Automatic vehicle obstacle avoidance using the Unity Machine Learning Toolkit

Rui Wang
School of Computer Science
University of Nottingham
Nottingham, UK
psxrw10@nottingham.ac.uk

1. Introduction

In the field of artificial intelligence, driverlessness has always been a popular and cutting-edge research. Driverless cars are considered to be one of the technologies that will bring about a huge revolution in human daily life in the future. With the gradual maturation of pilotless technology, driverless cars seem to be very close to us. However, people seem to have limited confidence in its safety [1]. For driverless cars, the ability to accurately avoid obstacles is a crucial concern, as it is a matter of safety for the driver and passengers on board.

Unity, one of the world's game engine giants, has released an open-source plugin machine learning agents toolkit (ml-agents) in recent years. This plugin allows developers to train intelligent agents using machine learning algorithms, and the resulting agents are highly satisfying.

This report uses this open-source plugin to implement automatic obstacle avoidance for driverless cars in unity. The results from training in a random environment are relatively acceptable.

2. Related Works

2.1 Unity ml-agents

The rapid development of deep reinforcement learning in recent years couldn't have happened without a variety of powerful and scalable simulation platforms. Many research platforms are now developed based on the Atari 2600, Quake III, and Minecraft, and this will be a long-term tendency for artificial intelligence research platforms [2].

It is well known that Unity and unreal are two of the most powerful engines for video games, and it is obvious that there are many benefits to using Unity for artificial intelligence research. For example, the Unity engine is able to render high fidelity images, can simulate physical phenomena using Nvidia's PhysX, can supply abundant C# scripts and so on [2].

The Unity Machine Learning Agents Toolkit, an open-source plugin, enables researchers to use reinforcement learning more easily, imitation learning, Q-learning and other techniques to train

agents in unity [3]. In the learning environment created by ML-Agents, there are three key components, the perceptron, the agents, and the academy. Within it, sensors are used to collect data, agents react to the data collected by the sensors, and academy coordinates the entire simulation as a whole [2].

2.2 Reinforcement Learning

Reinforcement learning consists of an agent, environment, state, action and reward. After an action is performed by agent, the environment will change to a new state, for which the environment will give a reward signal (positive or negative reward). The intelligence then performs a new action according to a certain strategy based on the new state and the reward given by the environment [4].

Through reinforcement learning, an intelligent agent can know what actions it should take to maximize its reward in what state. Since the way that an intelligent agent interacts with its environment is similar to the way that humans interact with their environment, reinforcement learning can be considered as a general learning framework for solving general artificial intelligence problems. This is why reinforcement learning is also known as a general artificial intelligence approach for machine learning.

2.3 Proximal Policy Optimization (PPO) Algorithms

John et al. (2017) proposed a new family of policy gradient methods for reinforcement learning [5]. The Policy Gradient algorithm is very sensitive to step size, but it is difficult to choose a suitable step size. PPO proposes a new objective function that allows for small batches of updates in multiple training steps, solving the problem of difficult to determine step sizes in the Policy Gradient algorithm. The algorithms used in ML-Agents are PPO and SAC, and the researcher can choose their own according to their needs.

2.4 Curiosity Model

Rewards are the bridge that makes the reinforcement learning algorithm work, and agents learn behaviour through rewards after each round of training. However, these rewards are external to the agents, which requires us to set up the reward strategy every time we design an agent. This is feasible, but not scalable. So, we need a generic strategy that will drive agents to get rewards. Curiosity is a function that uses prediction error as a reward signal [6]. The usage of curiosity to drive agents to learn can significantly increase the effectiveness of agents' learning.

The Curiosity module is provided in the Unity Machine Learning Toolkit. This paper will use this module to accelerate the learning speed of agents.

2.5 Unity-based automatic obstacle avoidance

Jiangyuan, Q (2021) trained a car agent for automatic obstacle avoidance based on Unity's ML-

Agents module [7]. He built a large runway scene in Unity as a training environment, with models of car agents, obstacles, and city buildings besides the runway. The car agent perceives obstacles using a laser sensor provided by Unity. When the rays emitted by the sensor hit an obstacle, it gives information to the agent. The car agent ran in the Unity scene and was trained using TensorFlow through the API provided by ml-agents. And in the end, a good outcome was achieved.

3. Driver Agent Details

3.1 Architecture

3.1.1 Agent

Agents are trained using the PPO algorithm. As the agent speeds up, it gets a bonus. If it hits an object, it will lose points. Also, if it hits an object or the edges cause the speed to be too slow, the environment would be reset.

3.1.2 Sensors

The agent designed in this paper uses two sensors to detect obstacles and other agents. One is a long-range sensor, and the other is a close-range sensor used to detect objects in front of and near the vehicle respectively.

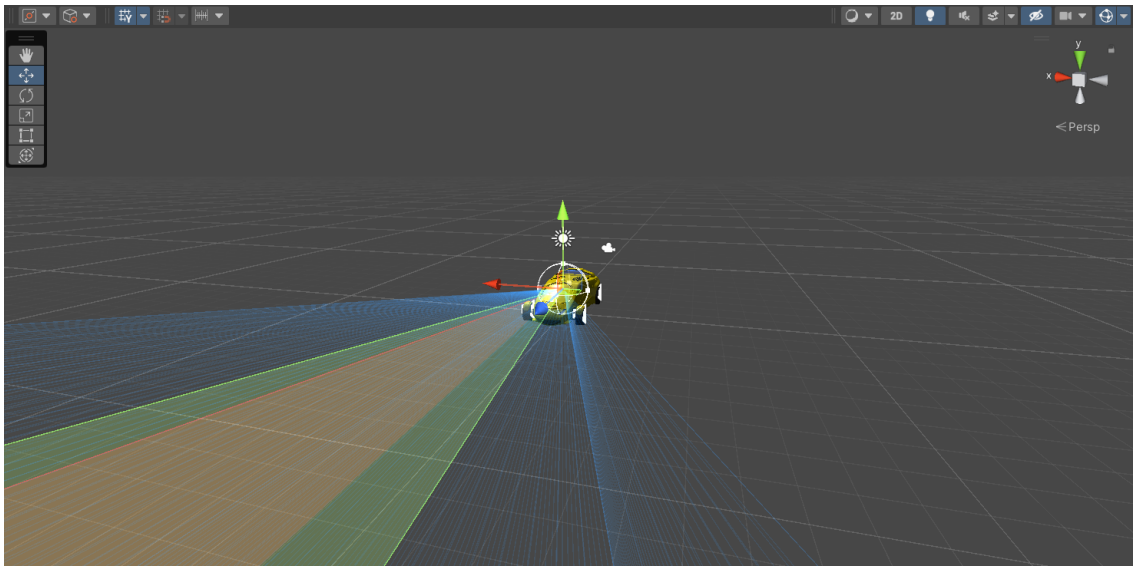

Figure 1. The Long-range Sensor

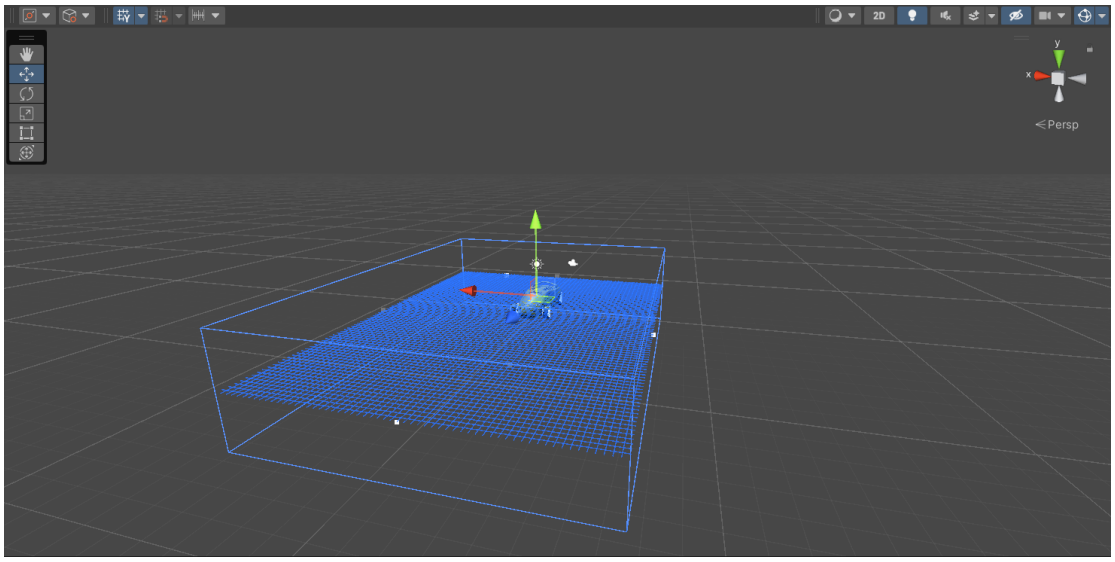

Figure 2. The Short-range Sensor

3.2 Unity Scenes

3.2.1 Road

Due to my computer's relatively low hardware specification, rendering a large runway would take up a lot of memory. So the solution I have adopted is to render only a constant length of track, and for each certain length of track that the car passes, it will continue to render a section of track ahead of it, whilst deleting the track that it has passed.

3.2.2 Obstacles

Three types of obstacles will appear in the scene: barrel columns, cone columns and roadblocks. They will be generated at the same time as the road is generated. Roads and obstacles are all randomly generated.

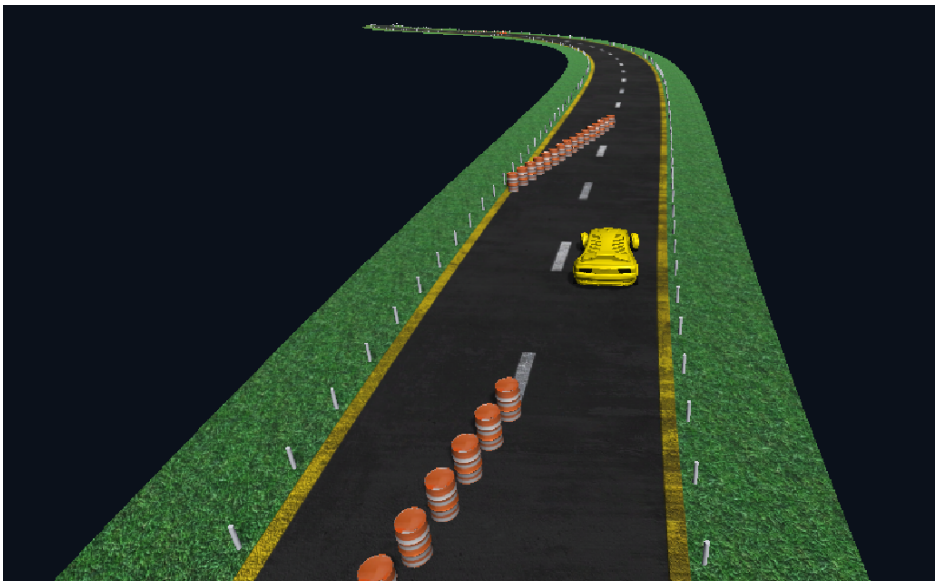

Figure 3. Obstacles

4. Experiments

This section will describe the detailed steps of training agents and the results obtained.

The experiments were undertaken with the following computer hardware.

- CPU: i5 10500
- GPU: Intel UHD Graphics 630
- RAM: 8GB
- Operating system: Windows 10 on a MacBook

4.1 Setting the environment

To complete the experiment, Anaconda needs to be installed first. Then create the environment for ml-agents in Anaconda Prompt and activate the environment. At last install the necessary tools such as PyTorch, Torchvision, Torchaudio and CudaToolkits.

The versions of the main tools involved in the experiment are as follows:

- Anaconda：Anaconda3 2021.11
- Python：3.6
- ML-Agents：release19
- PyTorch：1.1 stable
- CudaToolkits：11.3
- Unity：2021.3.1f1c1

Enter the command “mlagents-learn --help” in Anaconda Prompt and if no errors are reported, the environment is successfully setup.

4.2 Unity Scenes Construction

Open Unity, add the pre-defined roads, obstacles and vehicle agents to the scene and mount the corresponding scripts.

4.3 Training

Before training the training agents, we also have to configure the .yaml file that ml-agents needs to use. In the .yaml file we have to configure the environment settings, engine settings and other parameters. For instance, in the configuration file used in this paper, the training type is set to PPO; the learning rate is set to 0.003 and decreases linearly; two layers of convolutional neural network are used; only one hidden layer is used, each containing 512 neural network units; set reward rules, curiosity module, etc.

After configuring the .yaml file, use the command “mlagents-learn <trainer-config-file> --env=<env_name> --run-id=<run-identifier>” in Anaconda Prompt to connect to Unity. Then switch to Unity and click on the start button to start the training.

Figure 4. Setup ML-Agents Successfully

I trained the agent many times before using the curiosity module provided by ml-agent, but the agent learned extremely slowly. After more than a million steps, the agent had only just learned to make simple forward and awkward turns. After learning from the documentation provided by ml-agent, I turned on the curiosity module. With the same million steps of training, the average reward obtained by agent was significantly higher than without the curiosity module. The improvement was a staggering 80 times around.

4.4 Testing

Once the Agents have been successfully trained, ml-agents will automatically generate a neural network model file with the .onnx suffix. Mount it to the agents in Unity to see the results of the training.

This paper uses the rate of successfully passing obstacles as a metric. Counting the failures of the trained agent passing the obstacle 1000 times, the success rate was finally calculated.

4.5 Results and Analysis

By analyzing the training process, we can find that at the beginning of the training, the cars have a high collision rate and drive very slowly. As time goes on, the agent is capable of getting an increasing number of points and driving at a faster speed. This indicates that the agent has gradually learned to drive and avoid obstacles.

In order to achieve the final neural network file, a total of 2 million steps were trained in this paper. After testing, the trained model was successful on average 81.3 times per hundred passes over the obstacles, i.e. an average success rate of 81.3%.

```
[INFO] Driver. Step: 1850000. Time Elapsed: 19947.510 s. Mean Reward: 651.949. Std of Reward: 552.649. Training.
[INFO] Driver. Step: 1860000. Time Elapsed: 19983.995 s. Mean Reward: 745.123. Std of Reward: 464.203. Training.
[INFO] Driver. Step: 1870000. Time Elapsed: 20061.685 s. Mean Reward: 713.486. Std of Reward: 549.270. Training.
[INFO] Driver. Step: 1880000. Time Elapsed: 20140.263 s. Mean Reward: 809.282. Std of Reward: 536.050. Training.
[INFO] Driver. Step: 1890000. Time Elapsed: 20216.942 s. Mean Reward: 759.272. Std of Reward: 492.595. Training.
[INFO] Driver. Step: 1900000. Time Elapsed: 20294.124 s. Mean Reward: 640.350. Std of Reward: 586.454. Training.
[INFO] Driver. Step: 1910000. Time Elapsed: 20373.429 s. Mean Reward: 528.331. Std of Reward: 618.976. Training.
[INFO] Driver. Step: 1920000. Time Elapsed: 20451.723 s. Mean Reward: 993.040. Std of Reward: 511.632. Training.
[INFO] Driver. Step: 1930000. Time Elapsed: 20790.685 s. Mean Reward: 719.044. Std of Reward: 467.492. Training.
[INFO] Driver. Step: 1940000. Time Elapsed: 20865.434 s. Mean Reward: 463.403. Std of Reward: 415.913. Training.
[INFO] Driver. Step: 1950000. Time Elapsed: 20944.881 s. Mean Reward: 579.882. Std of Reward: 523.166. Training.
[INFO] Driver. Step: 1960000. Time Elapsed: 21023.657 s. Mean Reward: 607.727. Std of Reward: 632.892. Training.
[INFO] Driver. Step: 1970000. Time Elapsed: 21101.449 s. Mean Reward: 674.575. Std of Reward: 546.220. Training.
[INFO] Driver. Step: 1980000. Time Elapsed: 21176.003 s. Mean Reward: 1390.978. Std of Reward: 521.528. Training.
[INFO] Driver. Step: 1990000. Time Elapsed: 21254.566 s. Mean Reward: 584.014. Std of Reward: 399.176. Training.
[INFO] Driver. Step: 2000000. Time Elapsed: 21699.771 s. Mean Reward: 668.528. Std of Reward: 537.945. Training.
```

Figure 5. Two million steps for Training

Meanwhile, notice that during the training, every time the reward curve rises to a higher level it is preceded by a larger dip downwards. I think this can be explained in two ways: one is that in the reinforcement learning model agents need to keep making mistakes in order to learn a new behaviour, and the other is probably due to the fact that the experiment was set up with fewer reward points and more penalty points.

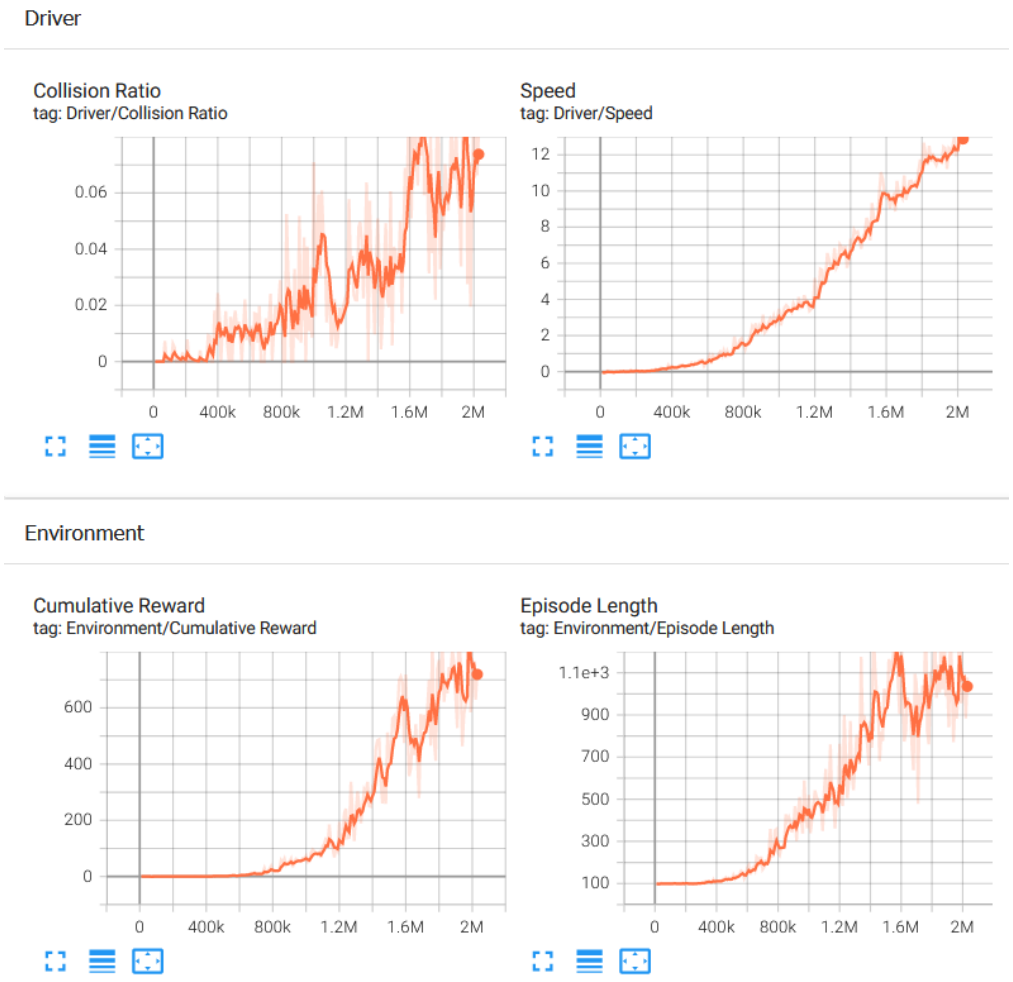

Figure 6. Training Result Part1

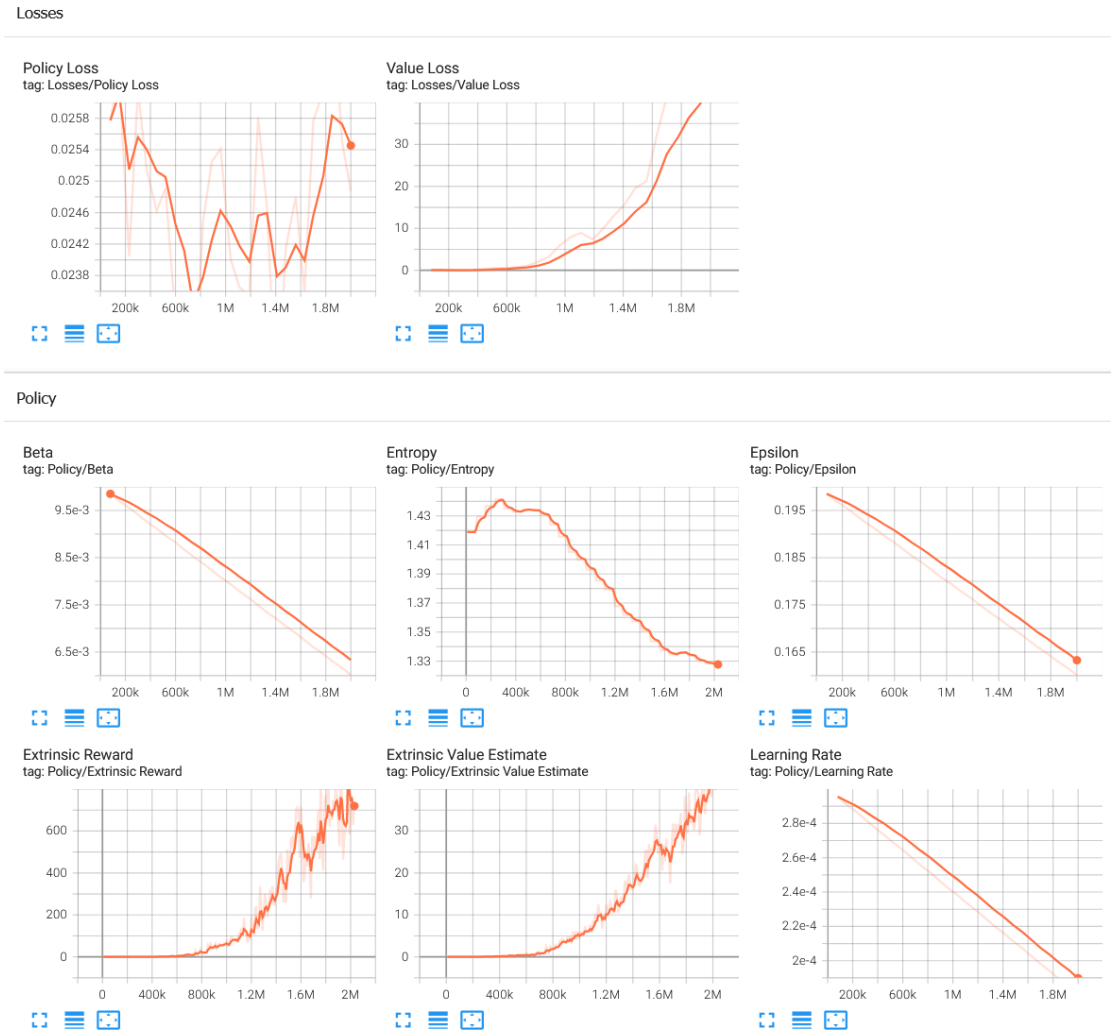

Figure 7. Training Result Part2

5. Conclusion and Discussion

In this paper, a self-driving car is trained to automatically avoid obstacles using ml-agents and PyTorch, based on the Unity engine. The average success rate of avoiding obstacles is 90%, which is still acceptable. The Unity engine is very efficient, but I still felt a lot of difficulty during the experiments in this article. The most important point is that although the Unity engine is capable of rendering high fidelity images, this also consumes a lot of the computer's memory resources. It is quite unfriendly to computers using an integrated graphics card. So if you want to use the Unity engine for deep learning research, it is recommended to use a NVIDIA discrete graphics card with high hash rate and large graphics memory. For this experiment, using the Unity engine for the study provides a more intuitive view of the training process and training results, but perhaps using a 2D engine such as Atari would also be a good option.

It was intended to continue exploring how multiple trained agents would interact when placed in the same environment, for example whether they would avoid each other. One guess is that they should be able to avoid each other, and even if they can't then they will learn very quickly. Unfortunately, with the computer I'm currently using, I can't give an answer in the time allotted. In the future I will probably explore this based on other simple two-dimensional environments.

Reference

[1] Kaur, K. and Rampersad, G. (2018) Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars. **Journal of Engineering and Technology Management** 48: pp.87-96.

[2] Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., and Lange, D. (2018) Unity: A general platform for intelligent agents. **arXiv** preprint arXiv:1809.02627.

[3] Github mlagents

[4] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996) Reinforcement learning: A survey. **Journal of artificial intelligence research** 4: pp.237-285.

[5] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017) Proximal policy optimization algorithms. **arXiv** preprint arXiv:1707.06347.

[6] Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018) Large-scale study of curiosity-driven learning. **arXiv** preprint arXiv:1808.04355.

[7] Jiangyuan, Q. (2021) Simulation of auto obstacle avoidance based on Unity machine learning. **Journal of Physics: Conference Series** 1883(1) pp.012048.