Complex For Game Systems Evaluation  
By Jacob Cooper

## Description

For my complex game system I made fuzzy state machine logic which allows the user to create a state machine graph to simulate fuzzy logic and output a specified state for the AI to use. Comes with a few tools like a visualizer and debugger.

The brief for this project was followed accordingly and only had new stuff added, the base concept however stayed the same and is in the project.

## Implementation

To implement the fuzzy state machine you import the **FuzzyStateMachine Package.unitypackage** into your project using unity 2020.1+. (This does not require any dependencies!)

You can create a new graph by right clicking the assets folder, clicking *“Create/FuzzyStateMachine/New Graph”*, you can also go Graph/ to create a ShapeSet or Variable scriptable object. These allow you to create and manipulate a graph and data, to load a graph you put the StateMachineLoader component on an object, and to visualise/debug it put a StateMachineVisualiser on the object too.  
  
You can create rules by inheriting FuzzyRuleSet and states by inheriting StateMachineState. I highly recommend looking at the demo AI, it will help you understand implementation even further (seen below).

For any confusion, there is a demo included in this project under “*Scripts/FuzzyStateMachine/Demo”*which shows exactly how to use this modular system.

## Issues During Integration

There were a bunch of issues to do with integrating fuzzy logic mainly because no one has really made a public documentation or demo for fuzzy logic in video games, the only examples out there are bare bones fuzzy logic mathematical equations in case studies. So making the fuzzy logic work, and to make it then work with a state machine was incredibly difficult.

Creating the graph was hard as I was using an experimental system which was undocumented, so alot of experimentation and bug testing was at play. Saving the graph was simple as it was simply sent to a scriptable object, and loading the graph wasn’t too difficult either, the hard part was when I converted the graph logic into classes and functionality using activators which was incredibly mind bending.

The demo was also hard to make as even I was lost on how to implement my own software in, so a lot of changes had to be made to simplify user experience.

## Performance

Now to my surprise the system is actually pretty fast when loading a bunch of data and calculating it over and over again on a bunch of agents. I assumed it would be very expensive to constantly create classes from types using activators which inside would have a bunch of lists and dictionaries and scriptable object calls the list goes on, and somehow on average that alone would only take around 0.0003ms! And with only 13 demo testers it still performed surprisingly well.  
  
The performance was testing inside of the logic calculation and was debugged to the visualiser logs. So even after recreating classes from type names, there wasn’t much performance impact, most likely could be optimised even further by storing created classes instead of recreating them during function calls.

## System Changes

There are no system changes required, the only requirement is that unity version 2020.1 or above.