- Online Documentation (most up to date)
- Discord Invite

Quick Starts

Newcomers should start in the Newcomers Start Here section. The quick starts are for people who have already gone through the newcomer tutorial.

- Create a New Player Action Character
- Create Hitboxes
- Combine Actions with HitBoxes

Addons

Addons are additional optional packages to suit your needs. These come at an extra nominal price ($~15). However, the base source is exposed, so you can write your own addons as well if you so wish (see System Extensions).

- New Unity Input & Mobile
- Reaction Character
- Effects Character
- Action Camera
- Ultimate Action Character Bundle
    - Includes all addons listed above and base package at a discounted price
    - Existing users of Action Character can upgrade at a discounted price

Integrations

These are additional optional integrations into other 3rd party assets. These are freely available on the store.

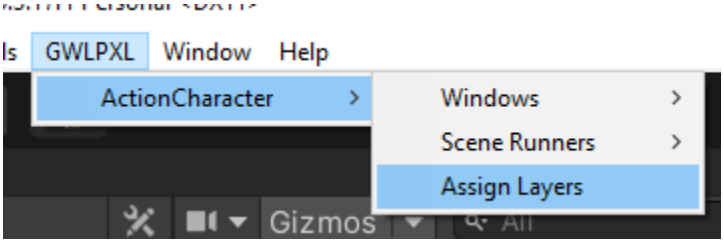
- Invector Locomotion
- Rewired
- Character Controller Pro
- JU TPS 2 : Third Person Shooter System + Vehicle Physics (Coming soon)

Newcomers Start Here

Some quick guides to get you

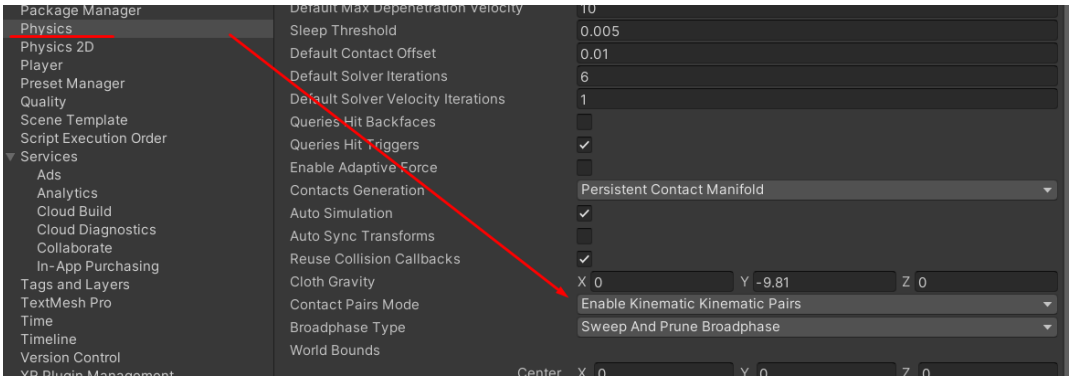
- Playing the demo
- Creating your first action!

Playing the Demo

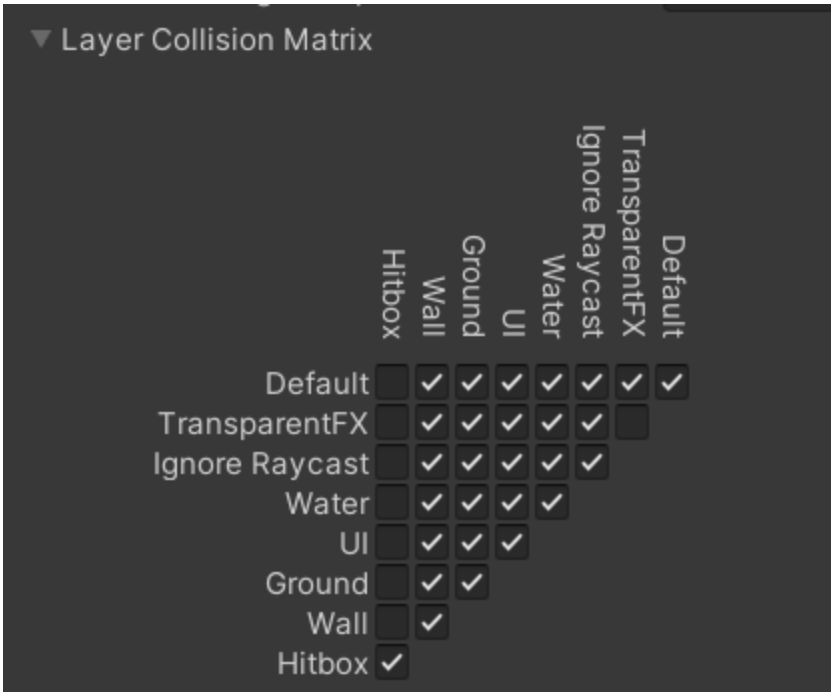
1. Load the demo scene Demo_01.
2. Assign the required layers (GWLPXL -> Action Character -> Assign Layers)
    a. 

    b. To customize the layers, you can edit the DemoDefaults.cs.
        i. Defaults include Hitbox layer (“Hitbox”, layer 19), Wall layer (“Wall”, layer 7”), Ground layer (“Ground”, layer 6).

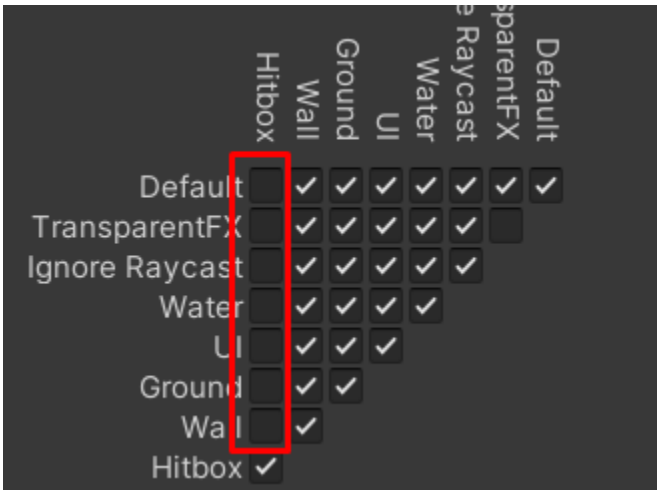
The demo also uses Kinematic + Kinematic contacts, so we need to enable that.

1. Edit -> Project Settings -> Physics.
2. Contact Pairs Mode -> Enable Kinematic Kinematic Pairs.
    a. 

3. While we are here, we also only want **only** Hitboxes and Hitboxes to interact with each other.
4. Go to the Layer Collision Matrix (also in the Physics settings here).

a. 

5. Enable so only Hitbox / Hitbox is toggled on.

a. 

6. Enter Play.

Demo Controls

- **Left Click** does Combo **A** attacks.
- **Right click** does Combo **B** attacks.
- **Spacebar** Jumps.
- **Middle Click** dash/rolls

Included Combos for Set 03 (the default set on the player in the demo scene).

Combo Set **A**

- A -> A -> A -> A
- A - > B
- A -> A -> B
- A -> A-> A-> B
- A-> A-> A-> A-> B

Combo Set **B**

- B -> B -> B -> B
- B - > A
- B -> B -> A
- B -> B-> B-> A
- B-> B-> B-> B-> A

Dash/roll will early exit (cancel) out of the combos.

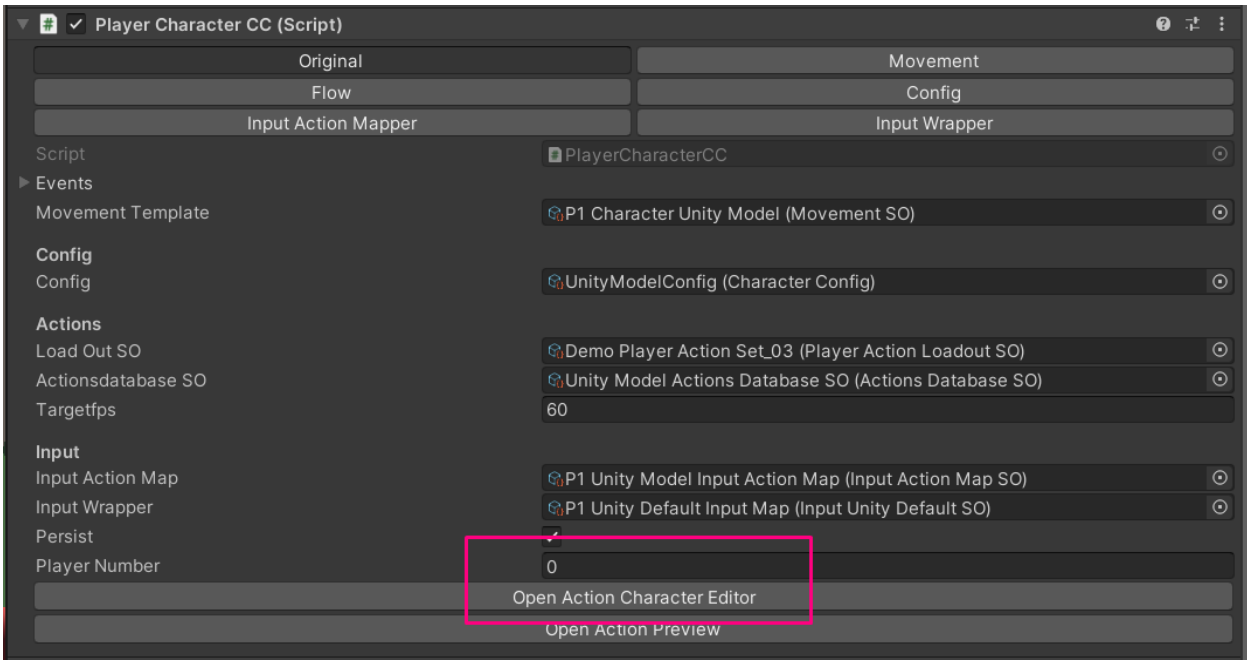
Demo Objects

- **‘Faux’** signals that the scripts are for demo purposes only, not meant for production use.
- **‘Faux Placeholders’** on the Placeholders object let’s you configure how many enemies to enable and the buttons to enable/disable them at runtime.
- **PlaceHolderEnemy** is an example of an “other” character that the player can interact with. It includes Faux scripts for movement and knockback.
- **PlayerActionController** is a player configured prefab.

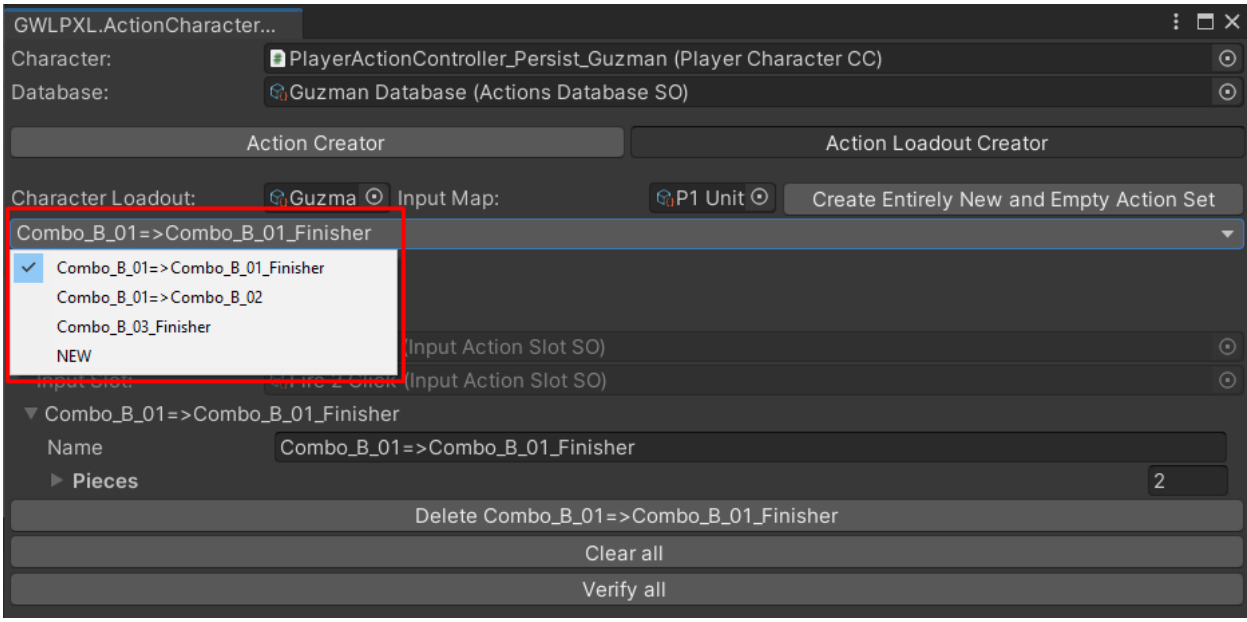
Creating Your First Combo!

On the player, change the loadout to Demo set 01. A loadout contains the actions currently available to the player and their combo properties.

If you enter Playmode, we now only have combo chains for our left click attack (repeatedly tap left click) but our right click attack doesn't (it just repeats the same move over and over). That's because this loadout only has information for Combo_A. Let's modify it so that our right click combos into our left click.

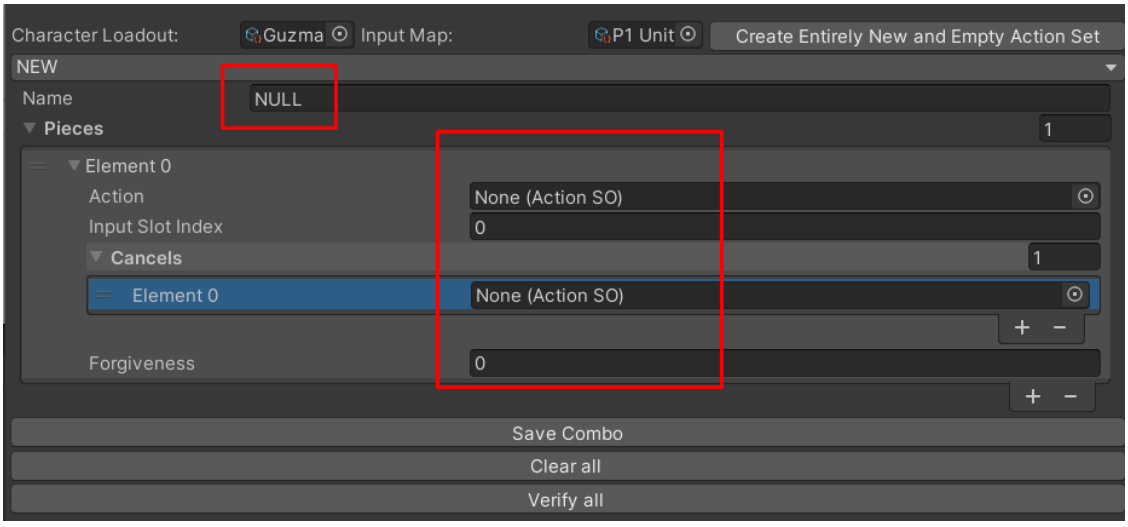
1. Click on the Action Character Open the Action Editor (remember to have Demo Set 01 loaded).
    a. 

2. Click on Action Loadout.
3. Make sure Demo Player Action Set_01 is in there (should be if it was set on the action character previously).

a. 

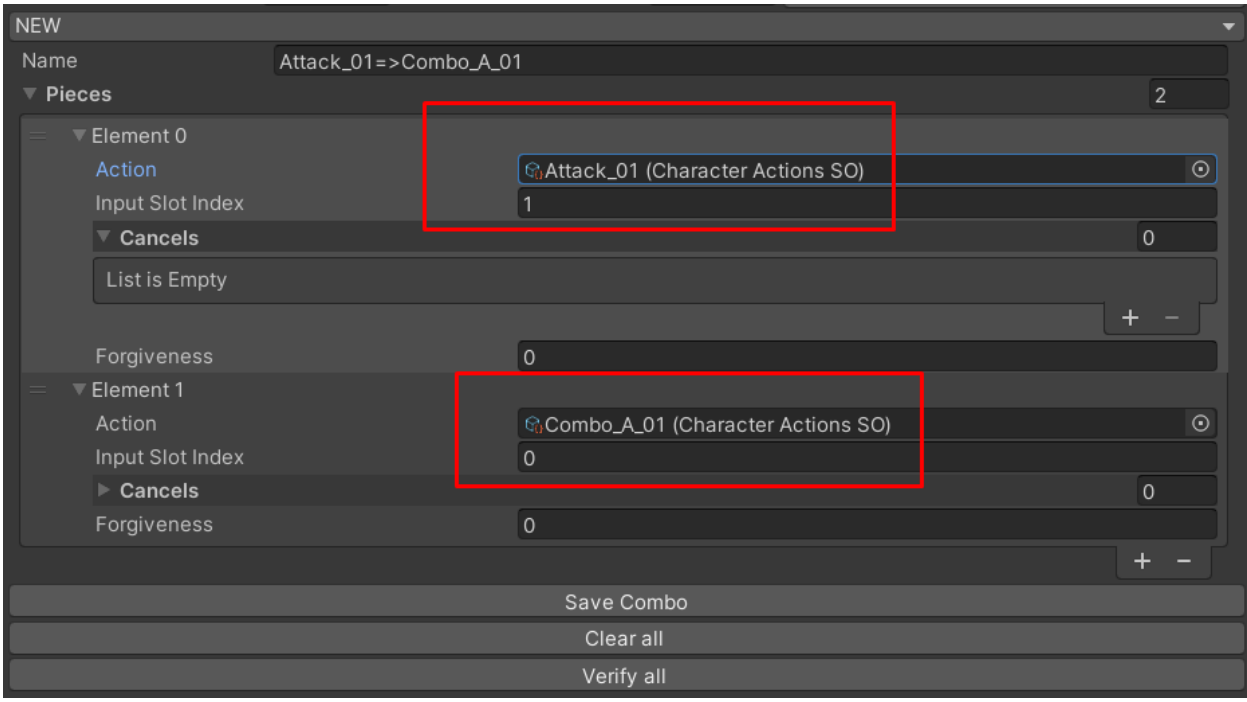
   i. The dropdown shows any currently assigned Combos to the loadout.

4. Now choose “NEW”.

a. 

   b. Now you can create the combo by filling in the pieces.

5. If we want Attack_01 -> Combo_A_01 then fill in the pieces.

a. 

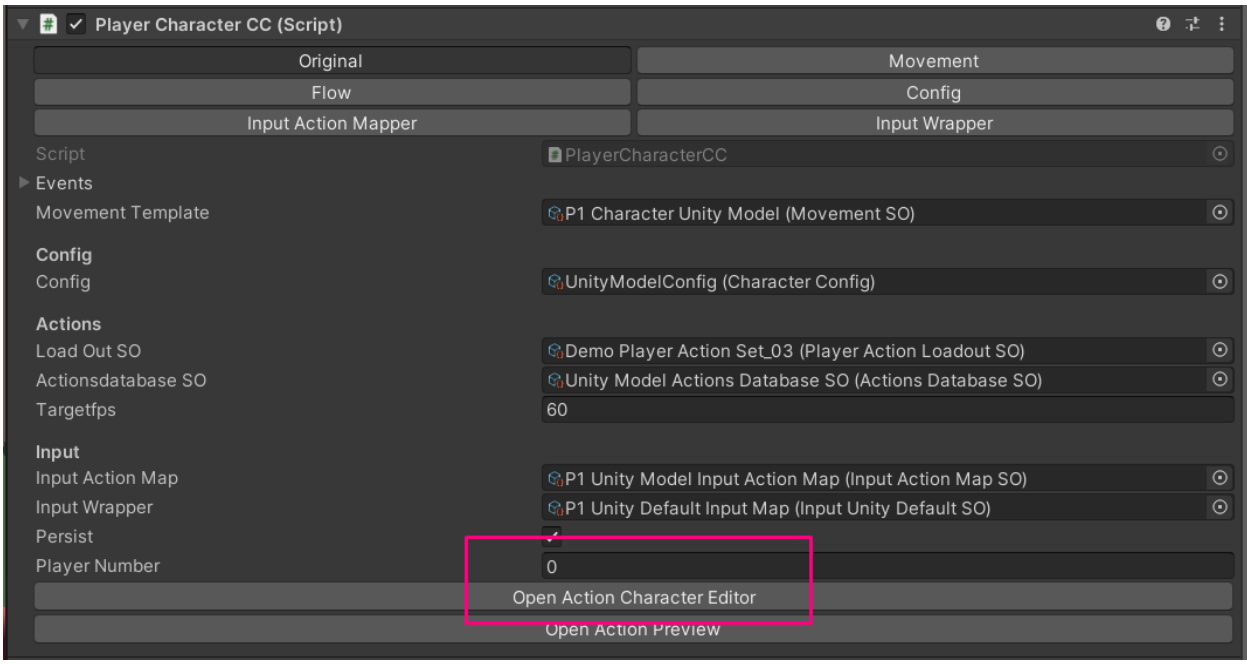
6. Click Save Combo.
7. That's it! You'll see it get added to our Combo List for this loadout.

Now if we right click **and then** left click, we can combo into Attack_01 -> Combo_A_01. Success!

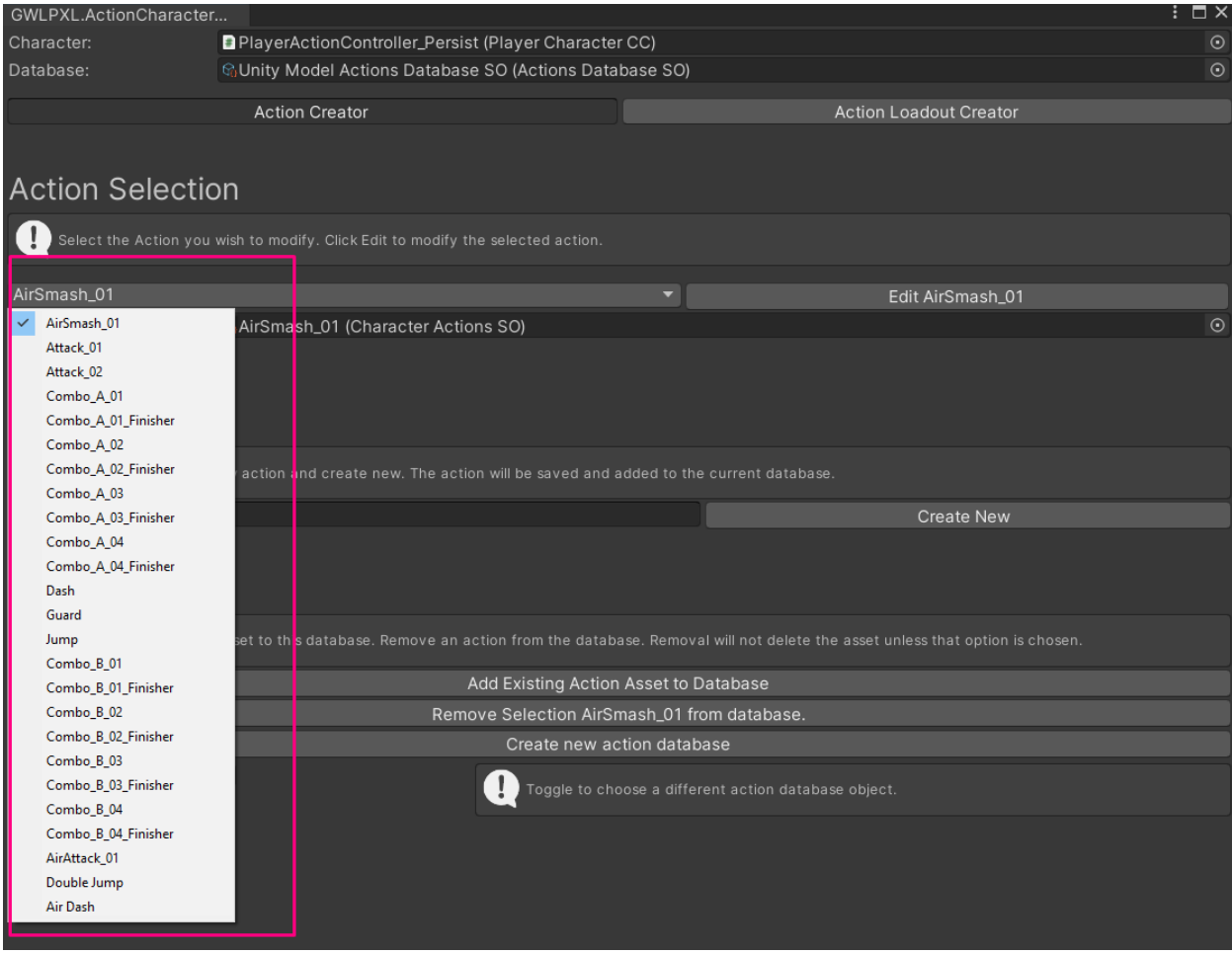
This demonstrates the basics for how the combo system works. Let's check out more of the **Action Character Editor** in the next step!

Modifying Your First Action!

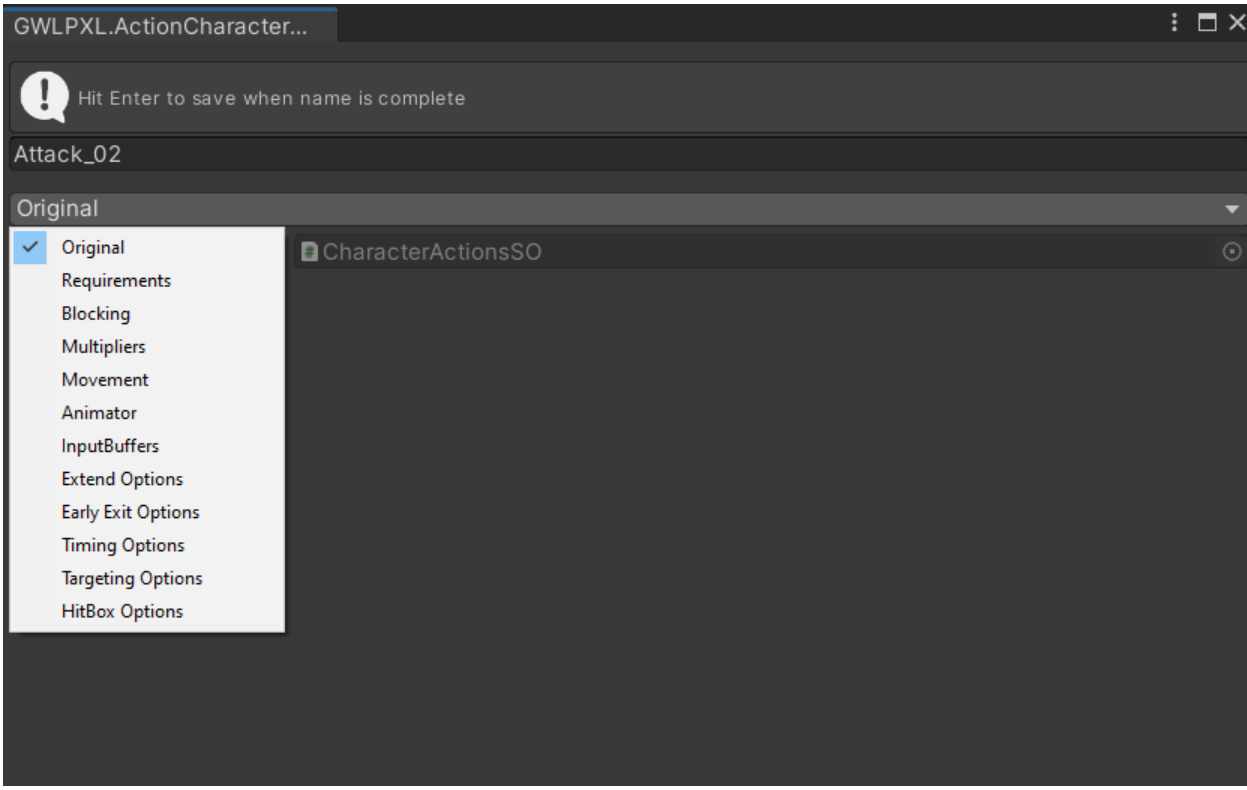
Action Assets are Scriptable Objects and the provided examples can be found in the Actions folder. For now, let's introduce ourselves to the Action Character Window to edit actions and eventually to use for all of our changes.

1. Open the Action Character Window.
2. Click the Open Action Character Editor button on your Action Character.
    a. 

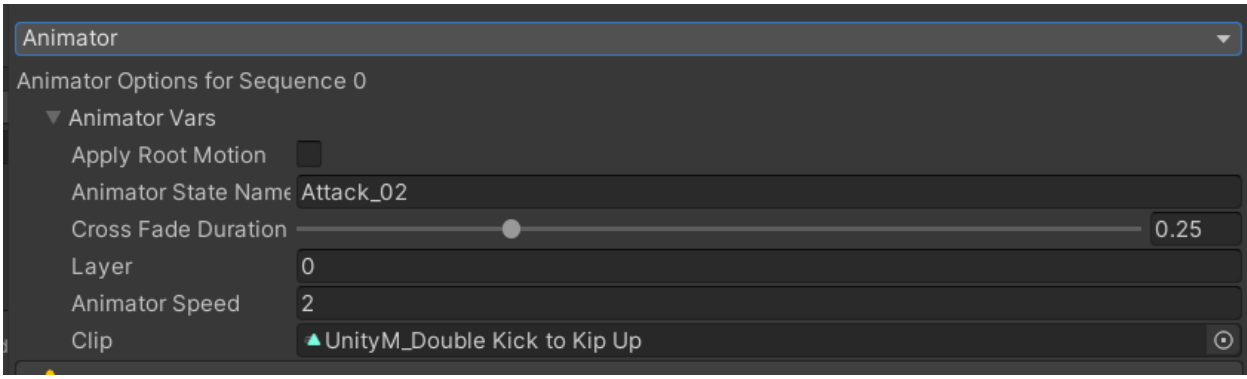
3. We now see a list of our actions.

a. 

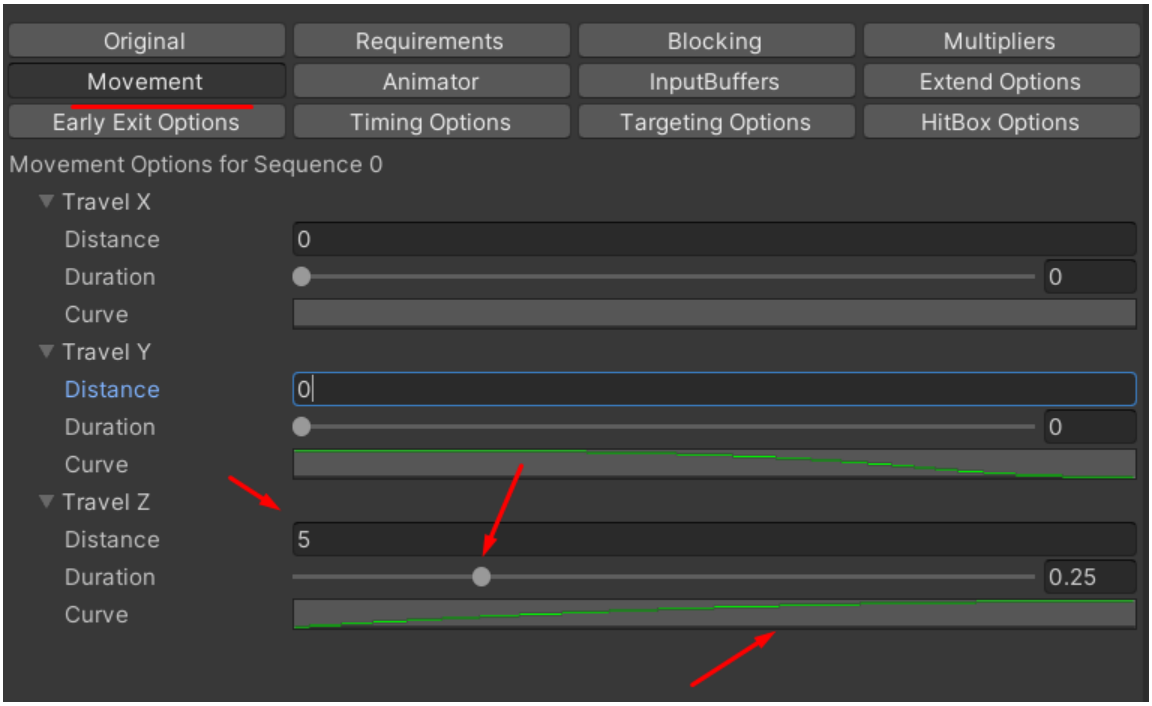
4. Our right click action is currently Attack_01.
5. Select Attack_01.
6. Let's customize this as a dash attack.
7. Click 'Edit Attack_01'.
8. We now see the action window.

a. 

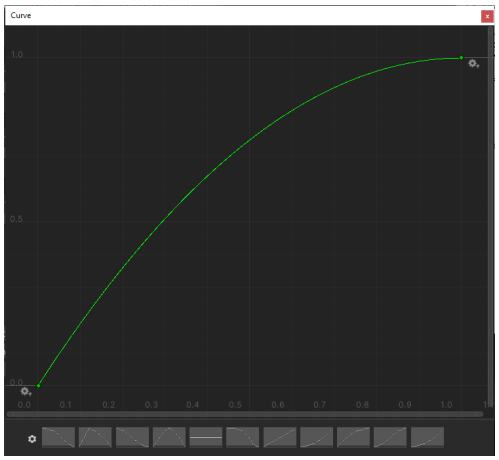
9. Select Animator.
10. Disable Root Motion (we want to control it ourselves as a dash).

a. 

11. Select Movement.
12. Modify it so the Action Character goes 5 units forward.

a. 

   i. These are in coordinates relative to the action character: z is forward, y is up, x is right. Negative numbers will go the opposite (e.g. -5 is backwards).

13. Let's do Travel Z distance to 5, e.g. travel forward 5 units.
14. Duration is how long it takes, in this case how quickly to travel 5 units forward.
15. Duration is 0-1 because it's the normalized duration of the total action duration.
16. Let's set Duration to somewhere like 0.25f, e.g. reach 5 units forward when the total action is 25% complete.
17. Let's assign an animation curve. This is the acceleration curve of the movement, let's assign this one:
18. 

Enter playmode. Success!

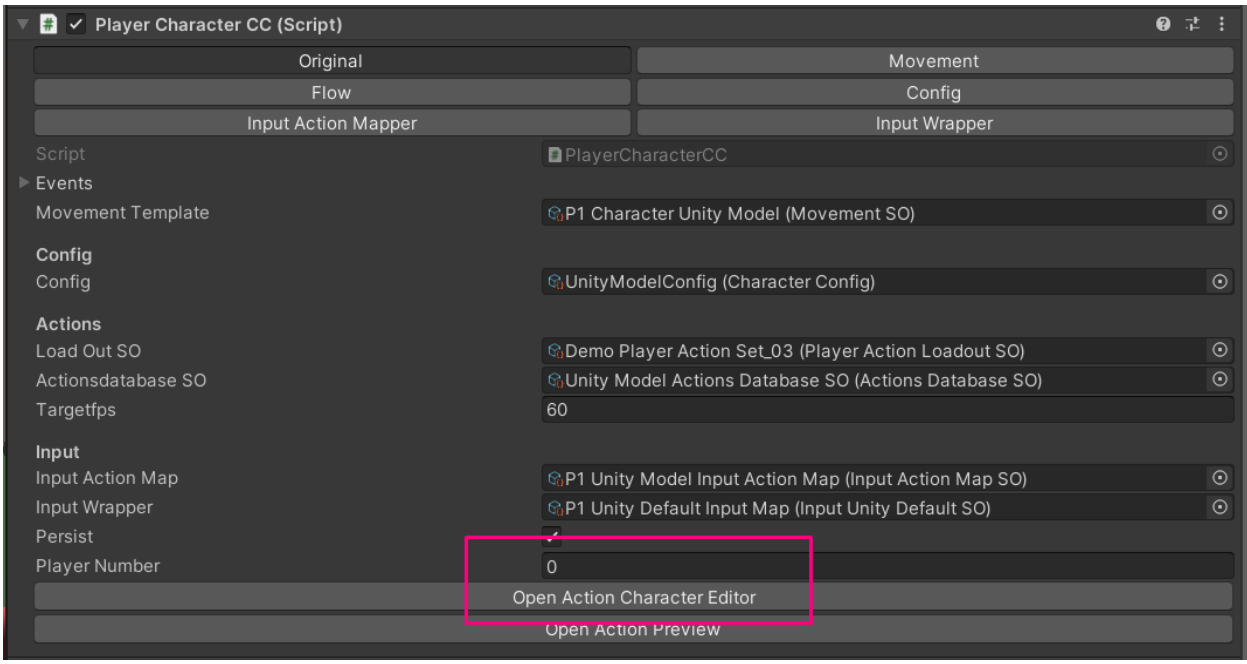
While in play mode, you can adjust the distances, durations, and curves to get the feel you want.
NOTE: If the info on the window disappeared, it's because you assigned the character from the scene. You must assign the character from its prefab in order to have it persist through playmode.

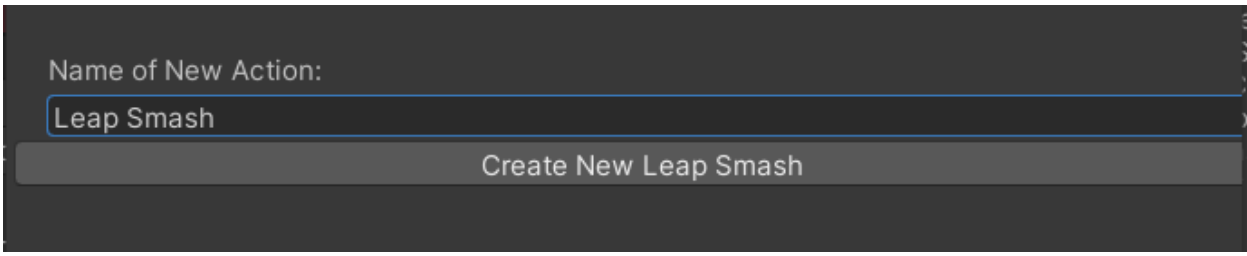
NOTE: if the character is still jumping with the action, remember to set the y movement value to 0. Setting either distance or duration to 0 will disable it.

Creating Your First Action!

Let's make a new action. We'll first create the action, then add it to a loadout.

Create the Action Asset

1. Open the Action Character Window.
    a. 

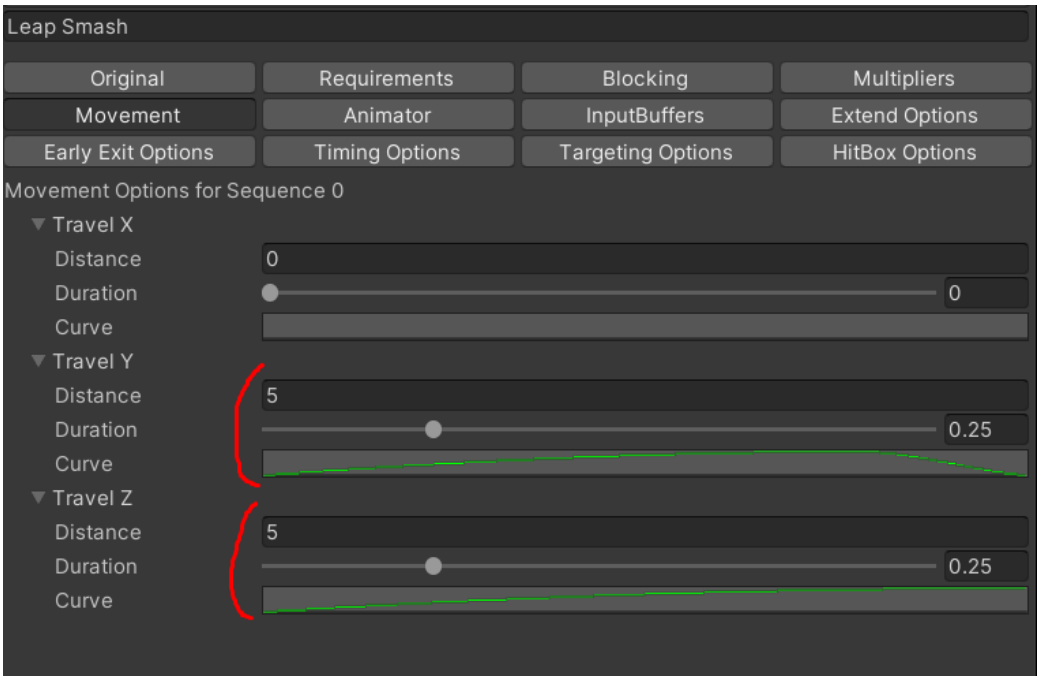
2. Let's call our new Action "Leap Smash".
    a. 

3. Click Create New Leap Smash.
4. Select where you want to store the asset.
    a. The demo action assets are stored at Assets/GWLPXL/Action Character/ActionCharacter/Actions but you'll probably want to store them somewhere unique to your project.

Edit the Action

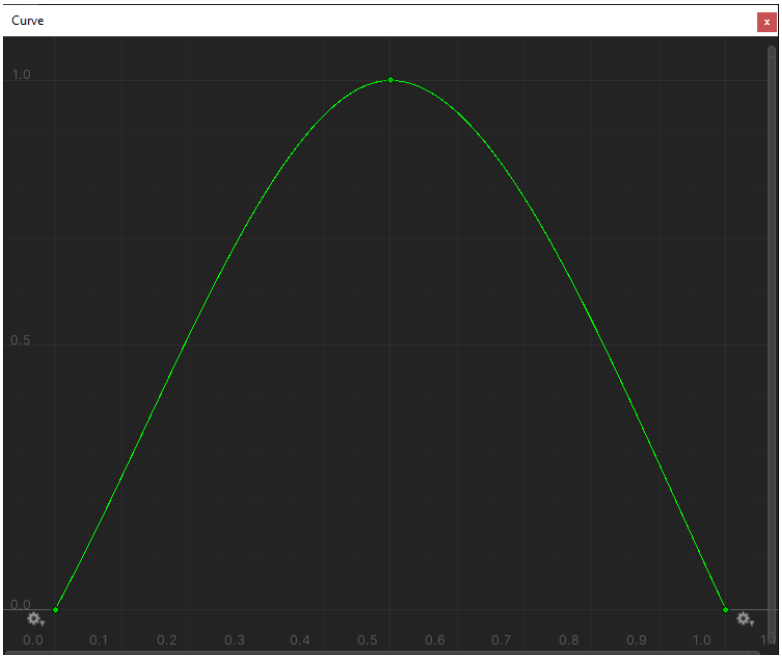
We can now edit the action! Let's make the new action a leap smash!

1. Edit the newly created action Leap Smash.
2. Under Movement, assign the following.

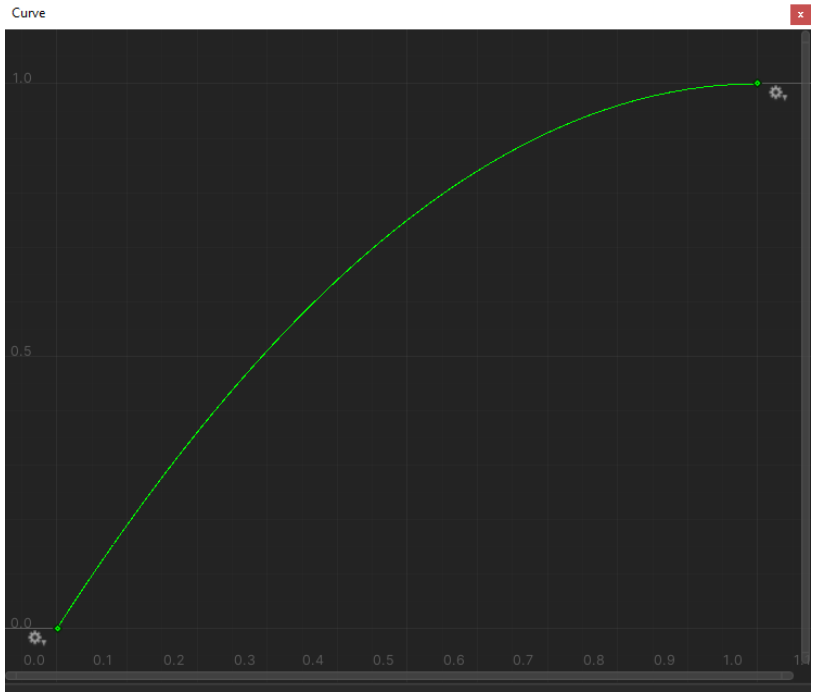
a. 

   i. Up 5 units and forward 5 units.

3. To get a Y curve that looks like a character is leaping and smashing down, adjust the Y Curve as so (imagine it as the vertical shape of the leap)

a. 

4. Let's choose a different curve for our Z.

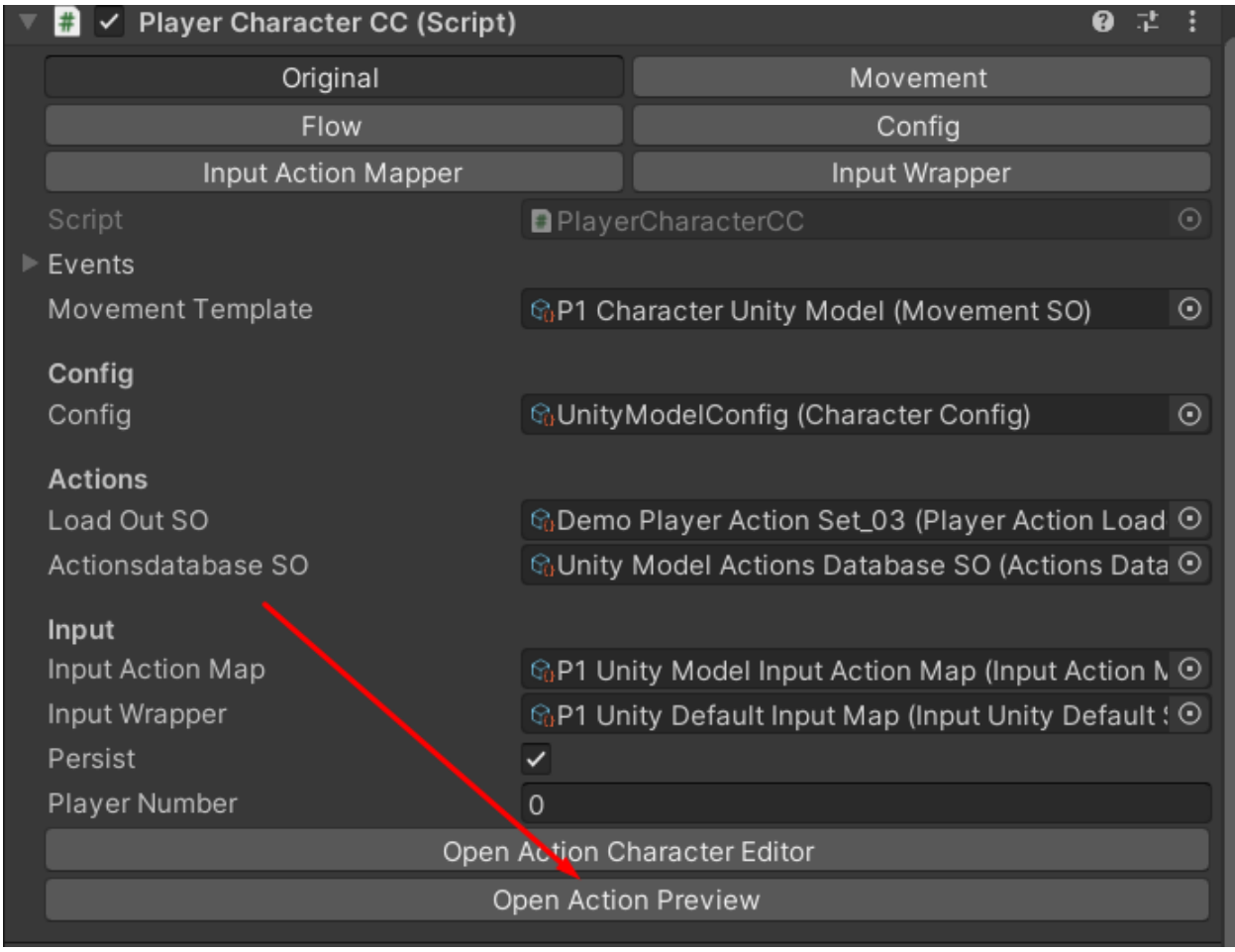
a. 

5. Now with the movement defined, we need to add the action to our loadout.

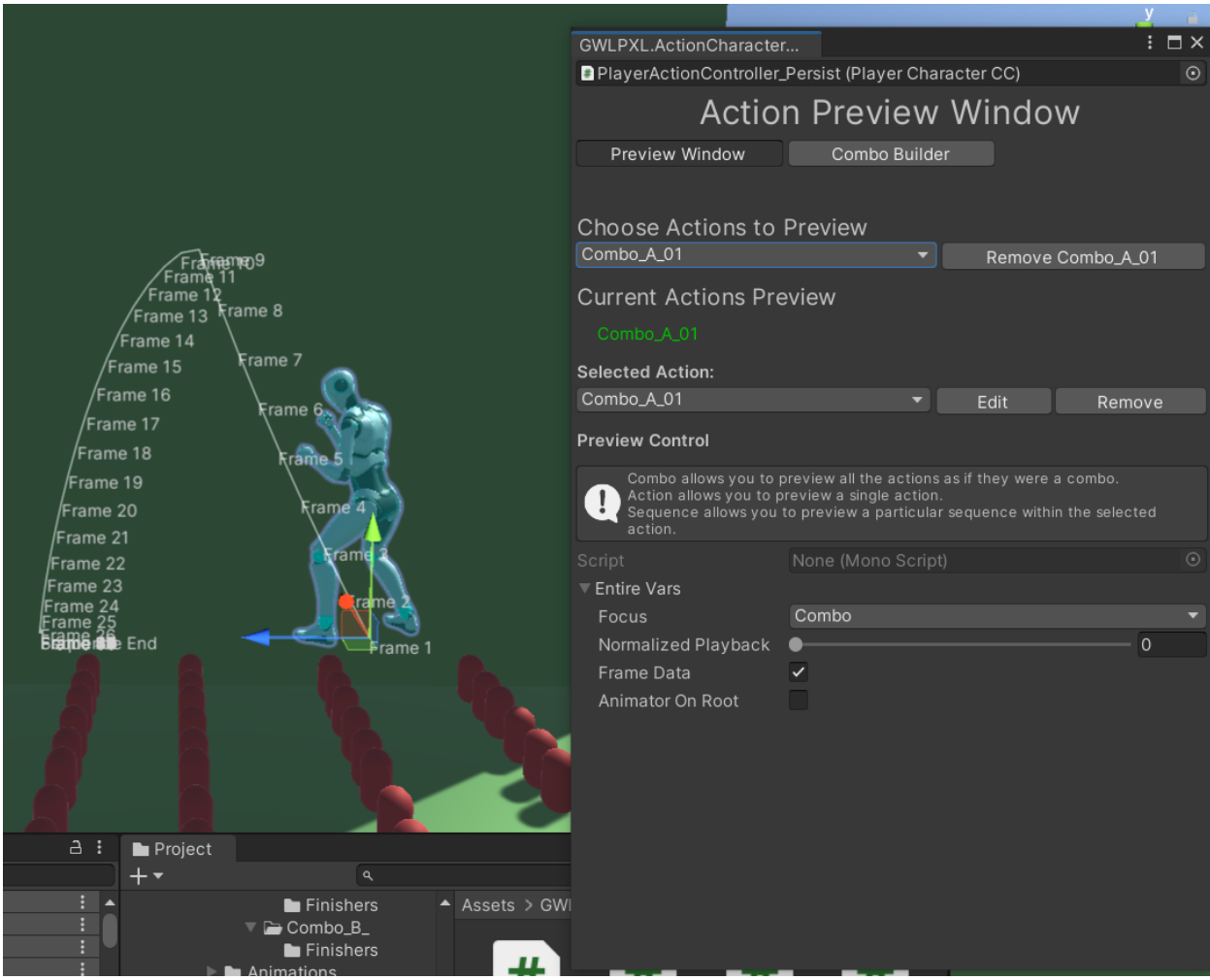
Previewing the Action

After creating an action, we might want to preview how it looks without entering Playmode all the time.

To see changes to your action after you've created one,

1. Open the Action Preview window.
    a. 

2. Select the Action to Preview and click "Add".
3. Change the Focus to "Combo".
4. Enable Frame Data.
5. Scrub the Normalized Playback to see the Action in action.

a. 

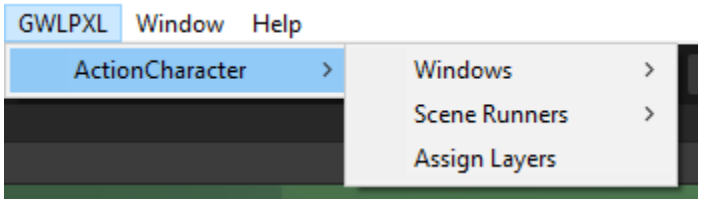
   i. You can add multiple actions to see how a combo chain looks.
   ii. You can change focus from Combo to Selected Action to Selected Sequence in Action.

6. Click Edit to open the Action editor window. Make changes and see them reflected in the editor without entering play!
7. Add multiple Actions and see how they look strung together!

You now have made an action and previewed it in the editor.

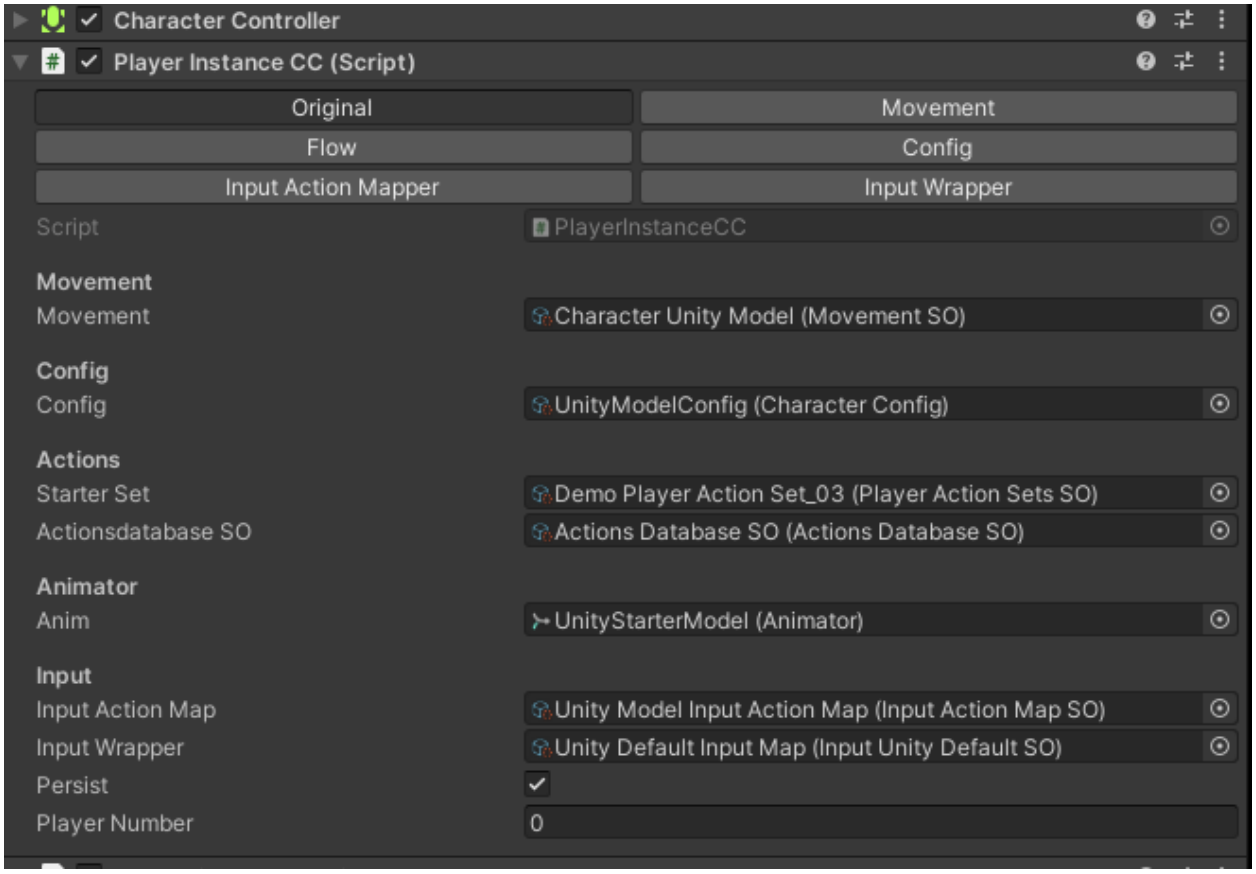
System Descriptions

Editor Menu



- Windows - Various custom editor windows for actions, characters, and others.
- Scene Runners - Singleton prefab required for both actions and hit boxes. These will auto create in the scene, but if you place them manually you can enable their debug options.
- Assign Layer- Will auto assign layers for the demo. You can find the layers and their names in the class ‘DemoDefaults.cs’.

Player Action Character

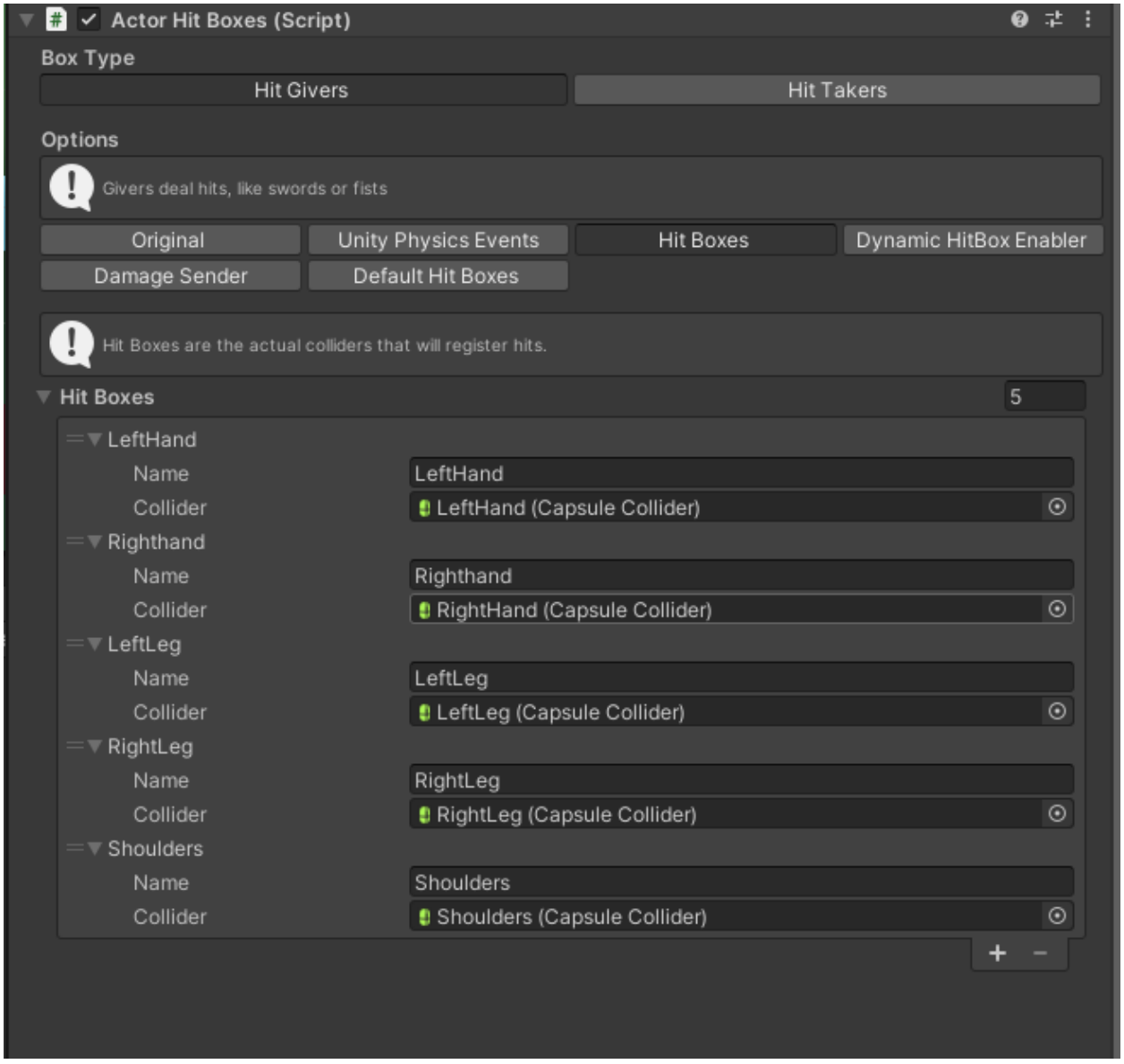


Player Instance CC inherits Player Character which inherits Action Character, which is the base abstract class for any characters that want to perform actions. In this case, the Player Instance CC overrides Player Character and uses Unity's Character Controller as the primary locomotion controller.

Extend Player Character to wrap the system in your own custom controllers.

Further details.

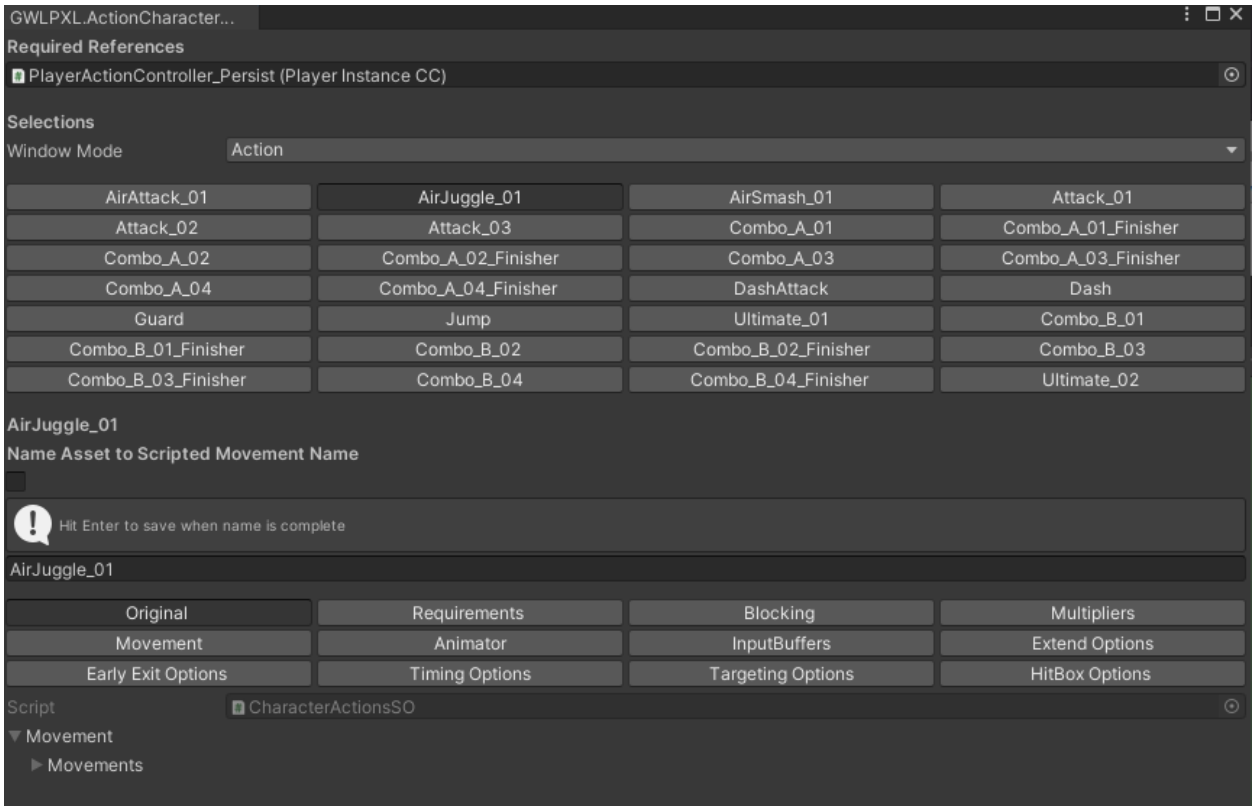
HitBoxes



HitBoxes can be any collider. Assign a unique name and collider to either the hitboxes of the HitGivers (e.g. things that do damage) or the hitboxes of the HitTakers (things that receive damage).

Further details.

Actions

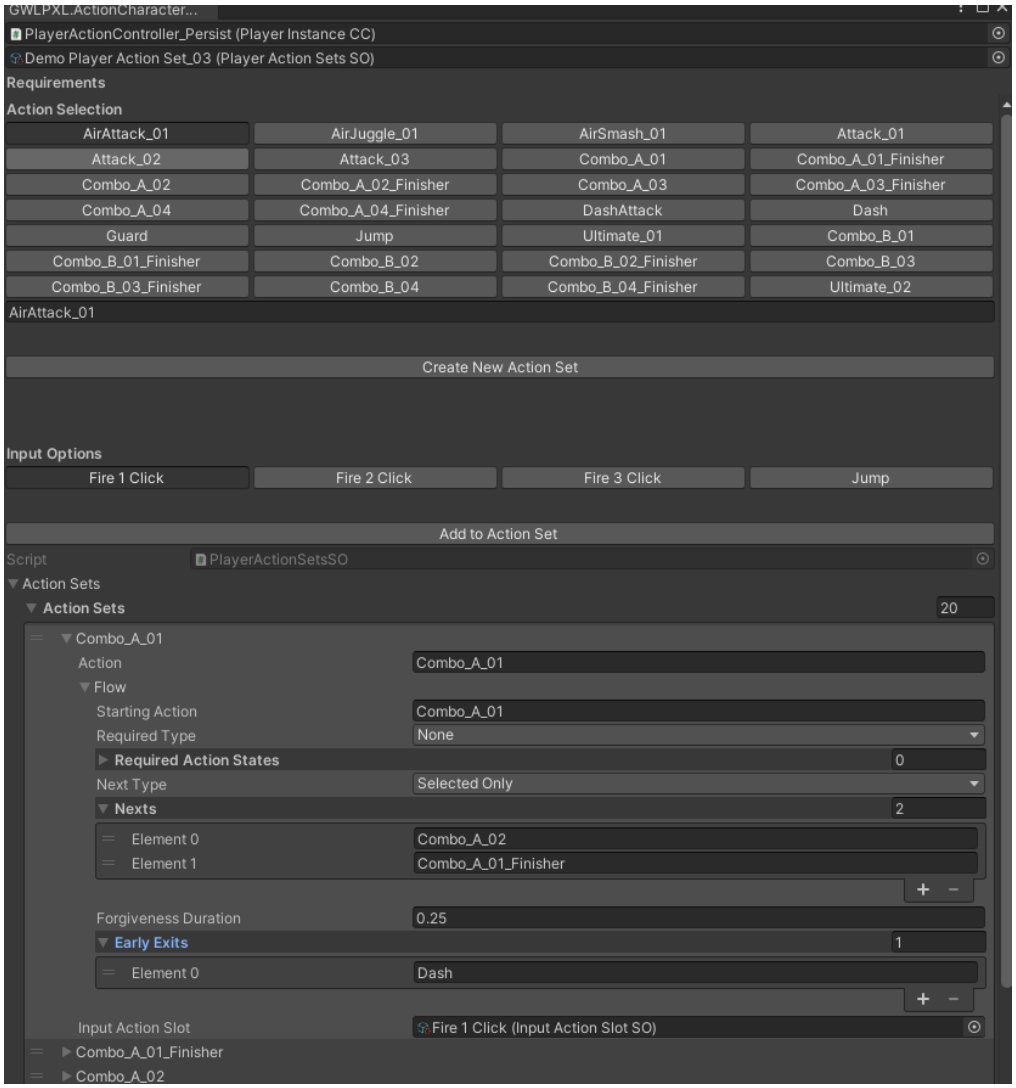


Actions are a unique state that a character enters. Actions can be customized in many ways.

[Further details.]

Action Sets

Actions, flow, and inputs (e.g. combo 1 to combo 2)



Action sets control the flow (i.e. combos) of the actions. For instance,

- Combo_A_01 can transition to Combo_A_02 or Combo_A_01 Finisher
- Dash can exit out of the action before it is finished.
- Inputs are only available for player actions.
- Forgiveness Duration means the action remains valid for a combo 0.25 seconds after complete.

System Extensions

Action Character and Hit Boxes have numerous events to subscribe to in order to customize them to your own games. Below are examples of the included scripts to help.

Action Subscribes

If you want to know when an action has started or ended, this is helpful.

ActionSub - base mono to inherit
ActionFXSub - example
ActionSFXSub - example

You can subscribe to the Action Enabled and Disabled events on the Action Character.

```
protected ActionCharacter charactercc;

     protected virtual void OnEnable()
     {
           charactercc.Events.OnActionEnded += ActionDisabled;
           charactercc.Events.OnActionStarted += ActionEnabled;
     }
```

The class “ActionSub” provides a base mono for you. Inherit and Override ActionEnabled and ActionDisabled to write your own custom code.

HitBox Subscribes

If you want to know when hit boxes are enabled and disabled, these are helpful.

You can subscribe to events in both HitGivers and HitTakers. Below are examples of how and also included base monos that you can use to inherit and override.

Hit Giver

HitBoxHitGiverSub - base mono
DamageAreaFX - example

```
 protected ActorHitBoxes boxes;

      protected virtual void OnEnable()
      {
            boxes.HitBoxes.HitGivers.OnHitGiverEnabled += HitGiverEnabled;
            boxes.HitBoxes.HitGivers.OnHitGiverDisabled +=
HitGiverDisabled;
      }
```

HitTaker

HitBoxHitTakerSub - base mono

```
protected ActorHitBoxes boxes;

      protected virtual void OnEnable()
      {
            boxes.HitBoxes.HitTakers.OnHitTakerEnabled += HitTakerEnabled;
            boxes.HitBoxes.HitTakers.OnHitTakerDisabled +=
HitTakerDisabled;
      }
```

Hit Damage and Take Damage Subscribes

If you want to know when a damage and take damage event is sent, these are helpful.

Hit Damage

HitDamage- base mono
DamageFX - example
FreezeFrameSub - example

```
protected ActorHitBoxes hitboxes;

      protected virtual void OnEnable()
      {

hitboxes.HitBoxes.HitGivers.HitSender.DamageEvents.OnHitCollision +=
DamageCollision;
            hitboxes.HitBoxes.HitGivers.HitSender.DamageEvents.OnHitTrigger
+= DamageTrigger;
      }
```

Take Damage

HitTakeDamage- base mono
FauxKnockback - example

```
  protected ActorHitBoxes boxes;
      protected virtual void OnEnable()
      {

boxes.HitBoxes.HitTakers.HitReceiver.HitTakerEvents.OnHitCollision +=
HurtCollision;

boxes.HitBoxes.HitTakers.HitReceiver.HitTakerEvents.OnHitTrigger +=
HurtTrigger;
      }
```