LECTURE 2

Basic Blueprint Programming Concepts

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Demonstrate how to create variables and modify their properties
- Show how to use arithmetic, relational, and logical operators
- Present functions, custom events, macros, and their differences
- Introduce program flow

Outcomes

By the end of this lecture you will be able to

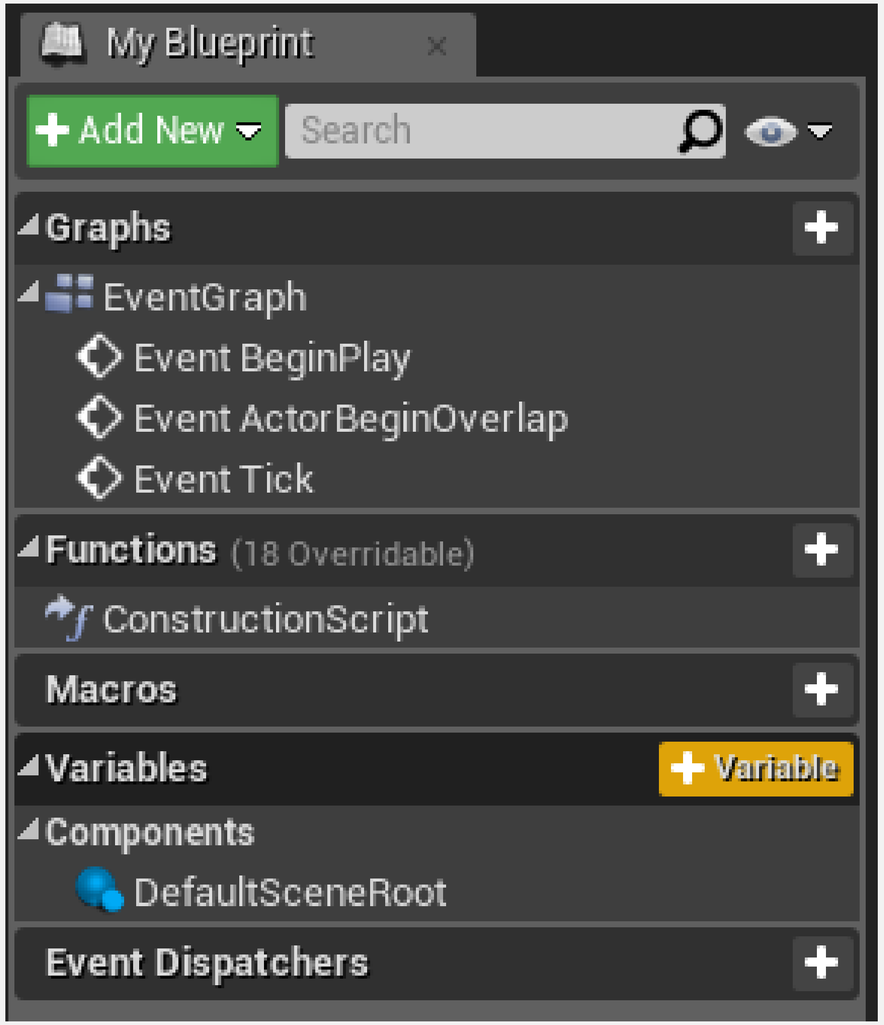
- Create variables using the correct type for each situation
- Create expressions using operators
- Understand when to use functions, custom events, or macros
- Use some basic nodes that control the program flow

VARIABLES

CREATING VARIABLES

Variables are used to store values and attributes in Blueprints that can be modified during the execution of the game. The variables can be of different types.

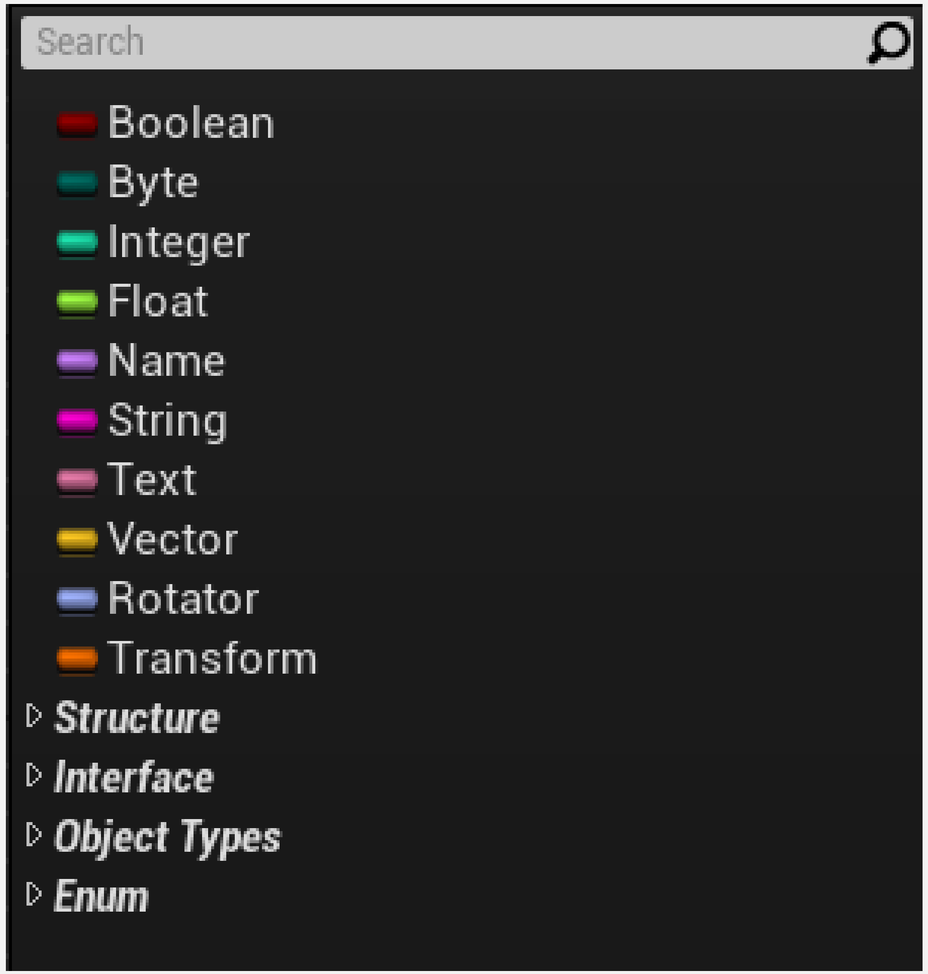
To create a variable, go to the **My Blueprint** panel in the **Blueprint Editor** and click the “**+**” button in the **Variables** category.



VARIABLE DATA TYPES

Following are common types of variables:

- **Boolean**: Can only hold the value “true” or “false”.
- **Integer**: Used to store integer values.
- **Float**: Used to store decimal values.
- **String / Text**: Used to store text. The Text variable is preferable since it supports localization.
- **Vector**: Contains the float values X, Y, and Z.
- **Transform**: Used to store location, rotation, and scale.

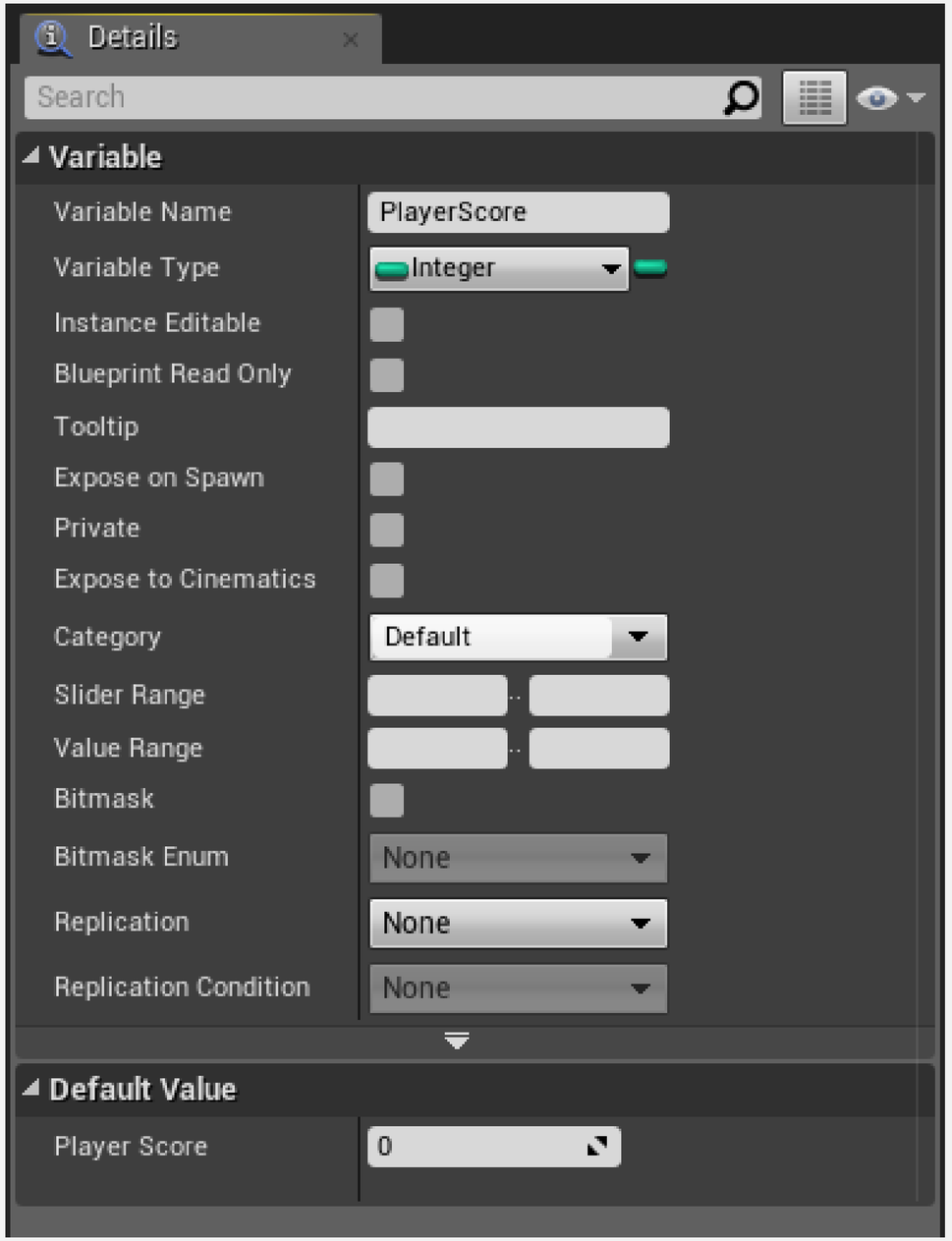


DETAILS PANEL

When a variable is selected, its properties are displayed in the **Details** panel. This is where changes can be made to the variable's name and type.

Other properties found in the Details panel include the following:

- **Instance Editable**: If checked, the variable can be changed in the instances that are in the Level.
- **Blueprint Read Only**: If checked, the variable cannot be changed by Blueprint nodes.
- **Tooltip**: Contains information shown when the cursor hovers over the variable.
- **Expose on Spawn**: If checked, the variable can be set when spawning the Blueprint.

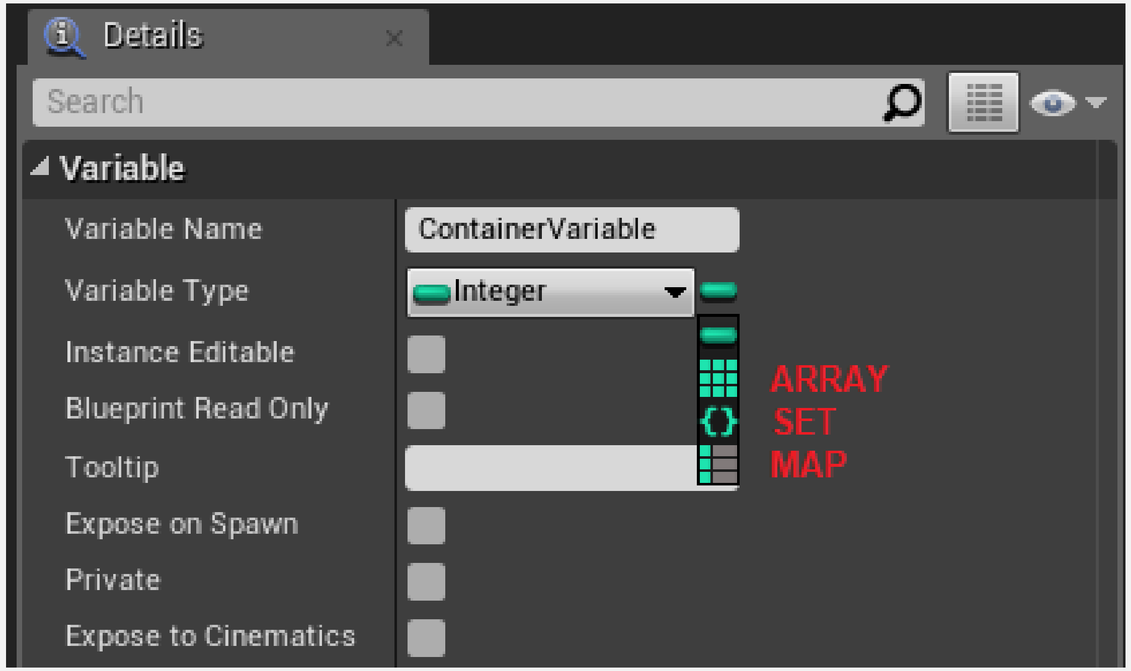


ARRAY, SET, AND MAP

The **Variable Type** property includes a button that is used to convert the variable into a **container**.

A container can store several elements of the same type. Listed below are the types of containers available.

- **Array**: An ordered list of values that are accessed using an index value.
- **Set**: An unordered collection of values. Duplicate values are not allowed.
- **Map**: A list that uses a key-value pair to define each entry. Duplicate key values are not allowed.



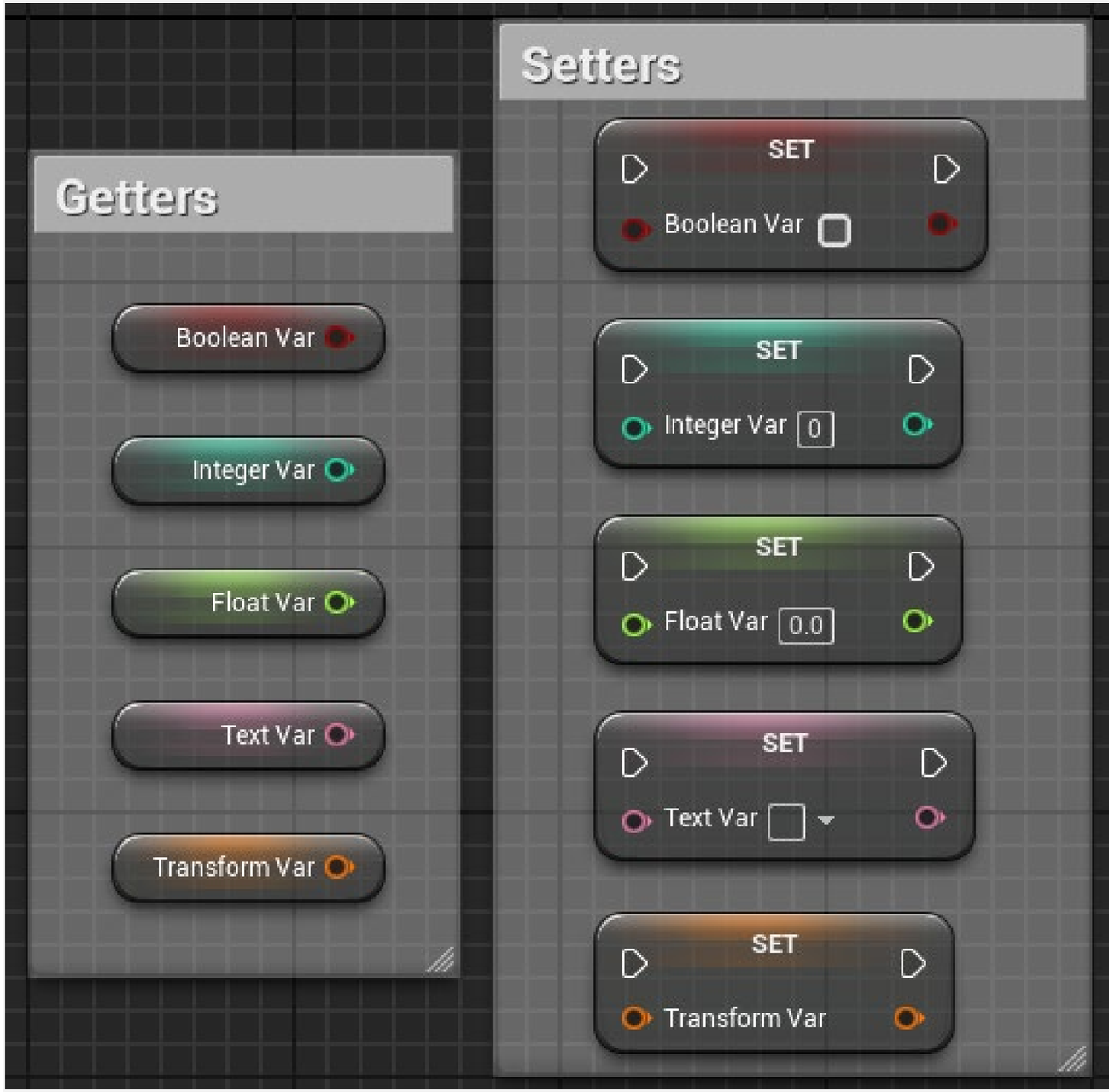
GETTERS AND SETTERS

When a variable is dragged and dropped into the Event Graph, a context menu appears with the options **Get** and **Set**.

Get nodes are used to read the value of the variable.

Set nodes are used to store a new value in the variable.

There are useful shortcuts to create **Get** and **Set** nodes. To create a **Get** node, press the **Ctrl** key when dragging and dropping a variable. The **Set** node is created using the **Alt** key.



OPERATORS

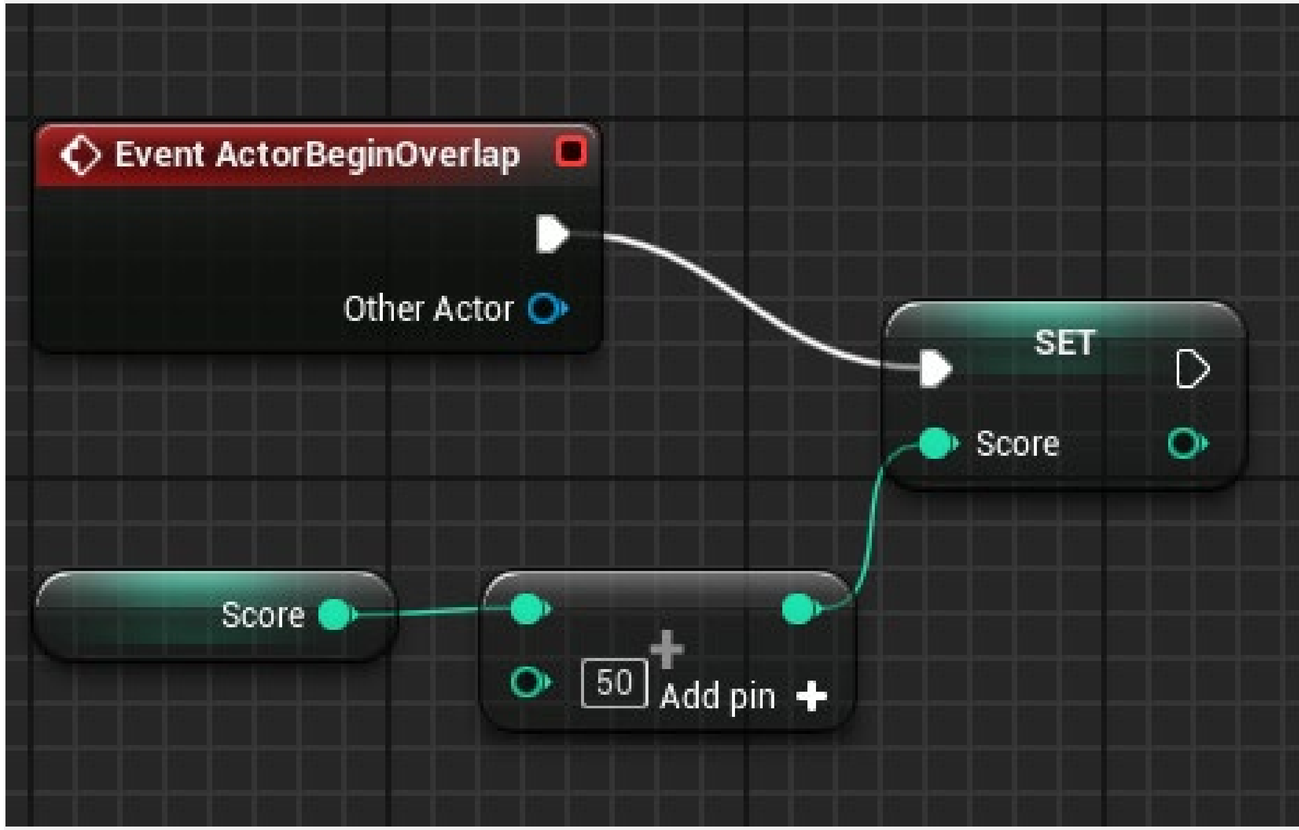
ARITHMETIC OPERATORS

The **arithmetic operators** (**+**, **-**, ***, **/**) can be used to create mathematical expressions in Blueprints.

The image on the right shows a simple expression that adds a value of “**50**” to the current **Score** variable, then sets the newly calculated value in the **Score** variable.

The “**+**” operator receives two input values on the left and gives the operation result on the right. To use more than two input values, just click on the **Add pin** option.

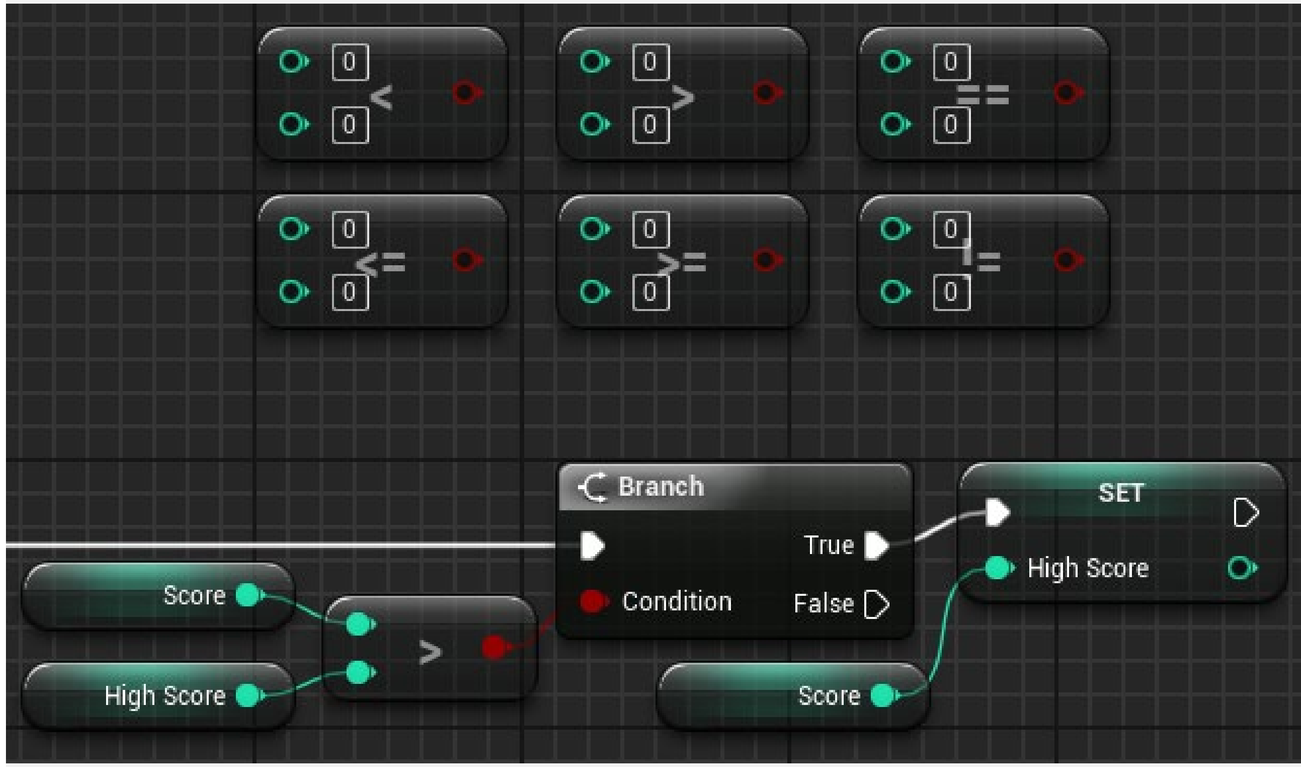
The input values can be entered directly into the nodes or can be obtained from variables.



RELATIONAL OPERATORS

Relational operators perform a comparison between two values and return a Boolean value (“true” or “false”) as a result of the comparison.

The image on the right shows the relational operators and an example using a **Branch** node. At the end of a game, the current player’s score (**Score** variable) is compared with the highest recorded game score (**High Score** variable). If the player’s score is higher, the value of the **Score** variable will be stored in the **High Score** variable.

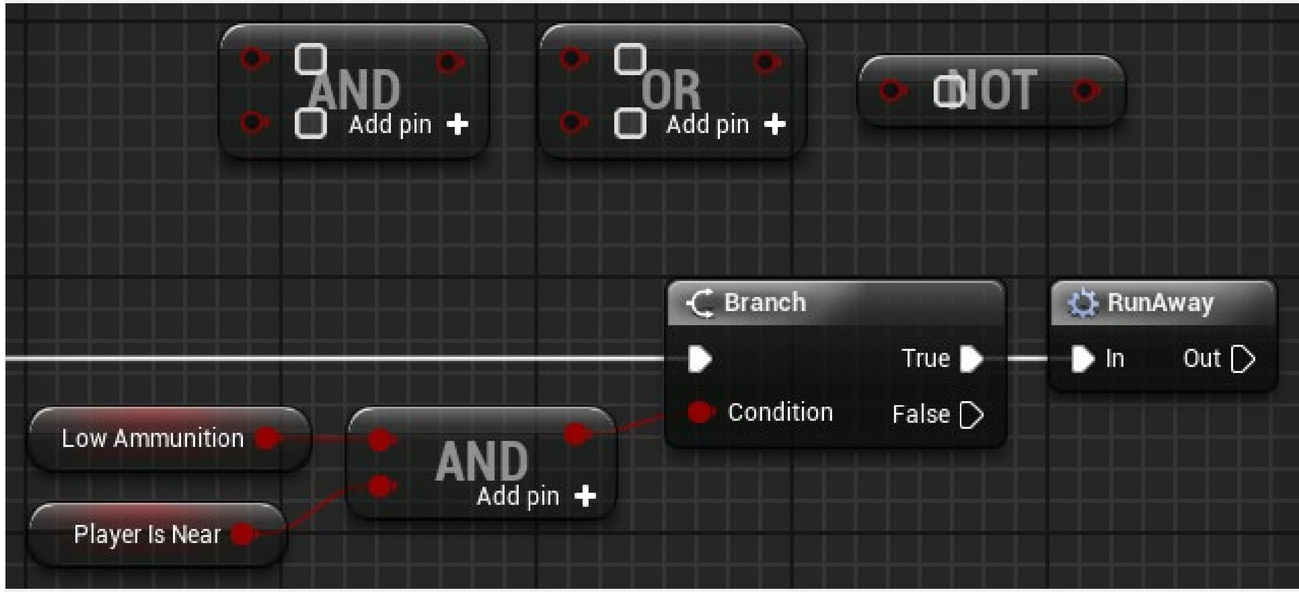


LOGICAL OPERATORS

Logical operators perform an operation between Boolean values and return a Boolean value (“true” or “false”) as a result of the operation. The main logical operators are as follows:

- **OR**: Returns a value of “true” if any of the input values are “true”.
- **AND**: Returns a value of “true” only if all input values are “true”.
- **NOT**: Receives only one input value, and the result will be the reverse value.

The example on the right simulates a simple decision of an enemy in a game. If the enemy is low on ammo (**Low Ammunition** variable) and the player is nearby (**Player Is Near** variable), then the enemy decides to run away.



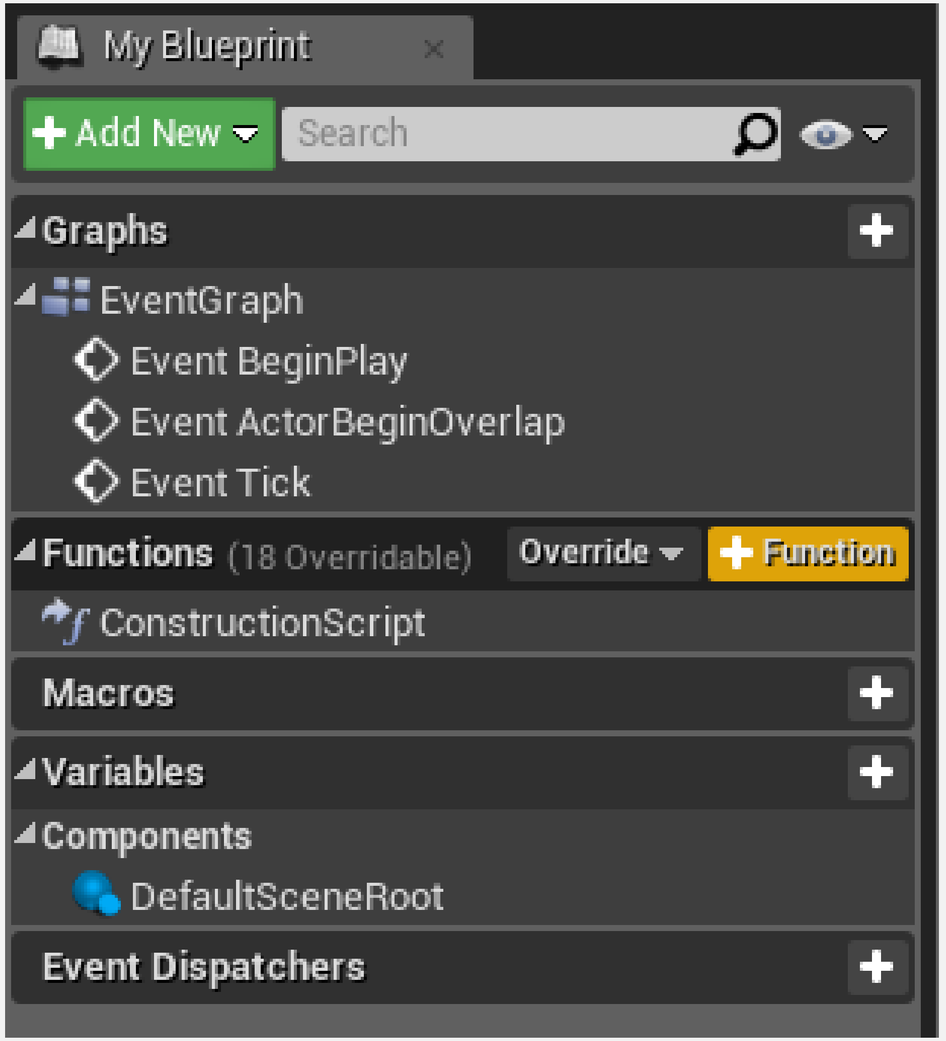
FUNCTIONS, EVENTS, AND MACROS

FUNCTIONS

Functions allow a set of actions that are executed in various parts of the Blueprint to be gathered in one place for easy organization and maintenance of the script.

Functions can be called from other Blueprints and allow the use of input and output parameters.

To create functions, go to the **My Blueprint** panel in the **Blueprint Editor** and click the “**+**” symbol in the **Functions** category.



FUNCTIONS:
INPUTS AND OUTPUTS

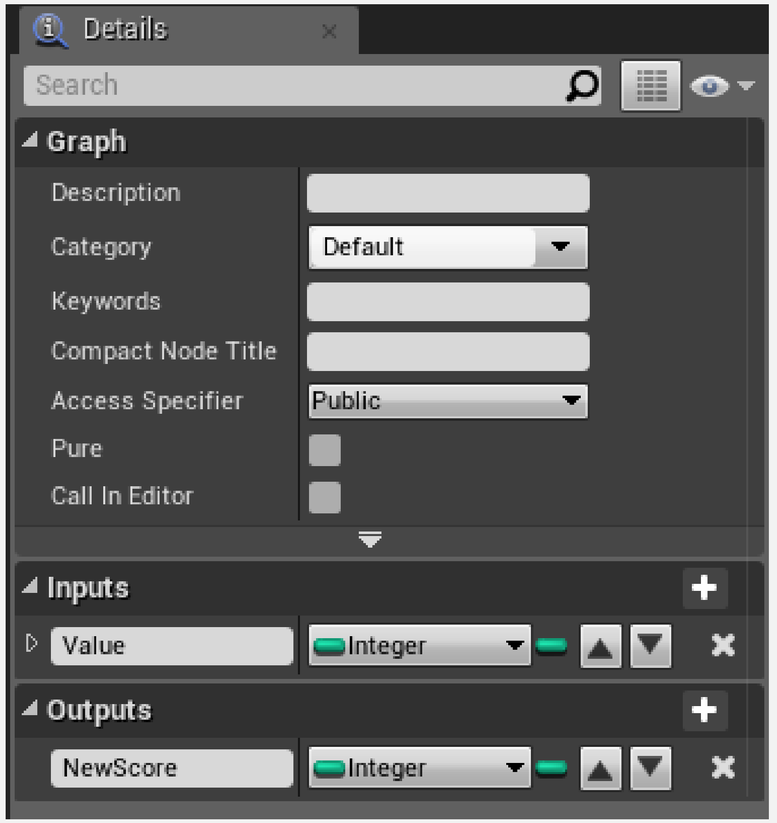
Input parameters are values that can be passed into a function.

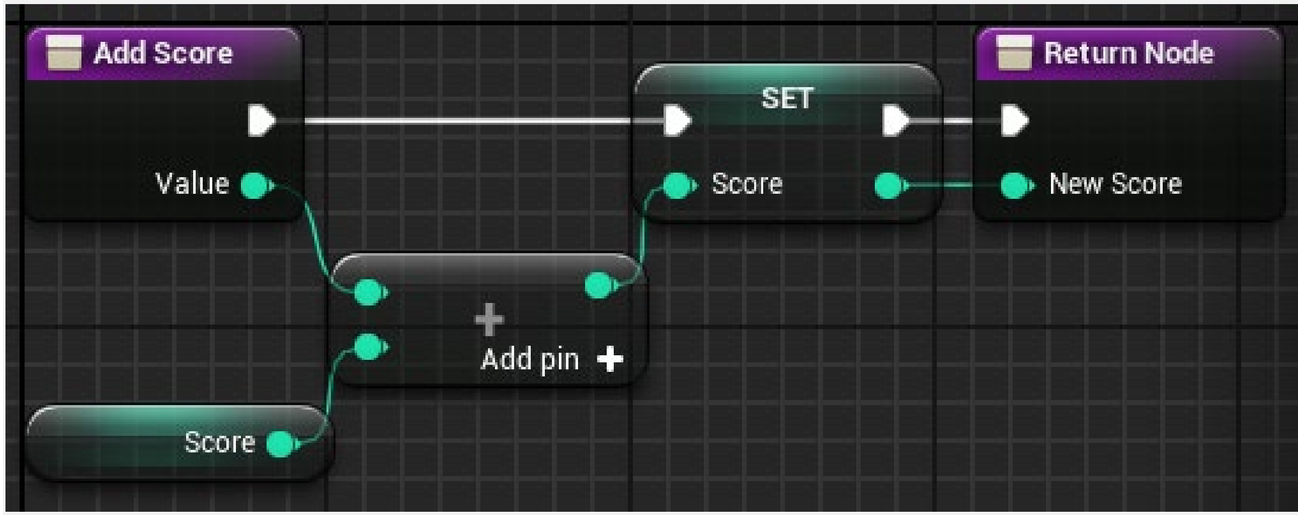
Output parameters are values that can be returned from a function.

To add input or output parameters, select the function in the **My Blueprint** panel and use the **Details** panel.

The images on the right show a function with an input parameter named “**Value**” that is added to the **Score** variable.

The result of the sum is set in the **Score** variable, then returned with the output parameter named “**New Score**”.

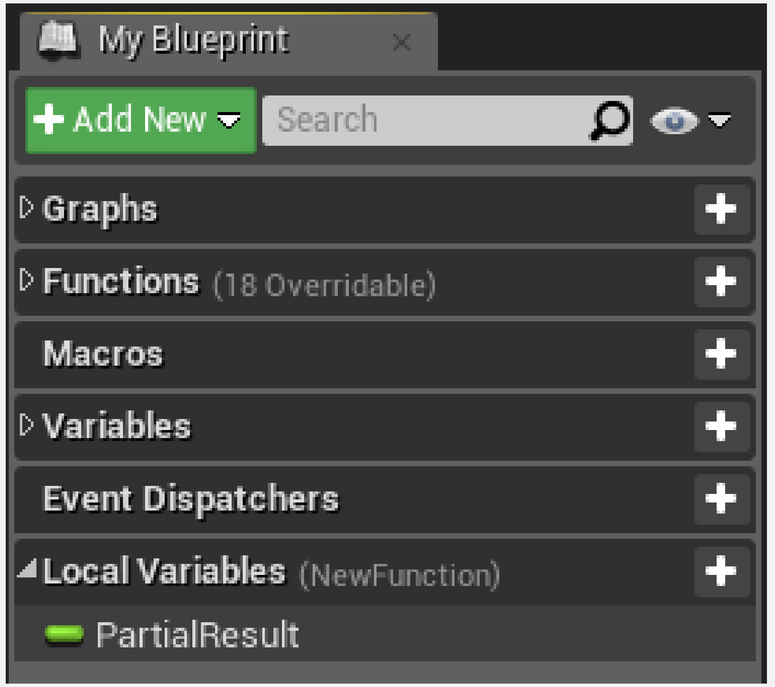




FUNCTIONS:
LOCAL VARIABLES

Functions allow the use of **local variables** that are only visible inside the function. They are very effective at assisting in complex functions and do not mix with the other variables of the Blueprint.

To create a local variable, double-click on a function to edit it, then look at the **My Blueprint** panel. At the bottom of the panel, you will find a category named **Local Variables** with the name of the function in parentheses. Click the “**+**” button in the **Local Variables** category.

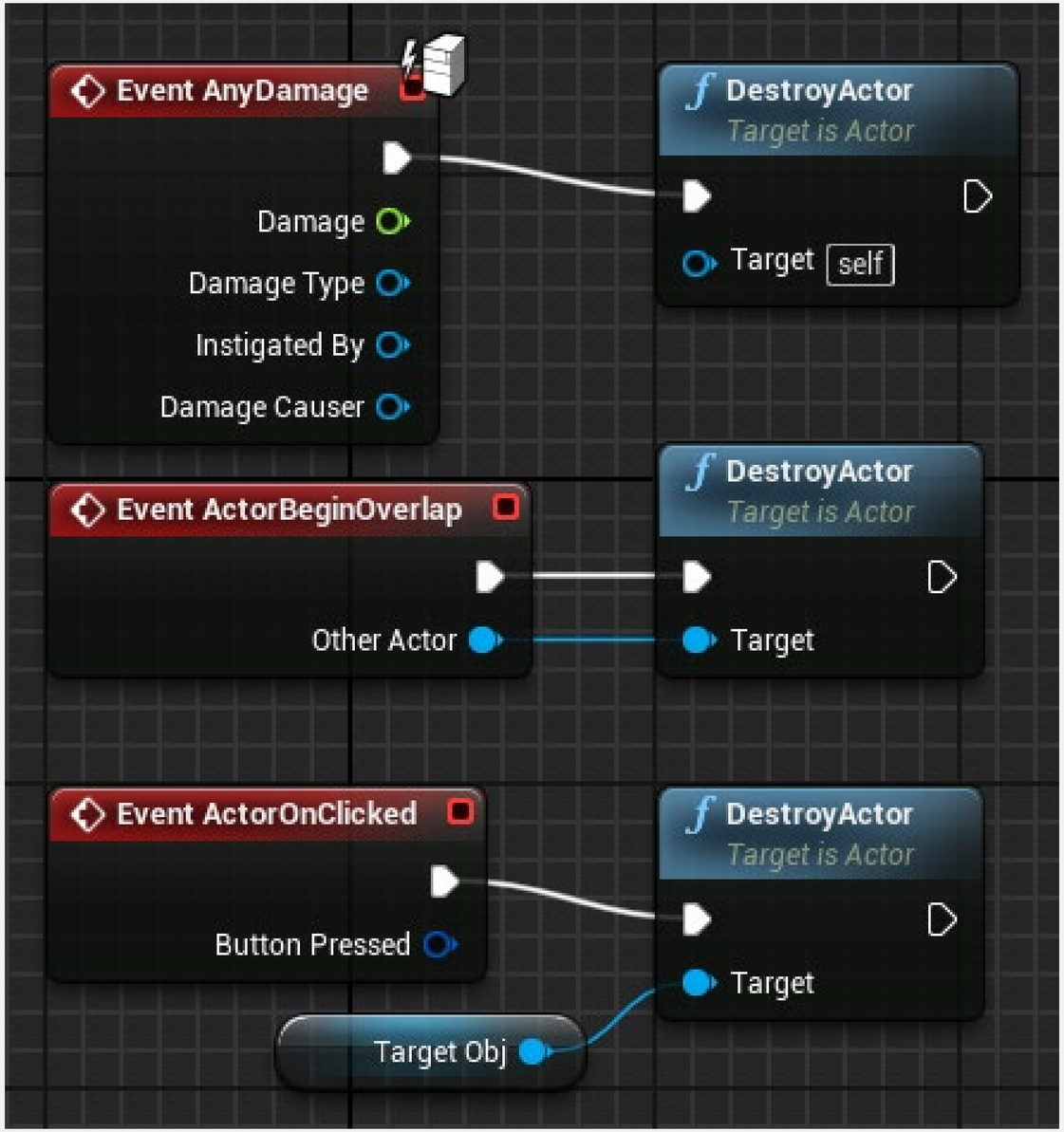


FUNCTIONS:
THE TARGET PARAMETER

The **Target** parameter is common to several functions and indicates the object that will be modified with the function call.

The default value for this parameter is “**self**”, which is a special reference to the Actor or Object instance that owns the script being executed.

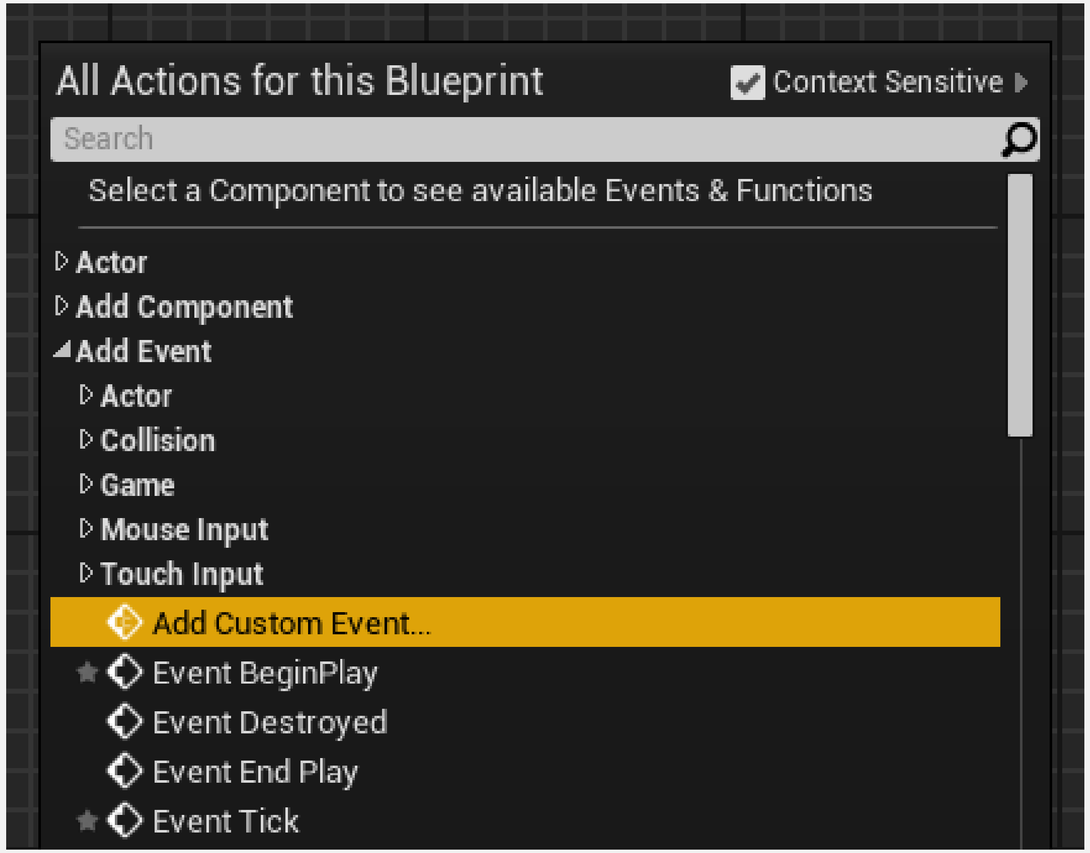
The image on the right shows different ways to use the **Target** parameter of the **DestroyActor** function.



CUSTOM EVENTS

Unreal Engine provides a number of predefined events, but it is possible to create new ones to use in a Blueprint. These events can be called from both the Blueprint they are defined in and other Blueprints.

To create a **custom event**, right-click in the **Event Graph**, expand the **Add Event** category, and select "**Add Custom Event...**".

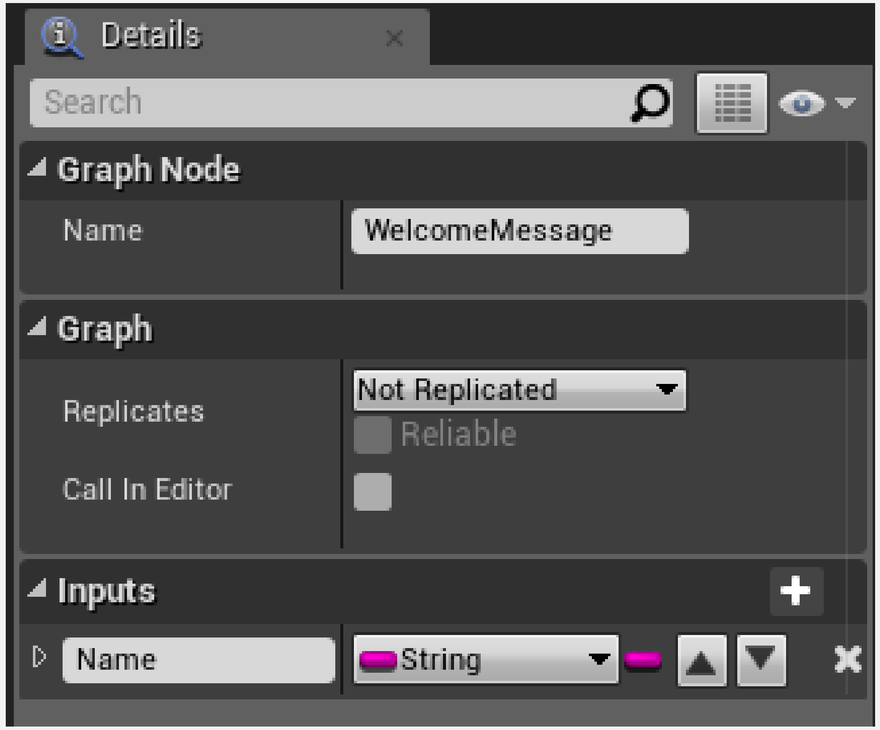


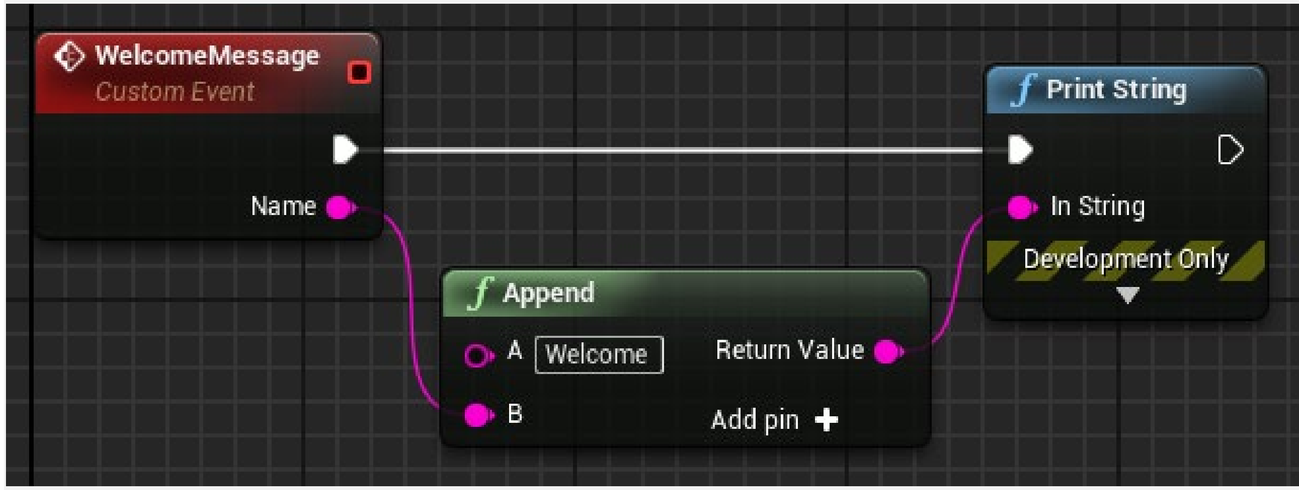
CUSTOM EVENTS: INPUT PARAMETERS

Selecting a Custom Event node allows you to manage the event name and input parameters. Events do not have output parameters.

The images on the right show a custom event named “**WelcomeMessage**” with an input parameter called “**Name**”.

The event will create and print on screen a custom message using the name passed as a parameter.

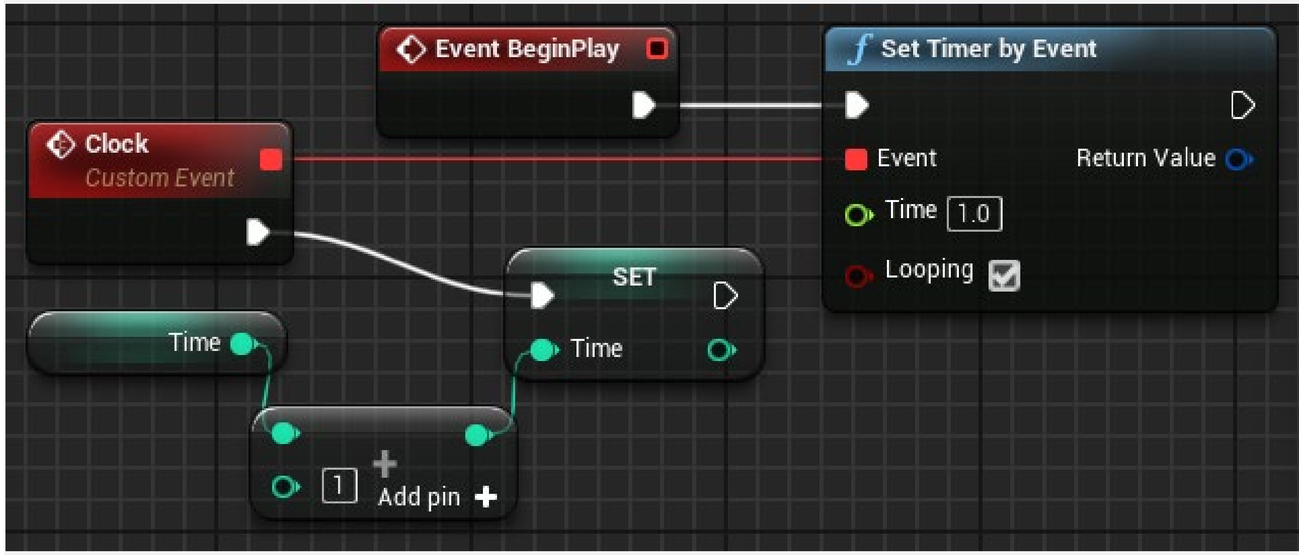




CUSTOM EVENTS: DELEGATES

An event has a small red square in the right corner that is known as a **delegate**. This is just a reference to the event. Some actions receive an event as a parameter, and using the delegate makes that possible.

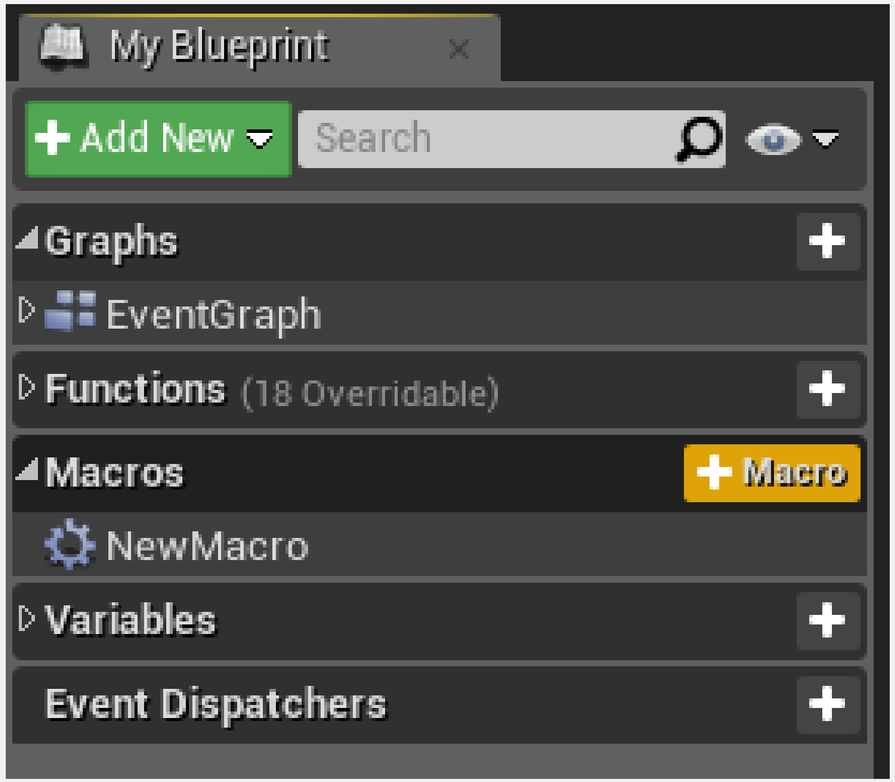
In the image on the right, the delegate of the custom event named “**Clock**” is wired to the **Set Timer by Event** node’s **Event** input pin, so the **Clock** event will be called every second.



MACROS

Another way to gather actions in one common place is to use **macros**. A macro is similar to a collapsed graph of nodes. At compile time, the actions of a macro are expanded in the places that the macro is being used.

Macros can have input and output parameters, as well as several input and output execution pins.



MACROS VS. EVENTS VS. FUNCTIONS

Macros, custom events, and functions provide different ways to organize script. Each of them has its advantages and limitations. One thing they have in common is that they all have input parameters.

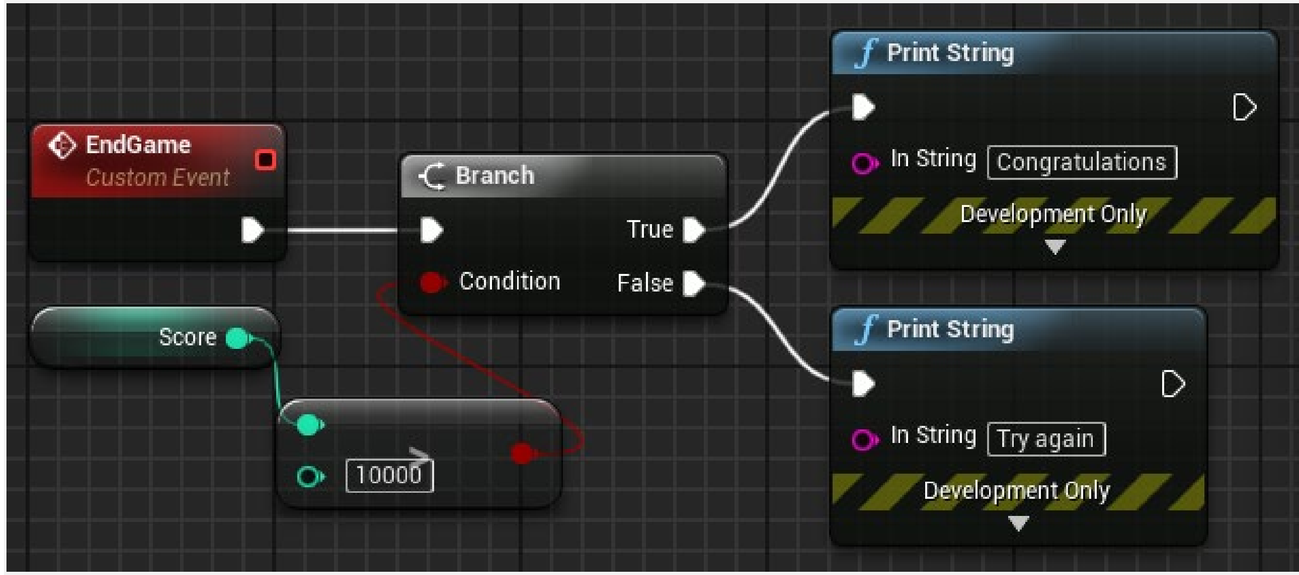
- **Macros** have output parameters and can have many execution paths. They cannot be called from another Blueprint.
- **Events** do not have output parameters. They can be called from other Blueprints and have the “delegate” reference. They support Timelines.
- **Functions** can be called from other Blueprints and have output parameters. Functions do not support latent actions, such as the Delay action.

PROGRAM FLOW

BRANCH NODE

The **Branch** node directs the flow of execution of a Blueprint based on the value of the Boolean input “Condition”, which can be “true” or “false”.

In the image on the right, there is a custom event that is called at the end of the game. The **Branch** node is used to test if the score is greater than “**10000**”. A different message will be shown based on the result.



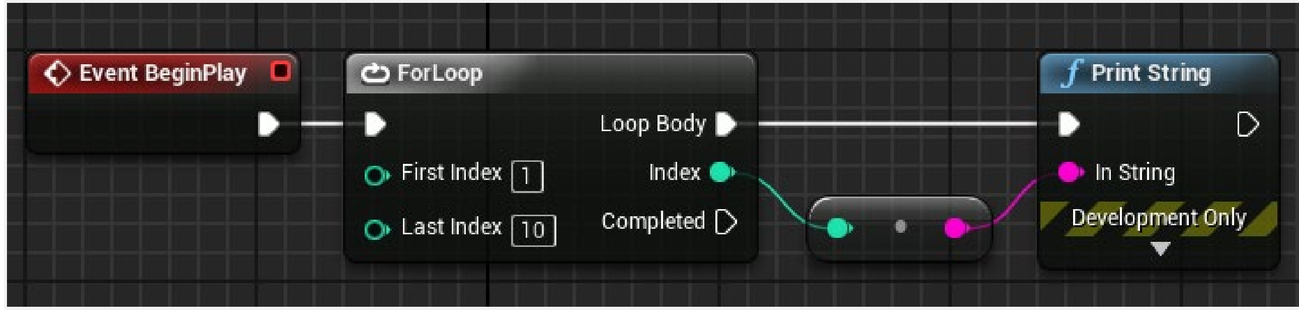
FOR LOOP NODE

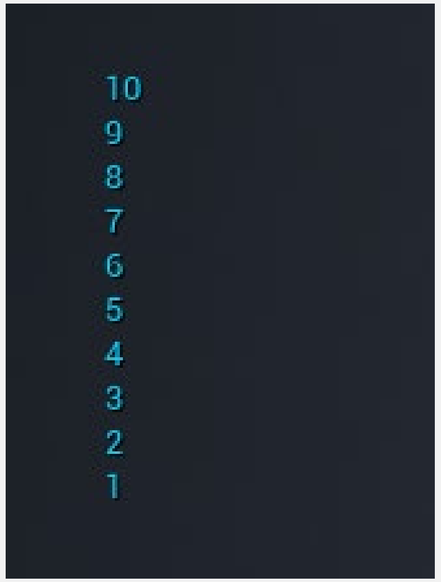
The **ForLoop** node performs the set of actions that are associated with the output pin **Loop Body** for each index.

When the **ForLoop** node completes its execution, the output pin **Complete** is triggered.

On the right, a **ForLoop** node is used to execute the **Print String** node ten times. The value of the **Index** output pin of the **ForLoop** node is used as the input for the **Print String** node.

The conversion node is automatically created by the Editor when an integer value is connected to a string input.

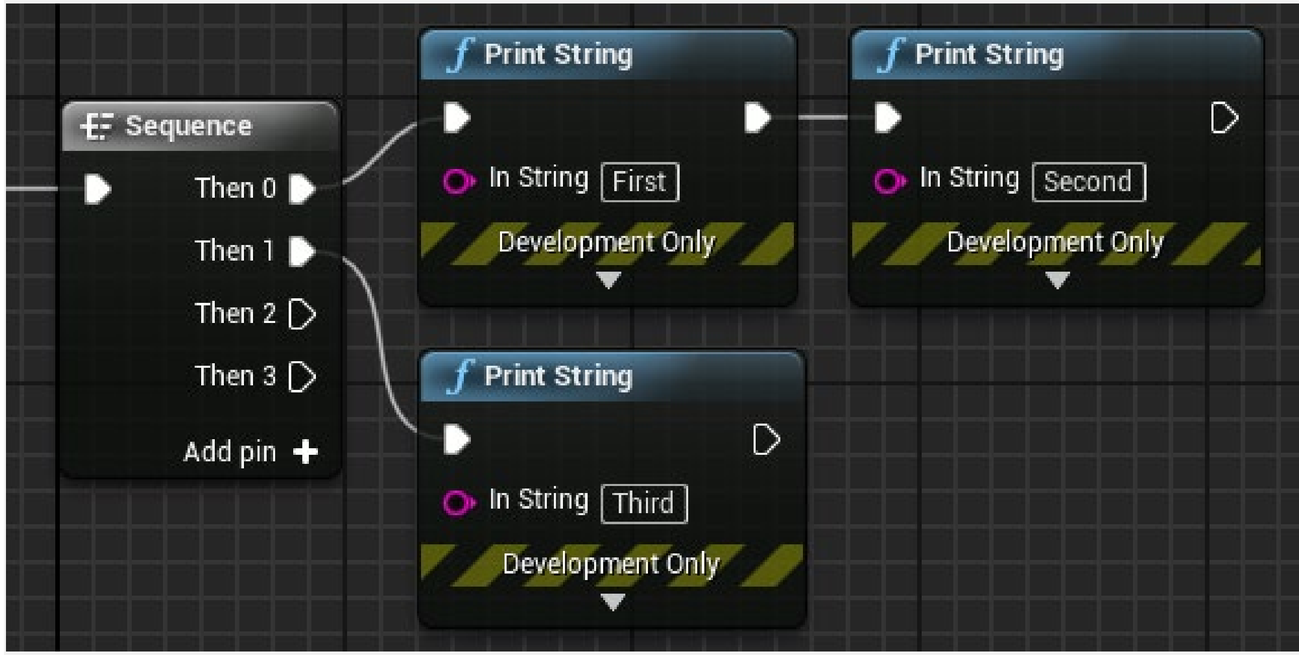




SEQUENCE NODE

A **Sequence** node can be used to help organize other Blueprint actions. When triggered, it executes all the nodes connected to the output pins in sequential order—that is, it executes all the actions of pin **Then 0**, then all the actions of pin **Then 1**, and so on.

Output pins can be added using the **Add pin +** option. To remove a pin, right-click on the pin and choose the **Remove execution pin** option.



SUMMARY

This lecture showed how to create variables and how to use operators to create expressions.

Functions, custom events, and macros were presented, and the lecture introduced some basic nodes that control the program flow.