HOUR 14

Blueprint Editor: Introduction to Visual Scripting

INTRODUCTION

The Blueprint visual scripting system is a powerful and fully functional scripting environment that is used throughout UE4.

It gives artists and designers the ability to modify existing gameplay elements, prototype ideas, and create entire games. This lecture serves as an introduction to the Blueprint Editor and basic scripting concepts.

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Become familiar with the Blueprint Editor interface
- Learn about events, functions, and variables
- Learn about nodes, wires, execs, and pins

Outcomes

By the end of this lecture you will be able to

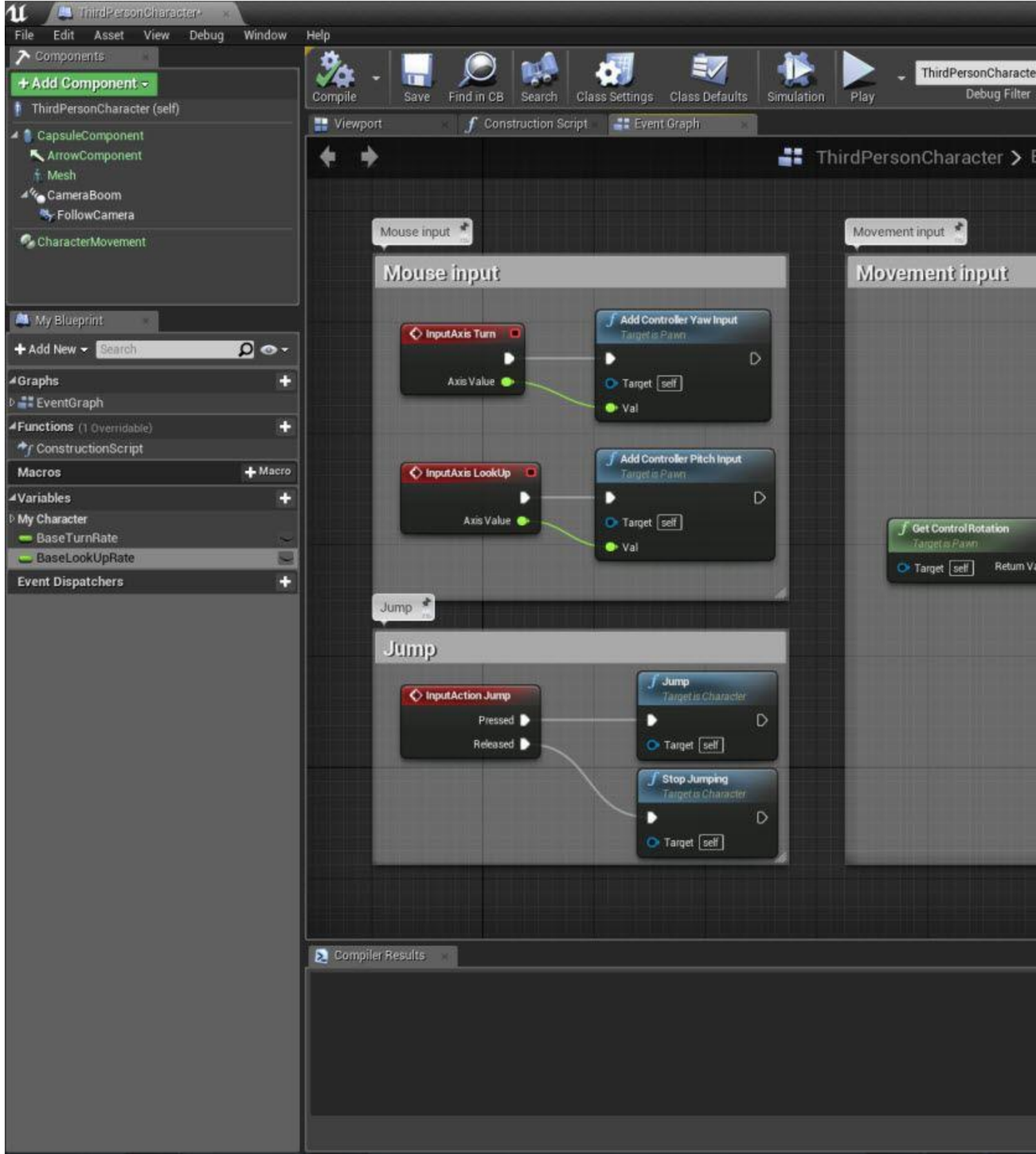
- Navigate the Blueprint Editor
- Understand how to place nodes and connect them with wires
- Create variables

VISUAL SCRIPTING

Basic Concepts

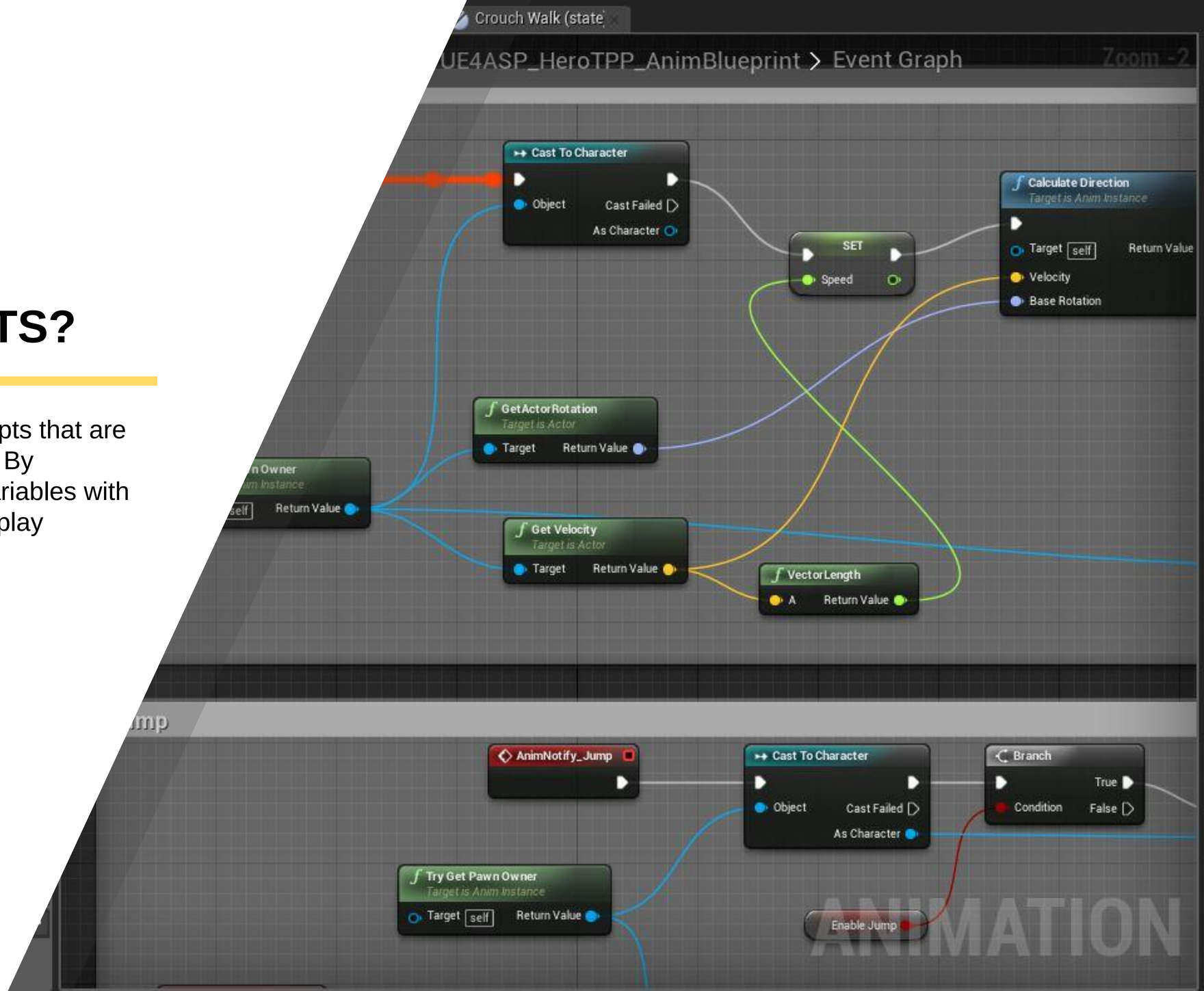
BLUEPRINT WORKFLOW

The Blueprint visual scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within the Unreal Editor. As with many common scripting languages, it is used to define object-oriented (OO) classes or objects in the engine.



WHAT ARE BLUEPRINTS?

In their basic form, Blueprints are visual scripts that are used to add new functionality to your game. By connecting nodes, events, functions, and variables with wires, it is possible to create complex gameplay elements.

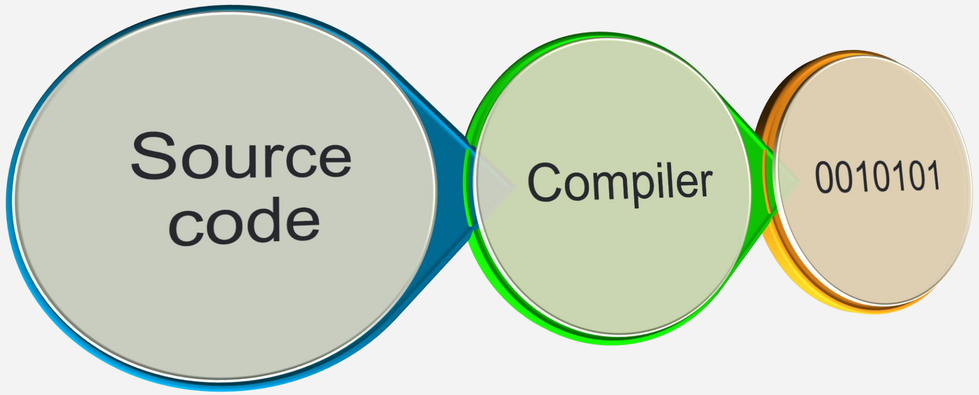


BASIC SCRIPTING TERMS

Before you begin working with Blueprints, you should have a basic understanding of common scripting terms.

When a Blueprint script has been created, it needs to be compiled.

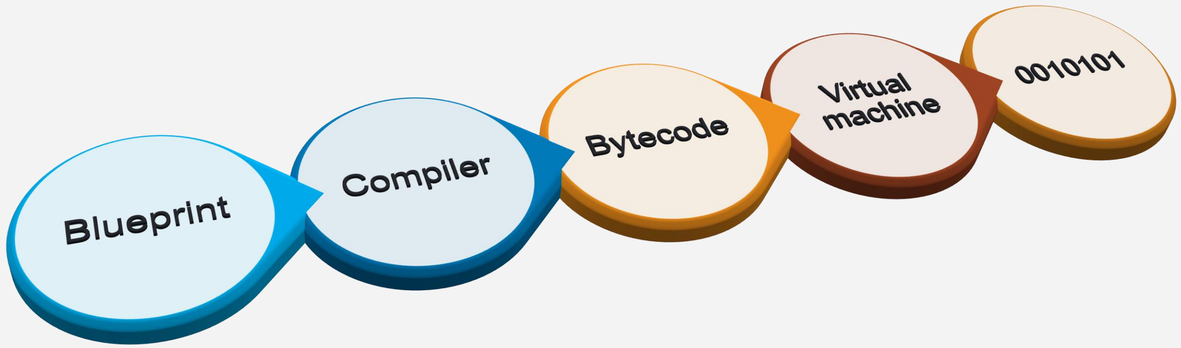
- Compiling is the process of turning source code into machine language that can be executed by the CPU. Compiling requirements change depending on the hardware and operating system.
- A compiler is the software used to compile source code written in a programming language.



BASIC SCRIPTING TERMS

Blueprints are compiled to the bytecode level and processed by a virtual machine (VM) instead of hardware.

- A virtual machine is software that translates bytecode into instructions that the hardware can understand and process.

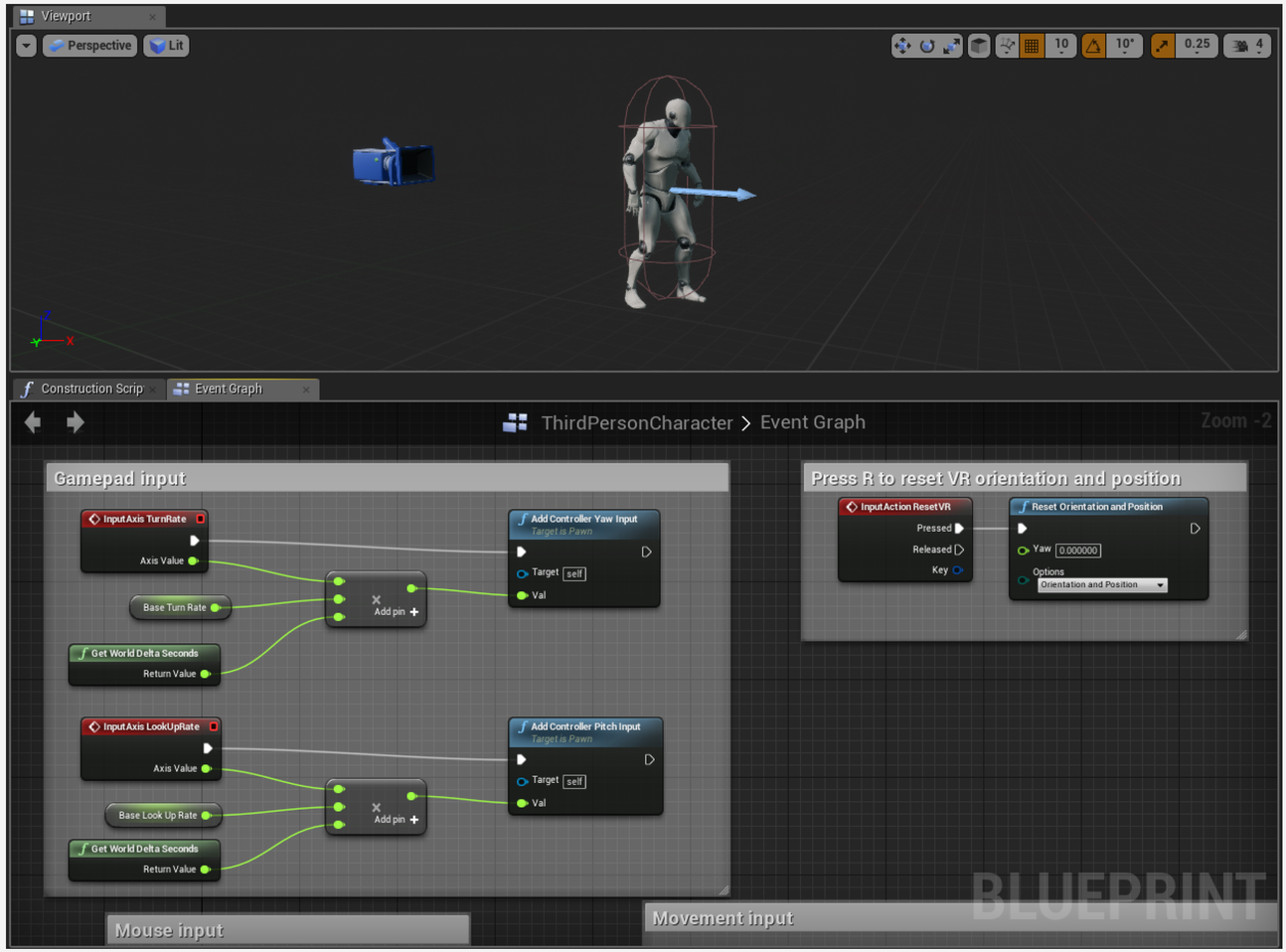


By default, Blueprints are compiled to the bytecode level and processed by a virtual machine.

Because of this extra layer, Blueprints tend to be less efficient than compiled C++ code. However, it is possible to “Nativize” Blueprints during the packaging process. This effectively converts most aspects of Blueprint to C++, which can then be compiled normally.

BLUEPRINTS

Blueprints are used to drive Level-based events, to control internally scripted behaviors for in-game Actors, and even to control complex animations for game character systems.

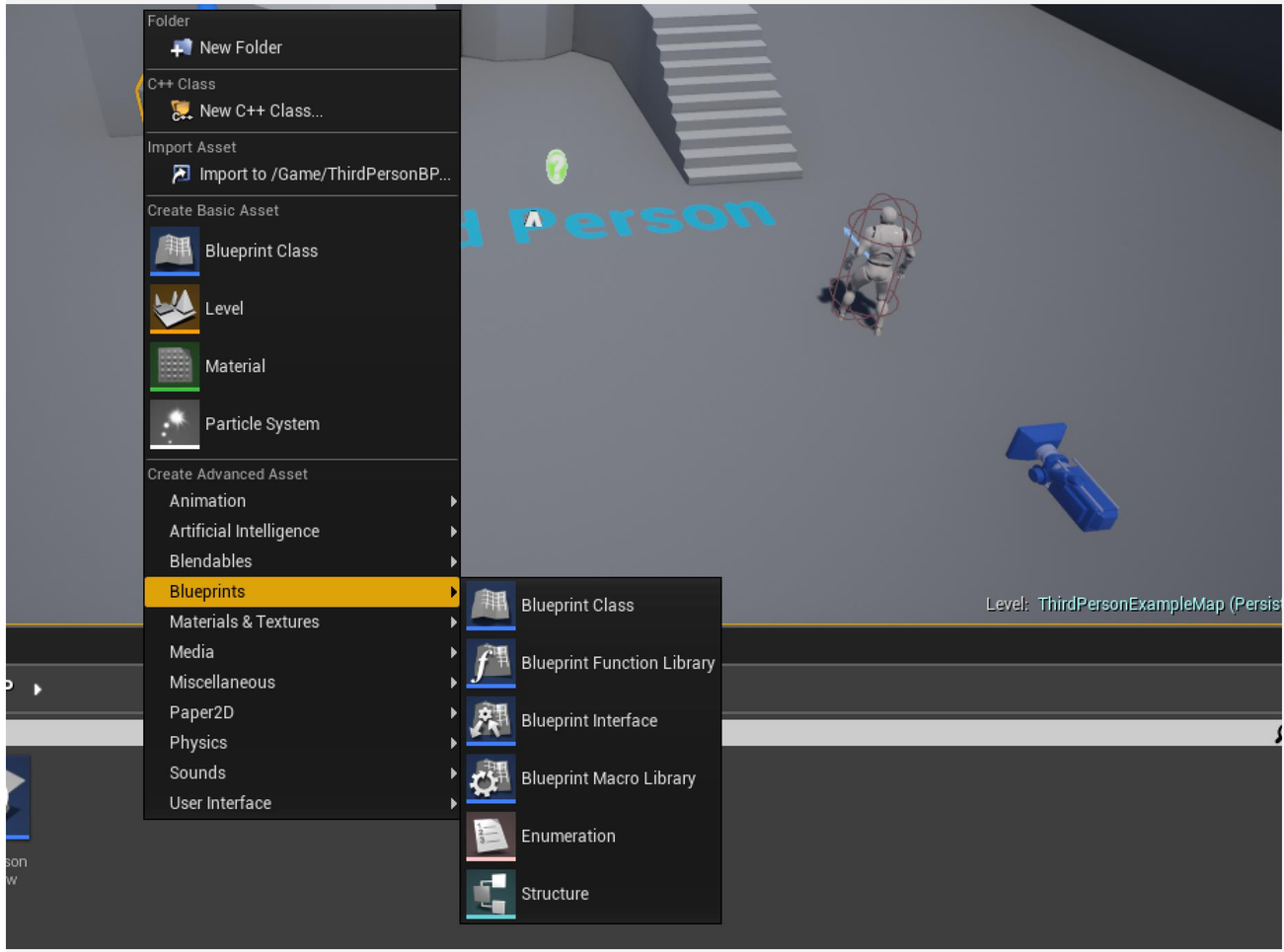


TYPES OF BLUEPRINTS

There are several different types of Blueprints:

- Level Blueprints
- Blueprint class
- Data-Only Blueprint
- Blueprint Interface
- Blueprints Macros
- Animation Blueprints

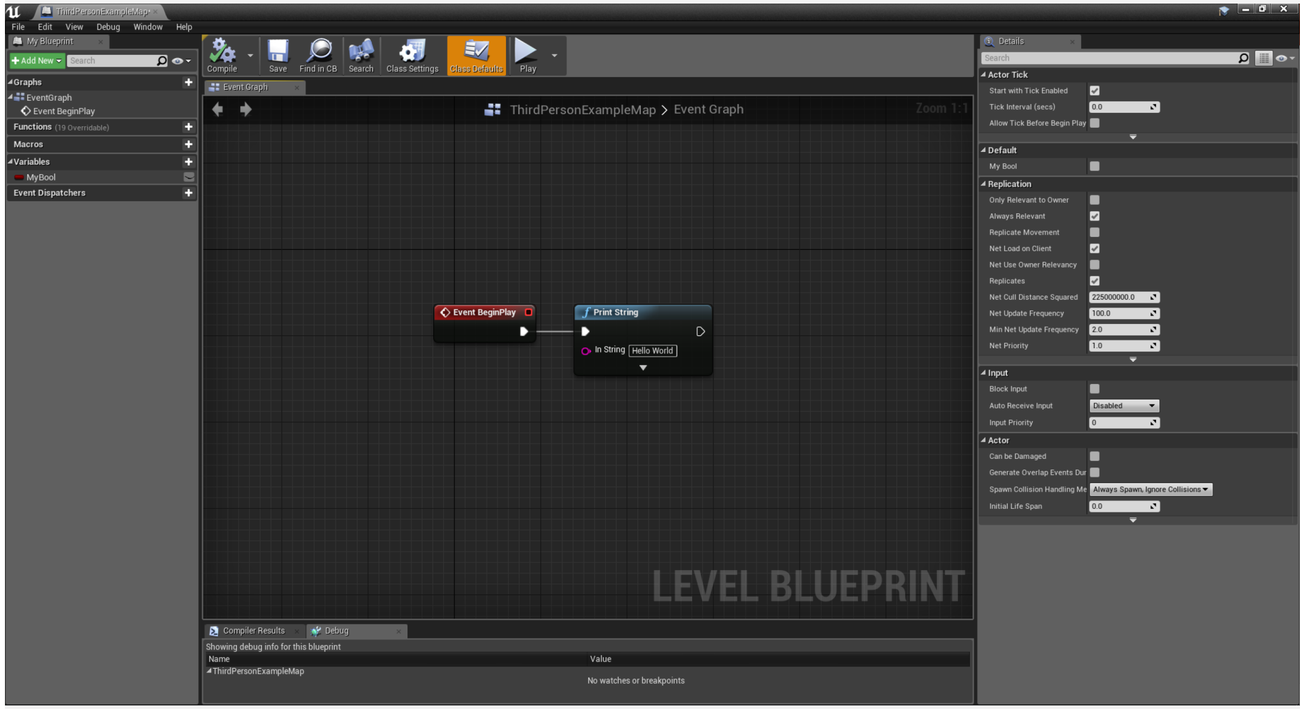
Level Blueprint and Blueprint classes are the two most common types you will use.



LEVEL BLUEPRINT

A Level Blueprint provides a control mechanism for Level streaming and binding events to Actors placed in the Level.

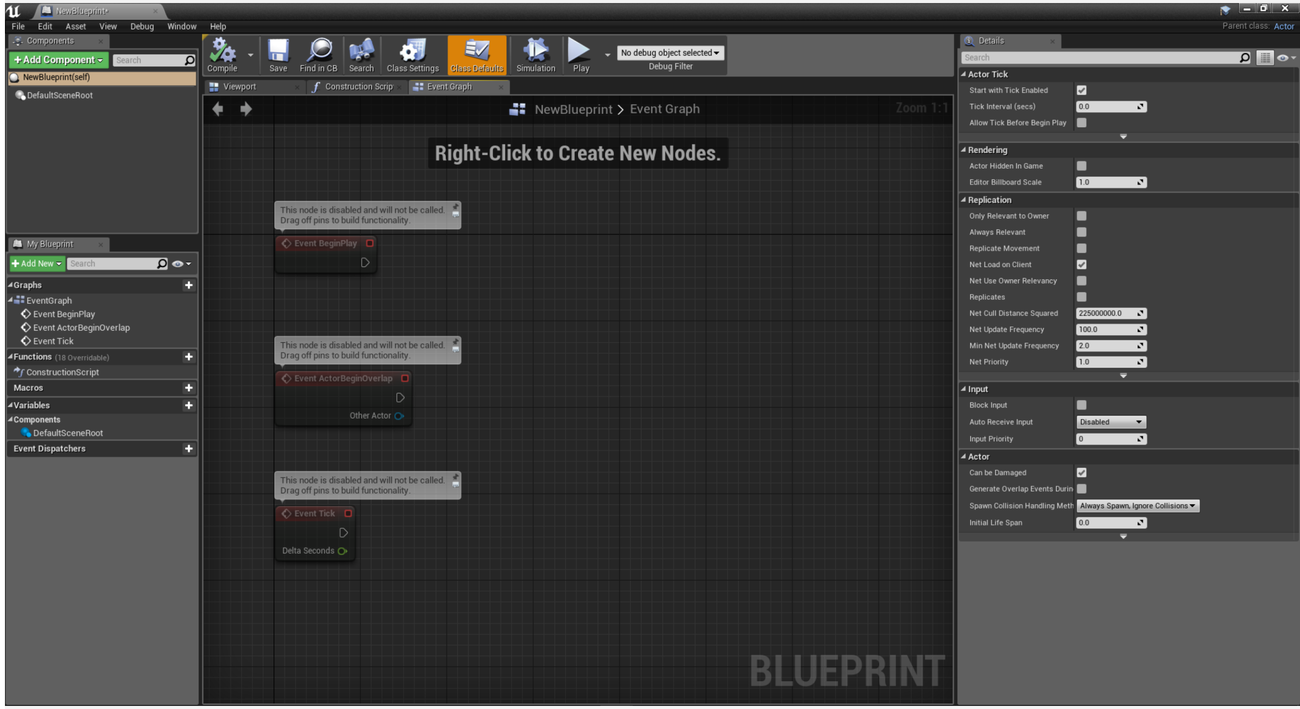
It is a specialized type of Blueprint that acts as a Level-wide global event graph. Each Level in your project has its own Level Blueprint created by default, and it is automatically saved when the Level is saved.



BLUEPRINT CLASS

Often shortened as “Blueprint,” a Blueprint class is an asset that allows content creators to easily add functionality on top of existing gameplay classes. Blueprints essentially define a new class or type of Actor that can then be placed into maps as instances.

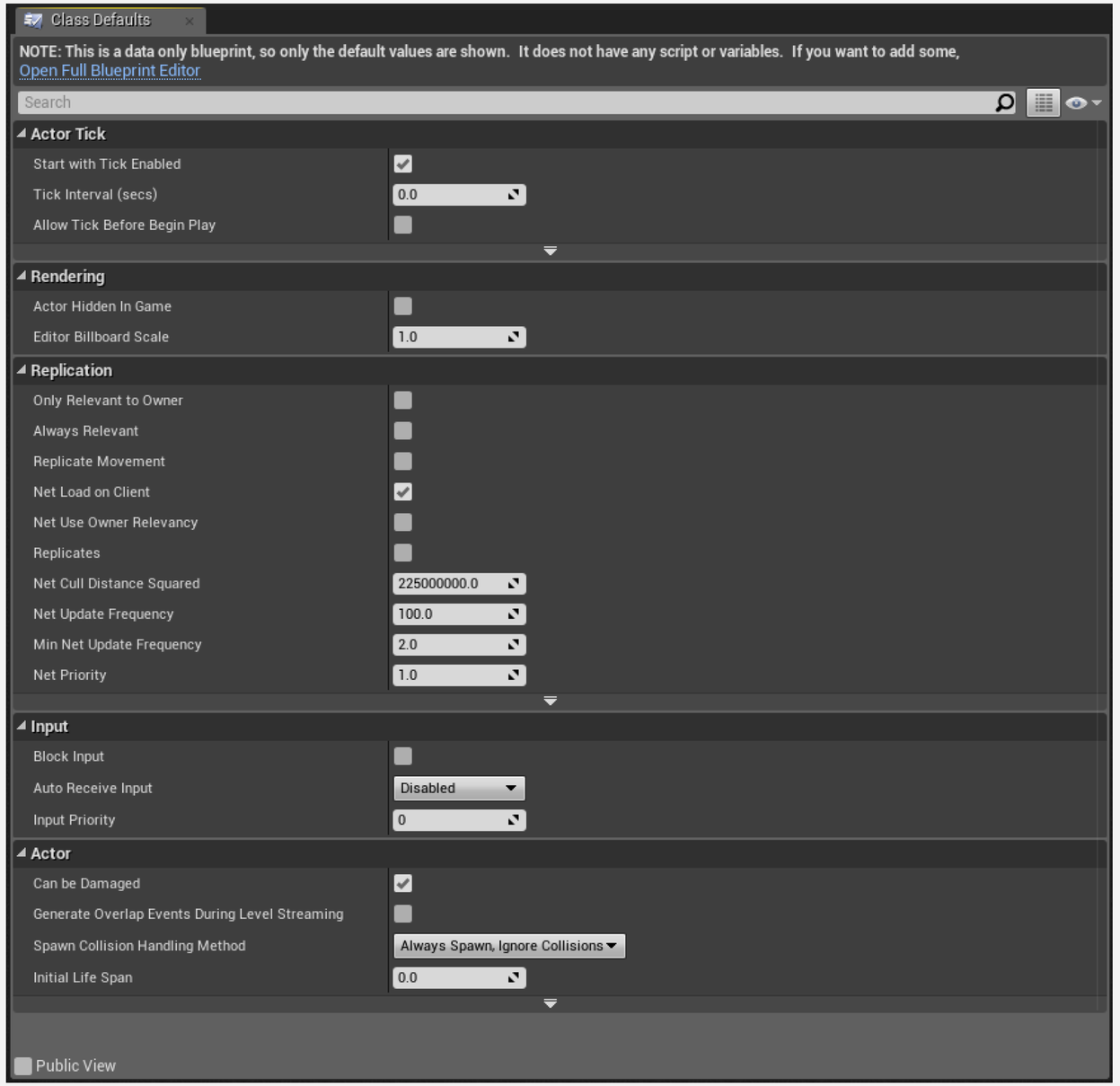
- A Blueprint class asset is created and stored in the Content Browser.
- A Blueprint Actor is an instance of a Blueprint class asset placed in a Level.



DATA-ONLY BLUEPRINT

A Data-Only Blueprint is a Blueprint class that contains only the code (in the form of node graphs), variables, and components inherited from its parent. Inherited properties can be tweaked and modified, but no new elements can be added.

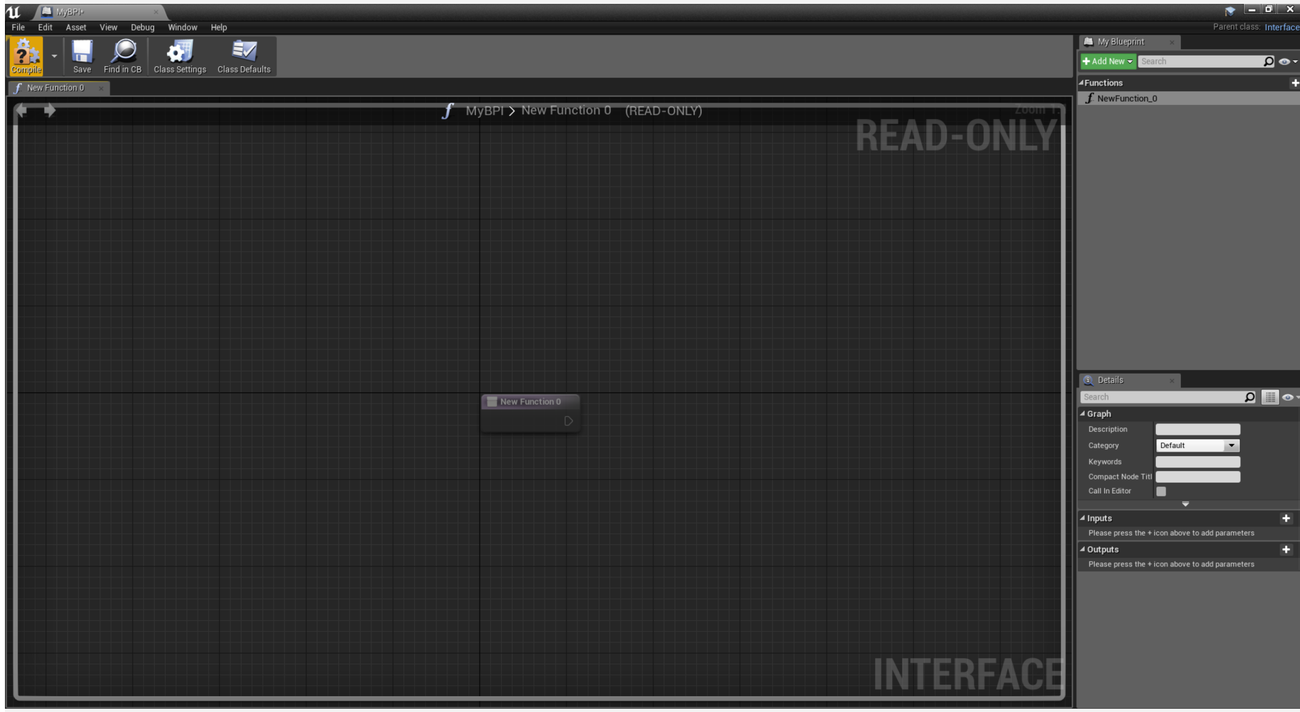
- Data-Only Blueprints are essentially a replacement for archetypes. Designers can use them to tweak properties or set items with variations.
- They are edited in a compact property editor, but they can also be “converted” to full Blueprints by simply adding code, variables, or components using the full Blueprint Editor.



BLUEPRINT INTERFACE

The Blueprint Interface is a collection of one or more functions—name only, no implementation—that can be added to other Blueprints. Blueprint Interfaces allow different Blueprints to share data and to send data to one another.

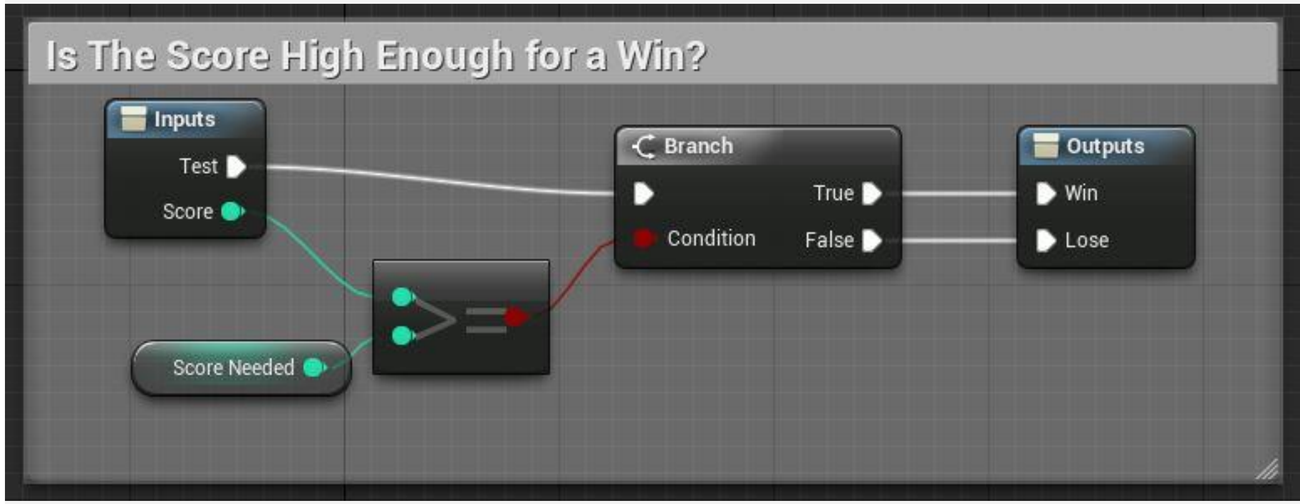
Blueprint Interface assets are created in the Content Browser and edited in the Blueprint Interface Editor.



BLUEPRINT MACROS

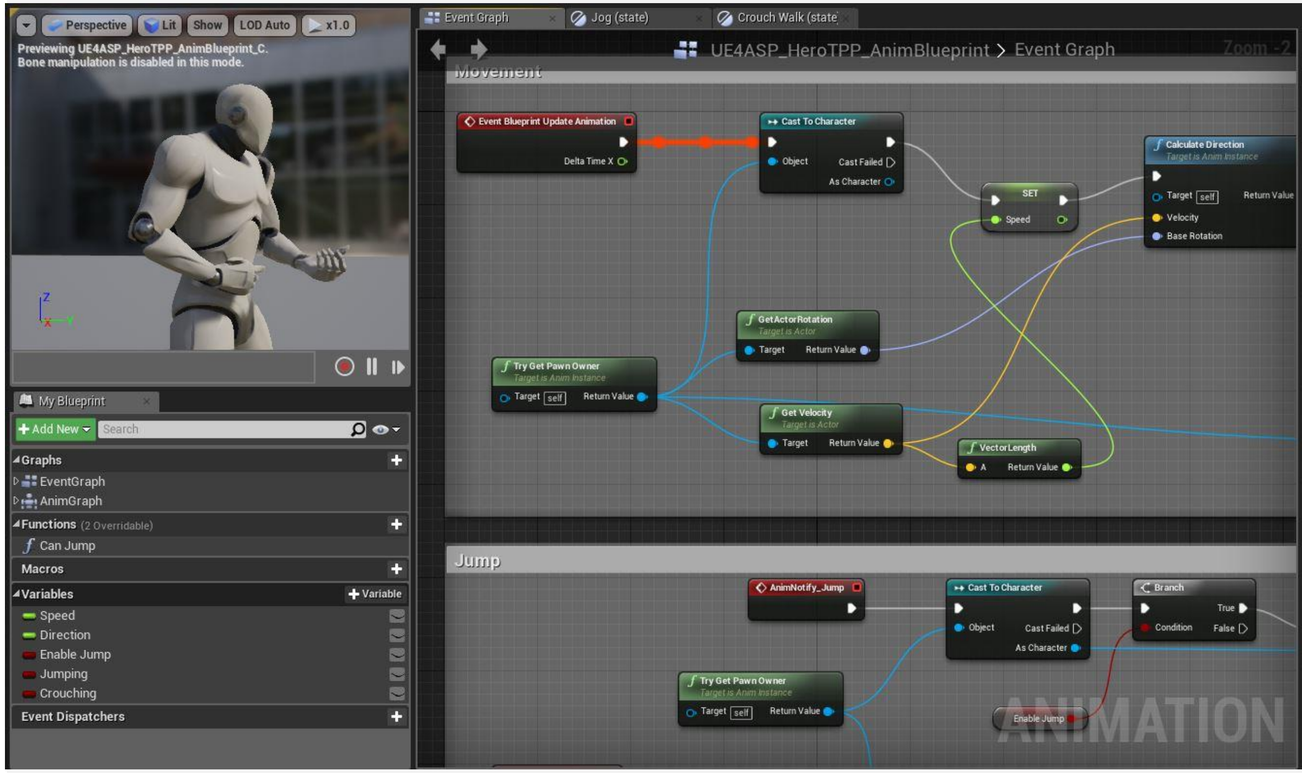
A Blueprint Macro Library is a container that holds a collection of Blueprint Macros or self-contained graphs that can be placed as nodes in other Blueprints. These can be time-savers, as they can store commonly used sequences of nodes complete with inputs and outputs for both execution and data transfer.

Macros are shared among all graphs that reference them, but they are auto-expanded into graphs as if they were a collapsed node during compiling. This means that Blueprint Macro Libraries do not need to be compiled. However, changes to a Blueprint Macro are only reflected in graphs that reference that macro when the Blueprint containing those graphs is recompiled.



ANIMATION BLUEPRINTS

Animation Blueprints control the animation of a Skeletal Mesh. Graphs are edited inside the Animation Blueprint Editor, where you can perform animation blending, directly control the bones of a Skeleton, or set up logic that will define the final animation pose for a Skeletal Mesh to use per frame.



As you use UE4, you'll often find that objects defined using Blueprint are colloquially referred to as just "Blueprints."

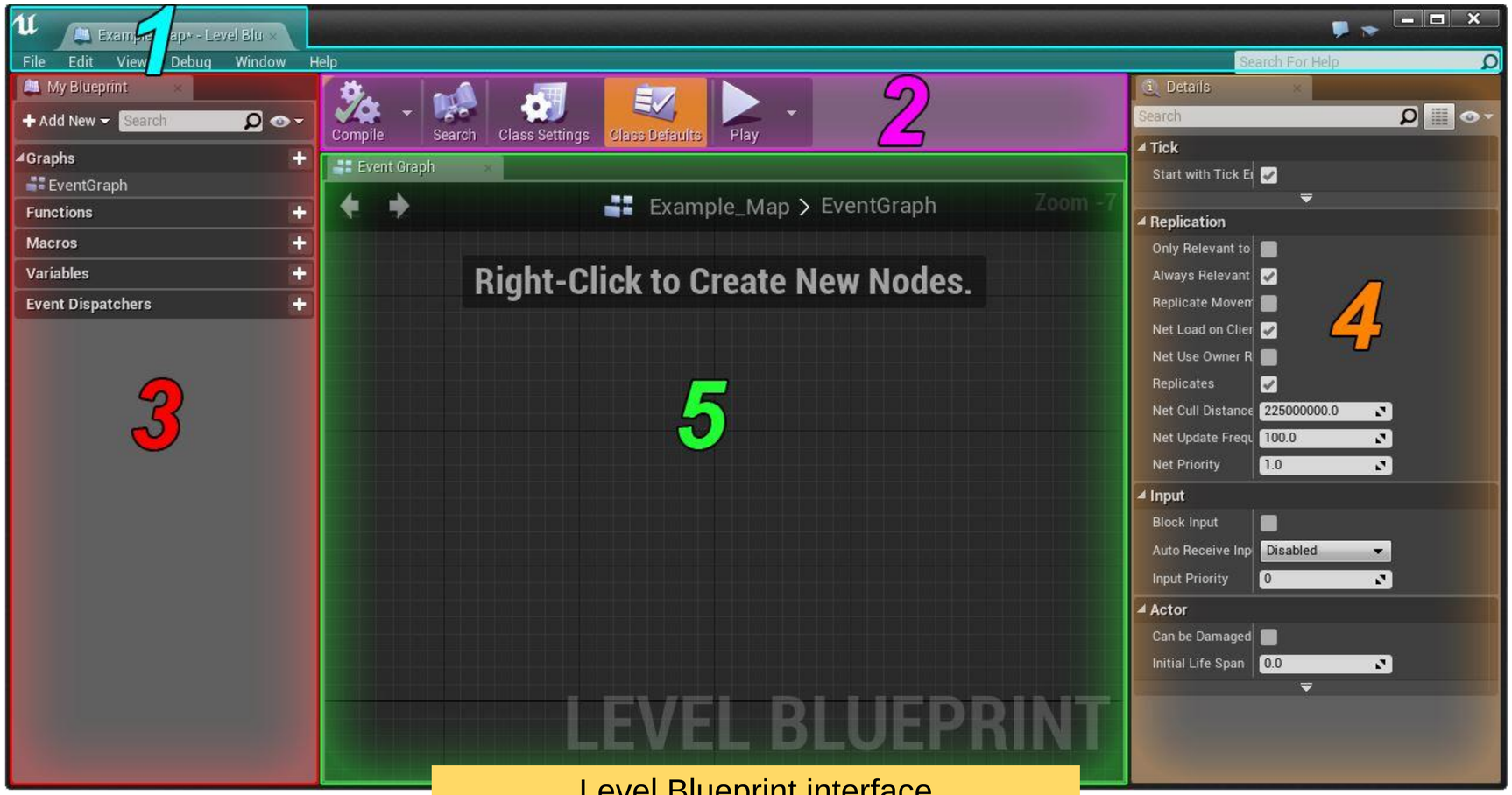
No matter the differences, the Blueprint Editor always performs the same key task: it enables you to create and edit powerful visual scripts to drive various aspects of your game.

The Blueprint Editor’s location and available toolset will be slightly different based on the type of Blueprint you are currently editing.

For each type of Blueprint, the location where you edit it, as well as the tools available to you, will change. This means that there are actually multiple places and different ways within UE4 that the Blueprint Editor can appear.

BLUEPRINT EDITOR

Interface



Level Blueprint interface

BLUEPRINT EDITOR INTERFACE

1. **Menu bar**: The menu bar contains various menus for managing Blueprint.
2. **Toolbar**: The toolbar provides a number of buttons for controlling the Blueprint Editor.
3. **My Blueprint panel**: This panel is used to manage graphs, functions, macros, and variables that are in your Blueprint.
4. **Details panel**: Once a component, variable, or function is added to a Blueprint, you can edit its properties in the Details panel.
5. **Graph Editor panel**: This is where the core functionality of the Blueprint is scripted.

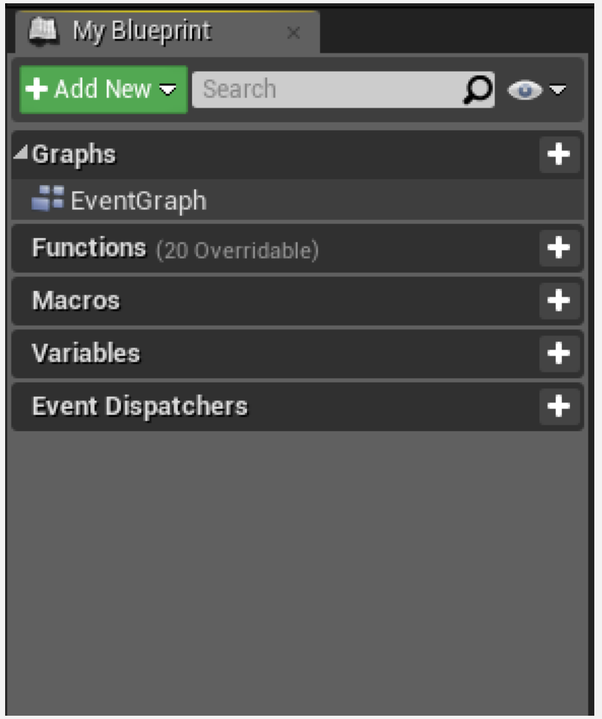
TOOLBAR

The toolbar is displayed at the top left of the Blueprint Editor by default.

The Blueprint Editor toolbar buttons provide easy access to common commands needed when editing Blueprints. The toolbar provides different buttons depending on which mode is active and which Blueprint type you are currently editing.

MY BLUEPRINT PANEL

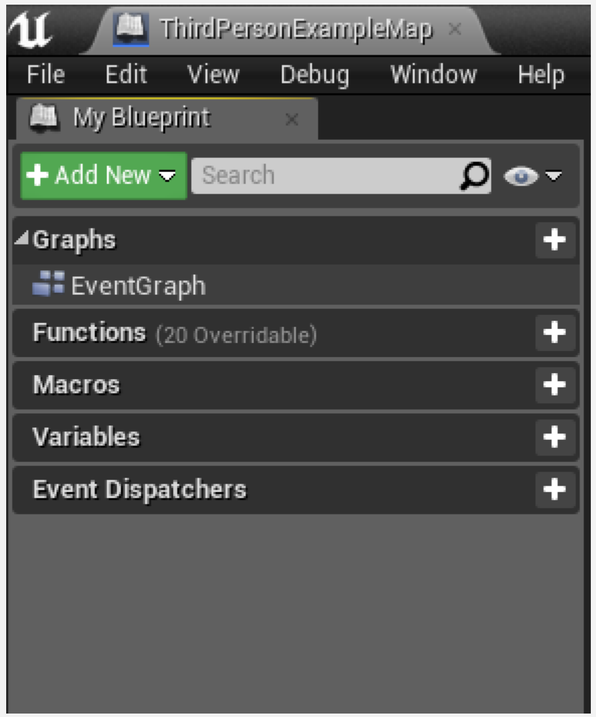
The My Blueprint tab displays a tree list of the graphs, scripts, functions, macros, and so forth contained within the Blueprint. It is essentially an outline of the Blueprint that lets you easily view existing elements of the Blueprint, as well as create new ones.



MY BLUEPRINT PANEL

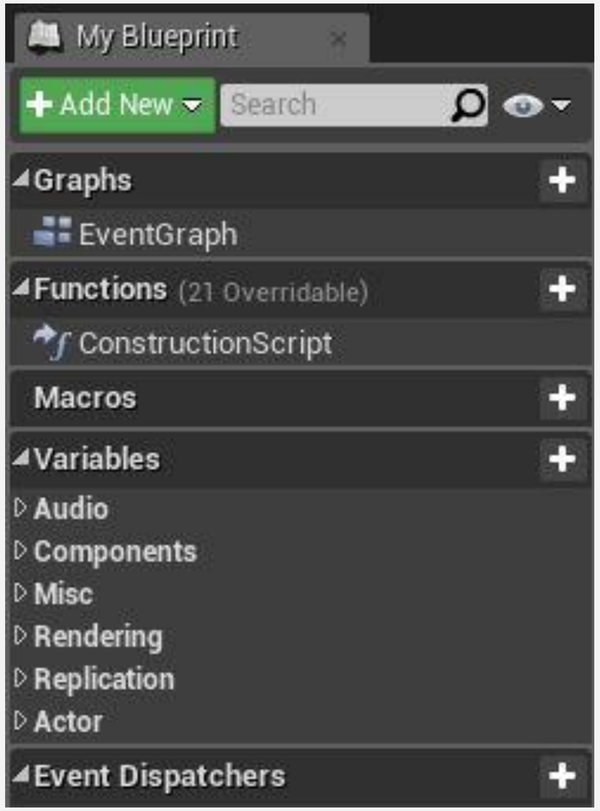
Different types of Blueprints will have different types of items shown in the My Blueprint tab tree list.

For example, a normal Blueprint will always have a Construction Script and an Event Graph. In addition, any functions created within the Blueprint will be displayed. A Level Blueprint will only have an Event Graph and any functions created within it.



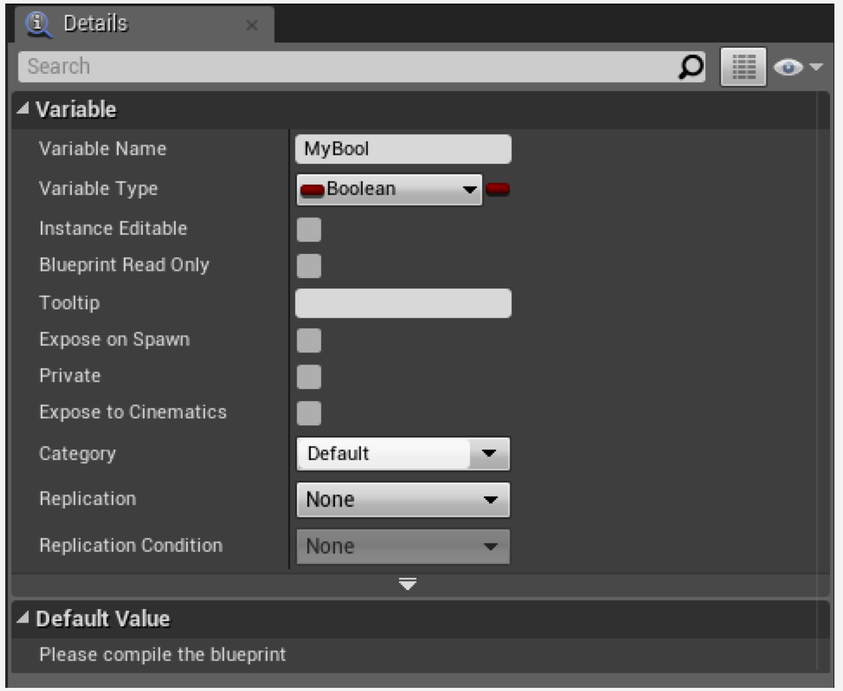
MY BLUEPRINT PANEL

The My Blueprint tab has shortcut buttons (the + at the end of each heading) for creating new variables, functions, macros, Event Graphs, and event dispatchers.



DETAILS PANEL

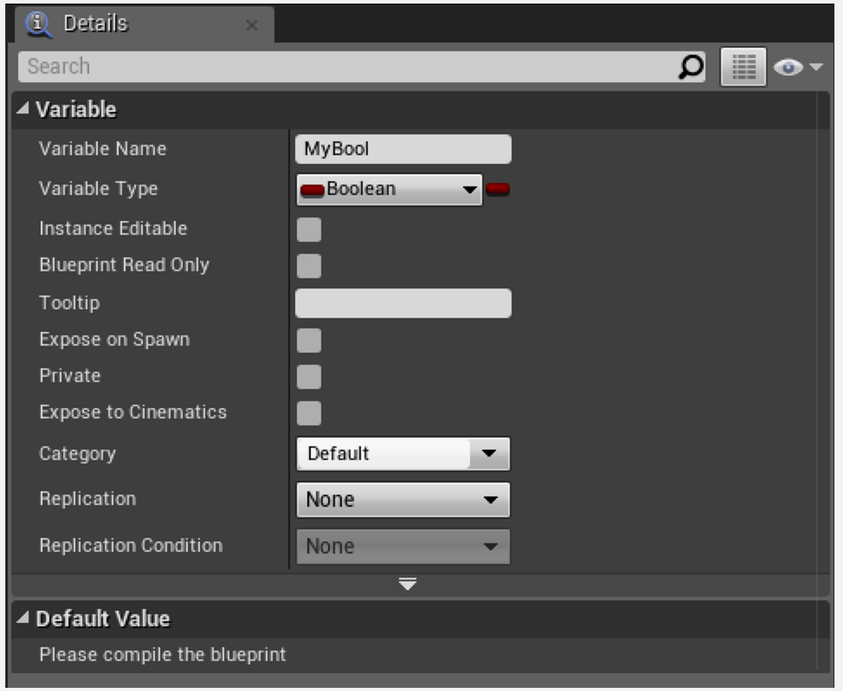
The Details panel is a context-sensitive area that provides editing access to the properties of selected items within the Blueprint Editor. It is composed of a search bar for fast access to specific properties and will generally contain one or more collapsible categories to organize the included properties.



DETAILS PANEL

The Details panel is also where you will handle many Blueprint editing tasks, including the following:

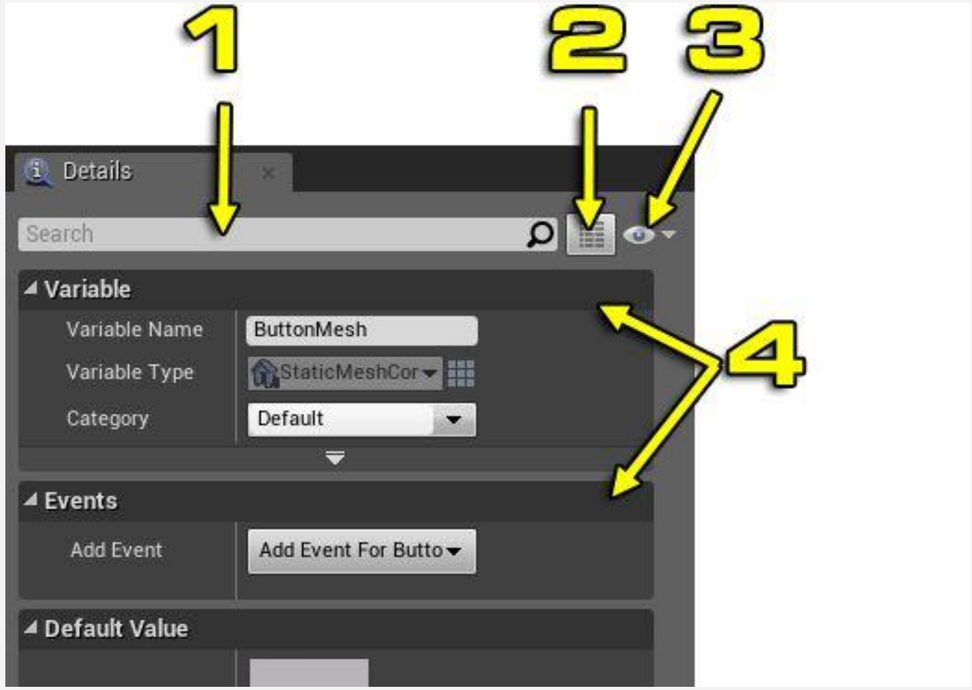
- Editing process for Blueprint variables, including changing name, type, and whether the variable is an array
- Implementing Blueprint Interfaces after clicking the Blueprint Props button
- Adding inputs and outputs for Blueprint functions
- Adding events for selected components



DETAILS PANEL

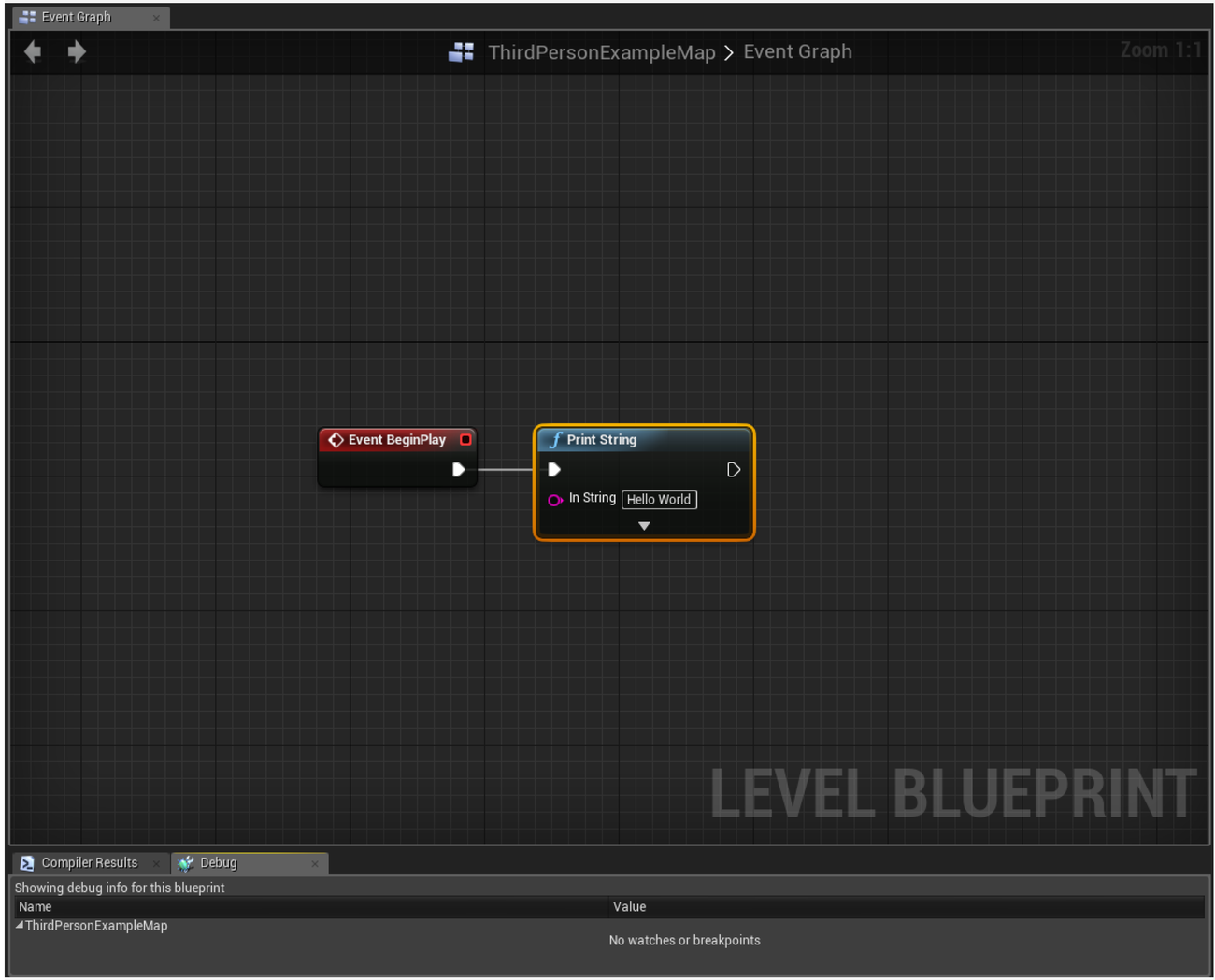
1. **Search filter**: Type the name of the property you need into this box, and the properties below filter down.
2. **Property Matrix**: Click on this button to open the Property Matrix panel for spreadsheet-like editing of available properties.
3. **Visibility filter**: Clicking on this icon allows you to show or hide modified or advanced properties, as well as collapse or expand all categories.
4. **Collapsible categories**: These areas are used to group together related properties, and they can be expanded and collapsed using the small white triangle to the left of the name.

.



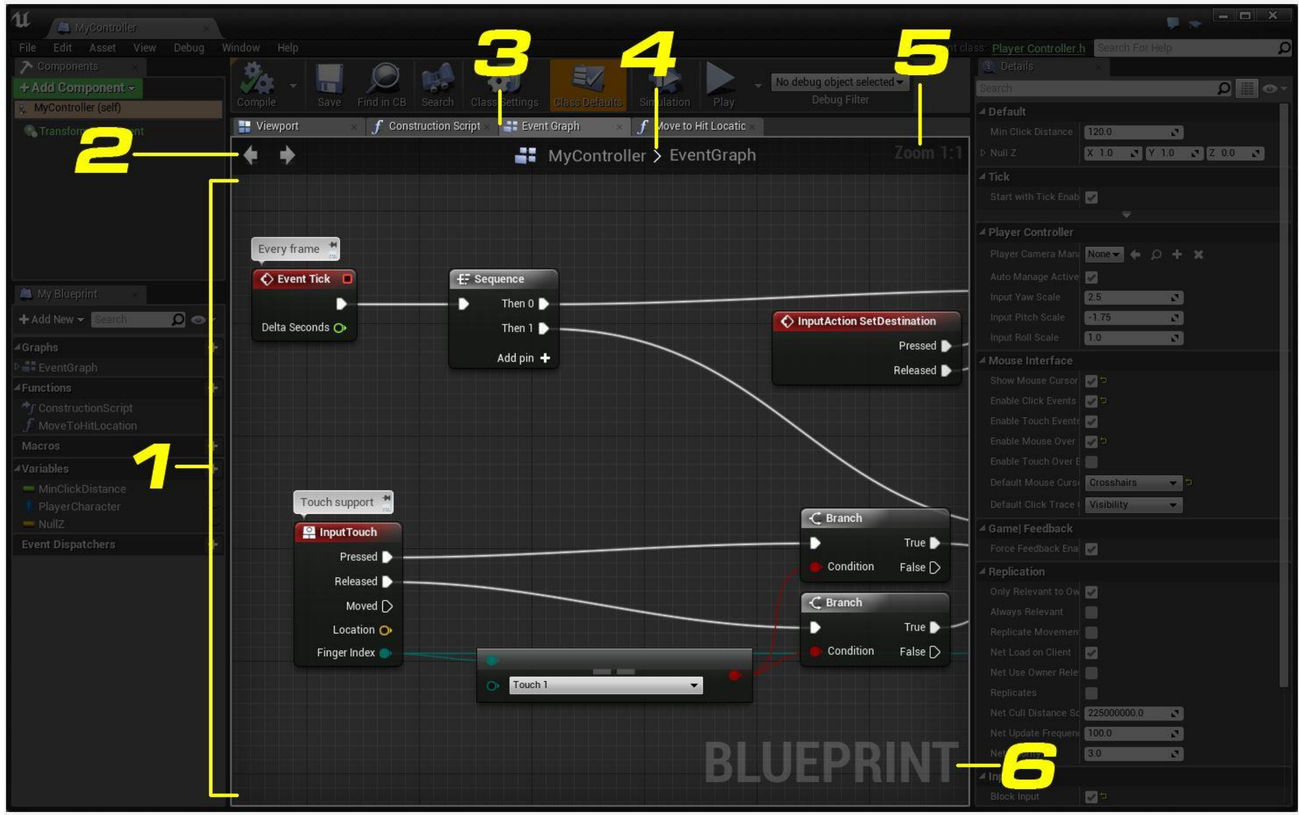
GRAPH EDITOR PANEL

The Graph Editor panel is the heart of the Blueprint system. It is where you create the networks of nodes and wires that will define your scripted behavior. Nodes can quickly be selected by clicking on them and repositioned via dragging.



GRAPH EDITOR PANEL

1. **Graph area:** This is where you will lay out all of your nodes.
2. **Forward and Back buttons**: These allow you to jump between different graphs very much as you would navigate a web browser.
3. **Tabs area**: As you open different graphs, tabs for each one will open here, allowing you to quickly jump between the graphs.
4. **Breadcrumbs**: These show a progression of graphs and subgraphs. As you step down into functions or collapsed graphs, the breadcrumbs will show you where you are within your network.
5. **Zoom Factor**: This simply shows the current zoom ratio in the Graph Editor.
6. **Blueprint label**: This shows the type of Blueprint you are editing. As you edit Blueprint Interfaces, Animation Blueprints, Blueprint Macros, and other types of Blueprints, this label will update.



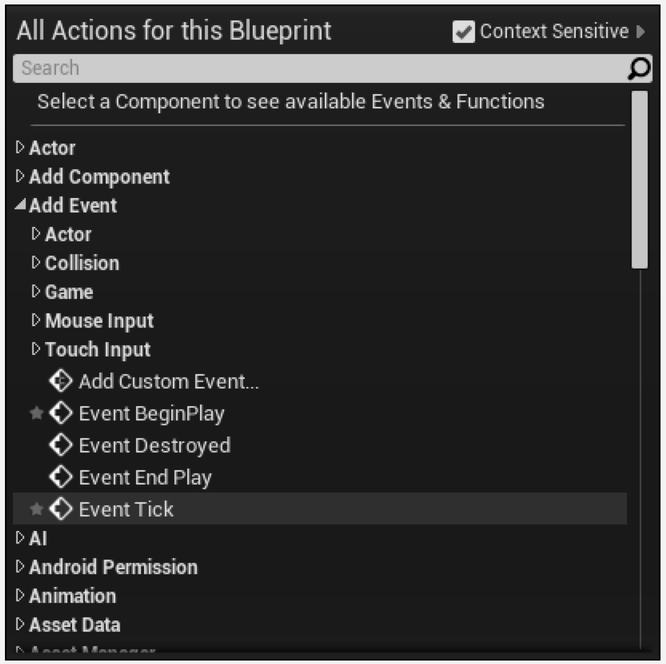
GRAPH EDITOR CONTROLS

The Graph Editor panel can be navigated using the controls outlined in the table to the right.

Control	Action
Right-click+drag	Pans the Event Graph.
Mouse scroll	Zooms the Event Graph in and out.
Right-click in empty space	Brings up the Blueprint Context Menu.
Click on node	Selects the node.
Click+drag in empty space	Selects the nodes inside the marquee selection box.
Ctrl+click+drag in empty space	Toggles selection of the nodes inside the marquee selection box.
Shift+click+drag in empty space	Adds the nodes inside the marquee selection box to the current selection.
Click+drag on node	Moves the node.
Click+drag from pin to pin	Wires the pins together.
Ctrl+click+drag from pin to pin	Moves wires from the origin pin to the destination pin.
Click+drag from pin to empty space	Brings up the Blueprint Context Menu showing only relevant nodes. Wires the original pin to a compatible pin on the created node.
Alt+click on pin	Removes all wires connected to the pin.

CONTEXT MENU

The Blueprint Context Menu is used to add events, functions, and conditional nodes to the Graph Editor. It is context sensitive by default and can be opened by right-clicking on an empty location in the Graph Editor.



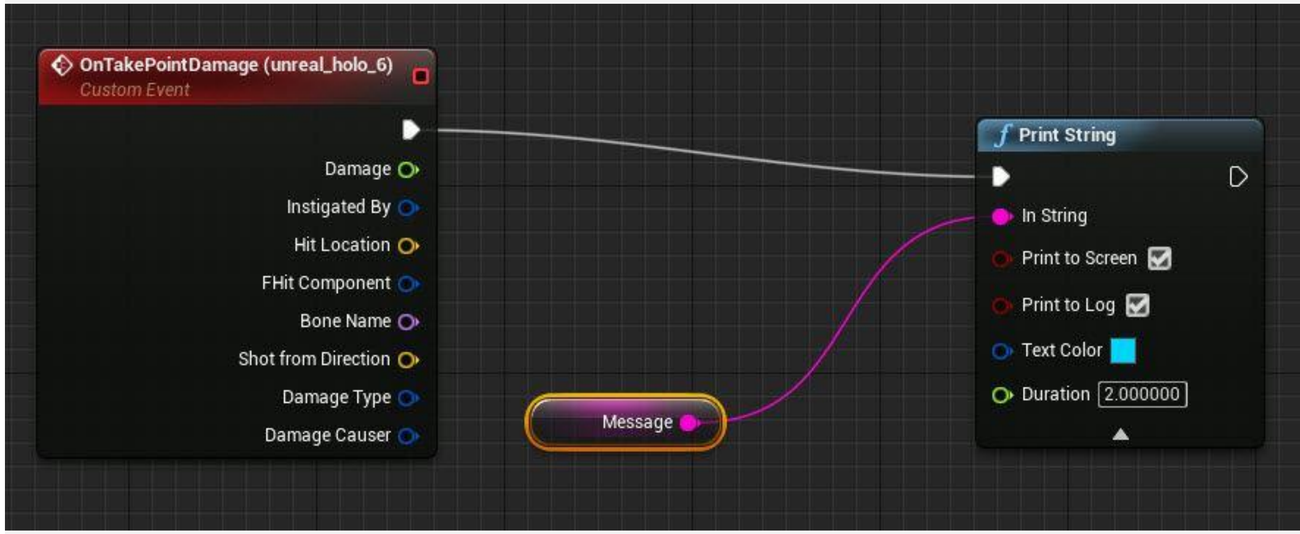
GRAPH EDITOR

Nodes, Pins, Execs, and Wires

NODES

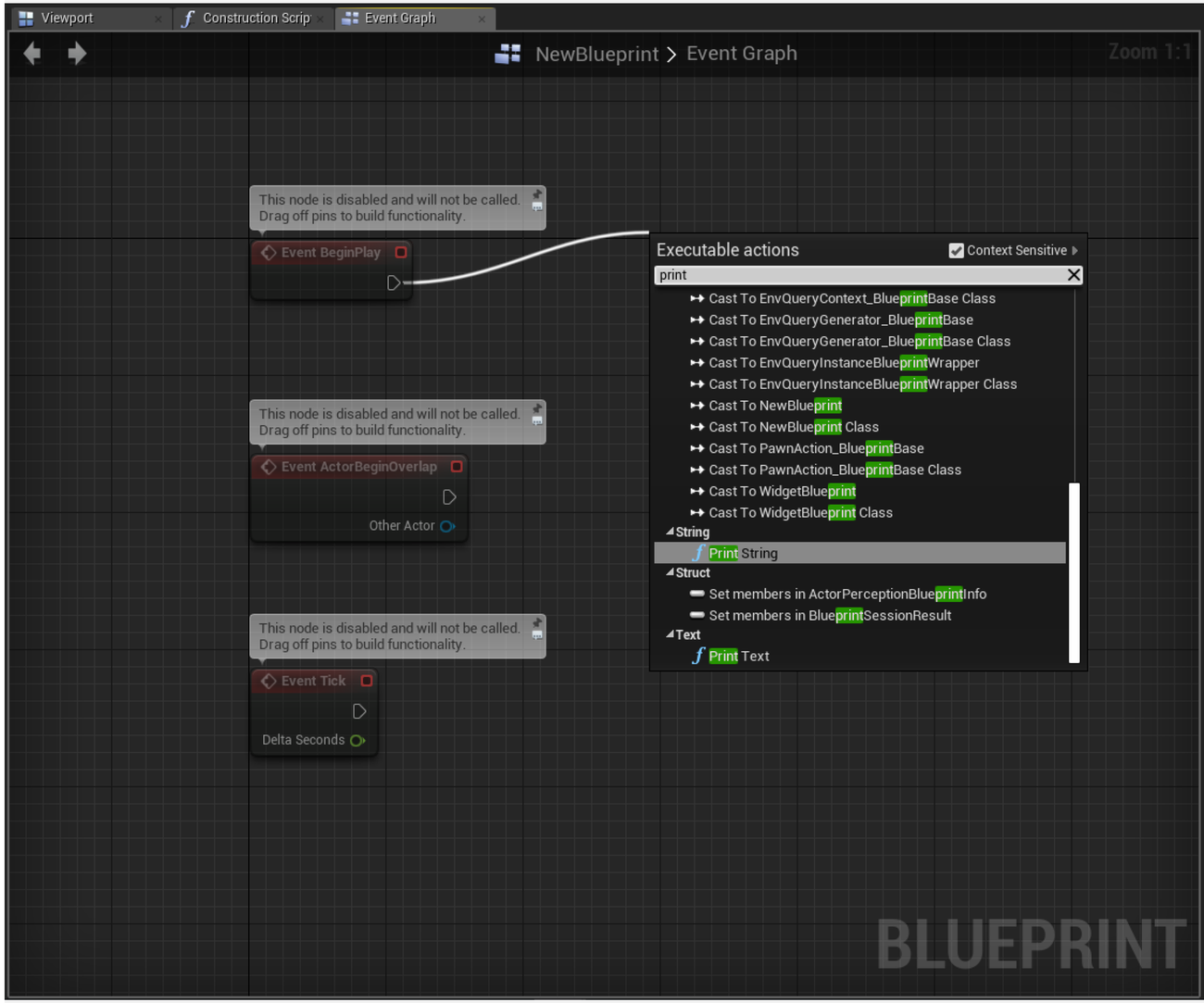
Nodes are objects such as events, function calls, flow control operations, and variables that can be used in graphs to define the functionality of the particular graphs and Blueprints that contain them.

Each type of node performs a unique function; however, the way in which nodes are created and used is common to all nodes. This helps make for an intuitive experience when creating node graphs.



PLACING NODES

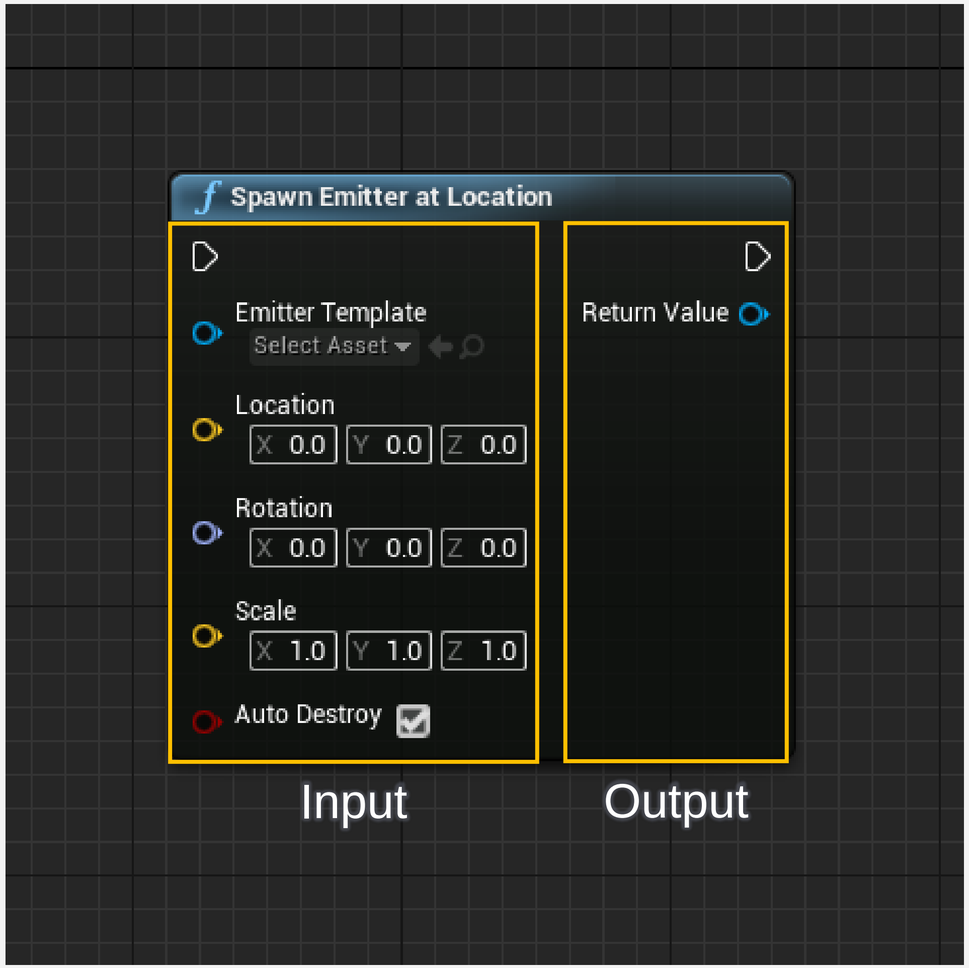
New nodes are added to a graph by selecting the type of node from the Blueprint Context Menu. The node types listed in the context menu depend on how the list is accessed and what is currently selected.



PINS

Nodes can have pins on either side. Pins on the left are input pins, and those on the right are output pins.

There are two main types of pins: **execution pins** and **data pins**.

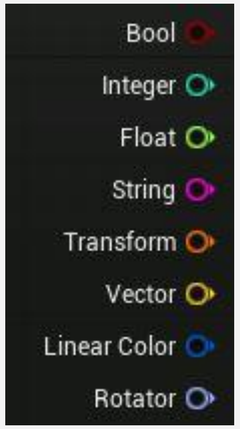


EXECUTION PINS

Execution pins are used to connect nodes together to create a flow of execution. When an input execution pin is activated, the node is executed. Once execution of the node completes, it activates an output execution pin to continue the flow of execution.

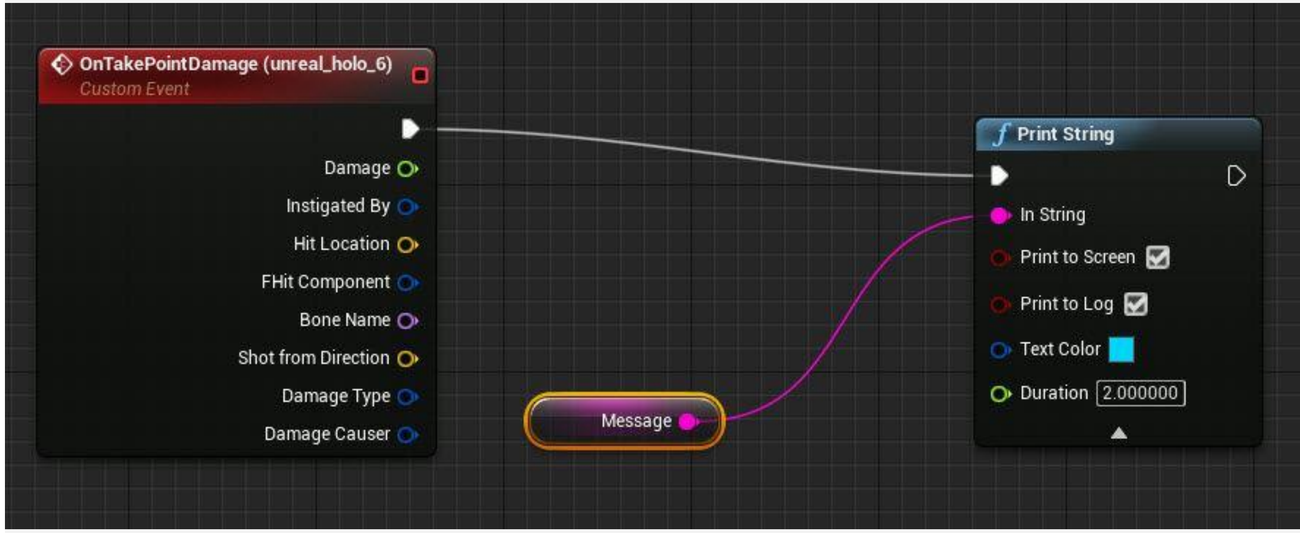
DATA PINS

Data pins are used to take data into a node or to output data from a node. Data pins are type-specific and can be wired to variables of the same type (which have data pins of their own) or a data pin of the same type on another node. Like execution pins, data pins are displayed as an outline when not wired to anything, and solid when wired.



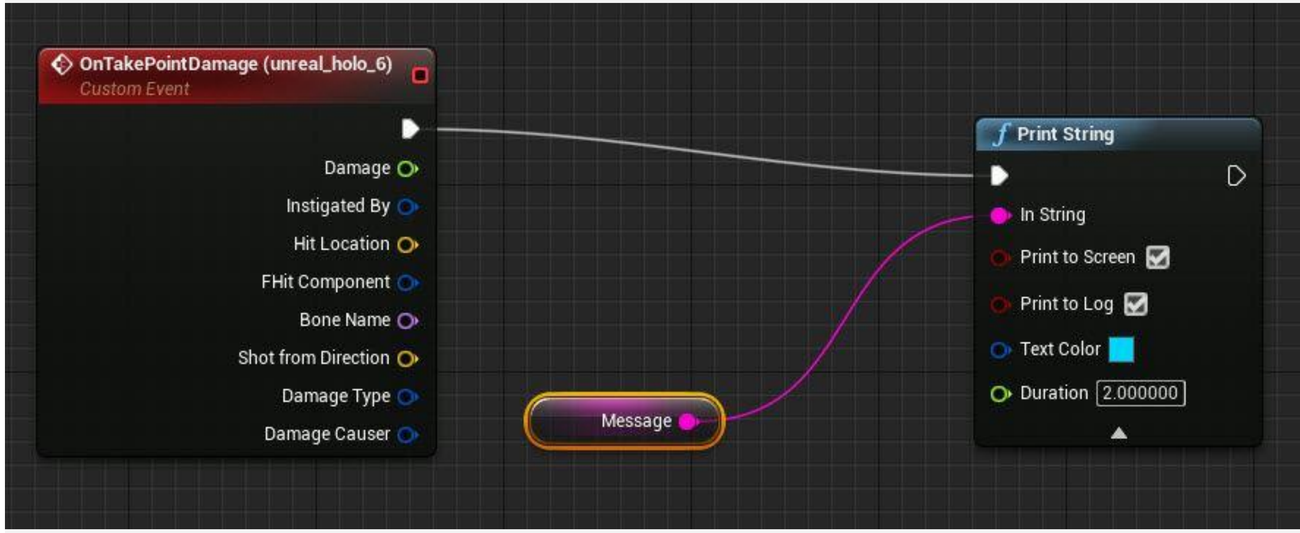
WIRES

Connections between pins are called wires. Wires can represent either the flow of execution or the flow of data.



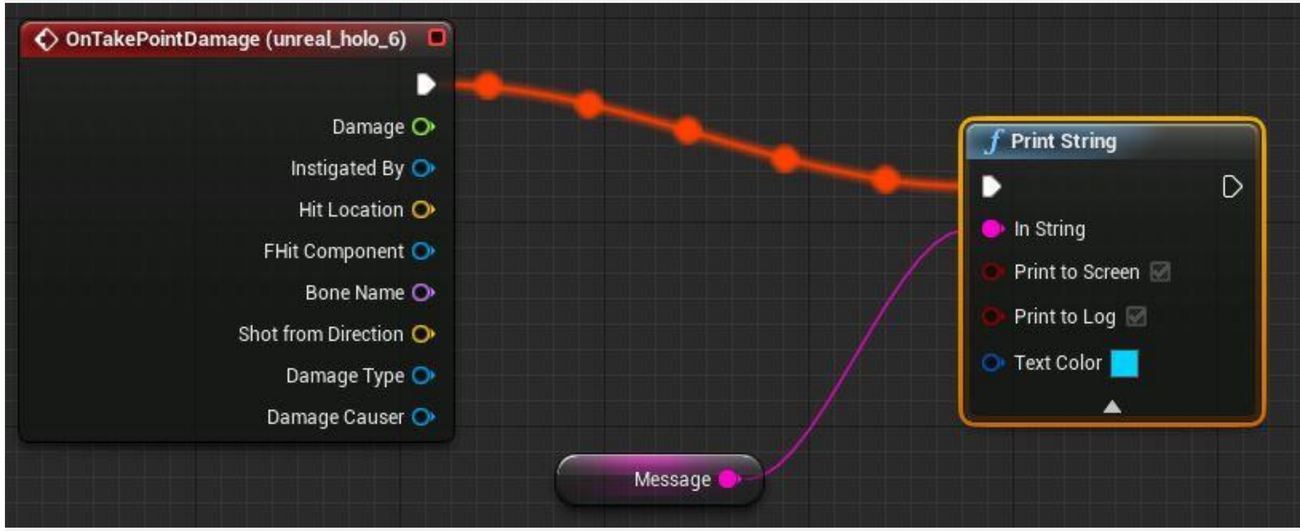
EXECUTION WIRES

Wires between exec pins represent the flow of execution. Exec wires are displayed as white arrows spanning from an output exec pin to an input exec pin. The direction of the arrow indicates the flow of execution.



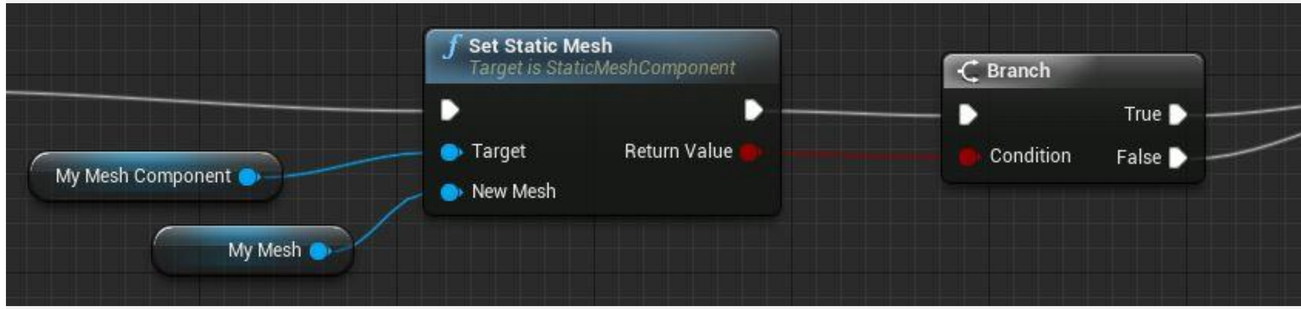
EXECUTION WIRES

Exec wires produce a visual indicator when being executed. During play, as one node completes execution and the next is activated, the wire between their execution pins highlights the fact that execution is being transferred from one node to another.



DATA WIRES

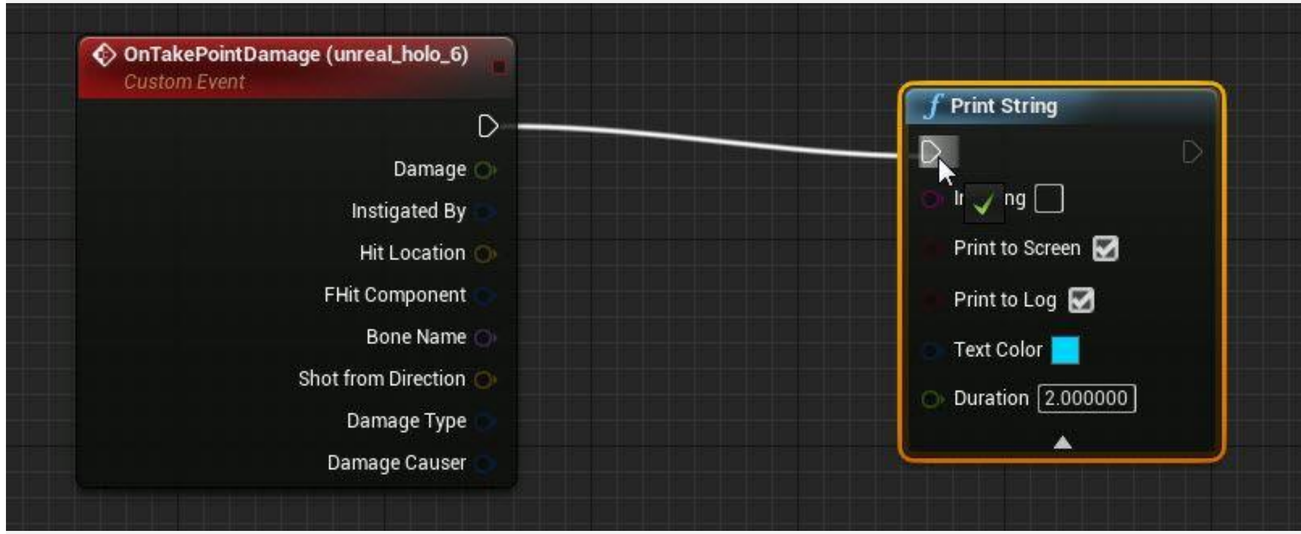
Data wires connect one data pin to another data pin of the same type. They are displayed as colored arrows and are used to visualize the transfer of data, with the direction of the arrow representing the direction the data is traveling. The color of a data wire is dependent on the type of data, just as the color of the data pins are.

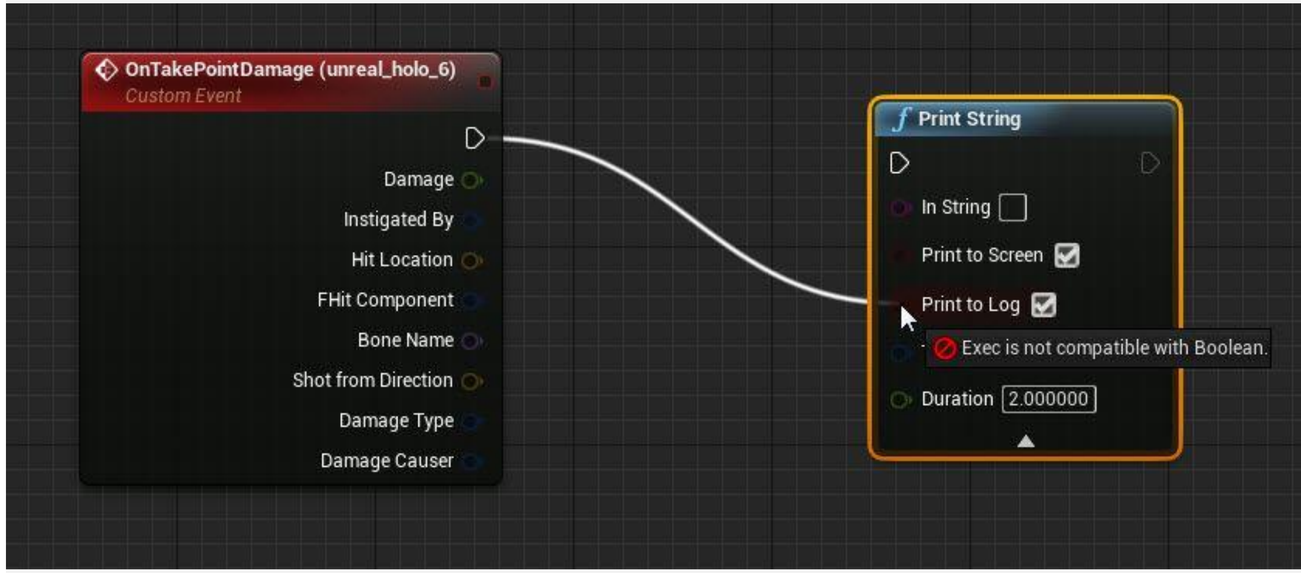


WORKING WITH WIRES

Wires are created in the Graph Editor panel using one of two methods.

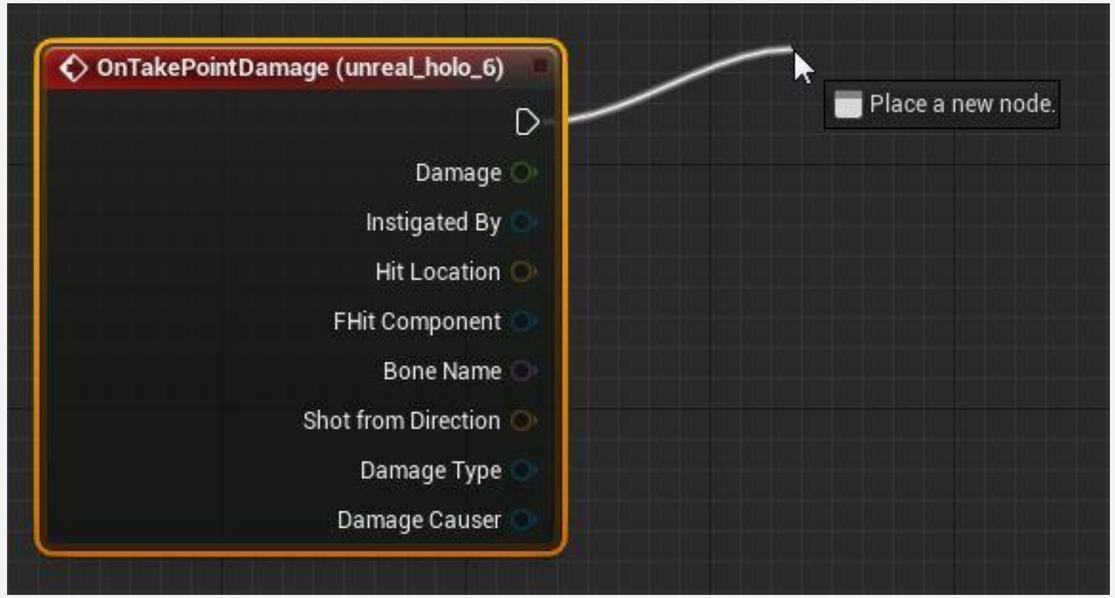
- One method is to click on a pin and drag and release on another pin of the same type to create a direct connection.
- A connection can only be made between two compatible types of pins. If you click on one pin and drag and release on a pin that is not compatible, an error will be displayed informing you the connection cannot be made.

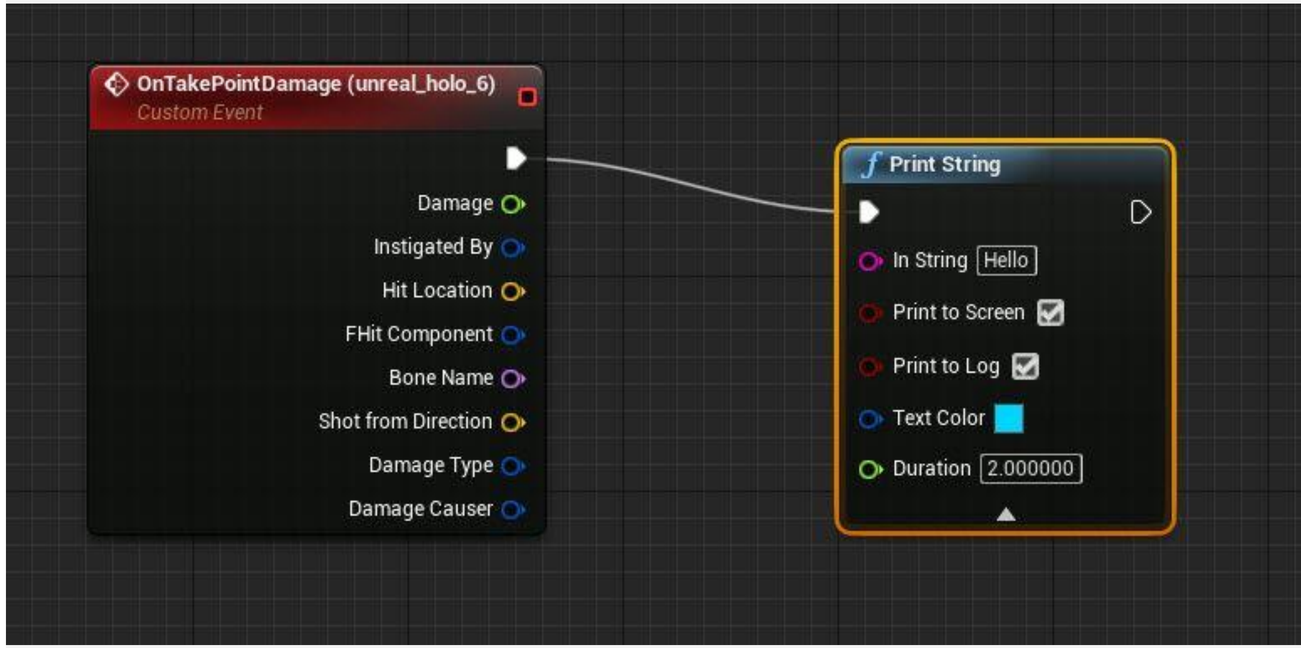




WORKING WITH WIRES

- Another way to add a wire is to click on a pin and drag and release in an empty space to summon a context-sensitive menu that lists all the nodes that are compatible with the type of pin the connection is originating from.
- Selecting a node from this list creates a new instance of that node and makes a connection to a compatible pin on the new node.

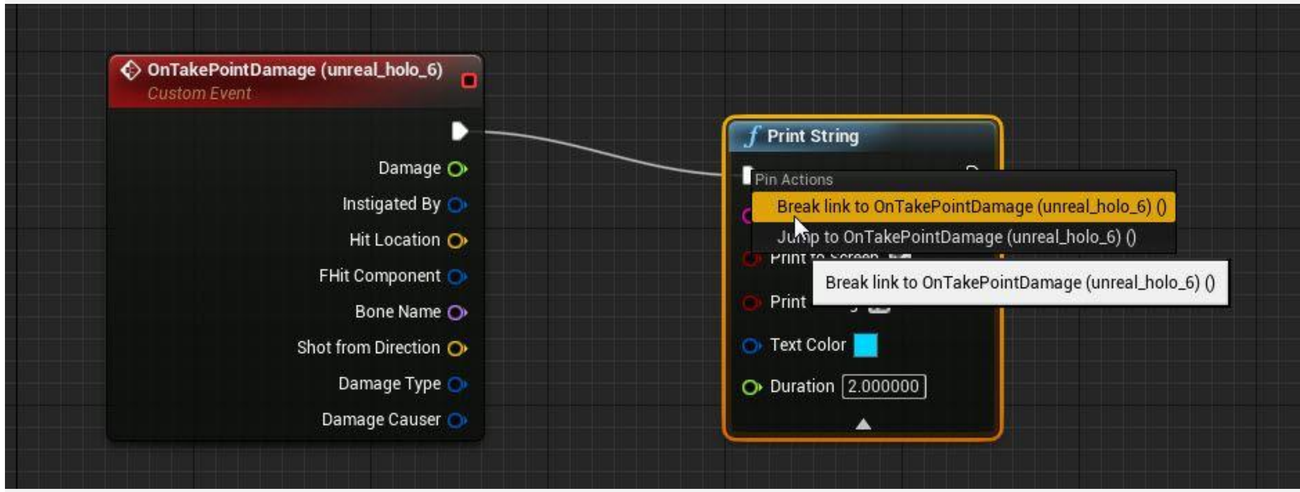




WORKING WITH WIRES

A wire between two pins can be broken in one of the following ways :

- Alt+click on one of the connected pins.
- Right-click on one of the connected pins and choose “Break link to . . .”

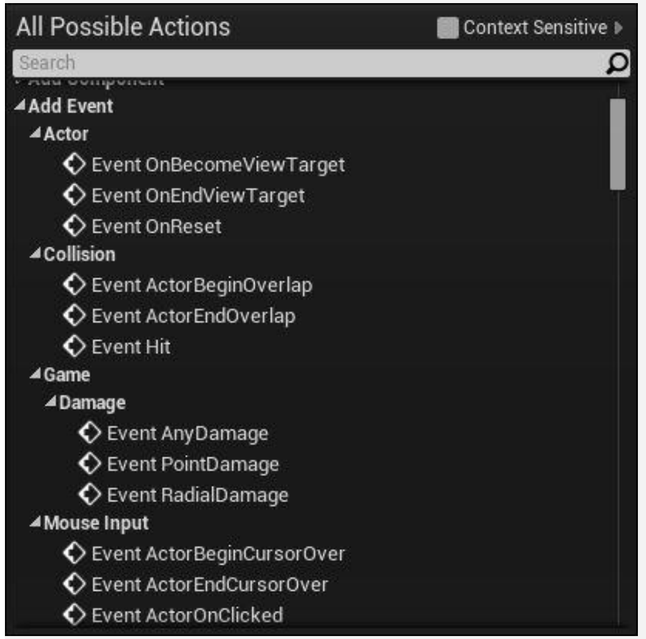


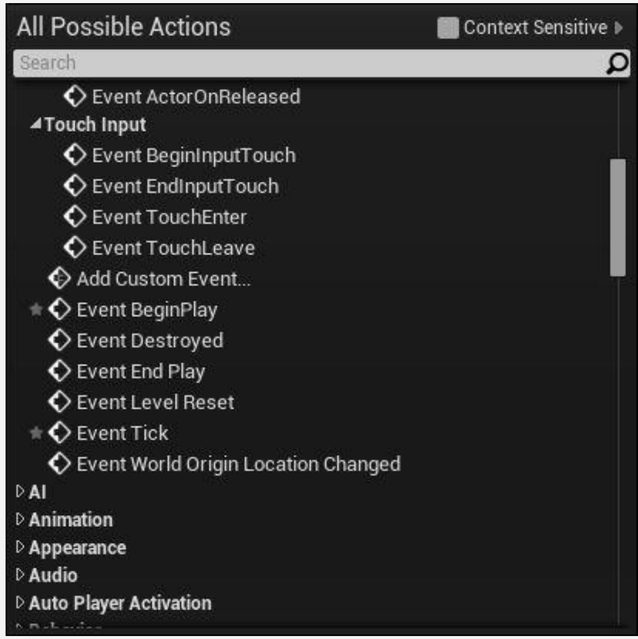
VISUAL SCRIPTING

Events, Function Calls, Variables,
Operations, and Conditionals

EVENTS

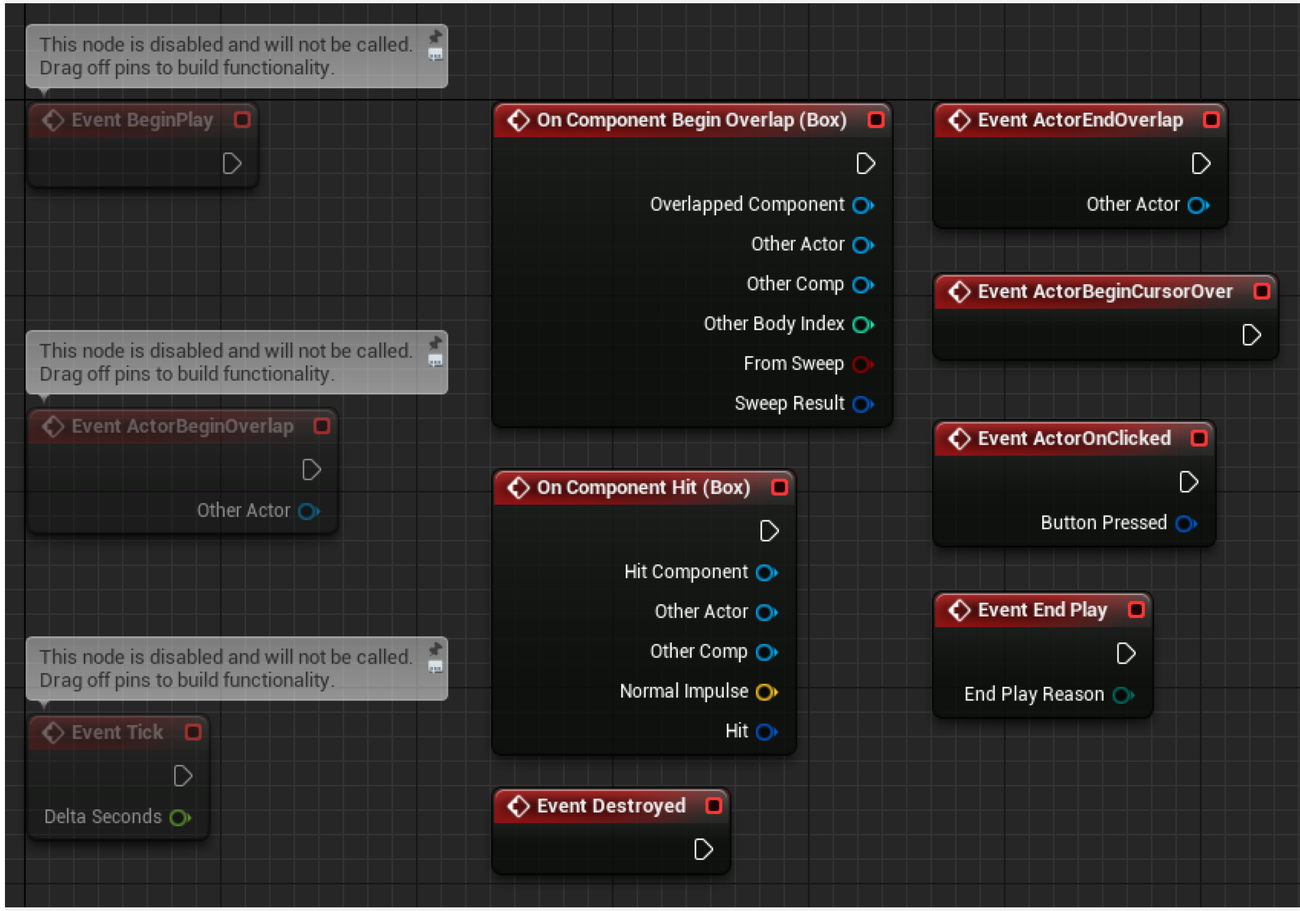
In its simplest form, an event is something that happens during gameplay. It can be two objects colliding, the player Pawn entering a specified location, a Level restarting, or the mouse curser hovering over an Actor. There are many different events that can happen during gameplay.





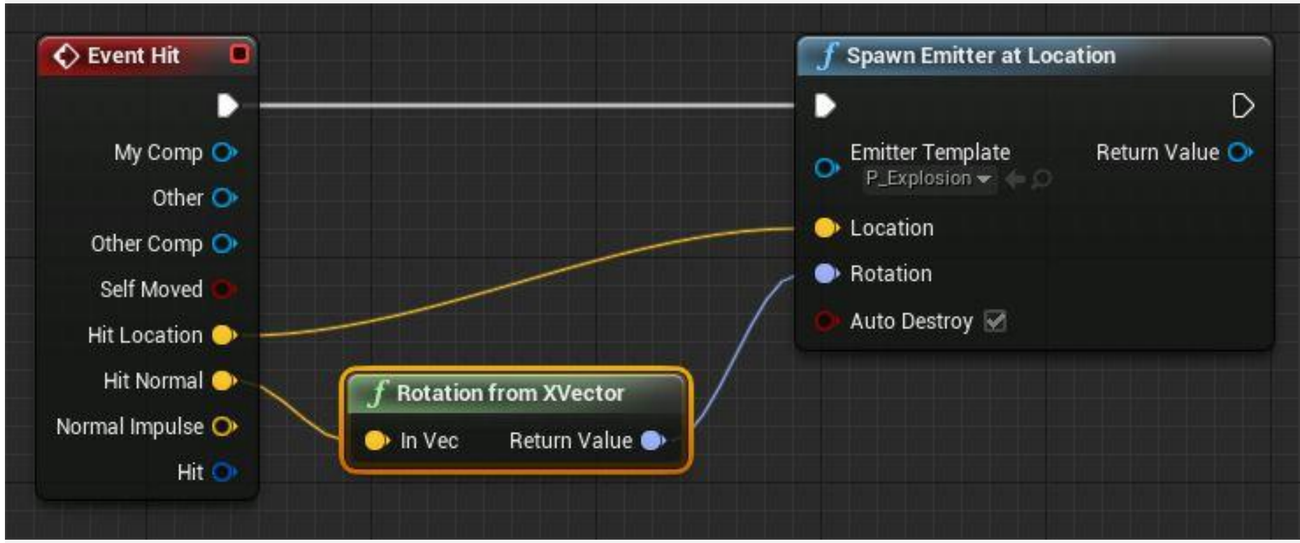
EVENT NODES

Events are nodes that are called from gameplay code to begin execution of an individual network within the Event Graph.



EVENTS

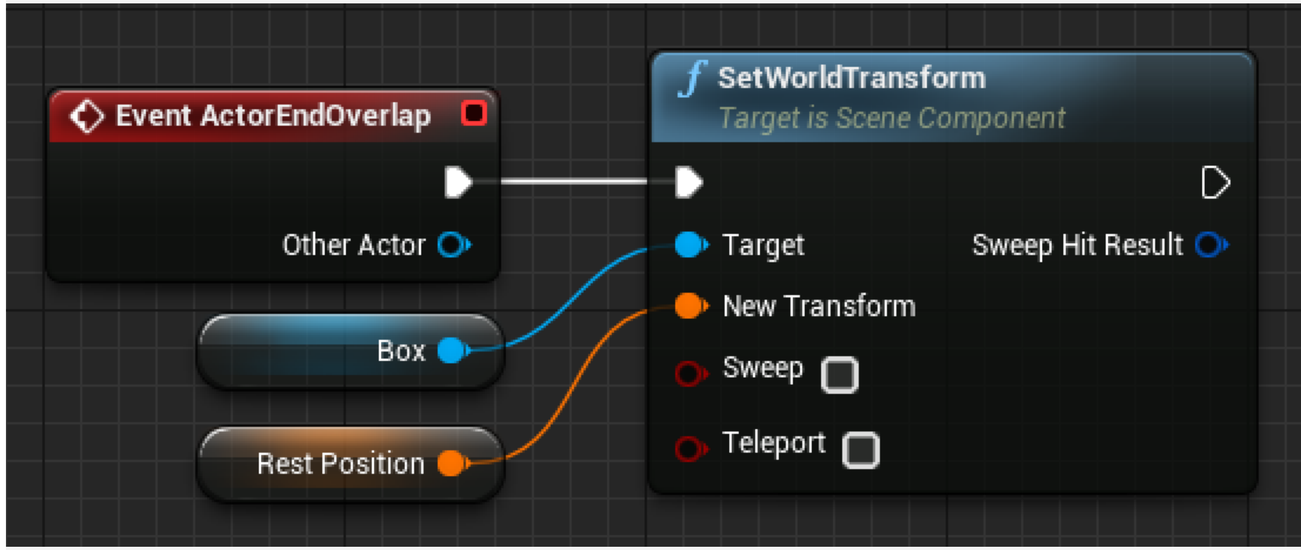
Events can be accessed within Blueprints in order to implement new functionality or override or augment the default functionality. Any number of events can be used within a single Event Graph, though each one must have a unique name.



FUNCTION CALLS

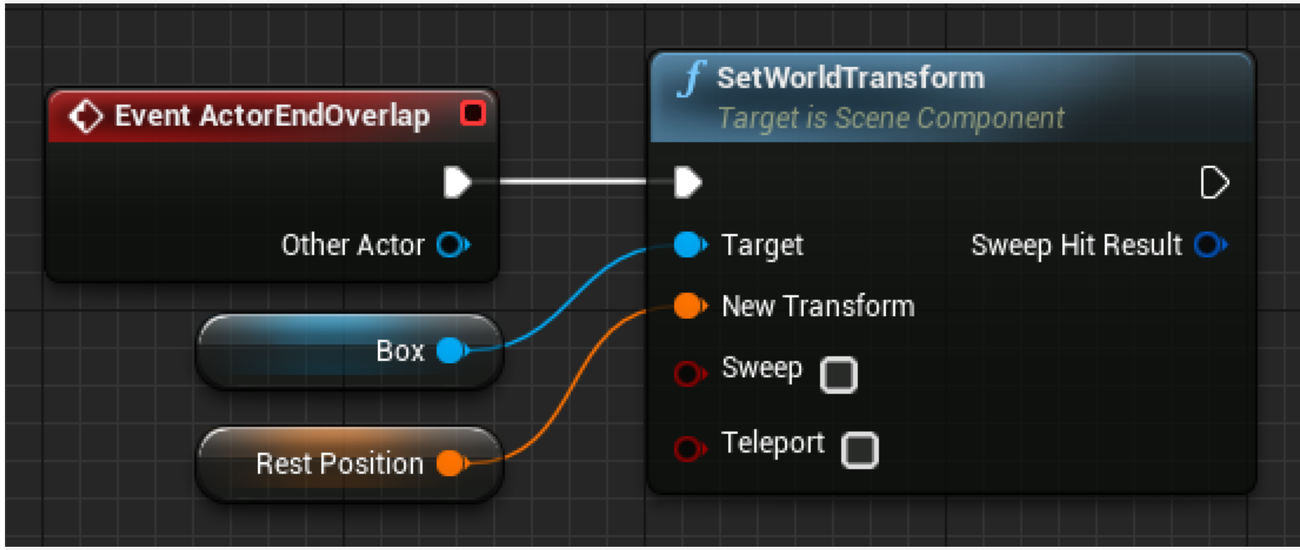
Function calls perform specific operations. They take in data stored in variables, process the information, and in most cases return a result or perform an action on a “target.”

In the example on the right, when the event fires, the SetWorldTransform function call will change the transform of the Static Mesh component called Box.



FUNCTION CALLS

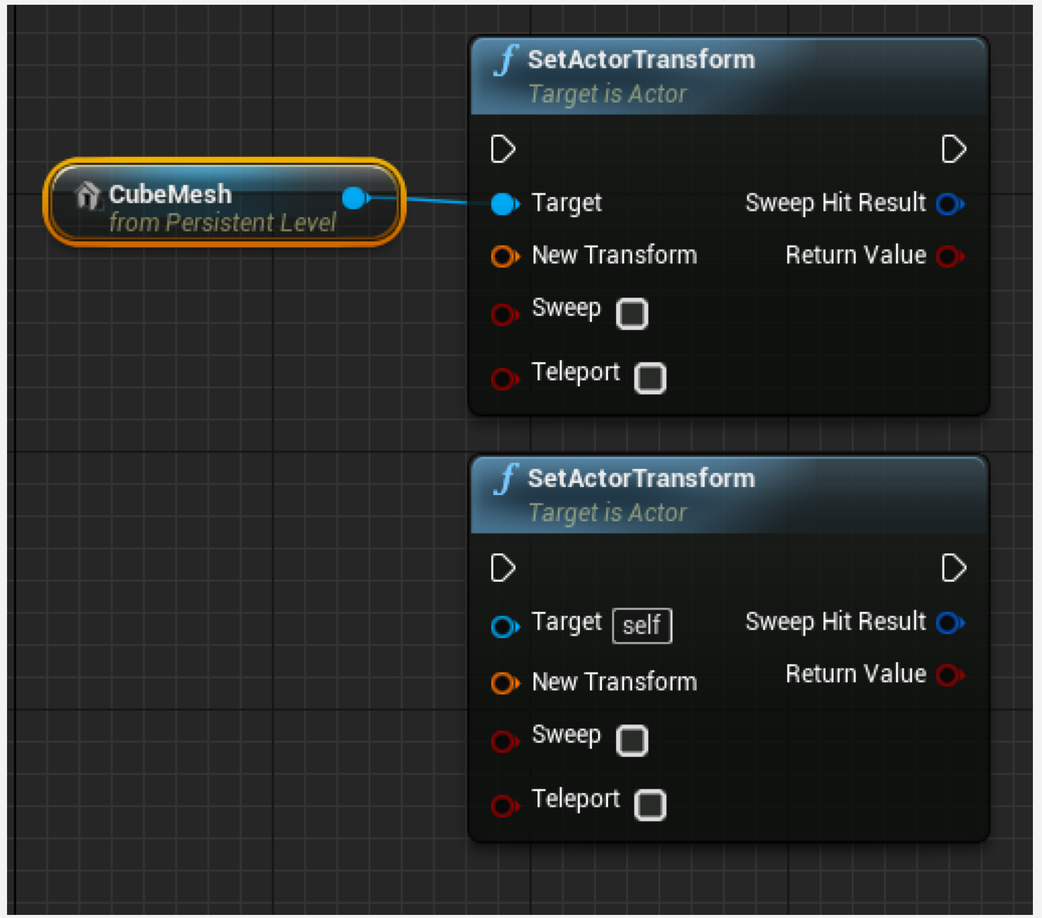
When a function is placed in the Event Graph, you typically see a target data pin on the left side of the function’s node.



FUNCTION CALLS

A target is typically a variable that stores a reference to an Actor or a component of an Actor on which the function operations will be performed.

If a target has not been assigned, you will see “self”, which refers to the Blueprint class the function call is in.



VARIABLES

Variables store different types of data. When a variable is created, the computer sets aside a certain amount of memory, depending on the data type. That memory is then used to store or retrieve the information at that memory location. Different variable types use different amounts of memory.

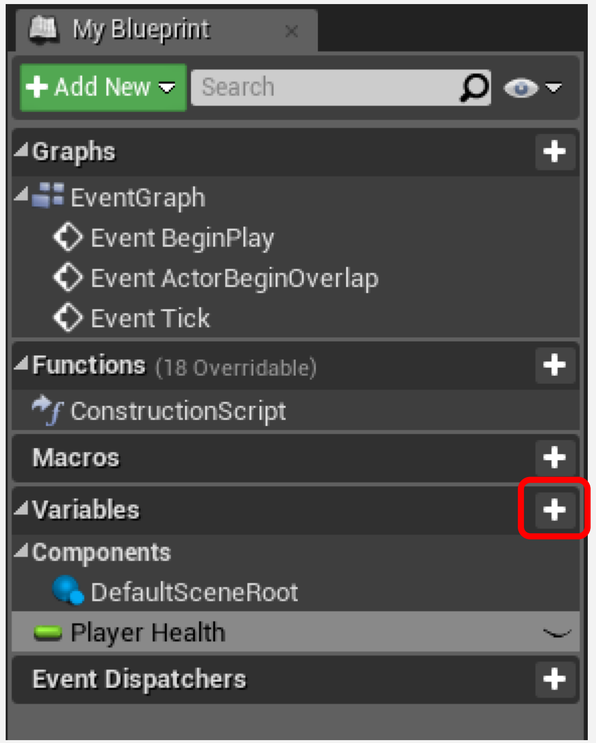
VARIABLES

In the Blueprint Editor, variables are color-coded so that you can quickly identify what variable type you are working with.

Variable Type	Color	Description
Boolean	Red	Red variables represent Boolean (true/false) data.
Integer	Cyan	Cyan variables represent integer data, or numbers without decimals, such as 0, 152, and –226.
Float	Green	Green variables represent float data, or numbers with decimals, such as 0.0553, 101.2887, and –78.322.
String	Magenta	Magenta variables represent string data, or a group of alphanumeric characters, such as “Hello World”.
Text	Pink	Pink variables represent displayed text data, especially where that text is localization aware.
Vector	Gold	Gold variables represent vector data, or numbers consisting of float numbers for three axes or elements, such as *xyz* or RGB information.
Rotator	Purple	Purple variables represent rotator data, which is a group of numbers that define rotation in 3D space.
Transform	Orange	Orange variables represent transform data, which combines translation (3D position), rotation, and scale.
Object	Blue	Blue variables represent Object references, including Lights, Actors, Static Meshes, Cameras, and Sound Cues.

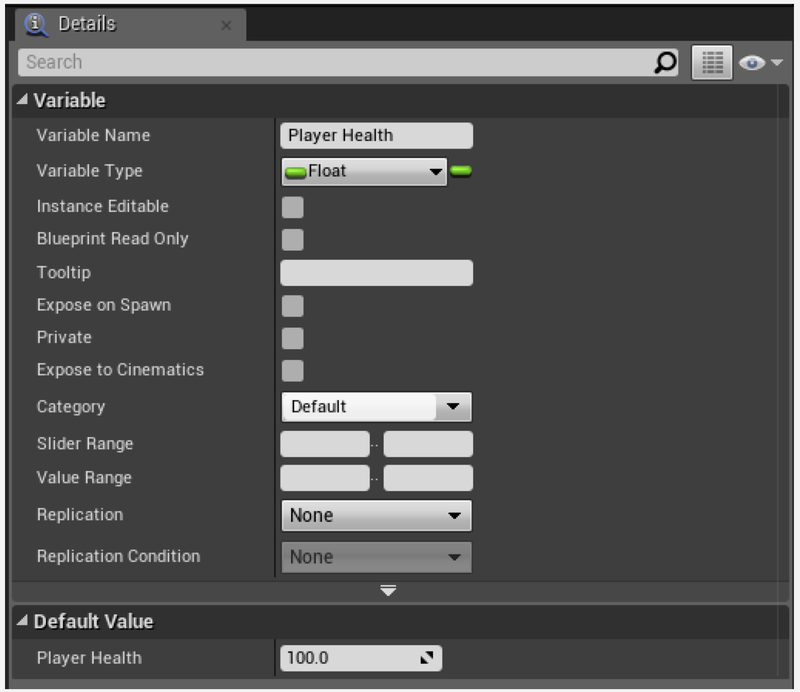
VARIABLES

Variables are declared in the My Blueprint panel under the Variables tab. Click the plus symbol on the right side of the tab.



VARIABLES

Once created, variables can be given an initial value in the Blueprint Details panel. You will need to compile the Blueprint at least once to assign the value.



VARIABLES

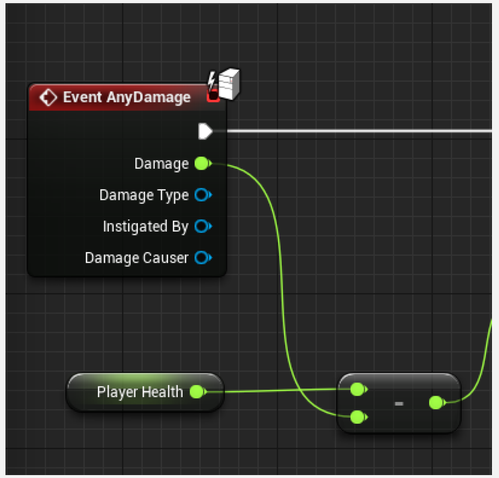
In the Graph Editor, variables can be assigned a value using a Set node, or their data can be retrieved using a Get node.

OPERATORS

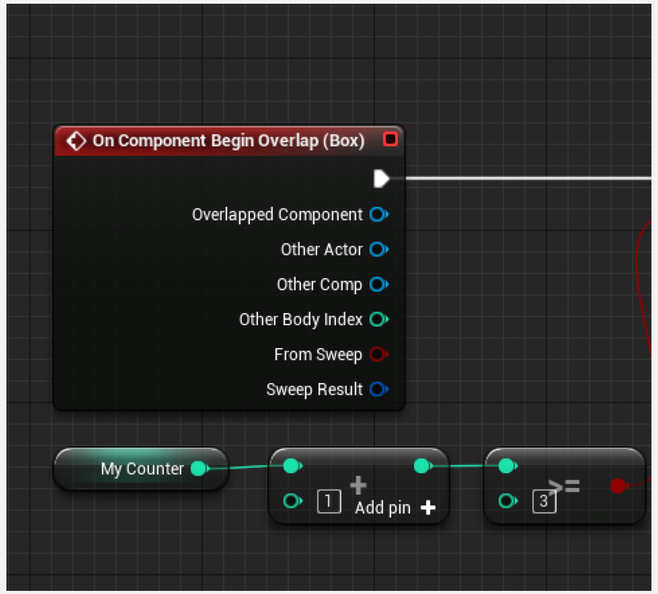
Variables are only good if you can change their states and compare them. Operators are nothing more than mathematical operations such as addition, subtraction, division, and multiplication.

Operators are found under the Math heading in the context menu and are separated by variable type.

Float subtraction operator



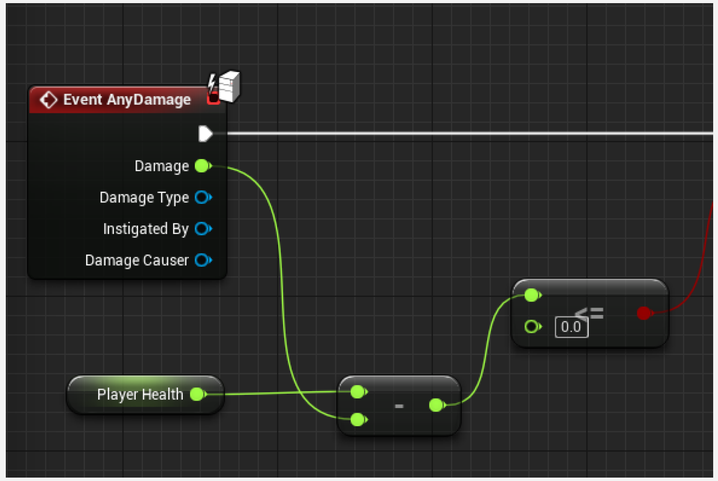
Integer addition operator

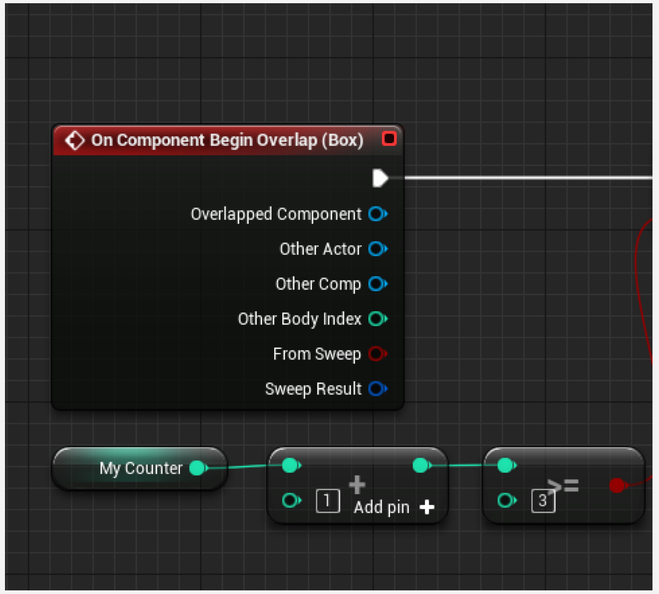


CONDITIONALS

Conditionals such a >, <, and = allow you to compare variables. Depending on the result of the comparison, you can perform different function calls.

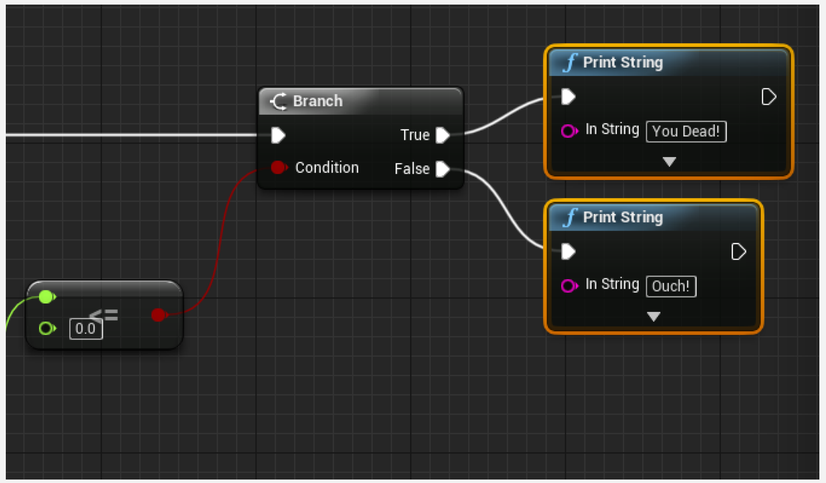
Conditionals are also found under the Math heading in the context menu and separated by variable type.

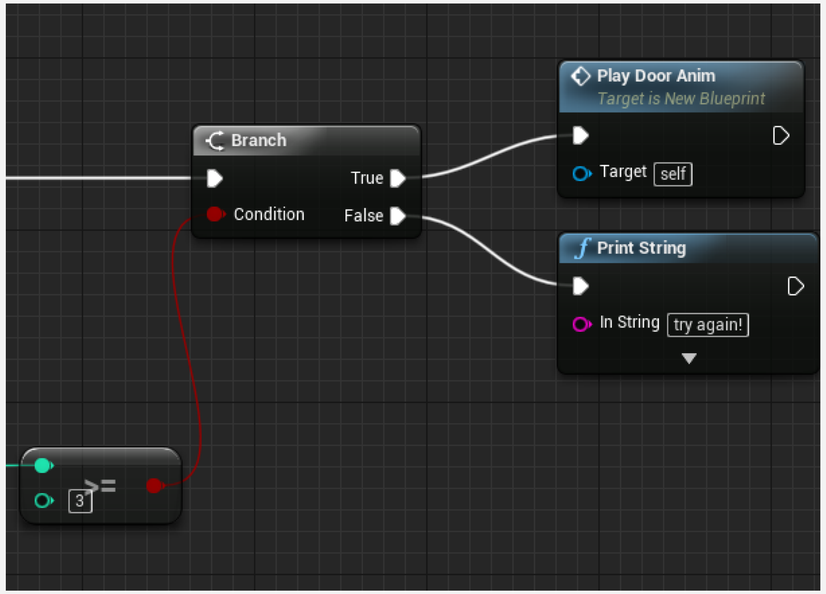




FLOW CONTROLS

Flow control operations allow the flow of execution to be controlled explicitly in Blueprints. This control can be in the form of choosing one branch of the graph to execute based on whether some condition is true, executing a specific branch multiple times, executing multiple branches in a specific order, and so on.





FLOW CONTROLS

The default flow control operations include branches (if statements), loops (for and while), gates, and sequences.