Flow Control

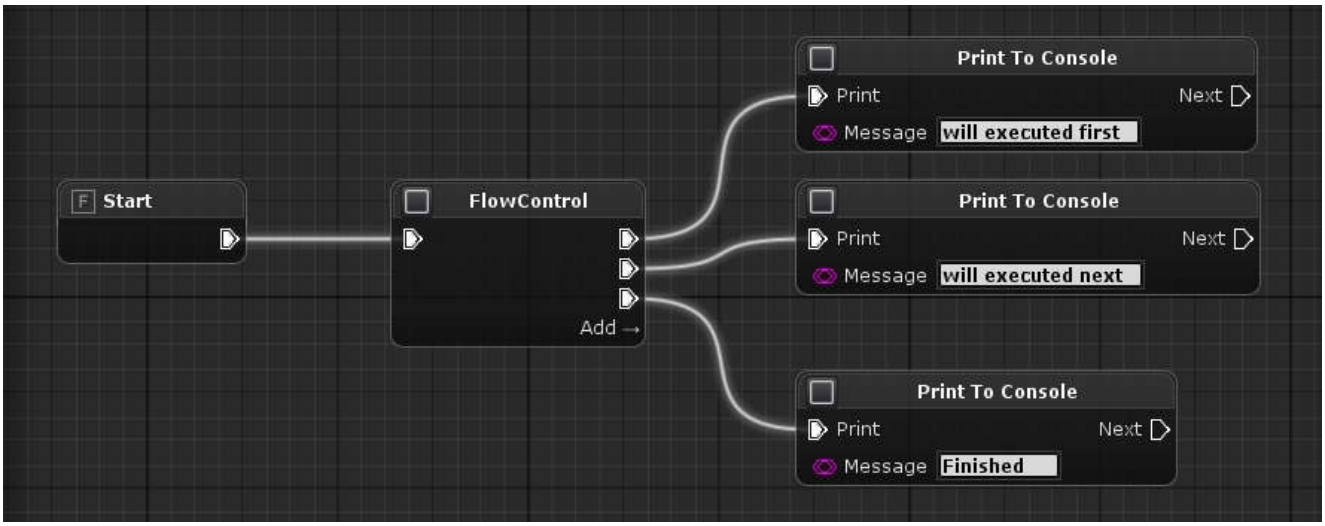
The **Flow control** node allows for a single execution pulse to trigger a series of events in order. The node may have any number of out connectors, all of which get called as soon as the Sequence node receives an input. They will always get called in order, but without any delay.

Add - this button allows you to add as many outputs as you like.

Outputs can be removed by Right-clicking and choosing **Remove Pin**.

Using example

In this example, the **Flow control** node is called at the beginning of the level. It then fires off 3 **Print To Console** nodes in order. However, without a meaningful delay, it will seem as if the messages appear at almost the same time as one another.



If

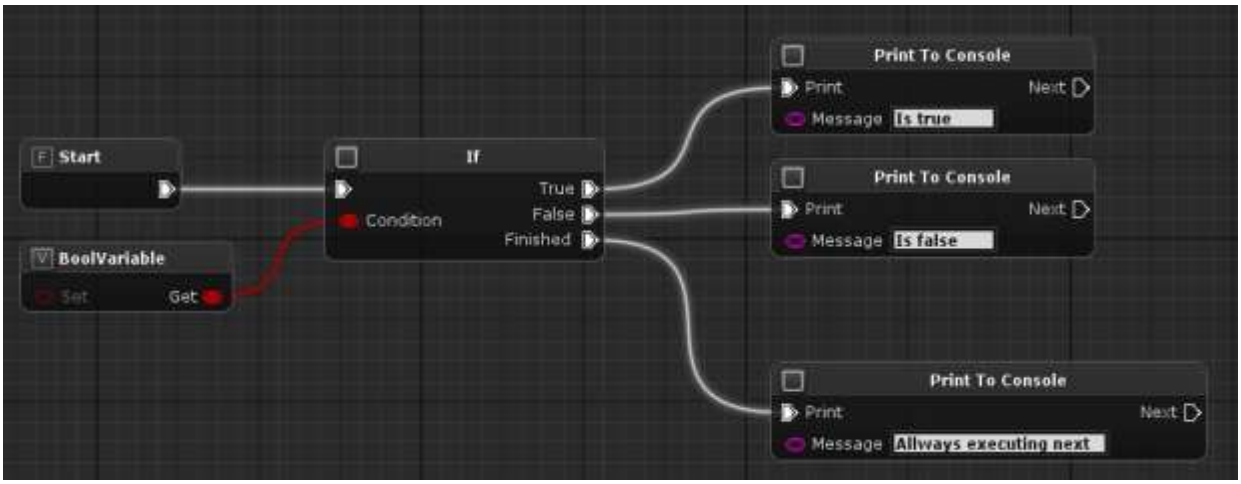
The **If** node Identifies which statement to run based on the value of a Boolean expression.

The **True** connector code will be executed if the input pin **Condition** is true, otherwise the **False** connector code will be executed.

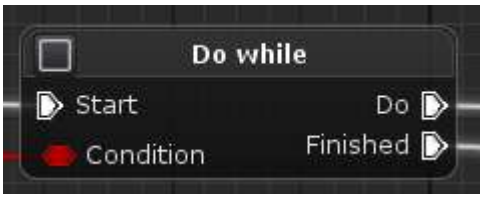
The **Finished** connector code will be executed always after the True or False code.

Using example

In this example, the **If** node will be called at the beginning of the level. If the value of the variable BoolVariable is true the "**Is true**" message will be printed to the console, otherwise the "**Is true**" message will be printed. After the **True** or **False** code was executed, the third message will be printed to the console anyway:



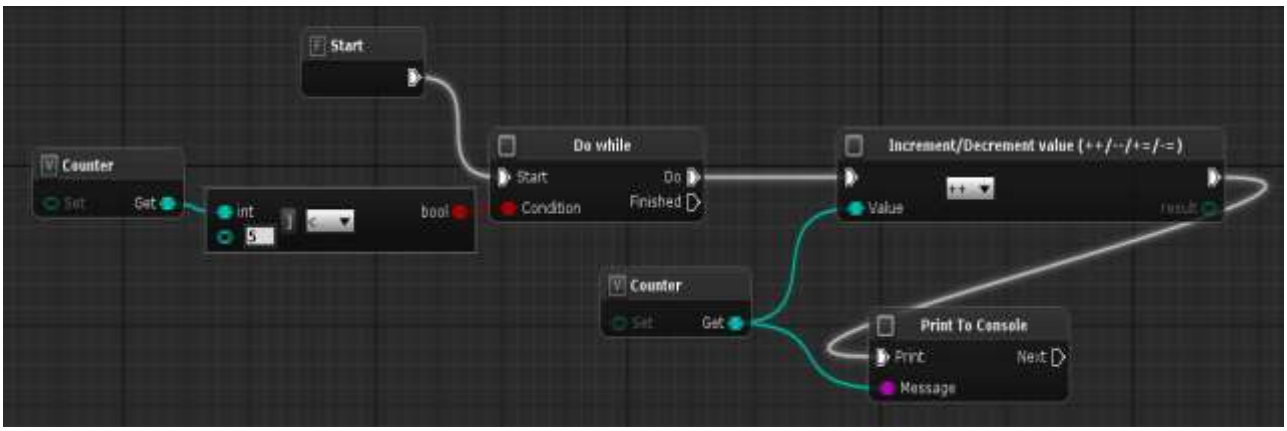
Do while

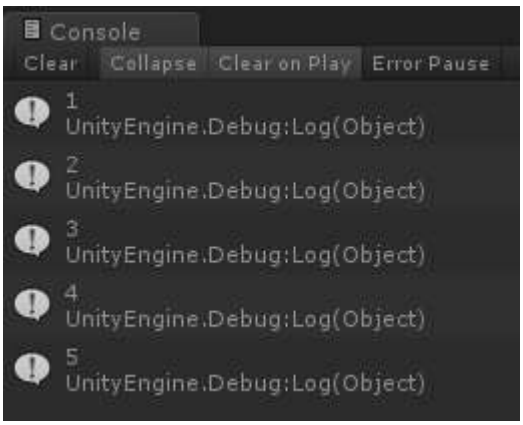


The **Do while** node will execute the **Do** code then checks the **Condition**. If it is true the **Do** code will be executed again, otherwise the **Finished** code will be executed.

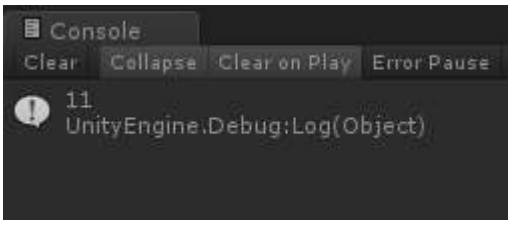
Using example

In this example the **Do while** node will increase the variable and print its value to the console while the value of the variable is less than 5.





If we set the initial value of the **Counter** variable to 10 the output will be:

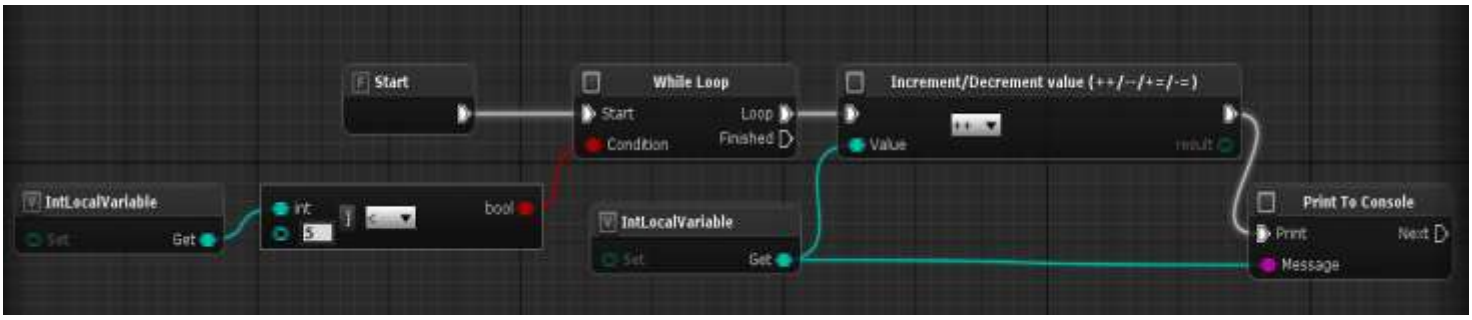


While loop

The **While Loop** node will execute the **Loop** code as long as a specific **Condition** is true. During each iteration of the loop, it checks to see the current status of its input boolean value. As soon as it reads false, the loop breaks. After that the **Finished**code will be executed.

Using example

In this example the **While Loop** node will increase the variable and print its value to the console while the value of the variable is less than 5:

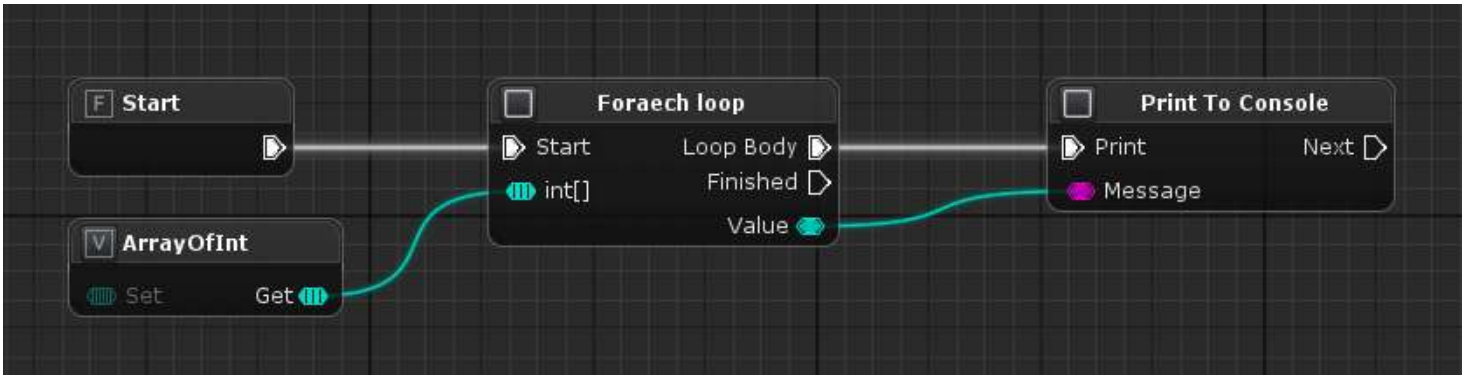


Foreach loop

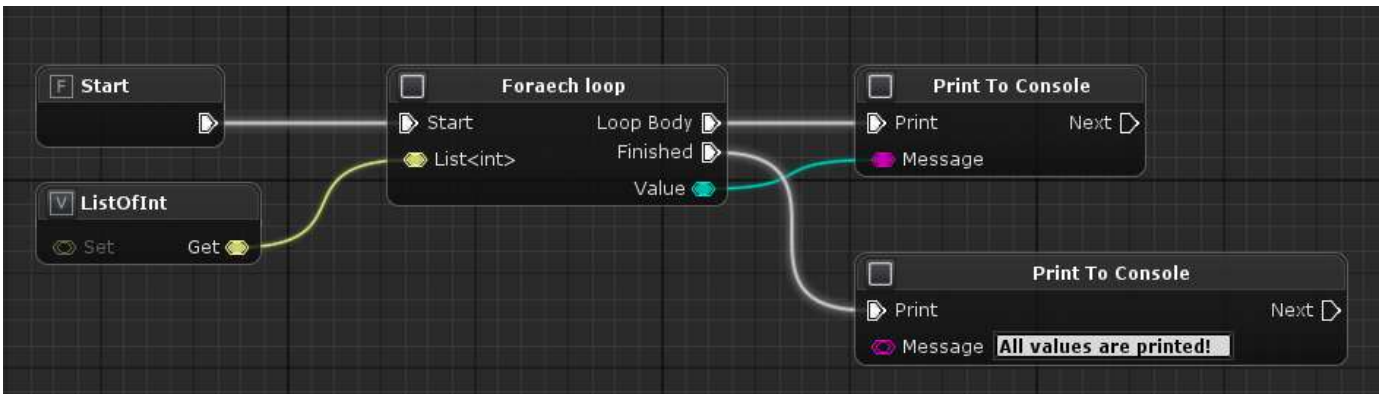
The **Foreach Loop** node will iterate through all of the elements of the connected array or collection. On the each iteration this node will execute the **Loop Body** code and set the value of each element to the **Value** pin. After completion of the iteration the **Finished** code will be executed.

Using example

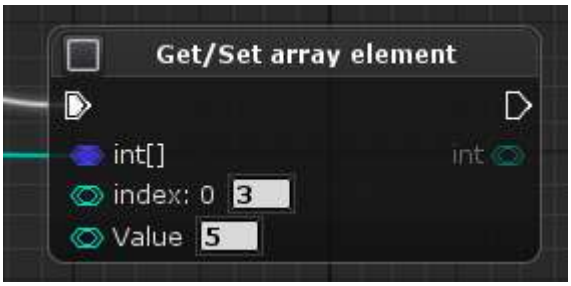
In this example the **Foreach Loop** node will iterate through all of the elements of the connected array and print their values to the console:



Iterating through a **List<>** collection is the same. After printing all values of the elements to the console, the "**Print to console**" node will be executed:



Get/Set array element



The **Get/Set array element** node is used for getting or setting the value from an array on defined index.

Pins

Array - the array from which you will get or set value;

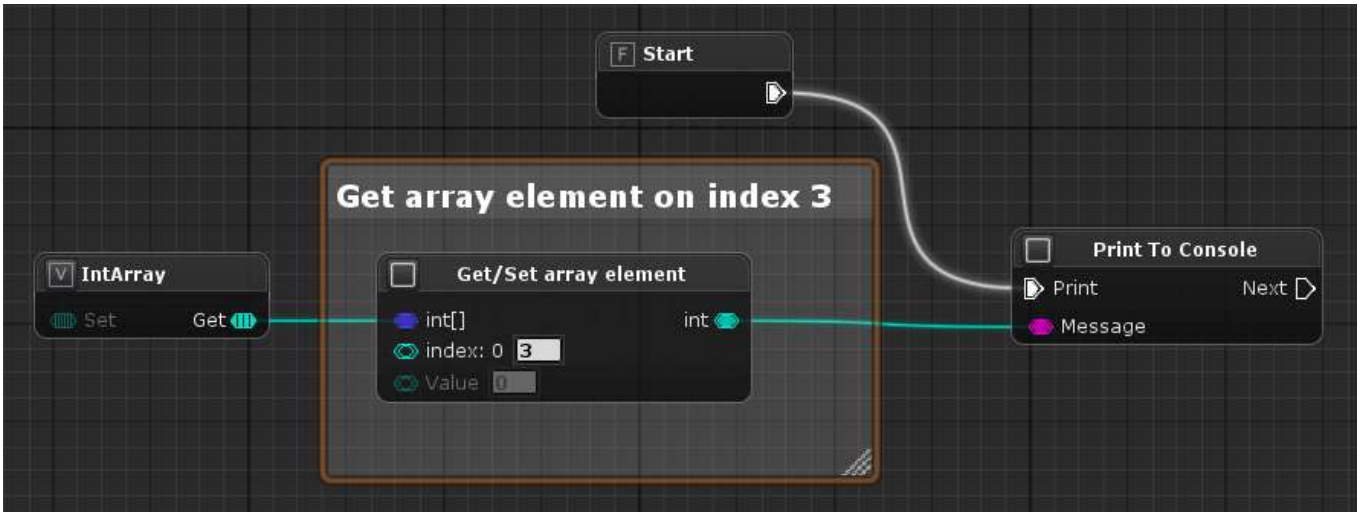
Index - index of the element in array;

Value - the new value of element.

Using example

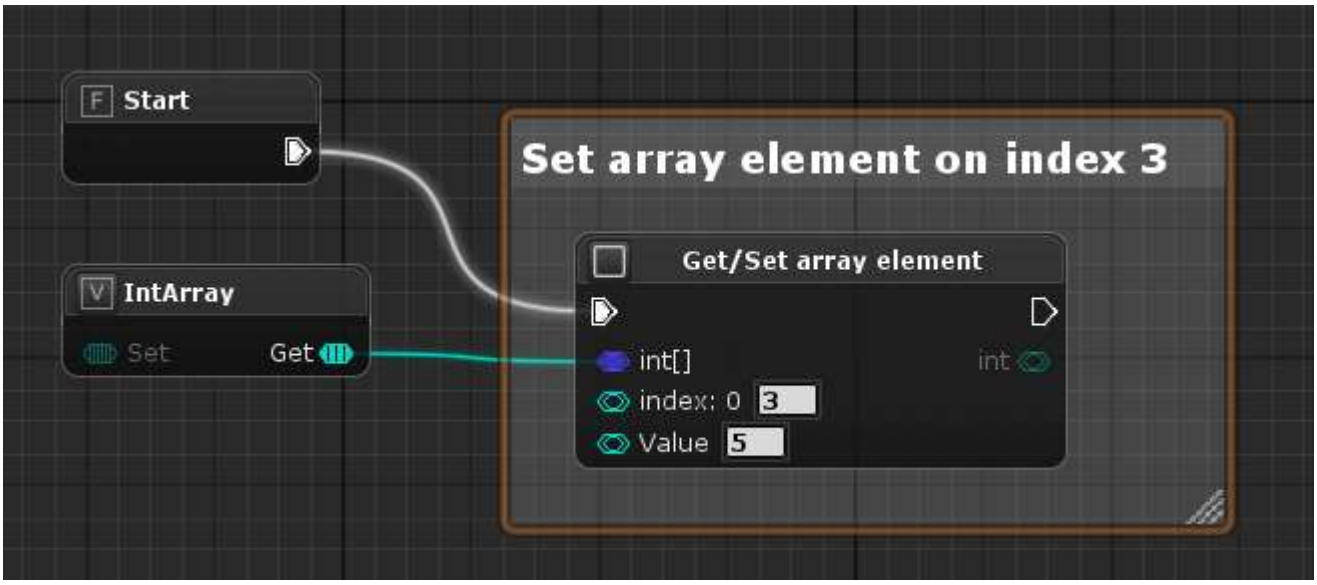
Get element:

Connect the array to Value pin of node and set the **index** of element. The out pin will return the element value:



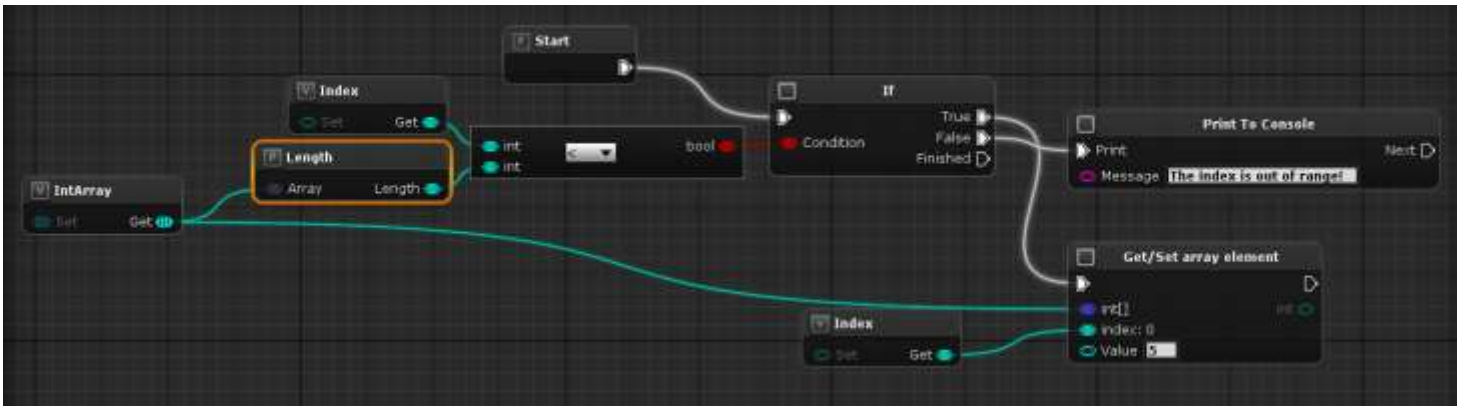
Set element:

Connect the array to the Value pin node, set the index of element and the new **Value** of element:



Note: be sure that the **index** value is **LESS** than the **array length**, otherwise it will cause an out or range error.

To check the array length use the "**Array.Length**" node:

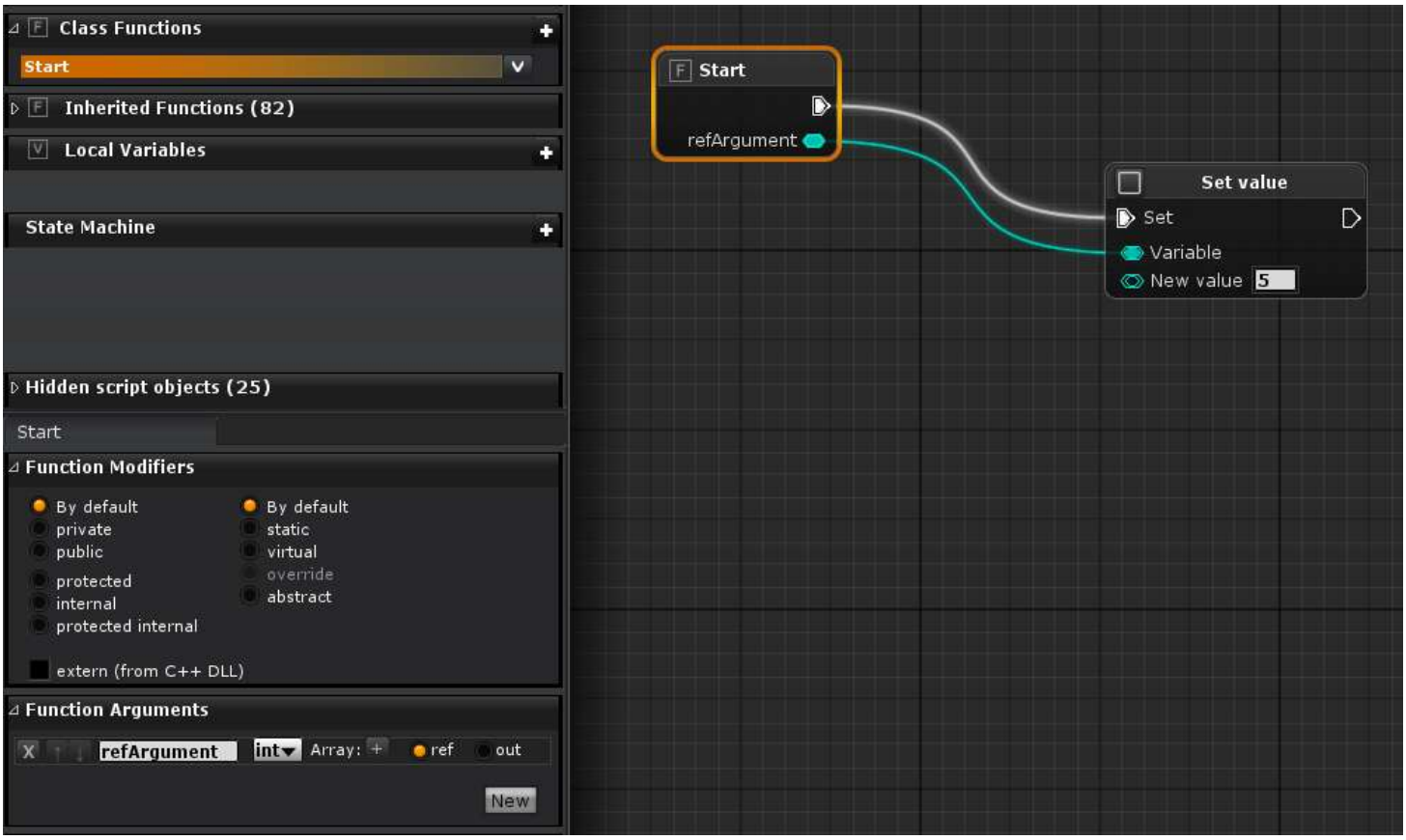


Set value

The "**If**" node is used when you need to set a value of "**ref**" or "**out**" to an IN function.

Using example

Connect "**Set value**" node to "**ref**" or "**out**" pin in the entry node of function:

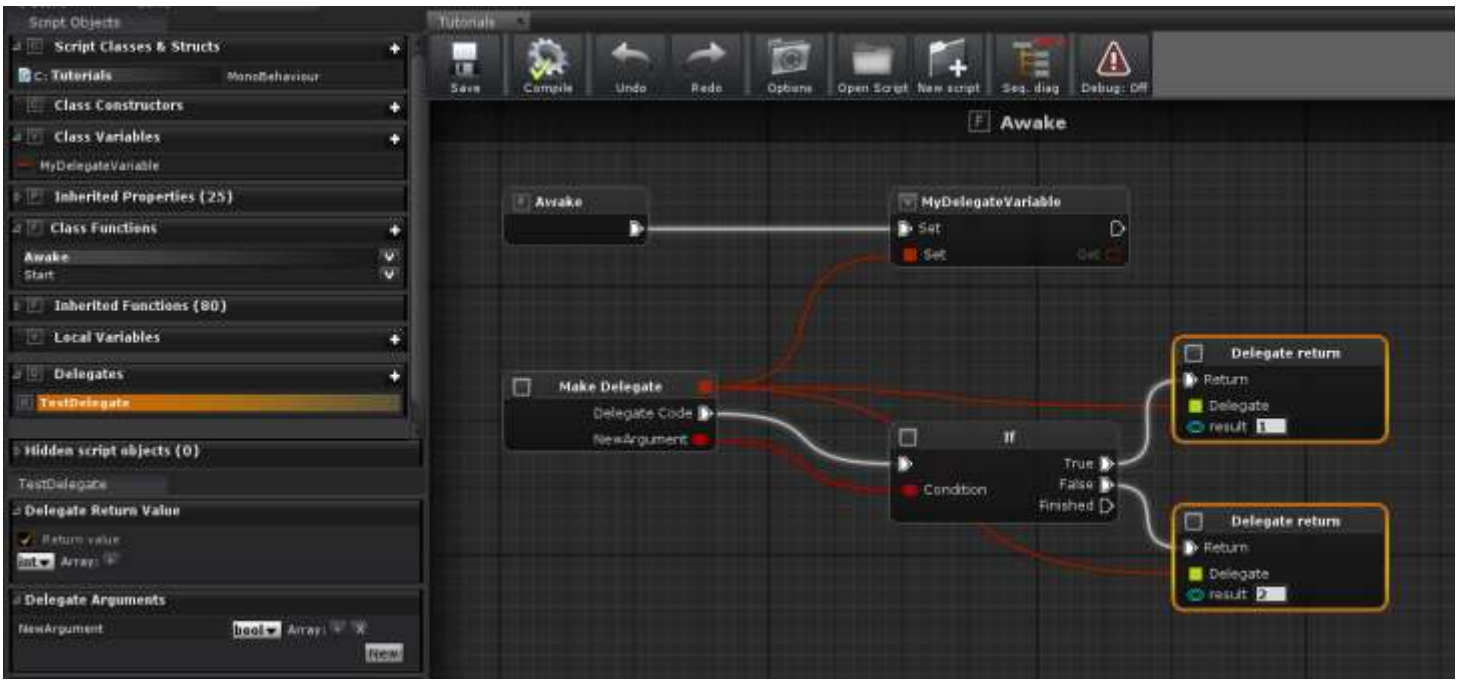


Delegate return

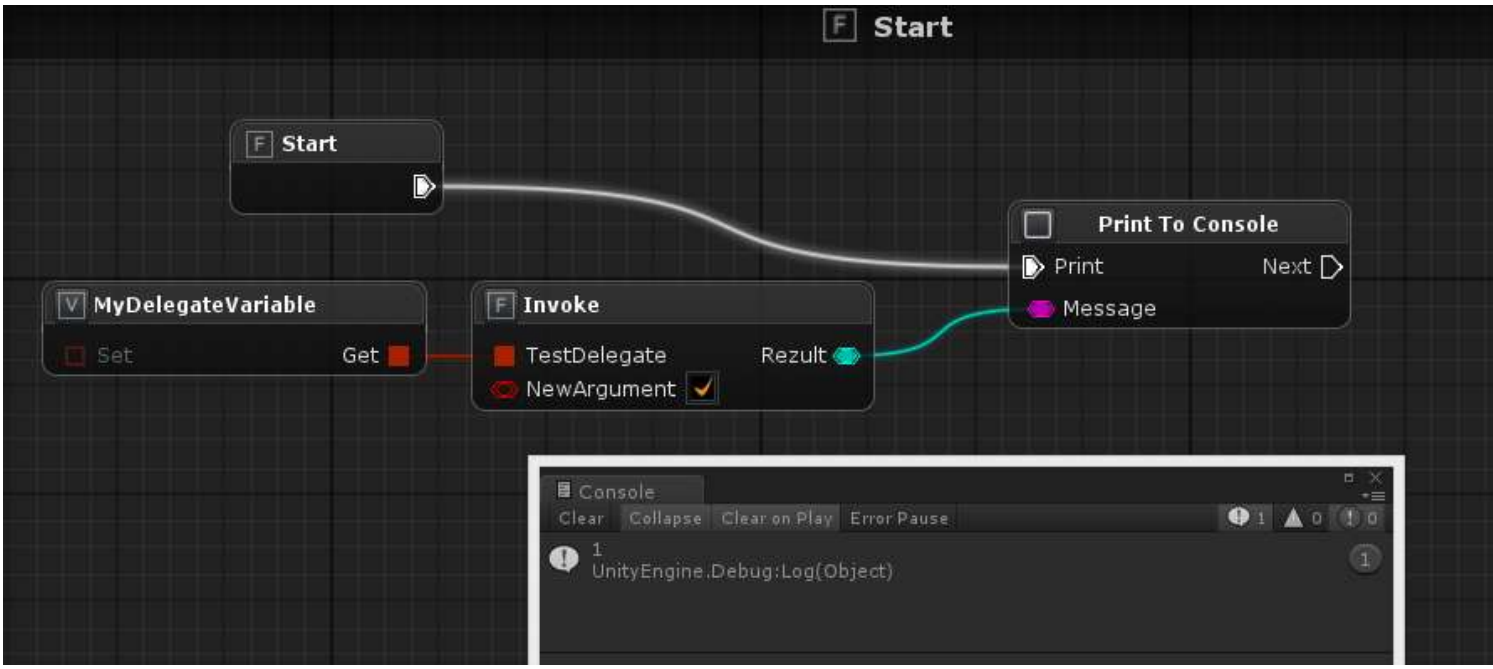
The **Delegate return** node is used to return or return values in delegate code.

Using example

In this example we set the code of a delegate **TestDelegate** (with a **bool** argument and a return type **int**), and to return the **int** value from it we use the **Delegate return** nodes:



After invoking this delegate we get the returned value from it:

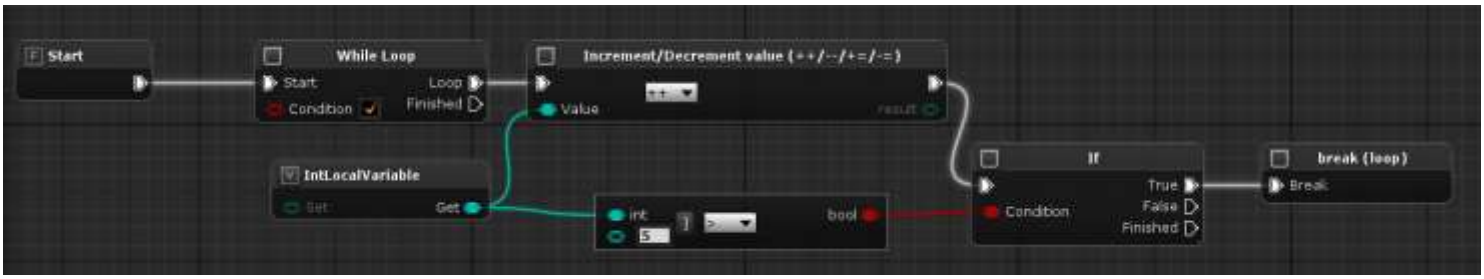


Break

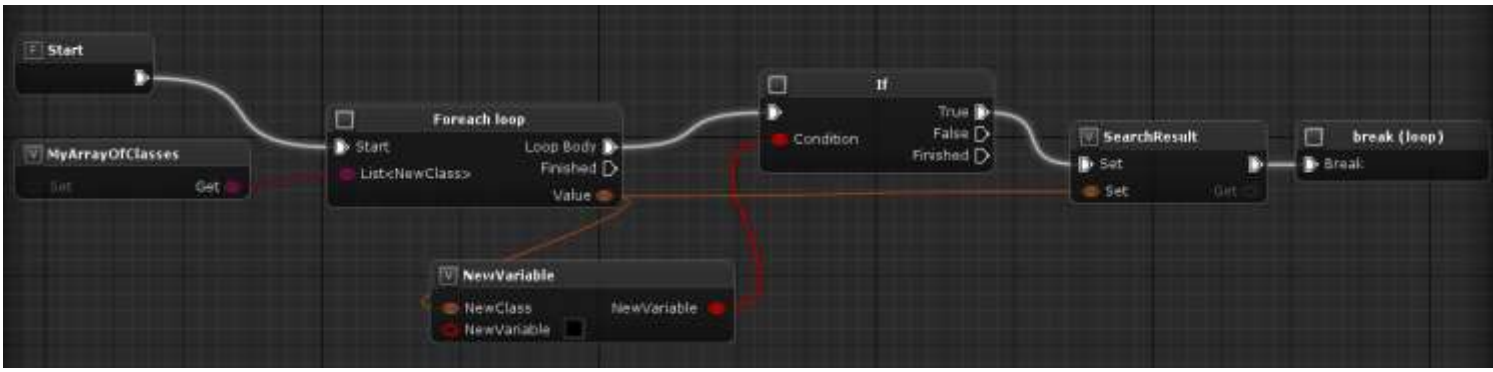
The “**break**“ node is used to stop iteration and exit from any loop node (foreach, for, while, do).

Using example

In this example we “break” the while loop when the value is more than 5:



In another example we iterate through all objects in a collection and stop the foreach loop when we find the object that we are looking for and it makes no sense to continue the loop:



Continue

The “**contunue**“ node is used in loop nodes (foreach, for, while, do) to skip the execution of the rest of the iteration loop code.

Using example

In this example we iterate through all objects in a collection and skip iterations in the **foreach loop** that does not satisfy the **If** statement:

Print To Console

The "**Print To Console**" node is used for logging messages to the Unity Console. It's a macro for a Debug.Log node.

Using example

In this example we log the message "**The game is started!**" to the console at the beginning of the level:

Try-Catch-Finally

The **Try-Catch-Finally** node consists of a **Try** connector followed by one or more **Catch** connectors, which specify handlers for different exceptions.

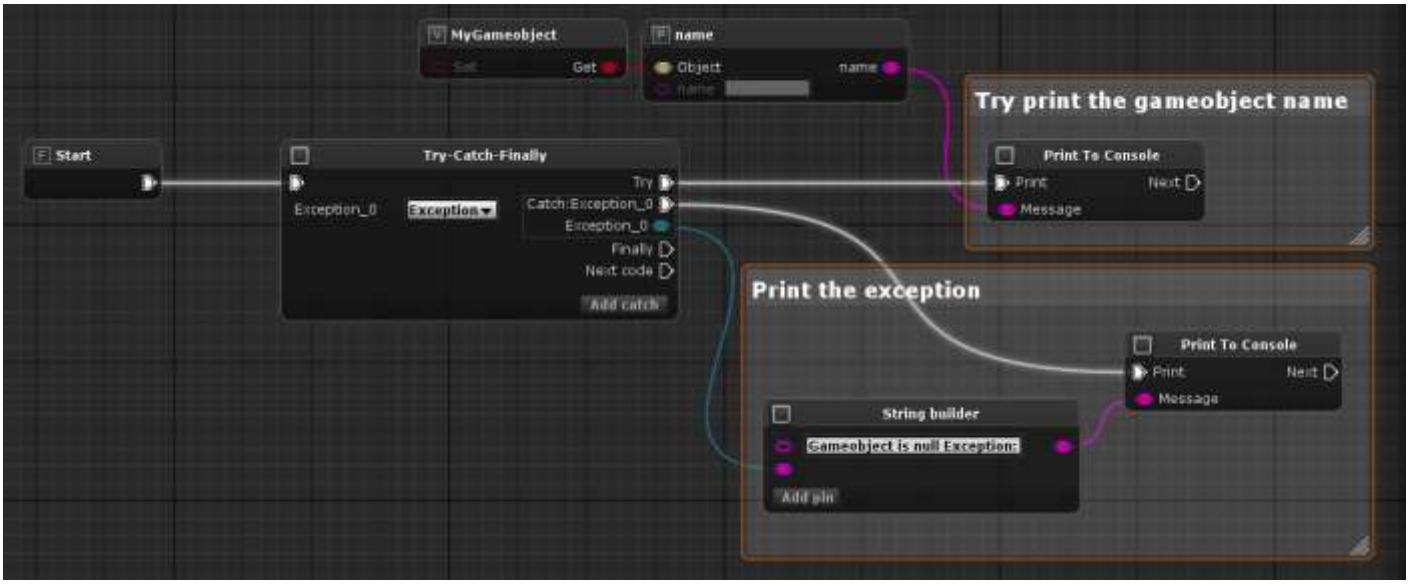
You can execute some code in a **Try** block that can contain errors, and if they happen the **Catch** block will be executed and you can get the exception from the **Catch:Exception_N** pin.

The **Finally** connector will be executed anyway. You can clean up any resources that are allocated in a **Try** block, and you can run code even if an exception occurs in the try block.

The **Next code** connector is used for executing code that goes after the **Try-Catch-Finally** node.

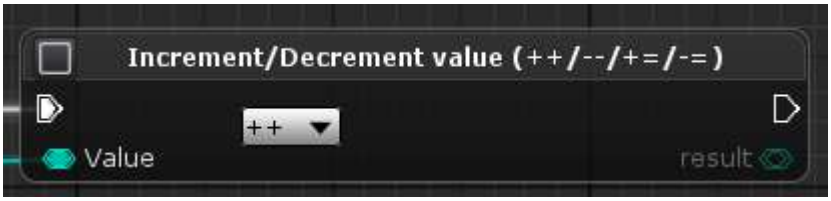
Using example

In this example we will try to print the name of the GameObject, and if it is null the Catch block will be executed with an exception (null reference exception), which will be logged into the console:



Note: in this case it's better to use nodes "**If**" and "**is null**" node instead of a **Try-Catch-Finally**.

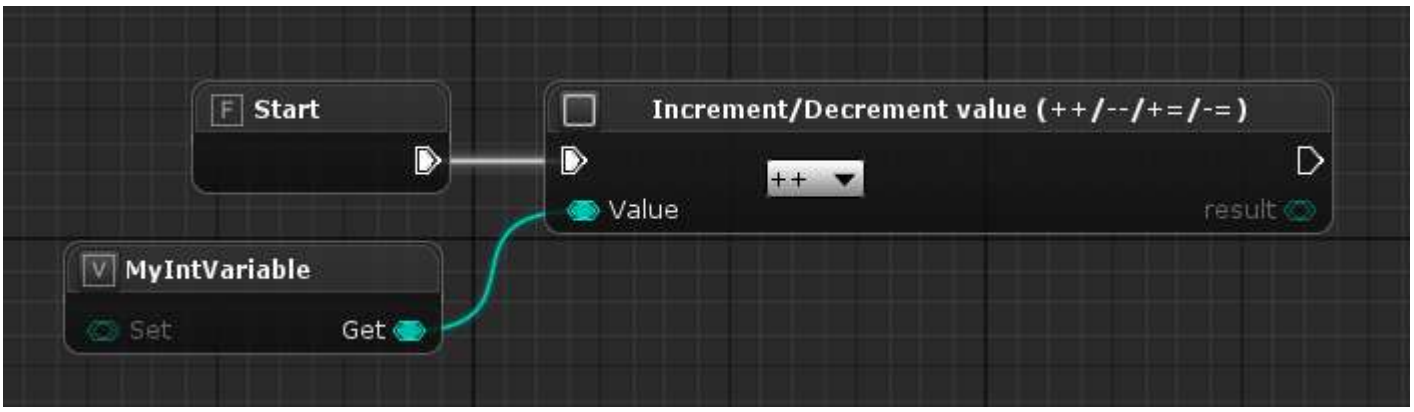
Increment/Decrement value



The **Increment/Decrement value** node is used for mathematical operations on numerical values.

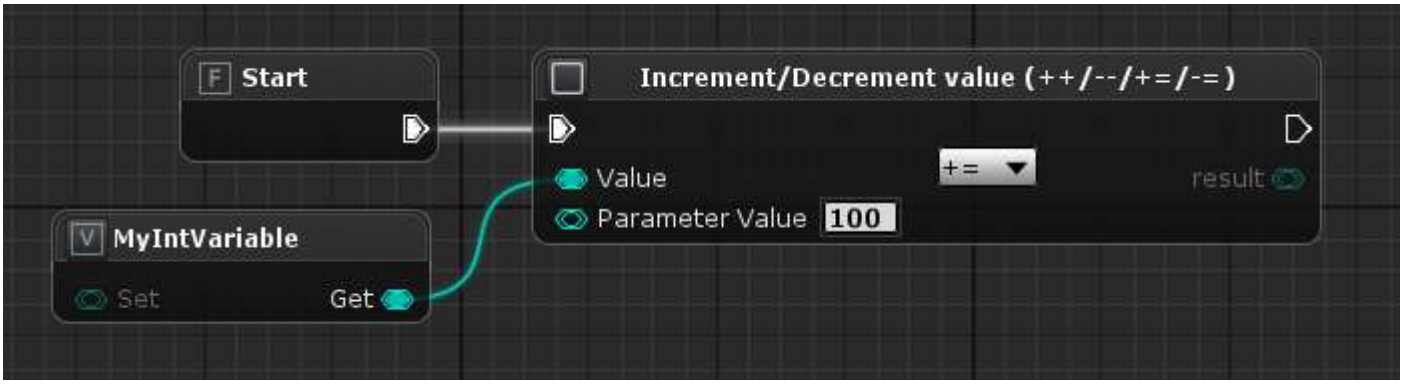
Using examples

For **post-incrementing** the integer value connect it to the IN pin and select the '**++**' operation. After executing this node the value will be incremented by one:



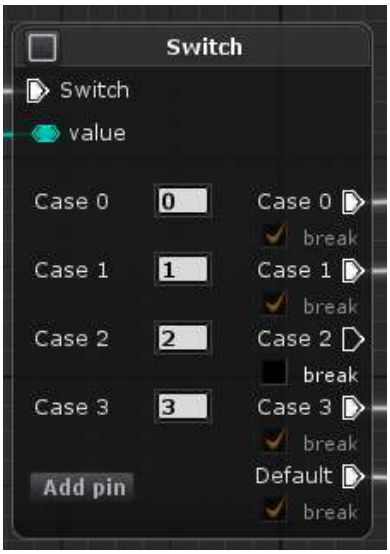
The **post-decrementing** operation can be done in the same way but by selecting the '**--**' operation. After executing this node the value will be decremented by one.

Also this node is used for incrementing the variable by some value:



Also the **Increment/Decrement value** node is used for working with delegates. For more information check the **Delegates** documentation page.

Switch



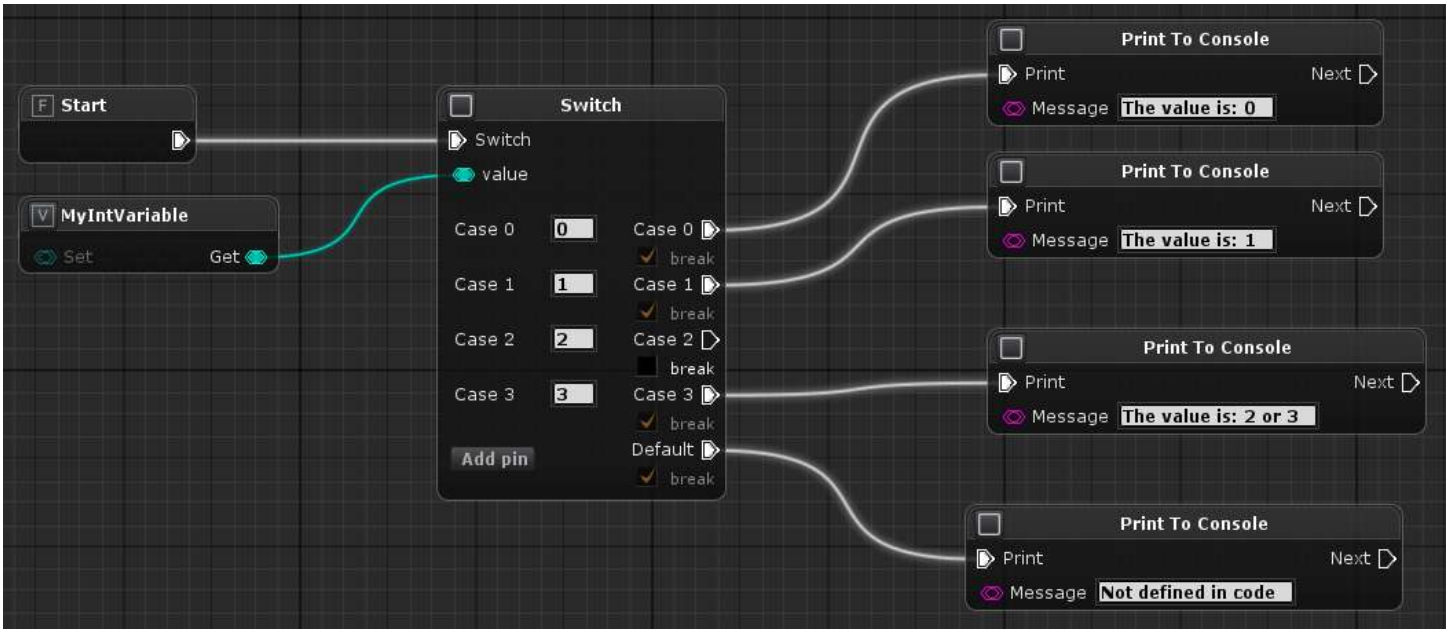
The **Switch** node is used for identifying which statement (**Case**) to run based on the IN pin value. You can set the values near each case that identifies the Case executing condition. If there is no code in some case connector- the next case code connector will be executed if the **break** option is not set.

The **Default** connector will be executed if no condition is passed.

Note: Duplication of condition values is not allowed.

Using example

In this example we will execute the code based on the IN pin value. If the variable value is 2, Case #3 will be executed because there is no code to execute and option break in Case #2 is not set:

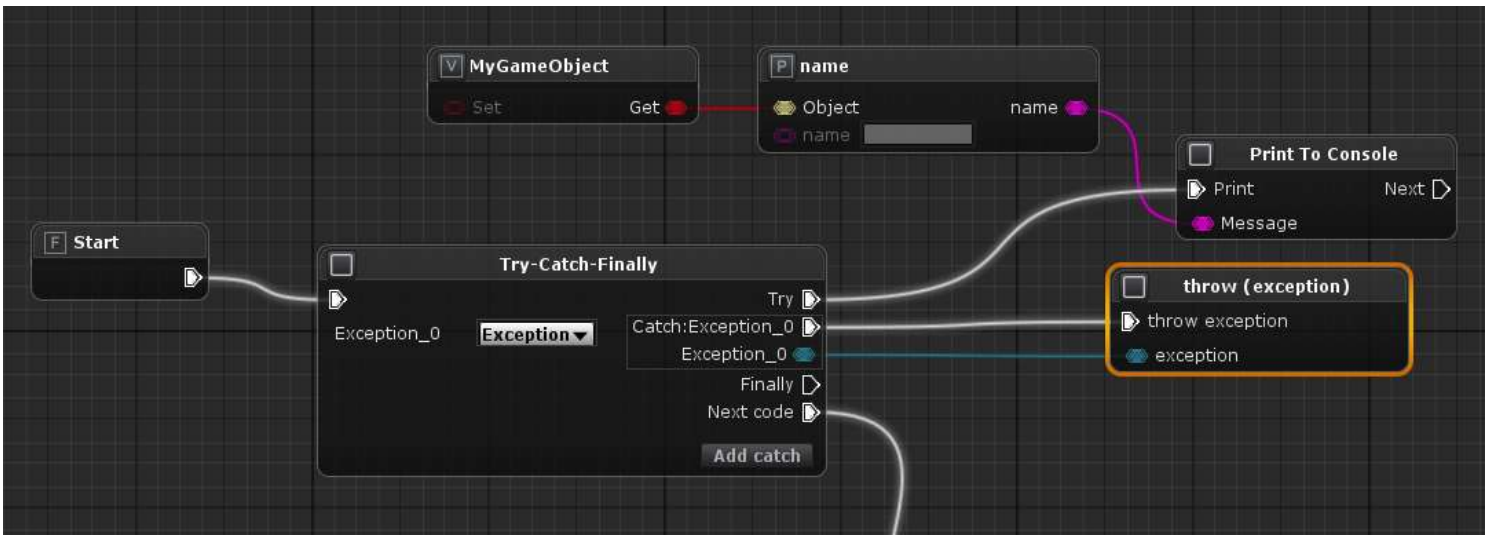


Throw

The **throw** statement is used to signal the occurrence of an anomalous situation (exception) during program execution.

Using example

It can be used for throwing exiting exceptions from a **Try-Catch-Finally** node:



Also it is used for throwing custom exceptions:

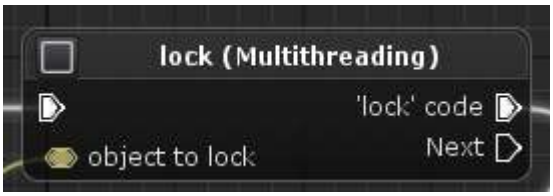
using (Disposable)

The **using (Disposable)** node is used for temporary usage objects with an IDisposable interface. It means, that this object will be destroyed after using and it will clean up all resources that are used by this object.

Using example

In this example we will read text from a file using the StreamReader class. After reading all text the StreamReader class instance will be destroyed:

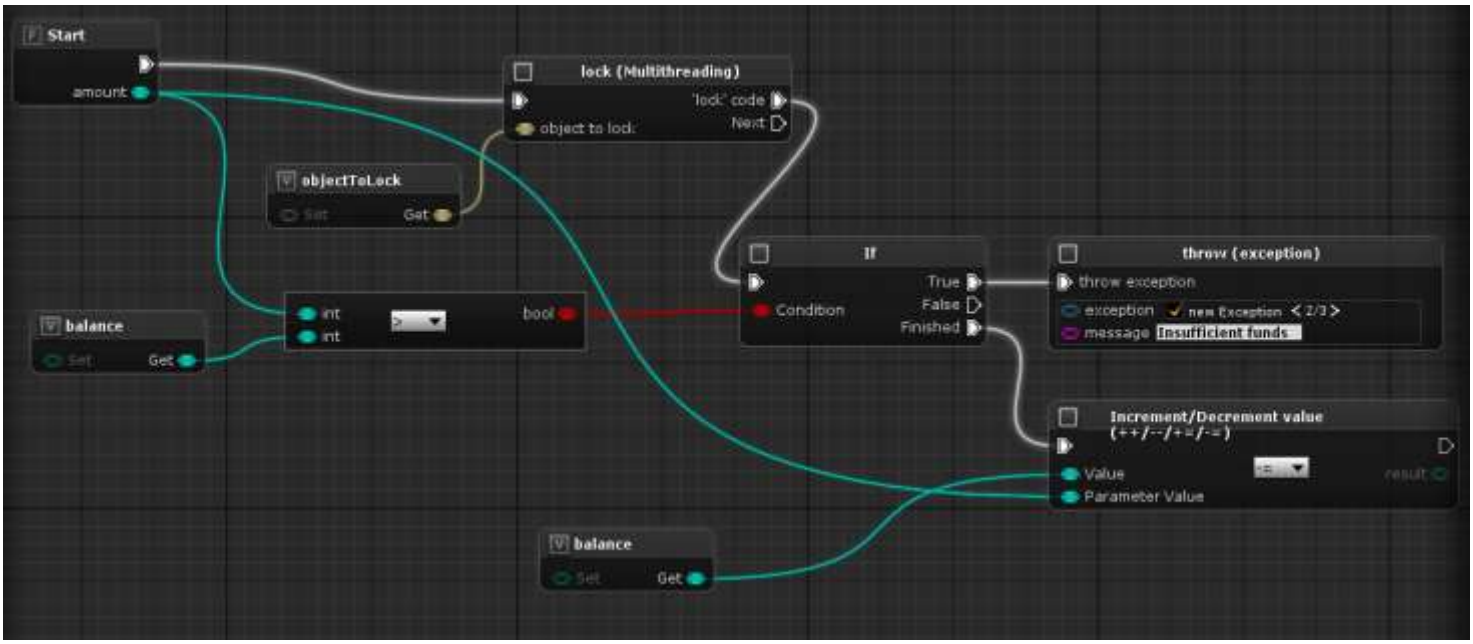
lock (Multithreading)



The **lock (Multithreading)** node is used to ensure that one thread does not enter a critical section of code while another thread is in a critical section. If another thread tries to enter locked code, it will wait, block, until the object is released.

Using example

The following example includes a lock statement. If the other threads try to execute this function they will wait until the current thread unlocks this object:



AND

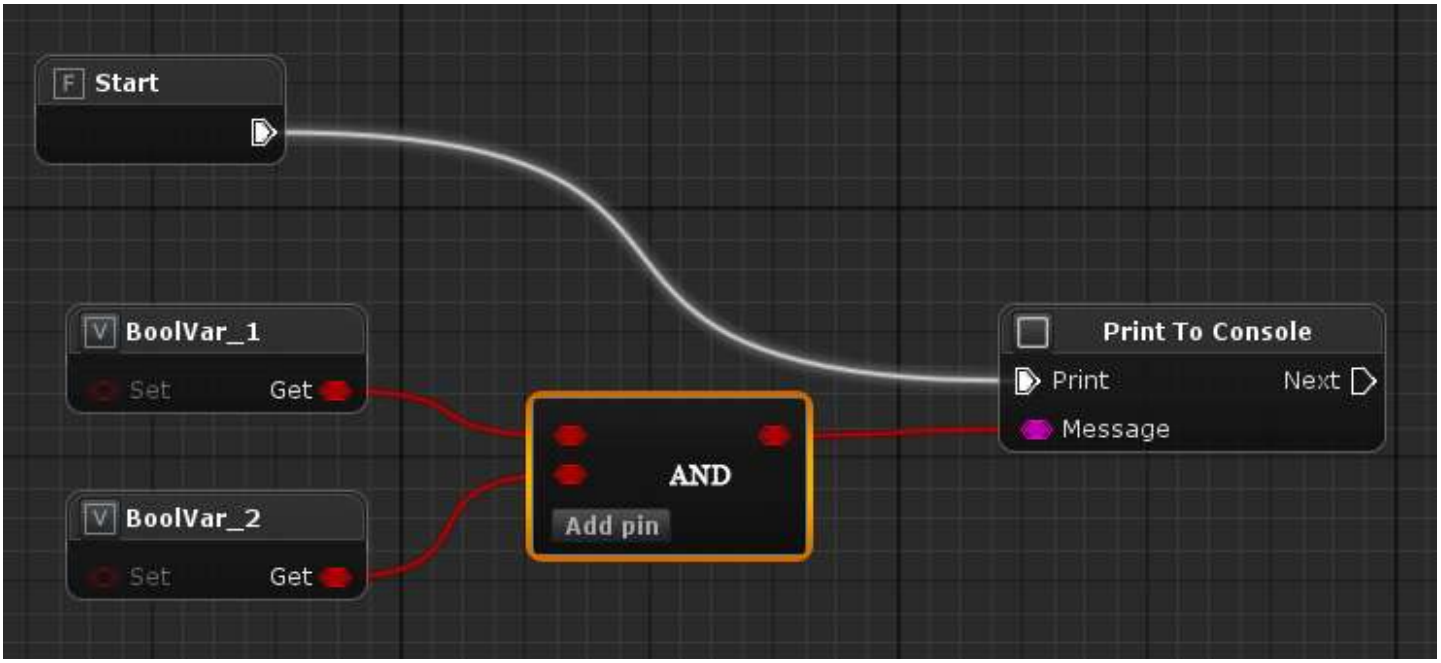
The **And** node is used for the boolean logic operation “And” (&&). The result will be **true** if all IN pin values are true. Otherwise, the result will be **false**.

Add pin - this button allows you to add as many IN pins as you like.

IN pins can be removed by Right-clicking and choosing **Remove Pin**.

Using example

In this example we apply a logical operation to two variables and print the result to the console. The log message will be **true** if both values of the variables are true. Otherwise, the result will be **false**.



OR

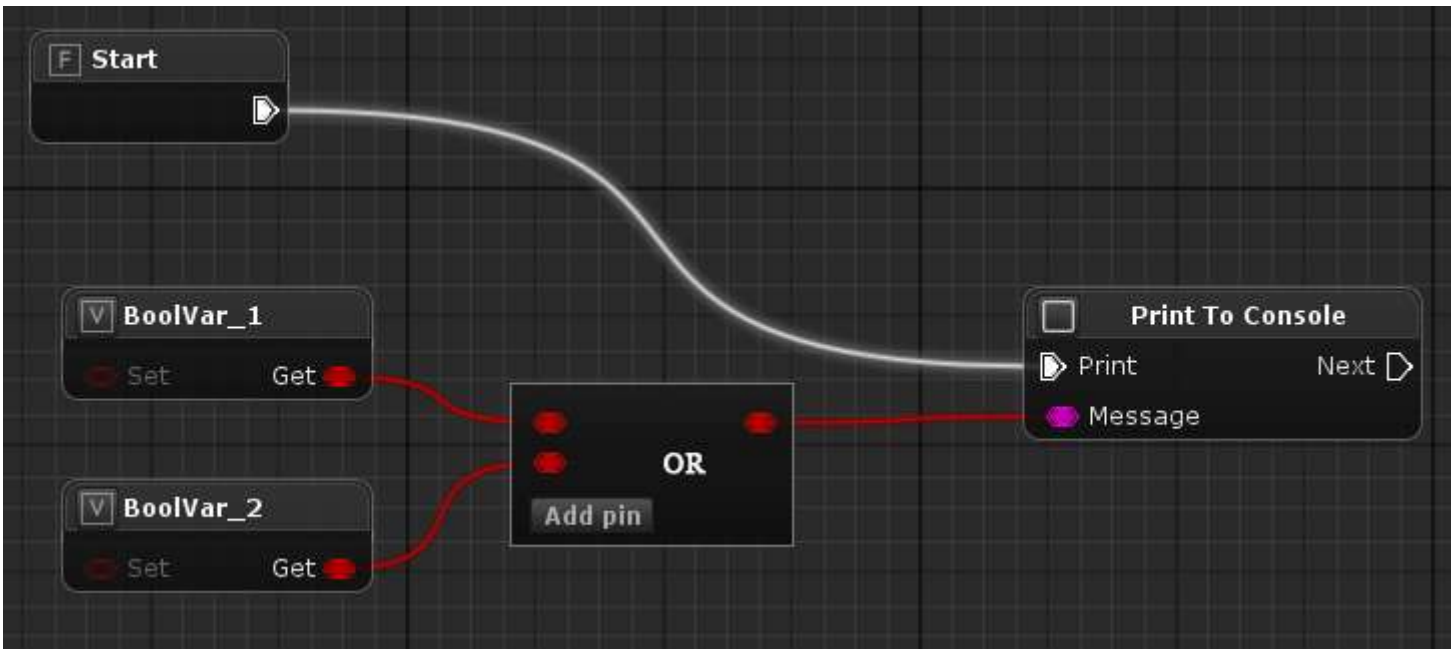
The **Or** node is used for the boolean logic operation “Or” (||), and the result will be **true** if **any** of the IN pin values are true. Otherwise, the result will be **false**.

Add pin - this button allows you to add as many IN pins as you like.

IN pins can be removed by Right-clicking and choosing **Remove Pin**.

Using example

In this example we apply a logical operation to two variables and print the result to the console. The log message will be **true** if any of the IN variables are true. Otherwise, the result will be **false**.

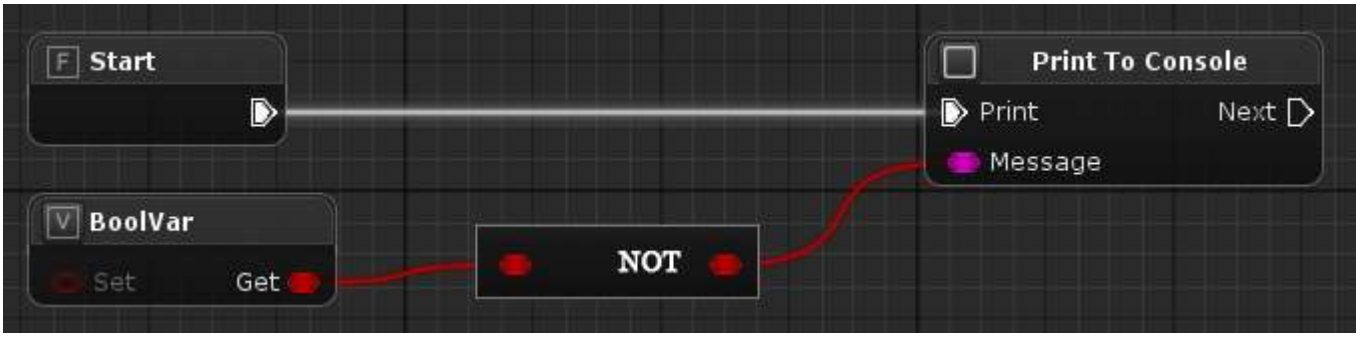


NOT

The **Not** (Inverter) node is used for logical negation operations (!) that invert the boolean value, and the result will be **true** if the IN pin value is **false**. Otherwise, the result will be **false**.

Using example

In this example we apply logical operation to a variable and print the result to the console. The log message will be **true** if the input value of a variable is **false**. Otherwise, the result will be **false**.

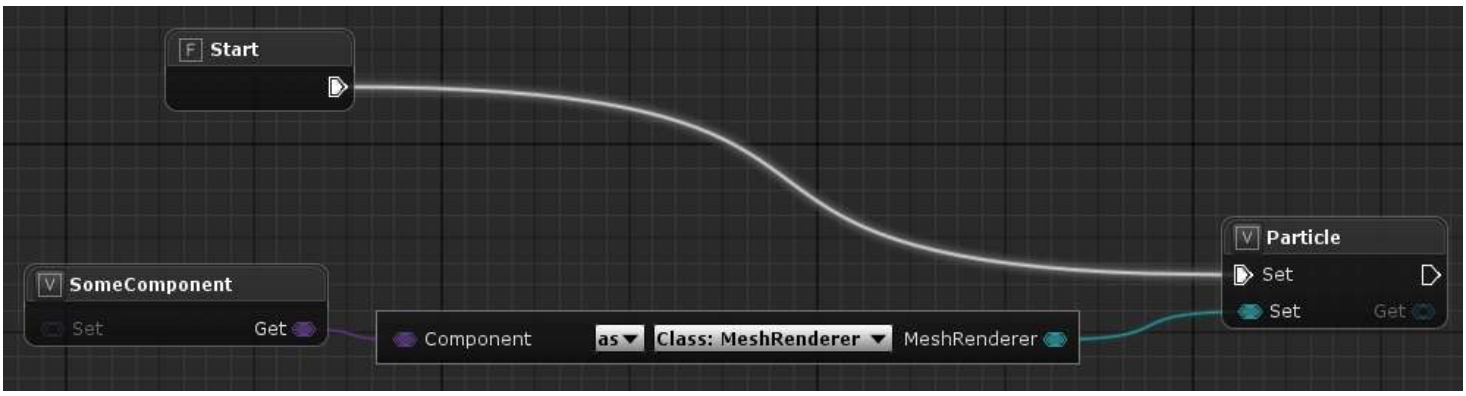


As

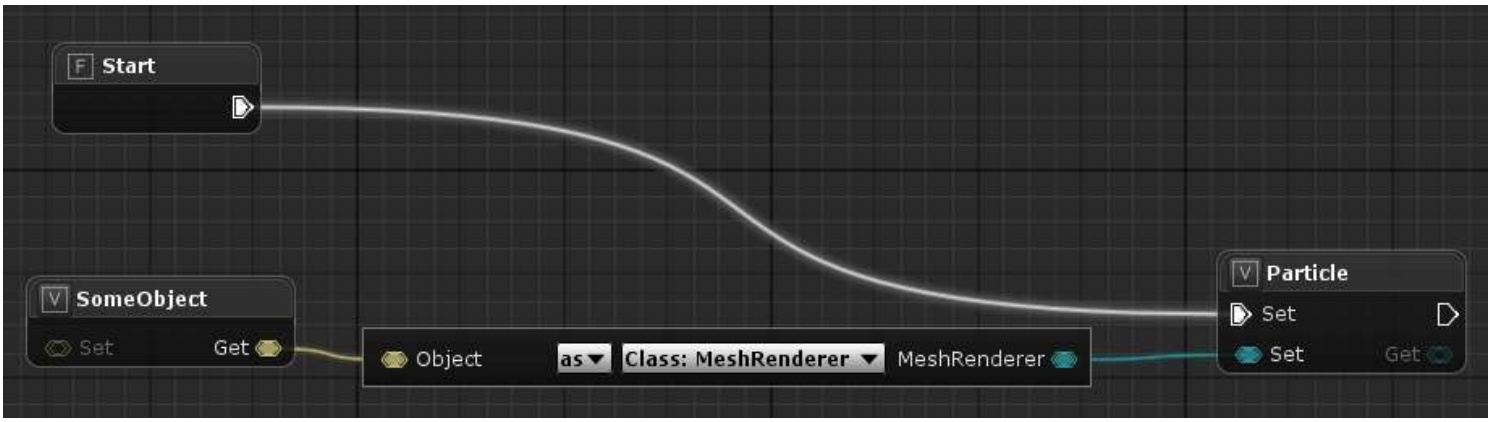
The **As (T)** node is used for performing certain types of conversions between compatible reference types.

Using example

In this example we will convert the **Component** type variable into the **ParticleSystem** type (which is derived from Component). Connect the component variable to the IN pin and the ParticleSystem to the out pin. The rest will be adjusted automatically.



In another example we will convert the **object** variable to a **ParticleSystem**.



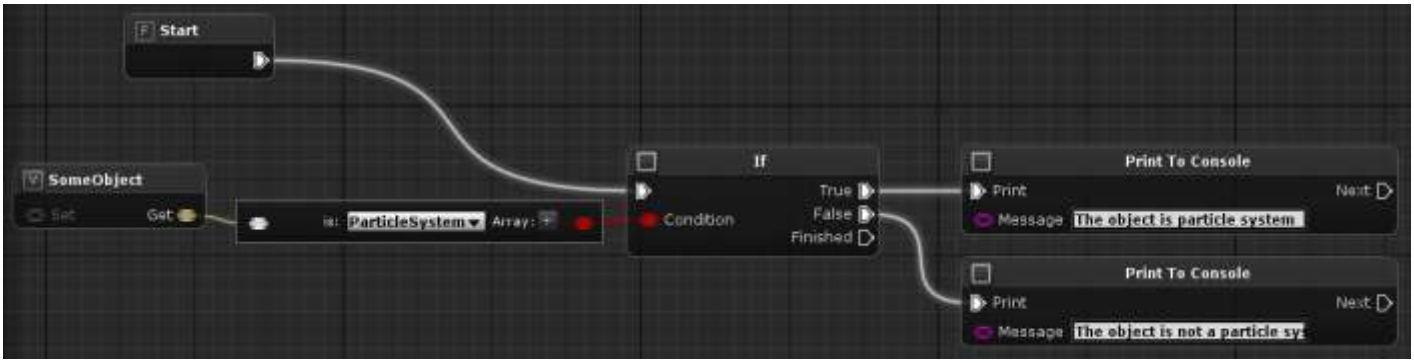
Note: if the object variable doesn't contain the ParticleSystem class instance, the result will be null. You can also check this before conversion using the "**is**" node.

Is

The **Is** node is used to check if the object is compatible with a given type. The result will be **true** if the variable type is not null and is equal to the selected type, Otherwise the result will be false.

Using example

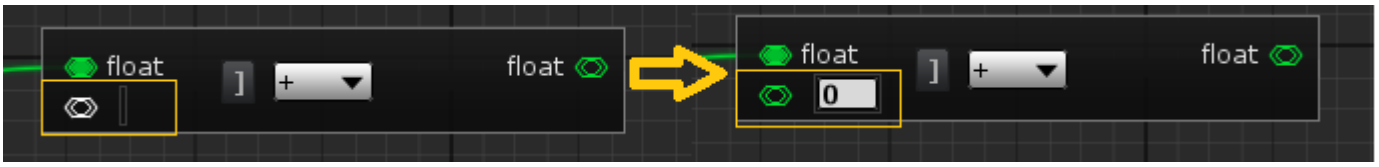
In this example we will check if the **Object** variable is a **ParticleSystem** type:



Math operation

The **Math operation** node is used for performing mathematical and logical operations between numerical or boolean values.

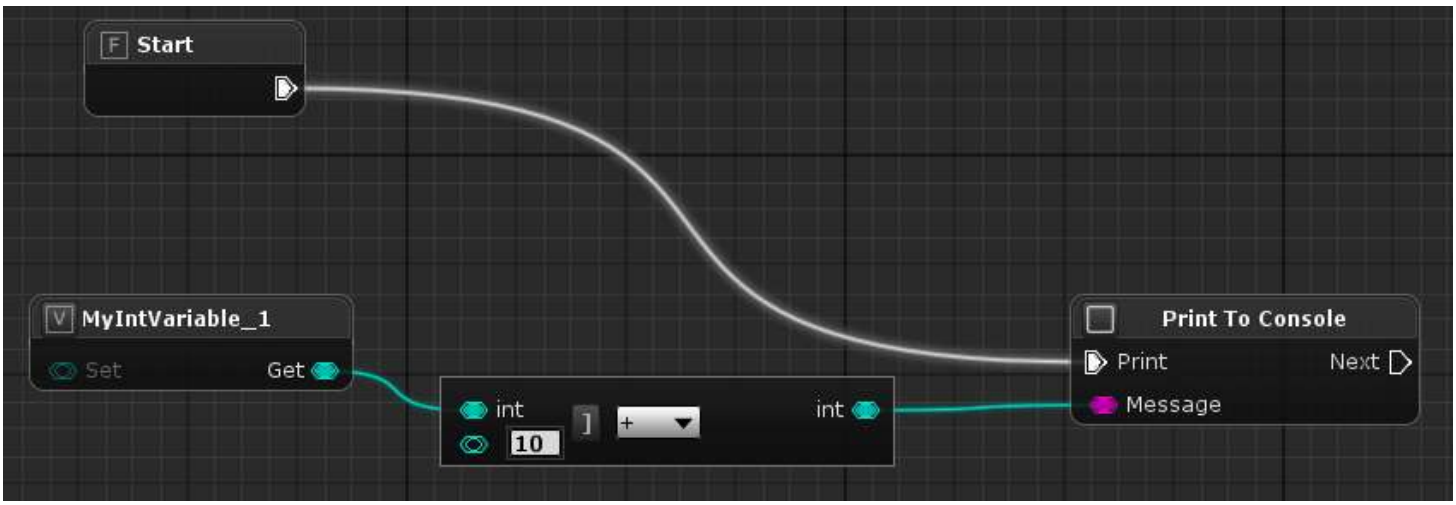
The "**]**" button is used when you want to set the second pin type as connected type:



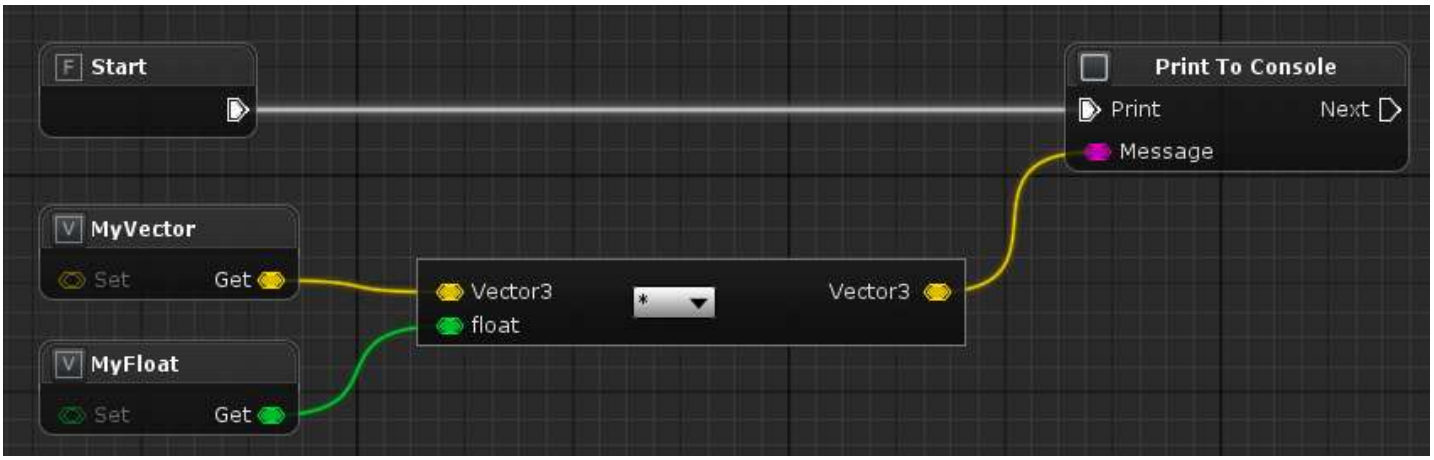
Because sometimes you need to operate with different types of values it makes no sense to set both IN pins types as a connected type.

Using examples

In this example we will increment the "**MyIntVariable_1**" by **10**. Connect this variable to the first IN pin of the Math node and press the "**]**" button. It will set the second pin type to int. After that you will able to write the value **10**:



The second example shows how to multiply a **vector** bya **float** and log the result to the console:

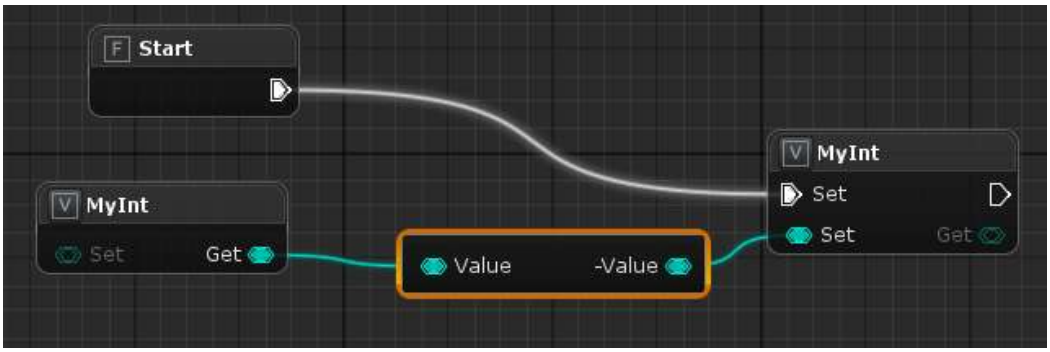


Unary

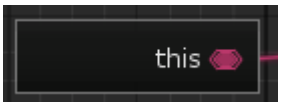
The **unary** node is used for performing "**Unary minus**" operations with numerical variable types. It will change the sign of number as well as we multiply it by **-1** or subtract it from zero.

Using example

In this example we will invert the sign of an integer variable:



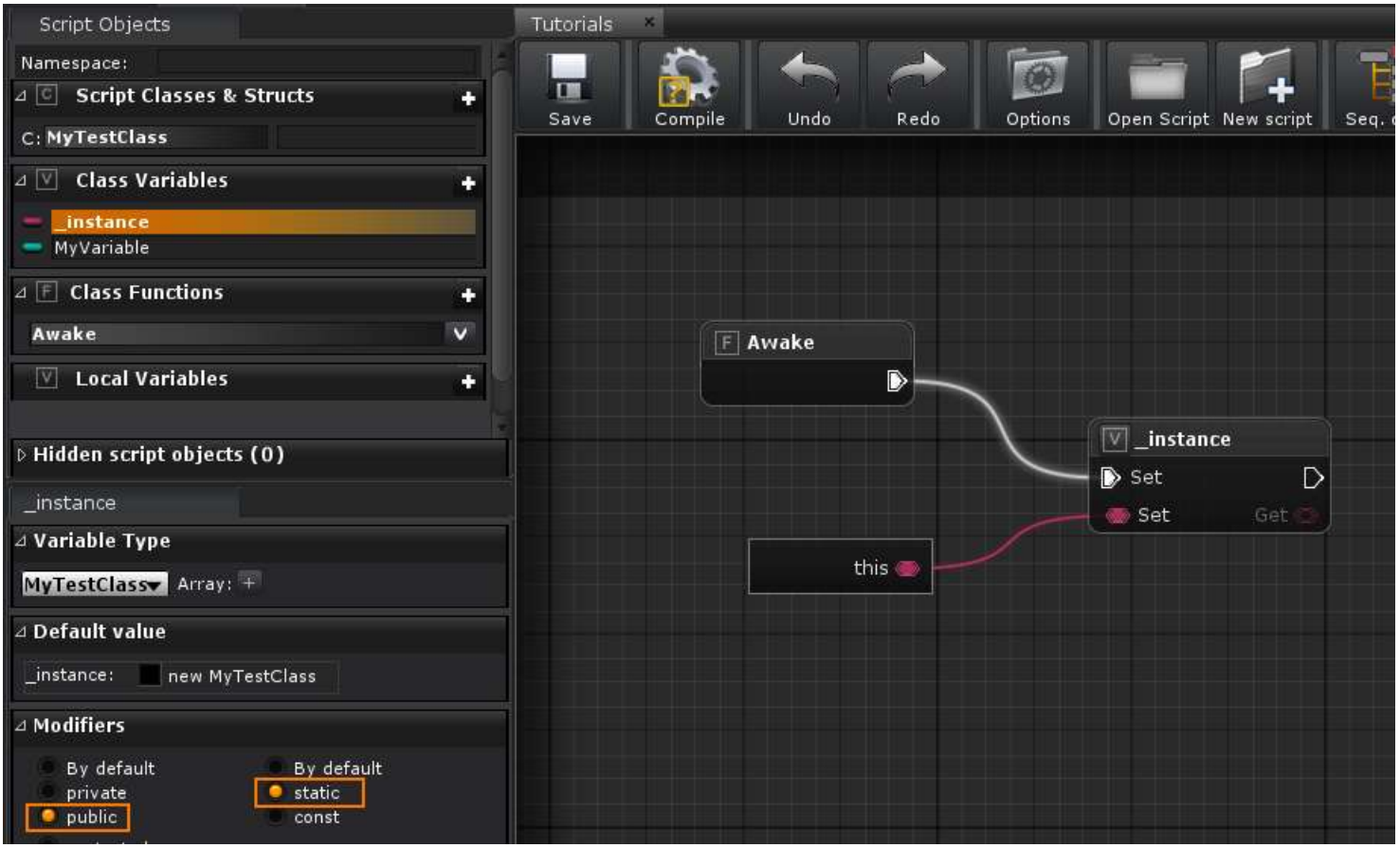
This



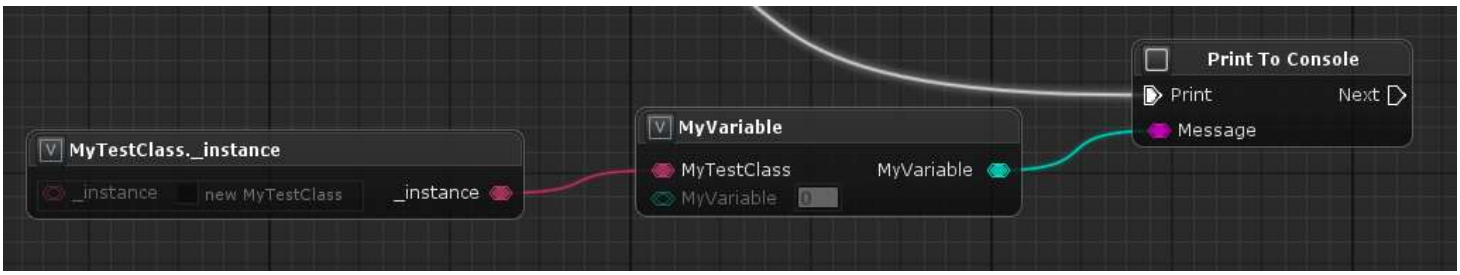
The "**this**" node returns the current class instance.

Using example

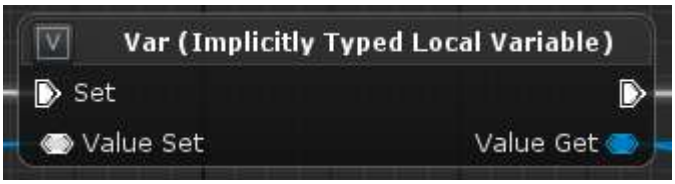
You can use it to set a value of a public static variable to this class:



…and you will have easy access to this class from other classes:



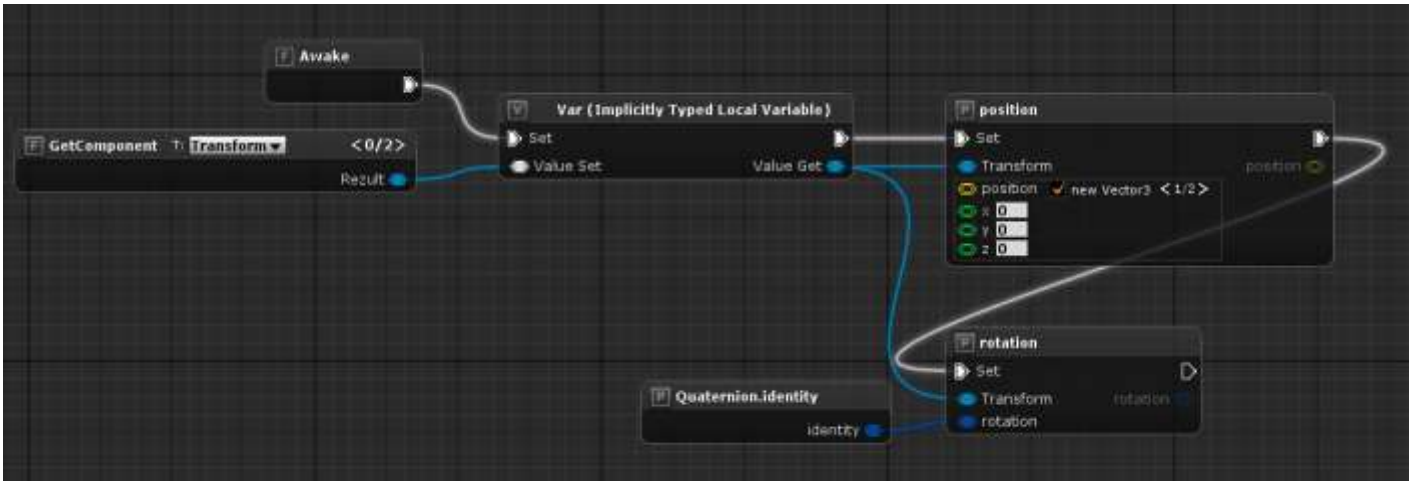
Var



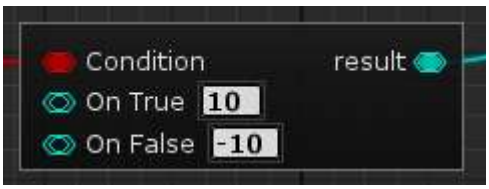
The **Var** node is used for temporarily holding the value (like a local variable) instead of creating the local variable. Also, you don't need to set the type of the variable, it will be defined automatically.

Using example

In this example we will temporary hold the **Transform** value from "**GetComponent**" node and use it in the next nodes:



Conditional expression

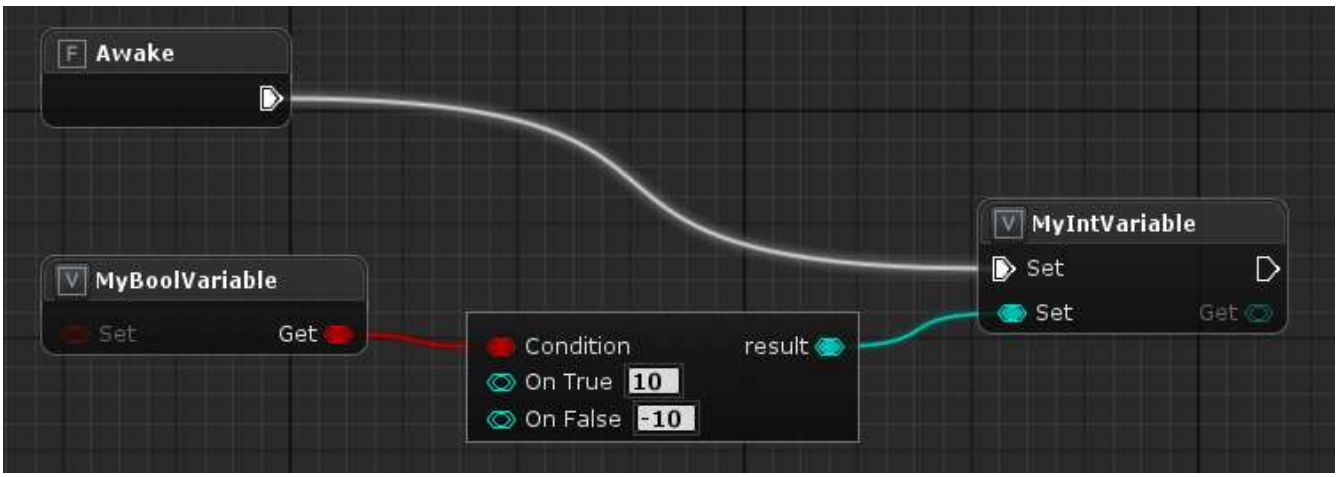


The **Conditional expression** node returns one of two values depending on the value of a Boolean condition.

Using example

In this example we set the **int** value of the **MyIntVariable** depending on the bool value of the **MyBoolVariable** value.

If the bool value is **true**- the result will be "**10**" (**On True** pin), otherwise it will be "**-10**" (**On False** pin):

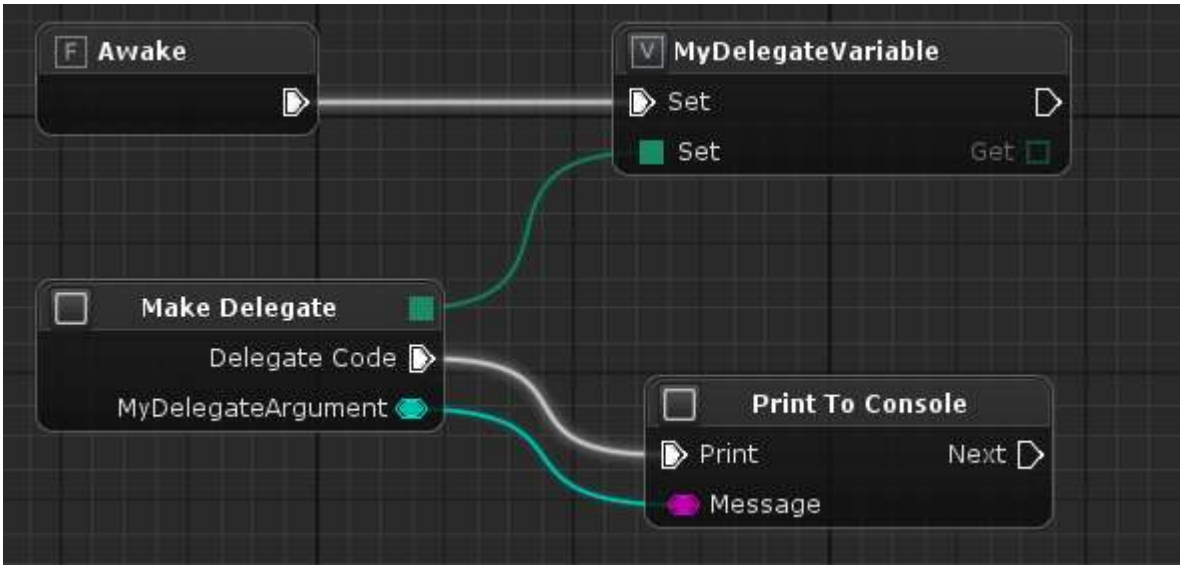


Make Delegate

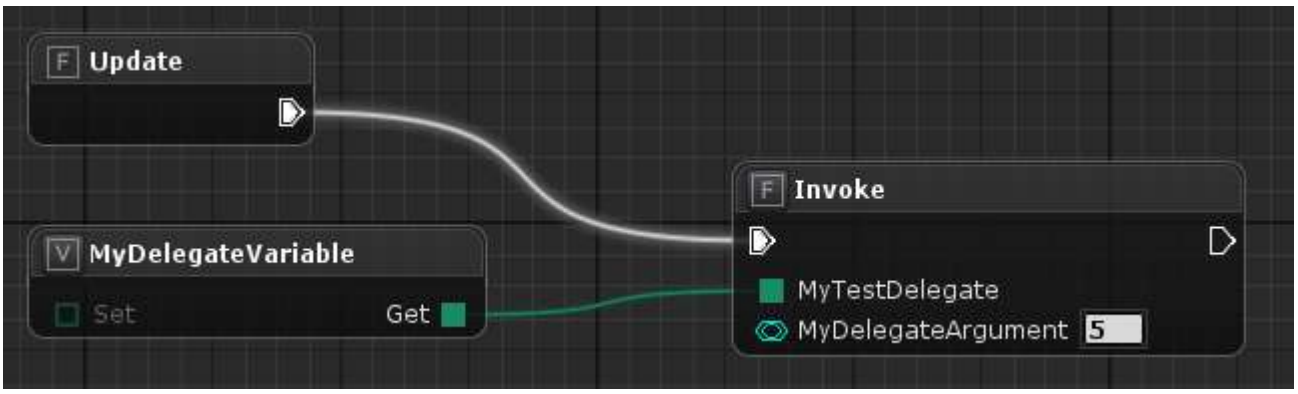
The **Make Delegate** node is used for initialization of a delegate variable and setting the custom code into it.

Using example

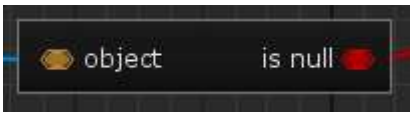
In this example we set the code for a delegate variable:



After that we can execute this code by invoking the delegate variable and it will print the value of the delegate argument to the console:



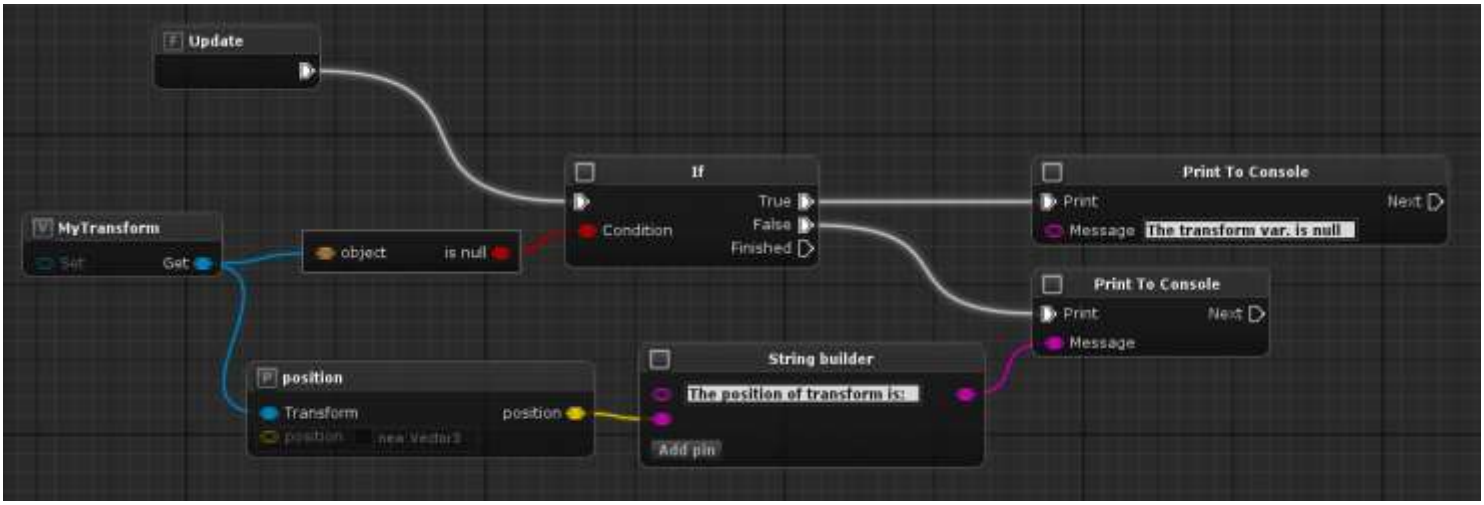
Is null



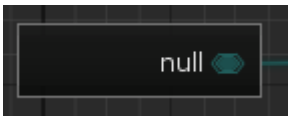
The **Is null** node return **true** if the connected variable is **null**.

Using example

In this example we will check the **MyTransform** variable and if it is not null we print its position to the console:



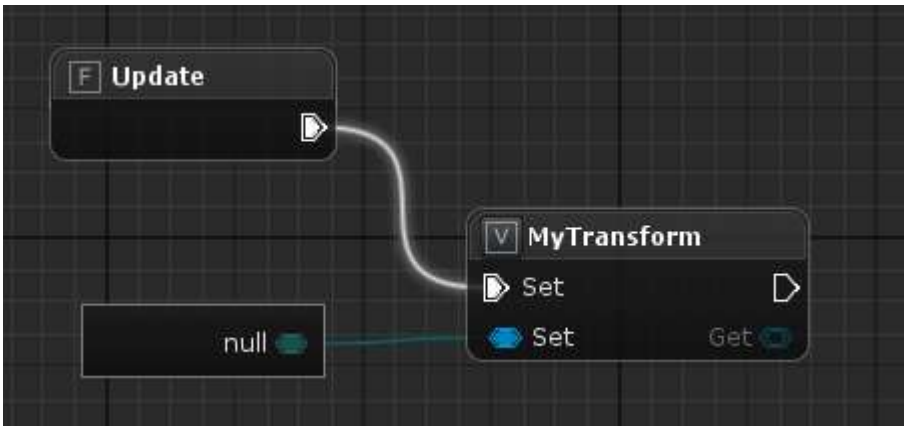
Null



The **null** node returns a **null** value.

Using example

The **null** node can be used to set the value of a variable to a **null** value:

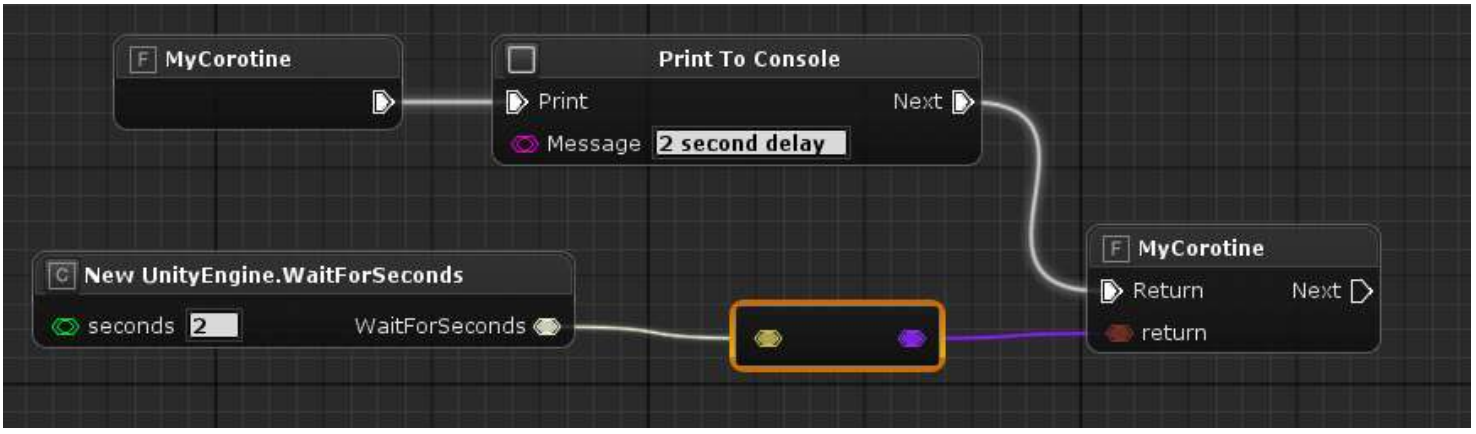


yield return

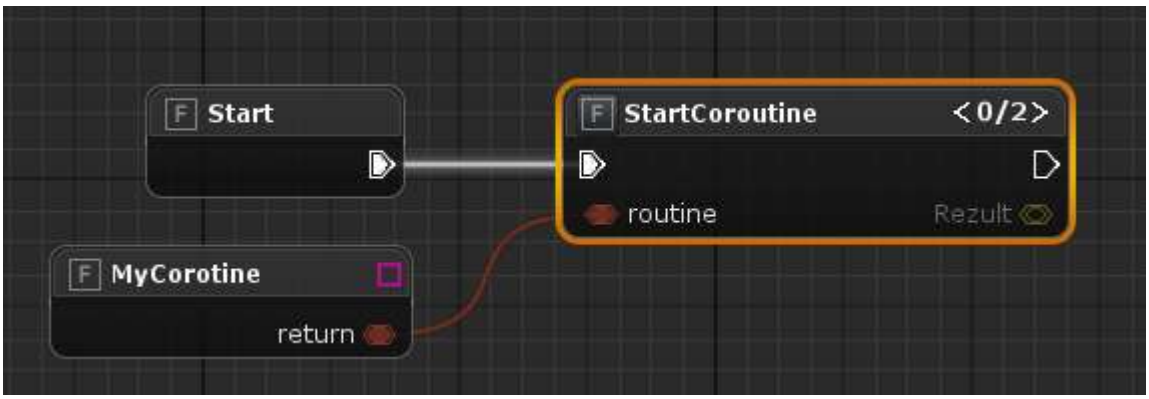
The **yield return** node is used in coroutines to convert the yield instruction to an **IEnumerator**, which controls the behaviour of the coroutine.

Using example

In this example we will use the **yield return** node in the Coroutine function to convert the **WaitForSeconds** instruction into an **IEnumerator** and return it:



Then we can start our coroutine by using the **StartCoroutine** method (which was inherited from the Monobehaviour class):



Make array

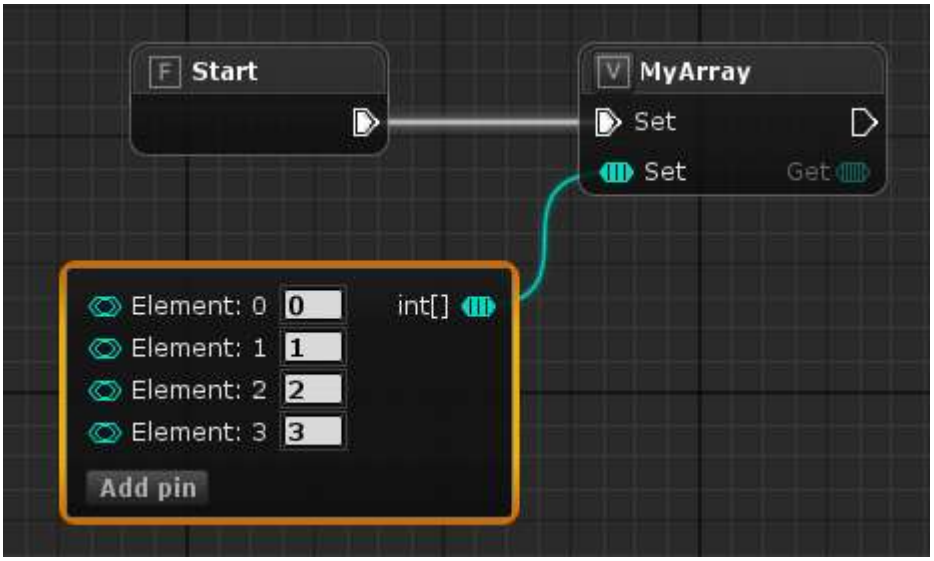
The **Make array** node is used for the initialization of arrays.

Add pin - this button allows you to add as many IN pins as you like.

IN pins can be removed by Right-clicking and choosing **Remove Pin**.

Using example

Connect the out pin of the node to any array IN pin and set the values of the elements:

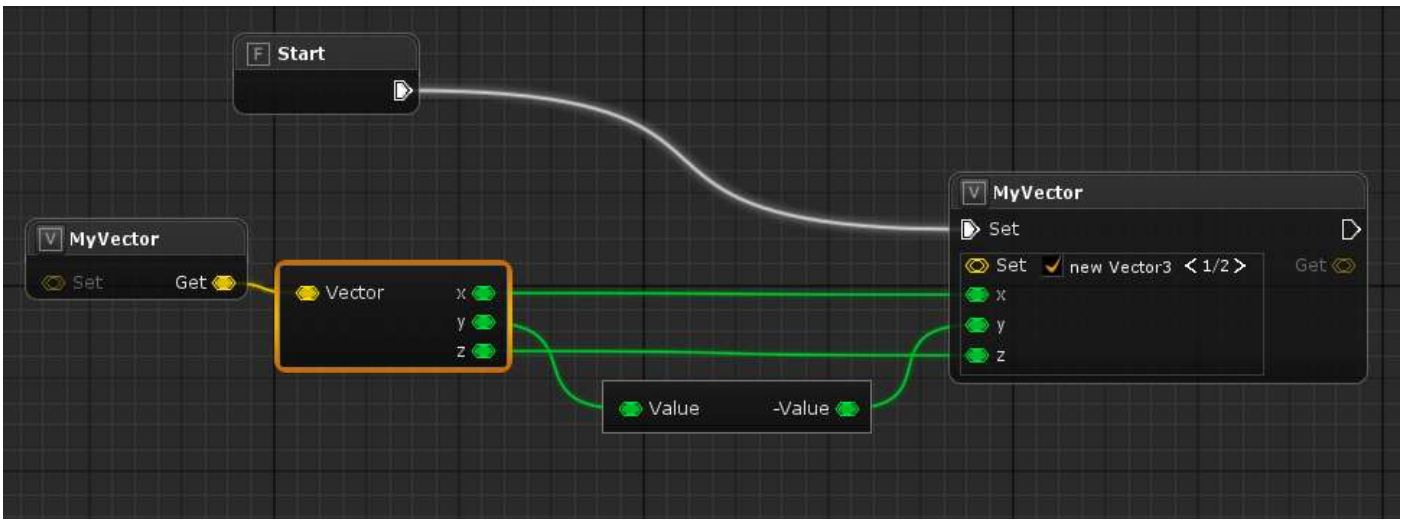


Break vector

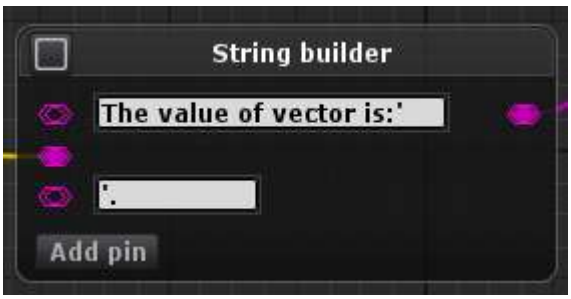
The **Break vector** node breaks a vector apart into its **X**, **Y**, **Z** float values.

Using example

In this example we break the vector of variable "**MyVector**" into **X**, **Y** and **Z** values and making a new vector with the **Y** value inverted.



String builder



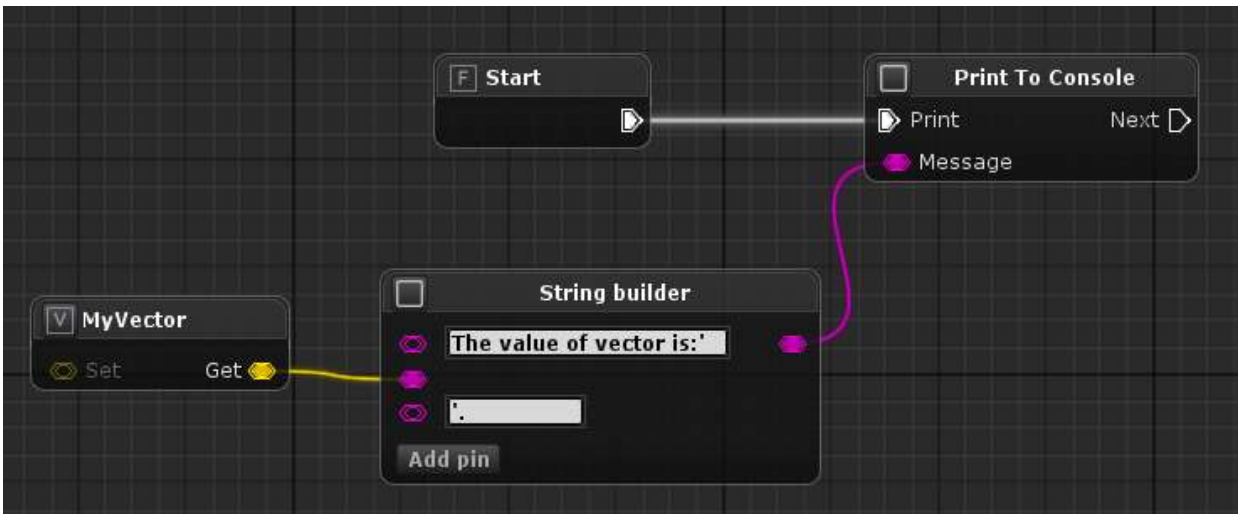
The **String builder** node is used for making a string from separate strings.

Add pin - this button allows you to add as many IN pins as you like.

IN pins can be removed by Right-clicking and choosing **Remove Pin**.

Using example

In this example we make a string message from the separate strings and a value of a vector:



Lambda Expression

The **Lambda Expression delegate** node is used to create an anonymous function that you can use to create delegates. By using this node, you can write local functions that can be passed as arguments or returned as the value of function calls.

Using example

In this example we will use the **Lambda Expression delegate** node to create a search filter for the "**FindAll**" node. After connecting the **Lambda Expression delegate** node to a "**match**" pin it will be initialized and we can use the delegate argument for making a filter based on a **Math Operations Node**. The output must be a **bool** value which we can connect to a "**result**" pin of the lambda delegate node. All values which are more than 10 will pass the filter: