AI-based 3D Game Simulator

A Senior Honors Thesis

Submitted in Partial Fulfillment of the Requirements  
for Graduation in the Honors College

By

Yuhang Liao

Computer Science Major

The College at Brockport  
May 10, 2019

Thesis Director: Dr. Ning Yu, Associate Professor, Computer Science

*Educational use of this paper is permitted for the purpose of providing future   
students a model example of an Honors senior thesis project.*

Contents

[1. Introduction 2](#_Toc7997433)

[1.1 Background 2](#_Toc7997434)

[2. Games and Simulators 3](#_Toc7997435)

[2.1 The relationship 3](#_Toc7997436)

[2.2 An Example: AlphaGo 3](#_Toc7997437)

[3. Two Examples 4](#_Toc7997438)

[3.1 Cars in Random City 4](#_Toc7997439)

[3.1.1 Policy-based AI 5](#_Toc7997440)

[3.1.2 Physics-based Car Driver Class 7](#_Toc7997441)

[3.1.3 Procedural Generation Algorithm 9](#_Toc7997442)

[3.1.4 Conclusion 11](#_Toc7997443)

[3.2 Physics-based ML AI Car 11](#_Toc7997444)

[3.2.1 Basic Idea of Reinforcement Learning 12](#_Toc7997445)

[3.2.2 The Map Structure 13](#_Toc7997446)

[3.2.3 Core Algorithm of the Agent 13](#_Toc7997447)

[3.2.4 Pros and Cons 22](#_Toc7997448)

# Introduction

## 1.1 Background

The world is full of smart devices now, and many of them have already featured with Artificial Intelligence (AI), which makes devices be more friendly and accessible to more and more people. It truly pushes the technology forward.

Basically, there are three components of AI – computation power, algorithms and data. Computation power usually means the hardware to run AI programs, such as CPU and GPU. Algorithms are instructions telling computers how to run the AI program. Data is the fuel for AI program, especially for AI powered by machine learning, because AI program requires huge amount of data to be trained. It is also the most important that this paper will talk about.

Usually, AI companies are collecting data they want to use from users, therefore it is a severe violation of personal privacy. For example, when a user searches something on Google, then Google may show some advertisements based on what the user searches. The user unconsciously shares his/her data to Google, and he/she cannot control how Google will use his/her data. However, when the user loses something, he/she will also gain something. For example, if the user wants to search nearest restaurant. Google can recommend some good restaurants to user based one Google’s huge database. It is very helpful to make sure people not go to bad restaurants.

So, is there any possibility to provide users a good service without collecting huge amount of Data? It is the topic that this paper will talk about. The paper will show two computer games as examples to prove the idea. The two games are randomized city and machine learning car. Both are games with cars, so they are also called car simulators.

# Games and Simulators

## 2.1 The relationship

The idea about making games as the simulator seems to be very crazy. However, it is truly implementable. For example, AI can play racing game to learn how to drive a car. Usually, the roads in racing games are much more irregular and difficult to drive than common roads in cities. Therefore, if computers can learn how to drive in such difficult games, they may also be able to drive real cars on real street. It is the idea that this project is trying to prove.

Secondly, computers can generate more and more data while they are playing games. For example, while computers are playing racing games, they can generate data such as suitable speed for the street, how much steering angle should be applied to make a good turn or how much torque that can drive the car safe and sound. Therefore, researchers can collect data from computers while they are playing cars in the simulators rather than people. Although training an AI model without any data from people is impossible, it can reduce the amount of data from users at least. As the result, researchers can get the same result with minimum data from people.

## 2.2 An Example: AlphaGo

The best example for this idea is AlphaGo [1]. AlphaGo is an AI program playing Go game made by DeepMind partnered with Google, which is one of the hardest board games in the world. It successfully beat human champions, which made the whole world surprised. In the early versions of AlphaGo, researchers let the machine play millions and millions of Go games. They also gathered incredibly huge amount of data from professional human players. While the robot was trying to beat human players, it was also trying to beat itself. The biggest advantage that computers have but people don’t have is that computers can repeatedly play Go games fast and nonstop. Therefore, it could play millions of times more of Go games than human in one day. That was also another way where the data set came from.

Since AlphaGo could generate data while it was playing games by itself, did it really require data from human? AlphaGo zero [2], which was the latest version of AlphaGo family proved that AI program can still generate data by itself without humans intervene. AlphaGo started everything from scratch without any professional data from human players. It began with learning basic rules of Go game, tried to get maximum scores and beat itself. Therefore, it tried advanced techniques of Go game that made it win the game easily. The robot tried beat itself repeatedly, so it did not require any data from human. Then, surprisingly, it defeated its ancestors, who required data from professional human players and beat human champions, using very short learning time. It truly proved that AI program could still be trained with scarce data resources.

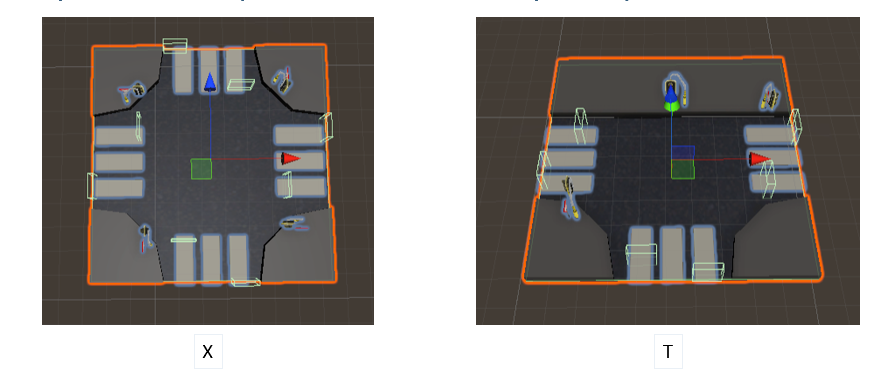
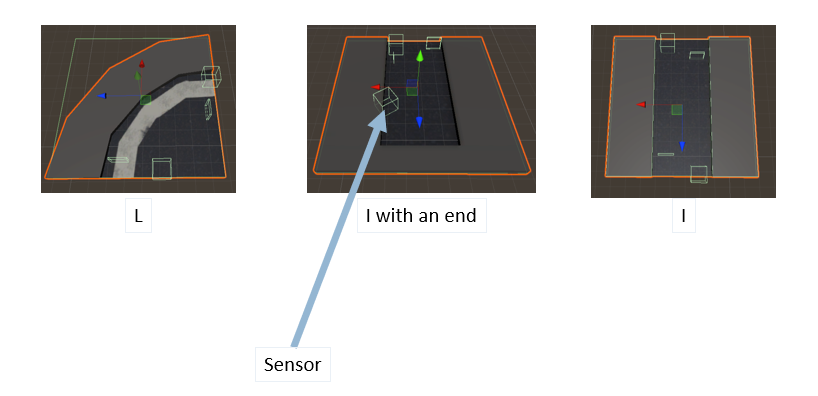
# Two Examples

## 3.1 Cars in Random City

The first project I will introduce is about cars in a randomized city. Traditional AI algorithms are used in this project, which means this project doesn’t have anything about machine learning. Cars are automatically driven by themselves following hard-coded policy-based rules. Every decision and behavior that determines what is the next step are totally predefined in code. Therefore, this project doesn’t require any external or internal dataset. Notice, any code provided below is just for demonstration, it is not the same code used in the real program.

### 3.1.1 Policy-based AI

The city is also randomly generated based on random generation algorithm. The algorithm can generate pseudo-random numbers, which means it is not truly random; it is random enough for this project. Buildings, trees and other miscellaneous game objects are also randomly generated in the game map; however, they are just decoration of the game, they don’t play any important role in AI algorithms. Although roads are generated by machine, but all roads are regular, every turn is strictly 90 degree. There are five shapes of roads:



Each road has some green objects called sensors. When a car collides a sensor, the sensor will tell what is the next waypoint that the car should go. The first three roads are very simple, because they don’t have traffic lights. Car can just move forward without worries. Although the second one is the place that car makes U-turn, sensors just need to tell car the speed and angle information to finish the job.

The last two roads include traffic lights, so cars may need to wait for the lights before they go. Here is the hardest part of this project that AI should solve.

Firstly, there is a class called TrafficPolice that does job like a traffic police. The TrafficPolice class has a queue that keeps track of every car that enter the traffic light area. As soon as a car touches the sensor of road with traffic lights, it is considered as that the car enters the traffic area. There are two queue structures that keep track of two types of directions, which are north-south and east-west. The declaration of the queues is like this:

private Queue<CarInfo> queueNS = new Queue<CarInfo>();

private Queue<CarInfo> queueWE = new Queue<CarInfo>();

CarInfo saves the current status of cars. The first queue is for cars coming from north or south direction, the second is for west or east direction. When cars touch the sensor, the cars will stop and the TrafficPolice class will add them to the queue accordingly. Then it will follow some rules to release the cars.

If there are two cars face-to-face and both go straight, both will be released immediately. If both want to turn right, they will also be released immediately. If there is one who wants to turn left, it must wait until the one come before to go first, then it can turn left. If there are two cars come from different directions. That usually means there are some cars who want to turn right when the light is red. It needs to wait until what is in the queue before successfully passes the road.

Basically, the idea is to make sure there are no car collisions. The system should not allow cars who may cause collision to pass. For example, if there are cats face-to-face, one wants to turn left and another one wants to go straight. They may cause car accident in the middle of the road. The system needs to prevent it from happening.

That is core code of the AI part of the project. Basically, it is policy-based approach of AI programming. Everything is hard-coded. The main idea of this method is to consider all dangerous situations that the program may encounter, then code the program with rules that can avoid those dangerous situations. However, the game in this project doesn’t simulate all situations. For example, if it is a rainy day in the city, the roads should be very hard to drive, but this program doesn’t simulate this situation. The program only simulates traffic lights and crowded streets with many cars. It ignores many other situations that are very important for real roads due to limited computation resources, such as rainy roads, snowy roads and pedestrians. Simulating a real road requires a high-end device, but I don’t have enough resources to do.

### 3.1.2 Physics-based Car Driver Class

Another core code of this project is about cars and roads. Driving cars in the game is not just vector additions; the cars are based on physics. The physics functionality of this game is provided by built-in physics engine made by Unity Technologies and NVIDIA. It is called PhysX [3]. This technology brings reality to game. Players can feel the cars in the game like real car on the streets. PhysX is also exclusively designed for NVIDIA GPU. GPU has excellent performance in matrix computation and physics requires extensive matrix calculation. Therefore, using this technology can dramatically increase the performance; it also speeds up the AI training process.

The implementation of this technology is also very simple. Developers don’t need to know the fundamental algorithms of physics; they just need to use the API developed by Unity3D engine team. Every car has four wheels. Four wheels are furtherly grouped into two groups: front and rear. Front wheels are controlling the directions while rear wheels are providing power to move forward. When the car wants to move forward, those two lines will be used:

WheelColliderRL.motorTorque = MotorTorque;

WheelColliderRR.motorTorque = MotorTorque;

It means add motor torque forces to rear wheels. Only rear wheels can be added with torque forces. RL means real left wheel and RR means rear right wheel. The amount of torque force is defined as MotorTorque. Different car has different motor torque. It also determines how fast the car can move.

Here is the code to brake a car:

WheelColliderRL.motorTorque = 0;

WheelColliderRR.motorTorque = 0;

WheelColliderRL.brakeTorque = Mathf.Infinity;

WheelColliderRR.brakeTorque = Mathf.Infinity;

Two rear wheels provide power to move; they also provide power to stop. Physically, stopping something still requires force applying upon the object. The first two lines mean cancel all forces of cars. The last two lines mean give infinite brake torque force to stop a car. It makes sure that cars can be immediately stopped to avoid any collision. The brake torque can be set to zero if the car is ready to release the brake. After the brake is released, car can move forward by adding forces to its rear wheels.

Here is the code to turn a car:

WheelColliderFL.steerAngle = Angle;

WheelColliderFR.steerAngle = Angle;

Only front wheels can be turned around. FL means front left wheel and FR means front right. Both wheels will be given a same angle to make the car turn. The angle is fixed for every car, which is 35 degree.

All cars are also equipped with ray collision function. It simulates cars’ vision, so cars can see what happens in front of them. Every car produces ray with a fixed length. When the ray collides something in front of them, the car will be braked automatically. It simulates obstacle detection functionality of an auto drive car.

Every car also has a method called CheckMovable. There are two situations that a car cannot move: something in front of it and very close to it and the street is red light. Cars can sense the surrounding environment to make decisions by calling this method. If the car is immovable, it will call brake method to immediately stop the car.

Those are some simple methods of car driver class. The class also makes the drive function be visible to AI program, so AI program can drive cars using the API. For example, AI program just need to call brake method when it feels something in front of it, which is given by ray collision function.

### 3.1.3 Procedural Generation Algorithm

Road generation algorithm is based on pseudorandom number generator. It is the formula generating random numbers for the generation method. However, the number is not totally random. Every random generator function is given with a value called seed. If two random methods are given with a same seed value, those two will generate same random number set. Usually, the seed value is given with current timestamp of computer system. It improves the randomness of the function.

The roads in this city are totally rectangular. There is a road point that marks which place should be built road, and it starts at a random point in the map, then, it decides which direction to go. randomly It has only four choices: north, south east and west, it can be done with this code:

direction = Random.Range(0, 2);

increment = Random.Range(-1, 1);

if (increment == 0)

{

increment = 1;

}

Random.Range method provides a random integer between left parameter and right parameter, but the right parameter is exclusive, which means the highest value this method returns is the right parameter minus one. Therefore, the first line will generate two values: 0 or 1. Each indicates which direction the road point should go: north-south or west-east. The second method gives the increment of the road point. It also generates two values: -1 and 0. However, if the value is 0, if should be changed to 1, or it is not able to change anything. Then, combining the value of direction and the value of increment will produce 4 possibilities: north, south east and west. The road point is moving one unit each time until it reaches the limit of partial length. It defines how much the road point can move in the direction decided by the mixed value of direction and increment. When the limit is reached, the road point should change a direction. However, it can change to the same direction as it previously was. The algorithm is terminated when the total length is reached. Then, the place that the road point went will be the point that road will be built.

After the program knows where to build roads, it starts building it. As mentioned above, there are two special roads with traffic lights: X road and T road. X road is determined by a point if it is marked by road point and all 4 points surrounding it are also marked by road point. T road is determined by a point if it is marked by road point and only 3 points surrounding it are also marked by road point. Other three roads follow the similar idea. L road is determined by a point is it is marked by road point and only 2 points that are not face-to-face are marked by road point. If two points that are face-to-face should give the city an I road. If there is only one, it gives the city an I road with an end.

After roads are generated, all core parts are finished. Other game objects, such as buildings, trees or parks, will not be mentioned in this article, because they are just decoration and have nothing with AI algorithm.

### 3.1.4 Conclusion

In conclusion, this AI algorithm is good for roads in cities. Cities’ roads are usually well organized, and they are not changed very often. Therefore, every city can have a center that directs all traffic in the city. Every part of road can have some sensors that tell how do cars drive. Besides traffic problems, hard-coded program is also very useful for other issues if we know all situations the system may encounter and solutions for the issues. Therefore, the program does not need to collect any personal data from people, it protects both our privacy and convenience.

However, it also has some disadvantages. Usually, we cannot consider all situations, because they are too huge for us to handle. The algorithm may not work if we cannot do that. This system is not for irregular road, such as roads in forest and mountains. A rainy day may cause dramatic change to those kinds of roads. It may destroy the algorithm that used to be effective.

## 3.2 Physics-based ML AI Car

Is there any solution for irregular roads? The answer is yes. In this project, the game simulates a physics-based car in a forest. The trail is not rectangular, and it can be any shape. Unfortunately, the map in this project must be static. It is not procedurally generated each time when the game is being initialized.

Why the map should be fixed? The algorithm used in this project is called reinforcement learning. It is a subset of machine learning algorithms. The tool for this algorithm is ML-Agents developed by Unity Technologies [4]. The algorithm contains an incredibly long loop and extensive matrix computation, both are core of machine learning algorithm. It is one of the worst disadvantages of the algorithm. Neural network is also used in this project, which occupies huge size of memory and requires extensive computation of CPU and GPU. If the map in this project is also randomly generated, it will bring enormous amount of dataset to the computer. It is the reason why the map must be fixed due to scarce resources.

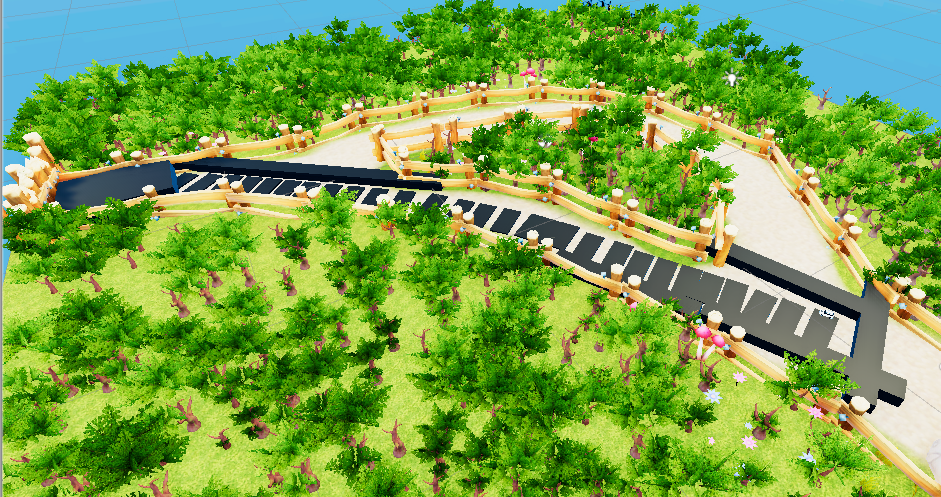
### 3.2.1 Basic Idea of Reinforcement Learning

The game map is exclusively designed for the algorithm and the basic idea of reinforcement learning is simple. Firstly, the AI agent tries some actions randomly. The agent may know small number of rules about the environment it currently in. For example, car driving agent may know how to move car forward, to turn and to brake. The agent can also see what happens in front of or behind it. However, it doesn’t know what the map looks like, what is the final goal or what happens if the agent collides a wall. It is the algorithm who tells the reward or the penalty for each action the agent makes. In this case, a scoring system will be introduced to finish this job.

The role of scoring system is very simple. When the agent makes a right decision, it will be rewarded, so it will try to do more right things. The agent will be panelized when the agent makes a bad decision, so it knows and avoid doing bad things. On the other hand, by getting score or losing score, parameters of agent will be changed. It is the way how agent learns the environment. It is more like a child human, parents give candy when the child does good things, or criticize the child when he/she does bad things. Then, the child knows what is good and what is bad.

### 3.2.2 The Map Structure

Firstly, it is important to know what the map looks like:

****

**Start Point**

**Destination**

As the picture shows, the map simulates a forest. The car starts on the right side of the map, and it ends on the left side of the map. There are two ways the car can choose. There are some black points in the map. Some of them are bad points, and others are checkpoints. All black points will be invisible in play mode. Fences are also siting on the side of the road, which prevent cars from going too far from the destination. If there are no limitations in the map, such as fences, the car will run to some random directions, which may not never able to achieve the goal. All other elements in this game, such as trees, ground, colors of ground and wind effects are totally for decoration and have nothing related to the algorithm.

### 3.2.3 Core Algorithm of the Agent

#### 3.2.3.1 Introduction of Reinforcement Learning and PID Model

Although this project is about machine learning, but I it is not necessary to write any fundamental machine learning code for this project. Because Unity has its own machine learning API called ML-Agents, which is exclusively designed for machine learning in Unity 3D engine, the only thing that developers need to do is scoring system. Scoring system is the key of reinforcement learning algorithm. It is the system that rewards or penalizes agents.

Although it seems too advanced that the project is based on machine learning, there is still some traditional method that controls the algorithm. There is a very famous control model called PID model in industry. PID stands for proportion, integral and derivative (or differential). The scoring system in this project borrows some ideas from the model. The main purpose of this model is for automatic process control [5]. The car in this project is also about automatic control, so this model is very useful. Here is a brief introduction of the model.

Reinforcement learning and PID model shares one mutual idea. For example, when chef is cooking a new cuisine, he/she may feel the fire is too big for the cuisine, then he/she turns the fire to a smaller level. Then, the fire may be too small, so the chef will turn it to a higher level, but it will not be as high as the first time. If the fire is still to high, he makes the fire be smaller, but will not be as small as the second time. He/she will repeat this process until he/she finds the right fire strength.

How does reinforcement learning implement in this situation? The chef is learning a new cuisine, and it is the training target. Unlike other machine learning algorithm, which usually tries to minimize the errors the algorithm gets. Reinforcement learning is trying to get more “errors” to complete the job. Notice that the word “error” in machine learning doesn’t necessarily mean “mistake”, it is one or a group of numbers that indicate the training result. In reinforcement learning, score is usually to described training result, not “error”, but the term is still mentioned in some situations. When the chef turns the fire too hot, program will reduce some scores as penalty. Then, the chef will turn the fire cooler, but it may be too small. The program will still reduce some of his points. The chef will turn fire bigger again, but he/she will not turn it as hot as the first time, because he/she knows the program reduces his points if he does that. The fire could still be too hot for the cuisine, however, because it is better than the first time, the program reduces less points than the first time. As the process repeats several times, the program will reduce less and less points until he gets the right fire. The program will reward him huge points for accomplishing this mission. However, his/her whole learning process is still not completed. Most people cannot master doing something from the first try, so the chef must do it again. At the second time of learning, he/she may do some mistakes again, but, ideally, he/she will do better than the first time. Each time the chef tries is called epoch in machine learning. A strong reinforcement program usually requires more than millions of epochs to master the task. It is like a person, who learns to play the piano, who needs to practice repeatedly to master the instrument. If the chef successfully does better in his/her second try than his/her first try, the points he/she lost will be smaller than the first time, so his/her total points will be higher. Eventually, the chef will get a very high score that indicates him/her successfully masters doing the task.

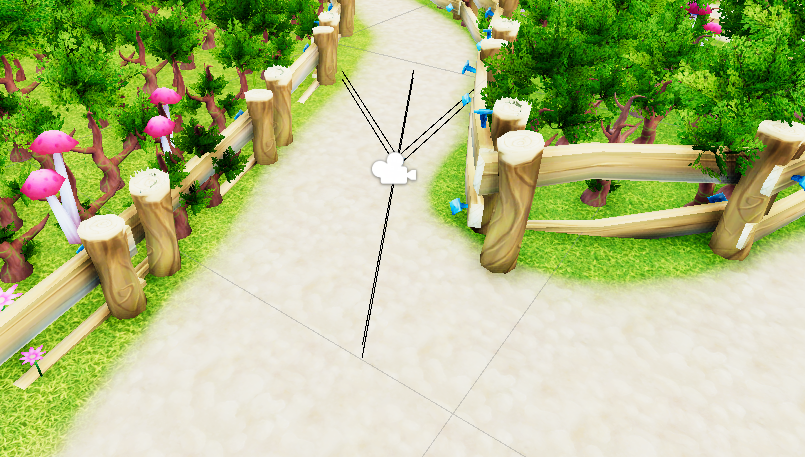
How does PID model apply for this situation? Three components of PID model are working together; it is not step-by-step. In proportional control, it means the proportional between error and points the chef loses. The more errors the chef makes, the more points the chef will lose. When the fire is too hot, the chef will get points off. If the temperature is close to the ideal temperature, the less points the chef will lose. In integral control, it means summing up. Every time the chef tries to do something, he/she automatically gets point off. Assume the chef needs to complete the job one time, therefore the more time he/she spends, the more points he/she will lose. Oppositely, if the chef finishes the task quickly, the less points he/she will lose. In this situation, the losing points will be summed up, it forces the chef to finish the task quickly. In derivative model, it means the change rate. This model monitors how much does the chef change the temperature. If the fire is too high, and it is turned to too low by the chef in a very short time, it means the change rate is too high and the lack of stability. The chef will also be punished if he/she does this, because cooking requires the fire to be stable. However, this model seems to add up the punishing points with proportion model, because the chef must cool down the temperature more when it is too hot, which brings a very big change rate. Indeed, according to the article [5], proportion and derivative are usually working together. The derivate model can protect when the temperature is right, then the chef should keep the stability if it is a suitable temperature. If chef still tries to turn the right temperature to high or low, it brings instability, then he/she should be punished by derivative model. On the other hand, it can also make sure when the temperature is just little higher or lower than the target temperature, the chef will not over change the temperature to too low or too high.

In conclusion, the algorithm of reinforcement learning focuses on how the program learns to play the game and how the neural network been built. The PID model provides how to reward or punish agent when it does good or bad things, so it focuses more on scoring system.

#### 3.2.3.2 Details of the Agent

Initially, the agent if very stupid at the start of the program. It can see the objects in front of or behind the agent. However, it doesn’t know what the game object exactly is when the game starts. Therefore, it may try to collide anything at the beginning. For example, the car can see there is a fence in front of the car, but it will still try to go to the fence due to it knows nothing about the environment. The system doesn’t directly tell the agent that it must avoid any fence. It is the scoring system’s responsibility to tell the agent where it should go indirectly.

The vision of car is achieved with ray detection algorithm. The car emits many rays forward or backward with a fixed length, which mean how long the car can see. Then, the collision system of Unity3D engine will check is there is one of the rays hitting something. The collision system will send back the tag of collided game object back to the car, so the car knows the environment it currently in. This picture shows an example of the vision of the car, which is the back lines. Notice that this is not like the picture previously showed, because there are some game objects hidden from players.



Each game object is tagged with a string, such as “fence”, “checkpoint” and “badpoint”, which tells the agent and the scoring system what the game object is. Also, when the car collides something, scoring system will check which object the agent hits by checking the tag, then it will do something accordingly. Although when player runs the game, there are some invisible game objects that people cannot see, the agent can still see them and do something accordingly. Making them invisible to players is just for beautifying the game, so players will not see wired black objects while they are playing game. It has nothing related to the core algorithm. The physics-based driving function is like the previous project, so it doesn’t need to be introduced again. On the other hand, the car also knows the distance between the car and the destination, the velocity and the angular velocity, the full declaration of detectable objects are:

var rayDistance = 5f;

float[] rayAngles = {112.5f, 90f, 67.5f, -90f};

var detectableObjects = new[] {"fence", "destination", "checkpoint", "badpoint"};

AddVectorObs(Ray.Perceive(rayDistance, rayAngles, detectableObjects, 0f, 0f));

AddVectorObs(Ray.Perceive(rayDistance, rayAngles, detectableObjects, .23f, 0f));

AddVectorObs(Vector3.Distance(transform.position,

TargetPoint.transform.position));

AddVectorObs(CarDriver.GetComponent<Rigidbody>().velocity);

AddVectorObs(CarDriver.GetComponent<Rigidbody>().angularVelocity);

Notice there are only four detectable objects in the game for the agent. Although there are some other objects, such as trees and the ground, but they will do no effect to the agent. Therefore, they will not be checked by the system to save some computation power. 4 detectable objects are grouped by 2 groups: penalty and reward. Fences and bad points are the objects that the car must not touch, so they are in penalty group. Checkpoints and the destination are the goal that the car should go, so they are in reward group. Both penalty and reward are applied according to PID model.

The car is learning the environment by being punished or rewarding. Each collision between the car and other game object gives feedback to the agent by sending the tag of the game object to the agent program. The program can do something accordingly. For example, if the car hits a fence, then the car gets a result of negative point. It teaches the car that it must not hit any object tagged with “fence”. It is the general idea of reinforcement learning that teaches the car where it should go and where it should avoid. The whole system repeats this process more than one hundred thousand epochs to make sure all parameters in the algorithm are optimized. How much punishment or reward the car should get is determined by the scoring system in this project, which is powered by PID model.

#### 3.2.3.3 PID Model in Punishment

In proportion model, the distance between the car and the destination determines how much the car will lose. The longer the distance means the bigger error it is, so the car should be punished with more points. This model tells that the car must move closer and closer to the destination, so it prevents the car from staying at somewhere and not moving. And integral model reduces the points of the agent consistently until the task is accomplished. The purpose is same as what mentioned above, which forces the car to complete the task on time. These two models can be merged together as this line of code:

AddReward(-(1f / agentParameters.maxStep)\*(Vector3.Distance(transform.position,

TargetPoint.transform.position) /

StartDistance));

The left side of the multiplication is the application of integral model. maxStep means the maximum times that the agent can try. However, the value of max steps can be as big as 500 thousand, which is too high for punishment, which causes instability to the algorithm, so the value should divide one to make it smaller. The official document of Unity3D ML-Agents recommends that the absolute value of punishment or reward should be not greater than one. The right side is about proportion model. Vector3.Distance returns the distance between the car and the destination. The value can also be too large for the algorithm to handle, so it is divided by StartDistance, which means the initial distance when the program starts or when the car touches a checkpoint. The result of the multiplication is executed each step the car does any action, therefore the multiplication itself is an integral model, because the losing points are summing up until the end.

The derivative model is handled by the program indirectly. This model makes sure that the agent can drive the car smoothly to avoid any obstacles, such as fences and bad points. The map is set up exactly for the car if it is too fast or the angular velocity is too step, it will collide some obstacles. Here is the code for fence collision, and bad point collision is similar:

private void OnCollisionEnter(Collision collision)

{

if(collision.collider.transform.parent?.tag == "fence")

{

AddReward(-CumulativeCheckPointReward);

CumulativeCheckPointReward = 0.05f;

if (chances-- <= 0)

{

Done();

}

CarDriver.GetComponent<Rigidbody>().velocity = Vector3.zero;

CarDriver.GetComponent<Rigidbody>().angularVelocity = Vector3.zero;

transform.position = LastCheckPointTransformPosition;

transform.rotation = LastCheckPointTransformRotation;

}

}

Firstly, the program checks if the car really hits a fence. Then it will take points off and reset the cumulative reward gained by touching checkpoints. It is the integral model of reward function, which will be introduced later. The agent has chance of 5. Each collision with an obstacle will decrease the chance by 1. If the chance is below 0, the whole program will be restarted, and the car needs to run the map again. The idea is also like integral model. The error of collision with obstacles will be summed up, if there are too many such errors, the whole program restarts. The last 4 lines is to reset the velocity and angular velocity of the car, which removes any all forces on the car so that it can stop the car and bring it back to latest checkpoint if the car still has chance.

#### 3.2.3.4 PID Model in Reward

The agent can get rewarded indirectly in proportion model. The program doesn’t have a direct code for adding points to the agent in proportion model. However, the agent gains from less and less punishment, which means if the distance is closer and closer, the loss is smaller and smaller. Therefore, the agent can gain the points indirectly from this situation. Similar idea applies to derivative model. The program doesn’t have direction code in derivative model either. The agent will not be punished if the velocity and the angular velocity are stable and smooth, the car will not touch any obstacle, then the car will not lose any point. The agent can also gain points from this situation indirectly.

The only direct code is for integral model. Here is the code for the agent touches any checkpoint:

AddReward(CumulativeCheckPointReward);

CumulativeCheckPointReward += 0.05f;

sender.SetActive(false);

StartDistance = Vector3.Distance(transform.position,

TargetPoint.transform.position);

LastCheckPointTransformPosition = sender.transform.position + Vector3.up \* 2;

LastCheckPointTransformRotation = sender.transform.rotation;

Each time when the car touches a checkpoint, the program rewards the car with a cumulative reward. The reward will be bigger and bigger as the car touches more and more checkpoints. The point starts with 0.05f in float format. The first checkpoint the car touches will give the car a point of 0.05f. The second will be 0.10f, etc. This method means the reward is summed up, which is the main idea of integral model. However, if the car collides any obstacle, the cumulative reward will be taken off and reset, so the car will be rewarded back to 0.05f if the car touches a checkpoint after the accident. It is a severe punishment to the agent if it does something wrong. The following line sets the start distance to the distance between the checkpoint and the destination. The last two lines record the latest checkpoint, so the car can be brought back if it does something wrong and still have chance. When the car successfully touches the destination, it will be rewarded with 2 times larger as the current cumulative points it has, and the program is reset for next epoch:

AddReward(CumulativeCheckPointReward \* 2);

Done();

#### 3.2.3.5 PID Conclusion

In conclusion, PID is proved effective in most industries as well as this project. It gives us a fundamental theory of building a scoring system. Scoring system guides the reinforcement algorithm how to control the penalty and the prize the agent can get for each action, so the algorithm can be more effective.

### 3.2.4 Conclusion

The biggest advantage of this method is that the AI program doesn’t need a large dataset from people’s personal data. The program can simulate every situation by itself, so it can generate a huge dataset by itself for the agent to learn. On the other hand, the algorithm works for irregular roads, it can be more general for real situation that most roads one this planet are irregular.

However, it doesn’t prove that collection of personal data is avoidable for all AI programs, there is still some data collected from our daily life to train the AI, but this project can at least prove that we can use simulation technology to reduce the collection of personal data. The program also requires very long time and extensive computation power to achieve. My small program spent around 14 hours on getting the most optimal result. The scale of computation cost grows exponentially as the scale of project grows. Simulation itself also needs to collect some data from daily life, because we must consider every situation we may encounter to simulate. It is also very hard for us to consider all cases in our life.

# Conclusion

In conclusion, we’re already living a life which used to only in sci-fi movie. We used to only focus the advantages of the technology in movies, but we ignore the costs that require to achieve the goal. Fortunately, we still have some methods to avoid the issues. On the other hand, the simulation technology developed by making computer games is also extremely useful in other application. Games can both be fun and useful. Hopefully, there will be an algorithm in the future can both protect people’s privacy and make more powerful software to improve our quality of life.

# Appendix

## I. Toolset

Unity3D [6] is a game engine developed by Unity Technologies. It is the game engine developed for games which can be played in multiplatform, such as Windows, iOS, Android and much more. The programming language of Unity3D is C#. The AI program is powered by ML-Agents, which is the same company who develops Unity3D engine. The program is exclusively designed for Unity3D engine. The AI model is trained with NVIDIA GPU [7]. And the art resources are from many authors from Unity Asset Store.

## II. Reference

[1] DeepMind, AlphaGo, <https://deepmind.com/research/alphago/>

[2] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017, <https://arxiv.org/abs/1712.01815>

[3] NVIDIA, PhysX, <https://www.geforce.com/hardware/technology/physx/technology>

[4] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

[5] Dataforth, Introduction to PID Control, Dataforth Application Note, <https://www.dataforth.com/introduction-to-pid-control.aspx>

[6] Unity Technologies, Unity Official Website, <https://unity.com/>

[7] Unity Technologies, Unity Documentation, https://docs.unity3d.com/Manual/index.html

[8] Google Inc., Tensorflow API Documentation, https://www.tensorflow.org/api\_docs/

[9] Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, The MIT Press, 2016

[10] Jorge Palacios, Unity 2018 Artificial Intelligence Cookbook Second Edition, Packt Publishing Ltd., 2018

[11] Sebastian Raschka, Vahid Mirjalili, Python Machine Learning Second Edition, Packt Publishing Ltd., 2017

[12] Oriol Vinyals, Timo Ewalds, Sergey Bartunov et al., StarCraft II: A New Challenge for Reinforcement Learning, https://deepmind.com/documents/110/sc2le.pdf

[13] Fletcher Dunn, Ian Parberry, 3D Math Primer for Graphics and Game Development, CRC Press, 2011

[14] Ian Millington, John Funge, Artificial Intelligence for Games Second Edition, Morgan Kaufmann, 2009

[10] Vincent-Pierre Berges and Leon Chen, Puppo, The Corgi: Cuteness Overload with the Unity ML-Agents Toolkit, https://blogs.unity3d.com/2018/10/02/puppo-the-corgi-cuteness-overload-with-the-unity-ml-agents-toolkit/?\_ga=2.268135113.654368819.1540749958-630411105.1536294885, October 2, 2018

## III. Art Resources

[1] LuGus Studios, European Cartoon City, <https://assetstore.unity.com/packages/3d/environments/urban/european-cartoon-city-12591>

[2] David y Gael Fun Games, SkyBox Fantastic, <https://assetstore.unity.com/packages/2d/textures-materials/skybox-fantastic-6089>

[3] Hedgehog Team, Skybox Volume 2 (Nebula), <https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-volume-2-nebula-3392>

[4] Synty Studios, Simple Town - Cartoon Assets, <https://assetstore.unity.com/packages/3d/environments/urban/simple-town-cartoon-assets-43500>

[5] Area730, Stylized Simple Cartoon City, <https://assetstore.unity.com/packages/3d/environments/urban/stylized-simple-cartoon-city-50095>

[6] MyxerMan, Simple Town Pack, <https://assetstore.unity.com/packages/3d/environments/urban/simple-town-pack-91947>

[7] Singularity Art Studio, POLY STYLE - City Pack, <https://assetstore.unity.com/packages/3d/environments/urban/poly-style-city-pack-98033>

[8] HK3dStudios, Simple Vehicle Pack, <https://assetstore.unity.com/packages/3d/vehicles/simple-vehicle-pack-107170>

[9] ZugZug Art, Hand Painted Forest – Zugrand, <https://assetstore.unity.com/packages/3d/environments/fantasy/hand-painted-forest-zugrand-42897>

## IV. GitHub

[1] Yuhang Liao, AI-based 3D Game Simulator, <https://github.com/ComarPers922/AI-based-3D-Game-Simulator>

(Notice: Only game code is in the repo, any other elements, such as 3D models or game scenes are not included in the repo. Two executables for the two games are in the repo.)