ME5406

Project 1: Frozen Lake Problem

AY 2022/23 Sem 2

Submitted by

Nguyen Tuan Dung (A0201264W)

Email: dung.nguyen.t@u.nus.edu

Github Link: https://github.com/tuandung161/ME5406-Project-1.git

I. Introduction

a. Objective

This project aims to demonstrate competence in implementing a set of model-free reinforcement learning techniques in a small-scale problem setting, as well as understanding the principles and implementation issues related to this set of techniques. The specific techniques that will be used in this project include Monte Carlo, SARSA and Q-learning.

b. Problem statement

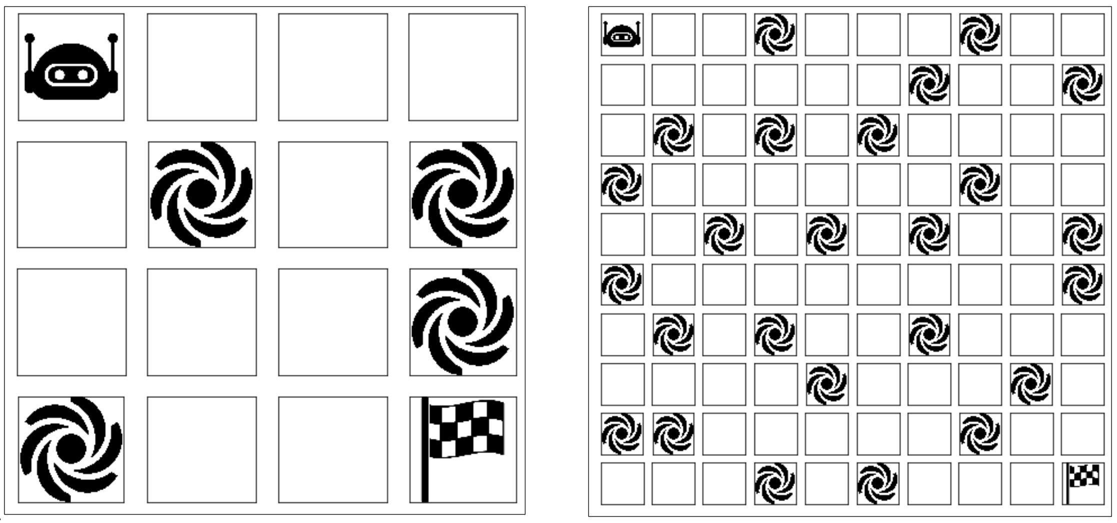
The problem addressed in this project involves navigating a robot on a frozen lake, which has four holes covered by patches of very thin ice. The robot's objective is to move from the top left corner of the grid to the bottom right corner to retrieve a frisbee while avoiding the holes. The RL cast for the project is as follow:

- State space: 4x4 or 10x10 grid with 4 different types of tiles: start, ice, hole, and frisbee (goal)
- Action space: The robot can move in one of four directions (left, right, up and down).
- Reward Structure: +1 for reaching the frisbee, -1 for falling into a hole, and 0 for all other cases
- Completion Criteria: when the robot either reaches the frisbee or falls into a hole.

The project focuses on 3 tabular reinforcement learning techniques and include 2 main tasks:

i. Implementation of First-visit Monte Carlo control, SARSA, and Q-learning.
ii. Extends the implementation to a larger grid (10 x 10) with randomly distributed holes and repeats the three techniques.

The visualization of the frozen lake environment used for learning are shown below:



\

Figure 1: 4x4 (left) and 10x10 (right) grid environment

Customized learning environments for the robot were created for both 4x4 and 10x10 grid size. The code implementation can be found in "Environment.py".

II. Result

a. Hyperparameter Settings:

For comparison purposes the hyper parameter for all 3 algorithms will be set to be the same. Tuning and experimenting on the parameter will be discussed in section III.

No of Episode for Training	10,000
No of Steps (Monte Carlo)	1,000
No of Episode for Testing	1,000

Epsilon (ε)	0.1
Gamma (γ)	0.9
Learning rate (α) (SARSA & Q Learning)	0.1

Table 1: Hyperparameter setting for learning.

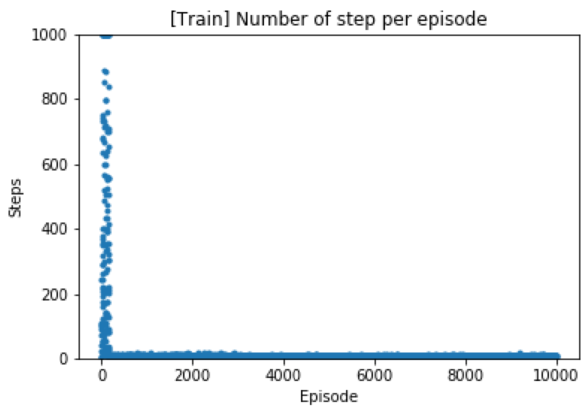
b. Monte Carlo

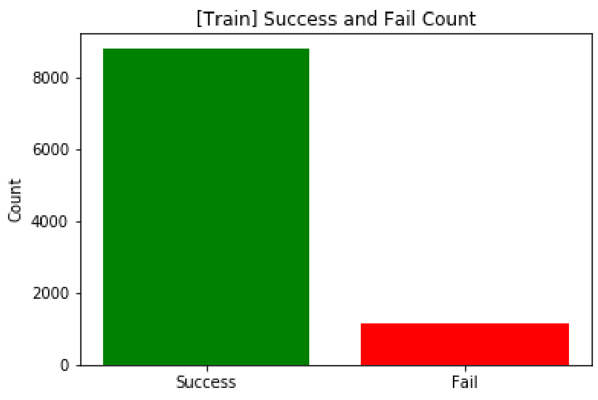
i. Overview:

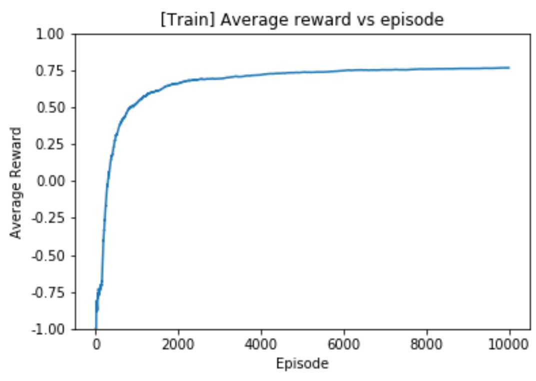
Monte Carlo approximates the value function of a policy using the concept of sampling. It estimates the value function by averaging the returns observed during multiple episodes of interaction with the environment. In Monte Carlo, the agent plays through an entire episode until termination and then updates the value function based on the observed returns:

$$V(S_t) = V(S_t) + \frac{1}{n}(\gamma_t - V(S_t))$$

ii. 4x4 Grid





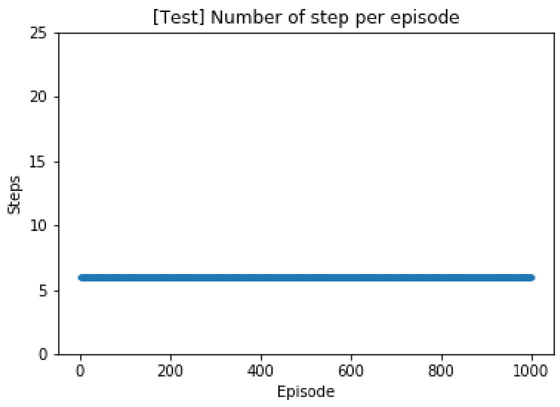


Train Duration (seconds)	4.34
Train success percentage	88.08%
Highest steps count	1,000
Average steps count	14.15
Average reward	0.7654

Figure 2: Training result for Monte Carlo 4x4

For training using Monte Carlo on 4x4 grid, the majority of the learning was made within the first 200 episodes. The average steps count for the first 200 steps were found to be 397.7 steps while the average steps count for the remaining episode was 6.3 steps.

Using optimal policy on the Q table, the result of the for the test set is as follow:



Test success percentage	100%
Average step counts	6
Average reward	1

Figure 3: Testing result for Monte Carlo 4x4

The success rate for the test set was 100% with all episodes completed within 6 steps which is the shortest path to reach the goal.

iii. 10x10 grid

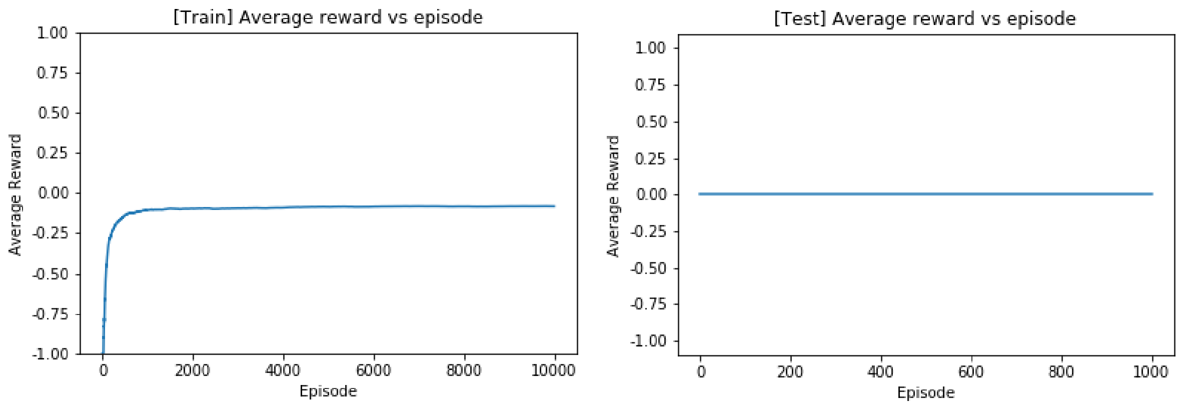
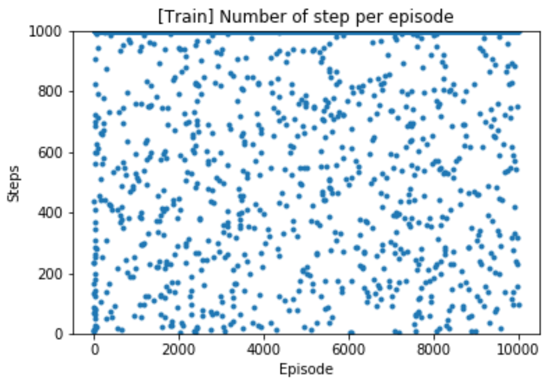


Figure 4: Average reward for train and test set in 10x10 grid



Train Duration (seconds)	81.83
Train success percentage	0%
Train Average step counts	955.5
Train Average reward	-0.0846
Test success percentage	0%

Figure 5: Results for Monte Carlo 10x10

When using Monte Carlo on 10x10 grid, the model failed to find the path to goal position after 10,000 episodes. The average step counts for episodes was 955.5 which suggests that most of episode reaches the step limits and terminated without being able to find the path to the goals.

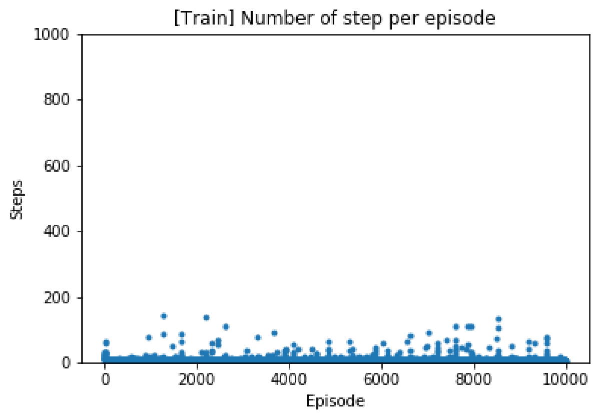
c. SARSA

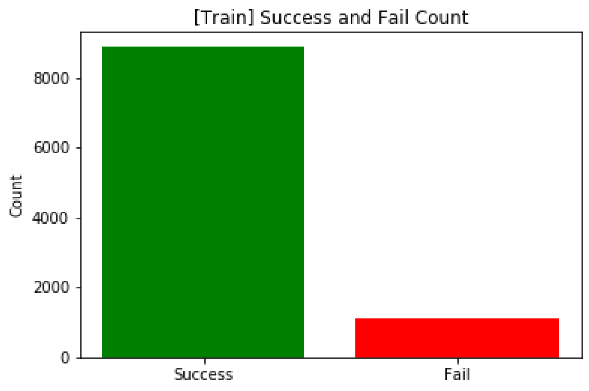
i. Overview

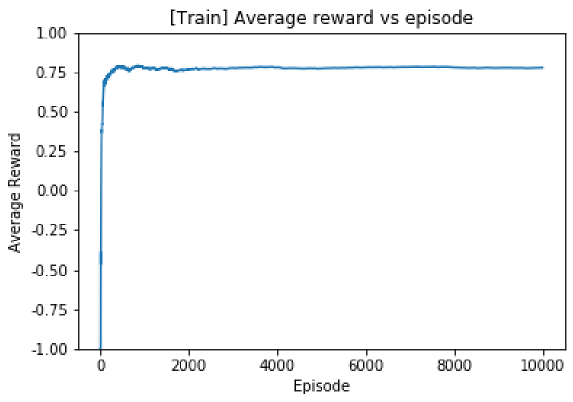
SARSA is an on-policy learning algorithm which learns from the current value function of the policy. The algorithm updates the Q-function after each transition using the following formula:

$$Q(S,A) = Q(S,A) + \alpha \times [reward + \gamma \times Q(S',A') - Q(S,A)]$$

ii. 4x4 Grid







Train Duration (seconds)	3.41
Train success percentage	88.88%
Highest steps count	143
Average steps count	6.86
Average reward	0.7776

Figure 6: Training result for SARSA 4x4

On 4x4 grid, SARSA completes 10,000 episodes in 3.41 seconds with 88,88% success rate. The average steps count for all episodes is 6.86 with the highest being 143 steps.

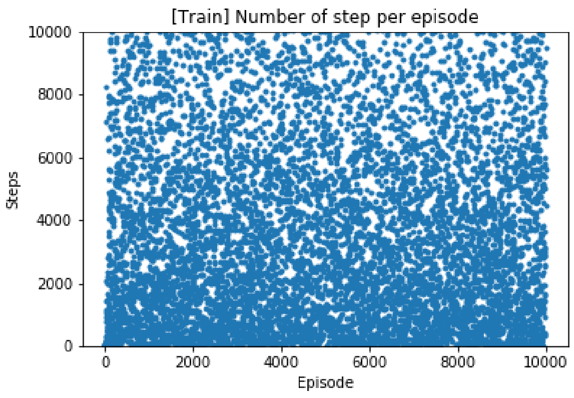
Test success percentage	100%
Average step counts	6
Average reward	1

Figure 7: Testing result for SARSA 4x4

Similar to Monte Carlo, when running test set using optimal policy, the policy from SARSA achieve 100% success rate, with the shortest path of 6 steps achieved for every episode.

iii. 10x10 Grid

When running for 10x10 grid, SARSA learning algorithm is found to be inadequate in finding the right policy to reach the final goal. Out of 10,000 episodes, only 0.31% was successful in reaching the final goal. The step count is also very high, averaging 7,486 steps per episode and the highest step count in an episode is 103,629 steps.



Train Duration (seconds)	684.14
Train success percentage	0.31%
Train Average step counts	7486.2
Train Average reward	-0.9938
Test success percentage	0%

Figure 8: Results for SARSA 10x10 grid

d. <u>Q-Learning</u>

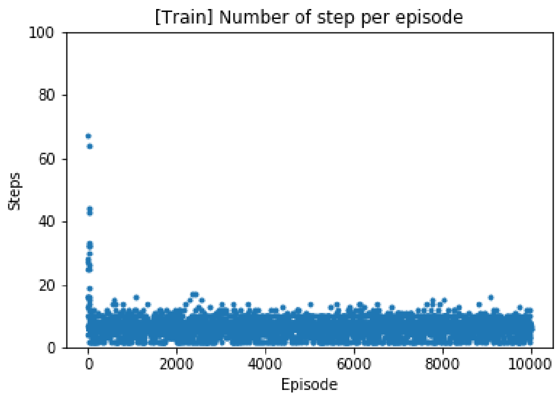
i. *Overview:*

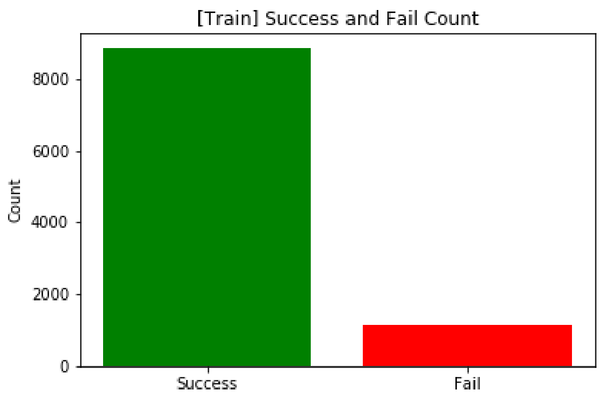
Q learning allows an agent to make optimal decisions in a given environment by maximizing a cumulative reward signal. As the Q values are updated over time, the agent learns to take actions that maximize its long-term rewards. The Q function is updated as follows:

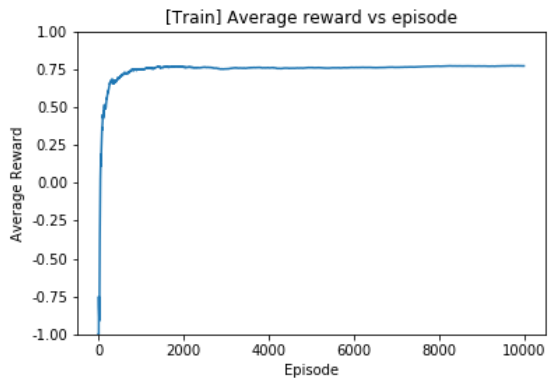
$$Q(S,A) = Q(S,A) + \alpha \times [reward + \gamma \times \max(Q(S',A')) - Q(S,A)]$$

ii. 4x4 Grid

On 4x4 grid, Q-Learning completes 10,000 episodes in 3.81 seconds with 88,53% success rate. The average steps count for all episodes is 6.4 with the highest being 67 steps.







Train Duration (seconds)	3.81
Train success percentage	88.53%
Highest steps count	67
Average steps count	6.4
Average reward	0.7706

Figure 8: Training result for Q-Learning 4x4

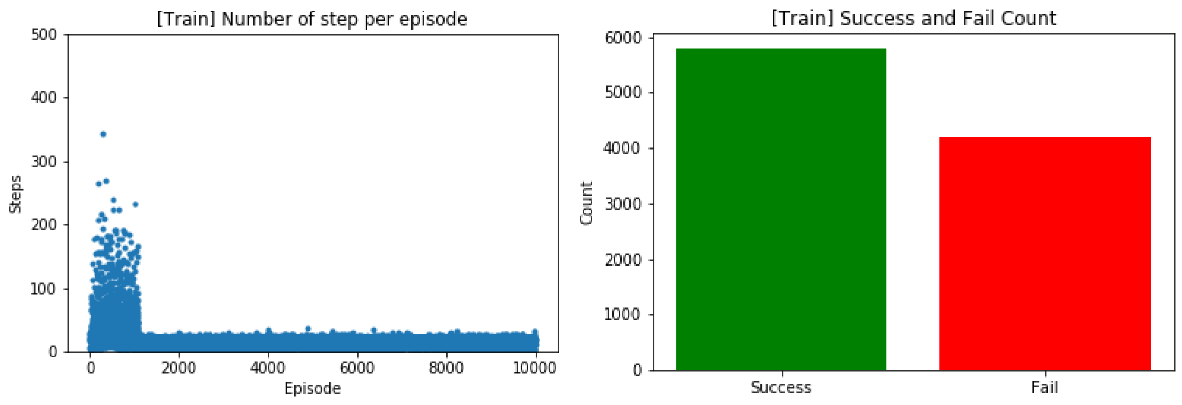
Similar to the previous algorithms, when running test set using optimal policy, the policy from SARSA achieve 100% success rate, with the shortest path of 6 steps achieved for every episode.

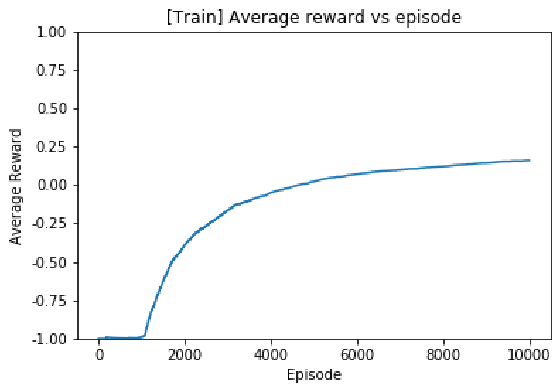
Test success percentage	100%
Average step counts	6
Average reward	1

Figure 9: Testing result for Q-Learning 4x4

iii. 10x10 grid

For 10x10 grid, Q-learning algorithm completes training set in 9.5 seconds with average reward of 0.16 and average steps per episode of 20.6 steps.

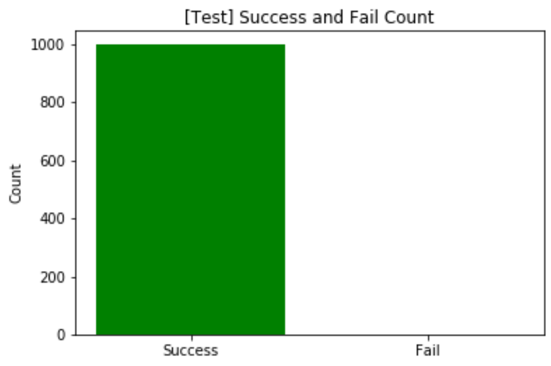




Train Duration (seconds)	9.5
Train success percentage	57.98%
Highest steps count	342
Average steps count	20.6
Average reward	0.1596

Figure 10: Training result for Q-Learning 10x10

The algorithm manages to train a Q function when run under optimal policy produces 100% success rate to reach goal location using shortest path (18 steps).



Test success percentage	100%
Highest steps count	18
Average steps count	18
Average reward	1

Figure 10: Test result for Q-Learning 10x10

e. <u>Comparison between Monte Carlo, SARSA & Q-Learning</u>

For small grid size (4x4), all 3 algorithms have relatively similar performance in terms of speed and success percentage, and all produce a policy that can be run optimally to reach 100% success rate in test set. The average steps count for SARSA (6.86) and Q-learning(6.4) is lower than that of Monte Carlo (14.15) which can be due to the high variance in estimates by Monte Carlo algorithm, which leads to slower convergence. However, for such a small grid size, the random exploration probability for the initial few episodes can greatly influence the eventual average.

The differences in performance between the 3 algorithms are more obvious in the case of 10x10 grid.

	Monte Carlo	SARSA	Q-Learning
Train Duration (s)	81.83	684.14	9.5
Train Success Percentage	0%	0.31%	57.98%
Highest Step Count	1,000	103,629	342
Average Step Count	955.5	7486.2	20.6
Average Reward	-0.0846	-0.9938	0.1596
Test Success Percentage	0%	0%	100%

Figure 11: Comparison table between MC, SARSA & Q-Learning (10x10 grid)

As shown in Figure 11, Q-learning is the best performing algorithm and the only to successfully create an optimal policy for reaching target location. The better performance can be explained by:

- Q-learning estimates the optimal Q-value function directly, without directly learning the policy. This can be faster than SARSA, which updates the policy during learning, and Monte Carlo, which estimates the Q-value of a policy based on complete episodes of experience.

- SARSA also requires more careful exploration-exploitation trade-offs to balance the exploration of the environment and the exploitation of the learned policy, while Q-learning is less sensitive to this trade-off

Between SARSA and Monte Carlo, SARSA typically has a longer train duration and step counts due to SARSA not having a steps limit while Monte Carlo is stopped after 1,000 steps. From the result, it can be observed that most Monte Carlo episodes are terminated after 1,000 steps without finding the goal or hole and get 0 reward. Meanwhile, SARSA episodes mostly run until it hits a hole and gets -1 reward.

III. Improvement

a. Monte Carlo

i. Hyper-parameter tuning:

For Monte Carlo, the suspected issue is the high variance when averaging the returns obtained from all episodes that visit the state-action pair. This leads to slow convergence to optimal policy. Due to this, the first approach was to increase the number of episodes used for training. However, this was found to be ineffective as the algorithm still fails to solve the maze when the episode count is increased to 50,000.

Another observation when observing the Q table is that the value is only updated until row 40+ which suggests that the robot often reaches max step before it could reach the lower row of 10x10 grid. Hence, the number of steps per episode was increased to 10,000 to enable the robot to travel deeper into the maze. However, even with this parameter tuning, Monte Carlo algorithms still fail to solve the maze.

ii. Dynamic Epsilon Value

To improve the success rate and convergence speed for the algorithms, one possible method is to have dynamic epsilon value which tuned as the episodes progress. Epsilon represents the probability of choosing a random action instead of the action with the highest expected reward, or exploration-exploitation trade-off. For more optimal exploration-exploitation trade-off relationship, epsilon should be higher in the beginning to encourage exploration and lowers overtime to encourage exploitation. The decay function is as follow:

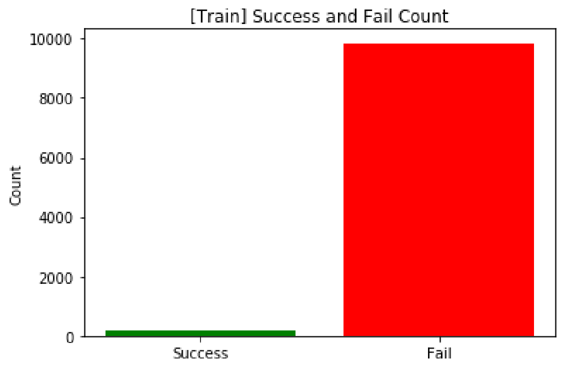
$$\varepsilon = \varepsilon_{start} \times \left(\frac{\varepsilon_{end}}{\varepsilon_{start}}\right)^{\frac{episode}{rate}}$$

After implementing dynamic epsilon value for Monte Carlo algorithm, it was observed that the rate of training increases as the agent takes more random action. However, the algorithm still failed to reach an optimal policy for the environment.

b. SARSA

i. Hyper-parameter tuning:

For SARSA, the suspected issue was the rate at which Q-value is being updated is too low. Causing the value to be stuck in local maxima and fails to reach convergence. Hence, the first approach was to increase the learning rate at which the Q-value is being updated. At the learning rate of 0.3, the train success percentage improves to 1.59%. The training speed also improves. However, the test success percentage is still 0% indicating that the optimal policy is yet to be achieved.

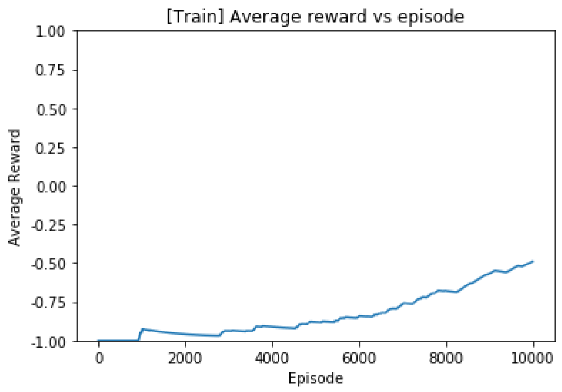


Train Duration (seconds)	258.5
Train success percentage	1.59 %
Train Average reward	- 0.9682
Test success percentage	0%

Figure 12: Training result for SARSA 10x10 at learning rate of 0.3

ii. Dynamic Learning Rate & Epsilon Value

To build up on the small improvement achieved in the previous section, using the same formula for decay rate section a, decaying learning rate and epsilon was implemented with both being high at the start and decay over time. This approach aims to encourage exploration and learning at the beginning stages of training and shift towards favoring exploitation as the episodes progress.



Learn rate decay	0.9 → 0.3
Epsilon decay	0.1 → 0.01
Train Duration (seconds)	563.31
Train success percentage	25.45 %
Train Average reward	- 0.491
Test success percentage	**100%**

Figure 13: Training result for SARSA 10x10 at with decaying epsilon and learning rate

After implementing decaying learning rate and epsilon, the SARSA algorithm managed to produce an optimal policy resulting in 100% success rate for test case. However, the optimal policy produced by SARSA does not result in the shortest route as compared to that of Q-learning.

IV. Conclusion

In conclusion, the project successfully implemented and compared three reinforcement learning algorithms: Monte Carlo, SARSA and Q-learning, to train a robot in solving a maze with randomized obstacles. The performance of the algorithms was compared and evaluated based on the success rate, training speed and optimal path lengths.

Monte Carlo achieved a good success rate for smaller grid size (4x4). However, the algorithm fails to handle larger grid (10x10) since it is affected by the long-term dependencies of the environment.

Both SARSA and Q-learning also achieved a good success rate for 4x4 grid. However, for 10x10 grid, SARSA required decaying learning rate and epsilon value to be implemented to achieve optimal policy. In addition, the training speed for SARSA was much slower than that of Q-learning. Finally, policy by Q-learning also produces the shortest path while this is not true for SARSA.

Overall, the frozen lake project has provided valuable insights into the strengths and limitations of different reinforcement learning algorithms in solving maze problems. The learnings from this project have provided the foundational knowledge which can help guide development of more advanced algorithms which can handle more complex tasks and environments.

V. Future work

While the epsilon-greedy policy is a simple and effective exploration-exploitation strategy, it may not be the most efficient or optimal way to balance exploration and exploitation. For future scope, alternative exploration strategies can be considered and tested. Some examples include:

- Upper Confidence Bound which encourages the agent to select actions with higher uncertainty or potential reward, resulting in more efficient exploration.
- Thompson Sampling which balances exploration and exploitation by maintaining an uncertainty estimate of the environment.