Self-Driving Car with Reinforcement Learning in Unreal Engine

Kaiyan Zhang, Shuaizhao Li

Abstract—As the development of hardware configuration, more and more interesting features in video games can be developed. With more powerful CPUs and GPUs, machine learning can be implemented in artificial intelligence module of game development field. Artificial intelligence in game development field has more potential after the joining of machine learning methods and has to be smarter with higher demands of players in the market. Many thoughts of reinforcement learning fit for the development of artificial intelligence in video games. A self-driving car with reinforcement learning can be introduced instead of behavior tree in some situations.

Keywords—Reinforcement Learning, Self-driving Car, Genetic Algorithm, Game Development

1 Introduction

In order to win the game, one of the car instances should learn from the road environment to go forward or turn left or turn right without any crash and then go at least one round after loops of neuron network training.

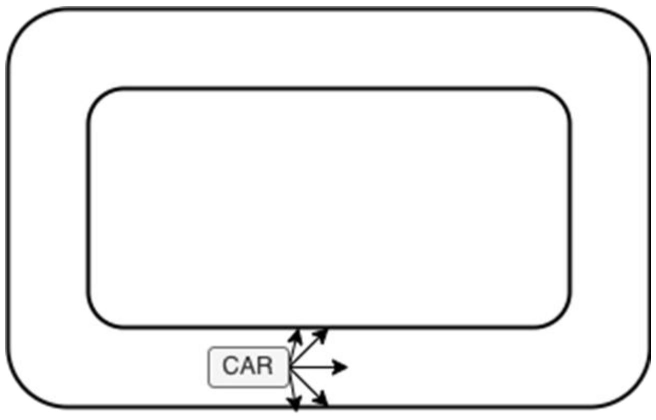


Fig 1: Top Down View

Self-driving car instances need learning to go forward and turn left or right for safety. Neural networks can provide us with the approach of learning to be smarter. We will implement reinforcement learning for this application project, which is popular in game artificial intelligence development field.

Agents (Cars) take actions in the race track environment and the environment can give feedback to the agents. The actions can be the output of the neuron network and the feedback can be the input of the neuron network.

For each car class, there are numbers of line sticks (trace) which can return hit results. What we need are “If Hit?” (Boolean), “Velocity” (Vector) and “Location” (Vector) to perform the distance to the nearest obstacle. Then a neural network is used to output a predicted going forward, turning left or turning right.

2 Related work

In some of the car racing games and modern city open world games, vehicle NPCs (Non-player Characters) are implemented with behavior trees, which can be complicated and hard for iteration and maintenance. Even for commercial projects, chaotic project management on behavior tree module of artificial intelligence can cause lots of bugs and bring bad experiences with players, for example, early versions of Cyberpunk 2077.

To solve these development problems, many game companies have founded machine learning departments for application research. Our job is to implement a simple self-driving car with machine learning in Unreal Engine 4, a mainstream commercial game engine.

We have developed some indie games and modules of commercial games with Unreal Engine 4, so we know something about this engine. For background learning, we have read some materials and try to create machine learning environment in this game engine.

Many game companies may do the application research and make it confidential. So there are not many documents on the Internet. According to my past internship experience, one of my research topics is vehicle system, in which the behavior tree is typically used. So far even if behavior trees have many drawbacks, it is still widely used and has the most technical support for the engine team.

Behavior Trees assets in Unreal Engine 4 can be implemented to create artificial intelligence for non-player characters in the project [1]. The Behavior Tree asset is used to execute branches containing logic and to determine which branches should be executed, the Behavior Tree relies on another asset called Blackboard which is regarded as the "brain" for a Behavior Tree.

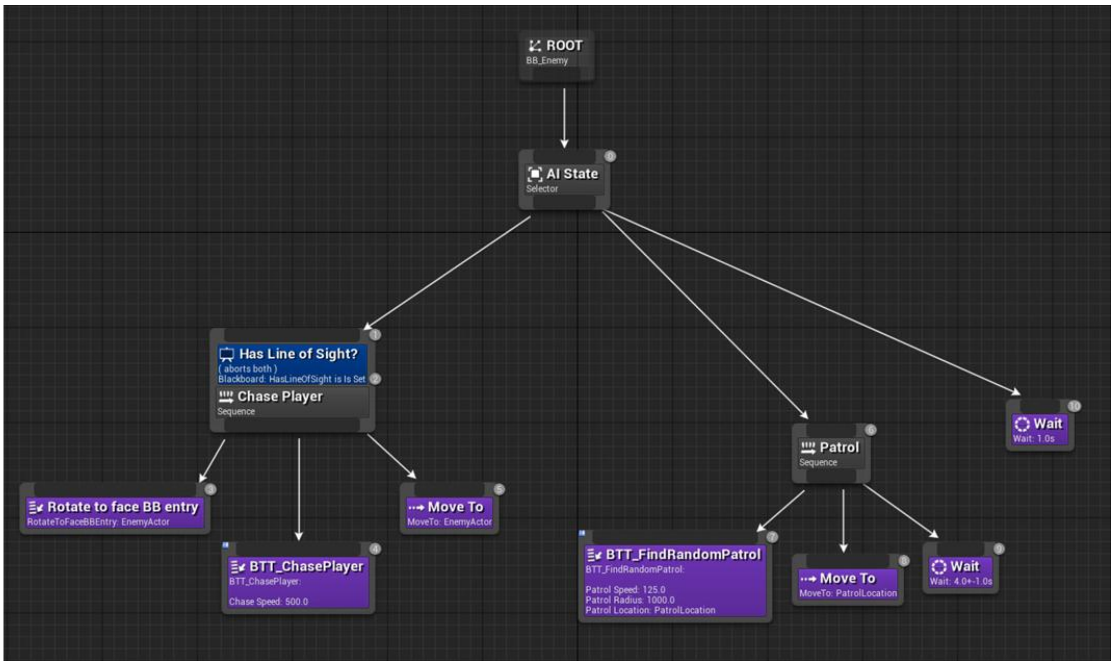


Fig 2: Behavior Tree Example

When looking at methods of machine learning, reinforcement learning has usually been fit for artificial intelligence in game development because there are agents and environment in the game level. In Unreal Engine 4, blueprint can be a convenient tool for implementing reinforcement learning [2]. There is a Environment Query System (EQS) in Unreal Engine 4 which can be used to query the environment for data [3]. The data can then be implemented to provide the AI with data used in the decision making process on how to proceed.

To make machine learning implementation better, many developers have successfully made open source python pools for Unreal Engine 4 [4]. Also, C++ can be used for neural network with fast computing speed. There may be something different when programming with C++ in Unreal Engine 4, so the documentation has to be consulted with [5]. In this project, genetic algorithm is implemented in our neural network for decision, which is a different way compared with behavior trees [6].

3 Dataset and Features

The data can be generated when the car interacts with the objects. For the car class in the project, line sticks (trace) can be used at the front of the car to measure the distance to the nearest object.

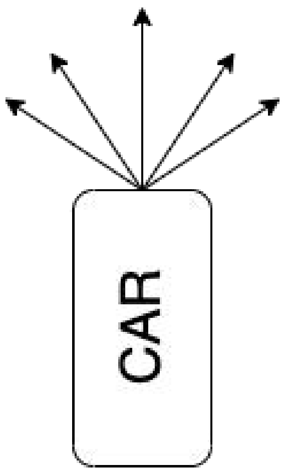


Fig 3: Car Class

According to line trace blueprint node, after breaking the hit result struct, we can directly get the distance data from "distance", which is a float type. If one car has 5 sticks, there are 5 inputs in the neural network.

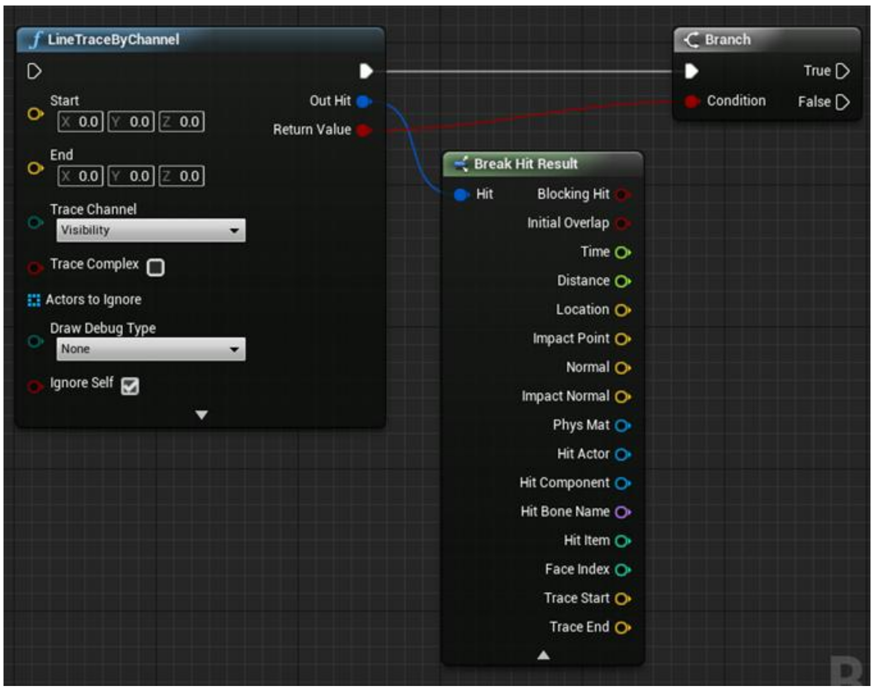


Fig 4: Line Trace Blueprint Node

4 Methods

The method tried in this project is genetic algorithm. The algorithm is designed and proposed according to the evolution law of organisms in nature. It is a computational model of the biological evolution process that simulates the natural selection and genetic mechanism of Darwin's theory of biological evolution. It is a method to search for the optimal solution by simulating the natural evolution process. The algorithm converts the process of solving the problem into a process similar to the crossover and mutation of chromosomal genes in biological evolution through mathematical methods and computer simulation operations. When solving more complex combinatorial optimization problems, better optimization results can usually be obtained faster than some conventional optimization algorithms.

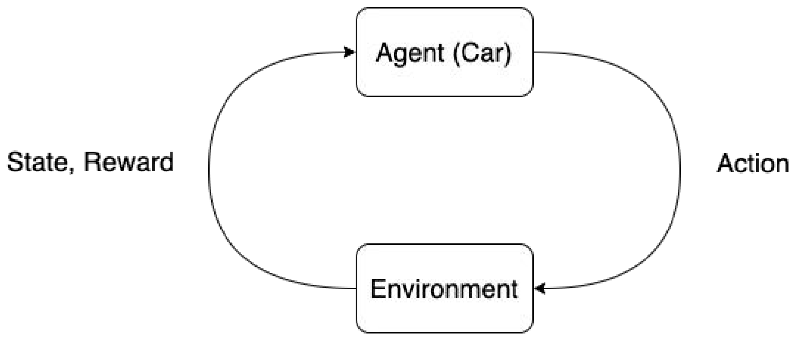


Fig 5: Reinforcement Learning Example

Genetic algorithm is implemented in the hidden layer. There are 4 parts: Initial population, Selection, Crossover and Mutation.

In this project -1 and 1 are used to represent the gene value. Then we need to design the neural network. If we use 4 line traces to measure and return the hit results, there are 4 inputs and 3 outputs. We get 4 layers in the hidden layers. To begin with the initialization, we can choose a random number in the interval [-1, 1] to initialize the weight values.

For selection, The fitness function can return a probability that passes genes to the next generation. The accuracy variable is used to show the percentage of the track travelled. If it completes the whole race track, we get 1(100%). The result will be in the interval [0, 1].

For crossover, genes are exchanged among parents to get a new instance which is made by progenitors.

For mutation, random number generator are used to simulate this process by changing some of the genes.

The genetic algorithm process is referenced with the paper talking about genetic algorithm method. In the paper, for selection process, it puts the emphasis on diversity and accuracy. For my project, I put the emphasis on accuracy because the goal for us is to complete the whole race track with 100% of accuracy, which serves as the standard.

All the code I used with C++ and blueprint is from Unreal Engine Documentation. And the blueprint is a high packaged C++. Now Unreal Engine 4 is an open source game engine and can be built from source on github. The Unreal Engine 4 source version I used from my past internship is 4.24. So it's better to use 4.24 or higher. Unreal Engine 5 has a different architecture and it is not recommended.

Here's the link of github to build Unreal Engine 4.24 from source with Visual Studio on Windows:
https://github.com/EpicGames/UnrealEngine/tree/4.24

5 Experiments/Results/Discussion

The race track constructed is shown in the figure. There are many types of bends which can evaluate our neural network better. Arrays of car instances are generated at the start point. A controller class is needed to mange generations. For each car instance, it has functions of initialization and learning, which is genetic algorithm.

The PlayerController uses functionality for taking the input data from the player and translating it into actions and is the interface between the Pawn (can be regarded as the car class in this project) and the human player who controls it [7]. As there are multiple players on one game client, it's better to handle input in the PlayerController. For this case, the PlayerController represents the will of human players and decides what to do. Then it issues commands to the Pawn.

The parameter "accuracy" can represent the degree of completion. From the figure there are points labeled as the percent of accuracy. In Unreal Engine 4, this is called "Spline", which is a convenient tool to construct curves. To better represent a curve, points (indexes) are added to split the curve into "components". At the start point, the first point is 0% of accuracy and at the destination, the last point is 100% of accuracy. The spline actor is referenced with the car class where the car class can measure if the car instance gets to the point.

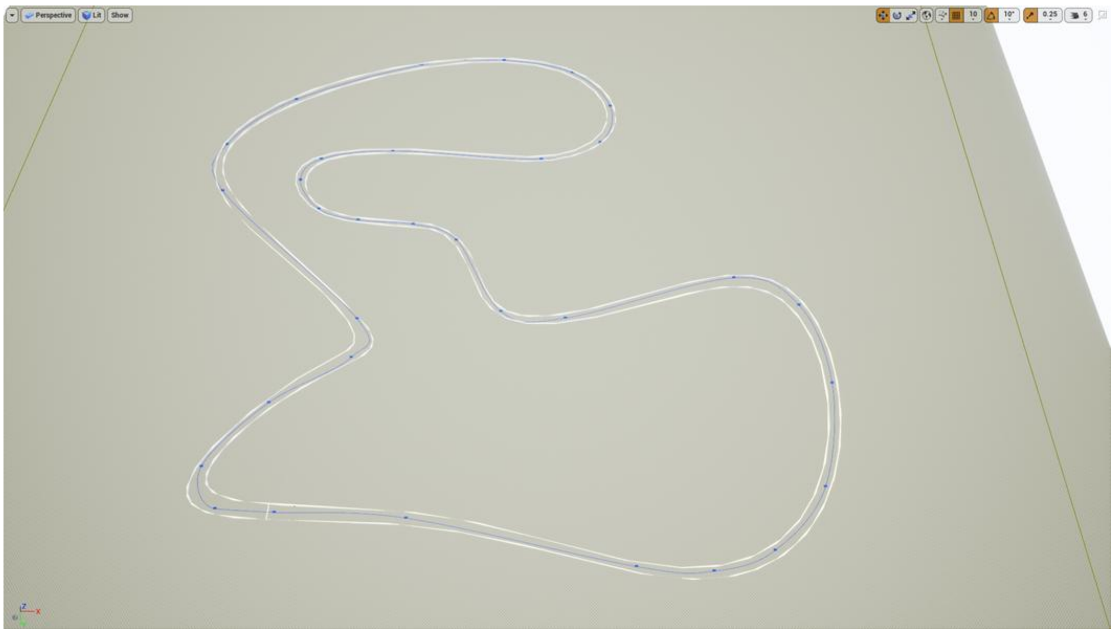


Fig 6: Race Track Top View

There are 2 videos uploaded to Youtube, which show part of the training process and the winning situation.

Here's the link of part of the training process:
https://www.youtube.com/watch?v=tk9mmJ2IjUE
Here's the link of winning process:
https://www.youtube.com/watch?v=AyLTHGC5tVc

For the training process, when a car instance collides with a object (wall), it stops immediately and when all the car instances stop, there will be a next generation with changed weights. An array of car is generated and the training process continues. With loops of training process, we can see if there is a car instance gets 100% accuracy.

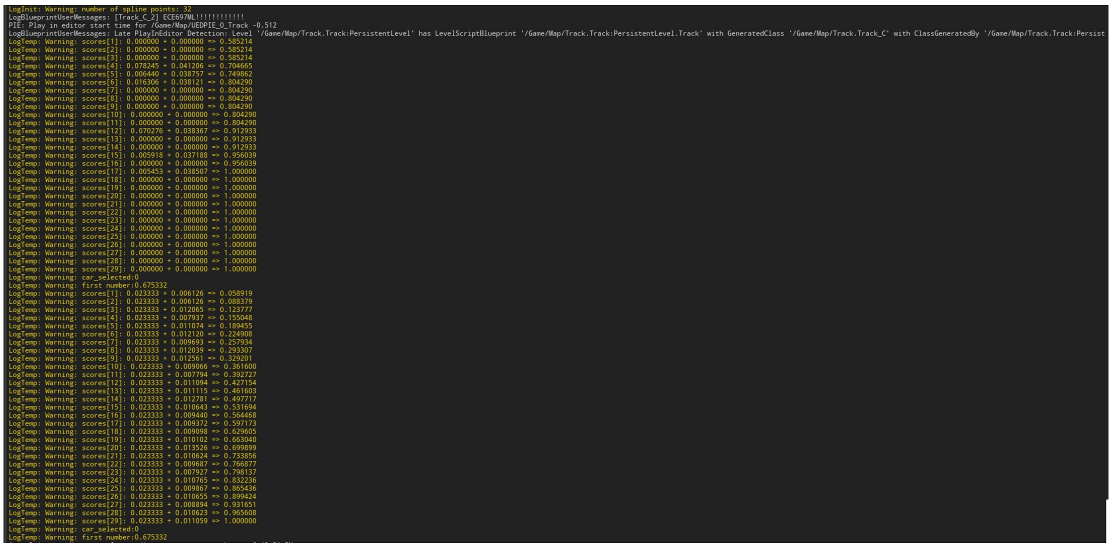
There are hyper-parameters used to improve the accuracy. We can focus on the car class and genetic algorithm.

For car class, there are number of car instances generated and rotation degree of the race track.

For genetic algorithm, there are selection threshold, crossover limitation and mutation random generator.

Some hyper-parameters can't improve the accuracy. For example, the speed of car class. However, it can still influence the speed of convergence.

We don't get a python environment for this projects, so some specific information can only be printed in output log. The figure shows the print of the highest accuracy of each generation for an array of car instances.



```
LogInit: Warning: number of spline points: 32
LogBlueprintUserMessages: [Track_C_2] ECE697ML!!!!!!!!!!!!
PIE: Play in editor start time for /Game/Map/UEDPIE_0_Track -0.512
LogBlueprintUserMessages: Late PlayInEditor Detection: Level '/Game/Map/Track.Track:PersistentLevel' has LevelScriptBlueprint '/Game/Map/Track.Track:PersistentLevel.Track' with GeneratedClass '/Game/Map/Track.Track_C' with ClassGeneratedBy '/Game/Map/Track.Track:Persist
LogTemp: Warning: scores[1]: 0.000000 + 0.000000 => 0.585214
LogTemp: Warning: scores[2]: 0.000000 + 0.000000 => 0.585214
LogTemp: Warning: scores[3]: 0.000000 + 0.000000 => 0.585214
LogTemp: Warning: scores[4]: 0.078245 + 0.041206 => 0.704665
LogTemp: Warning: scores[5]: 0.006440 + 0.038757 => 0.749862
LogTemp: Warning: scores[6]: 0.016306 + 0.038121 => 0.804290
LogTemp: Warning: scores[7]: 0.000000 + 0.000000 => 0.804290
LogTemp: Warning: scores[8]: 0.000000 + 0.000000 => 0.804290
LogTemp: Warning: scores[9]: 0.000000 + 0.000000 => 0.804290
LogTemp: Warning: scores[10]: 0.000000 + 0.000000 => 0.804290
LogTemp: Warning: scores[11]: 0.000000 + 0.000000 => 0.804290
LogTemp: Warning: scores[12]: 0.070276 + 0.038367 => 0.912933
LogTemp: Warning: scores[13]: 0.000000 + 0.000000 => 0.912933
LogTemp: Warning: scores[14]: 0.000000 + 0.000000 => 0.912933
LogTemp: Warning: scores[15]: 0.005918 + 0.037188 => 0.956039
LogTemp: Warning: scores[16]: 0.000000 + 0.000000 => 0.956039
LogTemp: Warning: scores[17]: 0.005453 + 0.038507 => 1.000000
LogTemp: Warning: scores[18]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[19]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[20]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[21]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[22]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[23]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[24]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[25]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[26]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[27]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[28]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: scores[29]: 0.000000 + 0.000000 => 1.000000
LogTemp: Warning: car_selected:0
LogTemp: Warning: first number:0.675332
LogTemp: Warning: scores[1]: 0.023333 + 0.006126 => 0.058919
LogTemp: Warning: scores[2]: 0.023333 + 0.006126 => 0.088379
LogTemp: Warning: scores[3]: 0.023333 + 0.012065 => 0.123777
LogTemp: Warning: scores[4]: 0.023333 + 0.007937 => 0.155048
LogTemp: Warning: scores[5]: 0.023333 + 0.011074 => 0.189455
LogTemp: Warning: scores[6]: 0.023333 + 0.012120 => 0.224908
LogTemp: Warning: scores[7]: 0.023333 + 0.009693 => 0.257934
LogTemp: Warning: scores[8]: 0.023333 + 0.012039 => 0.293307
LogTemp: Warning: scores[9]: 0.023333 + 0.012561 => 0.329201
LogTemp: Warning: scores[10]: 0.023333 + 0.009066 => 0.361600
LogTemp: Warning: scores[11]: 0.023333 + 0.007794 => 0.392727
LogTemp: Warning: scores[12]: 0.023333 + 0.011094 => 0.427154
LogTemp: Warning: scores[13]: 0.023333 + 0.011115 => 0.461603
LogTemp: Warning: scores[14]: 0.023333 + 0.012781 => 0.497717
LogTemp: Warning: scores[15]: 0.023333 + 0.010643 => 0.531694
LogTemp: Warning: scores[16]: 0.023333 + 0.009440 => 0.564468
LogTemp: Warning: scores[17]: 0.023333 + 0.009372 => 0.597173
LogTemp: Warning: scores[18]: 0.023333 + 0.009098 => 0.629605
LogTemp: Warning: scores[19]: 0.023333 + 0.010102 => 0.663040
LogTemp: Warning: scores[20]: 0.023333 + 0.013526 => 0.699899
LogTemp: Warning: scores[21]: 0.023333 + 0.010624 => 0.733856
LogTemp: Warning: scores[22]: 0.023333 + 0.009687 => 0.766877
LogTemp: Warning: scores[23]: 0.023333 + 0.007927 => 0.798137
LogTemp: Warning: scores[24]: 0.023333 + 0.010765 => 0.832236
LogTemp: Warning: scores[25]: 0.023333 + 0.009867 => 0.865436
LogTemp: Warning: scores[26]: 0.023333 + 0.010655 => 0.899424
LogTemp: Warning: scores[27]: 0.023333 + 0.008894 => 0.931651
LogTemp: Warning: scores[28]: 0.023333 + 0.010623 => 0.965608
LogTemp: Warning: scores[29]: 0.023333 + 0.011059 => 1.000000
LogTemp: Warning: car_selected:0
LogTemp: Warning: first number:0.675332
```

Fig 7: Output Log

The plot below shows the with more generations the highest accuracy, which is the highest completion percent of the race track.

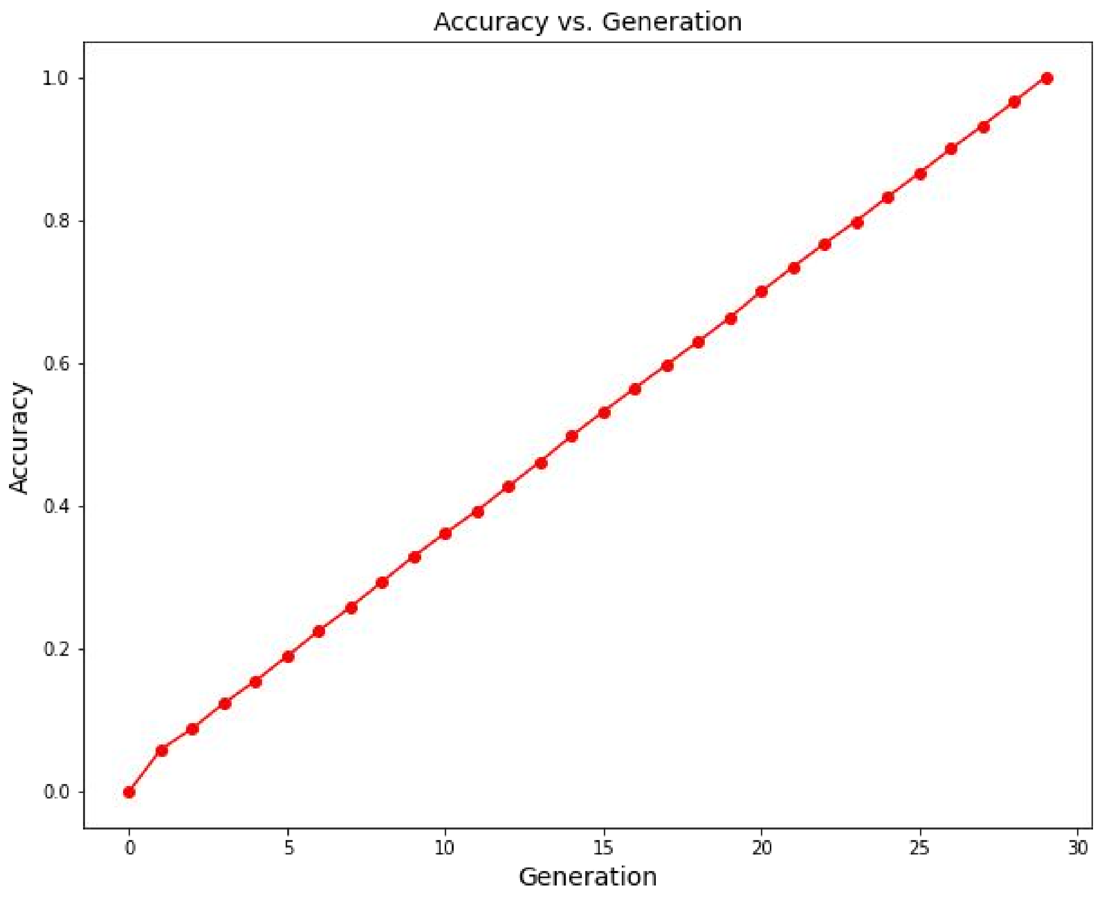


Fig 8: Accuracy vs. Generation

6 Conclusion/Future Work

From our experiments, it appears that with about 30 generations, at least one of the car instances can complete 100% the race track without crash and it may continue with loops. From the plot above we can see that the last generation number is not that large. For the completion model, it can be used as the instance of the artificial intelligence in our real game level.

For future work, if we want to develop a real race game or open world modern city game, artificial intelligence cars need to avoid dynamic obstacles, for example, passerby. They move dynamically, different from only the static wall in this project. To update the car class, an update decision pattern need to be implemented.

7 Contributions

Kaiyan Zhang:
Unreal Engine configuration and debugging.
Documentation.
Shuaizhao Li:
Test and evaluation.
Documentation.

References

[1] https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/BehaviorTrees/
[2] Reece A. Boyd, “Implementing Reinforcement Learning in Unreal Engine 4 with Blueprint”, Middle Tennessee State University, 2017.
[3] https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/EQS/
[4] https://github.com/20tab/UnrealEnginePython
[5] https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/
[6] Daniel López Montero, “Artificial Intelligence Self Driving Car in Unreal Engine 4”, Instituto San Mateo, 2018.
[7] https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Framework/Controller/PlayerController/

Code

Before opening this project, a 4.24 or higher version (5.0.0 not included) Unreal Engine 4 source is required.

Here’s the github link to access the project:

https://github.com/KyleZZZZZ/Self-Driving-Car

Open “.uproject” file then Unreal Engine Editor is opened. Press “Play” to start the training process.

When playing, press “F8” to eject. Hold right mouse button and move around the mouse to look around. Use “WASD” to move the developer camera.