Simulation of auto obstacle avoidance based on Unity machine learning

Qi Jiangyuan
School of Computer Science, Wuhan Donghu University
Corresponding author’s e-mail: qijiangyuan@wdu.edu.cn
Tel:15527894872

Abstract. This article mainly introduces the use of the TensorFlow platform, through the Unity3d engine, a 3d rendering tool, to simulate a car in a 3d world using TensorFlow training and learning, which can avoid obstacles in front and stay in the lane by itself, thus realizing a private Unity3d for machine learning Applications.

1. Introduction
In recent years, electronic games have accounted for an increasing proportion of people’s daily entertainment activities, and the application scenarios of Unity3d as a game development engine have become more and more extensive. From mobile games to medical simulation, it can reflect powerful performance and high scalability of the Unity3d engine.

With the continuous development of computer technology, the concept of machine learning was born in artificial intelligence. Machine learning can enable computers to acquire new knowledge and skills by simulating or realizing human learning behaviors, and to reorganize the existing knowledge structure to continuously improve their performance. Among them, TensorFlow is an artificial intelligence learning system developed by Google, and it is also one of the most popular machine learning frameworks.

This article uses the open source Unity machine learning plug-in ML-Agents to build a 3d world in Unity, where the car learns to avoid obstacles in front and keep driving in the lane through machine learning. Through the design and implementation of such a system, we can render the results of machine learning through a 3D engine. To train autopilot in a virtual environment is of vital importance to the study of artificial intelligence behavior in the game and to the simulation of the real world.

2. Construction of development environment
The system involved in this article needs to use the Unity3d engine and the Anaconda tool environment. First, you need to install the ml-agents development environment by using Anaconda. After the installation is complete, you need to activate this environment, and then the core part of TensorFlow is installed. Finally, install the Unity3d engine and install the ML-Agents plug-in package for the engine.

After all the above-mentioned environments are installed, the development environment required for this article has been built.
3. System design and implementation

3.1. Analysis of key technologies

ML-Agents is a Unity plugin, which is developed based on TensorFlow. By using this plug-in, we can use agents to train game AIs in game environments and simulation environments. These trained AIs can control their own behavior to achieve the design purpose of training supervisors.

For the research content of this article, we need to use ML-Agents to train the cars in the scene, each car is an independent agent, through the collection of environmental data in the game scene, the control of car behavior, and the analysis of the control results and calculations to get the final training data.

Specific to the Unity engine, the learning environment contains three additional components in the Unity scene:

Agent-It can be attached to a Unity game object and is responsible for generating its observations, executing the actions it receives, and distributing rewards in time. Each Agent is only associated with one Brain.

Brain-It encapsulates the decision logic of the Agent. In essence, the policy of each agent is stored in the Brain, which determines the actions the agent should take in each situation. More specifically, it is a component that receives observations and rewards from the Agent and returns actions.

Academy- It directs the observation and decision-making process of the agent. In Academy, you can specify several environmental parameters, such as rendering quality and environmental operating speed parameters.

4. Machine learning steps

4.1. Step 1

Collect Observations method collects various environments, positions, and speeds of the game, and hand them to ML-Agents.

| check   | values |
|---------|--------|
| Position| x,y,z  |
| Rotation| x,y,z  |
| Obstacle| Yes/No |
| In Lane | Yes/No |

4.2. Step 2

ML-Agents will calculate an operation array based on the submitted environmental information, and
determine the size of the operation array according to the behavior we want to operate the car. For example, we want to operate the x, y, and z axes Move, then this operation array will have three elements, representing the movement information in these three directions.

4.3. Step3
The operation array will be submitted to the AgentAction() method to implement one-step operation of the game. In this case, this method is mainly responsible for moving the car, judging whether there are obstacles ahead, judging whether the car is driving on the road, etc.

4.4. Step4
If the previous step judged that the car ran out of the road or the car hit an obstacle, then call the AgentReset() method to reset the scene environment. If there is no such penalty, return to the first step and continue.

| Reward                  | values         |
|-------------------------|----------------|
| Collide Obstacle        | 0 (Stop training) |
| Out of Lane             | 0 (Stop training) |
| Speed >50               | +0.1           |
| Avoid Obstacle          | +0.2           |

The Heuristic method encapsulates the manually input operation commands into an array format that AgentAction() can handle, and calls the InitializeAgent() method to initialize the Agent operation.

5. Preparation for machine learning

5.1. Unity3d scene construction
The whole scene is a virtual city environment. There are several roads. There will be different obstacles and some cars on the roads. After we run this game, we hope that the cars can drive on the road through machine learning and be able to take the initiative Avoid these obstacles.

The following parts are needed in the scene:
1. City and street model.
2. Car model
3. Obstacle model

![Figure 2. Scene in Unity.](image)

5.2. Obstacle detection
In real auto-driving cars, there may be obstacle detection methods such as ultrasound or laser. In this simulation game environment, we use the ray API provided by the Unity engine to determine obstacles. We can send a ray from the front of the car. If this ray hits an obstacle with a Collider in front, it will return the basic information of the collision point, such as the coordinates and the label of the colliding
object. After getting this information, we can easily figure out the obstacles ahead or the distance between the obstacle and the car.

5.3. Agent configurations
First, set each car in the scene as an Agent to participate in learning, that is, create a CarAgent script and attach it to the car model in the scene. This CarAgent class needs to inherit the Agent class provided by MLAgents and rewrite several methods:

1. AgentReset() method. Add the code to reset the initial state of the agent in this method. These initial states include the position, rotation, speed, acceleration and other attributes of the car.

2. CollectObservations() method, in this method add the observation of the speed attribute of the car and the information of the collision point after the ray collision. During the observation process, if a ray collides with an obstacle and the distance between the obstacle and the car is less than a certain value, we believe that the car will not be able to avoid colliding with the obstacle, and we believe that the training ended in failure and stop this Train again and call the AgentReset() method to reset the state of the agent.

3. AgentAction() method, in this method to get the operation of each step of the agent, and then apply these operations to the car. There are two data in the operation array, the first bit is the operation for acceleration, and the second bit is the operation for the direction. Finally, we need a reward feedback. By obtaining the current speed of the car, we can know whether the car is running normally. Multiply this speed by a coefficient as the reward for the current agent operation. Of course, the greater the speed, the greater the reward value. Otherwise, the smaller.

So far, the preparations for machine learning have been completed.

6. Machine learning process

6.1. Start training
1. Enter the Anaconda Prompt command line and run the activate ml-agents command to activate the ml-agents environment.
2. Run the mlagents-learn command to start training.
3. Go back to Unity and click the play button to perform training.

6.2. Improve training speed
Unity provides us with a virtual 3D environment. In this environment, we can easily copy the scenes and game objects created in the previous content, so we copy multiple cars in the scene, and each car can complete the training set before process. Therefore, the training speed can be improved in this way.
7. Results and discussion

During the training process, we can see that in the early stages of training, the car will directly hit obstacles if it has not yet mastered how to drive on the road. With the accumulation of training time, the number of training gradually increases, and the score that the car can get is getting higher and higher, which also means that the car can gradually learn to avoid obstacles and keep driving on the road.
The trained "brain" is an independent binary file. This file can be imported into the agent in the unity scene as a trained model, so that we can apply the training data to the car.

The above is the main content of this article. Through unity, we have constructed a simple virtual environment. In terms of car vision, it is only through ray collision to carry out preliminary simulation. Obviously, this project has a lot to improve, such as adding image recognition, and using computer vision to identify obstacles and roads in front of the car. Factors such as light and weather can be added to the virtual city, and elements such as basic traffic rules and traffic lights can also be added to the road. Through more realistic environment simulation, we can train smarter and more in line with our requirements.

8. Conclusion
By using the Unity3d engine to realize the realization of a machine learning system for cars to avoid obstacles autonomously, we can clearly see the advantages of 3D simulation technology in machine learning applications. The project mentioned in this article is just a simple reality simulation, and does not have as many attributes and complexity of the real world, but this training process and application method have been proven to be feasible. The virtual simulation world provides us with an effective way, giving us unlimited possibilities of trial and error, and the cost of all these advantages is not high, as long as we can incorporate as many factors as possible into our simulation world, we can train sufficiently good artificial intelligence.

References
[1] Abrahams, S., Hafner, D., Erwitt, E., Scarpinelli, A. (2016) TensorFlow For Machine Intelligence. Bleeding Edge Press, Santa Rosa.
[2] Hull, I. (2021) Machine Learning for Economics and Finance in TensorFlow 2. Apress, Berkeley, CA.
[3] Lai, J., Chen, X., Zhang, X. (2019) Training an Agent for Third-person Shooter Game Using Unity ML-Agents. In: International Conference on Artificial Intelligence and Computing Science.Hangzhou,pp.317-332.
[4] Juliani, A., Berges, V.P., Vckay, E. (2018) Unity: A General Platform for Intelligent Agents. arXiv e-prints, arXiv:1809.02627.
[5] Lample, G., Chaplot, D.S., (2016) Playing FPS games with deep reinforcement learning. arXiv e-prints, arXiv:1609.05521.