Integer
Chapter 02 - Lesson 01

The **Integer** type is used to hold values that are whole numbers. This means there are no decimal places with integer variables. With integers, decimal places are discarded in calculations. Numbers are rounded; the decimal places are discarded (i.e. thrown away and ignored).

The following are examples of Integer values:

- 29
- 0
- -3993
- 598291

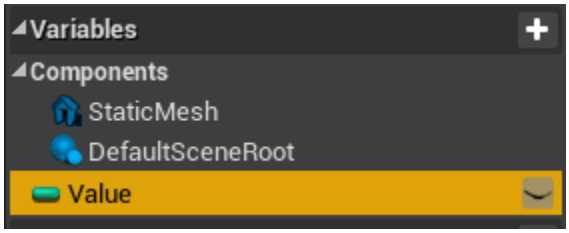
A familiar use of a variable in a game is “the score”. The score is usually a whole number and it starts at zero. More often than not, a negative value for the score isn’t possible, which in computer programming terminology would make that variable “unsigned”. UE does not support unsigned variables, but some programming languages do. This doesn’t mean that you can’t consider a variable “unsigned”, it’s just that UE doesn’t enforce it.

In an RPG game, the player might have several stats, such as XP, Stamina, and Mana. Those are good examples of things that would be represented by integer variables.

Open the level and Starter Blueprint

- Open Maps/Lesson01
- Open Lesson01/BP_IntAddSubtractBlock

The BP_IntAddSubtractBlock contains an Integer variable called Value. In programming terminology, Integers are referred to as “int” (it’s less syllables to say). Some languages use the actual word “int” and some use the word “Integer”.



Integer variable Value in the MyBlueprint panel of the BP_IntVariableBlock

An Integer can hold a value between **-2,147,483,648 and 2,147,483,647**. This is because internally, an Integer is stored as a 32-bit value. You don't need to memorize the minimum and maximum values, but you do need to be aware they exist. Understanding what a 32-bit number is, is also important, but it's not necessary to program.

Be aware that when the maximum value of an Integer is reached and an operation is performed on it that increases its value, it wraps around to its smallest negative value. Conversely, when the minimum value is reached, and the Integer is further reduced, it wraps around to its largest value. This side effect can be unnerving and the source of bugs if the programmer doesn't anticipate it occurring.

Also note that Unreal Engine includes an **Integer64** type which stores values as a 64-bit number. The range of values for Integer64 is −9,223,372,036,854,775,808 and 9,223,372,036,854,775,807. This type is referred to as a "long" in other programming languages such as Java and C/C++.

Math

Variables that were assigned values that never change can be useful (they're called constants), but for the most part, the values in variables are changed throughout the lifetime of a program. They are changed through math operations.

Add / Subtract / Assign

To add a value to a variable, use the Add node (+ operator). To subtract, use the Subtract node (- operator) and to assign a value use the Set node (= operator).

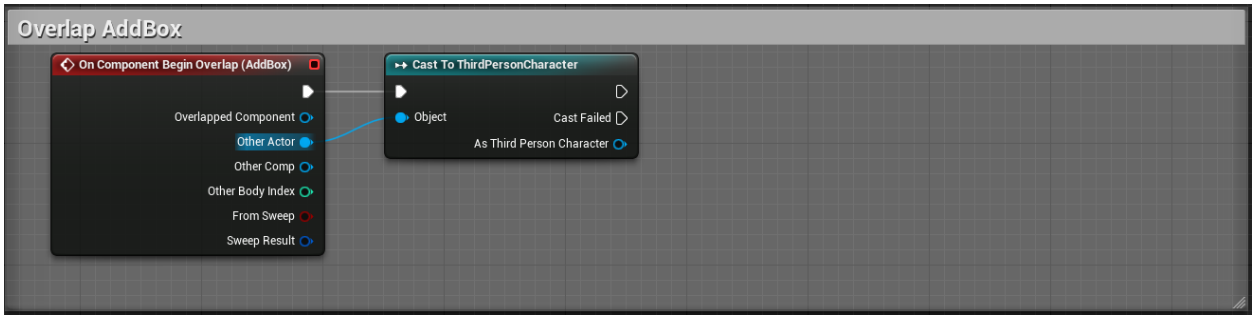
In math, these operations would be written as follows:

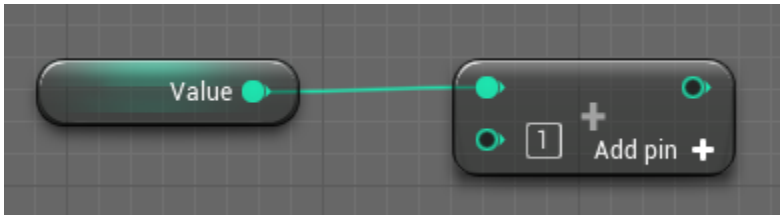
Value = Value + 10

That would take the value of “Value”, add 10 to it, and assign it back to Value. After the operation, the value of Value would be 10 more than before.

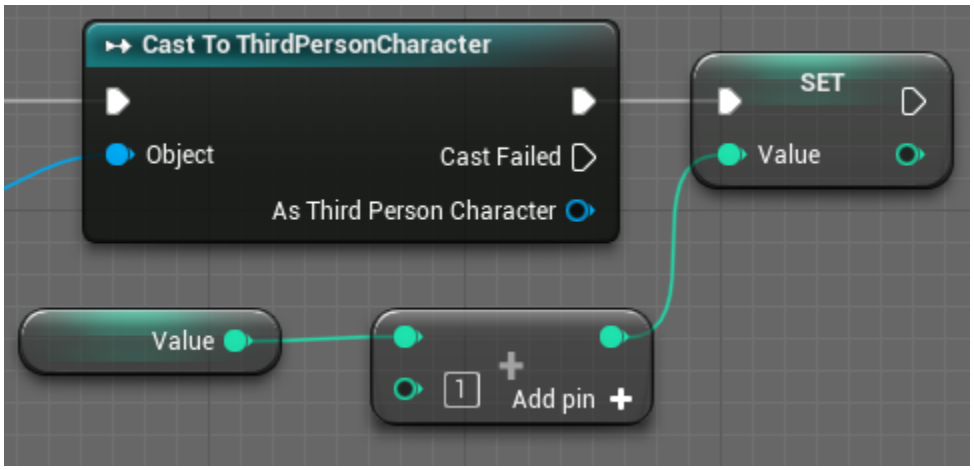
In UE, these operations are accomplished with nodes.

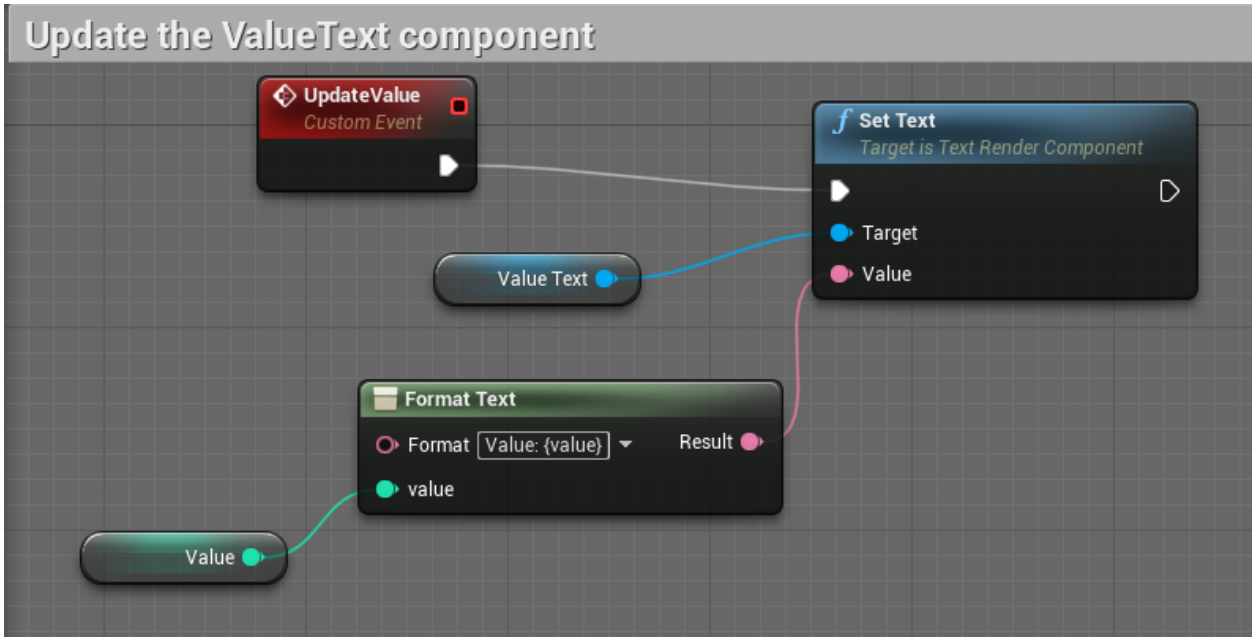
- Locate the nodes in the Event Graph that correspond to when the AddBox is overlapped:



- Get the variable Value by dragging it out from the MyBlueprint panel and holding down the control key. Or, drag it out, drop it on the Event Graph, and select “Get Value”.
- Drag off the Value variable and type “+” into the search.
- Select “int + int”.
- 

- Observe that the Add node accepts two operands. The first is the Value variable - it will take the value from that variable. The second is referred to as being “hard coded”, meaning it’s a constant value that is determined before the program is run. Its value won’t change (it’s constant; you can’t change the value of a constant).
- The result of the operation (addition in this case) will come out the pin on the right side as the sum of both numbers.
- If there are more operands needed, they can be added with the “Add pin” button.
- **The original value of the Value variable isn’t changed by the operation - this is important!**
- Drag the Value variable out of the MyBlueprint panel and hold down the Alt key. This produces a “Set” node. Or, drag it out, drop it on the Event Graph, and select “Set Value”.

- Connect the sum of the addition to the Value pin of the Set node.
- Connect the execution pin of the Cast to ThirdPersonCharacter to the Set node.
- 

- Drag in a call to the Update Value Custom Event so the output text is updated with the new value.
- The Update Value Custom Event was included in the blueprint for your use. It updates the TextRender component in the blueprint. Here's what it looks like:
- 

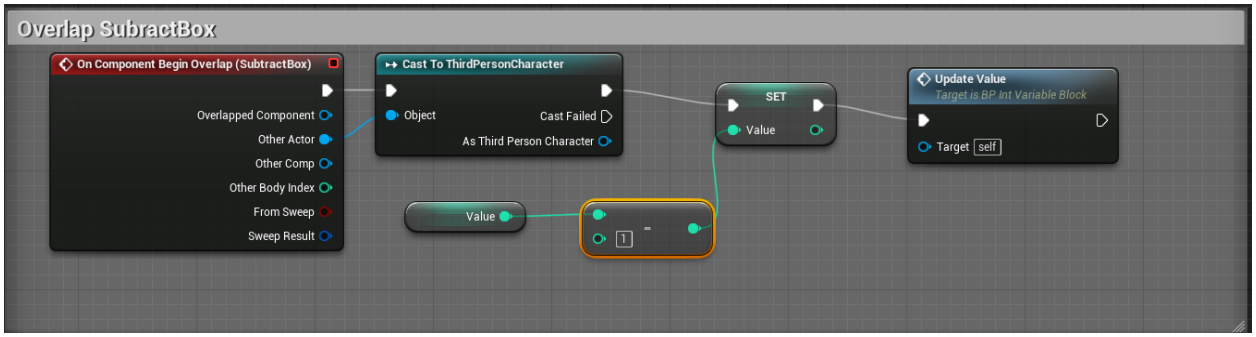
- Test the program by running it and walking into the yellow collision box on the left side of the BP_IntVariableBlock that was placed in the level.

On your own

Complete the code for the OverlapSubtractBox right below the code you just wrote on the Event Graph.

Be sure to test it by walking into the Add box on the right side.

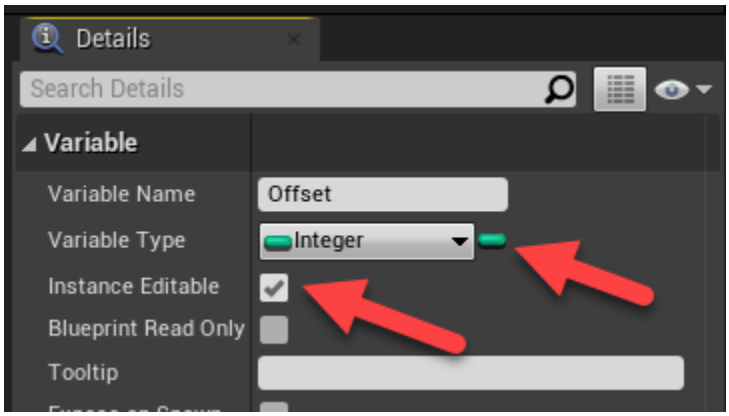
The completed code is found on the next page.

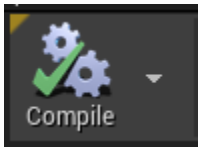


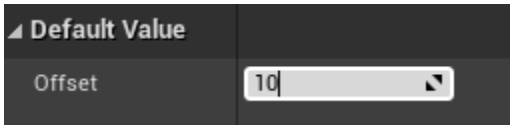
Solution to the previous challenge

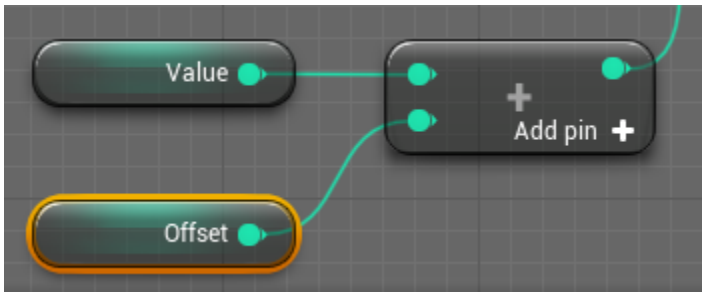
Add a variable

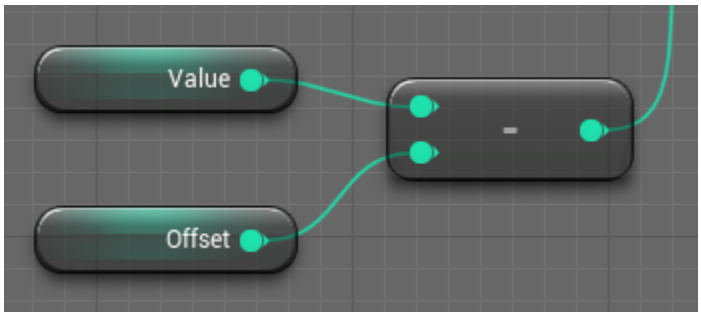
- Click the Add Variable button in the MyBlueprint panel.
- 

- Name the new variable “Offset”.
- In the details panel (on the right side), change the Variable Type to Integer.
- Change the Instance Editable value to true (checked).
- 

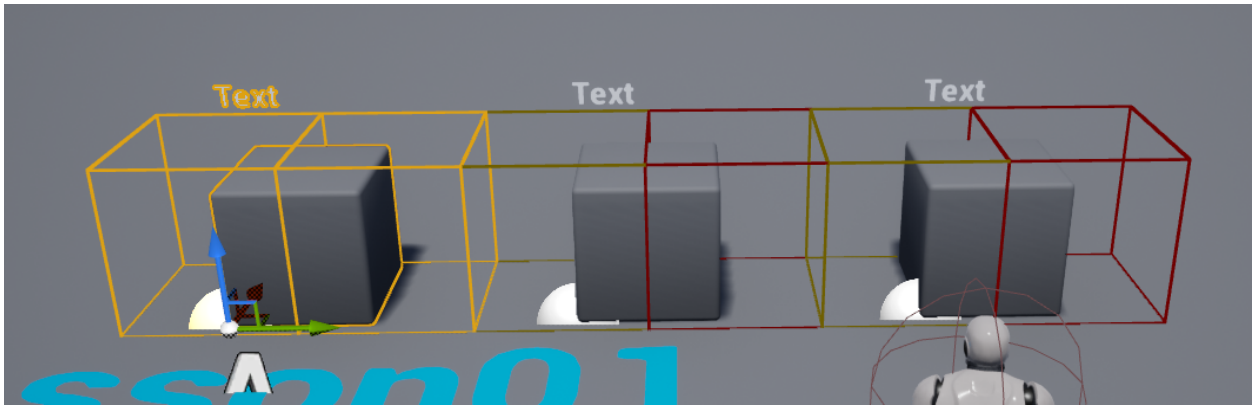
- Press the Compile button in the top toolbar.
- 

- In the details panel, change the default value of the Offset value to 10.
- 

- Modify the code that runs when the AddBox is overlapped by getting the Offset variable and using it for the second operand of the addition node.
- 

- Do the same for the SubractBox overlap code.
- 

- **Keyword: Instance Editable** - Because the Offset variable is “instance editable”, each instance of the Actor placed in the level can be edited and have a different Offset value.
- To demonstrate this, place two additional BP_IntAddSubtractBlock actors in the level next to each other.

- 

- The details for the currently selected actor are shown in the panel on the right side. Locate the Default section and observe that the Offset value can be modified.
- Change the Offset value on each of the actors to a different value.
- 

- Run the game and observe that each instance of the BP_IntAddSubtractBlock uses its own private copy of the Offset value when calculating the new Value to display.
- This demonstrates one of the core concepts of Object Oriented Programming: Encapsulation. We'll revisit these concepts later.
- This also demonstrates a great feature of Unreal, in that you can "expose" variables in the editor by making them "Instance Editable" so level designers can modify the way an actor behaves.

Challenge

Add a component to display the Offset variable and display it above the current value. Use the construction script to update the display of the Offset value when it's updated.

Computer Science with Blueprints and Unreal Engine

_

V1.0.0.1 2022_03_01