National University of Singapore

ME5406 Deep Learning for Robotics Project for Part I: The Froze Lake Problem and Variations

Submitted by **Teacher:** Prof. Peter C. Y. Chen

Li YiMing A0285133H

Email address: e1192690@u.nus.edu

A report submitted.

for the assignment of ME5406

in the

Department of Mechanical Engineering

National University of Singapore

Mar 4, 2024

Content

1. Introduction of algorithm

1.1 Monte Carlo

The Monte Carlo (MC) method is a calculation algorithm based on random sampling. It is particularly suitable for simulating and analyzing complex systems or processes. The core idea of the MC method is to use the statistical properties of random samples to approximate the mathematical expectation or other statistics of the problem. The Monte Carlo method is simple and flexible and is especially suitable for situations where it is difficult to obtain an accurate solution to the problem through analytical methods. However, the accuracy of the results depends on the number of samples, and many random samples are needed to obtain higher-precision estimates.

1.1.1 First-visit Monte Carlo control

First access Monte Carlo control is used to find the best strategy for a given task. The "first visit" aspect refers to how the algorithm estimates state-action values: it averages the rewards after the first visit to a state-action pair in the episode. Its characteristic is that it does not require environment models (i.e. transition probabilities and rewards). It learns directly from experience. It is widely used in reinforcement learning problems where the environment is too complex or unknown to be solved analytically. However, its reliance on episodic completion may make it inefficient for long episodic tasks or sequential tasks without clear termination states.

1.2 SARSA

SARSA (State-Action-Reward-State-Action) is used to learn a policy that specifies the best action to take in each state. It is an online strategy algorithm, which means it evaluates and improves the strategies used for decision-making. SARSA is particularly suitable for problems where the agent's behavior affects not only immediate rewards, but also subsequent states and therefore future rewards. SARSA can be applied to any task that can be structured as a Markov decision process. Due to its online policy approach, SARSA considers policy exploration strategies, which can lead to safer learning policies because it considers the potential downside risks of exploratory actions. Under certain conditions, its online strategic nature enables it to learn in uncertain or complex environments.

1.3 Q-learning

Q-learning is used to learn the value of each action in each state (i.e. Q-value) in order to find the optimal strategy without knowing the dynamics of the environment. It is an offline policy learning algorithm. This approach is particularly suitable when little is known about the environmental model. It directly learns the value of each state-action pair by updating the Q-table, which is simple and easy to implement. But a major challenge of Q-learning is that it may require a lot of time to explore all state-action pairs to learn the optimal policy, especially when the state space is very large.

1.4 ϵ-greedy behavior policy

ε-greedy strategy (used to find a balance between exploration and exploitation. This strategy aims to solve a key problem in the learning process: the agent should use the information it already knows to maximize immediate rewards (exploit) or explore unknown actions to obtain more information (exploration). In the ε-greedy strategy, ε is a parameter between 0 and 1, indicating the probability of choosing a random action, while 1-ε is It is the probability of acting according to the current optimal strategy. The implementation of ε-greedy strategy is simple and direct, and it is easy to apply in various reinforcement learning algorithms. But in the later stage of learning, when the agent already knows which action is the best action, ε-greedy strategy The policy will still perform random exploration with probability ε, which may result in suboptimal performance. And there is no general method to determine the optimal value of ε, and it usually needs to be adjusted through experimentation.

2. Experimental results

2.1 4*4 part

Since the pictures have the same reference, to avoid the problem of too large files and too many pictures, only the result collection is placed here. Note here that each part of the picture is in order: the image of episode via accuracy, the image of episode via cost, the optimal path finally found, the image of episode via average rewards, the image of episode via success rate, the image of episode via step, Image of the episode via step after the optimal path, comparison of the total number of results and the total number of failures. Below are the parameters used to obtain this result. (Independent images will be submitted as png in the folder)

2.1.1 First-visit Monte Carlo control

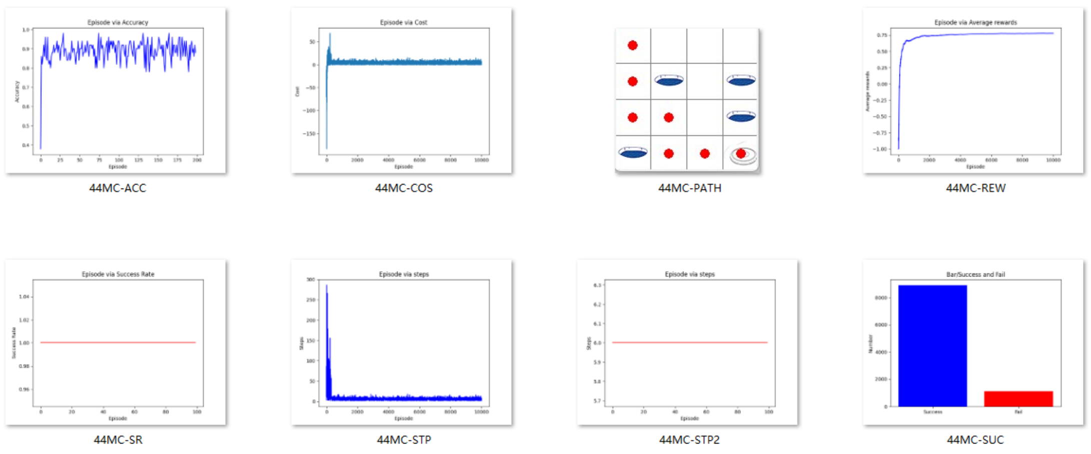


Figure 1 The sum of the result pictures of MC(4)

*NUM_STEPS=1000, NUM_EPISODES =10000, GAMMA=0.99, EPSILON=0.9

2.1.2 SARSA

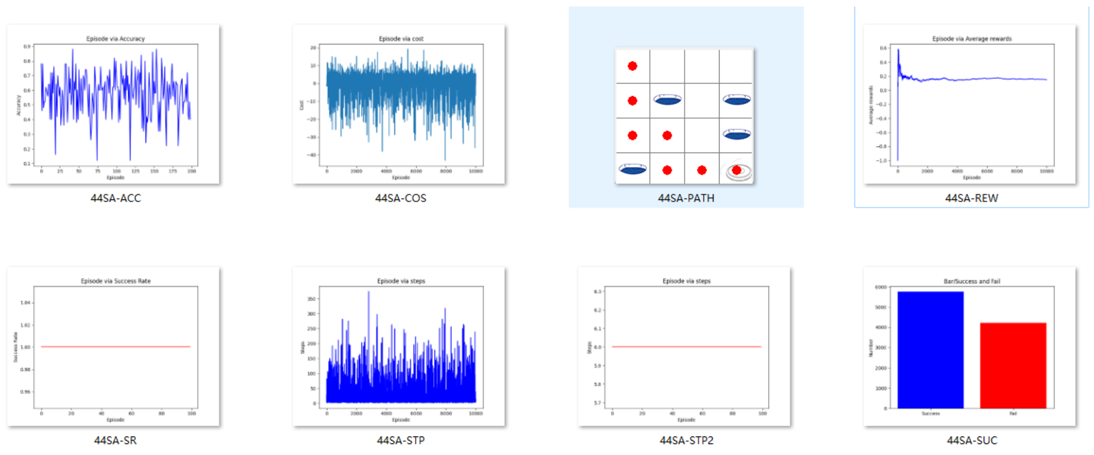


Figure 2 The sum of the result pictures of SARSA(4)

*NUM_STEPS=1000, NUM_EPISODES=10000, LEARNING_RATA=0.8,GAMMA=0.99, EPSILON=0.85

2.1.3 Q-learning

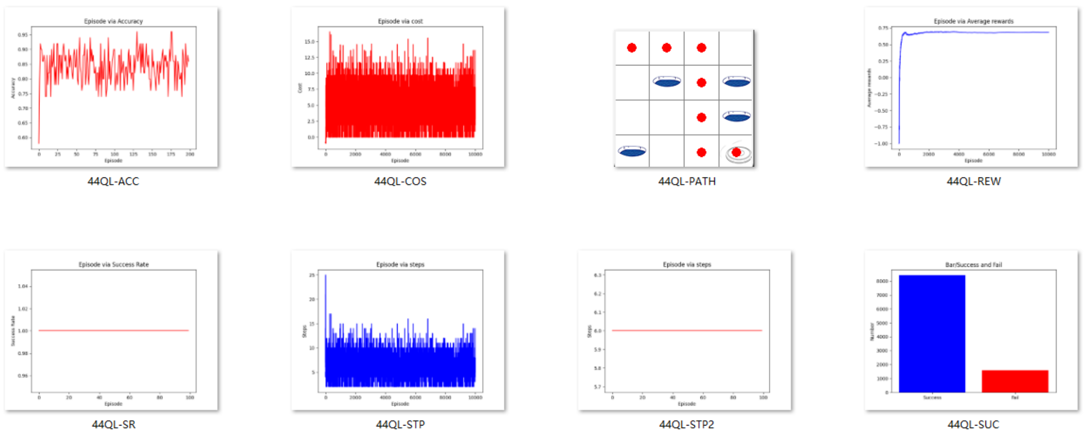


Figure 3 The sum of the result pictures of Q-learning(4)

*NUM_STEPS=1000, NUM_EPISODES=10000, LEARNING_RATA=0.9, GAMMA=0.7, EPSILON=0.75

2.2 10*10 part

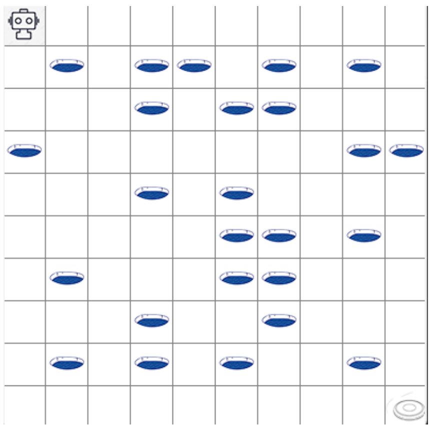


Figure 4 Simulated 10*10 map

2.2.1 First-visit Monte Carlo control

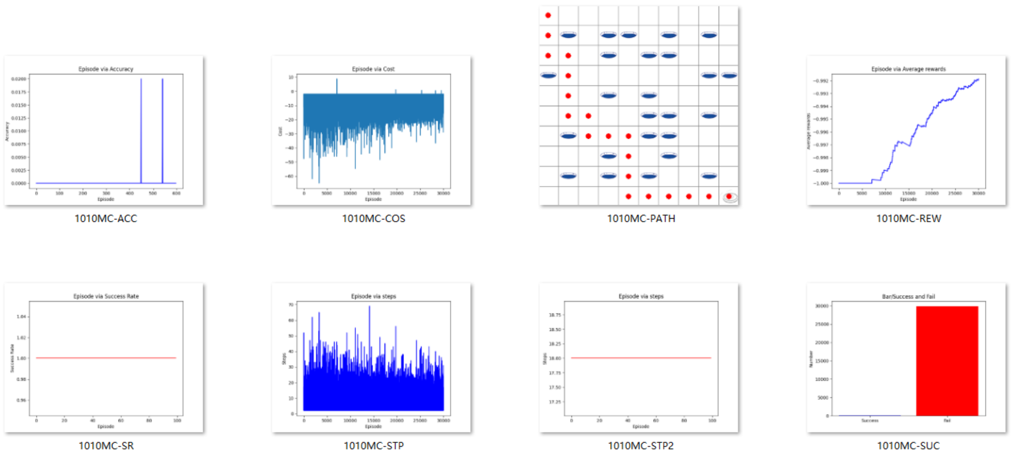


Figure 5 The sum of the result pictures of MC(10)

*NUM_STEPS=100, NUM_EPISODES =30000, GAMMA=0.9999, EPSILON=0.31

2.2.2 SARSA

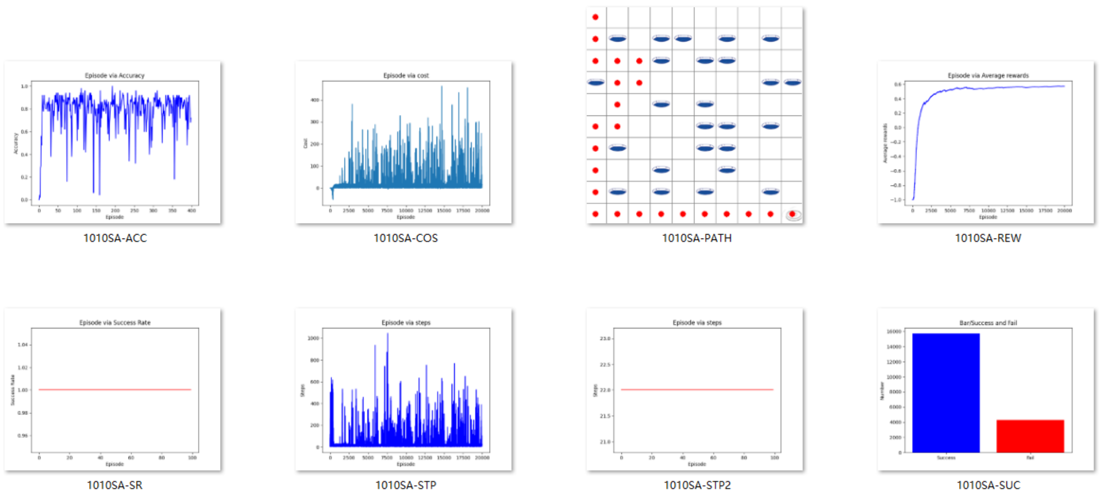


Figure 6 The sum of the result pictures of SARSA (10)

*NUM_STEPS=1000, NUM_EPISODES=20000, LEARNING_RATA=0.1,GAMMA=0.99, EPSILON=0.95

2.2.3 Q-learning

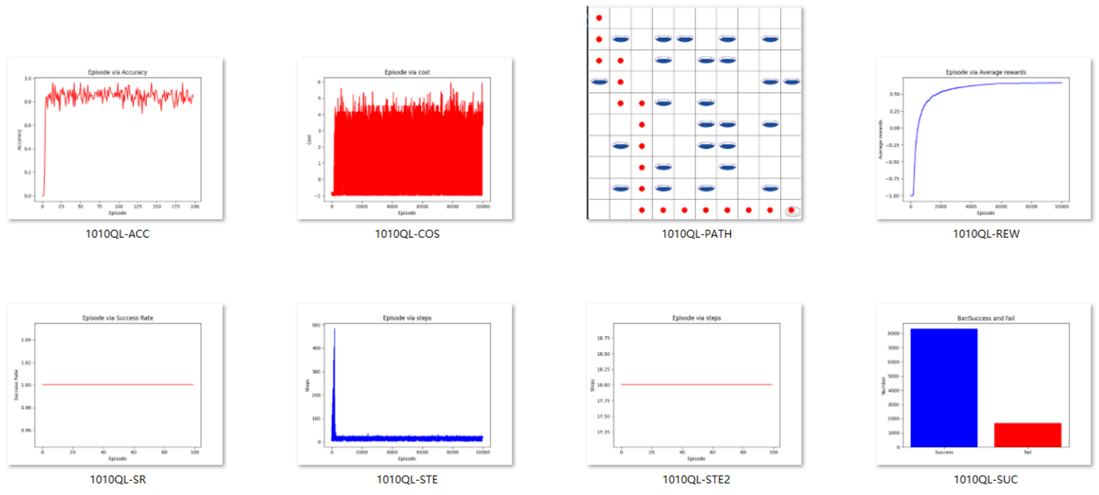


Figure 7 The sum of the result pictures of Q-learning(10)

*NUM_STEPS=1000, NUM_EPISODES=10000, LEARNING_RATA=0.8, GAMMA=0.7, EPSILON=0.05

3. Discussion of results

3.1 Performance and result comparison

In the 4*4 experiment, all three reinforcement learning techniques performed well. When the same number of iterations (10,000) is set, the First-visit Monte Carlo control method has the highest success rate in terms of success rate. This control method can avoid obstacles and reach the target point with the greatest probability. This is also the reason why its average reward is the highest and most stable. Secondly, the Q-learning method also has good accuracy, but its fluctuation range is large, which means that more aggressive parameters are needed when finding paths to obtain better performance. The worst performing algorithm is the SARSA algorithm, which has slightly less fluctuations than Q-learning. In general, it should be more conservative. However, because the exploration rate under optimal conditions is large, and because the environment is a 4*4 square grid, which is relatively small, the number of pitfalls also increases. In order to match the exploration rate, the discount coefficient is also increased at the same time, which means that it pays more attention to future interests, which allows it to eventually stabilize and plan a reasonable route even if the accuracy is low.

In a 10*10 environment, because the installation space is larger, the distribution of holes is also more complex. In this case, the first access to a Monte Carlo control strategy is almost impossible to succeed. As you can see, the success rate is very low, and a maximum number of iterations are required to produce results. Therefore, the average reward curve is not only low in absolute value, but also converges very slowly. For SARSA and Q-learning, the number of iterations required is much less and the convergence rate is much faster. You can see that the reward curve levels off (converges) early. But SARS fluctuated the most, also because the maximum exploration rate and discount factor were set. However, the average value of reward is the highest, which shows that although the parameters are biased toward radicalness, due to the conservative nature of SARSA itself, in the later stages of training, loopholes can be avoided with the greatest probability and the end point can be reached. Q-learning also performs well. Although the average reward value is slightly lower than SARSA, the curve is smoother. The parameter settings in the optimal case are very conservative, with EPSILON only 0.05. But it is precisely this that makes the exploration of Q-learning converge quickly. The number of exploration steps quickly drops to a reasonable range with the number of iterations. I intentionally set the upper limit on the number of steps very high (much higher than the number of steps needed to reach the end) because I want to give the agent more space and possibilities to explore. In this case, Q-learning can stabilize quickly and achieve almost the same success rate as SARS. Therefore, I think Q-learning is the optimal reinforcement learning method in this case.

3.2 Explanation of similarities and differences

3.2.1 Similarities

In each test, if it is successful, the number of steps of their optimal path is always the same. This is because no matter what the environment, if the result can be produced, the result will always converge they the number of steps required is always minimal.

For the 4*4 situation, the First-visit Monte Carlo control is consistent with the final planned route of SARSA, which may be caused by accident in this specific situation. The real-time accuracy and cost fluctuations of SARSA and Q-learning are similar, and the stable patterns of the number of steps are also similar. This is because their algorithm principles are similar. First-visit Monte Carlo control and Q-learning are similar in final accuracy, probably because the two algorithms are more suitable under simple and narrow conditions.

In the 10*10 case, First-visit Monte Carlo control is consistent with Q-learning's path planning, which may be caused by accident in this specific situation. For cost, average reward and final total correct rate, they are all similar in SARSA and Q-learning because their algorithm principles are similar, and the differences are very small.

3.2.2 Differences

There are many points of difference, and the difference in planning on the path should be the result of the interaction of algorithms, parameters, and environment. Also, in this topic, I set a small penalty (-0.01) when going out of bounds. The relative size of this value to the final rewards and penalties (+-1) also affects route selection.

In the 4*4 experiment, the average reward curve of SARSA is very different from the other two. It rises first and then falls, tending to be gentle. This is because its hyperparameters (exploration rate and discount factor) are set relatively aggressively. Average rewards are reduced for short periods of time when large-scale exploration fails. But since the environment itself is not complicated, it quickly stabilized. This is also the reason why its number of failures far exceeds that of the other two algorithms.

In the 10*10 grid environment, the paths planned by SARSA and MC methods are different. This is because the principles of the two methods are different (see Part 1 for details). Therefore, although there is the same optimal solution, the two approximation methods are different, so there will be a situation where the number of steps is the same, but the route is different. Additionally, the reward curve for first access to Monte Carlo controls is not only low in absolute terms but also weird. This is because the MC method may not be suitable for complex environments with limited number of iterations. At the same time, the pattern of application probability also leads to the fragmentation of its upward trend. The differences between SARSA and Q-learning in terms of rewards and step fluctuations have been explained in detail in the first subsection (3.1) of this section. To sum up, the difference in parameter settings within the algorithm far exceeds the difference in the algorithm itself, which leads to faster convergence of Q-learning, but SARSA itself is conservative, so the reward is higher.

Comparing these two situations, we can see that the three strategies perform differently in different environments. This is because the three algorithms themselves are different, and there will naturally be an algorithm that is more suitable for different situations. At the same time, parameters vary greatly under different environments, especially the First-visit Monte Carlo control method. Moreover, although I also set the number of iterations to 10,000 in the 4*4 case, according to the image, all three methods converged early. This is because the environment itself is relatively simple and the amount of calculation is not large.

3.3 Unexpected results and explanations

One of the results that surprised me the most at the beginning was that Q-learning had better reward fluctuation and convergence than SARSA regardless of the situation. The main reason is that when compared generally, the parameters of the two are consistent. In this case, SARSA is more conservative, and Q-learning is greedier. However, in actual situations, to obtain optimal results, the parameter settings within the algorithm vary greatly, exceeding the differences in the original algorithm, so this result occurs. One more thing, I originally thought that the Monte Carlo method should perform better in more complex situations, but the reality is that it is almost difficult to succeed. I think this is partly because I may not have found the optimal hyperparameters but instead picked a local optimum. In addition, it may be that the number of iterations is not enough. Finally, when conducting 10*10 simulations, I originally thought that changing the position of the holes would not cause any changes, but in fact, doing so will not only lead to changes in the optimal parameter settings, but also have a particularly significant impact. Effects on first-access Monte Carlo controls. This may be because changes in hole location can cause significant changes in individual strategies, thereby altering the optimal balance of exploration rates.

3.4 Difficulties

The biggest difficulty encountered is that when using the MC method in a 10*10 environment, a valid path cannot always appear, and the failure rate is 100%. I overcame the randomness by appropriately changing the position of the hole and adjusting a series of coefficients (especially the exploration rate and the number of iterations) including the penalty coefficient for going out of the boundary and simulated multiple times to at least get a planned path. I also encountered some difficulties when configuring the ice lake environment in the early stages, such as how to prevent it from going out of the boundary and pixel settings. I finally completed the environment settings by watching relevant videos and referring to other similar background building codes.

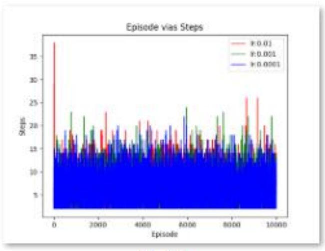
3.5 Own initiatives

For methods to improve efficiency, I established a visual dynamic window to monitor the exploration of different algorithms in real time. The advantage of this is that I can observe in time whether the parameter settings are reasonable and whether the results of this random seed run are excellent. If the resulting live video does not meet expectations, I can terminate the process early and avoid wasting time on tens of thousands of iterations. At the same time, I created a series of visualizations (mostly by number of iterations) so that I could analyze the results and modify the algorithm if problems arose.

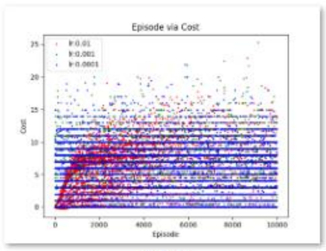
At the same time, the starting position, target position, and hole position in the environment are all variable, which can be used to simply solve similar problems without remodeling. In fact, I have tried to dynamically adjust the exploration rate to solve the problem of MC method having no solution in complex environments. This can reduce the number of iterations and improve the success rate. However, due to the requirements of this task (using the greedy strategy), it was not used in the submitted program.

To more intuitively compare and see the optimal strategies in the 4*4 and 10*10 environments and the impact of various coefficients on the algorithm, I wrote a horizontal comparison test file. Due to space limitations, I only show pictures of project2 and cannot discuss them in detail (all images will be placed in this folder as well as on github).

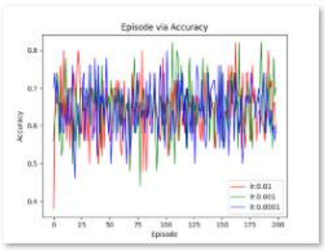
In general, in a 4*4 environment, MC is the optimal algorithm; in a 10*10 environment, q-learning is the optimal algorithm. When each coefficient changes within a certain range, although the function behaves differently at the beginning, it will show a convergence trend after enough iterations. The influence of various coefficients on the results is not increasing or decreasing, but an effective parameter group needs to be found to achieve the optimum.



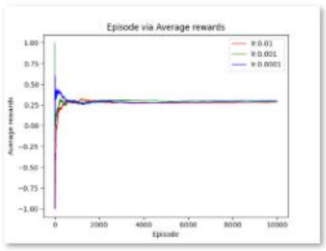
job2-1



job2-2



job2-3



job2-4

Figure 8 The result of job-2

3.6 Other important issues

I think parameter tuning is often the most confusing. Therefore, I think using methods such as Eligibility Traces can "memorize" the previous state-action pairs after the parameter adjustment is completed, reducing the subsequent workload. Also, I noticed that in case of high exploration rate and discount factor. Since the function fluctuates greatly, double learning can be used to reduce the estimation bias and improve the stability and efficiency of learning. Other techniques, such as deep Q-networks (which can improve the ability to handle complex environments), using linear or nonlinear function approximators (such as neural networks) to estimate value functions (to improve efficiency in continuous or high-dimensional state spaces) can also be helpful.

4. Appendix

https://github.com/LYM0905/ME-5406_project1