Conditional Statements
Chapter 03 - Lesson 01

Chapter Introduction

Objectives

- Apply conditional statements to programming logic.
- Apply a control structure to adjust program flow.
- Read variables of type Boolean change program flow based on them.

CTSA Standards

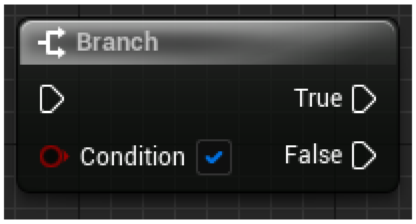
1B-AP-10	Create programs that include sequences, events, loops, and conditionals. Control structures specify the order (sequence) in which instructions are executed within a program and can be combined to support the creation of more complex programs. Events allow portions of a program to run based on a specific action. For example, students could write a program to explain the water cycle and when a specific component is clicked (event), the program would show information about that part of the water cycle. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. For example, students could write a math game that asks multiplication fact questions and then uses a conditional to check whether or not the answer that was entered is correct. Loops allow for the repetition of a sequence of code multiple times. For example, in a program that produces an animation about a famous historical character, students could use a loop to have the character walk across the screen as they introduce themselves.

Unreal Engine Documentation

See: https://docs.unrealengine.com/4.27/en-US/BlueprintAPI/Utilities/FlowControl/Branch/ for the official documentation on the branch node in UE.

Conditional statements: Branch node

Conditional Statements are a way of controlling the flow of a piece of code. Flow is referring to running code if a conditional is true, or other code if the condition is false. In many computer languages, this functionality is found in the “if” statement. In UE, it is achieved with the “Branch” node.



Branch/If note: The Branch instruction in UE is equivalent to the “if” instruction in other programming languages. In fact, if you are searching for a branch node, you can type “if” instead and the search will find the branch node.

Observe that the Branch node has a single execution wire leading into it from the left and two possible outputs based on the value of the Condition.

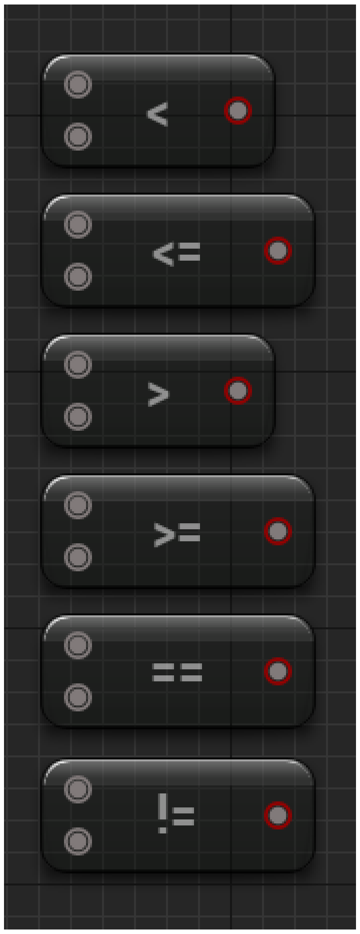
If the Condition is True, the flow of the program follows the path of the True execution pin, and if the condition is False, the program follows the path of the False pin.

It’s very rare that something isn’t plugged into the Condition of a Branch node. With that said, sometimes it’s useful while debugging or testing code to “hardcode” the value of the Condition.

Definition: **Hardcode** means to provide a fixed value that is known and not based on a variable or variable condition.

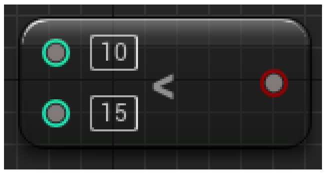
The Condition is a Boolean value (also called a “Bool”), meaning it must have a value of **True** or **False**.

The Condition can be the result of a conditional operator, such as <,<=, >, >=, ==, !=. These nodes look like this:



Each of these conditional nodes accept two arguments of the same type (i.e. integers, floats, etc…). They compare one to the other and the result is either True or False.

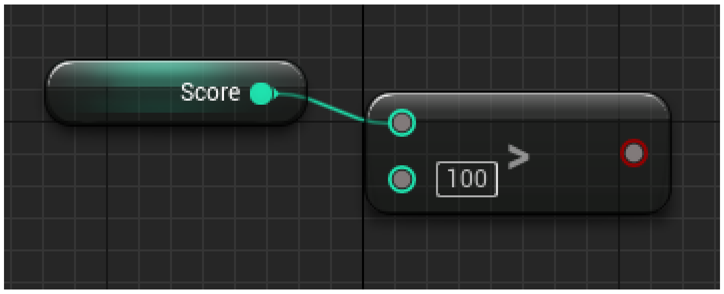
Take this code as an example:



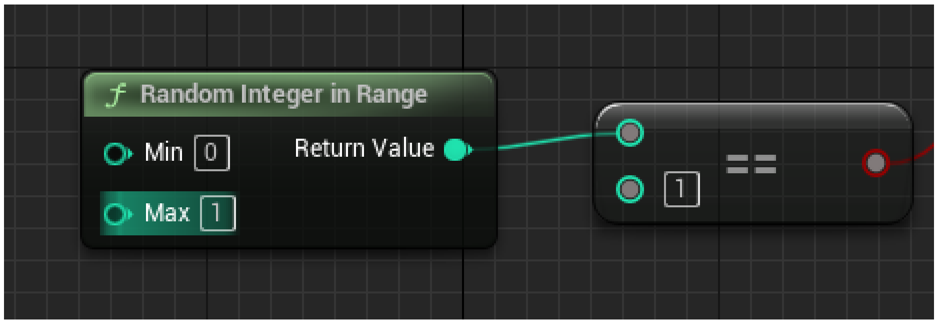
This code is checking if the value of 10 is less than 15. In a language such as Java, this would be written as **10 < 15**. Note that in the node above, the values are hardcoded.

Obviously the boolean result of that comparison is True; 10 is less than 15.

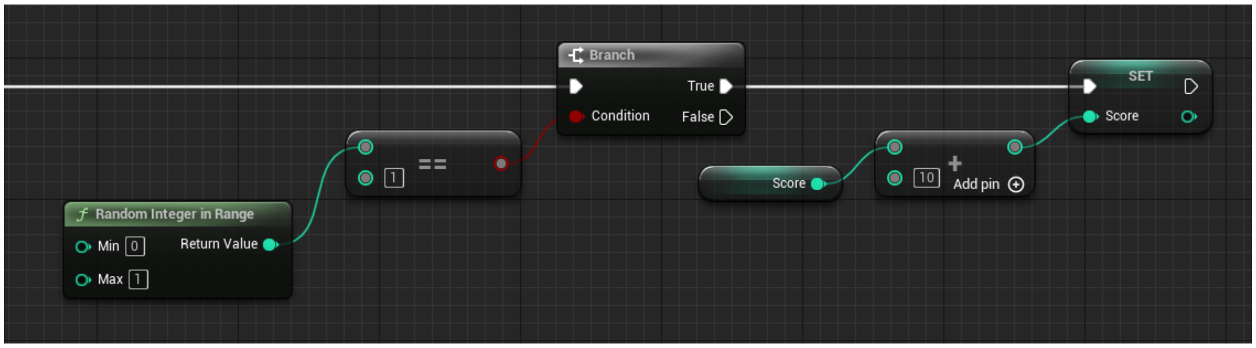
Variables are usually used to make these checks more useful. For example, a variable called Score could be checked to see if the player should be awarded an extra life:



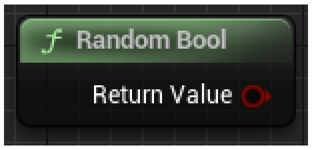
Or, a random number could be checked to see if the player should be awarded some points. This code will result in a True value approximately 50% of the time:



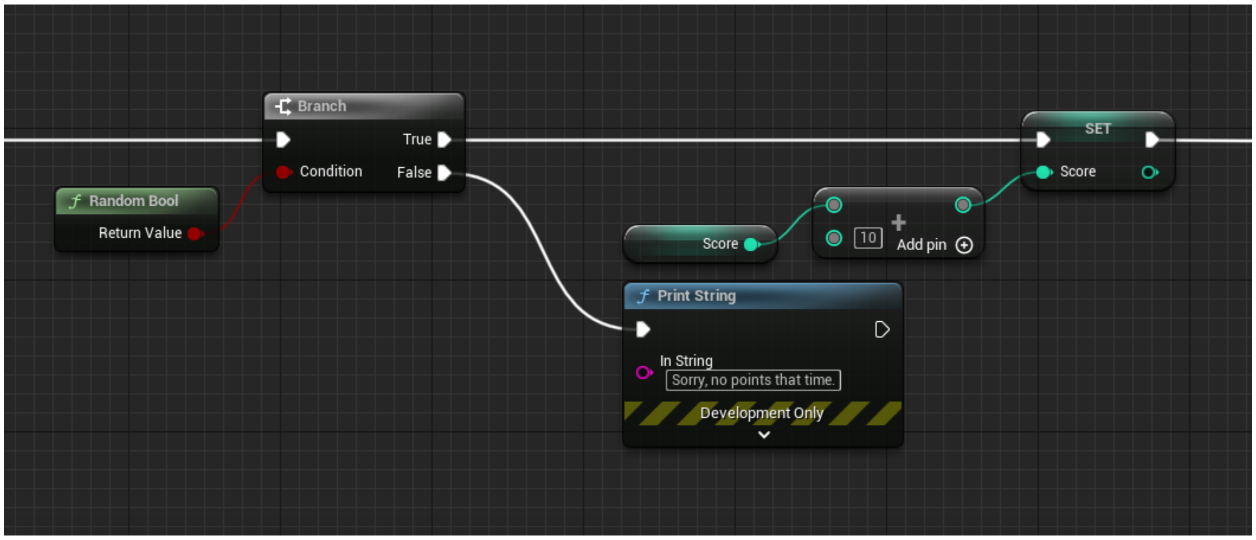
This code will award the player 10 points approximately 50% of the time:



In the case where a random True value is needed 50% of the time, the Random Bool node can be utilized instead:



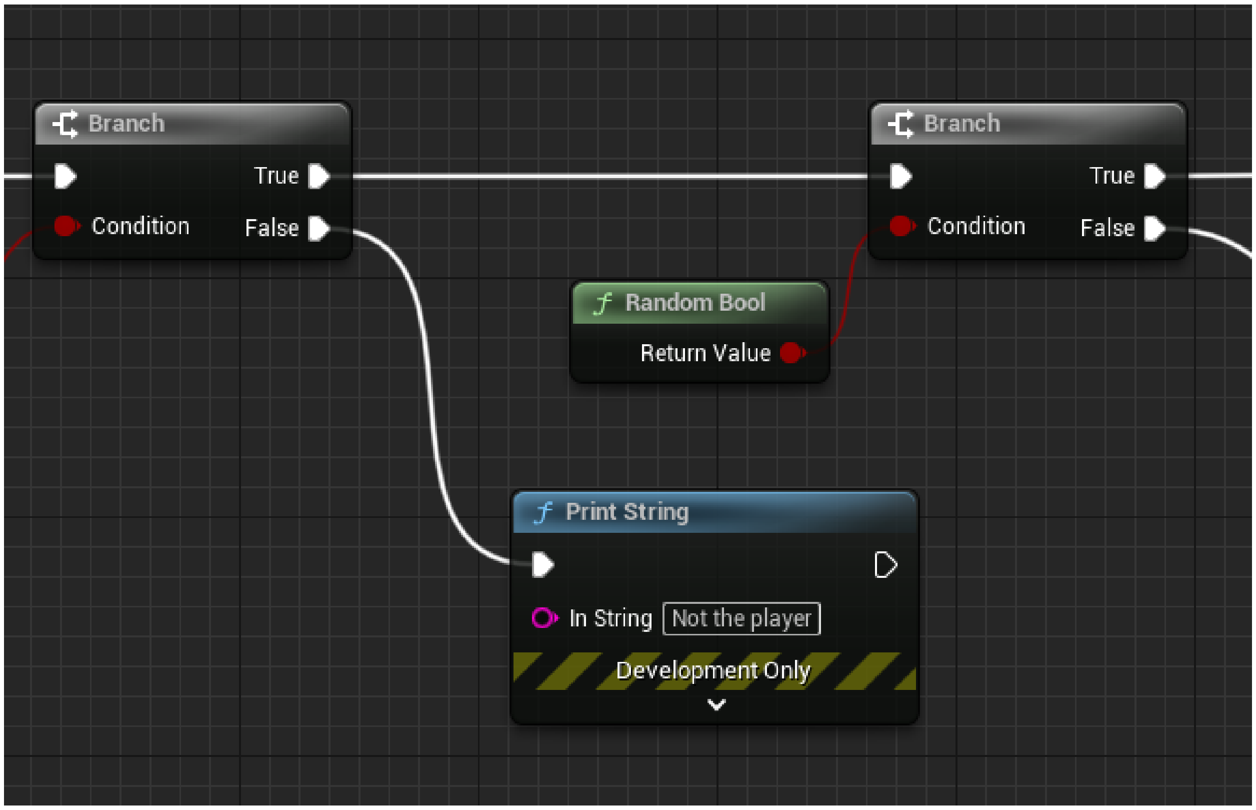
The code below uses a Random Bool and also accounts for the case where the value may be False.



Conditional statements: Nested Branch node

Oftentimes additional branch statements are used to evaluate multiple conditions. Also, please note, a later topic will cover using combinational logic (AND/OR) to evaluate multiple conditions at the same time. There are reasons to use one method over the other and they will be covered later as well.

A nest branch (also called a nested if) would look like this:

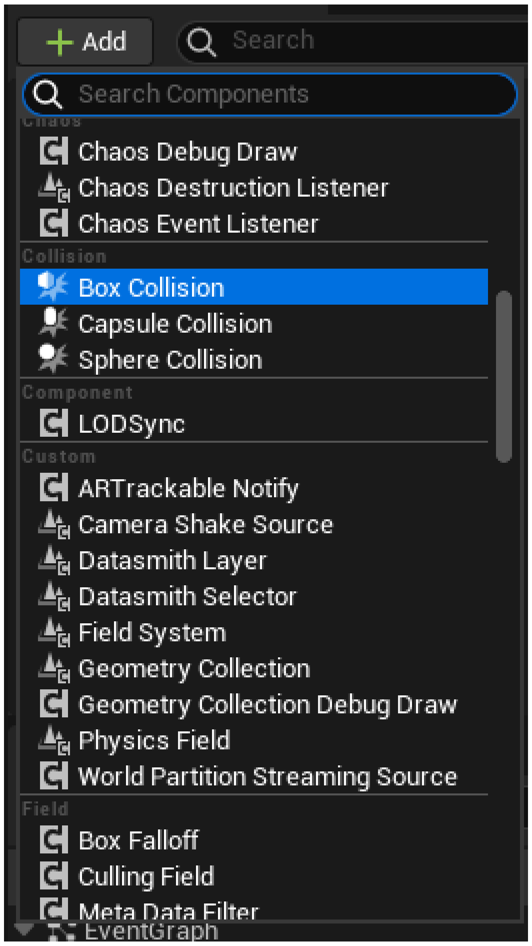


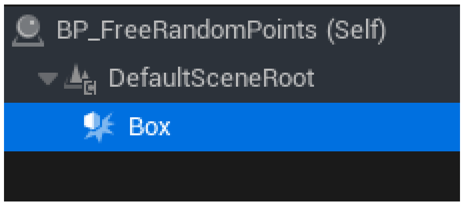
The branch following the True branch is said to be nested in the previous branch instruction. It will only run if the previous branch is True.

Exercise

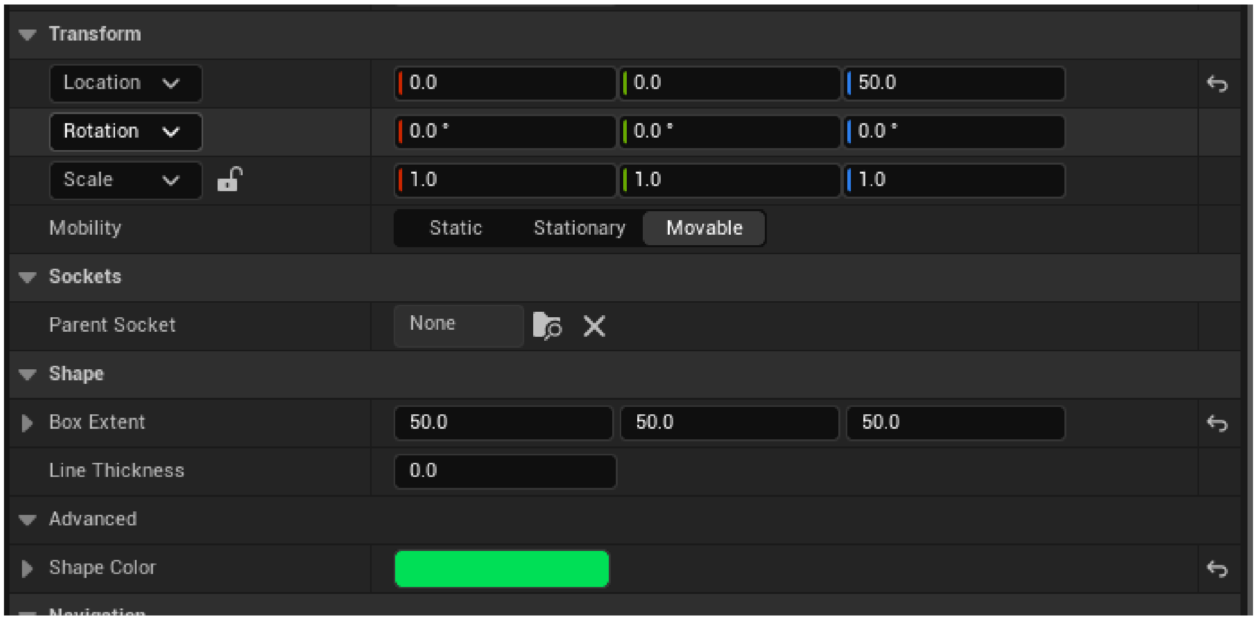
In Unreal Engine, it's often desirable to determine what kind of actor collided or overlapped another actor. A conditional statement can be used to determine this.

1. In a Third Person Project, create a new Blueprint of type actor called BP_FreeRandomPoints.
2. Add a Box Collision component.





1. Change the Z value of the Location to 50 and the Box Extent to 50, 50, 50. Change the Shape Color to a color of your choosing.


Transform

Location	0.0	0.0	50.0	
Rotation	0.0 °	0.0 °	0.0 °	
Scale	1.0	1.0	1.0	
Mobility	Static	Stationary	[✓ Movable]	

Sockets

Parent Socket	None

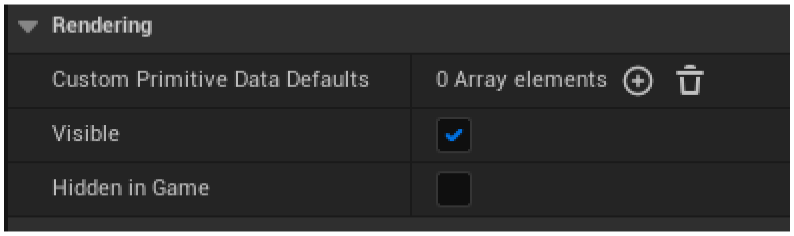
Shape

Box Extent	50.0	50.0	50.0
Line Thickness	0.0		

Advanced

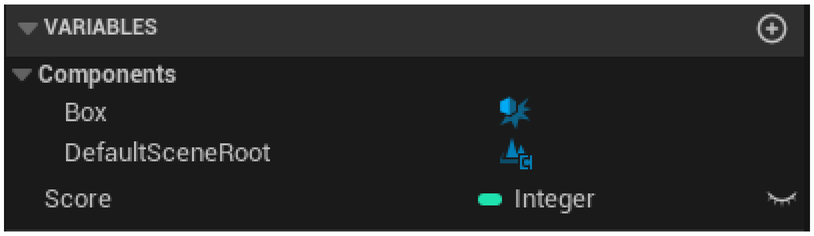
Shape Color	🟩

1. Untick the Hidden in Game attribute.


Rendering

Custom Primitive Data Defaults	0 Array elements
Visible	✓
Hidden in Game	☐

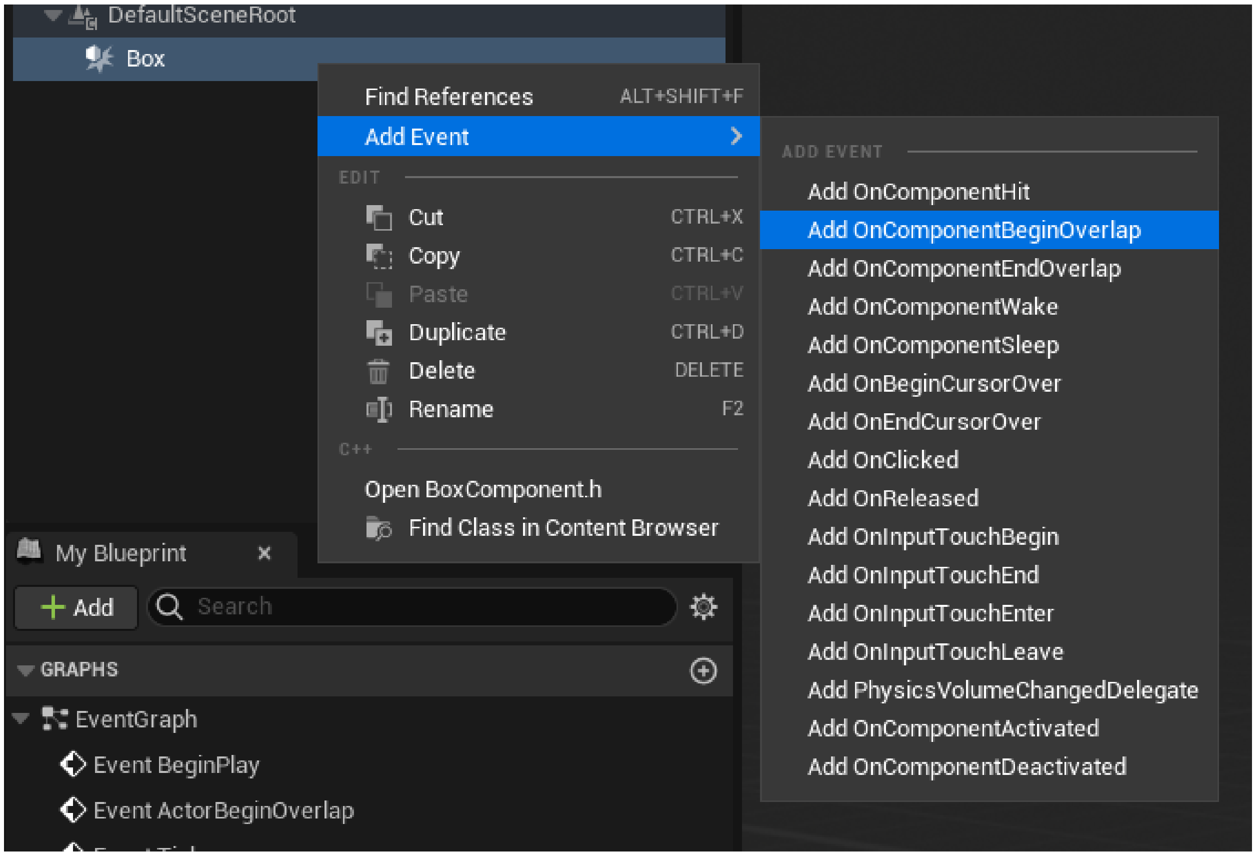
1. Add a variable of type Integer called Score.


VARIABLES

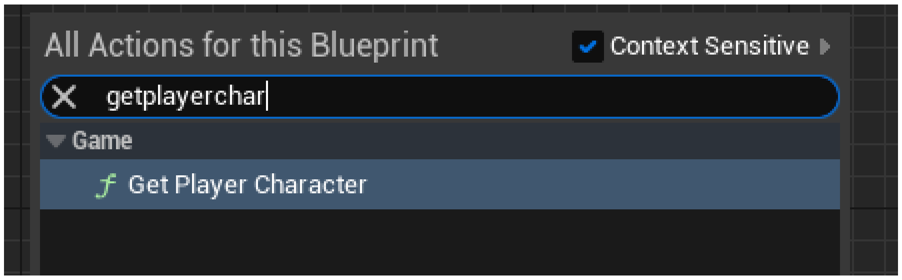
Components

Box	
DefaultSceneRoot	
Score	Integer

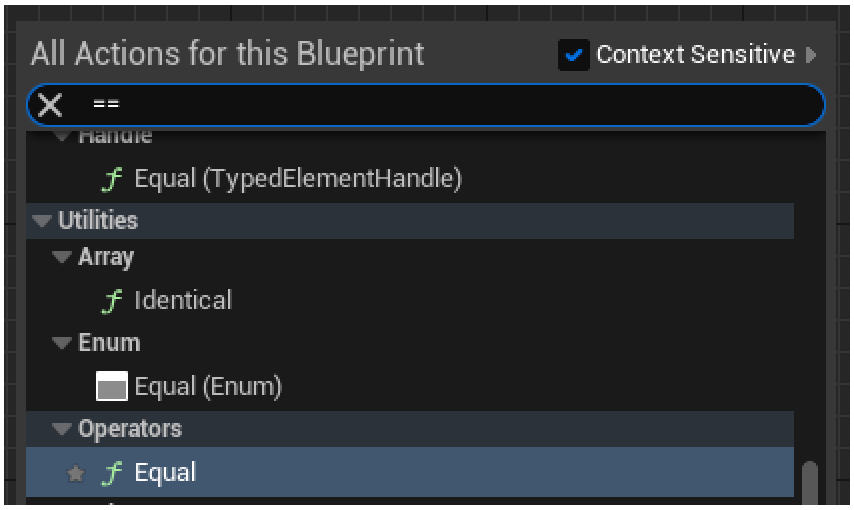
1. Right-click on the Box collision in the Components list and add an OnComponentBeginOverlap event.



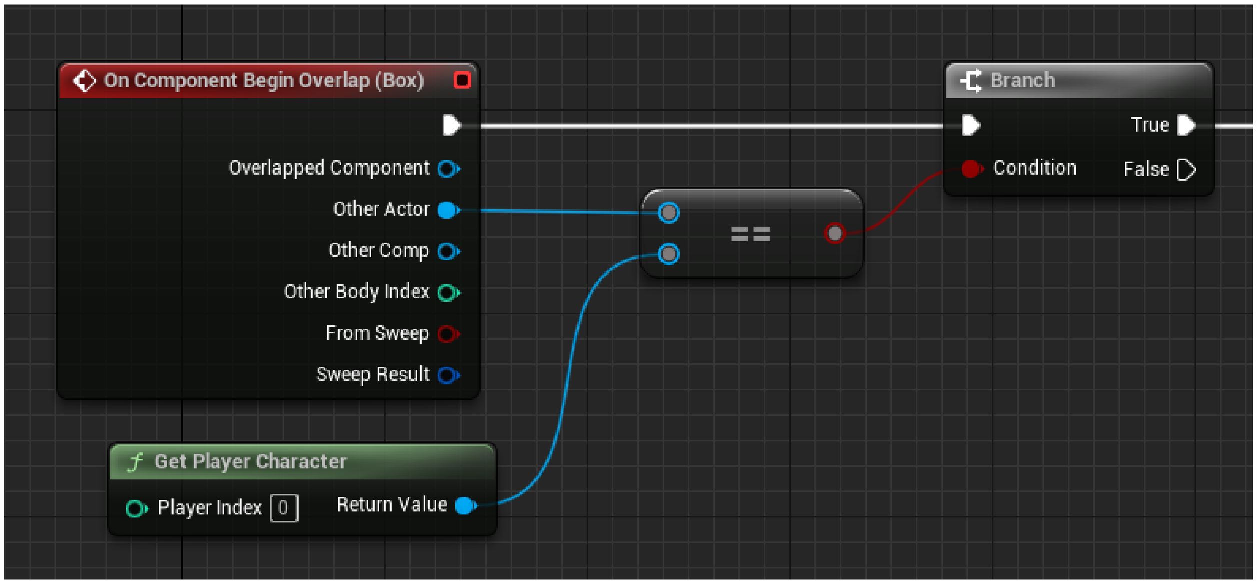
1. In the event, check to see if the overlapping actor is the player character by right-clicking and searching for GetPlayerCharacter.



1. Use a comparison node (==) to check if the overlapping actor is the player character.



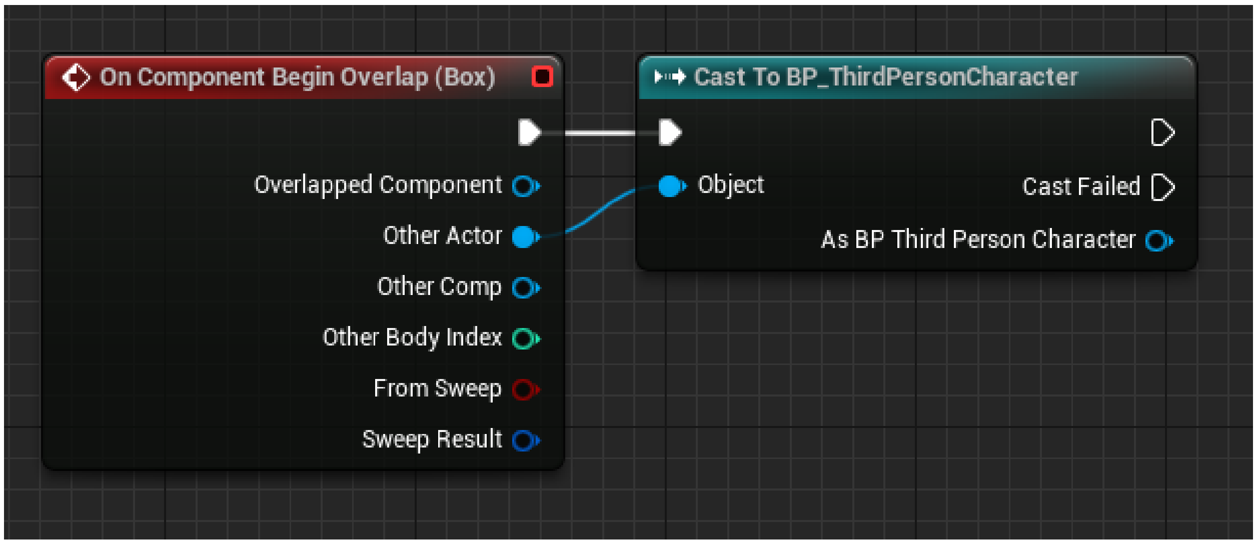
1. Add a Branch node that makes use of the comparison. The Branch node can be added using several different methods. In no particular order:
    a. Press and hold the B key on the keyboard and left-click.
    b. Drag off the boolean return value in the comparison and search for “branch” or “if”.
    c. Right-click in the event graph and search for “branch” or “if”
2. Connect the result of the comparison to the Boolean input of the branch and the execution wire of the overlap event to the branch.



By comparing the Other Actor pin (the actor who overlapped the box) with the player character (retrieved with the GetPlayerCharacter node), the game can determine if the box was overlapped by the player or some other actor in the game.

The comparison returns a True value if the other actor is the player character, and the Branch node will then follow the path of the True pin. If the comparison fails (i.e. the overlapping actor is not the player character), the Branch node will follow the path of the False pin.

Please note: there is another widely used method to do the above comparison:



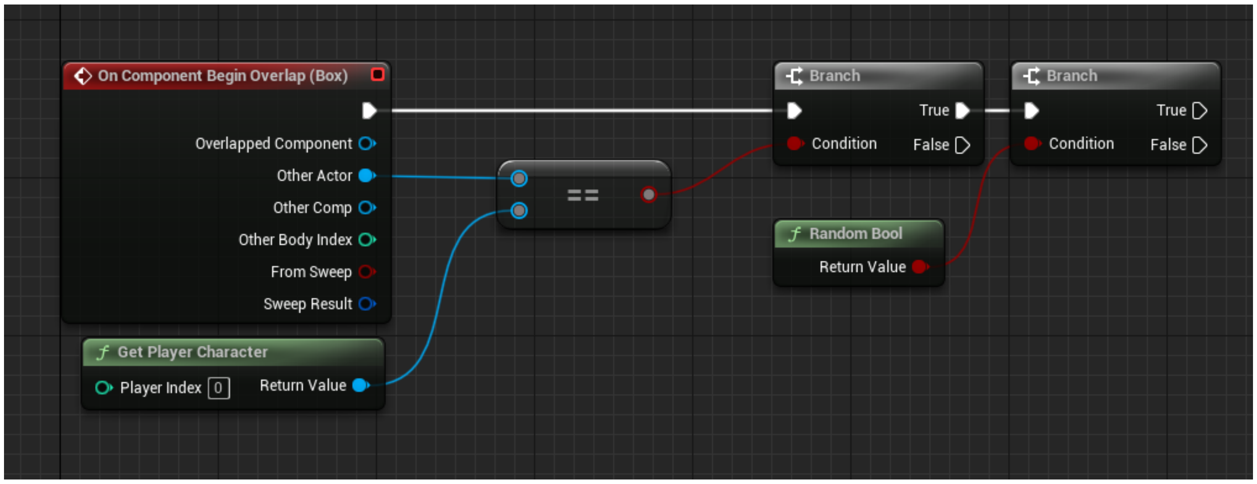
The code above Casts the Other Actor to a BP_ThirdPersonCharacter. The top exec pin is followed if the cast is successful (i.e. the Other Actor is a BP_ThirdPersonCharacter). The Cast Failed pin is followed if the cast fails.

Casting is a term used in programming which changes one object to another object. It succeeds if the object can be changed. This is the case in object-oriented programming if an object is a child class of another object, it can be cast to its parent. The parent can also be cast to a child, but only if the original object was that child class. The topic will be covered in detail later.

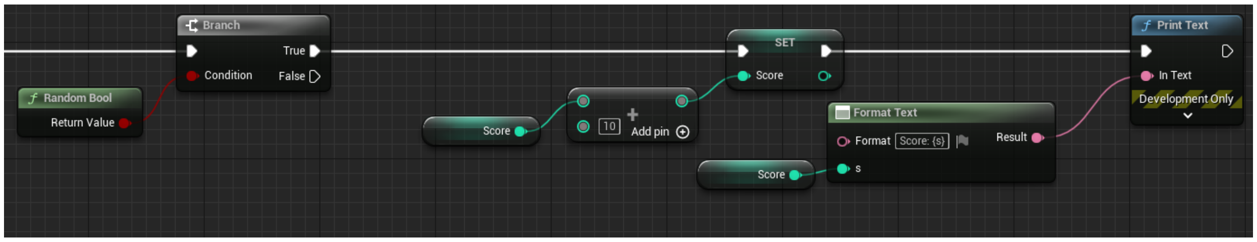
The benefit of using the method above is that if the cast succeeded, the “As BP Third Person Character” pin contains a reference to the actor as that type of actor. The variables and functions of that character can then be utilized.

There is also an argument that it is slower to use the cast method as compared to the comparison method. Investigation into both methods hasn't yielded a conclusive answer.

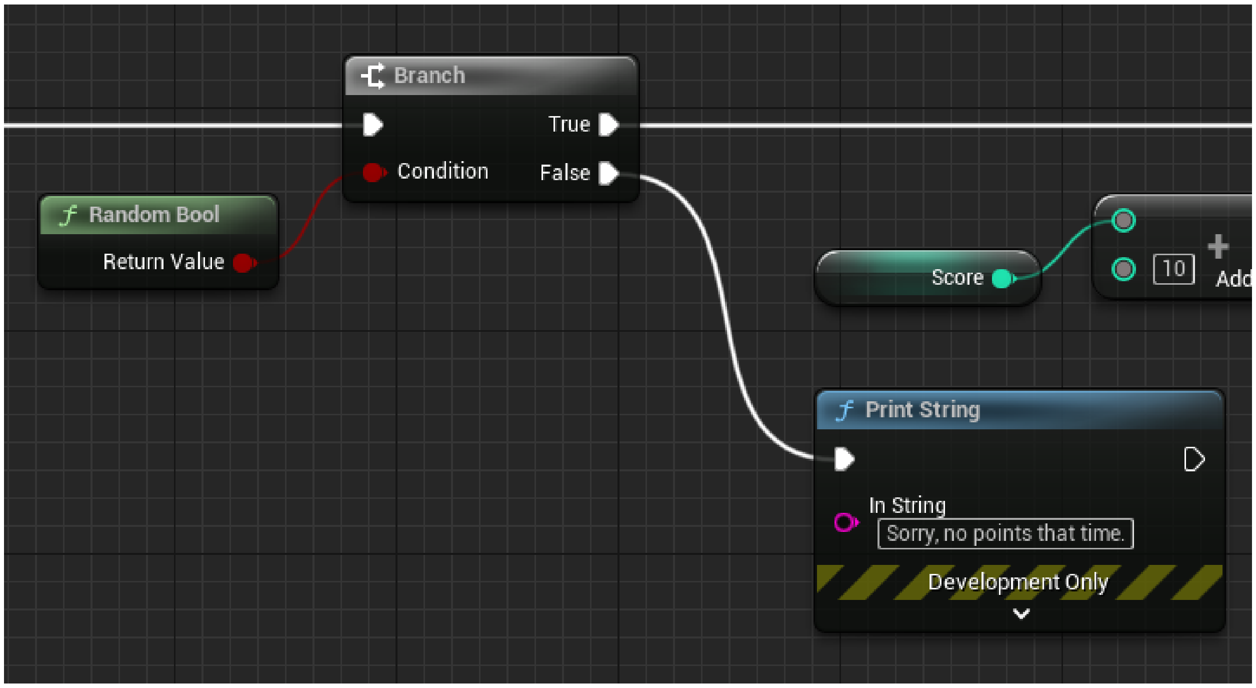
1. Add a Random Bool node and another Branch. Connect the random bool to the condition of the branch and the execution pin to the previous Branch's True pin.



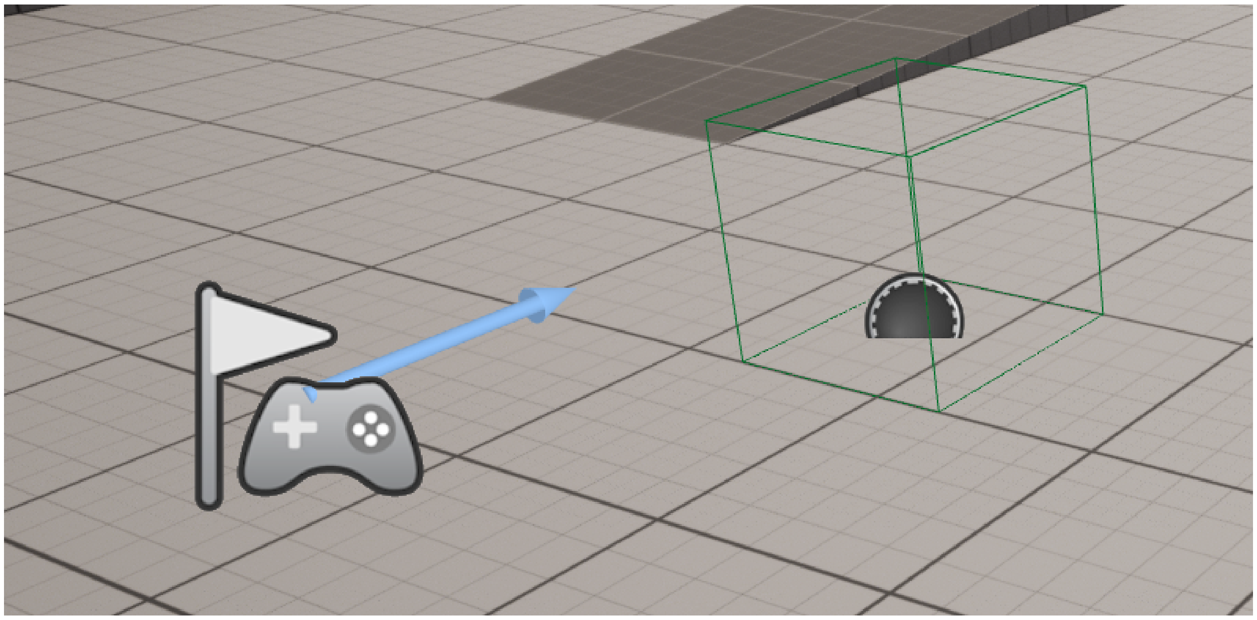
1. From the True pin, add 10 to the Score variable and output its current value.

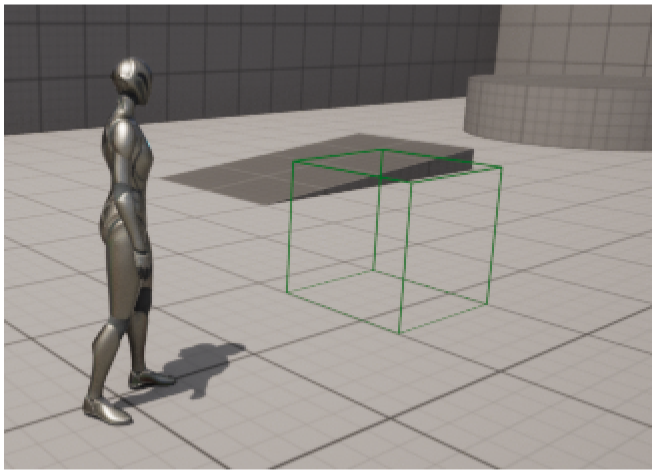


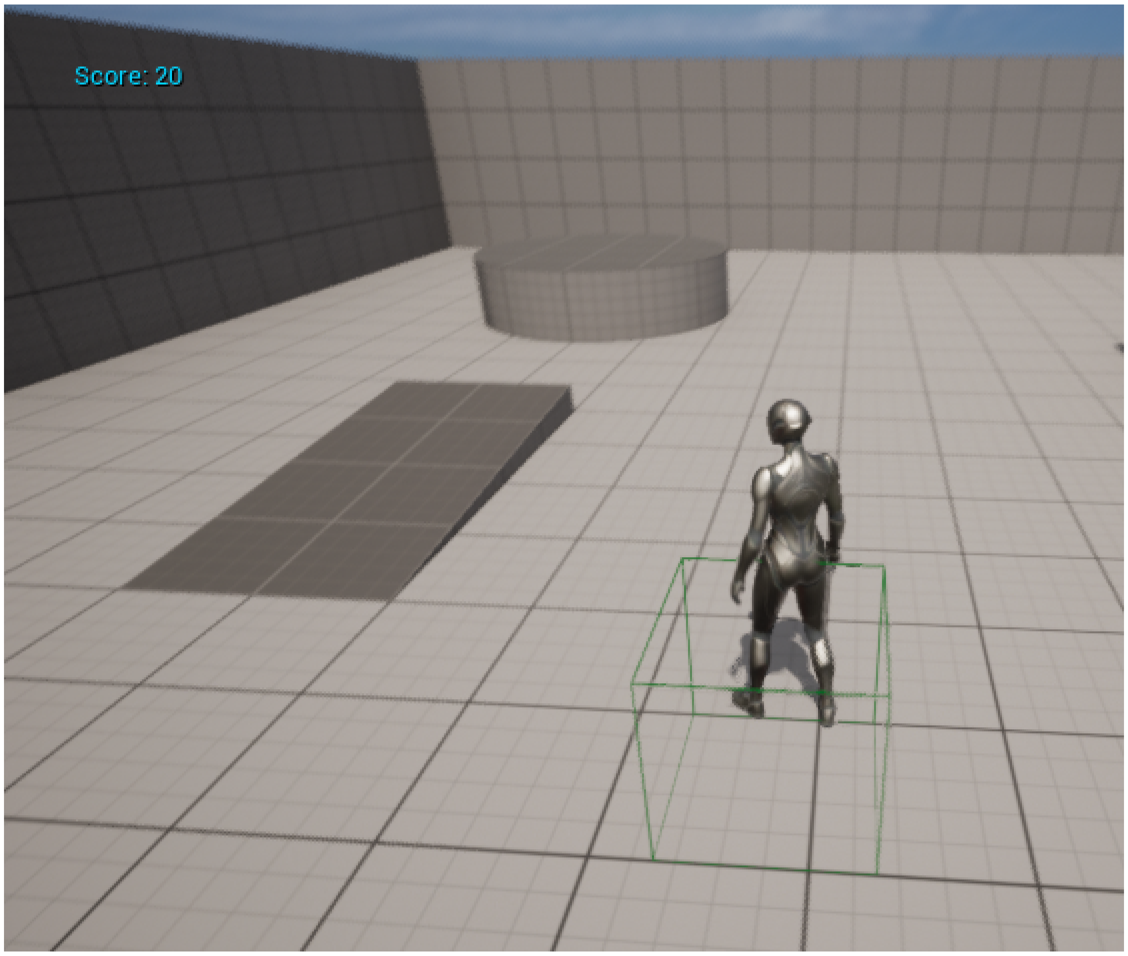
1. From the False pin, output a message stating the no points were awarded.

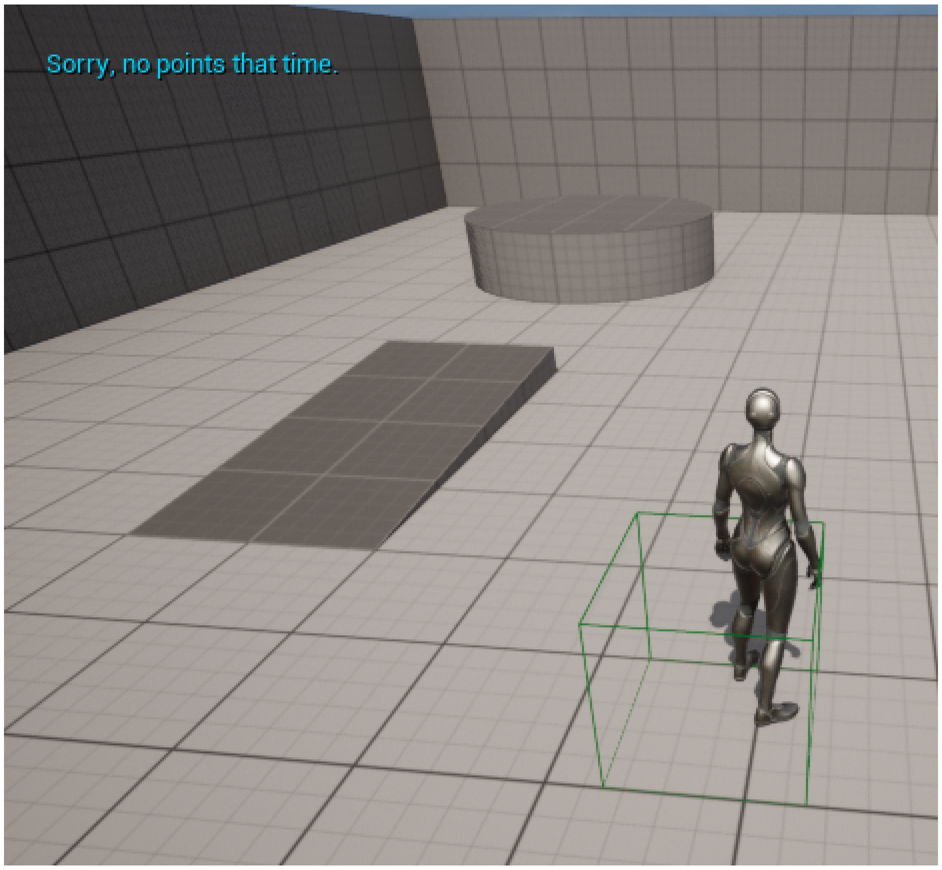


1. Place an instance of the BP_FreeRandomPoints actor in the level and test it by overlapping the box collision with the player character.






Score: 20


Sorry, no points that time.

Approximately 50% of the time, the player should be awarded points.

_
V1.0.0.2 2022_09_18