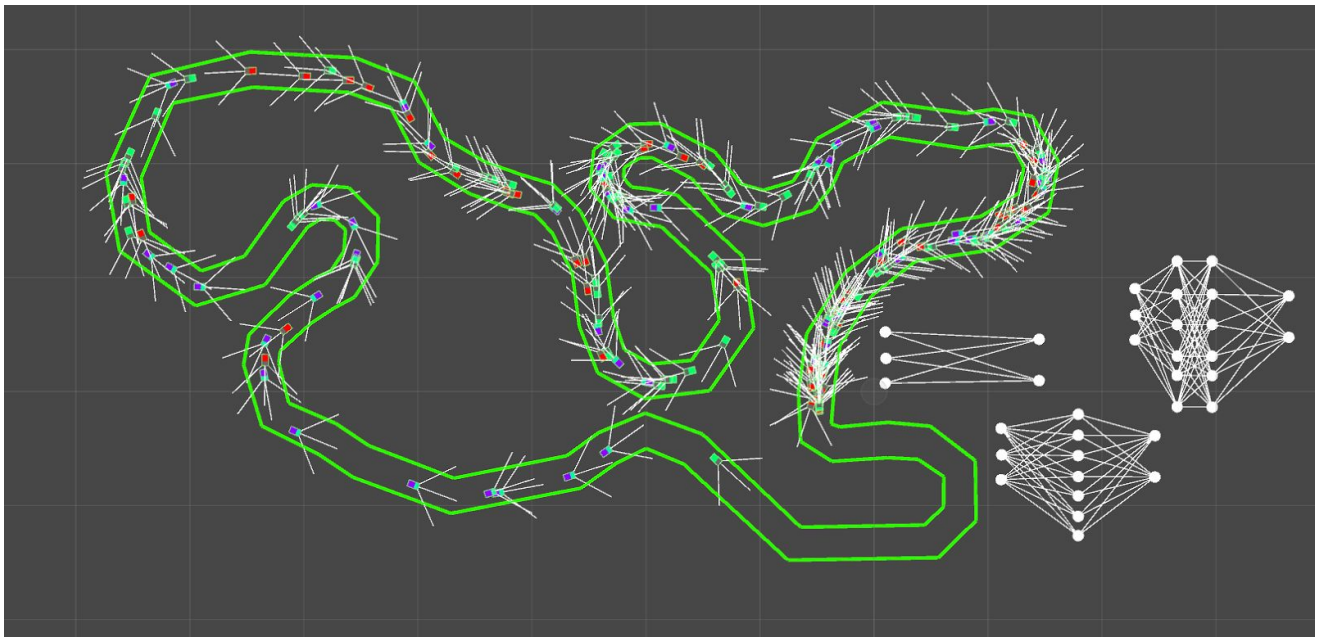
Neural network controlled car steering

By Bashar Saada

After coming across some multiple youtube video[1] demonstrating the uses of neural networks for basics AI behaviours, I was curious about how these networks were made. I saw lots of videos demonstrating their abilities and different uses in different AI behaviours, but I couldn't understand how they worked.
So I decided to build my own project with them so I can understand them better.

I decided to go with car steering behaviour, as that felt like the most interesting behaviour for me.
I found this video[2] explaining how neural networks work, and decided to implement the entire thing from scratch without the use of any machine learning libraries so I can understand the process from top to bottom.

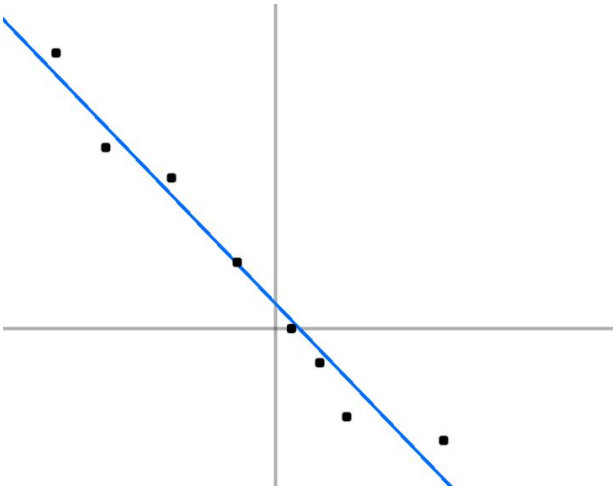
Picture of the cars going around the track using 3 different network configurations in the final implementation (each car color is using a different configuration):



[1] "Machine Learning for Flappy Bird using Neural Network ... - YouTube." 10 Aug. 2017, https://www.youtube.com/watch?v=aeWmdojEJf0. Accessed 20 Jan. 2018.
[2] "Beginner Intro to Neural Networks 1: Data and Graphing - YouTube." 9 Mar. 2017, https://www.youtube.com/watch?v=ZzWaow1Rvho. Accessed 20 Jan. 2018.

Implementing the neural node system:

After watching the tutorial on how neural networks worked, I had to build the neural node system.
So I started out by implementing the building blocks of neural networks, a single node. The context that I used to learn how to implement this was a simple graph, where I used regression to graph a line that best represents the data points I ploted, and using regression to lower the "cost" (The cost being how much does the graph stray away from the data points), and using the derivative of the cost to change my weights and bias to fit the graph.



After lots of tweaking and learning about how that equation is suppose to work, I managed to get it to work (Checkout "**Graph Scene**" in the unity (click to add data points)).

Deciding on the use of the network:

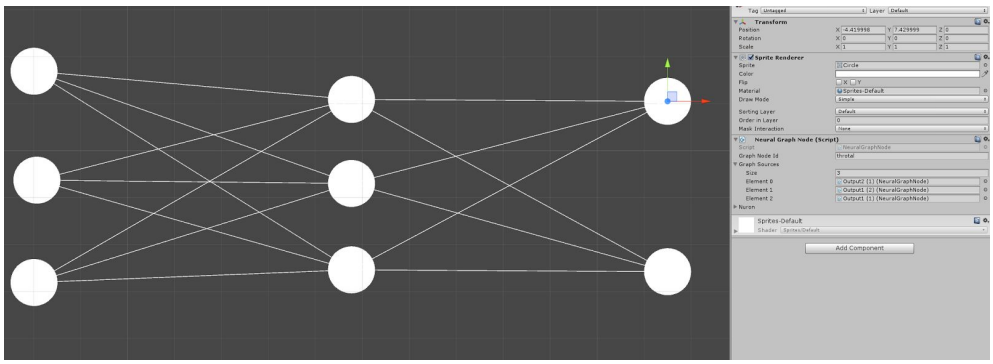
At this point, I was fairly sure I understood the basics of how a neural network is suppose to work. So I decided to pick an AI to implement using the network.
This is when I saw a video[3] of a cars implementation, where raycasts were used to determine proximity to walls, than feeding that into a neural network.
Note, there were no "tutorial" videos explaining how these implementations worked, I just used what I already knew about neural networks, to make educated guesses about the implementation, and used those ideas as a starting point to my own implementation.

[3] "Deep Learning Cars - YouTube." 23 Oct. 2016, https://www.youtube.com/watch?v=Aut32pR5PQA. Accessed 20 Jan. 2018.

Implementing the Neural network.

The first thing I needed to make, was the chain node system, to allow for custom network configurations.

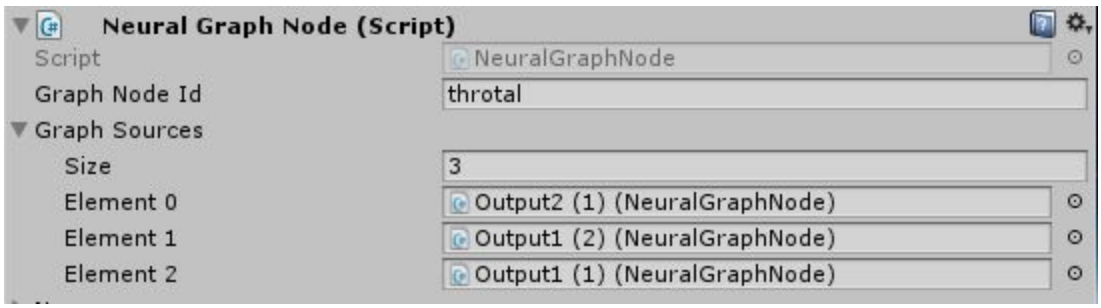
I felt that it would have been more intuitive for the system to be reusable so I could experiment with multiple types of network configurations as I had no idea what would work and what wouldn't. So I made a system for me to be able to setup the network in the inspector and the scene view in Unity so that I could iterate and try out different configurations of the network.



There are two different types of nodes:

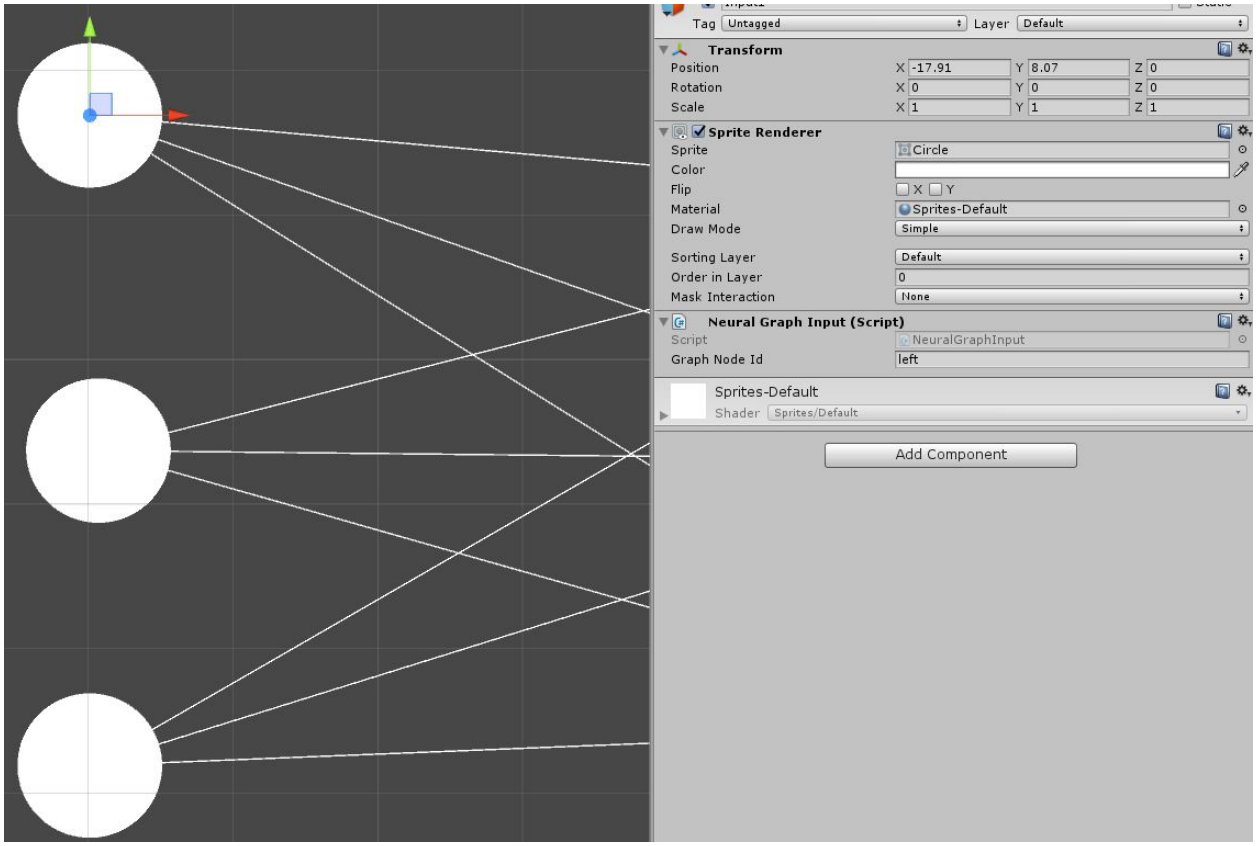
Neural Graph Nodes:

- When there values are requested, they poll an x amount of other nodes (**Notice the graph source array in the inspector in the picture below**) getting all their values and running them through a neural node (multiplies each of them by a weight and adds a bias, than runs them through a sigmoid before outputting them)).
- They also contain a Graph Node ID, this ID is used so there reference can be requested by a controller (these are generic and can be used in any AI, they are not specialized for the car steering behaviour).
- In this example, this node has a throttle ID, the steering behaviour script will poll this node to get set it's throttle every frame after updating the input Nodes
- These use a recursive process, where by polling the last graph node, every node will poll the previous node until the input nodes are reached, they don't need to poll anything as they just simply store the inputed value.



Neural Graph Inputs:

- These nodes only output a specific float that can be manually set by the controller script
- This node is used for input into the neural network.
- The Graph Node Id can be used to get a reference to this node from the script (In this example the normalized distance of the left ray is inputted into that node before the outputs are polled.)



The game AI itself:

Due to the nature of the next few things, I decided to explain the rest in video format :)
Make sure the audio is on, as the video audio is a bit quieter than expected.

(playlist)
https://www.youtube.com/playlist?list=PLdQ4YHKIaaAU_8DNcojl-1j2Q9HEY3Aey

Conclusion:

This concept of games AI turned out to be much more versatile and interesting than I expected. I thought I would have to run it for a few hours before I would get any responsible result. However it turned out be much better than I expected.

As you can see, I used instant respawn in the game, as oppose to the usual population based system, as when population is used, the game is only updating at the CPU's maximum capacity when the population is full, than when there is only a few cars left, the CPU is very under used. By allowing for instant respawn, it made training much much faster in real time. Which made it easier to test iterate :)

The use of evolution managed to get some really impressive results in very quick time. Even when tracks are swapped between games, once the network is trained enough it can play in a very effective way in lots of different scenarios.
Infact I was so impressed with it, I think I will look into using this sorts of an AI to train some other mini game AI's that I have in some of the other projects that i'm working on. I would train the network, save a couple of the different states and use them to play the game.

The best thing is that the network system itself is completely independent of the specific racing game, so I will be trying out different mini game types that I can abstract to see how far I could push this system.

These are the things that I wish I had time to do:

1. Making the evolutionary manager more generic so that it's even faster to reuse for different purposes (The neural network is generic, however the evolutionary manager could use to be more generalized, that way the only thing that needs to be customized is the controller of the specific controllers).
2. A more complex AI that has different objectives (Driving in the middle of the road, or minimizing turning).
3. A better process to eliminate bad versions using checkpoints.