Part 2: 3D Melee Combat Controller

Debrief – 08/24/2021

# Overview

Most of the work for this part was visual scripting in Mechanim to get the attack animations working properly. I decided to go with a collider-based attacking system, so complete and accurate animation transitions based on player input was paramount to creating an effective controller. In addition, I used a lot of UnityEvents to decouple animations from scripted behavior, but still allow for some degree of communication between the two systems. The result is a clean combat controller capable of (very basic) one-handed melee combat.

# Goals Recap

* Create base for 3D melee combat by adding attack logic to the game
  + There are several new classes I created to implement attack logic. They are as follows:
    - FighterComponent – The primary class for combat. It will be common to all humanoids in the game. It is responsible for all interactions the humanoid has with its weapon – drawing/sheathing it, attacking with it, and equipping (currently referred to as initializing) it.
    - CombatTarget – A class that reveals a UnityEvent in the inspector. Responsible for invoking that UnityEvent whenever the target is hit.
    - AttackStrategy – A ScriptableObject class with an Execute() method that dictates how the weapon will behave when it is used to attack. The current, basic melee strategy is a 4-attack combo that is executed sequentially if the player triggers a new attack within a time window.
    - WeaponObject – A Monobehaviour with a collider. This class is strictly responsible for detecting collisions and making the struck transforms visible to a Weapon class.
    - Weapon – A ScriptableObject responsible for triggering weapon events (like attacks) in compliance with it’s assigned AttackStrategy. It gathers collision info from its WeaponObject and calls the OnAttackRecieved event of the colliding transform’s CombatTarget component.
    - WeaponSlot – A class that holds a reference to two weapon positions (transforms) on the Humanoid’s HumanoidGraphics object. It is responsible for transferring the weapon from one slot (ex. the left hip) to the other (ex. the right hand), and back again.
* Add new animations to the 3D model for combat
  + Animations are tied to the combat logic mostly through UnityEvents. For example, the FighterComponent class has two UnityEvents, one for drawing a weapon and the other for sheathing. When the FighterComponent toggles the weapon’s drawn state, it invokes these events. The response to these events can be configured in the inspector. In this case, it calls the appropriate method in the HumanoidGraphics class to trigger a weapon draw or sheath animation.
  + I anticipate the animations for Humanoids in the game to get relatively complex. I found a helpful resource on the Unity3D website [here](https://unity3d.com/how-to/build-animator-controllers) that helped me to form an AnimatorController with several layers. I slotted in some granular layers, such as individual arm overrides, but the most important layers were Base Locomotion, Upper Body Override, and Full Body Override.
    - Base Locomotion – This is a layer I created in the previous part of this project. The running, crouching, and jumping motions are housed here. The only change for this part was adding in the Combat Locomotion blend tree. Now, when a weapon is drawn, the Humanoid has slightly different animations to account for the weapon and its weight.
    - UpperBodyOverride – The purpose of this layer is to ensure that attacks can occur while the humanoid is moving. In this layer, the leg movement is delegated to the Base Locomotion layer. This layer’s attack animations trigger when an attack is performed regardless of the state of the character’s motion. The humanoid can run, jump and crouch all while attacking thanks to the upper body being controlled separately in this layer.
    - FullBodyOverride – The purpose of this layer is to ensure attacks use the full body when the humanoid is standing still. In the animation pack I am using, by default the attacks have some footwork involved. I only wanted that footwork to be displayed if the humanoid wasn’t moving, to prevent the appearance of sliding around. So, this layer on triggers its attacks if the humanoid is stationary, and bails out back to it’s empty state if the humanoid begins moving during an attack.
  + I implemented the collider-based attacking system described in the brief. I hooked up animation events to UnityEvents in the HumanoidAnimationEventListener class, which allowed me to configure responses to the animation events in the inspector. The two animation events are AE\_WeaponColliderActivate and AE\_WeaponColliderDeactive. These events are triggered toward the beginning of the swing and towards the end of it, respectively , for each of the four provided one-handed attack animations. I configured responses to these events to have the FighterComponent reactive by activating its weapon’s collider and deactivating it at the beginning and end of the swings. The result is a realistic window in which the humanoid’s weapon may damage its target. I really loved this method of delegating animation event responses out to different humanoid components because it felt clean, designer centric, and easy to debug.
* Create 3, one-handed melee weapons for testing
  + I created a morning star (spikey mace), a war axe, and a longsword for the three one-handed melee weapons. Creating the axe was particularly helpful in debugging, as due to its asymmetry it highlighted rotation issues with the WeaponSlot object’s referenced graphic transforms. The pipeline to build new weapons was very easy thanks to the ScriptableObject based architecture. The steps are as follows:
    - Create a GameObject from middleware art
    - Give it a collider, position the collider to roughly where the painful bits of the weapon would be
    - Add a WeaponObject Monobehaviour script
    - Create a prefab of this GameObject
    - Create a ScriptableObject Weapon, name it something intuitive (like iron war axe)
    - Assign the GameObject prefab to the Scriptable Object
    - Choose an attack strategy for the Weapon
* Create a stationary target to test melee combat on, with some reaction to being hit (shaking)
  + I ended up just using a second Humanoid for a target because there were no free melee target art on the Unity Asset Store. I suppose I could have looked around on other websites, but a humanoid would get the point across.
  + I created a class called CombatTarget (described above) which delegates a response to being hit to a UnityEvent that can be configured in the inspector. So, any object in the game that has a CombatTarget component can react to getting hit, and how it reacts is very easy for a designer to set up with little to no coding required. This simple architecture was somewhat of an “ah-ha” moment for me, and alleviated a lot of the redundancy I encountered with a more C# generalized ICombatTarget interface.
  + The WeaponObjects pass any detected collider during an attack to their Weapon scriptable object, which then finds the struck object’s CombatTarget component and invokes its response to being hit. Later on, this will also apply damage or other on-hit effects. For right now, it just plays a stagger animation on the target’s humanoid graphics.

# Future Enhancements

* I think how fast the weapons swing will have to be based on their weight and perhaps player strength. For right now, its all uniform across each of the three weapons. This will make animations a little more complicated, as the parameters will have to change based on which weapon is currently active for the player.
* I put a lot of time into making these animations look smooth and crisp. It’s a shame this game is only a first-person view currently. I think I would like to add a third-person camera toggle so that those who want to see the animations in their full glory (like I did during testing) can do so.
* There are only 4 possible 1-handed attack animations currently. I think the animation controller I created, as well as the AttackStrategy architecture, is scalable enough to allow different animations for different weapon types. I have two more animation packs from the Unity Asset Store, so later I would like to dig deeper into the AttackStrategy class and deepen combat.

# Next Steps

We have all these beautiful weapons, but the player can only use one at a time! I think the next step will be to implement the beginning of an inventory system so the player can try out all three of the weapons on the poor red dude.