Here are a list of the Functions and Macros included within the Fuzzy Logic Function Library (Basic Edition):

## Functions:

1. **Fuzzy Random Fuzzy Value** (Produces a Random Fuzzy Logic value. An integer between 0 and 9.)
2. **Fuzzy Logic Inverter** (Takes a Fuzzy Logic Value as input and inverts the value so True becomes False and False becomes True.)
3. **Fuzzy Logic Add** (Takes two Fuzzy Logic Values as input and adds them, clamping the output between 0 and 9.)
4. **Fuzzy Logic Add3** (Takes three Fuzzy Logic Values as input and adds them, clamping the output between 0 and 9.)
5. **Fuzzy Logic Add4** (Takes four Fuzzy Logic Values as input and adds them, clamping the output between 0 and 9.)
6. **Fuzzy Logic Average** (Takes two Fuzzy Logic Values as input and outputs the average of the two values.)
7. **Fuzzy Logic Average3** (Takes three Fuzzy Logic Values as input and outputs the average of the three values.)
8. **Fuzzy Logic Average4** (Takes four Fuzzy Logic Values as input and outputs the average of the four values.)
9. **Fuzzy Logic Average5** (Takes five Fuzzy Logic Values as input and outputs the average of the five values.)
10. **Fuzzy Logic Average6** (Takes six Fuzzy Logic Values as input and outputs the average of the six values.)
11. **Fuzzy Logic Average7** (Takes seven Fuzzy Logic Values as input and outputs the average of the seven values.)
12. **Fuzzy Logic Average8** (Takes eight Fuzzy Logic Values as input and outputs the average of the eight values.)
13. **Fuzzy Logic Average9** (Takes nine Fuzzy Logic Values as input and outputs the average of the nine values.)
14. **Fuzzy Logic Average Array** (Takes an array of Fuzzy Logic Values as input and outputs the average of the values within the array.)
15. **Fuzzy Logic Subtract** (Takes a Fuzzy Logic Value and subtracts it from another, clamping the output between 0 and 9.)
16. **Fuzzy Logic Subtract2** (Takes two Fuzzy Logic Values and subtracts them from another Fuzzy Logic Value, clamping the output between 0 and 9.)
17. **Fuzzy Logic Subtract3** (Takes three Fuzzy Logic Values and subtracts them from another Fuzzy Logic Value, clamping the output between 0 and 9.)
18. **Fuzzy Logic MaximumOf2** (Selects the Maximum of two Fuzzy Logic Values.)
19. **Fuzzy Logic MaximumOf3** (Selects the Maximum of three Fuzzy Logic Values.)
20. **Fuzzy Logic MaximumOf4** (Selects the Maximum of four Fuzzy Logic Values.)
21. **Fuzzy Logic MaximumOf5** (Selects the Maximum of five Fuzzy Logic Values.)
22. **Fuzzy Logic MaximumOf6** (Selects the Maximum of six Fuzzy Logic Values.)
23. **Fuzzy Logic MaximumOf7** (Selects the Maximum of seven Fuzzy Logic Values.)
24. **Fuzzy Logic MaximumOf8** (Selects the Maximum of eight Fuzzy Logic Values.)
25. **Fuzzy Logic MaximumOf9** (Selects the Maximum of nine Fuzzy Logic Values.)
26. **Fuzzy Logic MaximumOfArray** (Selects the Maximum of an array of Fuzzy Logic Values.)
27. **Fuzzy Logic MinimumOf2** (Selects the Minimum of two Fuzzy Logic Values.)
28. **Fuzzy Logic MinimumOf3** (Selects the Minimum of three Fuzzy Logic Values.)
29. **Fuzzy Logic MinimumOf4** (Selects the Minimum of four Fuzzy Logic Values.)
30. **Fuzzy Logic MinimumOf5** (Selects the Minimum of five Fuzzy Logic Values.)
31. **Fuzzy Logic MinimumOf6** (Selects the Minimum of six Fuzzy Logic Values.)
32. **Fuzzy Logic MinimumOf7** (Selects the Minimum of seven Fuzzy Logic Values.)
33. **Fuzzy Logic MinimumOf8** (Selects the Minimum of eight Fuzzy Logic Values.)
34. **Fuzzy Logic MinimumOf9** (Selects the Minimum of nine Fuzzy Logic Values.)
35. **Fuzzy Logic MinimumOfArray** (Selects the Minimum of an array of Fuzzy Logic Values.)
36. **Fuzzy Logic Multiplier** (Multiplies a Fuzzy Logic Value by a float multiplier, and outputs the result.)
37. **Fuzzy Logic Weighted Add** (Takes two Fuzzy Logic Values as input and adds them, using a float value as a multiplier for the Fuzzy Logic Inputs, clamping the overall output between 0 and 9.)
38. **Fuzzy Logic Weighted Add3** (Takes three Fuzzy Logic Values as input and adds them, using a float value as a multiplier for the Fuzzy Logic Inputs, clamping the overall output between 0 and 9.)

39. **Fuzzy Logic Weighted Add4** (Takes four Fuzzy Logic Values as input and adds them, using a float value as a multiplier for the Fuzzy Logic Inputs, clamping the overall output between 0 and 9.)
40. **Fuzzy Logic Weighted Add5** (Takes five Fuzzy Logic Values as input and adds them, using a float value as a multiplier for the Fuzzy Logic Inputs, clamping the overall output between 0 and 9.)
41. **Fuzzy Logic Weighted Add6** (Takes six Fuzzy Logic Values as input and adds them, using a float value as a multiplier for the Fuzzy Logic Inputs, clamping the overall output between 0 and 9.)
42. **Fuzzy Logic Weighted Average** (Takes two Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the two values.)
43. **Fuzzy Logic Weighted Average3** (Takes three Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the three values.)
44. **Fuzzy Logic Weighted Average4** (Takes four Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the four values.)
45. **Fuzzy Logic Weighted Average5** (Takes five Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the five values.)
46. **Fuzzy Logic Weighted Average6** (Takes six Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the six values.)
47. **Fuzzy Logic Weighted Average7** (Takes seven Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the seven values.)
48. **Fuzzy Logic Weighted Average8** (Takes eight Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the eight values.)
49. **Fuzzy Logic Weighted Average9** (Takes nine Fuzzy Logic Values as input then uses a multiplier to adjust their values, finally outputting the average of the nine values.)
50. **Fuzzy Logic Weighted Subtract** (Takes a Fuzzy Logic Value and uses a float multiplier to adjust its value, then subtracts it from another Fuzzy Logic Value, clamping the output between 0 and 9.)
51. **Fuzzy Logic Weighted Subtract2** (Takes two Fuzzy Logic Values and uses a float multiplier to adjust their values, then subtracts them from another Fuzzy Logic Value, clamping the output between 0 and 9.)
52. **Fuzzy Logic Weighted Subtract3** (Takes three Fuzzy Logic Values and uses a float multiplier to adjust their values, then subtracts them from another Fuzzy Logic Value, clamping the output between 0 and 9.)
53. **Fuzzy Logic Boolean Value Mapping** (Maps a Boolean value to a Fuzzy Logic Value between 0 and 9, outputting a 0 for False and a 9 for True.)
54. **Fuzzy Logic Multi Boolean Value Mapping** (Maps an array of Boolean values to a Fuzzy Logic Value. Works with a minimum of 1 Boolean values in an Array. Maps an array of Boolean values to a Fuzzy Logic Value between 0 and 9 with a value closer to 0 being more false and a value closer to 9 being more true.)
55. **Fuzzy Logic Fuzzy To Boolean Mapping** (Takes a Fuzzy Logic Value as input and maps the value to either … True or False. If “Must be Mostly True” is selected, then there will be a higher burden to get a truth value.)
56. **Fuzzy Logic Float Value Range Mapping** (Takes an input float and a minimum and maximum value and maps it to a Fuzzy Logic Value.)
57. **Fuzzy Logic Int Value Range Mapping** (Takes an input Integer and a minimum and maximum value and maps it to a Fuzzy Logic Value.)
58. **Mixed Logic Average Boolean and Fuzzy Logic Value** (Takes an input Fuzzy Logic Value and an input Boolean and averages them, outputting the Fuzzy Logic Value product of the average.)

## Macros:

1. **Fuzzy Branch** (Branches the code execution based on a Fuzzy Logic Value input, routing the flow along one of 10 paths.)
2. **Fuzzy Inverted Branch** (Branches the code execution based on a Fuzzy Logic Value input but inverted, routing the flow along one of 10 paths.)
3. **Fuzzy Boolean Branch** (Branches the code execution based on a Fuzzy Logic Value input, routing the flow along a true or false path.)
4. **Fuzzy Ternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, routing the flow along a true, mixed or false path.)
5. **Fuzzy Quaternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, routing the flow along a true, mostly true, mostly false or false path.)
6. **Fuzzy Quinternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, routing the flow along a true, mostly true, mixed, mostly false or false path.)
7. **Fuzzy Inverted Boolean Branch** (Branches the code execution based on a Fuzzy Logic Value input, but inverted, routing the flow along a true or false path.)
8. **Fuzzy Inverted Ternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, but inverted, routing the flow along a true, mixed or false path.)
9. **Fuzzy Inverted Quaternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, but inverted, routing the flow along a true, mostly true, mostly false or false path.)
10. **Fuzzy Inverted Quinternary Branch** (Branches the code execution based on a Fuzzy Logic Value input, but inverted, routing the flow along a true, mostly true, mixed, mostly false or false path.)
11. **Fuzzy Boolean to Ternary Branch** (Takes two input Booleans and routes the execution chain along a true, mixed or false path.)
12. **Fuzzy Boolean to Quaternary Branch** (Takes three input Booleans and routes the execution chain along a true, mostly true, mostly false or false path.)
13. **Fuzzy Boolean to Quinternary Branch** (Takes four input Booleans and routes the execution chain along a true, mostly true, mixed, mostly false or false path.)
14. **2 Bit Branch** (Takes two input Booleans and routes the execution chain along one of four paths.)
15. **3 Bit Branch** (Takes three input Booleans and routes the execution chain along one of eight paths.)
16. **NOT Branch** (An inverted Branch node where a true value outputs False and a false value outputs True.)

# Fuzzy Branch Node


At the heart of traditional Boolean Logic is the Branch Node. It uses a condition, either true or false, to route the flow of the code accordingly. Fuzzy Logic takes the concepts of Branching Logic and evolves them beyond binary.
A Fuzzy Input is a Fuzzy Logic Value, or an integer clamped between 0 and 9.
From a value within this range, we are able to route the flow along many more possible directions, each representing a degree of truth with values closer to 0 being more false and closer to 9 being more true.

The standard Fuzzy Branch node will route the channel along the corresponding value. The Fuzzy Inverted Branch does the same, but the inverted value. Equivalent to using a Fuzzy Inverter Function on the input pin of a regular Fuzzy Branch node.

Like Boolean Logic, Fuzzy Logic can also be used to branch code along two pathways, either True or False.
How this differs from a standard branch node however, is its ability to factor degrees of truth within its input variable before making a final true or false choice.
That means the code setting the Fuzzy Input can involve nuanced decision making or a more complex set of conditional checks before the final decision is made.
For example, the Fuzzy Input could take the average of an array of Boolean values using (INSERT LATER), and then use that output and a different Fuzzy Logic Variable to select the maximum of the two, which is then set as the Fuzzy Input. The Fuzzy Input value from 0 to 9 is then compressed into a True or False execution channel for code.
This node's inverted counterpart simply performs True when the value is False and False when the value is True. It's equivalent to using a "Fuzzy Logic Inverter" as the Fuzzy Input pin.

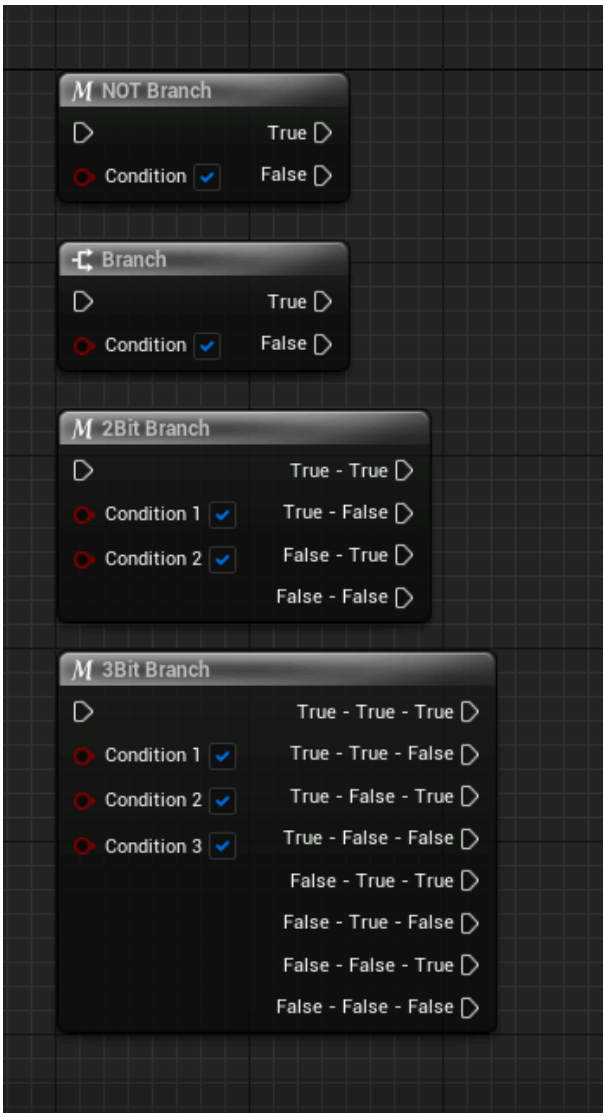
Ternary is a type of computing dealing with three values instead of two like in Binary computing. It's a different base number system (Base 3). A Fuzzy Logic Input Value is used to select from one of three code execution channels.
Three states is the lowest possible resolution for having Degrees of Truth represented by a spectrum.
This kind of decision making is very close to binary, also including "True" and "False", but with the additional "Mixed" output state.
This node's inverted counterpart simply performs True when the value is False and False when the value is True. Mixed stays the same. It's equivalent to using a "Fuzzy Logic Inverter" as the Fuzzy Input pin.

Quaternary (Base 4) allows you to have slightly higher resolution branching behavior represented by 4 states, "True", "Mostly True", "Mostly False" and "False". A Fuzzy Logic Input Value is used to select from one of these four code execution channels.
At this level of resolution, the "Mixed" option from the Ternary branch is split into two possible states each favoring either True or False. The perfect middle state is absent from this base number system because it has an even number of execution channels.
This node's inverted counterpart is equivalent to using a "Fuzzy Logic Inverter" as the Fuzzy Input pin.

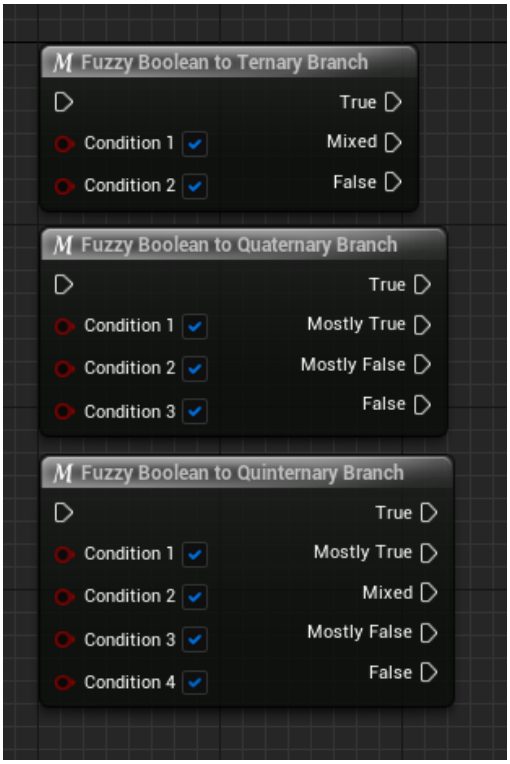
Quinternary (Base 5) allows you to have slightly higher resolution branching behavior represented by 5 states, "True", "Mostly True", "Mixed", "Mostly False" and "False". A Fuzzy Logic Input Value is used to select from one of these five code execution channels.
At this level of resolution, the "Mixed" option from the Ternary branch returns, while the "Mostly True" and "Mostly False" options remain as well. The perfect middle state is present from this base number system because it has an odd number of execution channels.
This node's inverted counterpart is equivalent to using a "Fuzzy Logic Inverter" as the Fuzzy Input pin.

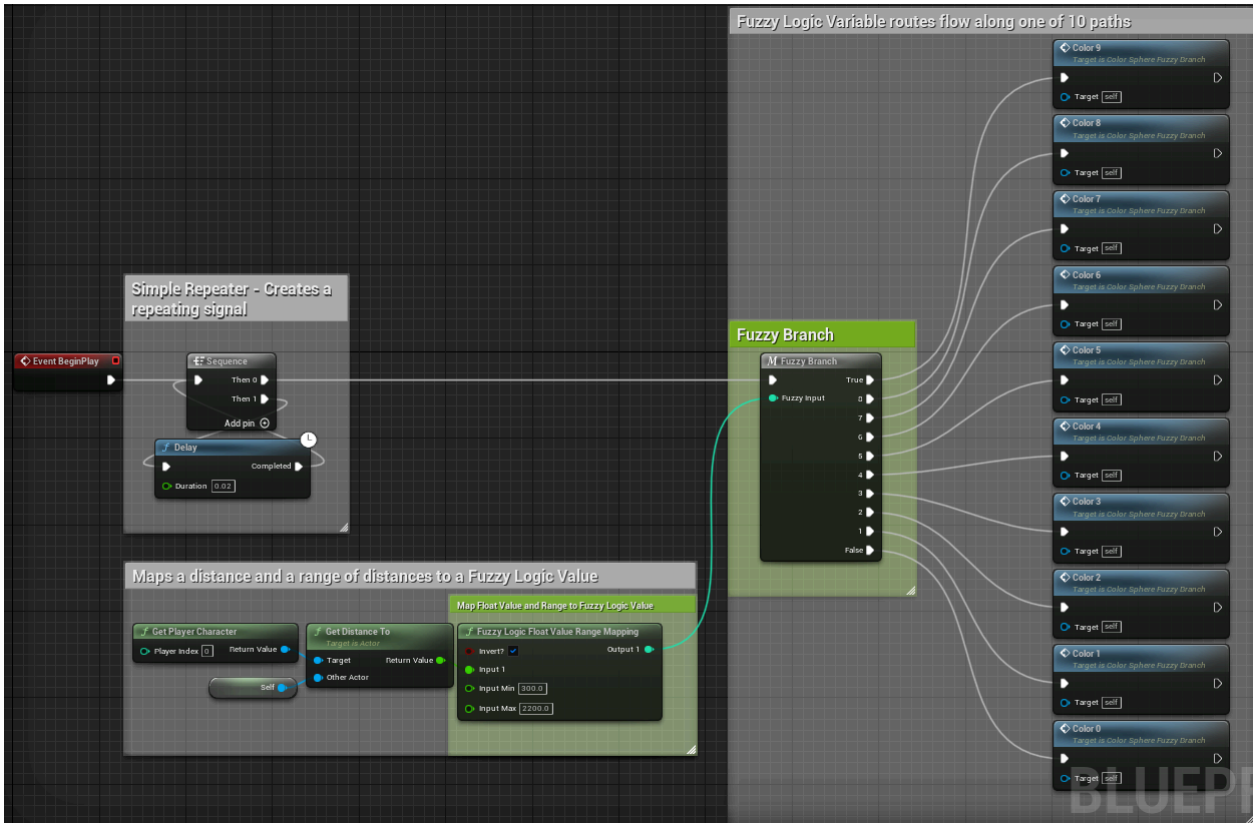
# Advanced Branch Node

The advanced Branch Nodes are a series of Boolean Branch nodes that enhance the functionality of Boolean Branching Logic.
The NOT branch is equivalent to using a NOT as an input condition for a standard Branch Node.
The 2Bit Branch takes two input conditions and produces branching behavior based on two sets of Boolean values, routing to one of four code execution channels.
The 3 Bit Branch does something similar but with three input Booleans being routed to one of eight code execution channels.

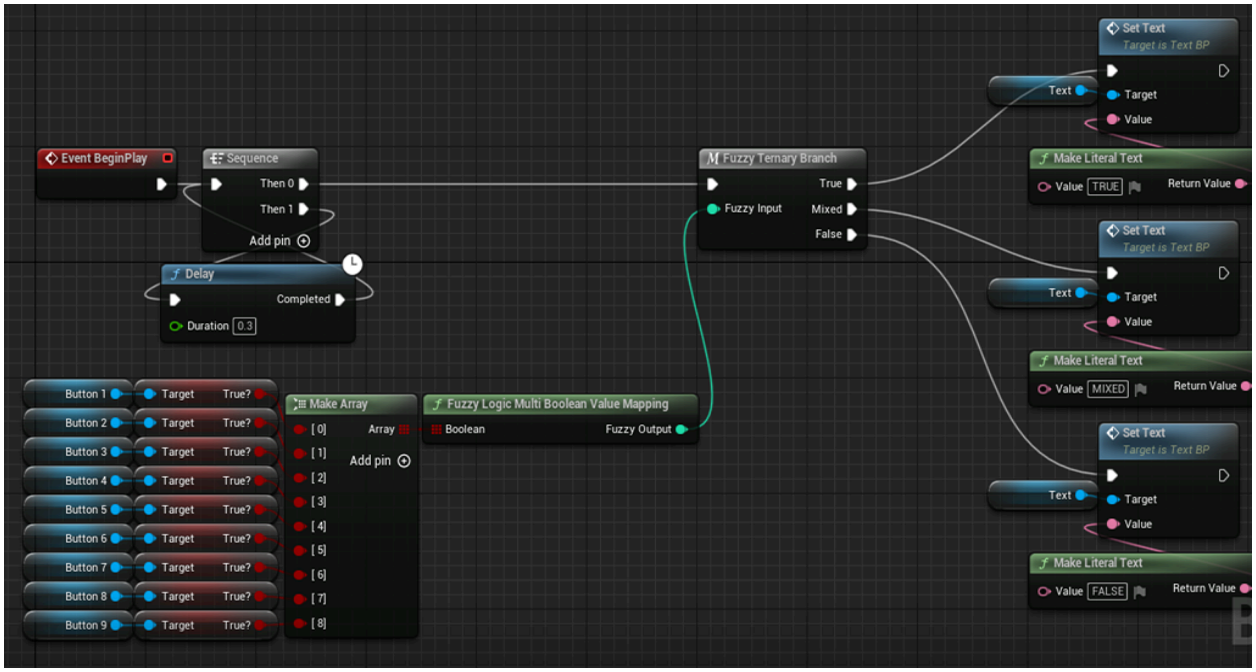


There are also Branch nodes in this toolkit that takes multiple Boolean Inputs and map them to fuzzy logic output channels. These Macros include:
Fuzzy Boolean to Ternary Branch, which takes two input Booleans and maps them to either True, Mixed or False output channels.
Fuzzy Boolean to Quaternary Branch, which takes three input Booleans and maps them to either True, Mostly True, Mostly False or False.
Fuzzy Boolean to Quinternary Branch, which takes four input Booleans and maps them to either True, Mostly True, Mixed, Mostly False or False.

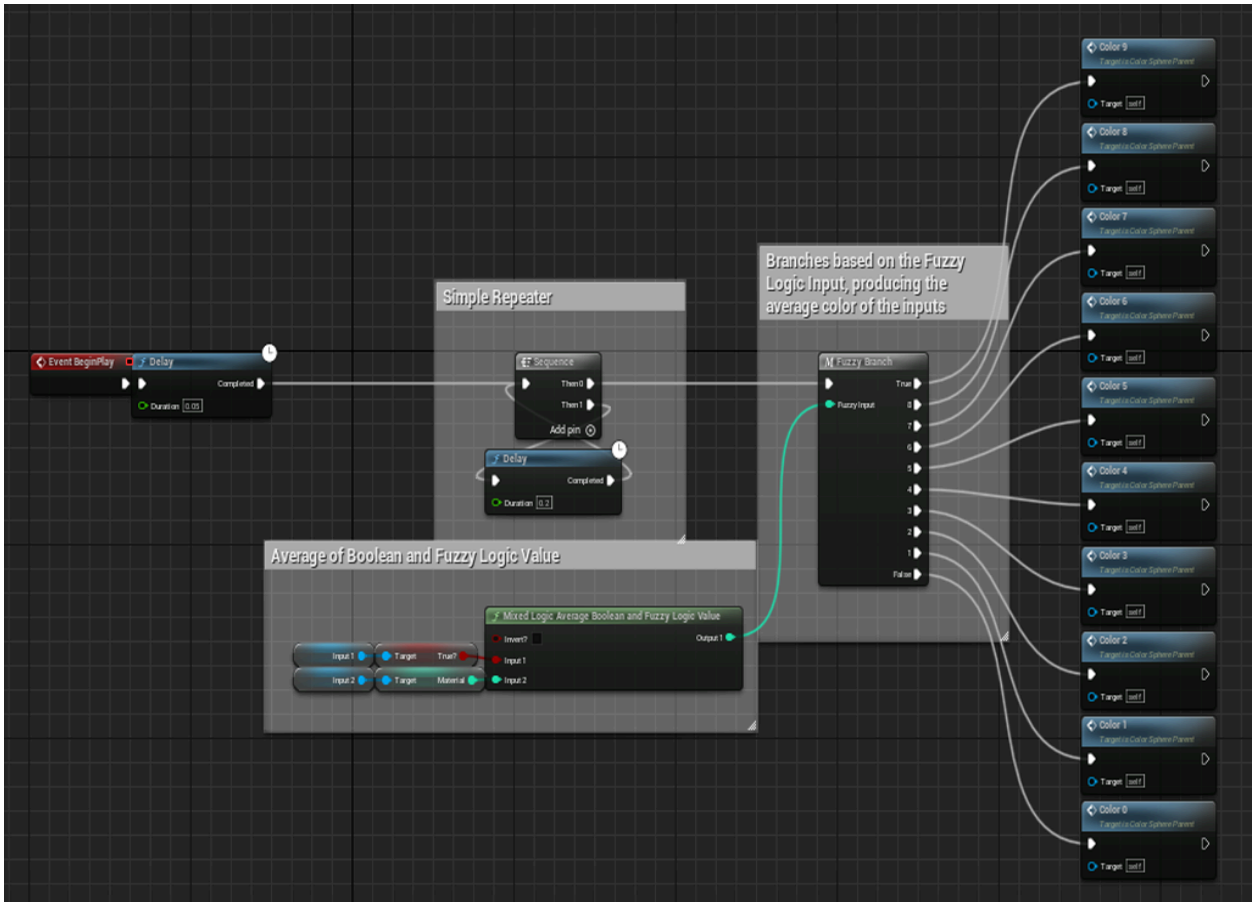




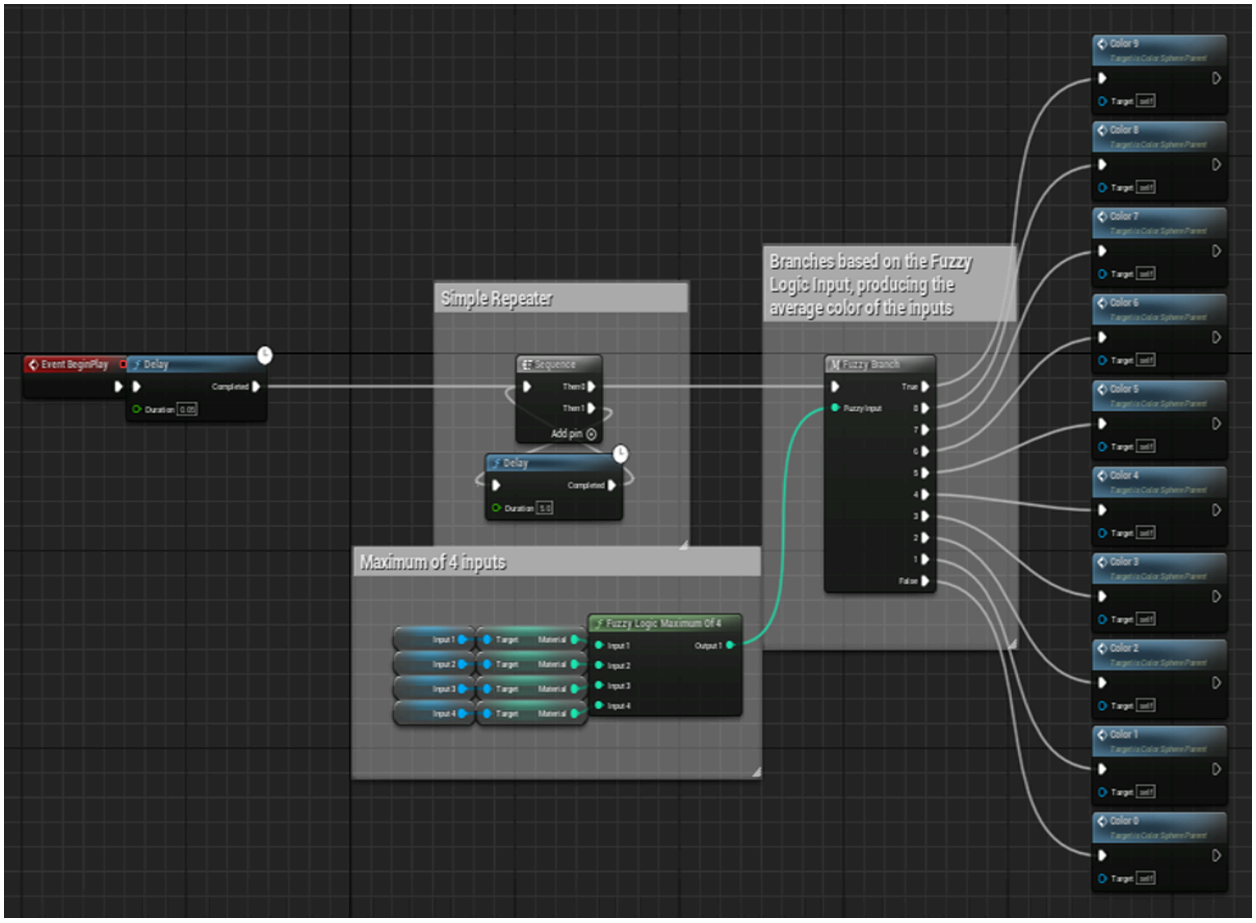

So there's a few things going on in this graph. The first I'll go over is the "Simple Repeater".
This isn't really to do with fuzzy logic but it's a trick I found that can produce a repeating flow execution periodically. This is mainly for testing purposes to demonstrate the color changing spheres in the example project, but feel free to replicate and use within your own projects should you ever desire an endless repeating loop not linked to Tick Events. Similar to using a timer.
The Fuzzy Branch Macro itself takes an input value from 0 to 9 and routes the flow along one of 10 paths. This particular Macro is designed to give you a high resolution spectrum of Truth with 9 True and 0 being False. Every value in-between represents a degree of Truth or Falsehood.
To produce the Fuzzy Logic value in this example Blueprint, we use a Get Distance To node to compare the distances between the character and the Sphere. We then use a Fuzzy Logic Float Value Range Mapping function to map the distance float to a Fuzzy Logic Value. To do this, we give a minimum and maximum value for the range the distance float occupies and then this data is used to map the information onto a Fuzzy Logic Spectrum. In this example blueprint, we set the Invert Boolean to true so that when the player is closer to the sphere, that will produce a more true value as opposed to a more false value.
There are instances when using fuzzy logic when you may need to invert the spectrums because two values you want to adjust could represent logically inverse concepts. For example, if you had "Player Karma is High" represented with a spectrum of degrees of truth, but also had "Player Luck is Low" represented with a spectrum of degrees of truth, you might encounter a scenario where you want to take both of these values into the decision making process. By applying an inverter to either value, you can make it easier to produce emergent Fuzzy Logic Values that are driven by more fundamental Fuzzy Logic Values.




In this graph here, we can see that an array of 9 boolean values is being mapped to a Fuzzy Logic Value (An integer between 0 and 9) by a ‘Fuzzy Logic Multi Boolean Value Mapping’ Function. The resulting variable is then used to decide from one of three execution paths for the code using a Fuzzy Ternary Branch. This is a good example of how Fuzzy Logic can harmonize with Boolean logic, enhancing the capabilities of both logic types.

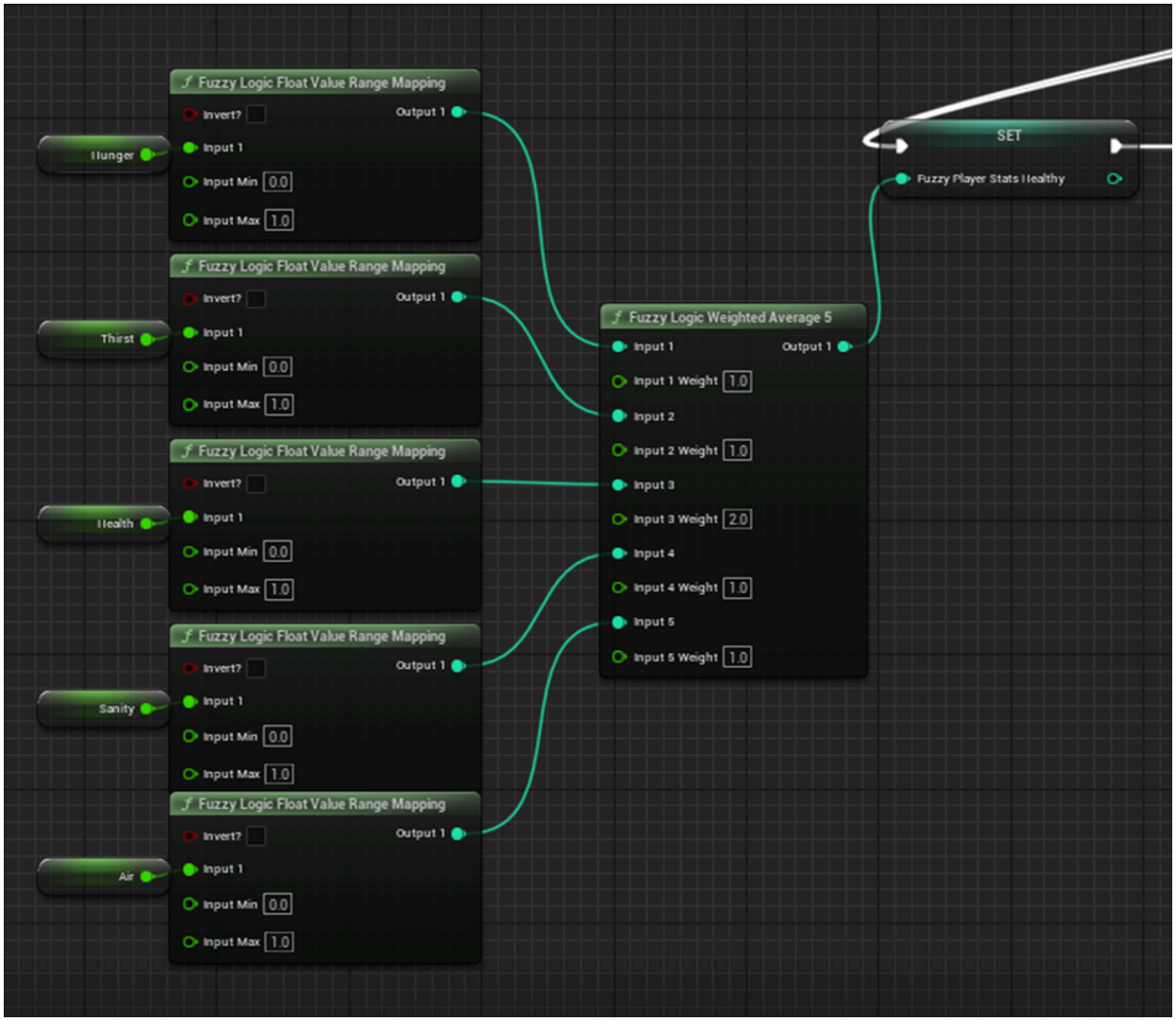



In this example, a Boolean value is averaged with a Fuzzy Logic value to collectively decide which code execution path should be taken using a “Fuzzy Branch” Macro.




In this example, we are getting the maximum of four input Fuzzy Logic values and then using the highest of these values to decide the execution path for the code using a “Fuzzy Branch” Macro.

Functions like the maximum and minimum can be used if your game’s require logic where any of several conditions being particularly high or low is important for decision making. For example: You could have several stats like Health, Hunger, Thirst and Sanity, and if you want logic where the player is forced to maintain high stats, you could take the minimum of these four stats and inflict a penalty based on Fuzzy Branching nodes based on how low the minimum value is.




Here's a real use case example from the game I'm developing called The Backrooms: Mass Extinction.

Here, you can see five stats (Hunger, Thirst, Health, Sanity and Air) are being mapped to a Fuzzy Logic value by the node "Fuzzy Logic Float Value Range Mapping", which is taking the float values for these stats and their minimum and maximum values, and they are projecting the stat onto a Fuzzy Logic Value.

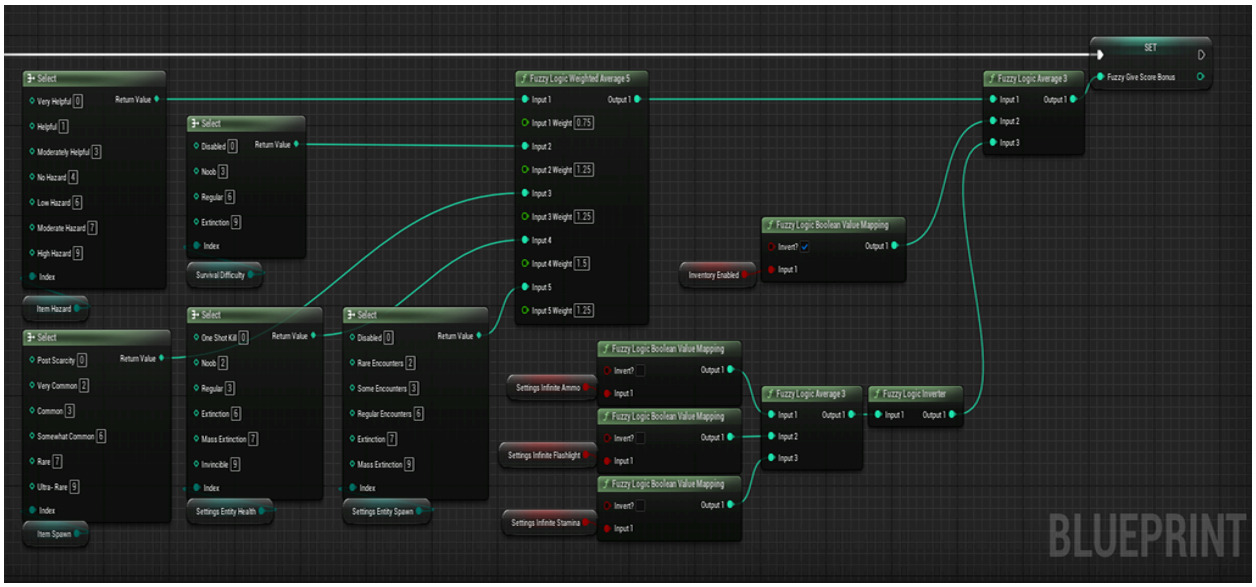
This can be a useful way to convert variables into Fuzzy Logic Variables.

The five mapped values are then averaged, but with some values having more weight than the others. This is done with the "Fuzzy Logic Weighted Average 5" function. In this example, Health has double the weight compared to the other Fuzzy Logic Variables.

The output value of this simple node graph is a value which determines the overall Truthfulness of this statement: "Player Stats Healthy".

This variable then gets used for decision making purposes in different parts of my project such as in the score system where a score is calculated at the end of the game, or the mercy system, which gives mercy to the player based on how poor their stats are.

There are a lot of creative ways to implement fuzzy logic!




Here’s another more complicated graph from my game The Backrooms: Mass Extinction.
If you ever want to use an Enumeration to produce fuzzy logic values, then use a select node and create an integer between 0 and 9 based on what value you want each element to correspond to. There’s a few ways you could implement something like that, such as the way I have above, turning difficulty settings for the game into Fuzzy Logic values that are then measured up in a weighted average.
This graph also has an example of inversion, where for the two fuzzy logic variables, in order to be used in a function together, one has to be inverted. This can happen when a variable represents a concept that needs to be inverted, such as: Player Hungry, to Player Not Hungry.