Setting Up Input on an Actor

How to set up input on an Actor

When developing your game, there may be instances when you want to allow the player to perform some form of **Input** on an Actor in your level. For example, perhaps you have a treasure chest that you want the player to be able to open when they approach it, or a light you want them to be able to turn on/off, or some other form of interactable object you want to affect in some way when the player presses a button.

Setting up Input for an Actor gives you a level of control over how and when the Actor responds to player input.

Implementation Guide

This guide covers the basic methods on how to **Enable Input** or **Disable Input** for an Actor. By Enabling Input on an Actor, you can allow a player to press a button or key and execute events that affect the Actor in some way (be it turn a light on or off, open or close a door, activate something, etc.).

The Enable/Disable Input Usage section also covers an approach to Enabling/Disabling Input using a **Trigger Volume** to determine if a player is near the Actor in the level. This is useful

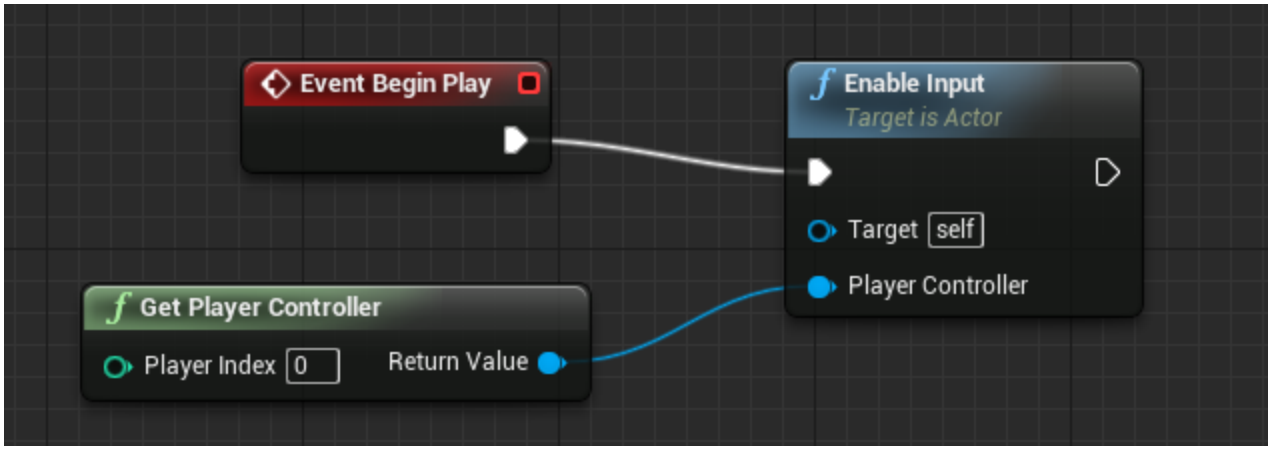
when you want a player to affect the Actor only when they are near or in range of the Actor to conceivably affect it in a real-world scenario.

Enabling Actor Input

> For this example, we are using the **Blueprint First Person Template with Starter Content**.

The steps below will show you how to Enable Input for an Actor:

1. Inside the **StarterContent/Blueprints** folder, open the **Blueprint_Effect_Fire** asset.
2. On the **Event Graph** tab, **Right-click** in the graph and search for then add the **Event Begin Play** node.
3. In the graph, **Right-click** search for and add the **Get Player Controller** node.
4. In the graph, **Right-click** search for and add the **Enable Input** node.
5. Connect the out pin of the **Event Begin Play** to the in pin of the **Enable Input** node.
6. Connect the **Get Player Controller** node to the in **Player Controller** pin on the **Enable Input** node.



This is the basic method in which you can enable input on an Actor.

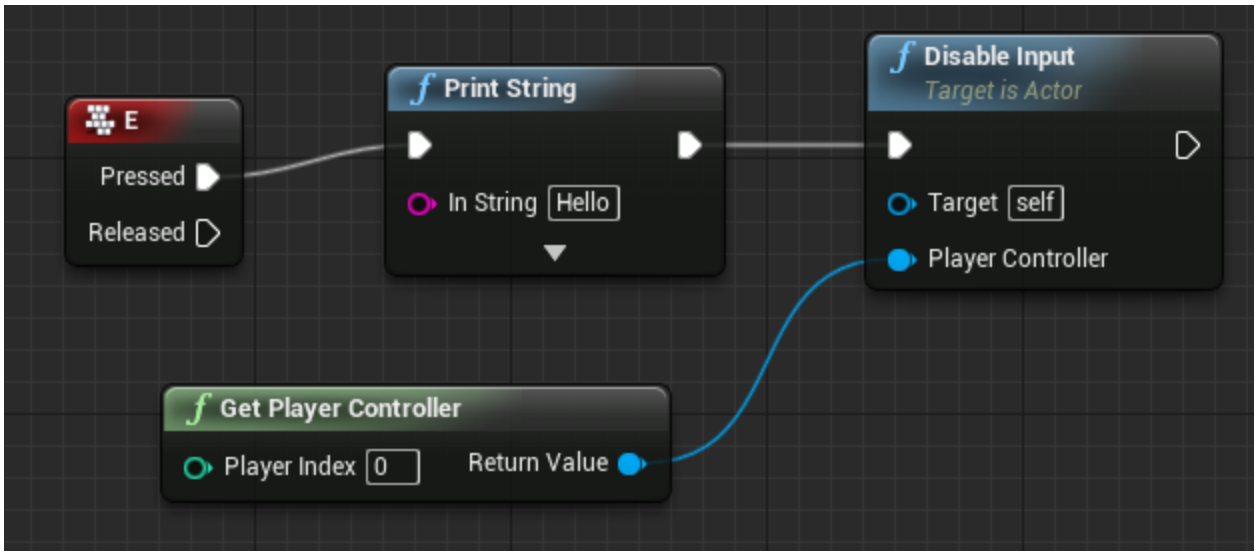
The **Enable Input** node requires that a **Target** (usually Self - the Actor itself) as well as a **Player Controller** (the player that will be providing the input) be specified. With this setup, you could now search for and add **Input Event** nodes (such as a Key or Mouse Button press) and perform actions that affect the Actor when those keys are pressed.

In the example above, the **Event Begin Play** node states that when the game is started (provided the Actor exists in the level), **Enable Input** for the **Player Controller** specified. The

default Player Controller uses **Player Index 0** for Single Player games, if you were to have a multiplayer scenario, you could specify which player through the **Player Index** value.

Disabling Actor Input

Just as you can Enable Input on an Actor, you can also Disable Input for an Actor by using the **Disable Input** node.



In the example above, we used the aforementioned method of enabling input on the Actor then added an **E** Key Event. When the **E** key is pressed, we print to the screen some text then Disable Input on the Actor so that they can no longer provide input to the Actor.

As with the **Enable Input** node, the **Disable Input** node requires that a **Target** and **Player Controller** be specified.

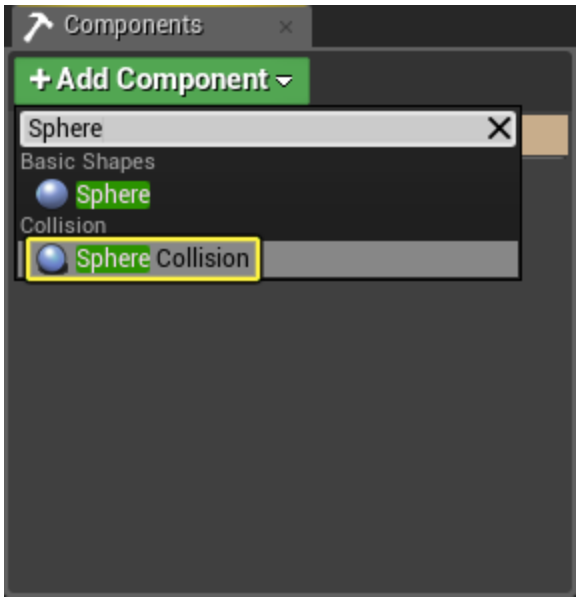
Enable/Disable Input Usage

While the methods noted above work for Enabling/Disabling Input, typically you would want some rules to govern when to allow the Actor to receive Input from a player. If we were to use the method above of Enabling Input on Event Begin Play and had a Key Press that turned on or off a light (for example), the player would be able to turn that light on/off from anywhere in the world.

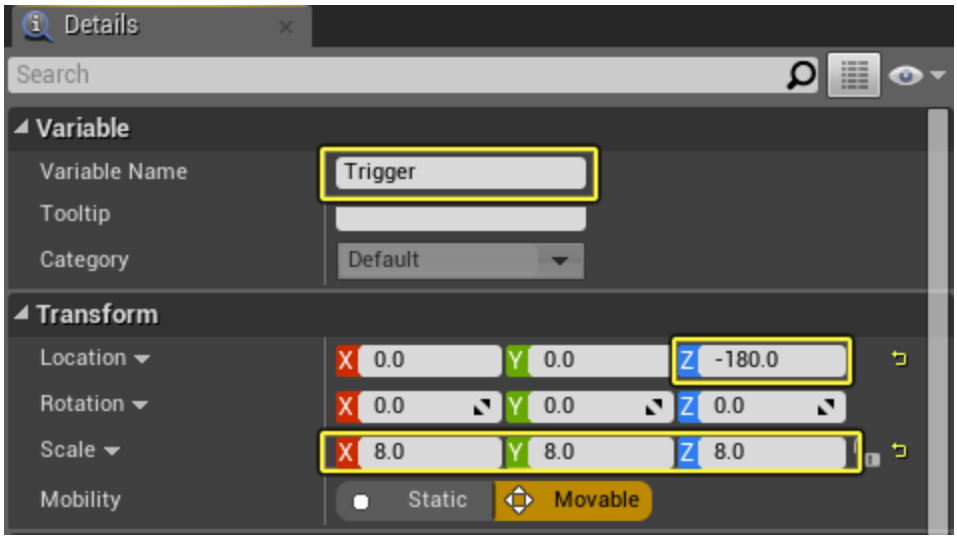
We would want to Enable/Disable Input based on if the player is near the light, similar to the below:

1. Inside the **StarterContent/Blueprints** folder, open the **Blueprint_CeilingLight** asset.

2. In the **Components** panel, click the **Add Component**, then search for and add **Sphere Collision**.

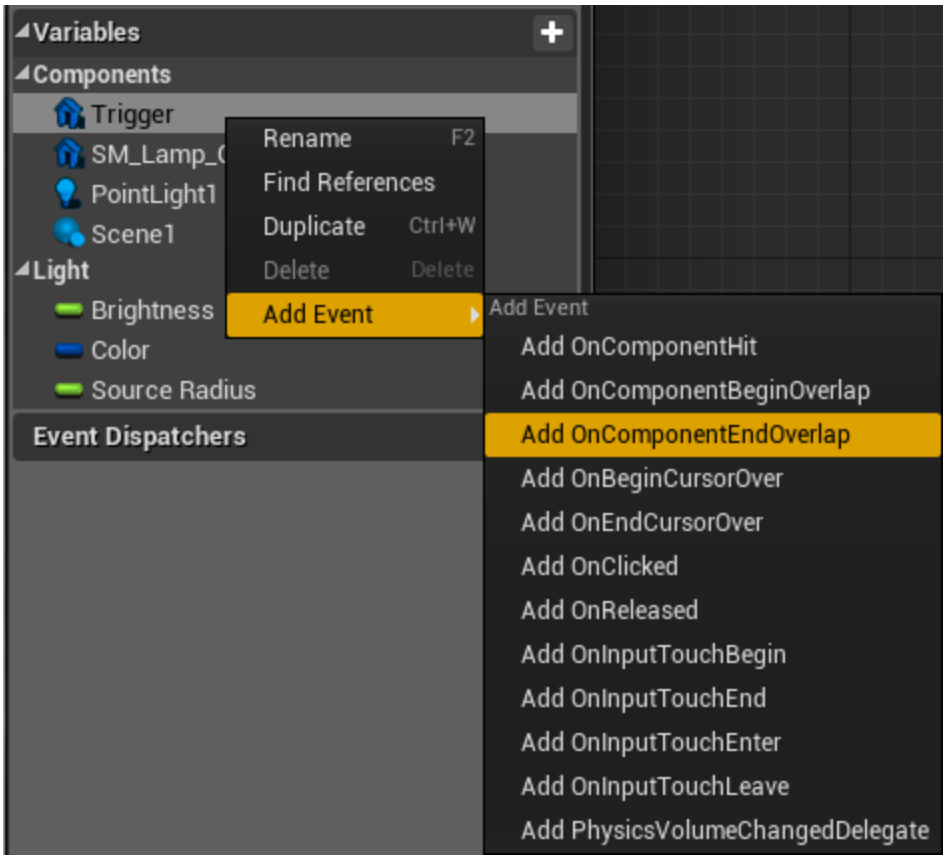


3. In the **Details** panel, set the **Variable Name** to **Trigger**, the **Z Transform** to **-180.0**, and **Scale** for **X, Y, Z** to **8**.

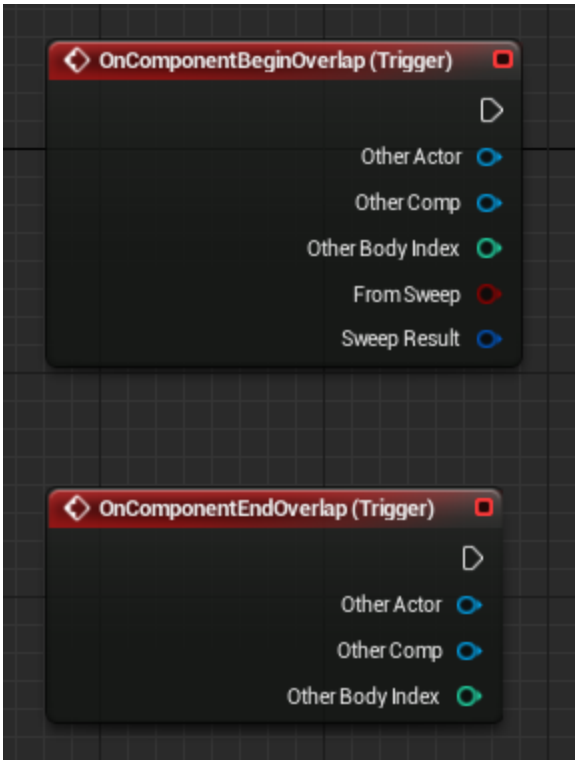


We are going to Enable Input (when the player is inside) or Disable Input (when the player is outside) of the sphere.

4. Return to the **Event Graph**.
5. In the **MyBlueprint** window, **Right-click** on the **Trigger** and choose **Add Event → Add OnComponentBeginOverlap**.

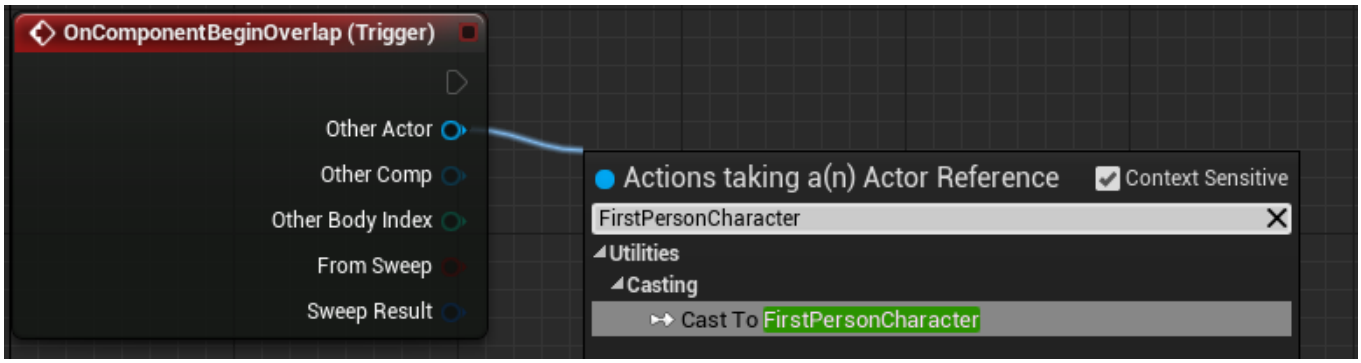


6. **Right-click** on **Trigger** again and choose **Add Event → Add OnComponentEndOverlap**.



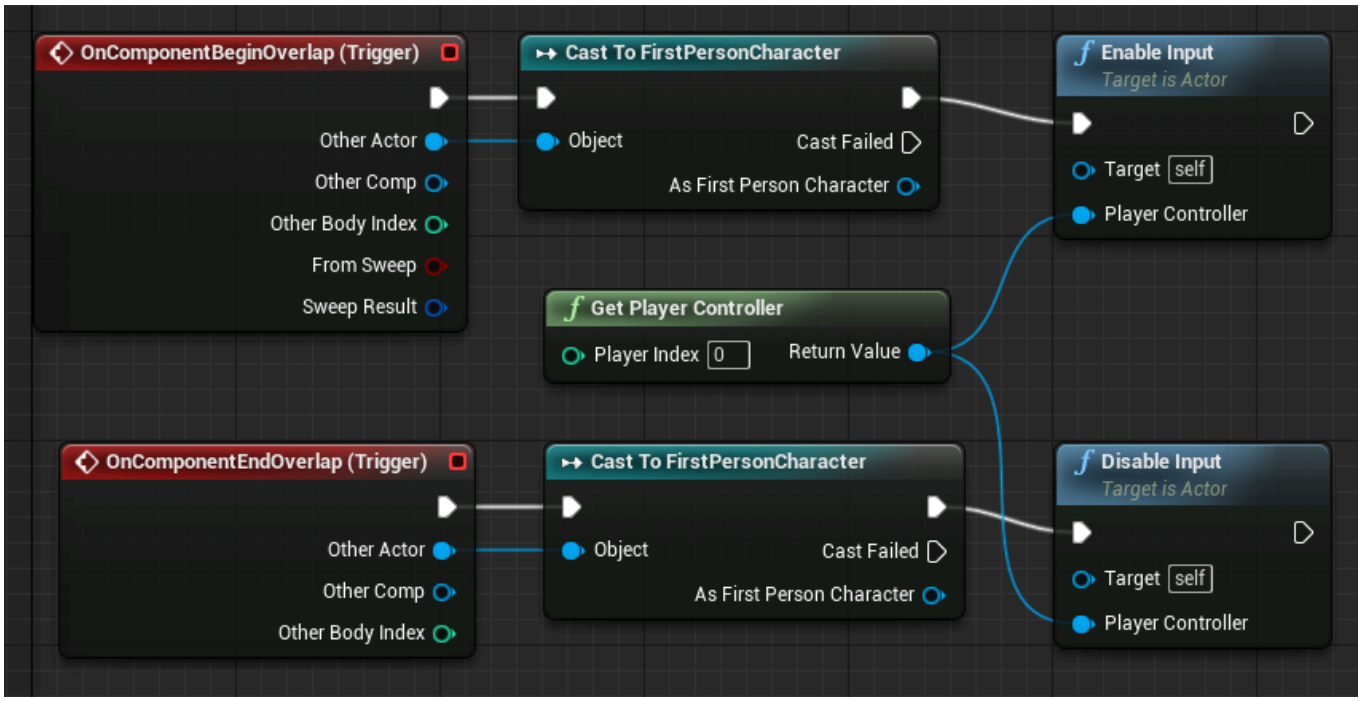
These two Event nodes should be added to the graph.

7. Drag off the **Other Actor** pin of the Begin Overlap node and search for and add the **Cast To FirstPersonCharacter** node.



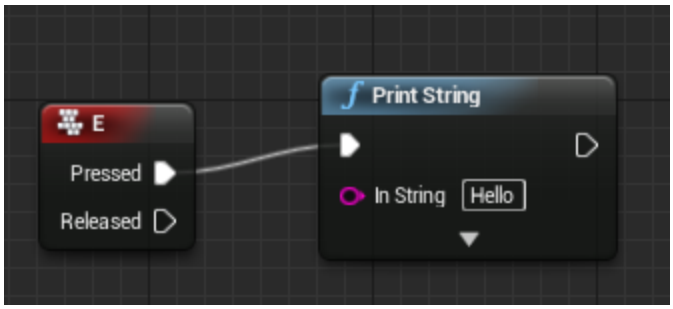
Here we are saying that when an Actor called **FirstPersonCharacter** (which is the default player character for the Project Template) overlaps the **Trigger**, do something.

8. Drag off the **Other Actor** pin of the End Overlap node and add the **Cast To FirstPersonCharacter** node.
9. **Right-click** and add the **Get Player Controller** node, **Enable Input**, and **Disable Input** nodes.
10. Connect the nodes as shown below.



Input will now only be enabled when the player enters the trigger and is disabled when the player leaves the trigger. This prevents the player from affecting the Actor from anywhere in the world and confines it to only when they are inside the trigger volume we created.

11. **Right-click** and add the **E** Key Event and it to a **Print String** node.



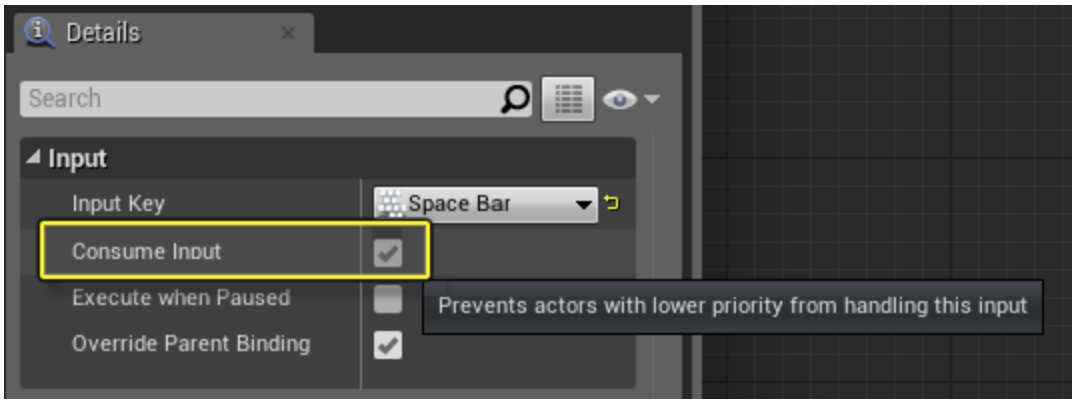
12. **Compile** and **Save**, then close the Blueprint. (If you receive a Warning, ignore it and proceed to the next step.)
13. Drag the **Blueprint_CeilingLight** into the level, then click the **Play** Button to play in the Editor.

When playing, if you press **E** when far away from the light, nothing should happen. When you get close to the light (inside the trigger) and press **E**, the text **Hello** should appear in the upper left corner of the window. For this example, we hooked up a **Print String** node, however you could toggle the light color, intensity, or other settings when the key is pressed and the player is inside the trigger sphere.

Input Details

Replace the **E** Key Event in the example above and try using a **Space Bar** Key Event instead. When you play in the Editor, notice what happens. You can Jump by pressing **Space Bar** when outside of the trigger for the light, however when you enter the trigger and press **Space Bar**, you no longer Jump but instead call the **Print String** and text **Hello**; this is due to **Input Priority**.

When Input commands are shared (as is the case here with Jump and the input we assigned in our Light both tied to **Space Bar**), lower priority actions are ignored. Inside the light Blueprint, if you click on the **Space Bar** Key Event and look in the **Details** panel, you should see an option for **Consume Input**.



If you un-check the **Consume Input** checkbox and play in the Editor again, you should now be able to Jump both inside and outside of the light's trigger volume. When inside the light's trigger volume, by pressing **Space Bar** you will also call the **Print String** node and the **Hello** text to be displayed.

Also in the **Input Details** window are options for **Execute when Paused** (which allows you to press the Key during the paused state and execute commands) and **Override Parent Binding** (which allows you to remove any bindings set in any parent classes).