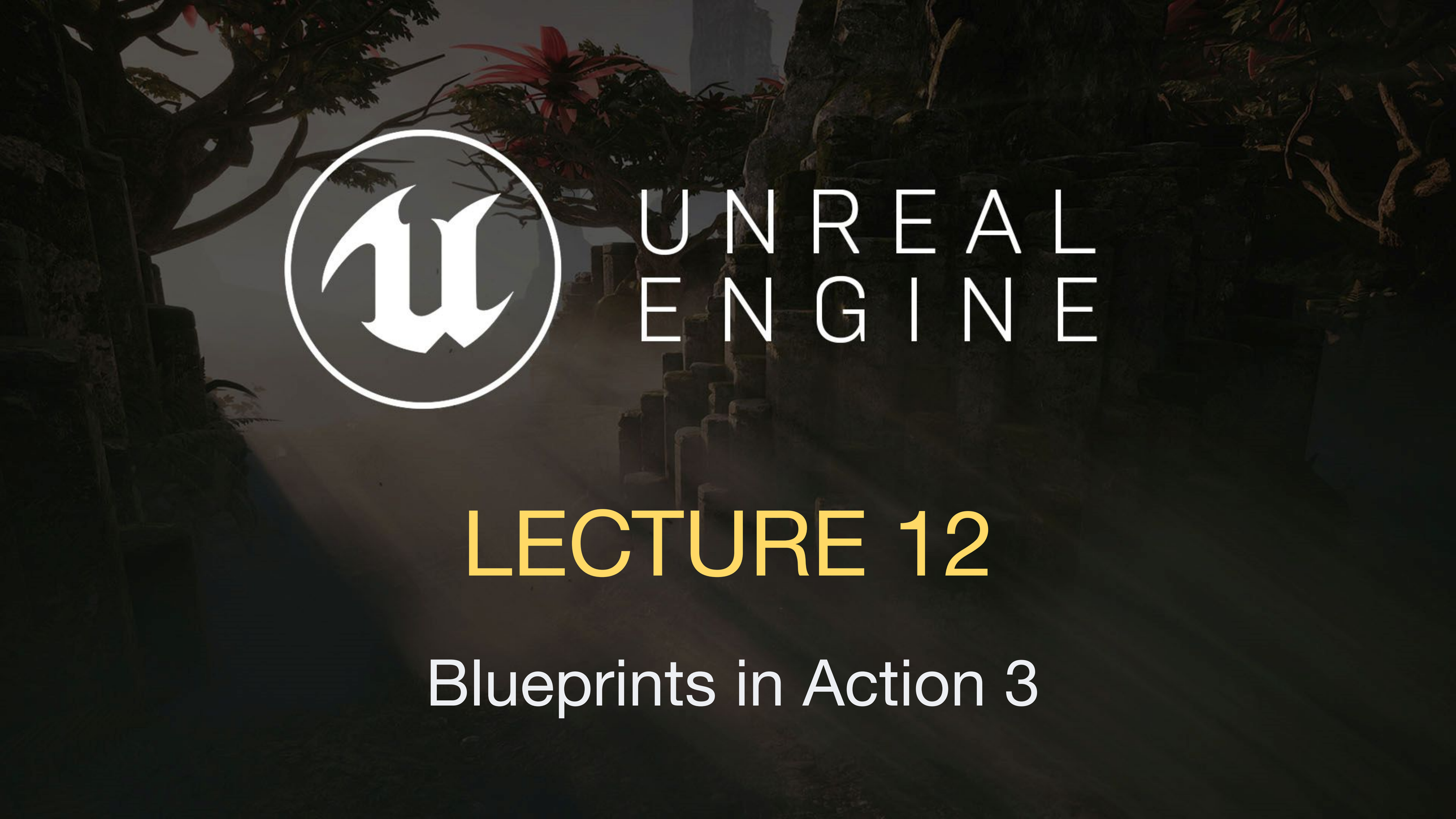


LECTURE 12

Blueprints in Action 3

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Present a variety of Blueprint functions
- Explain the Set Input Mode Game And UI function
- Show how to create a random point in a bounding box
- Explain the Select node

Outcomes

By the end of this lecture you will be able to

- Use the modulo (%) operation and the Clamp function
- Change a Material using a Blueprint function
- Change the input mode
- Use the Select node with various index types

MODULO (%)

The **modulo** operation, represented by the percentage symbol, returns the remainder of a division.

Input

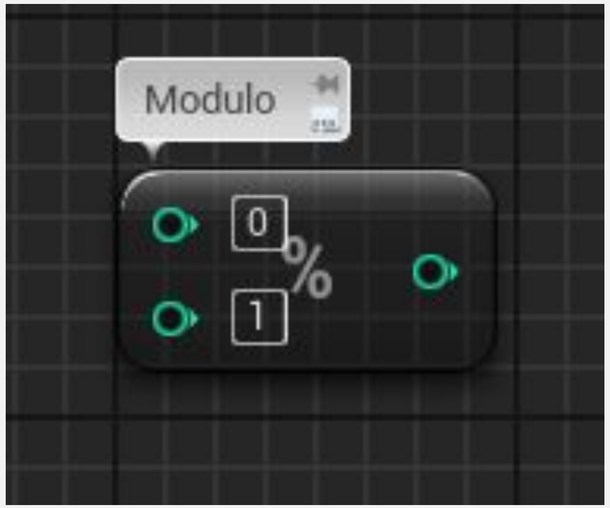
- **A**: Value to be divided.
- **B**: Value that will be used as the divisor.

Output

- **Return Value**: Remainder of the division.

Example

1 % 3 = 1
2 % 3 = 2
3 % 3 = 0
4 % 3 = 1
5 % 3 = 2
6 % 3 = 0

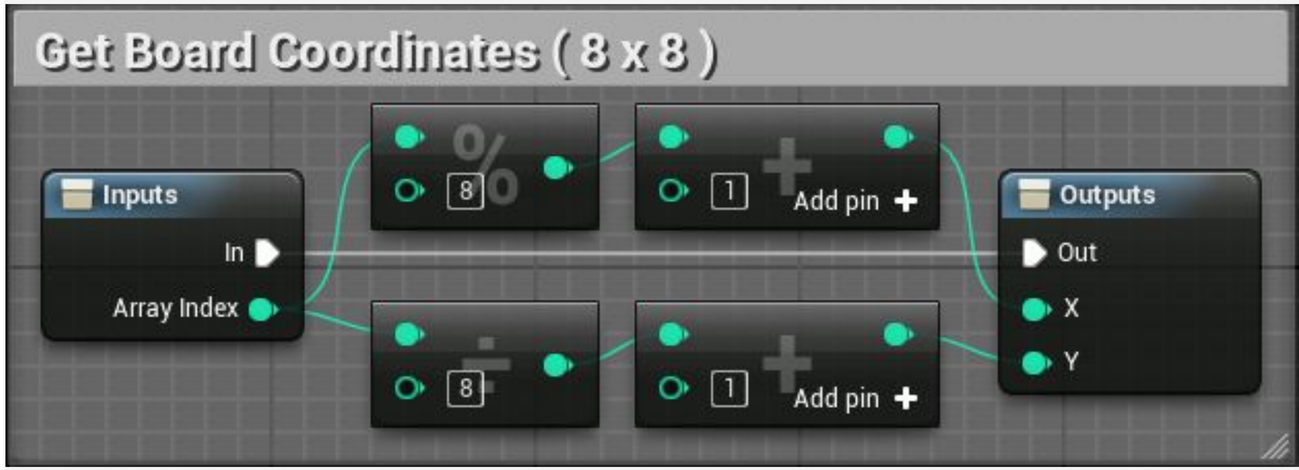


MODULO (%): EXAMPLE

It is common to use the modulo operation to convert the index of a one-dimensional array into a two-dimensional coordinate.

In the example on the right, an array of integers is being used to represent a chessboard. The array contains 64 positions, so its indexes range from “0” to “63”. Each position of the board is represented by a coordinate (X, Y).

The first position has coordinates (1, 1), and the last position has coordinates (8, 8). The macro seen in the example converts an array index to the X and Y coordinates of the board.



CLAMP

A **Clamp** function receives an input value and the minimum and maximum values. If the input value is between the minimum and maximum values, it is returned without modification.

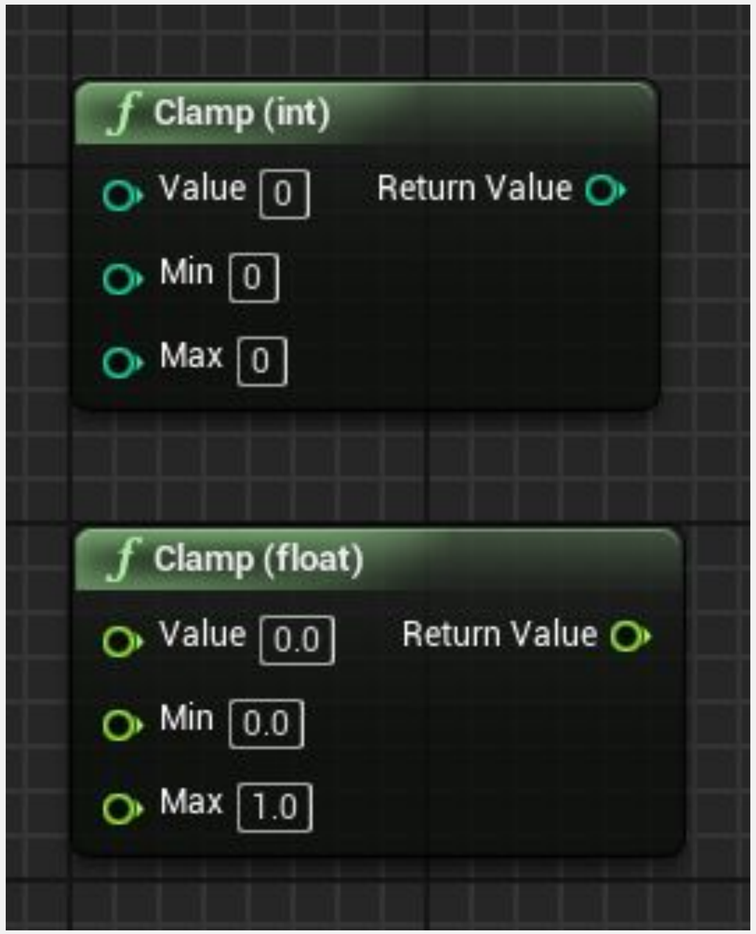
If it is below the minimum, the minimum value is returned. If it is above the maximum, the maximum value is returned.

Input

- **Value**: Input value that will be compared with the **Min** and **Max** values.
- **Min**: The lowest value that can be returned.
- **Max**: The highest value that can be returned.

Output

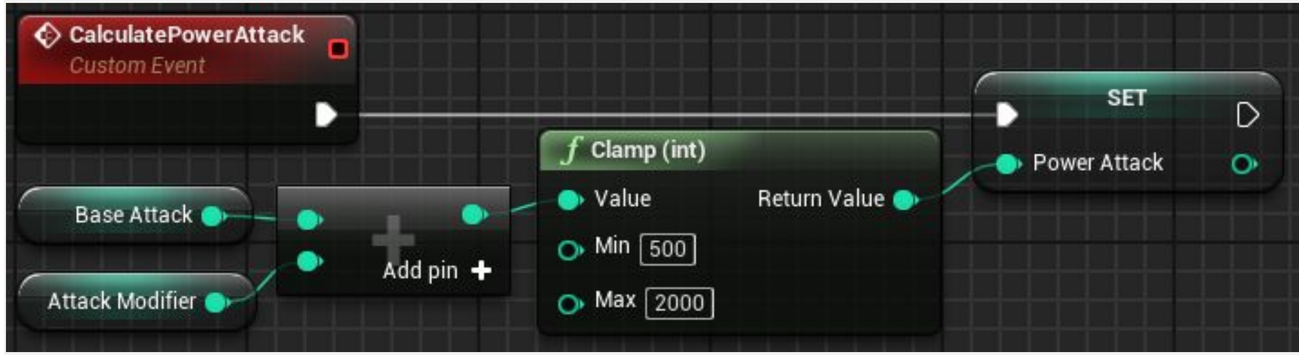
- **Return Value**: Value between the **Min** and **Max** values.



CLAMP: EXAMPLE

In the example on the right, there is an event that calculates the value of the power attack of a character. This value is based on the sum of the values of the variables **Base Attack** and **Attack Modifier**. The **Attack Modifier** variable can indicate attack bonuses, or it can have a negative value, indicating some bad state of the character.

But no matter how bad the character's attack penalties are, the minimum value of the power attack is "**500**". Similarly, the maximum value of the power attack is "**2000**", even if the sum of the values of the base attack and the attack bonus is higher.



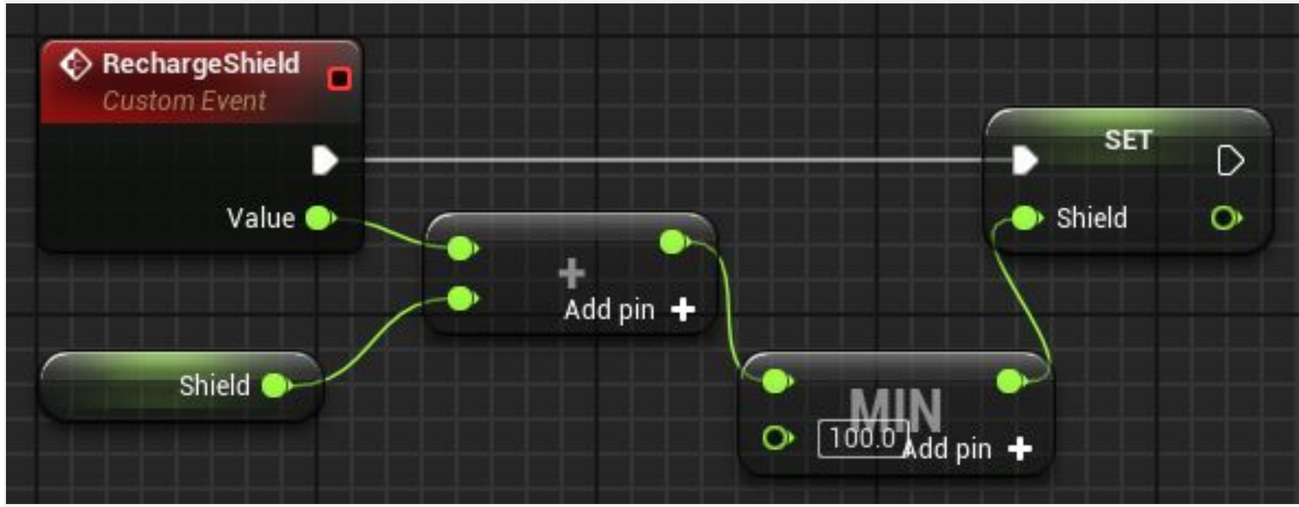
MIN AND MAX

The **Min** function returns the minimum value between the input values. The **Max** function returns the maximum value between the input values.

Many input values can be added in both functions by using the **Add pin** button.

Example

In the bottom image on the right, the **Min** function is being used to ensure that the value of the **Shield** variable is never greater than “**100.0**”.

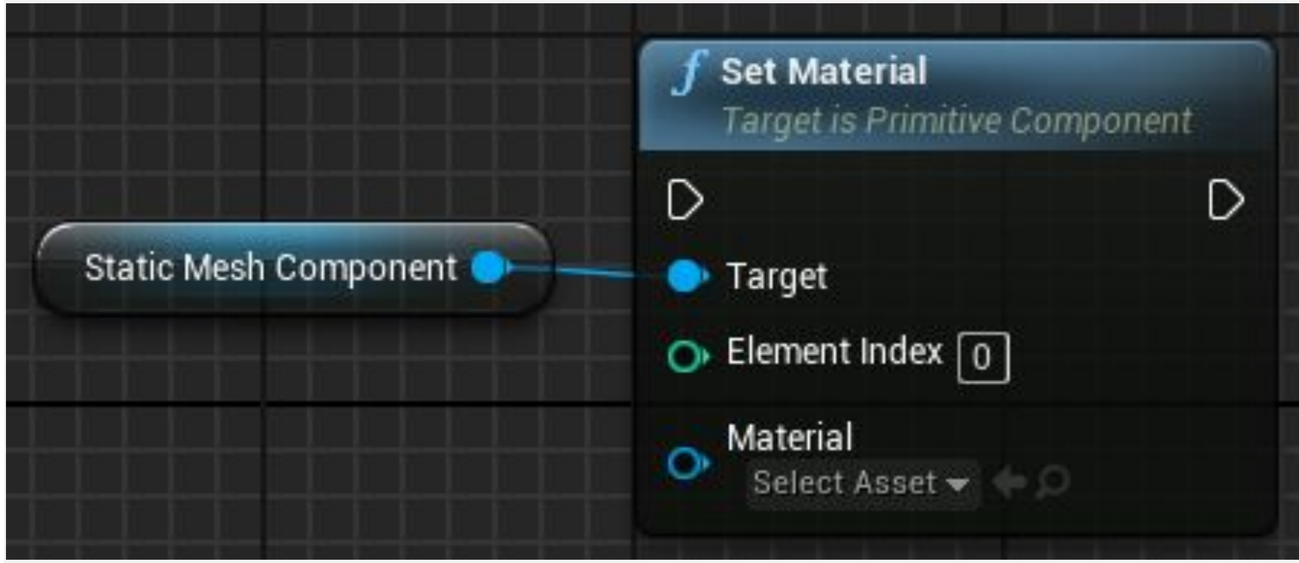


SET MATERIAL

The **Set Material** function changes the Material being used by a component.

Input

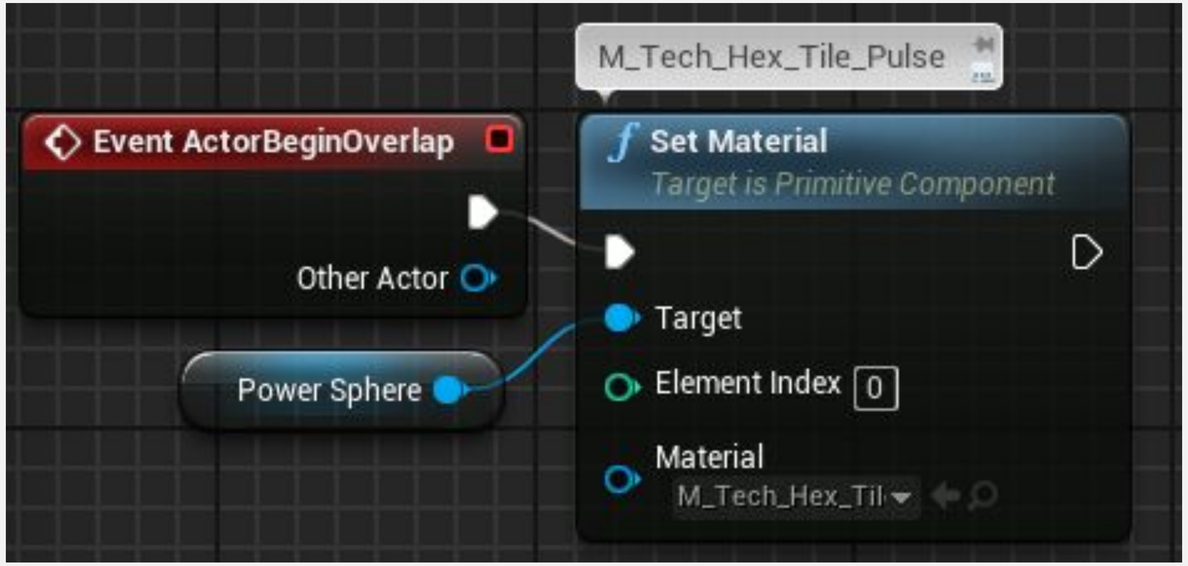
- **Target**: Primitive component that will have its Material modified.
- **Element Index**: Indicates which Material will be modified in the component.
- **Material**: Reference to the new Material that will be used.

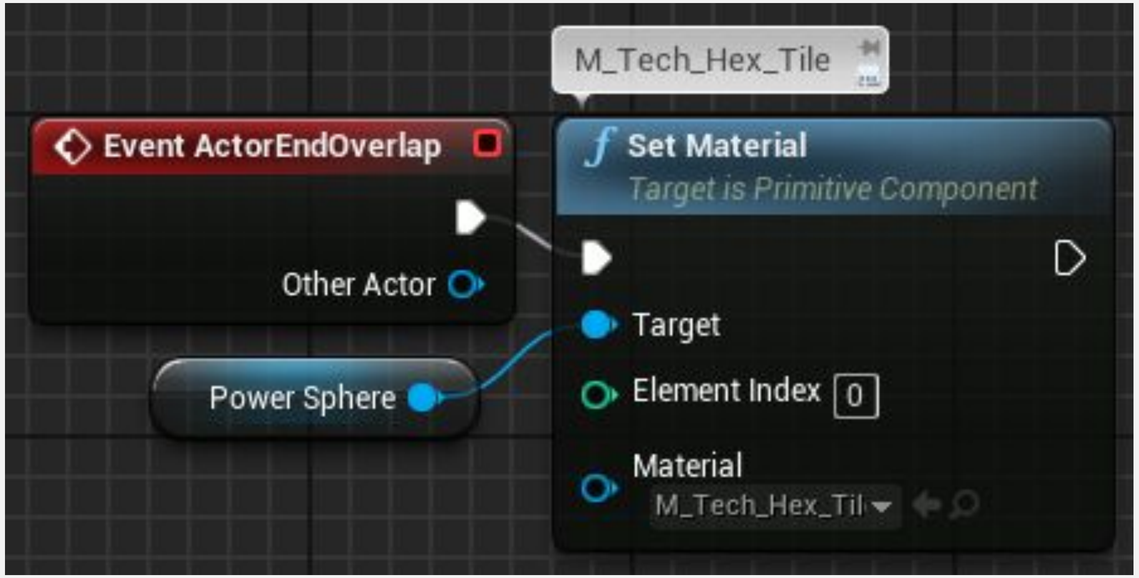


SET MATERIAL: EXAMPLE

In the example on the right, there is a Blueprint that contains a Static Mesh component called “**Power Sphere**”. When an Actor approaches the Blueprint, the **Power Sphere** Material is replaced with **T_Tech_Hex_Tile_Pulse** to indicate that it is active.

When the Actor moves away, the Material is replaced with **M_Tech_Hex_Tile**, indicating that the component is disabled.



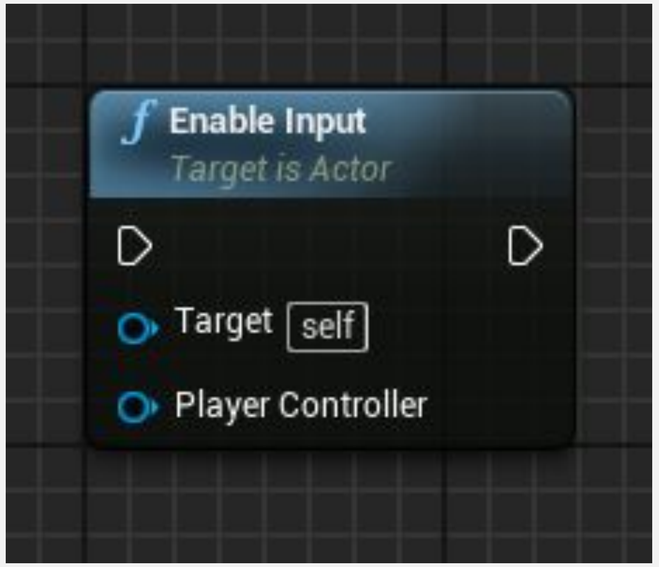


ENABLE INPUT

The **Enable Input** function is used to allow a Blueprint to respond to input events such as a keyboard, mouse, and gamepad.

Input

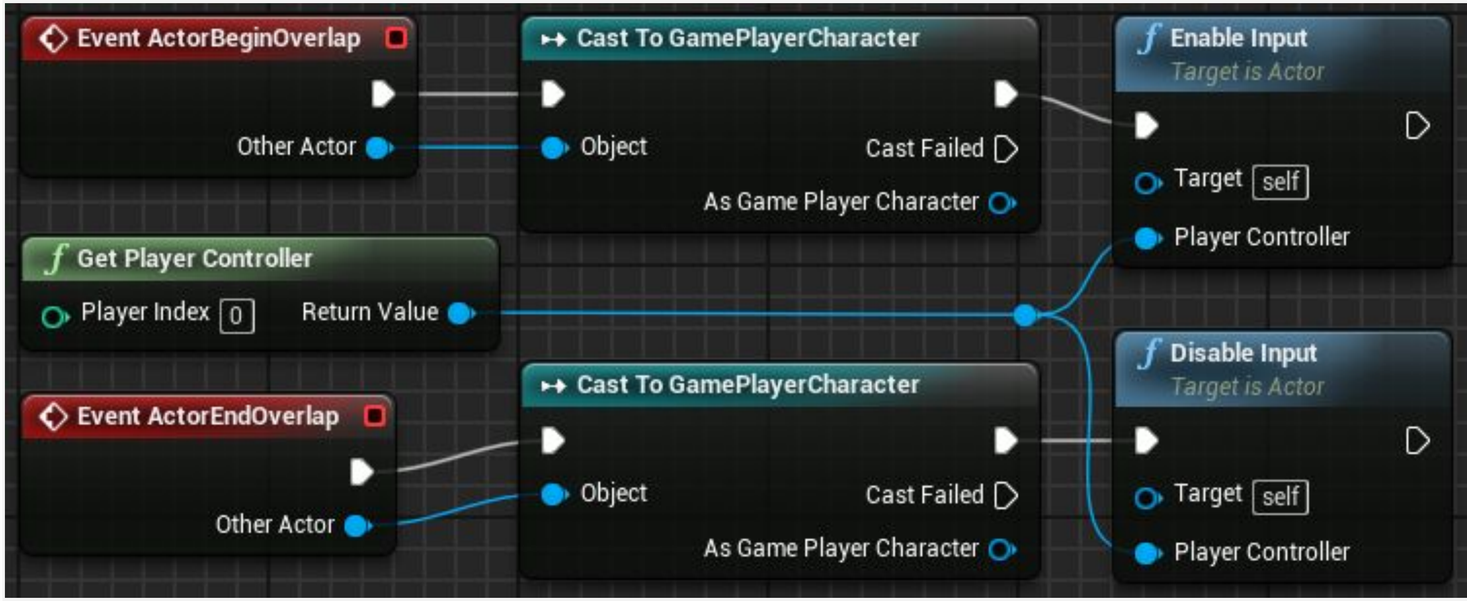
- **Target**: Actor that will have its input enabled. The “**self**” value is used to target the Blueprint that contains the node.
- **Player Controller**: A reference to the Player Controller in use.



ENABLE INPUT: EXAMPLE

In the example on the right, a Blueprint only responds to input events when the player is near it. The player is represented by a Blueprint named "**GamePlayerCharacter**". When the player is within the Blueprint's collision bounds, the **Enable Input** function is called.

When the player is out of the Blueprint's collision bounds, the **Disable Input** function is called so that this Blueprint no longer responds to input events.

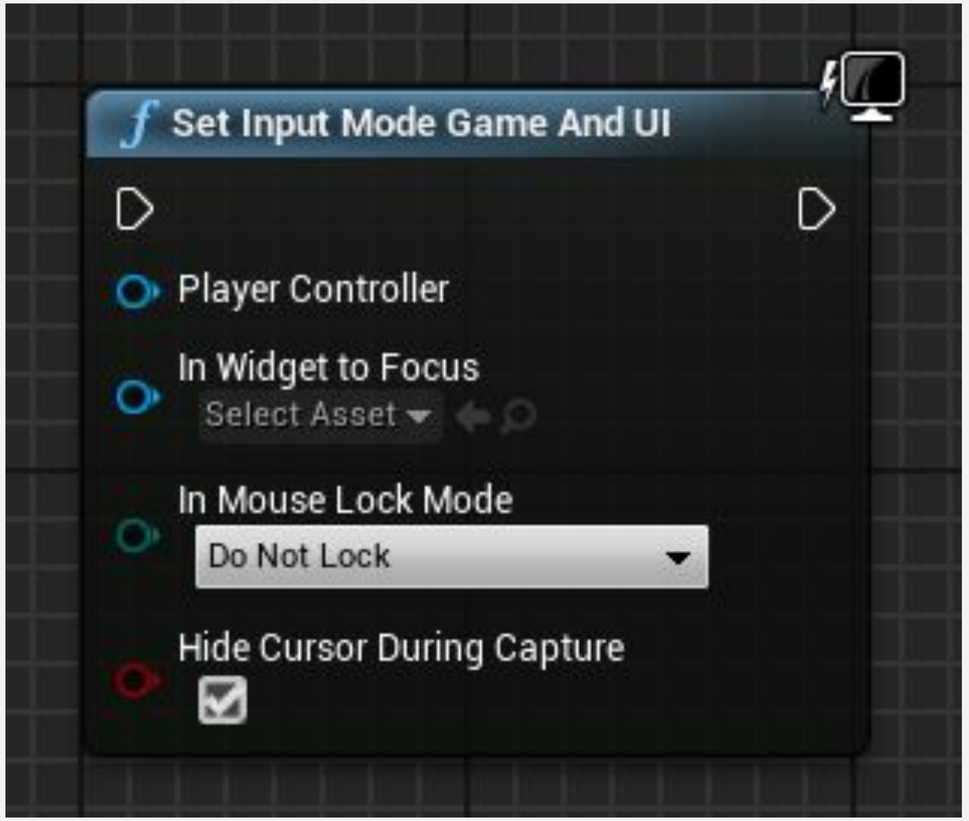


SET INPUT MODE GAME AND UI

The **Set Input Mode Game And UI** function defines an input mode in which the user interface (UI) has priority in handling user input events. If the UI does not know how to handle a particular event, the input is passed on to the Player Controller to be handled.

Input

- **Player Controller**: A reference to the Player Controller.
- **In Widget to Focus**: A reference to a UMG widget that will receive the player's input first before passing it to the Player Controller.
- **In Mouse Lock Mode**: Indicates how the mouse will be handled.
- **Hide Cursor During Capture**: Indicates whether the mouse cursor should be hidden.

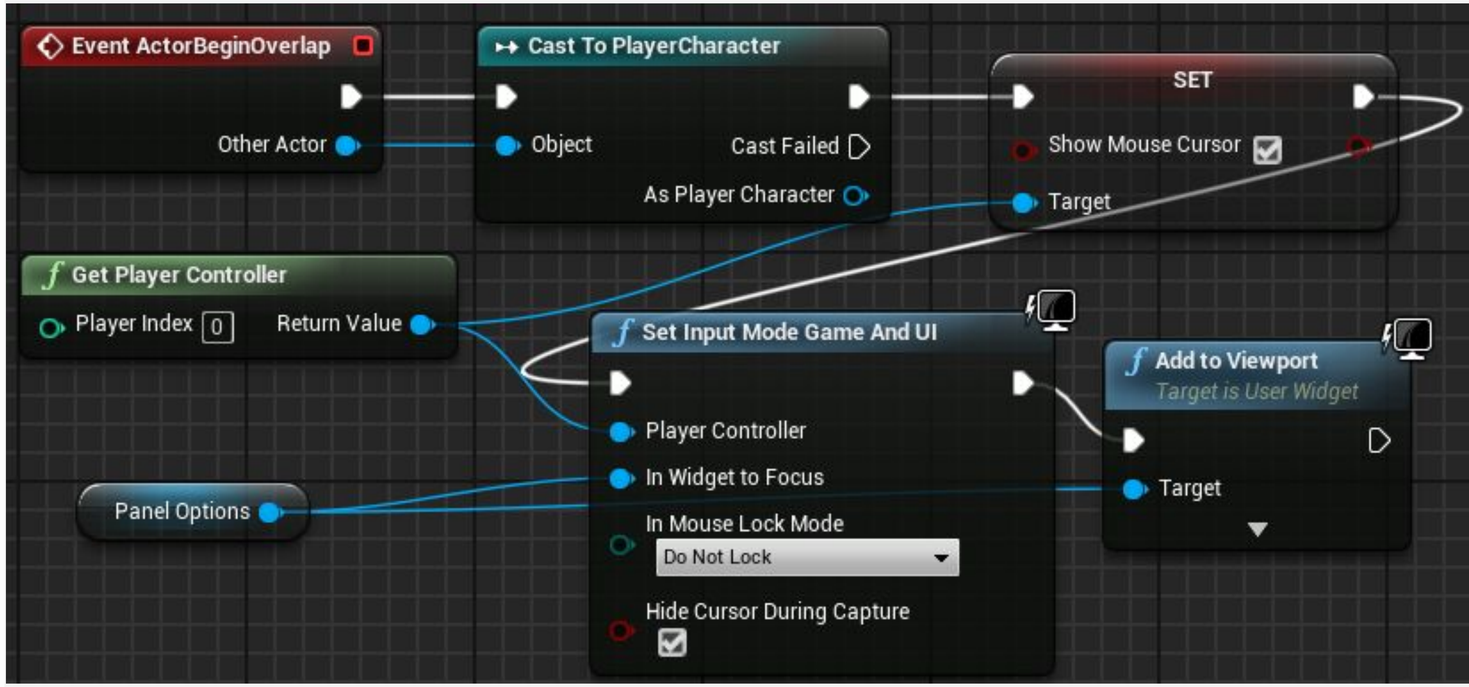


SET INPUT MODE GAME AND UI : EXAMPLE

In the example on the right, there is a Blueprint that interacts with the player in a first-person game. When the player overlaps this Blueprint, a UI is displayed with options for the player to choose from using the mouse.

The **Show Mouse Cursor** property of the Player Controller is set to “**true**” to display the cursor on-screen.

The **Set Input Mode Game And UI** function causes the mouse to control the cursor so that it can interact with the panel instead of changing the player’s view.



TELEPORT

The **Teleport** function moves an Actor to the specified location. If there is an obstacle at the location, the Actor is moved to a nearby place where there is no collision.

Input

- **Dest Location**: Destination location.
- **Dest Rotation**: Rotation applied to the Actor.

Output

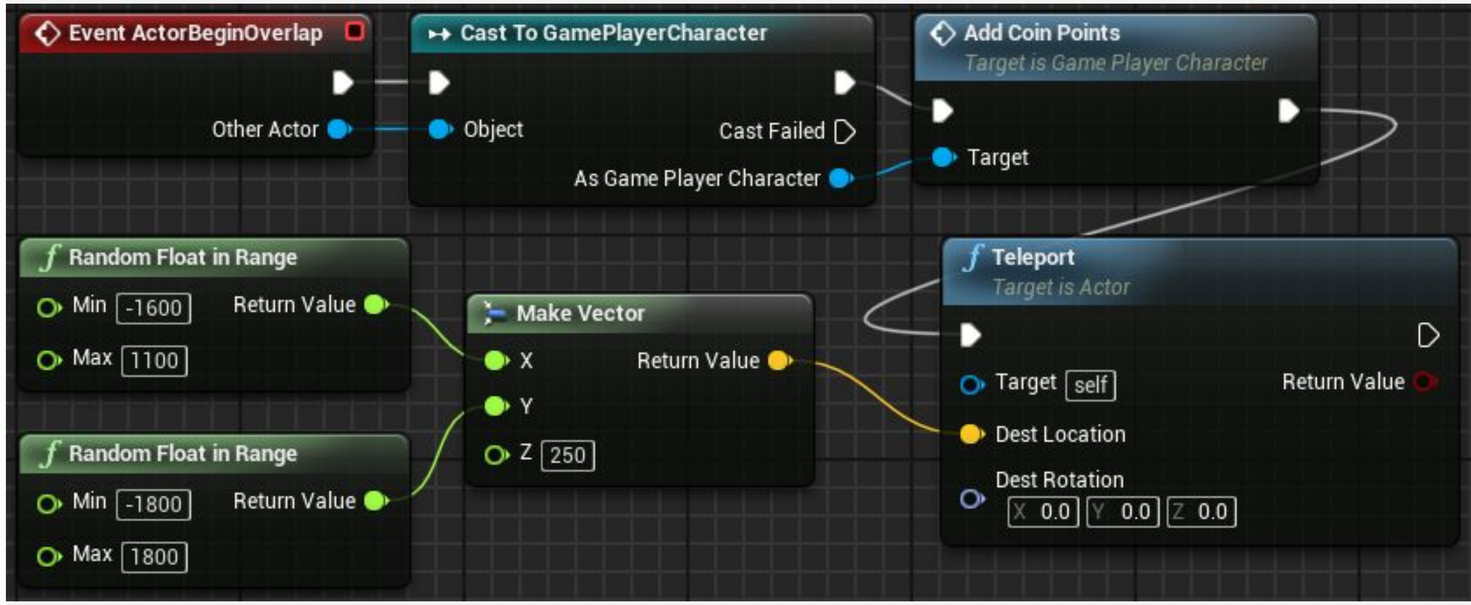
- **Return Value**: Boolean variable. If the value is “**false**”, the Actor could not be moved.

TELEPORT: EXAMPLE

In the example on the right, there is a game in which the player collects coins. The image shows the **ActorBeginOverlap** event of the Blueprint that represents the coins.

When the player collects a coin, the **Add Coin Points** function is called to register the points, and after that the coin is teleported to another location in the playing area.

The playing area has several obstacles, but the **Teleport** function will place the coin in a free place.



RANDOM POINT IN BOUNDING BOX

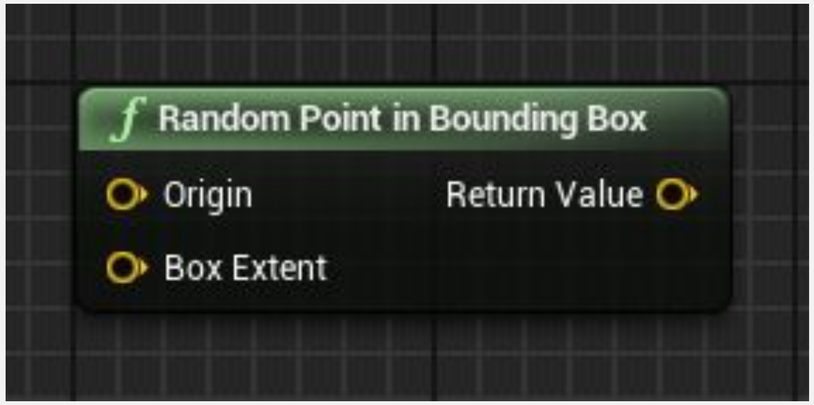
The **Random Point in Bounding Box** function returns a vector representing a random point located within a volume specified by an origin point and a box extent.

Input

- **Origin**: Vector representing the central position of the box extent.
- **Box Extent**: Vector that contains the dimensions of the box that defines the 3D volume.

Output

- **Return Value**: Vector representing a point that was generated randomly inside the defined box extent.

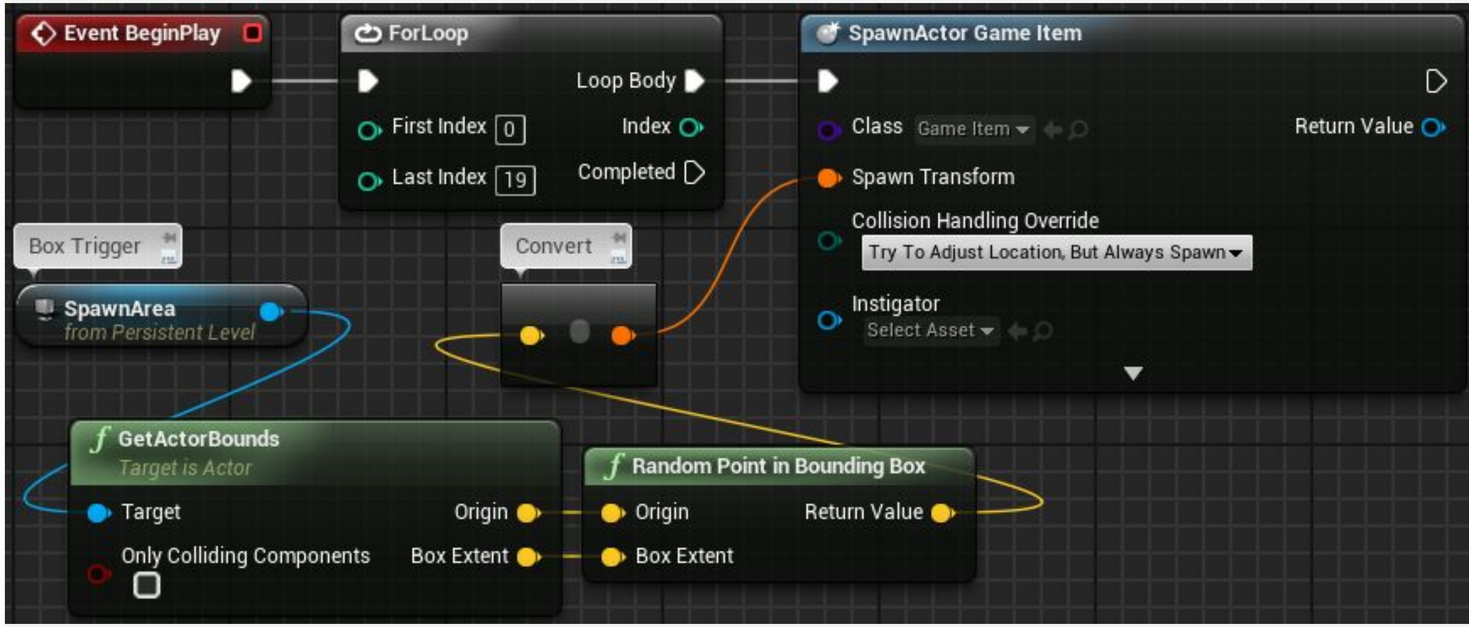


RANDOM POINT IN BOUNDING BOX: EXAMPLE

The image on the right is from a Level Blueprint. A Box Trigger named “**SpawnArea**” has been placed in the Level to define an area in which items will be created at random positions.

When the game starts, the **ForLoop** node executes the **SpawnActor** node 20 times to create 20 items in random positions.

The position of each item is determined by the **Random Point in Bounding Box** function using the origin point and box extent specified for the spawn area.



SELECT

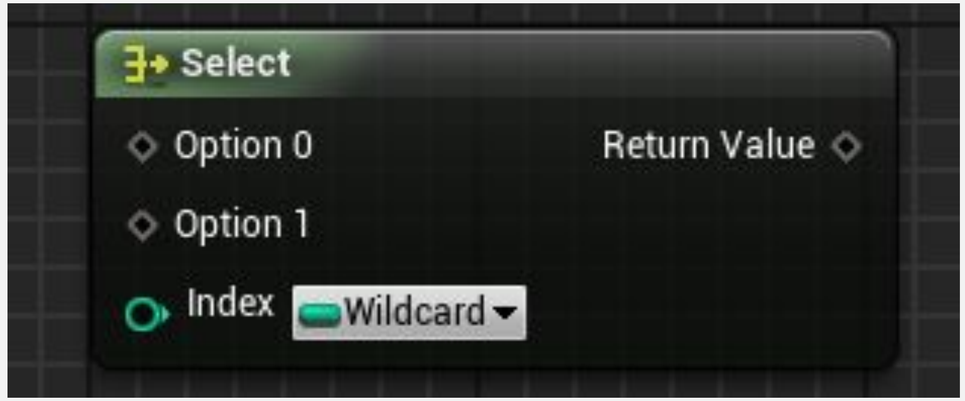
The **Select** node returns a value associated with the options that corresponds to the index that is passed as input. The **Select** node is generic and can work with several types of variables for the index and for the values of the options. To change the type of the options or the index, right-click on the parameter and choose “**Change Pin Type**”. You can also set the pin type by dragging a variable reference or wire onto the pins. To add more options, right-click on the node and choose “**Add Option Pin**”.

Input

- **Option 0, 1,** …: Values that can be returned by the **Select** node. The options can be of any type.
- **Index**: Value that will determine the option to be returned. The type can be “**Boolean**”, “**Byte**”, “**Integer**”, or “**Enum**”.

Output

- **Return Value**: The option indicated by the **Index** parameter.



SELECT: EXAMPLE 1

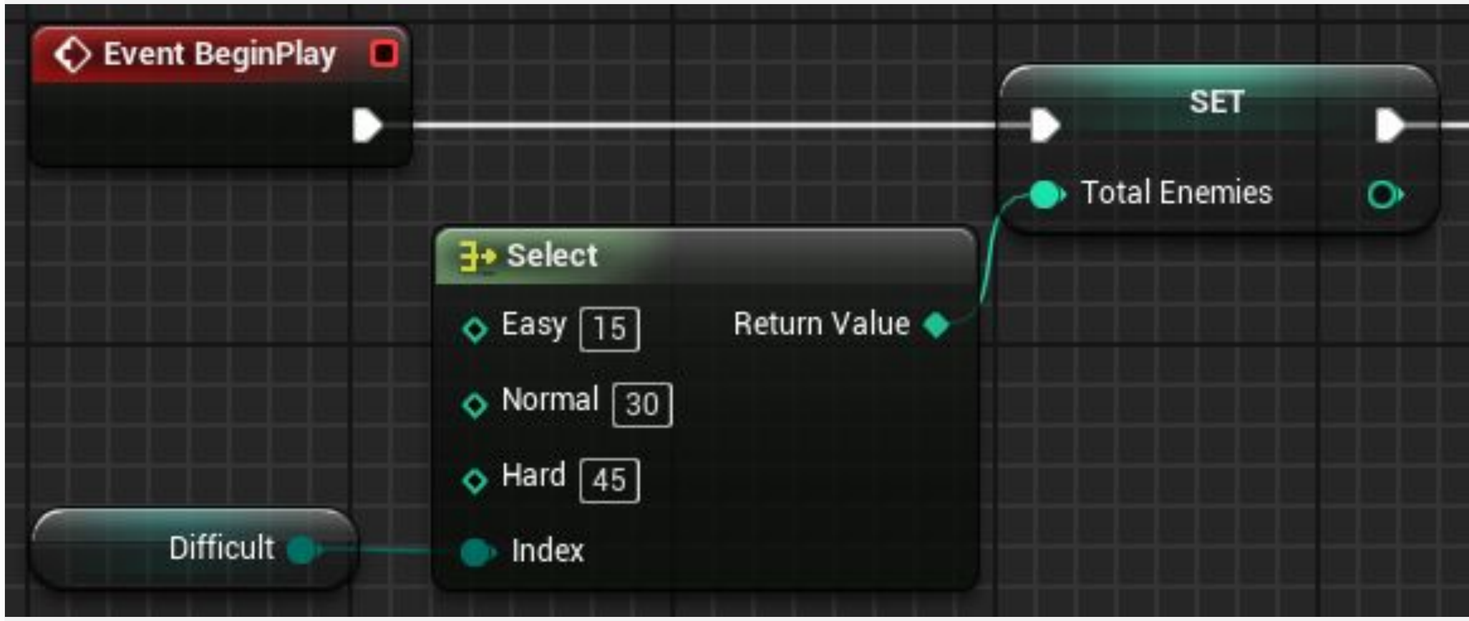
In the **Respawn** event seen in the image on the right, the **Select** node is being used to get the location of a spawn point based on the value of the integer variable **Spawn Point Index**.

If the value of **Spawn Point Index** is “**0**”, it will return “**(0,0,0)**”. If the value is “**2**”, it will return the value of the **Spawn Point 2** variable.

SELECT: EXAMPLE 2

In the example on the right, the index is an enum called “**Difficult**” that contains the **Easy**, **Normal**, and **Hard** values.

The type of the options is “**Integer**” and represents the number of enemies according to the value of the **Difficult** enum variable.



SUMMARY

This lecture presented various Blueprint functions.

It showed how to change the input mode, create random points, change a Material, and use the Select node.