LECTURE 7

Advanced Blueprint Concepts 2

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Present flow control nodes
- Show how to iterate an array using a ForEachLoop node
- Show how to work with strings
- Present the Math Expression node
- Explain random number functions

Outcomes

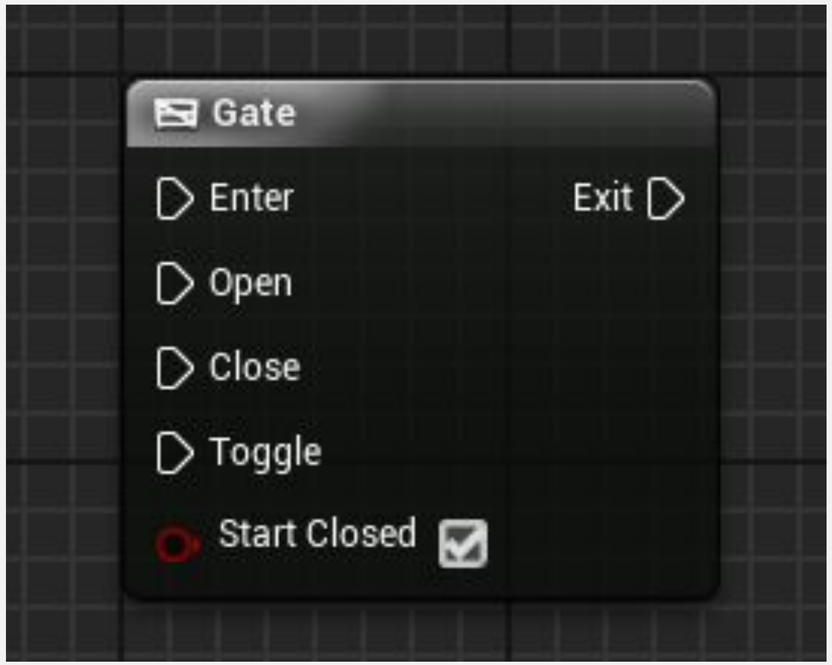
By the end of this lecture you will be able to

- Choose the best flow control node for each situation
- Access the array elements using a ForEachLoop node
- Format and convert strings
- Create expressions with the Math Expression node
- Use random numbers and Random Stream variables

FLOW CONTROL

GATE NODE

The **Gate** node is a flow control node that can be open or closed. If it is open, it allows the execution of the actions related to the **Exit** pin.



GATE NODE: INPUT AND OUTPUT

Input

- **Enter**: Execution pin that receives the current flow of execution.
- **Open**: Execution pin that changes the state of the gate to “open”.
- **Close**: Execution pin that changes the state of the gate to “closed”.
- **Toggle**: Execution pin that toggles the current state of the gate.
- **Start Closed**: Boolean variable that determines if the **Gate** node should start running in the “closed” state.

Output

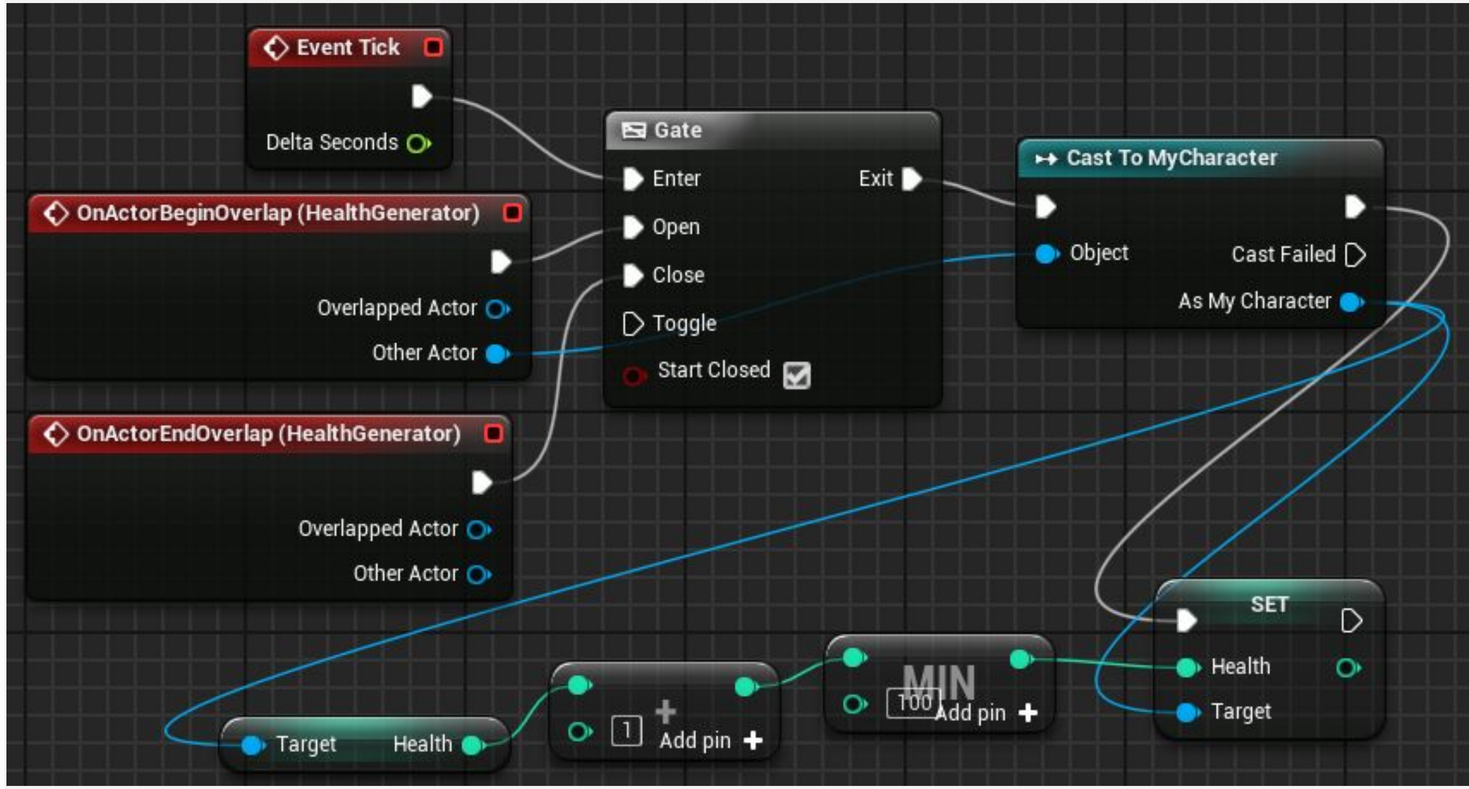
- **Exit**: Execution pin that will execute if the gate is open.

GATE NODE: EXAMPLE

In the example to the right, there is an Actor called “**HealthGenerator**”. When the player is colliding with this Actor, their health will be restored slowly with every **Tick** event.

If the player stops colliding with **HealthGenerator**, the gate will be closed and the actions that restore health will no longer be performed.

The **Min** action is used so that the value of the **Health** variable is never greater than “**100**”.

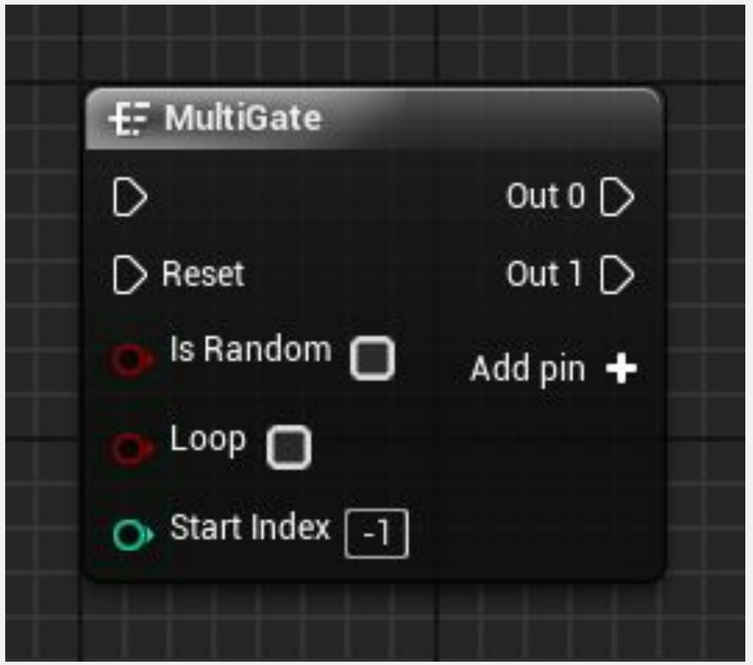


MULTIGATE NODE

A **MultiGate** node can have multiple output pins. At each execution of the **MultiGate**, only one of the output pins is executed. The order in which the output pins are executed may be sequential or random.

When all the output pins are executed and if the **Loop** option is not selected, the **MultiGate** node will stop executing the output pins. In order for the **Multigate** to run the output pins again, the **Reset** pin must be triggered.

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



MULTIGATE NODE: INPUT

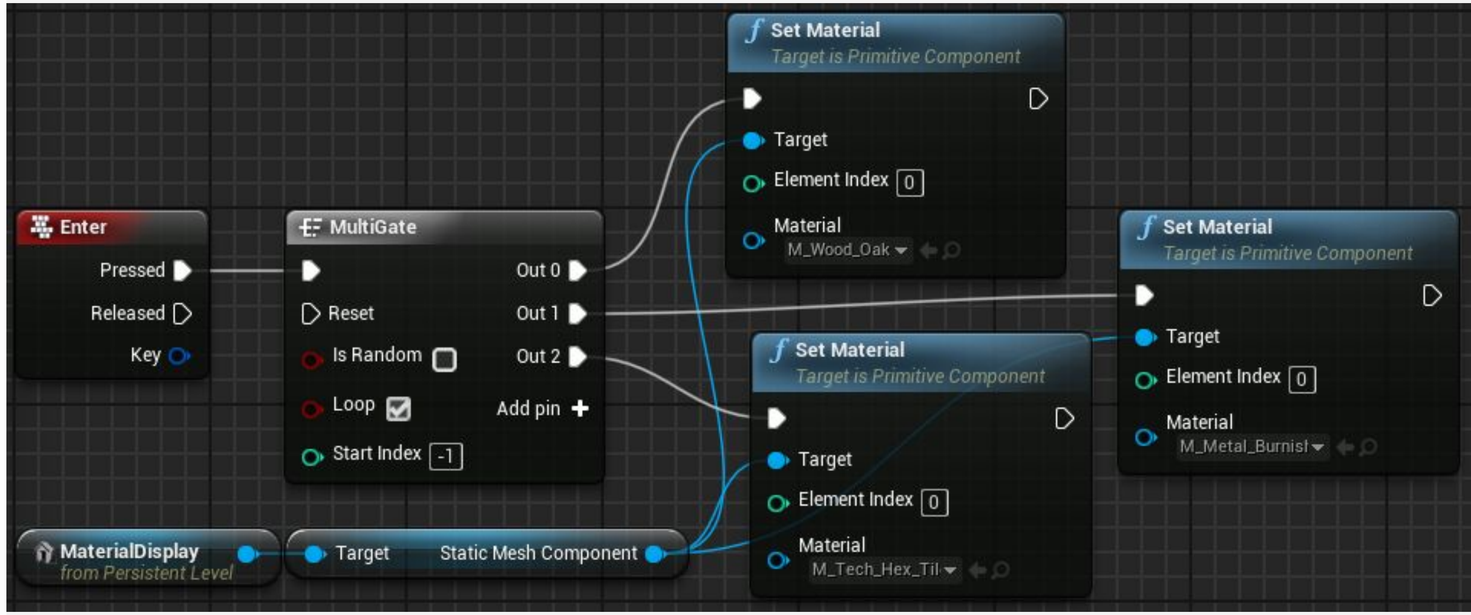
- **Reset**: Execution pin used to reset the **MultiGate** node and allow new executions of the output pins.
- **Is Random**: Boolean variable. If the value is “**true**”, the order of execution of the output pins is random.
- **Loop**: Boolean variable. If the value is “**true**”, the **MultiGate** continues to execute the output pins after the last output pin is executed.
- **Start Index**: Takes in an integer value indicating the first output pin to be executed.

MULTIGATE NODE: EXAMPLE

In the example on the right, there is an object in the Level called “**MaterialDisplay**” whose function is to display various Materials for the user.

When the **Enter** key is pressed, a **MultiGate** node is used to define a different Material at each execution.

Because the **Loop** parameter is checked, after the last output pin is executed, the **MultiGate** will continue executing the output pins, beginning with the first output pin.



DO ONCE NODE

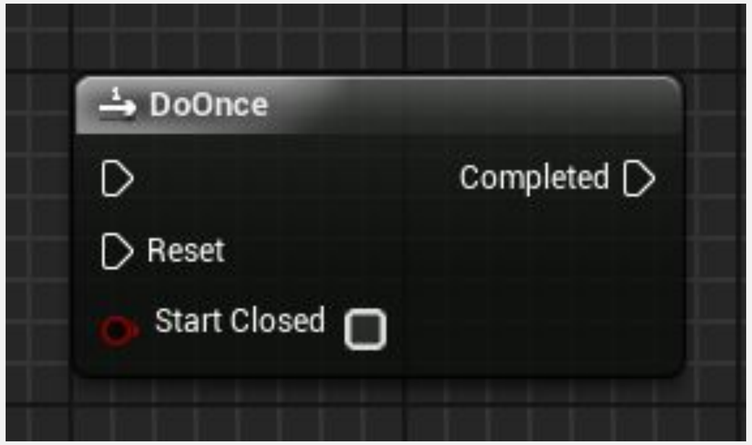
The **DoOnce** node performs the actions attached to the output pin only once.

After its first run, if the **DoOnce** node is called again, its output pin will not run.

In order for the **DoOnce** node to be able to execute the output pin again, the **Reset** pin needs to be triggered.

Input

- **Reset**: Execution pin used to allow the **DoOnce** node to run the output pin.
- **Start Closed**: Boolean variable. If the value is “**true**”, the **DoOnce** node needs to be reset to allow the first run.

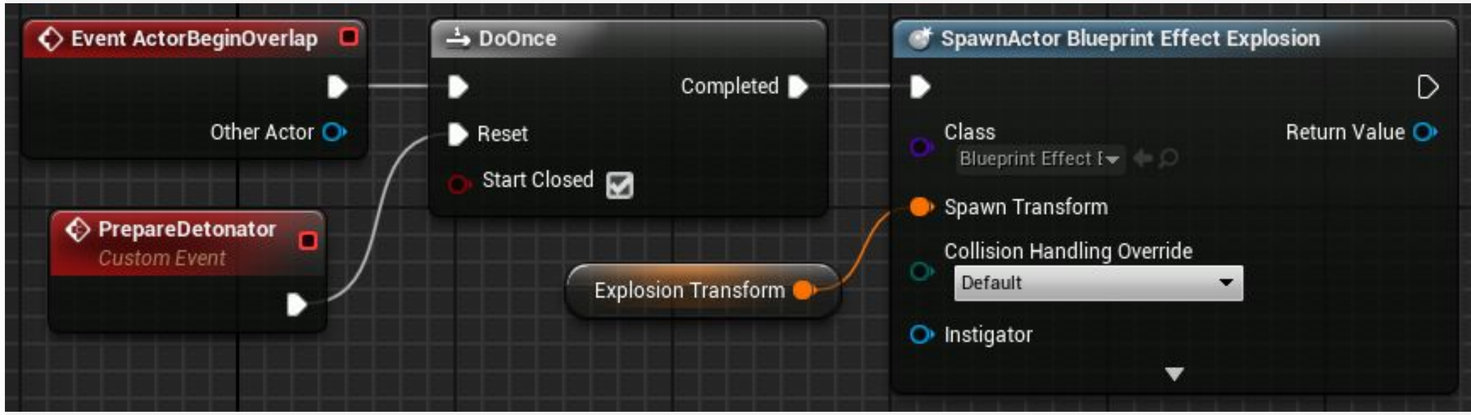


DO ONCE NODE: EXAMPLE

In the example on the right, there is a detonator that generates an explosion when the player collides with it. This detonator utilizes the **DoOnce** action. The **Start Closed** property is checked, indicating that the detonator starts disarmed.

The **PrepareDetonator** event needs to be executed to trigger the **Reset** pin of the **DoOnce** action.

After the **Reset** pin is triggered, if there is a collision with the detonator the explosion will be created. To allow a new explosion, the **PrepareDetonator** event needs to be executed again.



DO N NODE

The **Do N** node is similar to the **DoOnce** node, but instead of executing only once, the actions connected to the output pin can execute multiple times.

After the set number of executions have completed, the actions of the output pin will only be executed again if the **Reset** pin is triggered.

Input

- **N**: Sets the number of times the output pin actions can be executed.
- **Reset**: Execution pin used to reset the **Do N** count and allow new executions of the output pin.

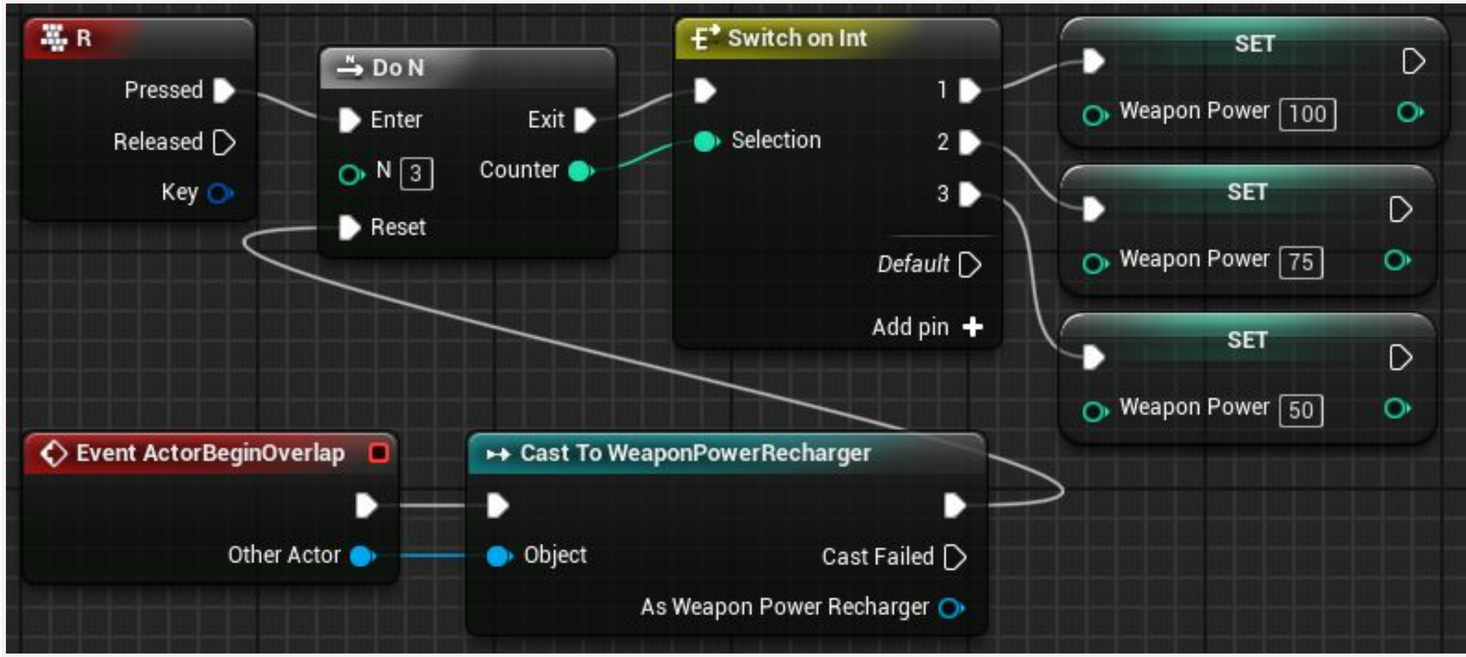
Output

- **Counter**: Outputs an integer value indicating the current execution number.

DO N NODE: EXAMPLE

In the example to the right, a player has a special weapon. When the weapon is fully charged, its power is at 100% capacity. The player can press the “**R**” key to recharge this weapon. The weapon can be recharged three times.

The first recharge leaves the weapon fully charged; the second recharge leaves it at 75% capacity; and the third recharge recovers only half of the weapon’s capacity. To recharge the weapon again, the player needs to collect an item of type “**WeaponPowerRecharger**”.



FLIP FLOP NODE

The **FlipFlop** node has two output pins identified as “**A**” and “**B**”. When the **FlipFlop** is executed, only one of the output pins is executed. On the next run, only the other pin will be executed.

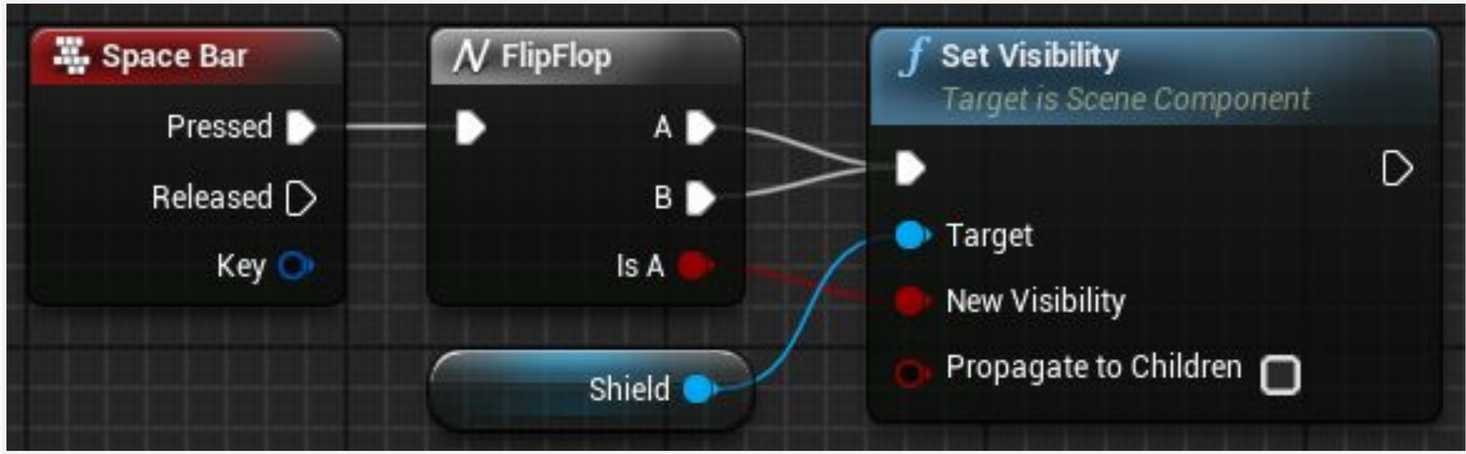
Output

- **A**: Execution pin that will execute if the value of the **Is A** pin is “**true**”.
- **B**: Execution pin that will execute if the value of the **Is A** pin is “**false**”.
- **Is A**: Boolean variable. If the value is “**true**”, pin **A** is running. If “**false**”, pin **B** is running.

FLIP FLOP NODE: EXAMPLE

In the example on the right, the **FlipFlop** node is being used to show or hide a shield when the **space bar** is pressed.

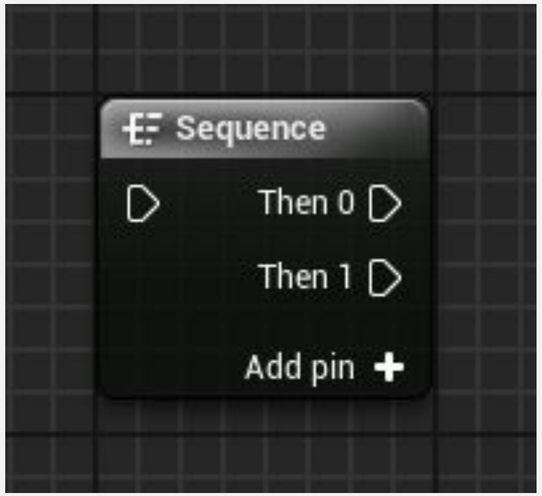
The value of the **Is A** output pin is being used to determine the visibility of the shield.



SEQUENCE NODE

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

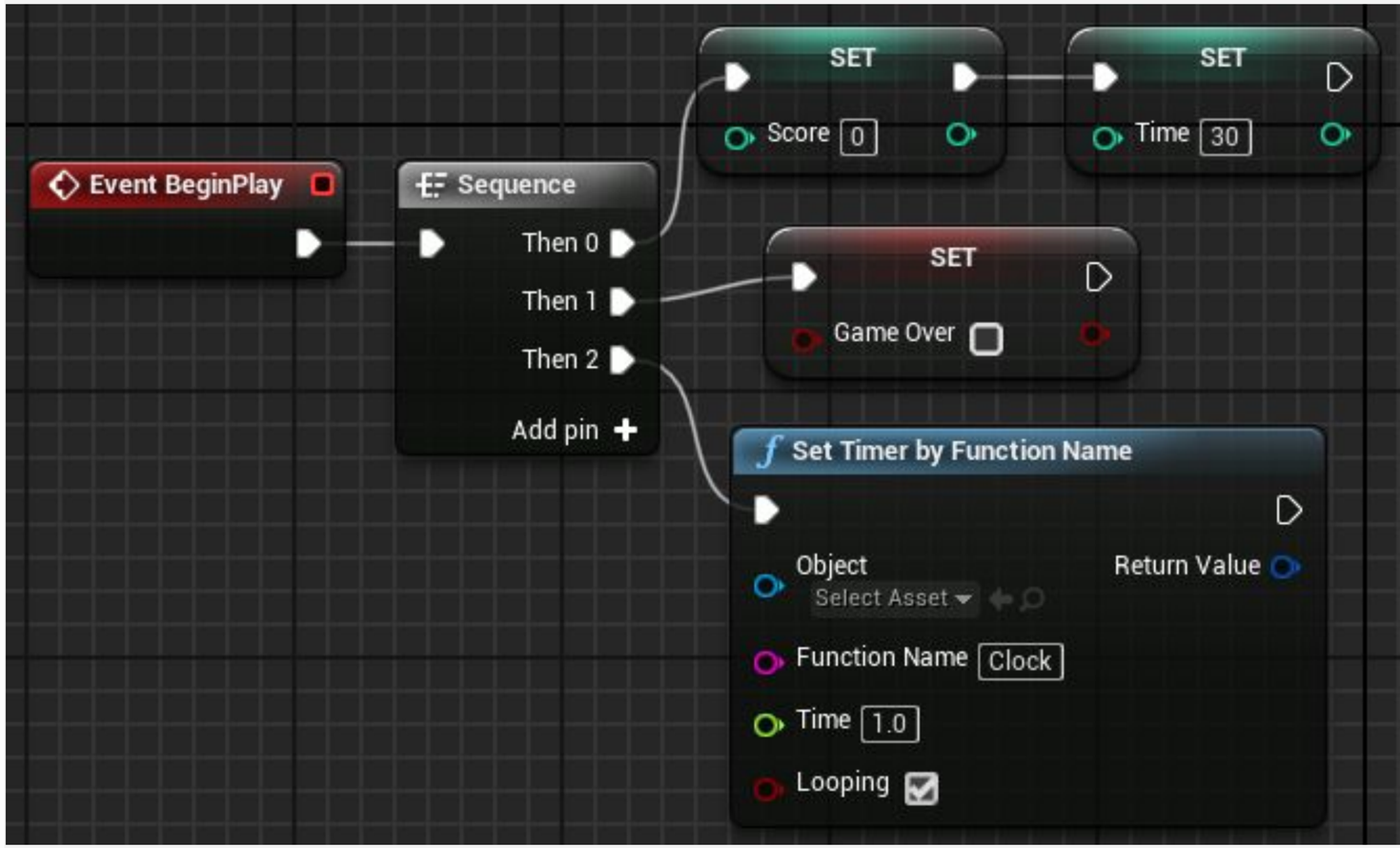
Output pins can be added using the **Add pin +** option. To remove a pin, right-click on the pin and choose the **Remove execution pin** option.



SEQUENCE NODE: EXAMPLE

In the example on the right, the **Sequence** node is used to organize the actions that will be executed after the **BeginPlay** event.

Instead of one execution line being used for all actions, the **Sequence** node is used to group the actions by similarity.



FOR EACH LOOP NODE

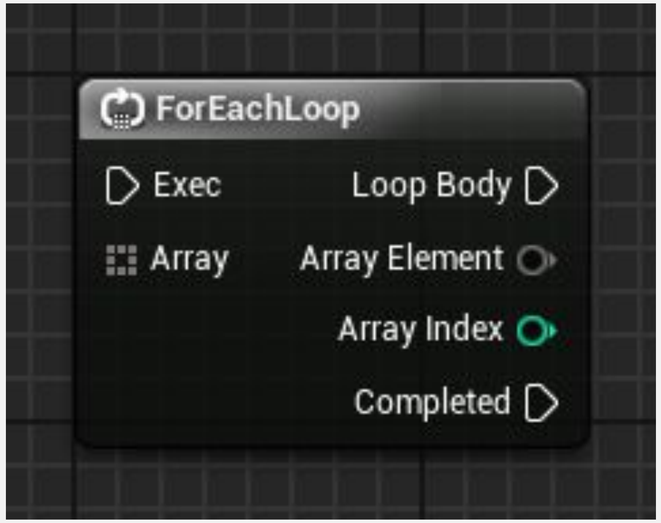
The **ForEachLoop** node takes in an array as an input parameter and performs the set of actions that are associated with the **Loop Body** output pin for each element of the array that can be obtained from the **Array Element** output pin. After that, the execution flow is directed to the **Completed** output pin.

Input

- **Array**: Takes in an array containing the elements that will be used in the loop.

Output

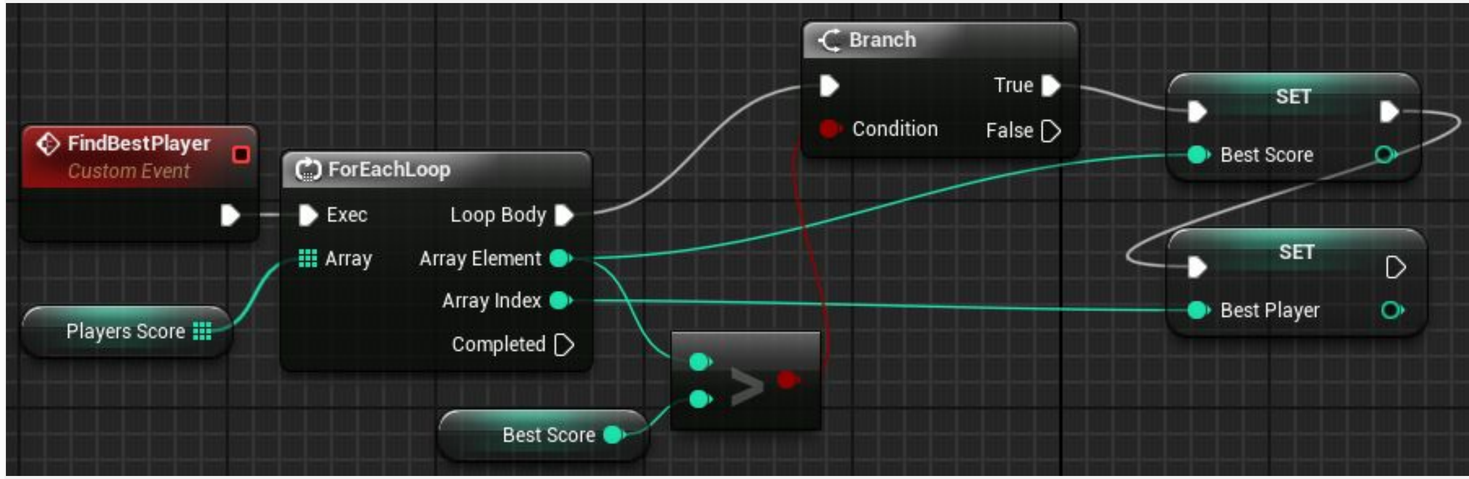
- **Array Element**: Outputs a reference to the current array element.
- **Array Index**: Outputs the index of the current array element.



FOR EACH LOOP NODE: EXAMPLE

In the example on the right, a **ForEachLoop** node is used to iterate through an array that contains the scores of the players.

For each value, a test is done to check whether it represents the highest score. If “**true**”, the value is stored in the **Best Score** variable and the player’s index is stored in the **Best Player** variable.



SWITCH ON INT NODE

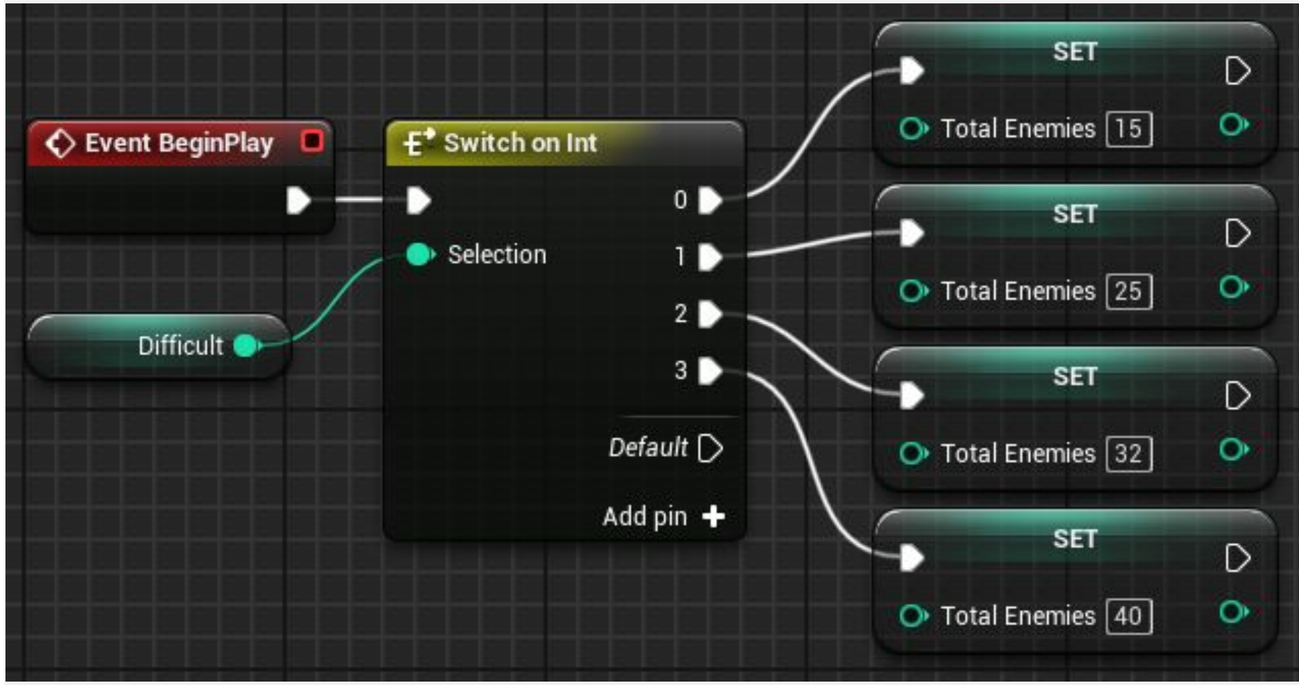
The **Switch on Int** node determines the flow of execution according to the integer input value. Output pins are added using the **Add pin +** option.

Input

- **Selection**: Takes in an integer value that determines which output pin will execute. If the value is not found, the **Default** pin will execute.

Example

In the example on the right, the difficulty of a game is stored in an integer variable named “**Difficult**” that can have values ranging from “**0**” to “**3**”. The total number of enemies is set according to the difficulty.



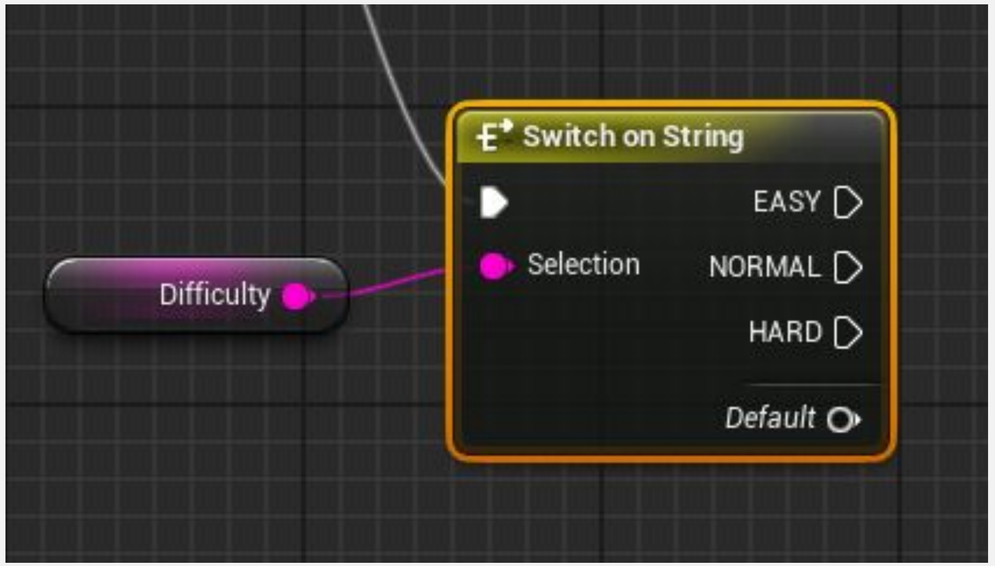
SWITCH ON STRING NODE

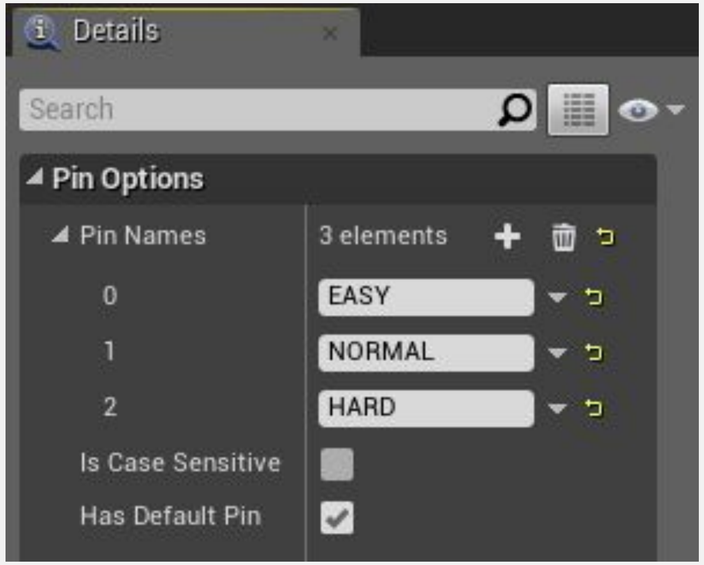
The **Switch on String** node determines the flow of execution according to a string input value. The input string is compared to each of the pin names, and if it matches, the pin will execute.

The output values must be added in the Details panel for the **Switch on String** node under “**Pin Options > Pin Names**”, as shown in the bottom image on the right.

Input

- **Selection**: Takes in a string value that determines the output.



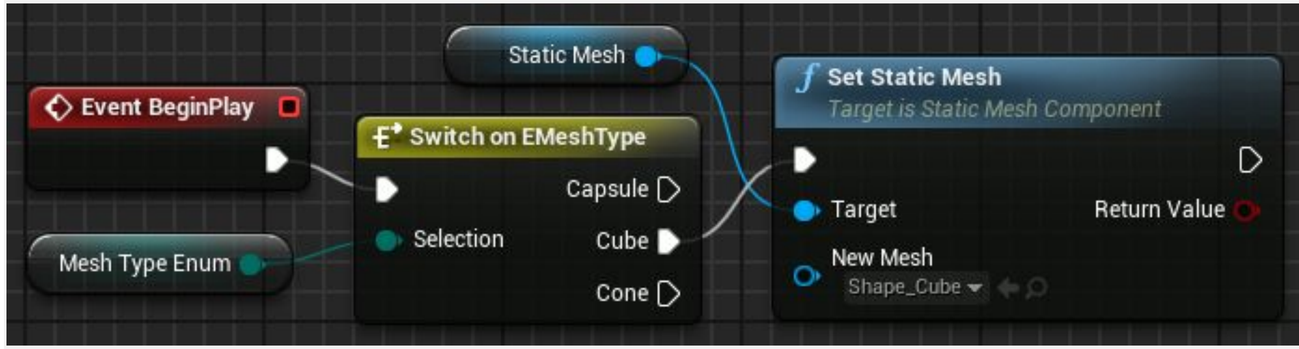


SWITCH ON ENUM NODE

The **Switch on Enum** node determines the flow of execution according to the enumeration input value. For each enumeration, there is a switch node equivalent.

Example

In the example on the right, the Static Mesh is chosen based on the value of the enumeration variable.



STRINGS / TEXT

FORMAT TEXT NODE

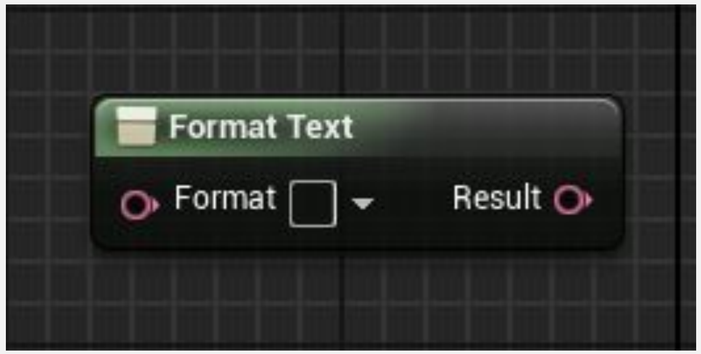
The **Format Text** node builds a text based on parameters that can be specified in the **Format** parameter.

Input

- **Format**: Takes in the text that will be part of the final result. To set parameters, simply put a name between the delimiters { } for each parameter.
- **Parameters defined in “Format”**: For each pair of curly braces, a new input parameter is generated with the name that is between the braces.

Output

- **Result**: Outputs the final text built with the values of the **Format** parameter and the other parameters.



FORMAT TEXT NODE: EXAMPLE

In the example on the right, a text will appear at the end of a match that displays the result. This text contains the values of four variables related to the names and scores of two players.

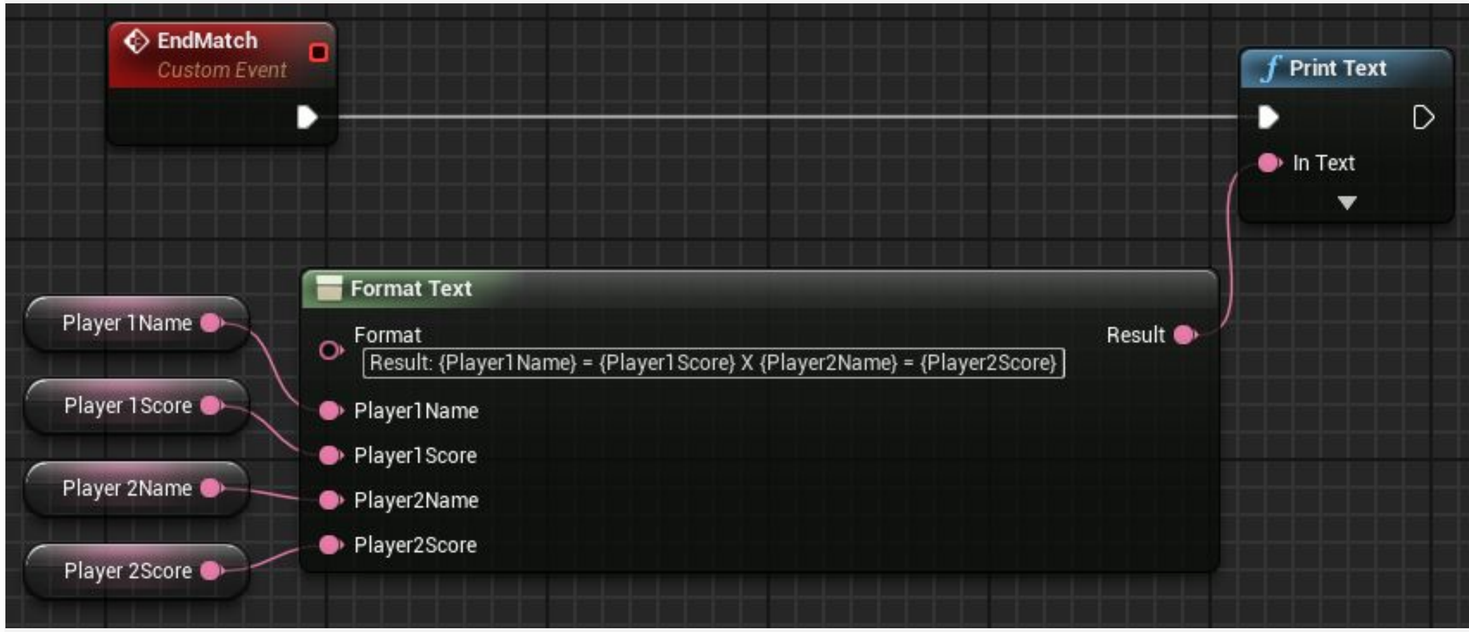
The text used in the **Format** parameter is as follows:

Result: {Player1Name} = {Player1Score} X {Player2Name} = {Player2Score}

After placing the value of the **Format** parameter, the Blueprint Editor generates the other input parameters.

The top image shows the **Format Text** node.

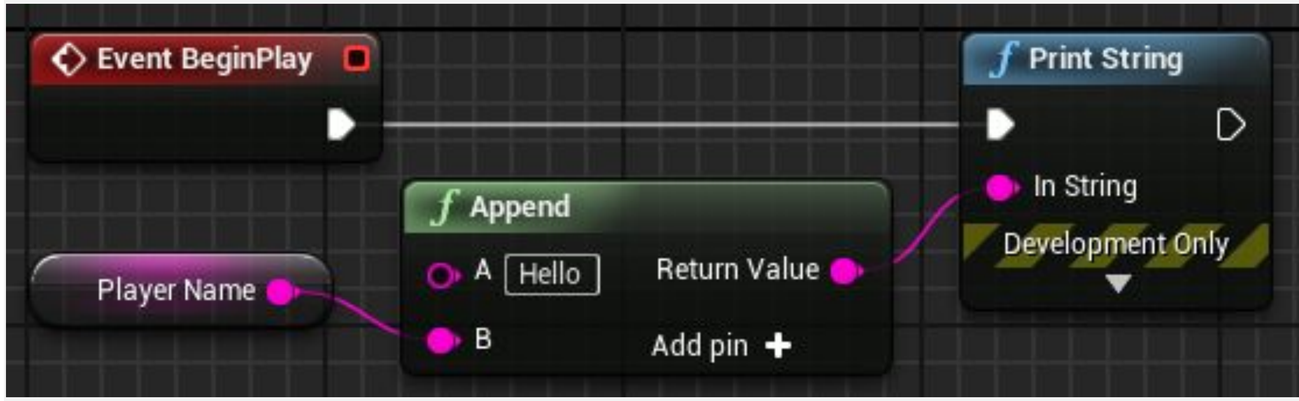
The bottom image shows an example of the generated text.



APPEND NODE

The **Append** node concatenates strings to create a new string. More string pins can be added using the **Add pin +** option.

In the example on the right, a custom welcome message is created using the player name, which is in a variable.

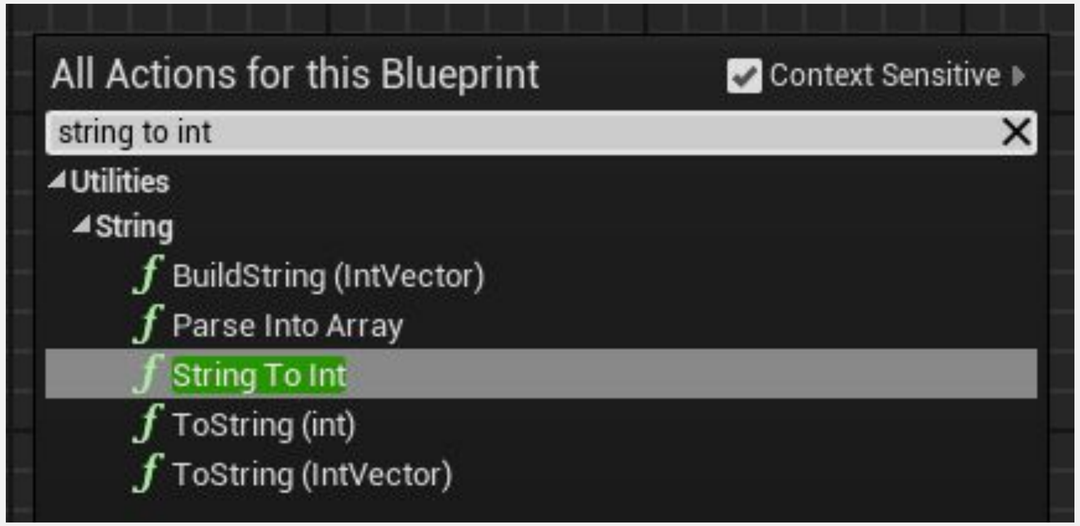


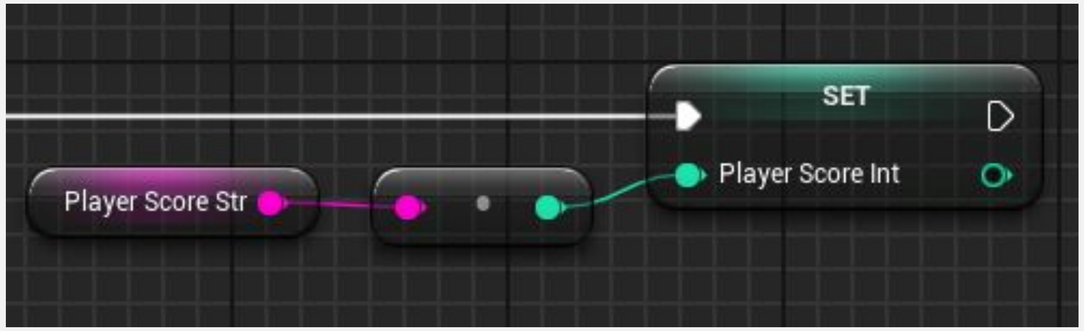
STRING TO INT NODE

Sometimes the same information contained in a string variable for the purpose of display is needed for use as an integer or float value for calculations.

To convert a string value to an integer value, you can use a function node called “**String To Int**”.

This conversion can also be done by just connecting the pin of a Get String node with the pin of an integer input. The Editor will automatically create the **String To Int** node, as seen in the bottom example on the right.





MATH

MATH EXPRESSION NODE

The **Math Expression** node is a special type of node that generates a subgraph using a specified mathematical expression.

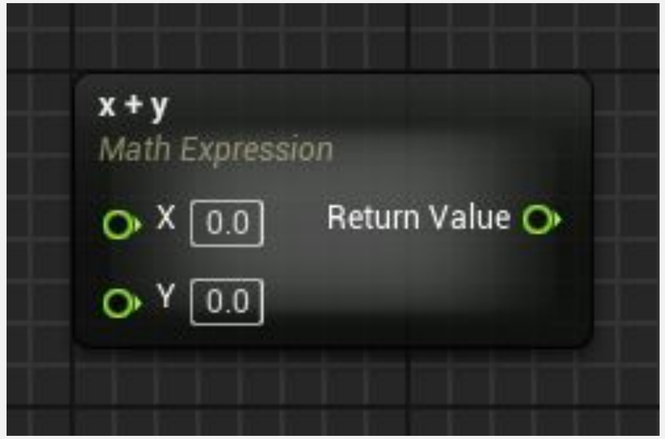
To use it, select “**Add Math Expression…**” in the **context menu**. The images on the right show a simple example using the expression “**x + y**”.

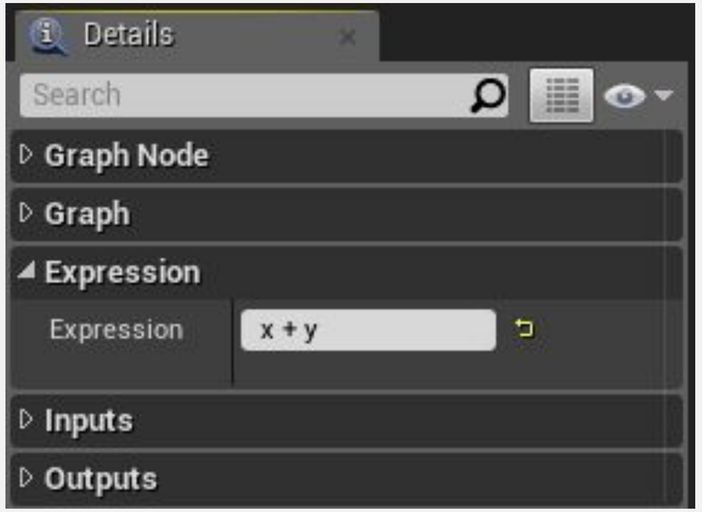
Input

- **Expression**: The expression that will be analyzed.
- **Parameters defined in “Expression”**: For each variable name in the expression, a new input parameter is generated.

Output

- **Return Value**: Outputs the result of the expression.



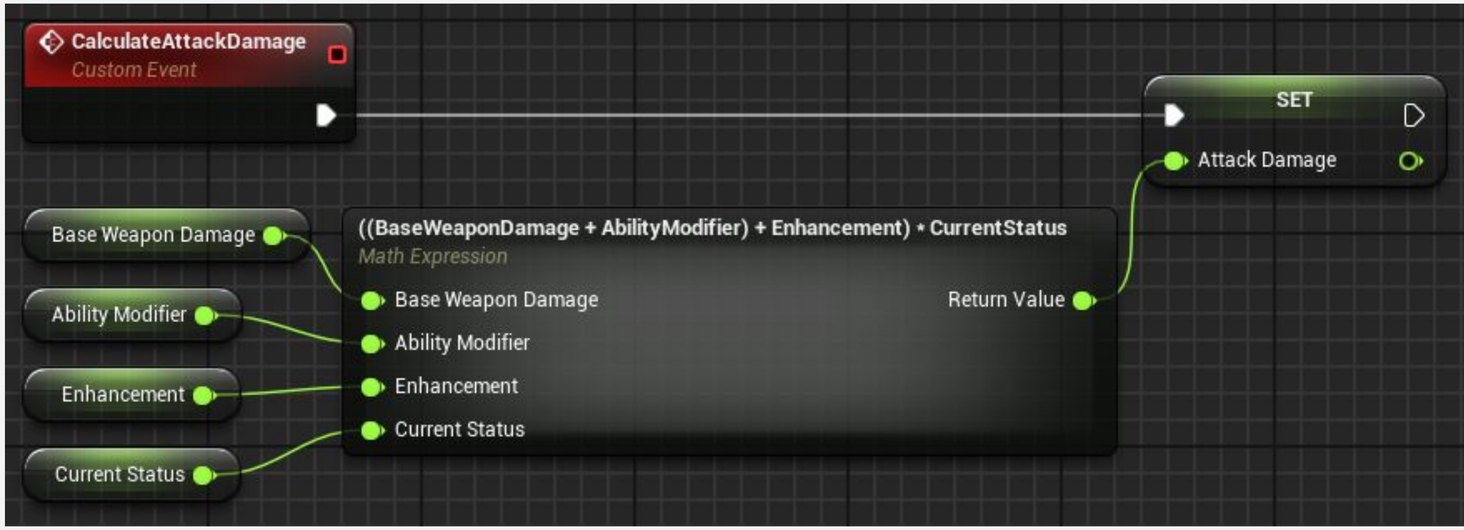


MATH EXPRESSION NODE: EXAMPLE

In the example on the right, the **CalculateAttackDamage** event calculates attack damage and stores the result in a variable. It uses a **Math Expression** node with the following expression:

((BaseWeaponDamage + AbilityModifier) + Enhancement) * CurrentStatus

The input parameters were generated based on this expression.



LERP NODE

“**Lerp**” is short for “**linear interpolation**”. This function node generates a value within a range of two specified values, based on the value of the **Alpha** parameter.

Input

- **A**: Takes in a float value that represents the lowest value that can be returned.
- **B**: Takes in a float value that represents the highest value that can be returned.
- **Alpha**: Takes in a float value between “0” and “1”. If the value is “0”, it returns the lowest value; if the value is “1”, it returns the highest value.

Output

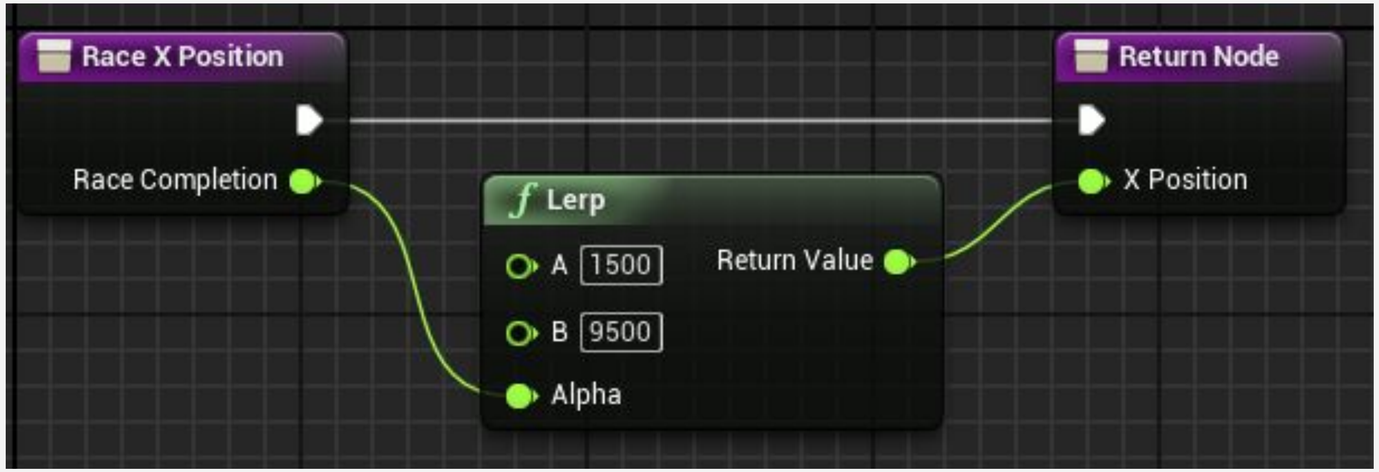
- **Return Value**: Outputs a float value between the values of the **A** and **B** parameters that is dependent on the value of the **Alpha** parameter.

LERP NODE: EXAMPLE

In the example to the right, there is a race that occurs along the X axis. The race starts at the position X = 1500 and ends at the position X = 9500.

The **Lerp** node is used with the value of parameter **A** set at “**1500**” and parameter **B** set at “**9500**”. The **Alpha** parameter takes in a value between “0” and “1” that indicates how much of the race has been completed and returns a corresponding value for the X position.

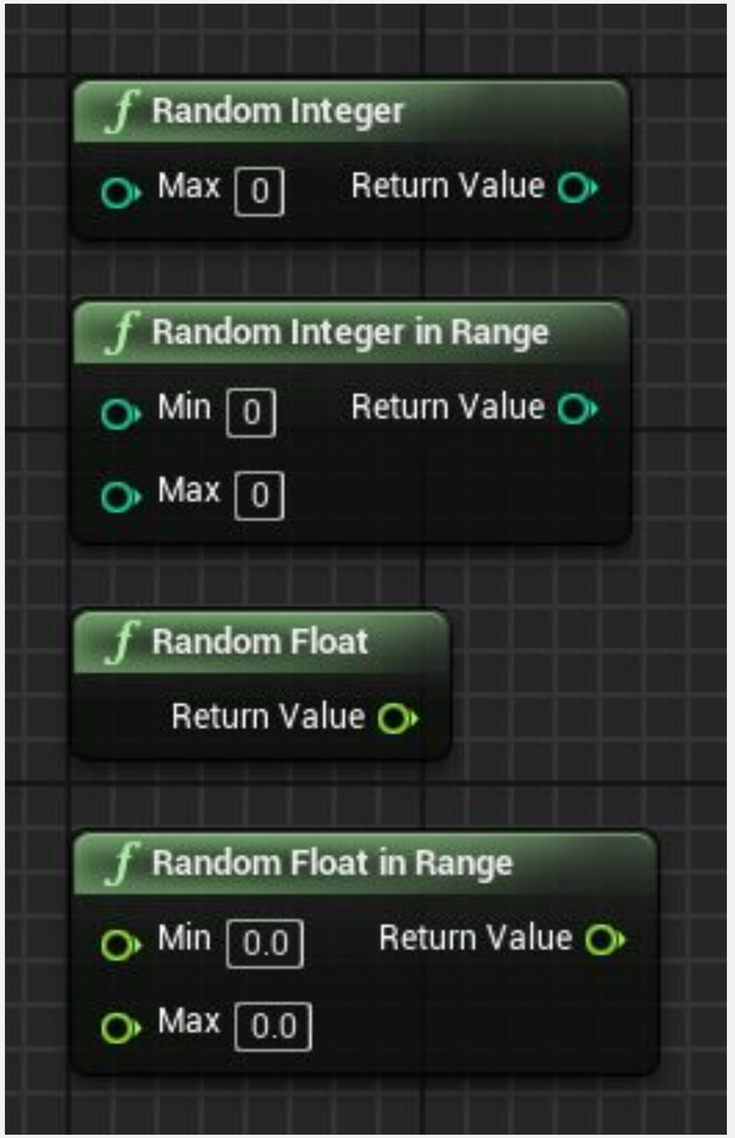
If the value of the **Alpha** parameter is set to “**0.5**”, the return value will be “**5500**”, representing the middle of the race.



RANDOM NUMBERS

There are random number functions that return a random value. The main ones are listed below:

- **Random Integer**: Returns an integer value between “0” and “Max – 1”.
- **Random Integer in Range**: Returns an integer value between “Min” and “Max”.
- **Random Float**: Returns a float value between “0” and “1”.
- **Random Float in Range**: Returns a float value between “Min” and “Max”.



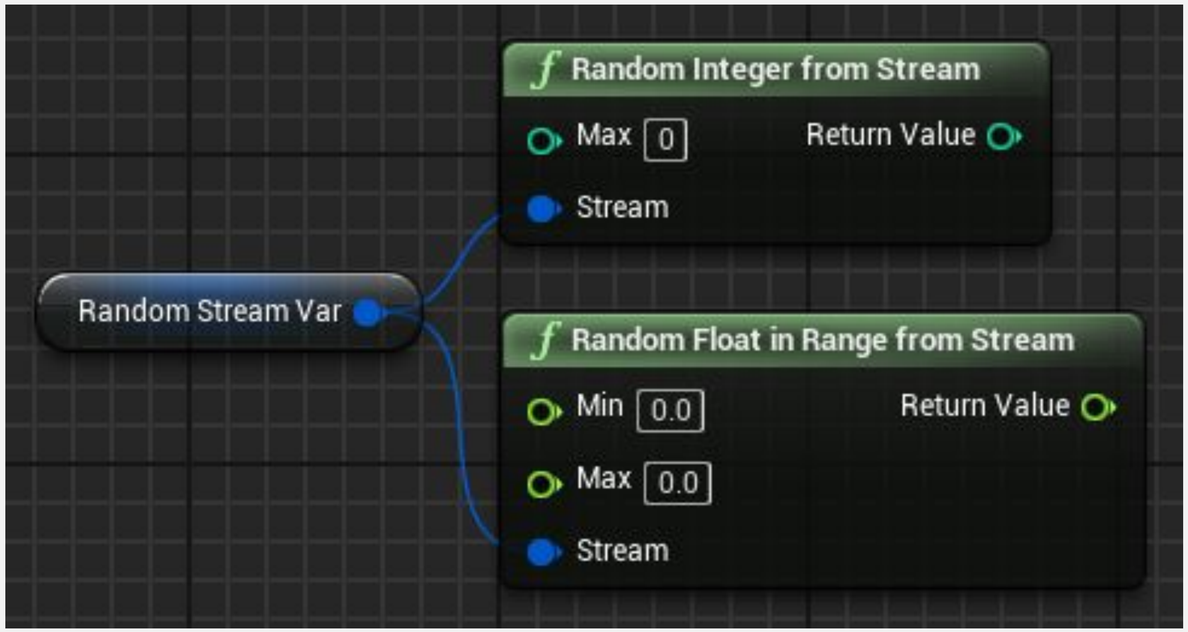
RANDOM NUMBERS FROM STREAMS

It is possible to create a sequence of repeatable random numbers using a **Random Stream** variable.

To do this, create a variable of type “**Random Stream**”. Set the **Initial Seed** property in the **Default Value** section of the **Details** panel for the variable.

The value of the **Initial Seed** property will define the sequence of the random numbers.

The image on the right shows some random number functions that use a Random Stream variable.



SUMMARY

This lecture presented many flow control nodes. It showed how to use ForEachLoop and switch nodes.

It explained how to work with string functions, the Math Expression node, and random number functions.