AI-based 3D Game Simulators

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, Assistant 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

1.1 Background

The world is full of smart devices now, and many are featured with Artificial Intelligence (AI), which makes devices more friendly and accessible to more and more people. It truly pushes the technology forward. AI is a technology that makes computers do things like or beyond humans. For instance, according to Docherty's research, AI can make medical service more affordable and improve patients' quality of life [1]. Doctors are human, they need to take a break. However, it makes harder for patients to get right treatment in time. On the other hand, doctors need to make living by curing people. Consequently, it makes medicine expensive. Fortunately, computers do not need to sleep. Therefore, patients can still get right treatment in time to cure their illness. If it is real, countless lives will be saved. Computers do not need as much money as doctors to "live". As the result, the prices will be affordable for most patients.

Essentially, 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, since AI program requires huge amount of data to be trained. It is also the most important that this paper will talk about.

Unfortunately, everything comes with price. AI companies are usually collecting data they want to use from their 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, which means the user unknowingly 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 on Google's huge database.

It is also a violation of law in some regions. For example, according to Art. 9, Chapter 2 in General Data Protection Regulation (GDPR), which is a law protecting personal information in European Union, it is prohibited for the collection of data from people in European Union, such as genetic data or biometric data, which are necessary for medical AI [2]. Although it gives an exemption chapter 2 of GDPR: If the purpose of the collection is for the improvement of the quality of personal life in European Union, it can collect such data. However, it does not give exemption for many other purposes. Therefore, developers still need to come up with some ideas that both improve the software and collect minimum personal data.

Fortunately, there is a possibility to provide users a good service without collecting huge amount of data. It is the topic that this paper will discuss. 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.

1.2 The Relationship between Games and AI

The idea about making games as the simulator seems to be unusual. 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 a real street. It is the idea that this project is trying to prove.

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 at least reduce the amount of data from users. As the result, researchers can get the same result with minimum data from people.

1.3 State-of-the-art Technologies in AI Games

The best example is AlphaGo [3]. AlphaGo is an AI program playing Go game made by DeepMind partnered with Google. Go game is considered one of the hardest board games in the world. However, the robot 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 the human players, it was also trying to beat itself. The biggest advantage that computers 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.

Since AlphaGo could generate data while it was playing games by itself, thus it did not require data from human. AlphaGo zero [4], which was the latest version of AlphaGo family proved that AI program can still generate data by itself without human's intervention. 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, thus 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.

The main idea of my project comes from AlphaGo Zero, which is an AI program without any data from human professionals. In the same vein, my projects are featured with AI without the collection of data from our real life.

2. Random City

The first project is about cars in a randomized city. The project is based on implementation of traditional data structure of two queues to achieve a simple and effective AI. The city is totally randomly generated, which means that the city is different from each time after it is initialized. It is a 3D game, and players can use a keyboard to move the camera freely in order to see all the details of the city. All 3D models in this project come from Unity Asset Store, and the links are provided in the reference page. Here is a picture showing a part of the city:

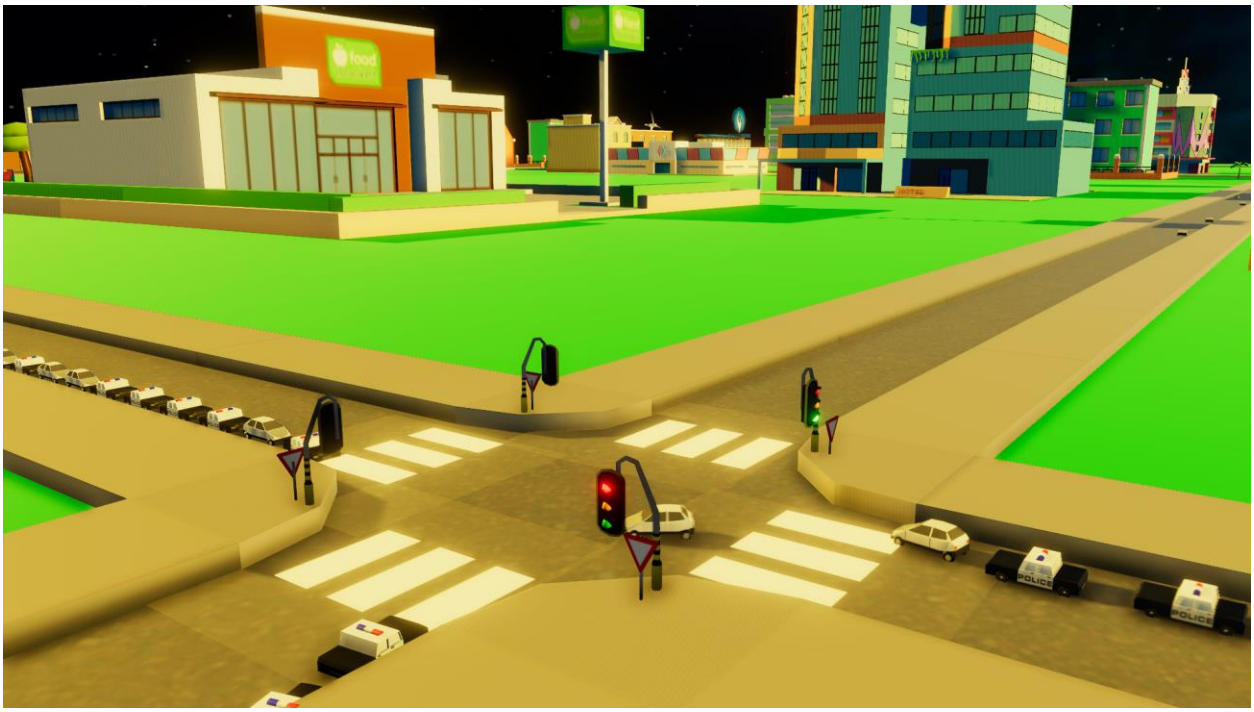

Figure 1. A busy X-road in the city.

As the figure shows, there are cars in this city, and they are the main component that this article will discuss. Cars are automatically driven by themselves following hard-coded policy-based rules. Every decision and behavior, which guide all cars to safely run on the road, are totally predefined in code. Therefore, this project doesn't does not require any data collection from people. Note that any pseudo code provided is just for demonstration.

2.1 Policy-based AI

The city is 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 the decoration of the game and do not play any important role in AI algorithms. For example, in Figure 2, there are some buildings, trees and Wind Turbines on the green ground, but none are standing on the roads. Therefore, they are not able to be touched by any car on the street. There are also some air balloons flying on the sky, and they will land on somewhere in this city. While they are flying, they can do nothing with cars. They do not land on the street either. Therefore, no cars have the change to touch them.

Figure 2. Decoration

Although roads are generated by machine, but all roads are regular, which means every turn is strictly 90 degree. There are five shapes of roads as shown in Figures 3 and 4. The algorithm is designed for the city, which is right-hand traffic. It is not effective for left-hand traffic. Each road has some green cubes shown on each road called sensors, which are invisible in play mode, thus players cannot see them in the game. However, the cars in the game can see those sensors. When a car collides a sensor, the sensor will tell what the next waypoint is. The waypoint is also a sensor. Then, after the car successfully touches the next waypoint, it will get next waypoint guiding the car move forward. The first three roads as shown in Figure 3 are very simple, because they do not have traffic lights. Car can just go forward to next waypoint. Although the second one is the place where car makes U-turn, sensors just need to tell car the speed and angle information to accomplish the task.

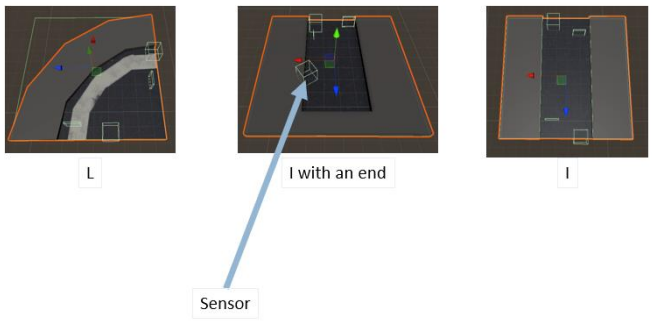

Figure 3 . Roads without traffic lights.

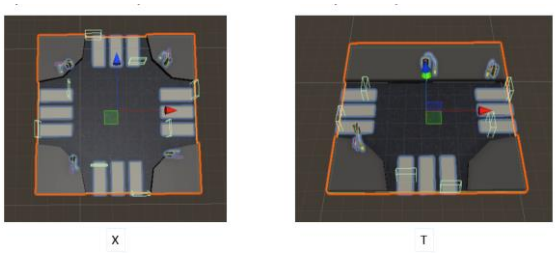

Figure 4. Roads with traffic lights.

The last two roads as shown in Figure 4 include traffic lights. Thus, cars may need to wait for the lights before they go. It is the hardest part of this project that AI should solve. It is

the policy-based algorithm that is supposed to prevent any car collision while making the traffic smooth. We have a few techniques to obtain such a goal.

Firstly, there is a class called TrafficPolice that does a job like a traffic police. The TrafficPolice class has a queue that keeps track of every car that enters the traffic light area. As soon as a car touches a sensor of traffic light road, it is considered as the car entered the traffic light area. There are two queue structures that keep track of cars from two 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 and 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 by the class 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 passing the road.

The purpose of this policy 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 cars 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 developers code the program with rules that can avoid those dangerous situations. However, the game in this project cannot 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 does not 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. Unfortunately, I do not have enough resources to do.

2.2 Physics-based Car Driver Class

Another core code of this project is about how to implement physics-based car. 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 API of this technology is also very simple. Developers do not need to know the fundamental algorithms of physics; they just need to use the components developed by Unity3D engine team. The API is also GUI-based, which makes the development easier. This picture shows an example of the API:

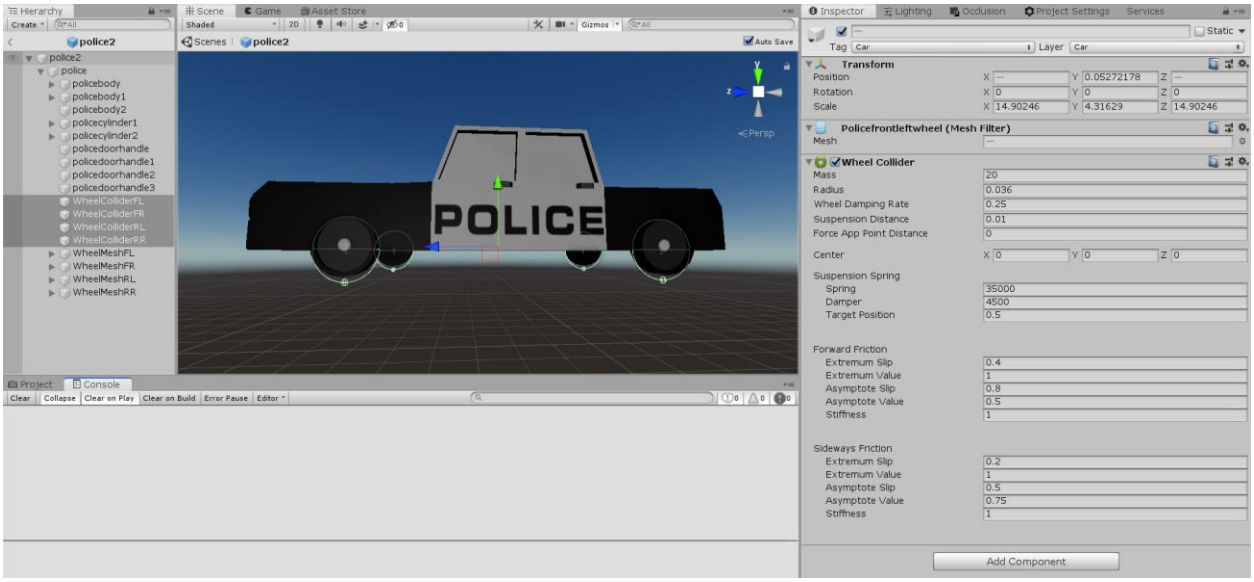

Figure 5. Example of GUI-based Physics API

The right side of Figure 5, which is called Inspector Window in Unity3D, contains all properties the car has. However, they are not important for this project. We do not need to change anything, because default values are enough for the game. Every car has four wheels. Four wheels are further grouped into two groups: front and rear. Front wheels are controlling the directions while rear wheels are providing car 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 denotes rear-left wheel and RR denotes rear-right wheel. The amount of torque force is defined as MotorTorque. Various cars have 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 upon cars. The last two lines mean giving infinite amount of 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. Both wheels will be given a same angle to make the car turn. The angle is fixed as 35 degree for every car.

All cars are also equipped with ray collision function. It simulates cars' vision. Therefore, 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-driven car.

Every car object also has a method called CheckMovable. There are two situations that a car cannot move: something that is very close and in front of it, or 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.

The methods mentioned above are the core of car driver class. The class also makes the drive function be visible to AI program, which enables the AI program to 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.

2.3 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. The seed value is usually given with current timestamp of the computer system, which 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 a 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;
}

 partialLengthCount = MinPartialLength;
```

This code is also the whole content of the method ResetParameters, which will be discussed later. Random.Range method provides a random integer between left parameter and right parameter, but the right parameter is exclusive, which means that the highest value that 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.

This code shows the whole idea of this algorithm:

```
       for (int i = 0; i < RoadLength; i++)
        {
            while (RoadMap[roadPoint[0], roadPoint[1]] == 1)
            {
                if (partialLengthCount-- <= 0)
                {
                    ResetParameters();
                }
                else
                {
                    roadPoint[direction] += increment;
                }
                if (roadPoint[0] < 0 || roadPoint[0] >= mapWidth || roadPoint[1] < 0 ||
roadPoint[1] >= mapHeight)
                {
                    roadPoint[0] = Mathf.Max(roadPoint[0], 0);
                    roadPoint[0] = Mathf.Min(roadPoint[0], mapWidth - 1);
                    roadPoint[1] = Mathf.Max(roadPoint[1], 0);
                    roadPoint[1] = Mathf.Min(roadPoint[1], mapHeight - 1);
                }
            }
            RoadMap[roadPoint[0], roadPoint[1]] = 1;
        }
```

The city has a maximum road length, which is given by a variable called RoadLength. The algorithm stops at the maximum road length been reached. Variable roadPoint is an array containing two members, which are X and Y. It represents current position the road point is. The road point is moving one unit each time until it reaches the limit of partial length, which is partialLengthCount in the code above. It defines how much the road point can move to the direction decided by the mixed value of direction and increment. When the limit is reached, the road point should change the direction by calling the method ResetParameters. However, it can change to the same direction as it previously was. The next few lines mark where the road should be built. RoadMap is a 2D array. The size of the matrix is exactly the size of the city. The values of the matrix initially are all zeros. Zero stands for the place where is not available for building a road. It is the map that will be marked to indicate where the roads are in the city. Then, there is an if-statement to check if the road point is outside the map. If it is, following four lines will adjust the offset to make sure the road point is always inside the map. The last line is the code

marking the place where road should be built, which is an assignment of a value one to the road map at the position given by the variable roadPoint.

There are two special roads with traffic lights: X-road and T-road. X-road is determined by a point marked by road point and all 4 points surrounding it are also marked by road point, which is the method CheckX. T-road is determined by a point marked by road point and only 3 points surrounding it are also marked by road point, which is the method CheckT. Other three roads follow the similar idea. L-road is determined by a point marked by road point and only 2 points that are not face-to-face are marked by road point, which is the method CheckL. If two points that are face-to-face, then the program should give the city an I-road, which is the method CheckI. If there is only one marked road point, it gives the city an I-road with an end, which is the method CheckIWithAnEnd. The whole code:

```
for (int i = 0; i < mapWidth; i++)
        {
            for (int j = 0; j < mapHeight; j++)
            {
                if (RoadMap[i, j] != 1)
                {
                    continue;
                }
                if (CheckX(RoadMap[i, j]))
                {
                     BuildX(RoadMap[i, j]);
                }
                else if (CheckI(RoadMap[i, j]))
                {
                     BuildI(RoadMap[i, j]);
                }
                else if (CheckL(RoadMap[i, j]))
                {
                     BuildL(RoadMap[i, j]);
                }
                else if (CheckT(RoadMap[i, j]))
                {
                     BuildT(RoadMap[i, j]);
                }
                else if (CheckIWithAnEnd(RoadMap[i, j]))
                {
                     BuildIWithAnEnd(RoadMap[i, j]);
                }
               }
```

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, which will not be mentioned in this article.

2.4 Discussion

In conclusion, the traditional algorithm mentioned above is good for roads in cities. City roads are usually well-organized, and they are not often changed. Therefore, every city can have a data center like the TrafficPolice class previously mentioned that directs all traffic in the city. Every part of road can have some sensors that tell how to 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. For example, if there are some pedestrians standing in front of car, we can just add another policy in the method CheckMovable, which was mentioned before, to stop the car. Therefore, the program does not need to collect any personal data from people, and it is still effective for the big city in this project. All cars in the city do not require people to drive them. Therefore, it can both protect our privacy and make our life more convenient.

However, it also has some disadvantages. We must add more code for countless situations we may encounter. For example, this system is not for irregular road, such as roads in forest and mountains, because we assume it is a city. However, if it is a city in mountains, the algorithm cannot work. We must add more code to deal this case. A rainy day may cause dramatic change to those kinds of roads. Water makes the roads smoother, then the cars may have trouble with driving. For example, the force implied to the cars in rainy day should be less than the cars in sunny day. Smoother roads make car be able to move forward with less force. If

the force is the same as the force in sunny day, the car may be too fast to avoid collisions. Therefore, we must add more code to deal with such a situation.

3. Physics-based Car with Reinforcement Learning

In contrast to the aforementioned car simulation game that drives on the regular road, in this game, it simulates a physics-based car that can drive an irregular road after a machine-learning training phase. The trail is not rectangular, and it can be any shape. In order to focus on the learning algorithm, the map in this project is kept static. It is not procedurally generated each time as the previous Random City when the game is being initialized. This project is also made with Unity3D engine, and it is also a 3D game. The 3D models are also from different authors in Unity Asset Stores. The roads of this project are not strictly 90 degree, which makes cars harder to control the angular velocity to make a good turn. Player camera is always following the car; they cannot move the camera freely in this game. It is called third-person-vision camera. Here is a picture that shows what can player see in this game:

Figure 6. Third-person-vision camera.

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, which results in one of the worst disadvantages of this algorithm, because it requires long time to train. The time for training the model can be more than several months. Neural network is also used in this project, which often occupies a 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, which is another reason that we keep the map static.

3.1 Basic Idea of Reinforcement Learning

The basic idea of reinforcement learning is simple. Firstly, the AI agent tries some actions randomly. The agent may know a small number of rules about the environment it is 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 does not know what the map looks like, what is the final goal, or what happens if the agent collides a wall. It is the algorithm that performs a reward or a 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 intuitive. When the agent makes a right decision, it will be rewarded. As the result, it will continue to do such right decision. For example, if the car successful touches any checkpoint in the game, it is considered as right decision. The agent will be panelized when the agent makes a bad decision, resulting in that it learns to know and avoid doing bad things. On the other hand, by getting score or losing score, the parameters of agent will be changed. It is the way how the agent learns the environment. It is more like a human

child; parents give a 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”.

3.3 The Map Structure

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

Figure 7. The whole map.

As Figure 7 shows, the map simulates a trail in 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, but it is forced to choose the most optimal one, which is the bottom trail in this map. There are some black points in the map. Some of them are bad points, and others are checkpoints. The black cubes that are at the same line with fences are bad points. They prevent the car from going wrong way. The black cubes that form a ladder-like shape are checkpoints. They guide the car to the destination, which is the biggest black cube sitting on the left side of this map. All black points will be set invisible in play mode, except the destination cube. Therefore, players do not see them when they are playing the game. 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 never be 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.

3.4 Core Algorithm of the Agent

3.4.1 Introduction of Reinforcement Learning and PID Model

Although this project is about machine learning, 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 so 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 [7]. The car in this project is also about automatic control. Therefore, this model is also effective.

Reinforcement learning and PID model share one mutual idea: The agent should be panelized for bad decision or rewarded for right decision. According to the article [7], I want to simplify it with a simple story.

When chef is cooking a new cuisine, he/she may feel the fire is too hot for the cuisine, then he/she turns the fire down. However, the temperature may be too low. Then, 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 too hot, he makes the fire be colder, but will not be as small as the second time. He/she will repeat this process until he/she finds the right fire strength.

It is an example of reinforcement learning. The chef is learning a new cuisine, which is the training target. Unlike other machine learning algorithms, 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 does not necessarily mean “mistake”, it is one or a group of numbers that indicate the training result. In reinforcement learning, score is usually to describe training result, not “error”, but the term is still mentioned in some other 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/her huge points for accomplishing this mission. However, his/her whole learning process is still not completed. Most people cannot master doing something just from the first try. Therefore, 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, which makes his/her total points higher. Eventually, the chef will get a very high score that indicates him/her successfully the task is accomplished.

PID model also applies to this situation. Three components of PID model are working together; it is not step-by-step. In proportion control, it means the proportion 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 in 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 [7] from Dataforth, 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 chef still tries to turn the right temperature to be higher or lower, 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 turn the temperature to too low or too high, or he/she will be penalized due to huge difference.

In conclusion, the algorithm of reinforcement learning focuses more 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, which focuses more on scoring system.

3.4.2 Details of the Agent

Initially, the agent is very naive at the start of the program. It can see the objects in front of or behind the agent. However, it does not 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 a fence in front of the car, but it will still try to go to the fence due to that it knows nothing about the environment. The system does not directly tell the agent that it must avoid any fence. It is the scoring system's responsibility to indirectly tell the agent where it should go.

The vision of car is achieved with ray detection algorithm. The car emits many rays forward or backward with a fixed length, which means how long the car can see. Then, the collision system of Unity3D engine will check if there is one of the rays hitting something. The collision system will send the tag of collided game object back to the car, so that the car knows the environment it is currently in. Figure 8 shows an example of the vision of the car, which is the black lines. Notice that this is not like the figure previously showed, because there are some game objects hidden from players.

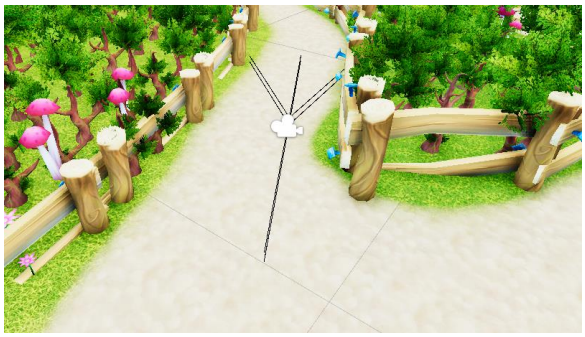

Figure 8. Vision of car.

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 they cannot see, the agent can still see them. Making them invisible to players is just for beautifying the game, so that 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, which does not 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 declaration of detectable objects is:

```
        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 that 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 in order to save some computation power. Four detectable objects are grouped by two groups: penalty and reward. Fences and bad points are the objects that the car must not touch, which are in penalty group. Checkpoints and the destination are the goal that the car should go, which are in reward group.

The car is learning the environment by being punished or rewarded. 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. Then, the program can do something accordingly. For

example, if the car hits a fence, then the car gets the tag “fence” and negative points. 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 well tuned. How much punishment or reward the car should get is determined by the scoring system in this project, which is provided by PID model.

3.4.3 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 that it prevents the car from staying at somewhere and not moving. Meanwhile, the 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 five hundred thousand, which is too high for punishment, which causes instability to the algorithm. Therefore, the value should divide one to make it smaller. The official document of Unity3D ML-Agents [4] recommends that the absolute value of punishment or reward should be not greater than one. The right side is from 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 that 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 being summed 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. The cumulative reward 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 equal or 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 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.

3.4.4 PID Model in Reward

The agent can get rewarded indirectly in proportion model. The program does not 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 does not have direction code in derivative model either. The agent will not be punished, if the velocity and the angular velocity are stable and smooth. Therefore, 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 that the car touches will give the car a point of 0.05f. The second will be 0.10f, etc. This method means that 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 that 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 that the car can

be brought back if it does something wrong but still have a 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();
```

In conclusion, PID has been proved effective in most industry control systems 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 that the agent is supposed to get for each action. It makes the algorithm more effective.

3.5 Discussion

The biggest advantage of this method is that the AI program does not need a large dataset from people's personal data. The program can simulate every situation by itself. As the result, 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 like the second project.

However, it does not prove that collection of personal data is avoidable for other AI programs. There is still some data that must be 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. For example, 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. It is also very hard for us to consider all cases in our life. Simulation itself also needs to collect some data from daily life, since we must consider every situation we may encounter to simulate.

4. Conclusion

We are already living a life which used to be only in the sci-fi movie. We get used to only focusing on the advantages of the technology in movies but ignore the costs that require to achieve the goal. Fortunately, we still have some methods, such as the article mentioned above, to avoid large collection from people. We can make AI with minimum data. On the other hand, the simulation technology developed by making computer games is also extremely useful in other application, such as visualization of traffic roads. Therefore, games can both be fun and useful.

The first project successfully proved that traditional data structures are still useful for AI program, such as queue data structure, which is one of the most fundamental data structures of computer science. Therefore, the algorithm is easy to implement. No data is collected for making the AI program. As the result, people do not need to share their huge information with companies that develop the program. However, it is hard to consider every situation and write code for it. As mentioned above, rainy day may change the road condition, which we must update the algorithm accordingly to adapt the change of the environment. We still need to collect some data from real life in order to make the simulator. For example, we must know how rainy day changes the road condition in order to simulate the situation.

The second project employs reinforcement learning algorithm, which currently is one of the most advanced algorithms. It is also an algorithm that successfully proves big data is not necessary for training AI. As previously mentioned, AlphaGo Zero is a robot that learns everything from scratch, and it does not need any data from human professionals. My project, which is the physics-based AI Car, also proves that simulation technology can help us with auto-driven car. PID model, which is one of the most used in modern industry, can also be useful for

the AI system. The cores of reinforcement learning are neural network and scoring system. Since the neural network API is already provided by Unity3D team, the only thing developers must consider is the scoring system, which successfully teaches AI how to drive a car on an irregular road. The advantage is that the program can generate huge data by playing the game. We do not collect any data from people for this project. On the other hand, driving on the irregular road successfully proves that there exists such algorithm that can both collect as least data as possible and be effective. However, the algorithm also has some disadvantages. I spent more than fourteen hours on training the little program. The real situation is much huger than this project. The time required to train grows exponentially as the scale grows. This project does not prove that no data is required for every situation. For example, as previously mentioned in the introduction of this article. Medical AI program may still need to collect some data, such as genetic or biometric. There are countless racing games for car simulation nowadays. However, there are extremely few medical games that teach people how to do surgery. Medical simulation is still difficult to implement. As the result, collecting such data is still necessary for medical application.

In the future, I plan to make a better algorithm that can both further reduce the collection of personal data and improve our quality of life. I also plan to make simulators with more advanced simulation technology that AI program can be trained in such virtual world with minimum collection of personal data in real world. Both are treading technology that changes the world.

Reference

[1] Docherty, A. (2014). Big data–ethical perspectives. *Anaesthesia*, *69*(4), 390-391

[2] European Parliament, 2016, *General Data Protection Regulation. Art. 9, Chapter 2*, https://gdpr-info.eu/art-9-gdpr/

[3] DeepMind, AlphaGo, https://deepmind.com/research/alphago/

[4] 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

[5] NVIDIA, PhysX, https://www.geforce.com/hardware/technology/physx/technology

[6] 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.

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

[8] Unity Technologies, Unity Official Website, https://unity.com/

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

[10] Google Inc., Tensorflow API Documentation, https://www.tensorflow.org/api_docs/

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

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

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

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

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

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

[17] Vincent-Pierre Berges and Leon Chen, Puppo, The Corgi: Cuteness Overload with the Unity ML-Agents Toolkit, October 2, 2018, 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/

Appendix

I. GitHub

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. The copyright of the code files is based on the MIT License. The detail is in LICENSE.txt file of the repo. However, art resources are not covered by the MIT license. The copyright policies of art resources are different from each original author, which may be found in their personal pages in Unity Asset Store.

II. Art Resources

- LuGus Studios, European Cartoon City, https://assetstore.unity.com/packages/3d/environments/urban/european-cartoon-city-12591
- David y Gael Fun Games, SkyBox Fantastic, https://assetstore.unity.com/packages/2d/textures-materials/skybox-fantastic-6089
- Hedgehog Team, Skybox Volume 2 (Nebula), https://assetstore.unity.com/packages/2d/textures-materials/sky/skybox-volume-2-nebula-3392
- Synty Studios, Simple Town - Cartoon Assets, https://assetstore.unity.com/packages/3d/environments/urban/simple-town-cartoon-assets-43500
- Area730, Stylized Simple Cartoon City, https://assetstore.unity.com/packages/3d/environments/urban/stylized-simple-cartoon-city-50095
- MyxerMan, Simple Town Pack, https://assetstore.unity.com/packages/3d/environments/urban/simple-town-pack-91947
- Singularity Art Studio, POLY STYLE - City Pack, https://assetstore.unity.com/packages/3d/environments/urban/poly-style-city-pack-98033
- HK3dStudios, Simple Vehicle Pack, https://assetstore.unity.com/packages/3d/vehicles/simple-vehicle-pack-107170
- ZugZug Art, Hand Painted Forest – Zugrand, https://assetstore.unity.com/packages/3d/environments/fantasy/hand-painted-forest-zugrand-42897

III. 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.