Table of content:

1. Introduction
   1. Objective
   2. Problem Statement
2. Result
   1. Hyperparameter setup
   2. Monte-Carlo
   3. SARSA
   4. Q-Learning
   5. Comparison between the three algorithms
3. Improvement
   1. Tuning of hyperparameter (epsilon, gamma & learning rate, number of episodes)
   2. Pros & Cons
4. Conclusion
5. Further Work
6. Introduction
   1. 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.

* 1. 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 three tabular reinforcement learning techniques and include 2 main tasks:

1. Implementation of First-visit Monte Carlo control, SARSA, and Q-learning.
2. 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:

\Shape

Description automatically generated Calendar

Description automatically generated

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”.

1. Result
2. 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 |  | Epsilon (ε) | 0.1 |
| No of Steps (Monte Carlo) | 1,000 |  | Gamma (γ) | 0.9 |
| No of Episode for Testing | 1,000 |  | Learning rate (a) (SARSA & Q Learning) | 0.1 |

Table 1: Hyperparameter setting for learning.

1. Monte Carlo
2. 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:

1. 4x4 Grid

Chart

Description automatically generatedChart, bar chart

Description automatically generated

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

Chart

Description automatically generated

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:

Chart

Description automatically generatedChart

Description automatically generatedChart, bar chart

Description automatically generated

|  |  |
| --- | --- |
| 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.

1. 10x10 grid

Chart

Description automatically generated Chart

Description automatically generated

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% |

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.

Chart, scatter chart

Description automatically generated

Figure 5: Testing result for Monte Carlo 10x10

1. SARSA
2. 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:

1. 4x4 Grid

Chart

Description automatically generated with medium confidenceChart, scatter chart

Description automatically generatedChart, bar chart

Description automatically generated

|  |  |
| --- | --- |
| 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.

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.

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

Figure 7: Testing result for SARSA 4x4

1. 10x10 Grid