Driverless Car in Unreal Engine using Reinforcement Learning

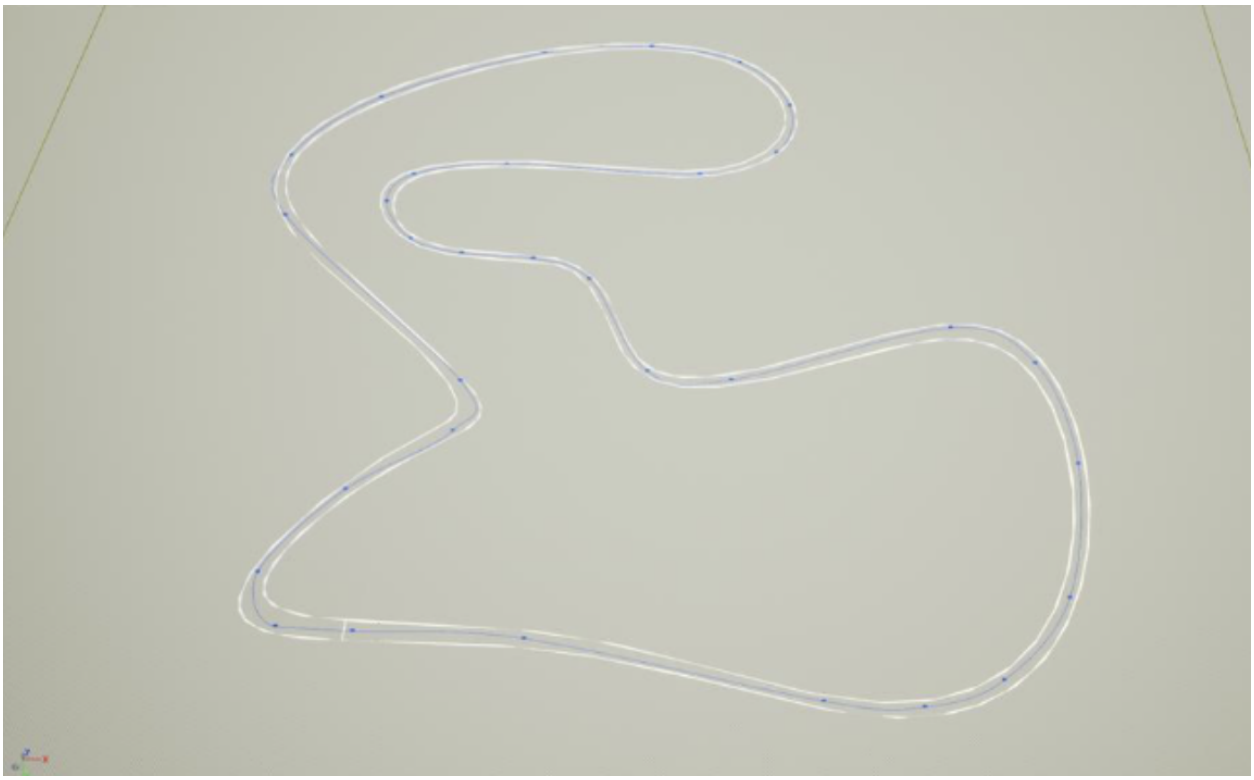
Abstract:

In recent years, the magnitude of technological advancement has made unprecedented strides especially in the gaming and autonomous driving industries. These advancements are only possible due to cutting edge artificial intelligence and machine learning research, and thus I wanted to experiment with using reinforcement learning (a subset of machine learning) to code a self-driving car. Since creating a driverless car in reality involves many more fields than those I am experienced with, as well as involving the requirement to comply with various laws and regulations, I decided to do the next best thing and code the self-driving car as a game in Unreal Engine. The motive of this project is two fold, as it also serves as an experiment in using machine learning for applications in the gaming industry. As demand for more realistic background play increases in video games, AI can be deployed to give the users what they desire, resulting in many game development studios employing various ML algorithms. Right now the industry standard for simulating car driving in video games is a structure known as the behavior tree, however, these trees bring an array of problems with them frequently resulting in glitches. As I discovered, reinforcement learning can be used to simulate a self-driving car and with enough training samples it can achieve really good accuracy, which means that the car can drive itself without any collisions around the developed track.

Introduction:

The entire point of this game is to see if a virtual car can effectively learn from its environment and correctly make a decision as to whether it should turn left, right, or continue straight in order to continue without a collision. A car takes actions on a race track by acting on the feedback the environment gives back to the car. The car is designed as a class of its own, and each instance of this class provides a simulation of the self-driving car. There are multiple trajectories that can result in the car colliding with the walls of the racetrack, and thus certain questions need to be answered such as - Did

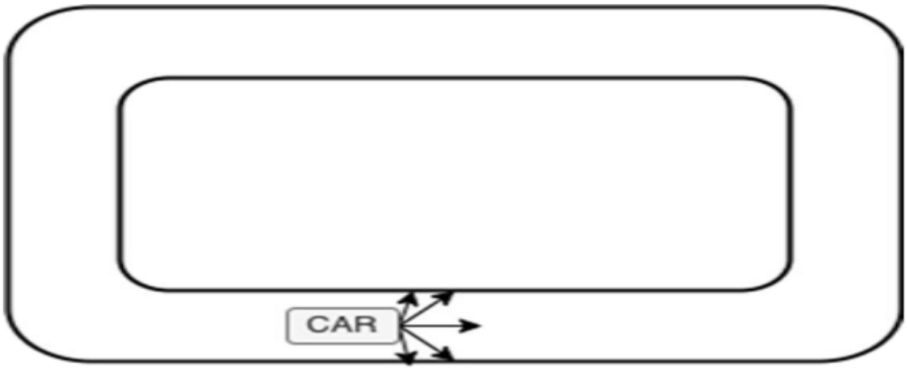
the car collide? If so, what was its speed at the time of collision? What is the location of the collision? - to calculate, through trial and error, the distance of the car to the wall. In subsequent trials, the car has learnt of this collision and it will try to avoid it by instead turning left or right as determined by a neural network. For the car to get a 100% accuracy, it will have to drive without a collision for at-least one lap of the racetrack.



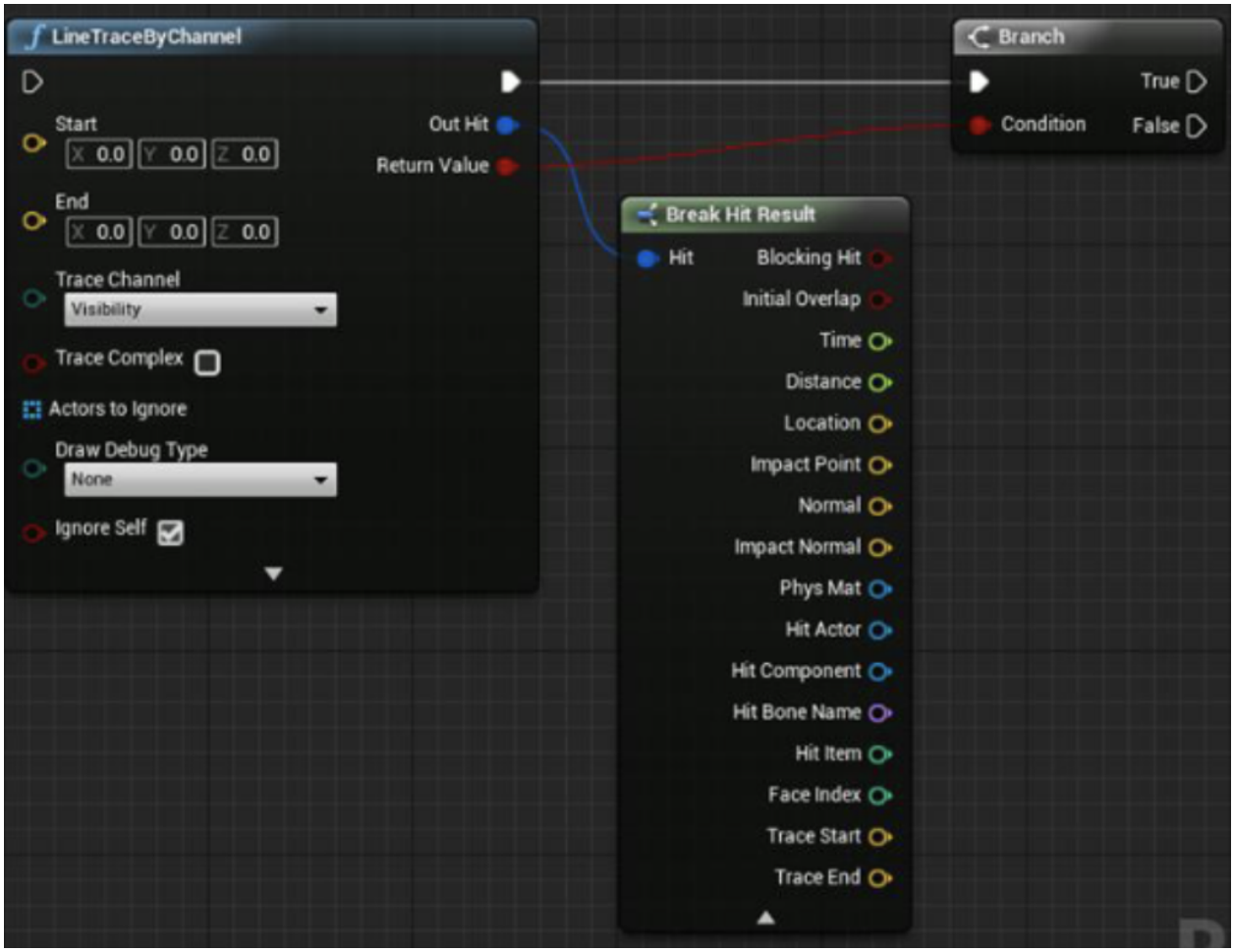
Designed Racetrack

<u>Dataset and Features:</u>

The data is generated through interactions of the car with the walls of the racetrack. The car has line traces that act like sensors and help us measure the distance to the nearest obstacle as shown below:



The more line traces there are on the car, the more inputs will be fed into the neural network since more data is being collected. The screenshot below shows how the line traces give out data such as distance, location, time, etc. obtained when a collision occurs so that they can be used to ensure the collision doesn't occur in the future.

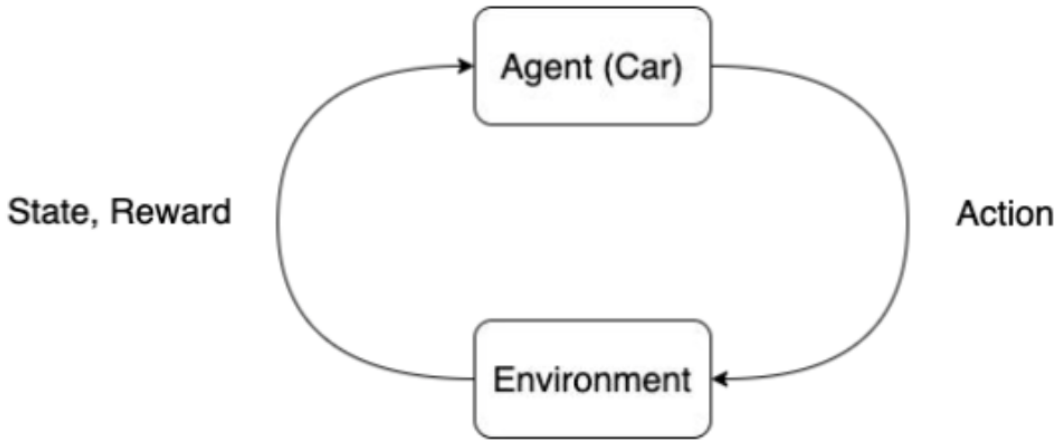


Methods:

Reinforcement Learning (RL):

RL is a subset of machine learning that I use in this project. RL is heavily used to determine how agents (the car in this project) should take actions based on the environment (racetrack in this project), and it helps in maximizing some part of the total reward. It works with the use of rewards and punishments for the agents, who are

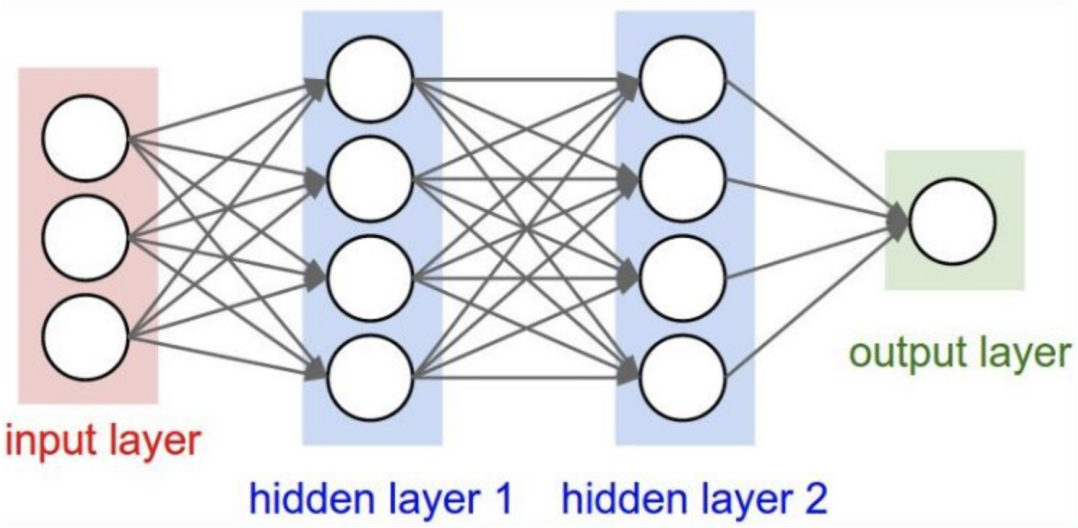
rewarded for performing correct actions, while being punished for wrong moves. These rewards and punishments basically serve as indicators for the agents to figure out which are the correct moves, and hence the agent will try to maximize the correct actions and minimize the wrong ones. In this project, the concept of reinforcement learning is implemented with a neural network as a deep learning method.



An example of how basic Reinforcement Learning workz

Neural Networks:

Neural Networks, is a subset of machine learning, try to mimic how the human brain behaves, by connecting different nodes to each other just like the human brain does with neurons. Neural networks are made up of different layers: an input layer, an output layer, and one or more hidden layers. Each node is in one of these layers and connects to another node and it will have a given weight and threshold. When the output of a given node is above the specified threshold, it will pass data onto the next node it connects to being passed across the layers; else the node will not pass the data onto the next layer. In this project the hidden layers of the neural network implement an algorithm known as the genetic algorithm.



Typical Neural Network structure

Genetic Algorithm:

Genetic algorithms are designed according to the law of evolution of organisms as seen in nature. These algorithms can be used to search for optimal solutions, by simulating the natural selection and evolution process as quantified in Darwin's theory of evolution. The algorithm has four parameters: Initial population, selection, crossover, and mutation. I use randomly produced numbers between -1 and 1 to initialize the population. A function that returns a probability (out of 1) that is indicative of the percentage of track driven without a collision is used as the selection; genes are also exchanged among parents to get new instances and this is the crossover. Random numbers are once again used to change some of the genes randomly to factor in the mutation.

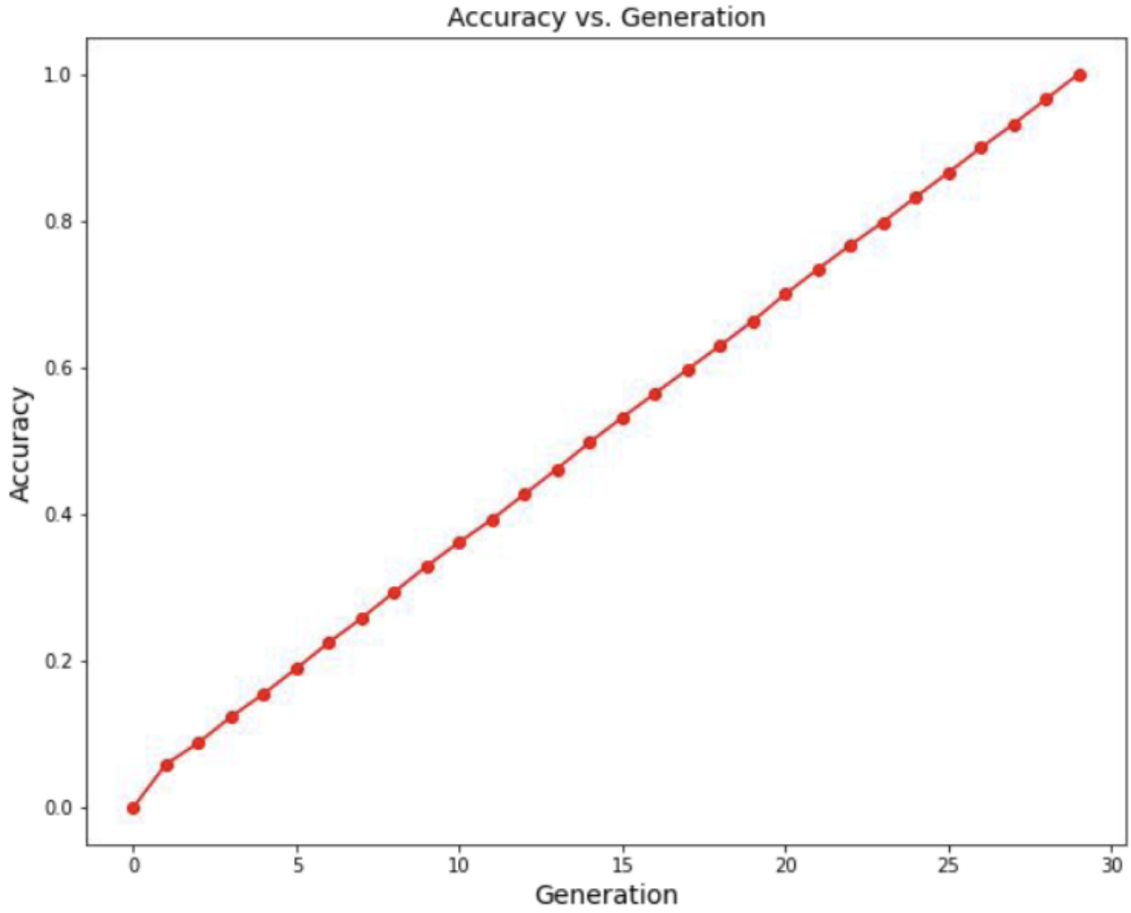
Discussion:

At the starting point, multiple instances of the car class are created, and they have the initialization process as mentioned in the genetic algorithm and it has to learn to drive itself. If a car gets into a collision, it stops driving. If all the cars get into a collision, a new generation of cars with new weights will be created at the starting point and continue the training process. The percentage of the track that a car completes is its

accuracy, therefore I want to see if any car can achieve 100% signifying that it went through an entire lap without any collisions.

Results:

The following graph of accuracy vs generation (or number of cars) tells a clear story. It is possible for a car to learn to drive itself and complete at least one lap of the track without collisions.



As more instances are created representing that the model has training for longer, the number of cars which can successfully complete a lap without collisions is likely to increase as well. The instances which can achieve 100% accuracy can be used as self-driving cars in another game where a self-driving car is desired.