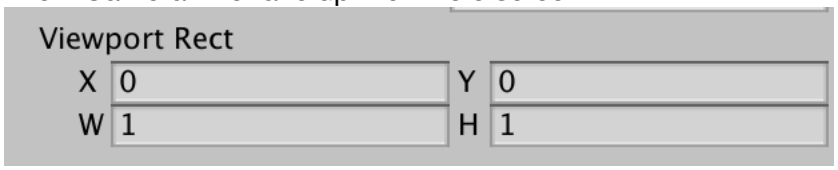
Quest 9 - Steps

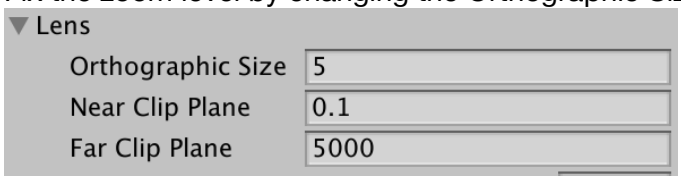
1) Setup

Quest 9 builds off of Quest 8 and uses the same project and scene.

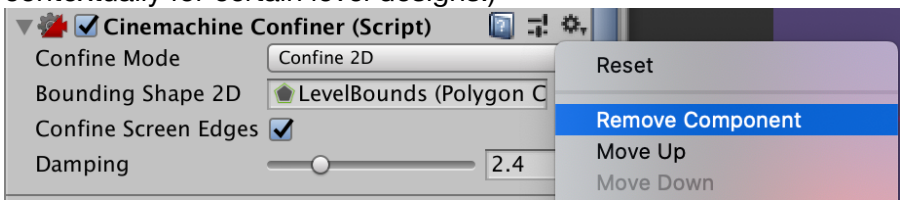
This project starts with a few changes from where Quest 8 left off:
- Delete Player1
- Delete Camera1
- Allow Camera 2 to take up the whole screen



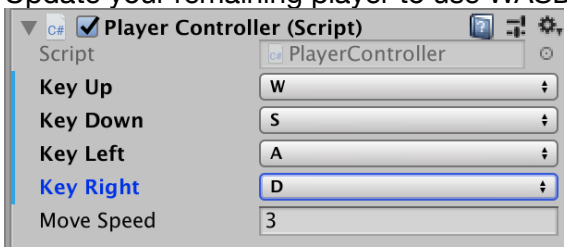
- Fix the zoom level by changing the Orthographic Size on the Virtual Camera



- Remove the Virtual Camera Confiner (This is an optional component that you can use contextually for certain level designs.)



- Update your remaining player to use WASD movement keys

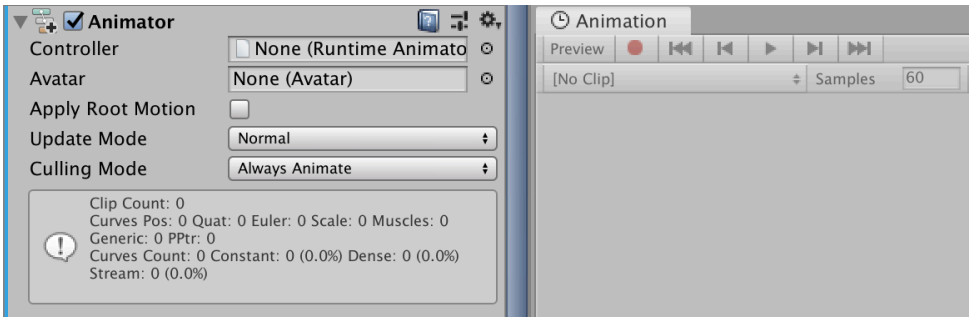


- Reset the Player Sprite Color to full White

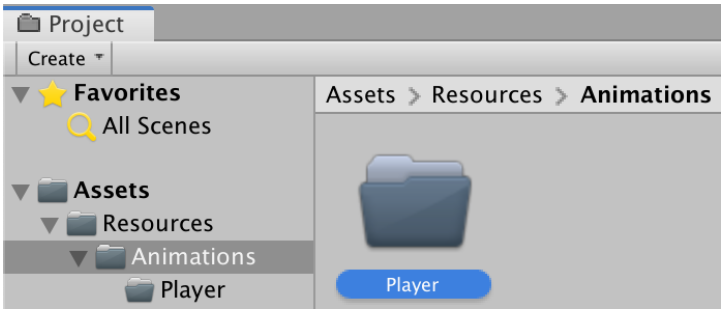
Playtest with this configuration ensure you have a single player that is able to move with the camera following.

2) Animator and Animations

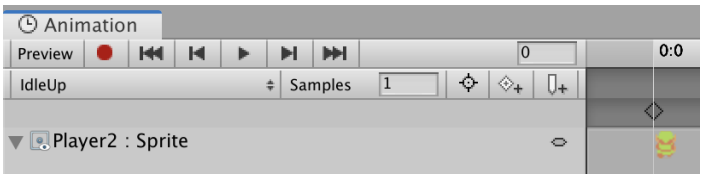
Attach an Animator Component to the Player and open the Animation Window



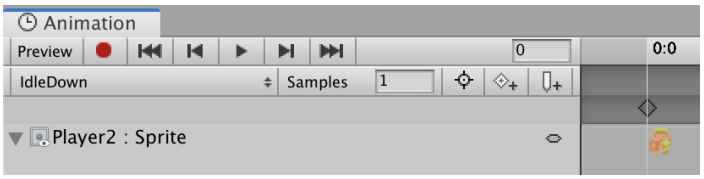
Store all of your Player animations in /Assets/Resources/Animations/Player/



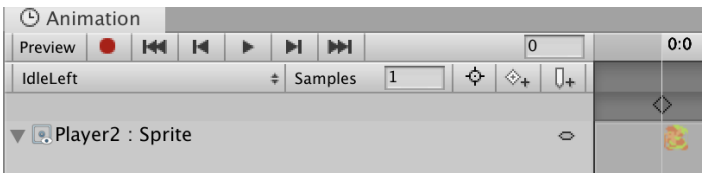
Create an IdleUp animation using frame 8 at a Sample rate of 1 frames per second.



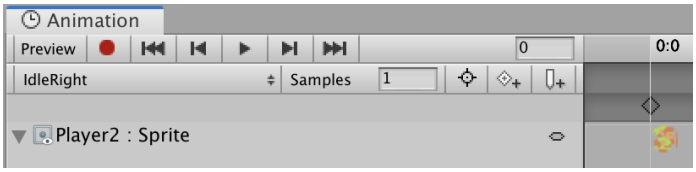
Create an IdleDown animation using frame 4 at a Sample rate of 1 frames per second.



Create an IdleLeft animation using frame 10 at a Sample rate of 1 frames per second.



Create an IdleRight animation using frame 6 at a Sample rate of 1 frames per second.

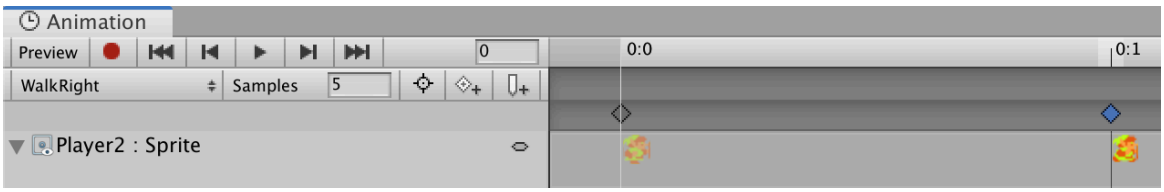


Create a WalkUp animation using frames 8 and 9 at a Sample rate of 5 frames per second.

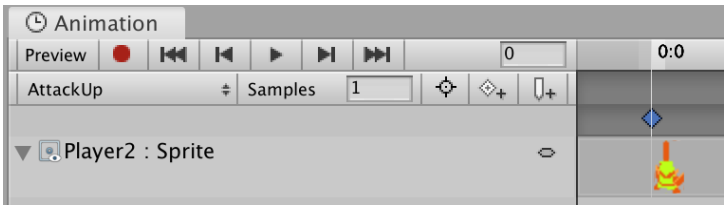
Create a WalkDown animation using frames 4 and 5 at a Sample rate of 5 frames per second.

Create a WalkLeft animation using frames 10 and 11 at a Sample rate of 5 frames per second.

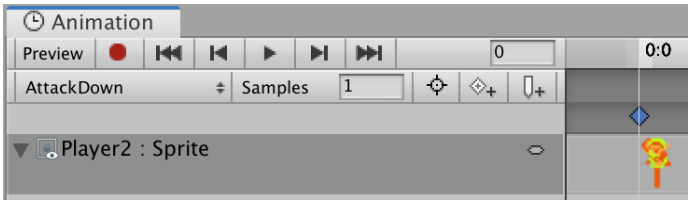
Create a WalkRight animation using frames 6 and 7 at a Sample rate of 5 frames per second.



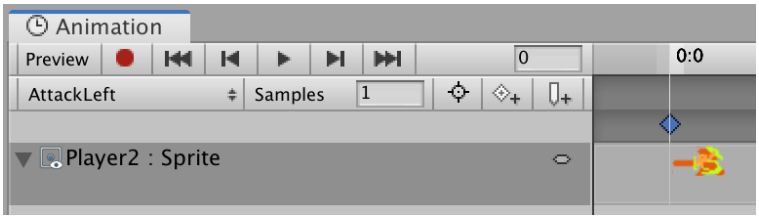
Create an AttackUp animation using frame 30 at a Sample rate of 1 frames per second.



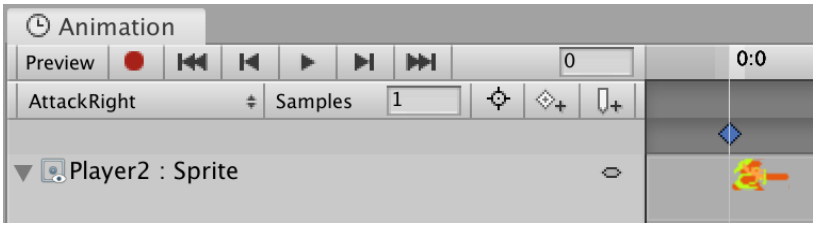
Create an AttackDown animation using frame 26 at a Sample rate of 1 frames per second.



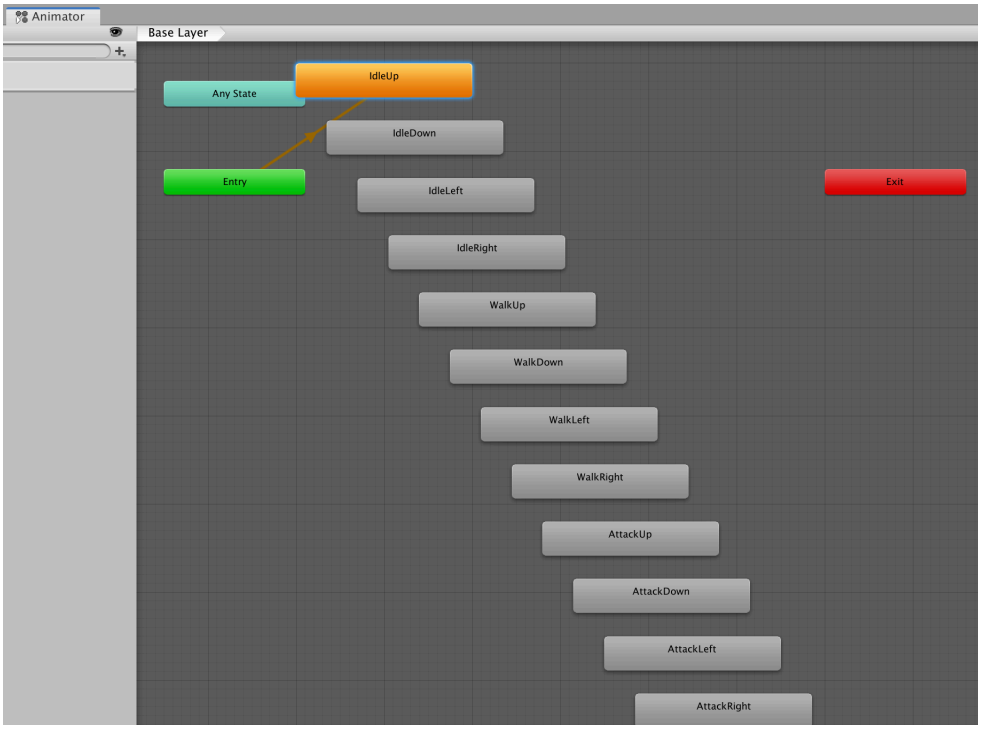
Create an AttackLeft animation using frame 35 at a Sample rate of 1 frames per second.



Create an AttackRight animation using frame 28 at a Sample rate of 1 frames per second.

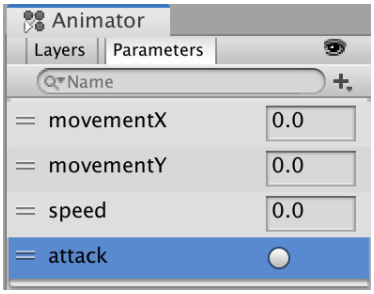


With all these animations, the Animator screen is full of a lot of possibilities and would actually be very unwieldy if you only used Transition techniques from previous projects.



3) Animator Parameters

Set up the following four Animator Parameters:
- movementX (float)
- movementY (float)
- speed (float)
- attack (trigger)



To reference the Animator Component from the PlayerController, add an outlet and fill the reference during the Start event.

```
// Outlets
Rigidbody2D _rigidbody;
Animator _animator;
```

```
// Methods
void Start() {
    _rigidbody = GetComponent<Rigidbody2D>();
    _animator = GetComponent<Animator>();
}
```

Physics interactions are in the FixedUpdate loop because it is synchronized with the physics engine, but we will handle input interactions during the Update loop because we will be affecting animations.

```
void Update() {
    float movementSpeed = _rigidbody.velocity.magnitude;
    _animator.SetFloat("speed", movementSpeed);
    if(movementSpeed > 0.1f) {
        _animator.SetFloat("movementX", _rigidbody.velocity.x);
        _animator.SetFloat("movementY", _rigidbody.velocity.y);
    }

    if(Input.GetKeyDown(KeyCode.Space)) {
        _animator.SetTrigger("attack");
    }
}
```

4) Blending Animation States

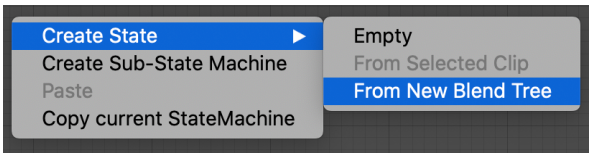
This project would benefit from a new Animation technique known as Blend States because our many animations fit into clear categories of “Idle,” “Walk,” and “Attack” and also fit into clear divisions of “Up,” “Down,” “Left,” and “Right.”

In chart form, our many animations are just the overlap of these categories and directions:

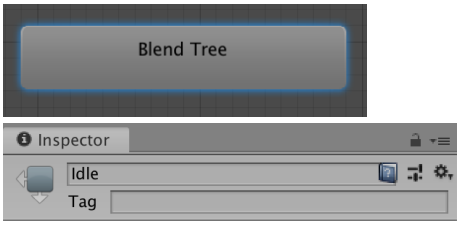
Direction/Category	Idle	Walk	Attack
Up	IdleUp	WalkUp	AttackUp
Down	IdleDown	WalkDown	AttackDown
Left	IdleLeft	WalkLeft	AttackLeft
Right	IdleRight	WalkRight	AttackRight

Our Categories will become 3 separate “Blend States” that blend our 4 separate Animation “Motions.”

First, we will actually delete all 12 animations from the Animator window. Then right-click and add a New Blend Tree.

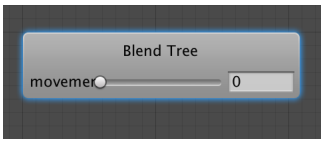


Click the new Blend Tree and in the Inspector, rename it “Idle.”



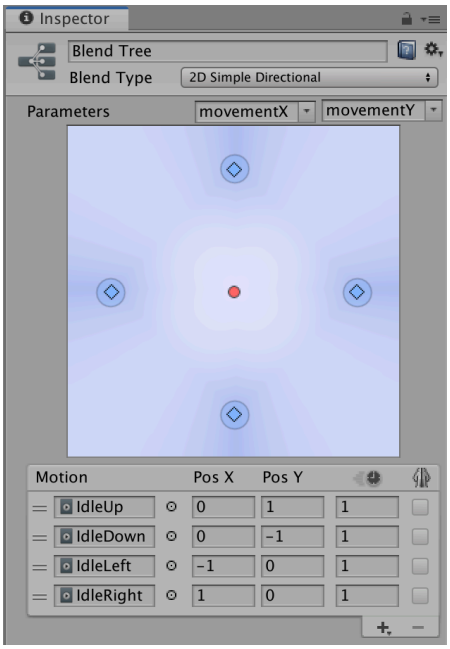
Double-click the Blend Tree to go into the details. Notice how the Animator portrays navigation and hierarchy.

Click the Blend Tree and view the details in the Inspector.

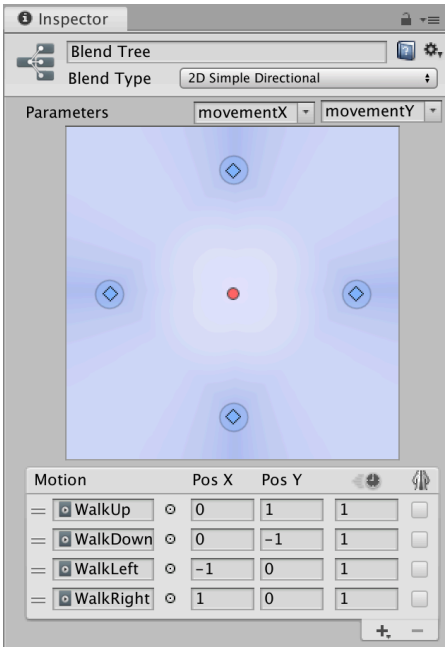


You can use parameters to specify the amount that you blend Animations/Motions together. If you use two parameters, you can create a two-dimensional blend. We will use the two parameters of “movementX” and “movementY” in combination to figure out direction (Up, Down, Left, Right).

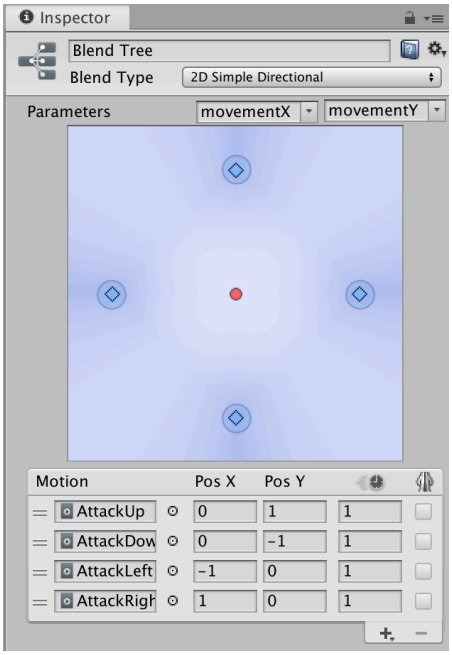
With these settings, any combination of X and Y movement values will activate the appropriate Up, Down, Left, or Right animation.



Return to the Base Layer and copy and paste the “Idle” state twice. Rename one to “Walk” and rename the other to “Attack.” Update the “Walk” blend tree to use walk animations.

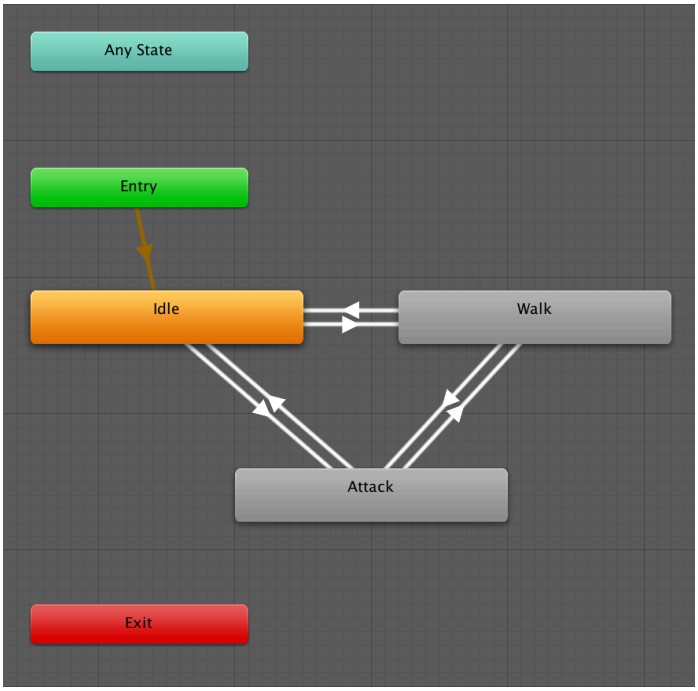


Update the “Attack” blend tree to use attack animations.

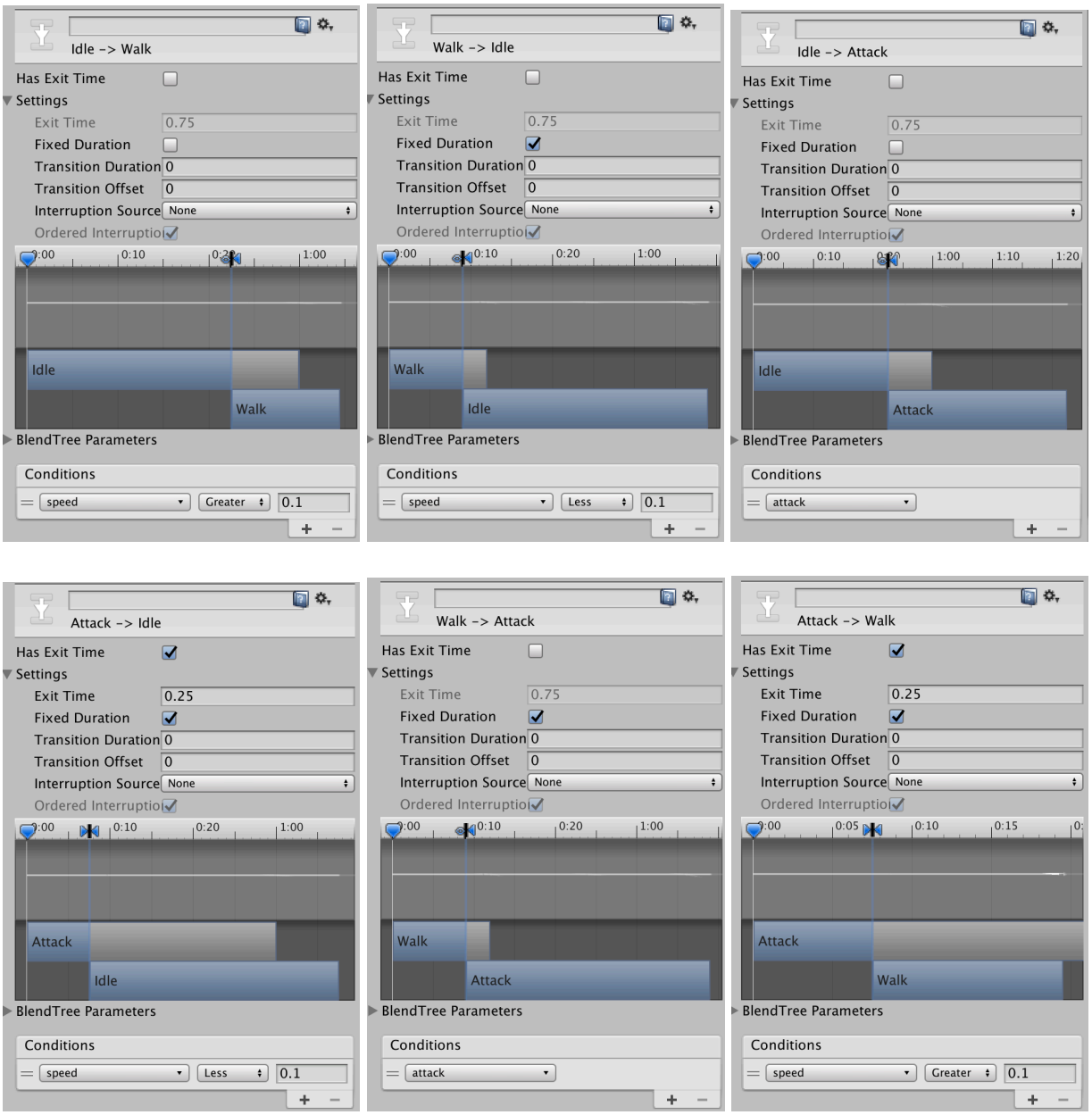


5) Transitioning Between Blend Trees

With all the direction animations categorized into different Blend Trees, transitioning between them works the same as prior animator techniques. Set “Idle” as the Default Layer State, and create transitions in both directions between the three states. There should be six transitions in total.



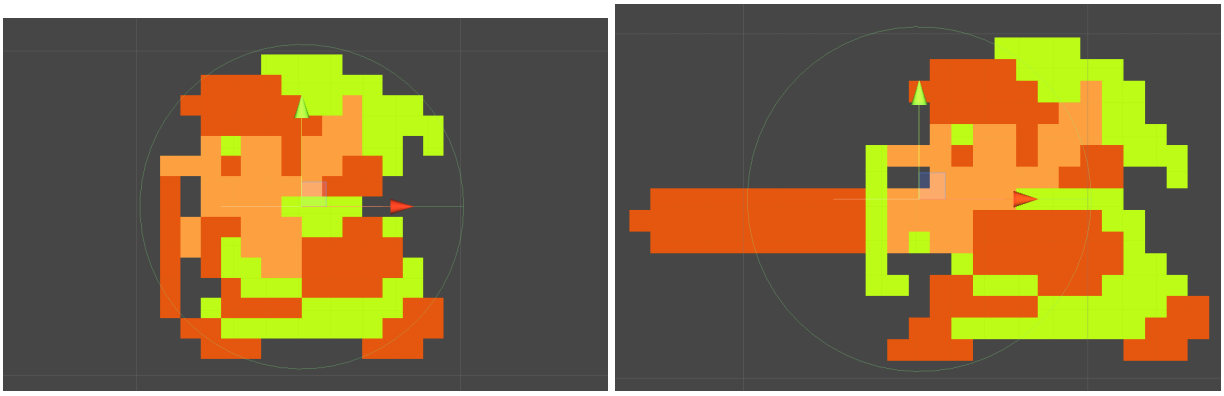
Implement the following settings for each Animator Transition. Be precise and make sure every field matches. Not all transitions are the same.



If you playtest the game now:

- You should be able to walk in all four directions and the character will appropriately face that direction.
- The player will switch between idle and walk animations depending on whether the character is actually moving.
- Pushing space bar will cause the player to stick out the sword, and after a 0.25 second delay, retract the sword.

There is one major bug at the moment: When you attack, the player sprite moves out of alignment because all the sprites are set to use their center as the pivot.

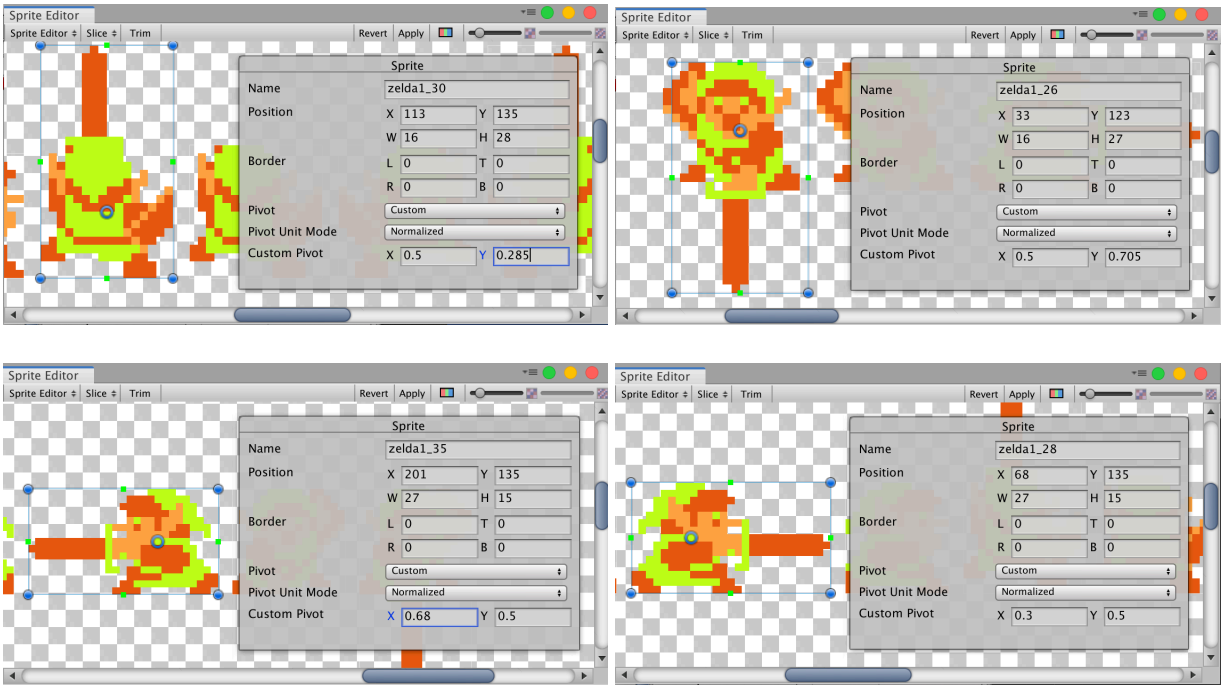


Notice how our character is pushed outside of his circle collider because of the mis-aligned sprite.

6) Fix animation/sprite pivot points

Sprite pivot points are changed back in the Sprite Editor. Select the zelda image file in /Assets/Resources/Textures/ and click the “Sprite Editor” button from the Inspector.

For each sprite that you used for AttackUp, AttackDown, AttackLeft, and AttackRight, set the Pivot to Custom and use the value shown in this image. These values realign the attack sprite to keep the pivot point in the same place as the matching Idle sprite.



Be sure to apply any changes when asked.

7) To be continued…

Playtest to ensure the player no longer jerks backward when doing attacks.