INTRO

I decided to set out to build a Neural Network that allowed a car to drive around a track without user input. The intention was to create a racetrack environment and have a car be able to detect the edges of the track. It would spawn x-amount of cars per generation and would eliminate the cars in the generation that crashed. This would continue until the car had completed a full lap and it would race around the track following one of the successful lines. It would detect some form of barrier around the track with ray cast hit detection in Unity. This method is a common solution fit for this application and has some of the more useful and plentiful source material, example footage and example projects.

Then 2020 hit me and made things awkward and time short, so I decided I would scrap the Neural Network, but still produce an AI for a car to be able to both drive around a racetrack, but this time with hard-coded nodes to follow, and to be able to drive to randomly generated nodes in a field of obstacles.   
 I began by taking to the internet to find some basic concepts about AI in racing games. Several types and forms began to crop up multiple times.

BREAKDOWN

The most basic type of “AI” in a racing game is not truly AI or machine learning. It is a developer that records their best time around a track or a series of attempts in different cars and these attempts get recreated when the player plays the game in the form of opponent racers. Often games have “developer ghosts” too, which often serve as an extra challenge, such as in Crash Team Racing or Mario Kart.

Next, we come to the – what I have seen referred to as the – “Pearl Necklace” racing AI. This is where a “best racing line” is generated and the computer-controlled opponents will all essentially follow the same racing line with little to no deviation from it, all driving in a long line, forming the “pearl necklace”. This is, in essence, the approach I took. I placed nodes along a racetrack and the car would find the quickest line to drive along to get to the next node.   
 If multiple cars were spawned in, they would all follow roughly the same racing line, depending of course on where they were spawned. This was an effective and understandable beginner driving AI to tackle that was not just a prerecording. The car is fully autonomous around the track.

More so than this would be a system where cars would follow a variety of racing lines to make the dynamic flow of the race more interesting for the user to race against. This system would take in external stimuli, like other cars braking in front of them or crashes/objects around them, and respond to them real time. Neural networks are ideal for this sort of simulation. It can train the opponents to act more “human-like” and provide a racing environment that more accurately mimics that of an online/party setting.

MY AI

The AI that I implemented was very much a Reactive AI. Running it around the track multiple results in the same reactions and driving lines. Dropping in new obstacles changes the racing line, but it will then react to that new obstacle in the same way every lap. Is this an ideal solution to a driving AI? Absolutely flipping not. Is it the best I could produce in the timescale with my current understanding of programming AI cars? Abso-flipping-lutely.

I followed a tutorial to get the primary understanding of how to get a car to find a node, traverse to it then find the next node. Onto this, I added the ability for a script to generate obstacles and nodes in random coordinates in a given area. The car would receive this list of random nodes and begin to traverse between them, avoiding the obstacles along the way.

I implemented a hit scan system that was originally intended for the neural network, but adapted it to fit the purpose of making the car avoid obstacles such as barriers and trees. Instead of having multiple cars spawn in a generation, I committed to one car that was tweaked until it could drive itself around the track. The centre of mass, suspension height and tension, mass of the car and tire slipping values were edited until the car could travel entirely around the track. These parameters could also be controlled by the original neural network to improve the time it takes for the optimal path to be traversed.

The ideal AI would have generated the ideal path, the optimal settings for the car and then been able to adapt to external stimuli, such as a player crashing into the AI, then adjusting the current route or calculating a path back to the original. It would also accelerate and decelerate to match the ideal speed for each corner. Currently the car turns to avoid an object and this circular motion automatically slows down the car, but does not apply the brakes. The user can manually stop the simulation by pressing “B” to make the car brake. The user can also reset the simulation by pressing “R”. This generates new nodes and new barriers in the second scene. Sometimes this is necessary as if the car gets completely stuck between several barriers and cannot get out. Ideally it would reverse or there would be a system in place to prevent barriers from spawning near each other, but time disallowed these systems to be implemented.

As it stands, the AI is fit for purpose, if not limited. It successfully demonstrates a car travelling around either a pre-determined or randomly generated path, whilst avoiding objects. It is not a neural network, but is not user controlled – except for restarting or pausing the simulation if the user wishes to. The footage of the AI shows there is room for improvement, but is definitely better than having a pre-recorded “ghost”.