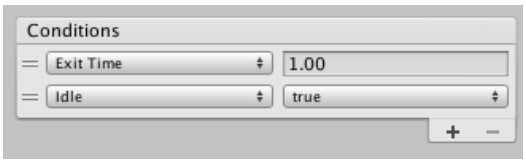
21. The last thing we're going to do is get the Idle animations working so that when the player stops moving they stop animating and take the Idle pose of the direction they were facing.

22. We are going to create the Idle Animations from scratch within unity. In the Asset window

do Right Click > Create > Animation and create 4 new animations each named ‘Up_Idle‘,’Down_Idle’,’Left_Idle’ and ‘Right_Idle’. Drag these into the Animation Controller and connect them to their corresponding animations so for example make a transition between Down and Down_Idle. Your animation controller should look similar to below:



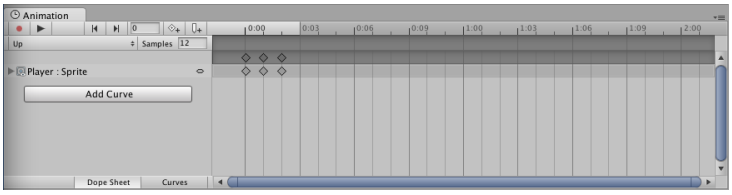
For the new transitions we are going to create a new Parameter called “Idle” and set each of the four new transition conditions to be Idle = true as shown here:



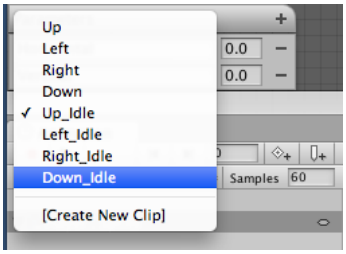
23. We now need to link this new parameter up to our script! So go back to the MovementScript and add the following code to the update function to set the Idle value to true when no input is present:

```
void Update()
{
    float inputX = Input.GetAxis("Horizontal");
    float inputY = Input.GetAxis("Vertical");

    if(inputY == 0 && inputX == 0)
    {
        animator.SetBool("Idle",true);
    }

    movement = new Vector2(
        speed * inputX,
        speed * inputY);

}
```

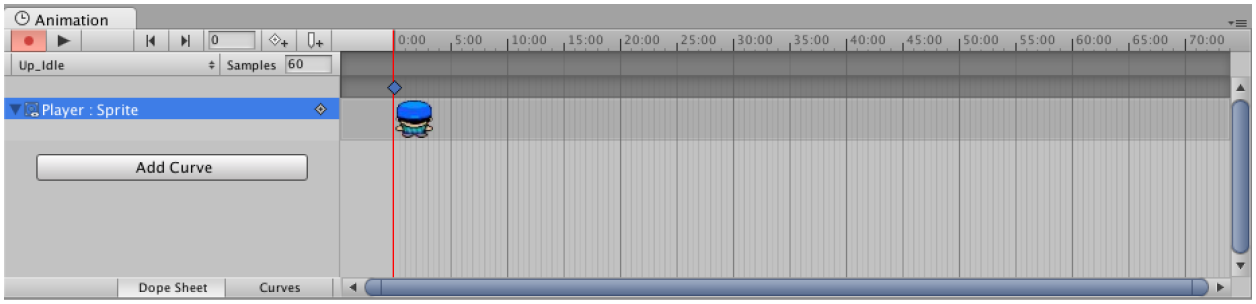
23. The new animations are currently empty right now so we need to edit them. To edit animations first make sure that the Animation window is visible. (Select Window > Animation from the main menu). Then select the Player within the Hierarchy and you should see some values appear in the Animation window.



24. Just under the Record and Play buttons in the top left you will see the name of one of your animations, click this to see a dropdown list of all the animations currently attached to our Animation Controller.



25. Choose one of the Idle animations to view it's timeline. Now to add animation frames to this timeline you can simply select the relevant Sprite from the Assets folder and just drag it onto the timeline. Do this for the four Idles. It should look similar to this:



26. Now click play and watch how your animations change from Walking to Idles as you move. The last thing you will want to do is to right click one of the Idle animations and choose ‘Set as Default’ to ensure that when the game starts the player isn’t animating.

27. This concludes the tutorial, I hope you found it a-maze-ing ;)