LECTURE 3

Building and Using Actor Classes

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Show how to add components to a Blueprint
- Introduce the Construction Script
- Present various types of events
- Demonstrate how to spawn, destroy, and reference Actor instances
- Show how to do input mappings

Outcomes

By the end of this lecture you will be able to

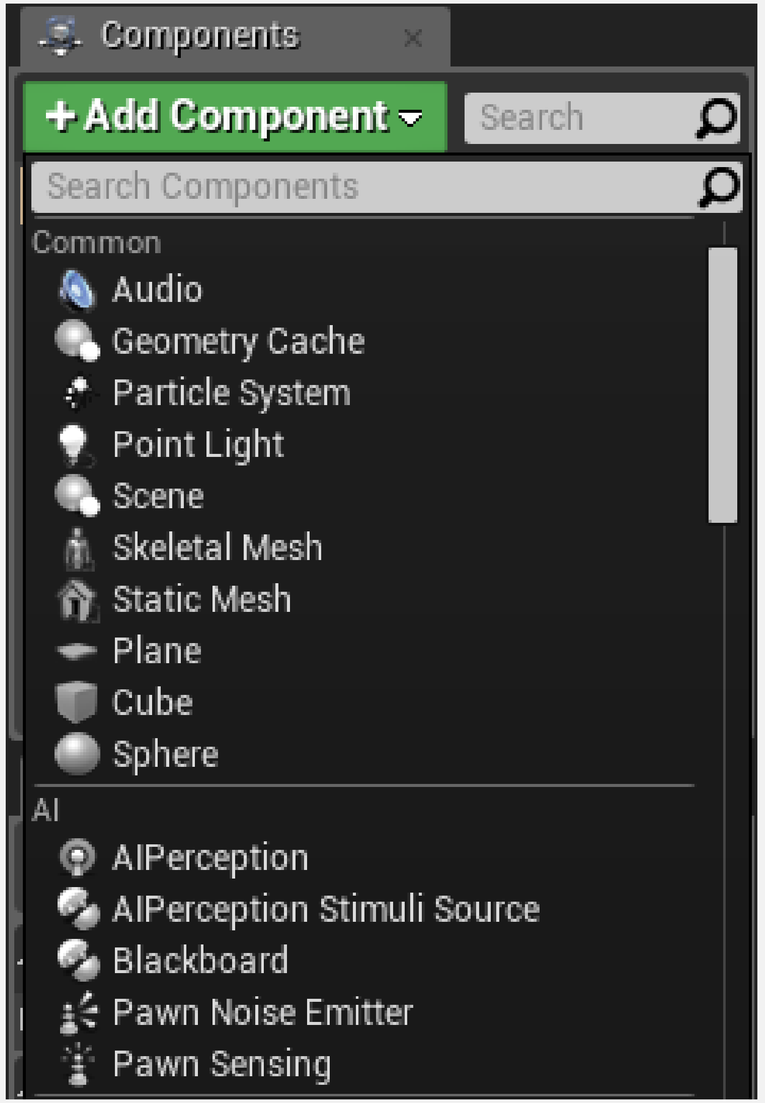
- Add various types of components to a Blueprint
- Manage Actor instances
- Create collision and mouse events
- Create input events and map keys and axes

COMPONENTS

Components are ready-to-use classes that can be used inside Blueprints. Several features can be included in a Blueprint using only components.

To add components to a Blueprint, use the **Components** panel in the **Blueprint Editor**.

The image on the right shows the **Components** panel for a new Blueprint with some component options that are displayed when the **Add Component** button is pressed.

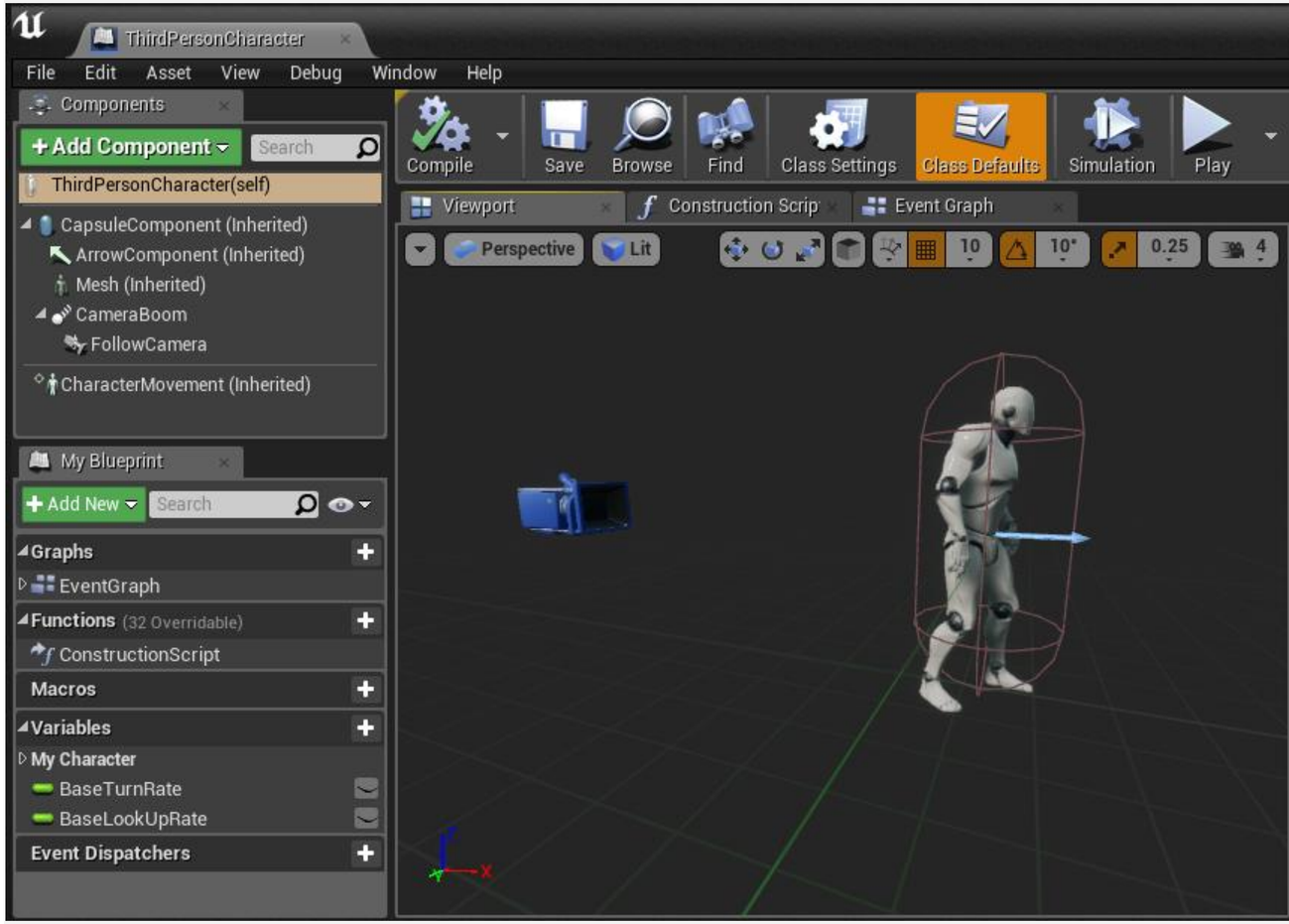


COMPONENTS: VIEWPORT

The visual representation of the components can be seen in the **Viewport**.

The image on the right shows the components that are part of the **ThirdPersonCharacter** Blueprint from the Third Person template. The components that have “(Inherited)” next to the name were inherited from the Character class.

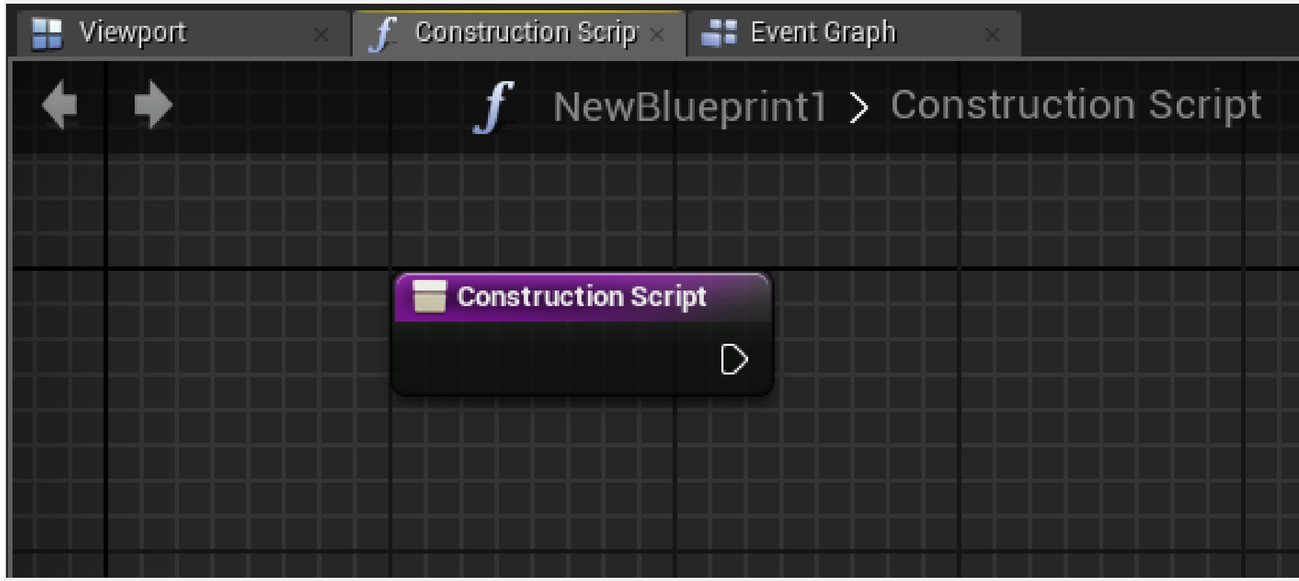
- The **CapsuleComponent** is used for collision testing.
- The **Mesh** component is the Skeletal Mesh that visually represents the character.
- The **FollowCamera** component is the camera that will be used to view the game.
- The **CharacterMovement** component contains various properties that are used to define the movement.



CONSTRUCTION SCRIPT

The **Construction Script** is a special function that all Actor Blueprints perform when the Blueprint is first added to the Level, when there is a change to its properties, or when the class is spawned at runtime. The Construction Script has a separate graph where the actions to be performed can be placed.

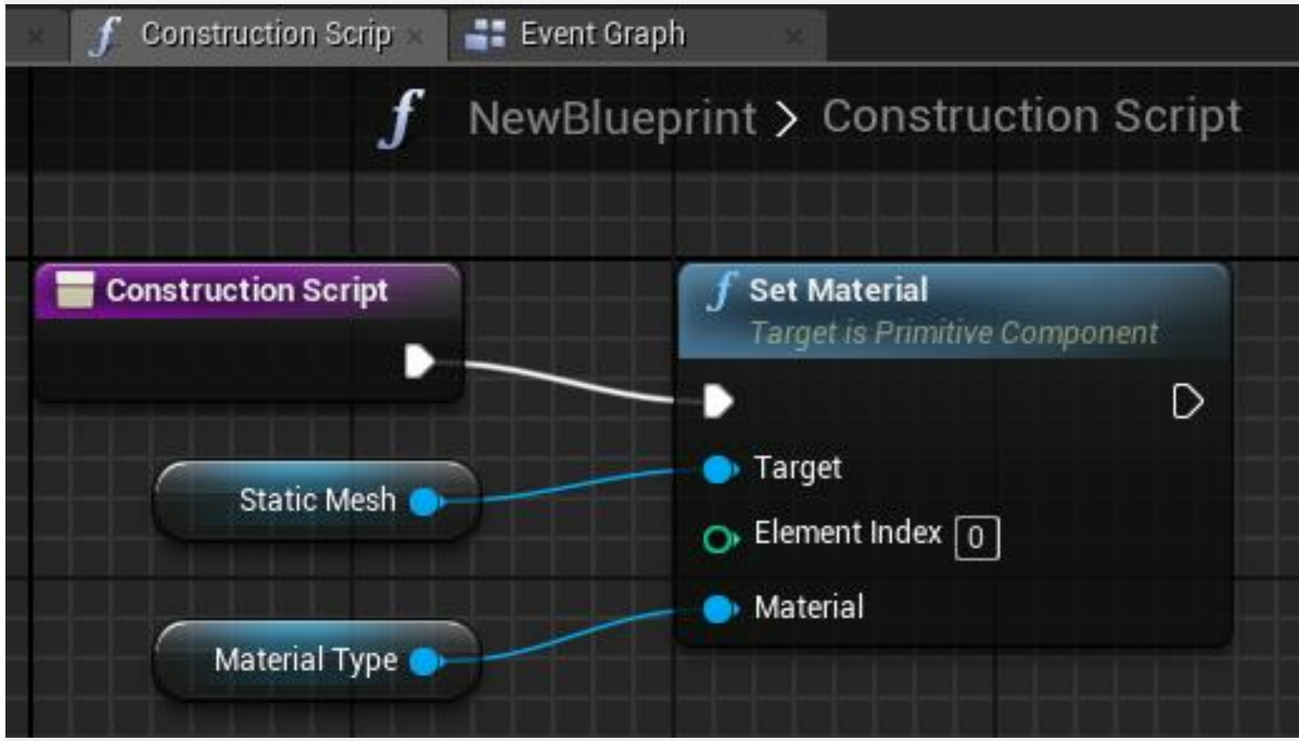
It is important to note that the Construction Script won’t run on placed Actors when the game starts.



CONSTRUCTION SCRIPT: EXAMPLE

The Construction Script seen on the right uses the **Set Material** function to define a **Static Mesh** component's Material type according to the Material selected in the editable variable **Material Type**.

Whenever the **Material Type** variable is modified, the **Construction Script** runs again, updating the object with the new Material.

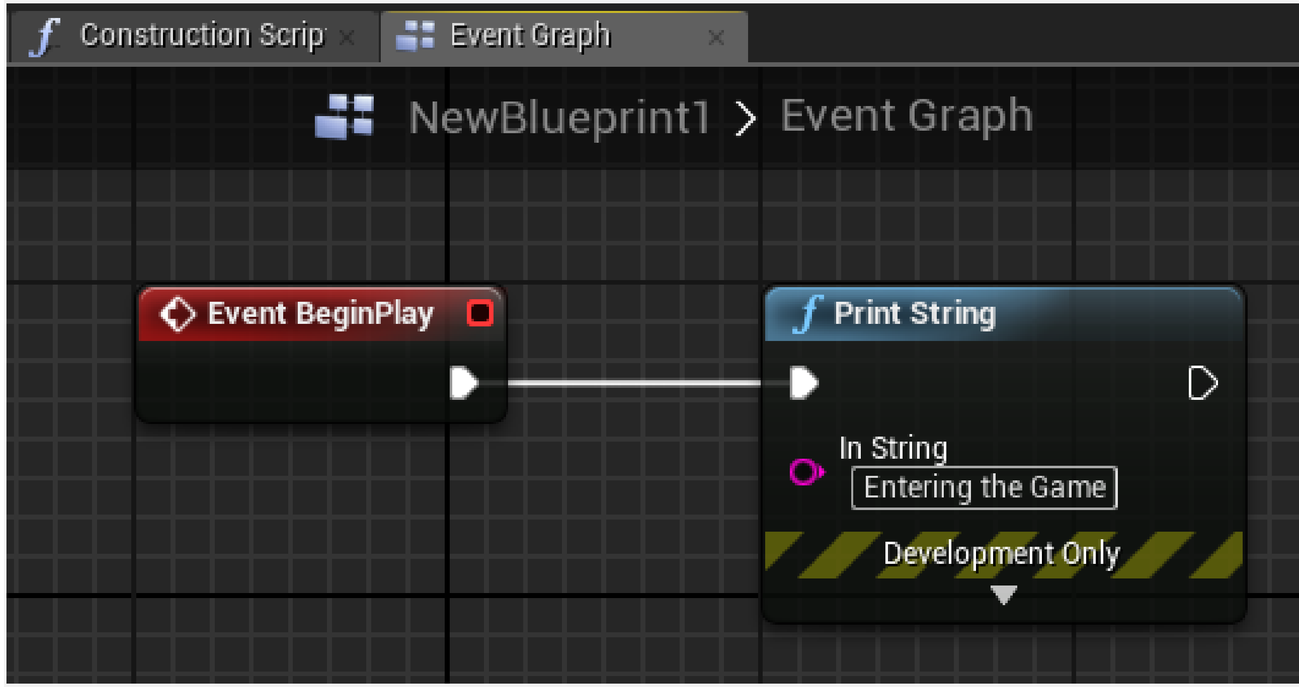


EVENTS

BEGIN PLAY EVENT

Events allow communication between Unreal Engine and Actors. A common example is the **BeginPlay** event.

The BeginPlay event is triggered when the game starts for an Actor. If the Actor is spawned in the middle of the game, then this event is triggered immediately.



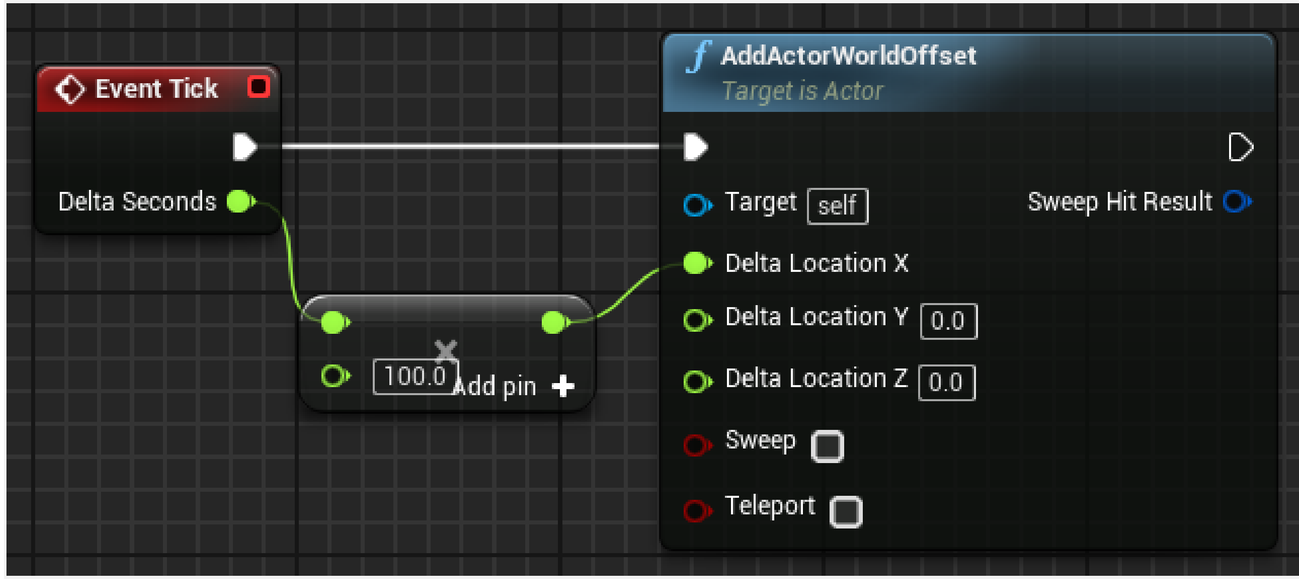
TICK EVENT

There is an event named “**Tick**” that is called every frame of the game. For example, in a game that is running at 60 frames per second, the Tick event is called 60 times in a second.

The Tick event has a parameter known as **Delta Seconds**, which contains the amount of time that has elapsed since the last frame.

In the Tick event illustrated on the right, an Actor moves along the X axis at the speed of 100 centimeters per second.

Use the Tick event only when necessary, as it can affect performance.



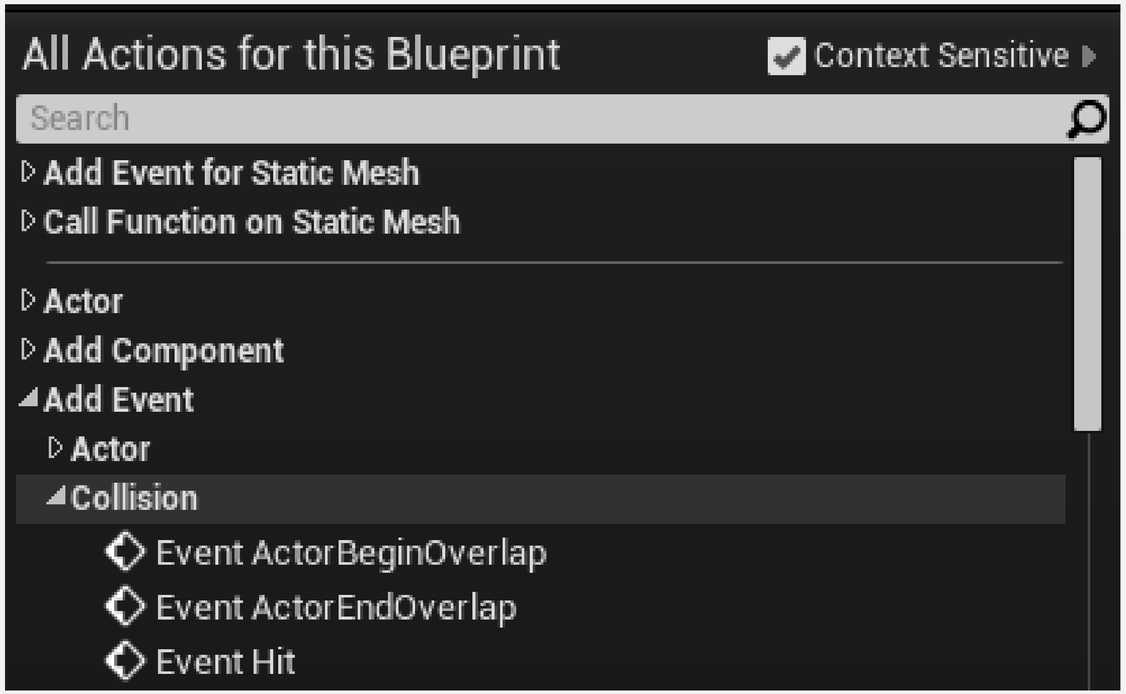
COLLISION EVENTS

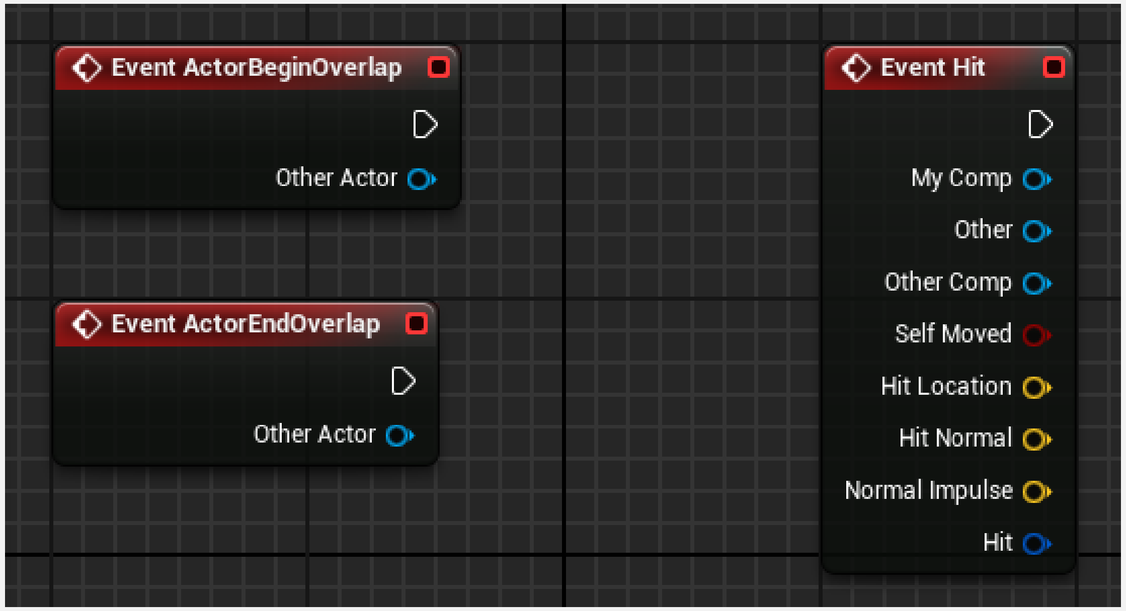
Collision events are triggered when two Actors collide or overlap.

Event ActorBeginOverlap will execute when two Actors start overlapping and the **Generate Overlap Events** property of both Actors is set to “**true**”.

Event ActorEndOverlap will execute when two Actors stop overlapping.

Event Hit will execute if the **Simulation Generates Hit Events** property of one of the Actors in the collision is set to “**true**”.





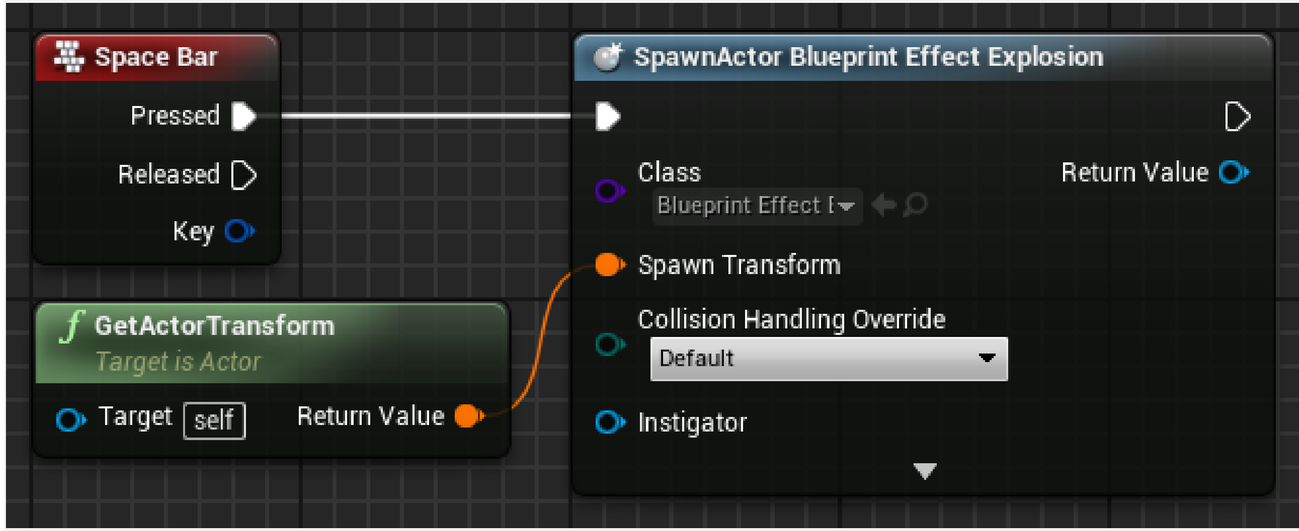
ACTOR INSTANCES

SPAWNING ACTORS

Spawn Actor from Class is a function that creates an Actor instance using the class and transform specified.

The **Collision Handling Override** input defines how to handle the collision at the time of creation. The output parameter **Return Value** is a reference to the newly created instance.

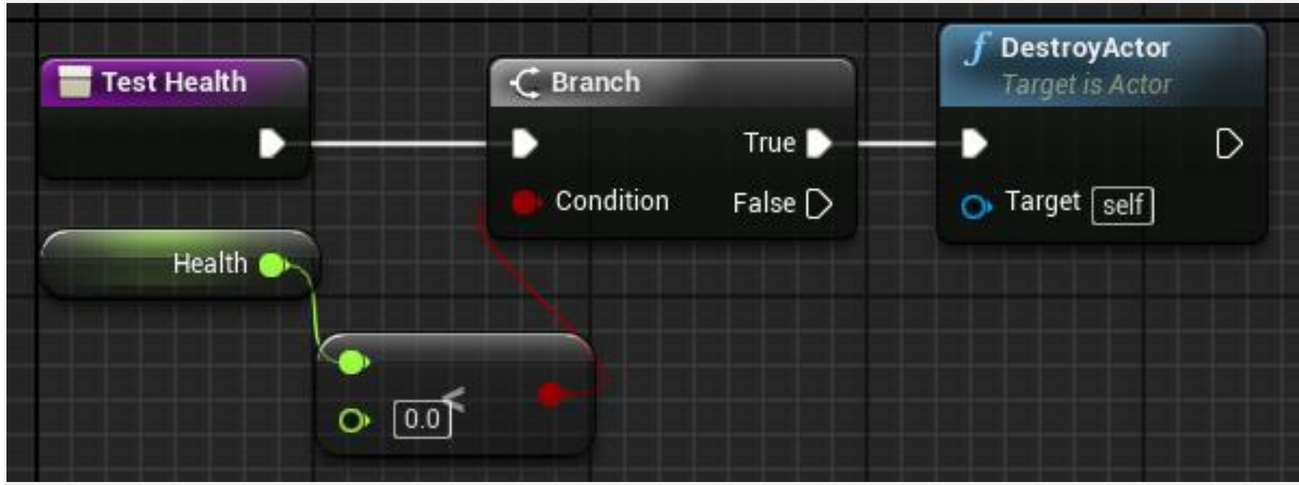
In the example on the right, when the **space bar** is pressed, an instance of the **Blueprint Effect Explosion** class is created at the same location (transform) of the current Blueprint.



DESTROYING ACTORS

The **DestroyActor** function removes an Actor instance from the Level at runtime. The instance to be removed must be specified in the **Target** parameter.

The image on the right shows a function named “**Test Health**” that will check if the value of the **Health** variable is less than zero. If “**true**”, the current instance of this Blueprint, which is represented by “**self**”, will be destroyed.



GET ALL ACTORS OF CLASS

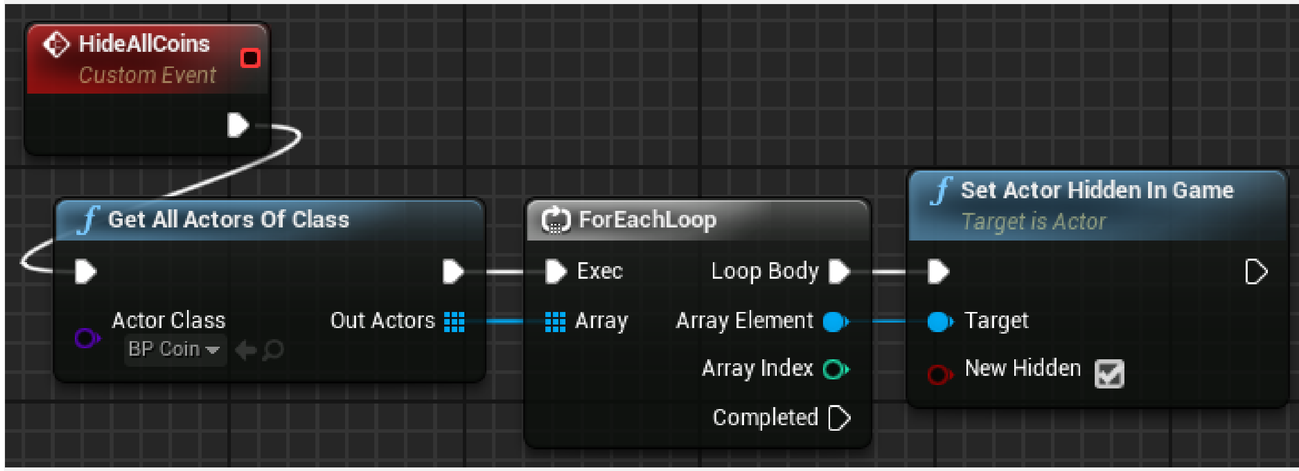
Get All Actors Of Class is a function that gets the references of all the Actors in the current Level who belong to a specified class.

The **Actor Class** parameter specifies the class that will be used in the search.

The **Out Actors** output parameter is an array containing references to the Actor instances of the specified class found in the Level.

In the image on the right, **Get All Actors Of Class** returns an array of **BP_Coin** Actors, then uses a **ForEachLoop** node to set the **New Hidden** property to "**true**" in the **Set Actor Hidden In Game** function for each Actor in the array.

This can be a costly operation. Do not use it in the Tick event.



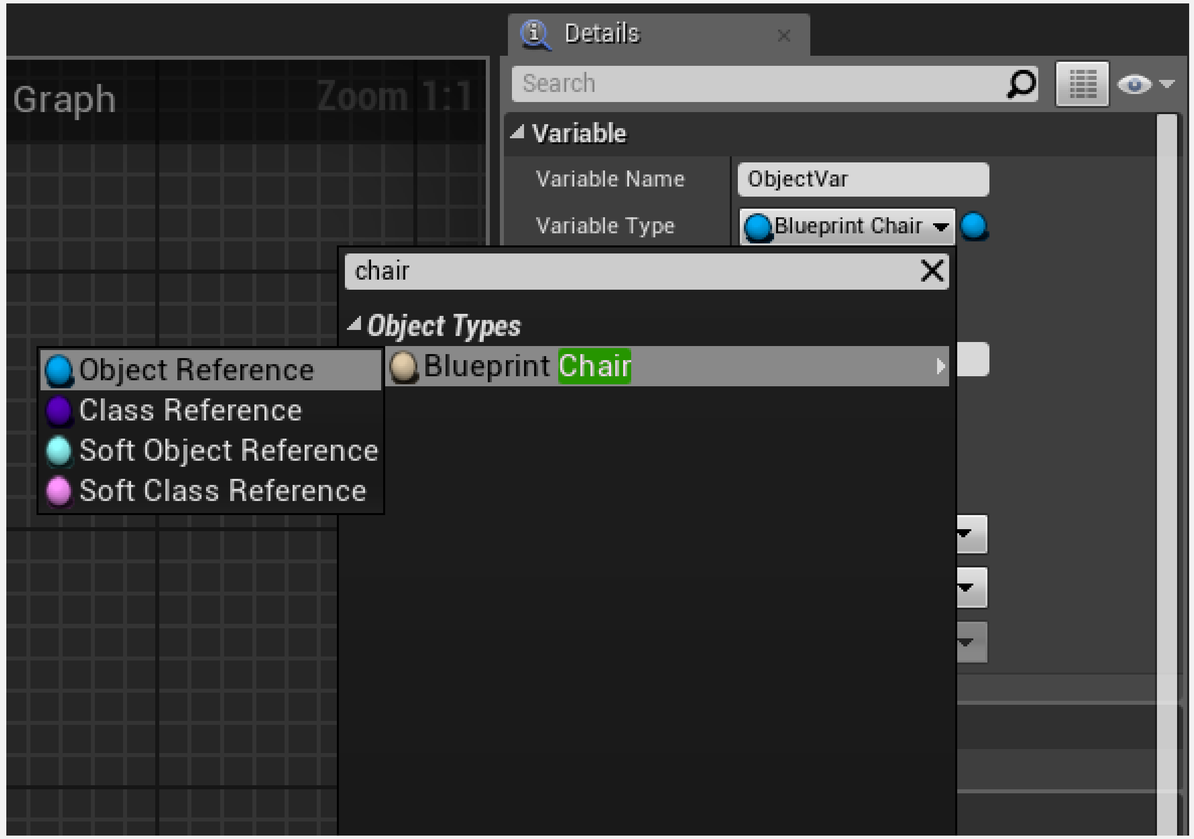
REFERENCING ACTORS

In a Blueprint, it is possible to create a variable that references an Object / Actor.

The example on the right shows the creation of a variable that references an instance of the **Blueprint_Chair** class. An **Object Reference** points to an Actor that is in the Level.

When created, the variable is empty. A way to set an instance to this variable is to check the **Instance Editable** property, add the Blueprint to the Level, and in the Details panel select an Actor that is in the Level.

Another way is to use the return value of the **Spawn Actor from Class** function.



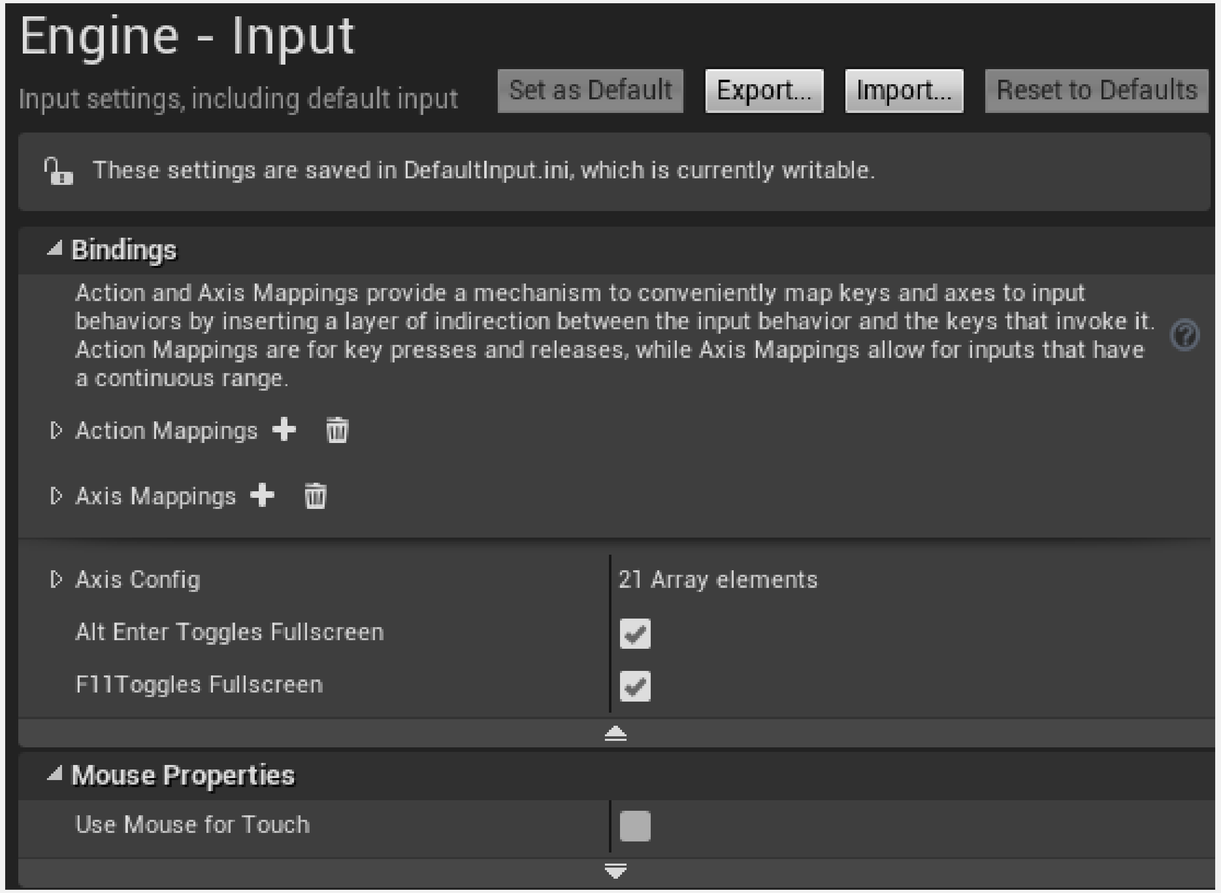
PLAYER INPUT

INPUT MAPPINGS

It is possible to create new input events that represent actions that make sense in the game.

For example, instead of creating an input event for the left mouse button that will trigger a gun, it is better to create an action event called “**Fire**” and map all keys and buttons that can trigger this event.

To access the **input mappings**, in the **Level Editor** menu, go to **Edit > Project Settings...** and in the **Engine** category select the **Input** option.

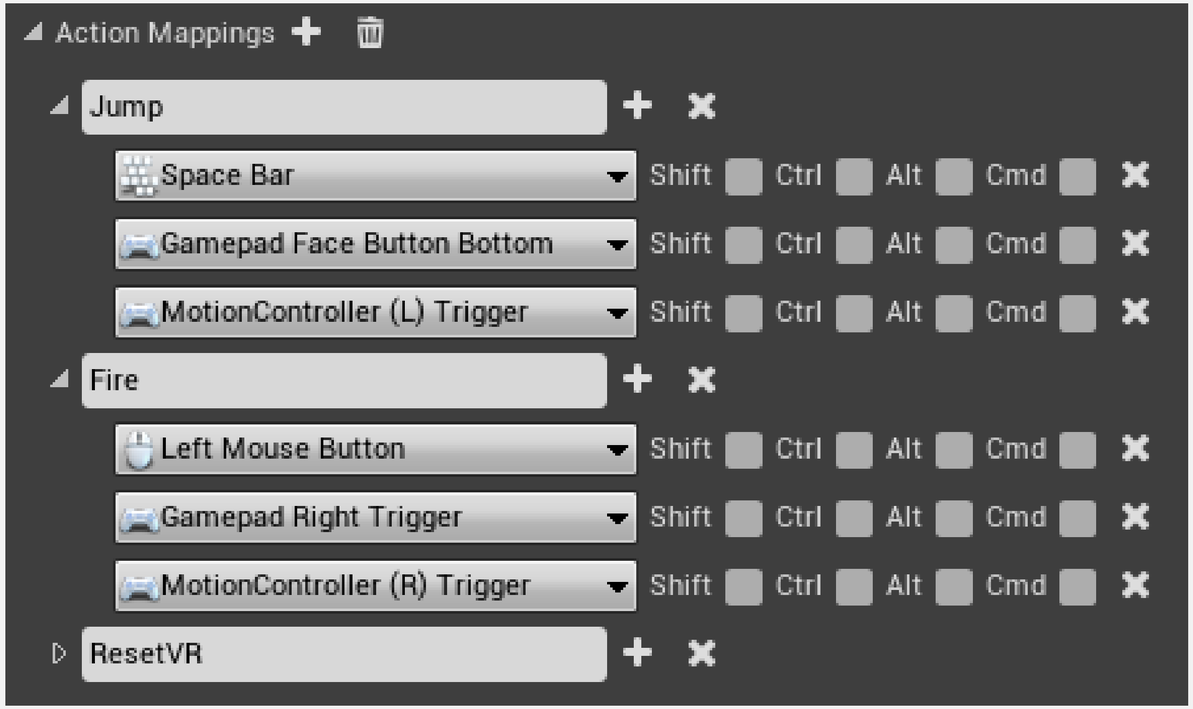


ACTION MAPPINGS

Action mappings are for key and button presses and releases.

The image on the right shows an example of action mappings from the First Person template.

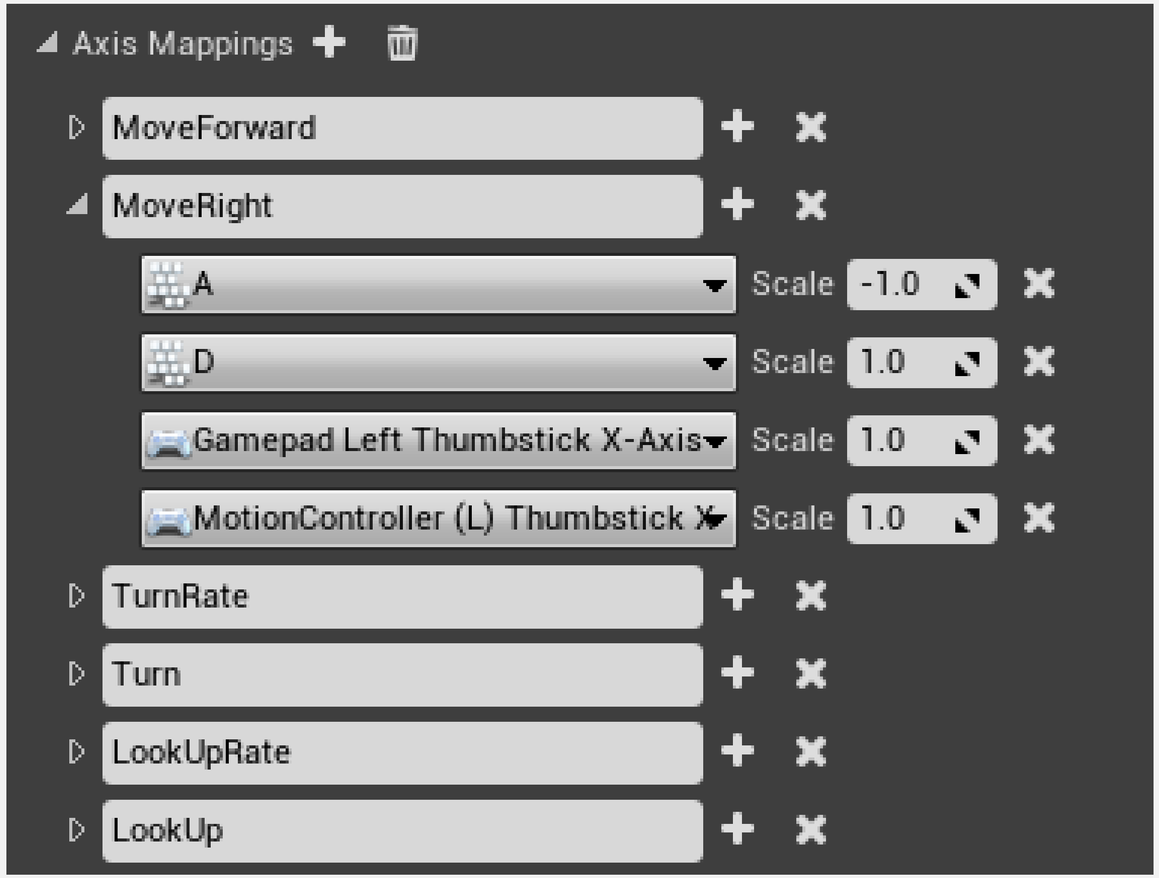
In this example, an action named “**Jump**” has been created that can be triggered by the **space bar**, the **bottom face button** of a **gamepad**, or the **left trigger** of a **motion controller**.



AXIS MAPPINGS

Axis mappings allow for inputs that have a continuous range, such as the movement of a mouse or the analog sticks of a gamepad.

Keys and buttons can also be used in the axis mapping. In the example on the right, the **MoveRight** action is mapped to the “**D**” key, with the value of the **Scale** property set to “**1.0**”, and to the “**A**” key, with the value set to “**–1.0”,** which represents the reverse direction.

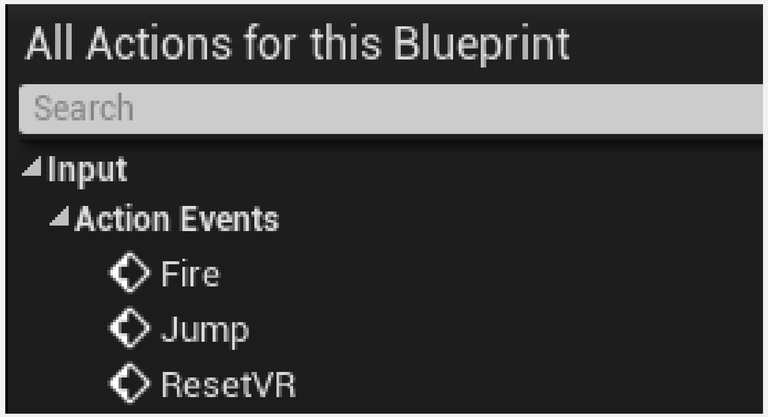


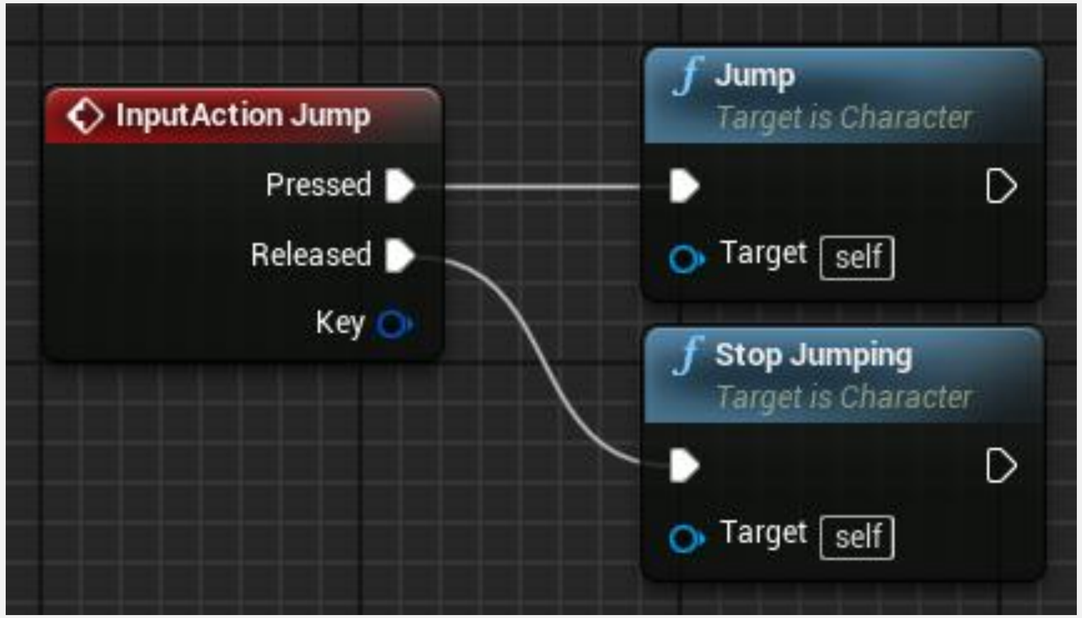
INPUT ACTION EVENTS

All **action mappings** are available in the Blueprint Editor under **Input > Action Events** in the context menu.

An **InputAction** event is generated when the keys or buttons associated with it are pressed or released.

The bottom image on the right shows an example of an InputAction event.



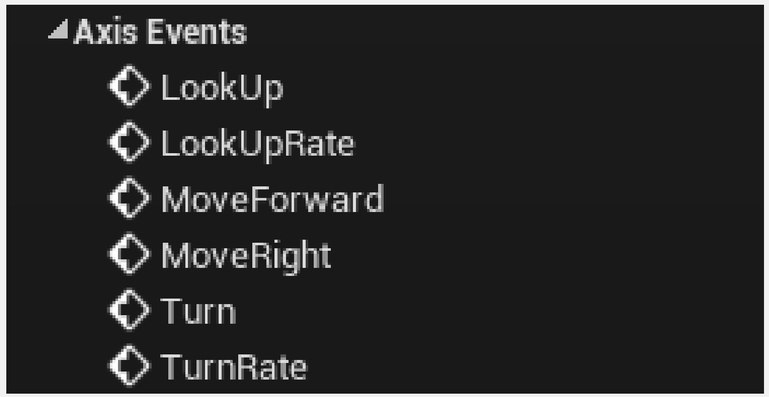


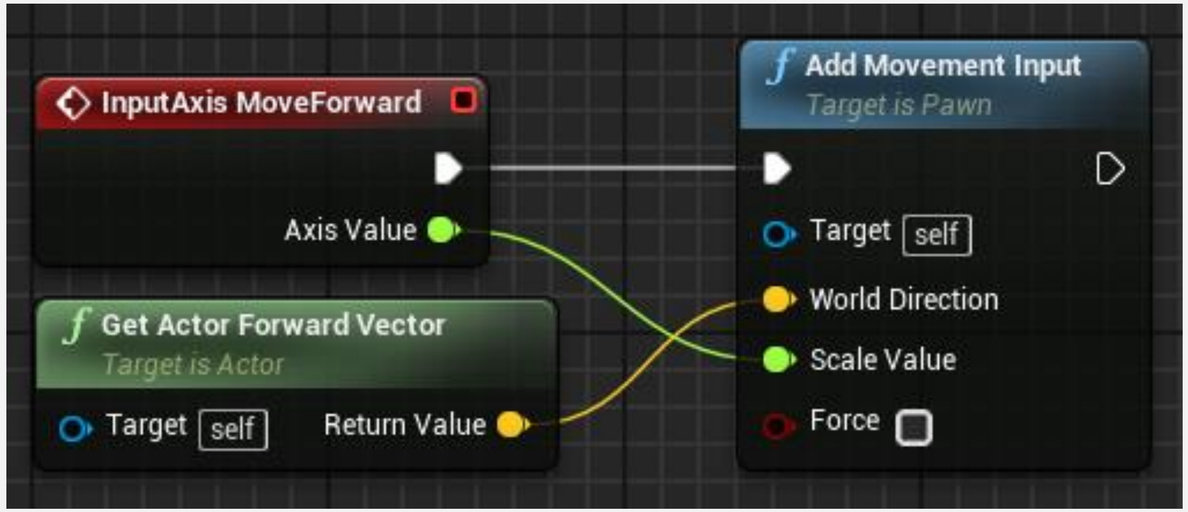
INPUT AXIS EVENTS

All **axis mappings** are available in the Blueprint Editor under **Input > Axis Events** in the context menu.

An **InputAxis** event continuously reports the current value of the axis.

The bottom image on the right shows an example of an InputAxis event.





SUMMARY

This lecture presented components and the Construction Script and showed how to add various types of events to a Blueprint.

It also explained how to manage Actor instances and how to do input mappings.