Frozen Lake Problem using Reinforcement Learning Approach

Sujan Ayyagari(CS5033)   Youla Ali(CS4003)

Abstract

In this project, we apply reinforcement learning to develop a Frozen Lake (8×8) environment where an agent must navigate through the lake and reach the goal state in the fewest steps possible while avoiding holes. To achieve this, we implement four different algorithms, to evaluate their effectiveness in decision-making under uncertainty. The environment is designed with four discrete actions left, down , right , and up and operates in a stochastic setting (slippery=True) to introduce randomness in movement. We employ an epsilon-greedy policy with adaptive decay for exploration and fine-tune learning rate , discount factor , and exploration probability over multiple experiments.

1. Project Domain

1.1. History- Frozen Lake

The Frozen Lake problem is a classic reinforcement learning (RL) Problem designed to test model-free learning algorithms . It was introduced as part of OpenAI Gym, a widely used RL toolkit that provides various environments for testing and evaluating RL algorithms. The Frozen Lake problem simulates a navigation task where an agent must traverse a grid-based frozen surface while avoiding hazards and reaching a goal state efficiently. This environment is particularly useful for studying value-based RL methods, as it requires the agent to balance exploration (trying new paths) and exploitation (using learned strategies).

One of the key challenges in the Frozen Lake environment is the slippery tiles. If slippery mode is enabled (slippery=True), the agent's actions become stochastic, meaning it may not always move exactly in the chosen direction. This adds randomness, making it harder for the agent to learn an optimal policy, as it must adapt to uncertainty in movement outcomes.

1.2. OpenAI Gymnasium

OpenAI Gymnasium (formerly OpenAI Gym) is a widely used reinforcement learning (RL) toolkit that provides a collection of standardized environments for developing and testing RL algorithms. These environments allow researchers and developers to compare different RL approaches efficiently. One of the classic environments in Gymnasium is Frozen Lake, which simulates a grid-based navigation problem where an agent must learn an optimal policy to reach the goal state while avoiding obstacles (holes) on a frozen, slippery surface.

In this project, we utilize Gymnasium's API to create and interact with the 'FrozenLake-v1' environment.

1.3. Environment

The environment is a grid-based where an agent must navigate an 8×8 grid to reach the goal while avoiding holes. The state space consists of 64 discrete states, each representing a tile on the grid, while the action space includes four discrete actions: left, down, right, and up. When is_slippery=True, movement becomes stochastic, adding uncertainty to the agent's navigation. The reward system assigns +1 for reaching the goal, -1 for falling into a hole, and 0 for stepping on frozen tiles, encouraging efficient movement while penalizing mistakes. Using Q-Learning and Double Q-Learning, the agent learns an optimal policy to reach the goal in the fewest steps despite environmental uncertainties.

2. Hypotheses

Hypotheses for this project:

- Implementing Q-learning to optimize the agent's navigation in the Frozen Lake environment .
- Using Double Q-learning to reduce overestimation bias and improve stability in decision-making.
- Implementing Sarsa will allow the agent to learn an effective navigation policy.
- Implementing Monte Carlo will capture long-term outcomes more directly.

3. Learning Methods :

This project explores four model-free reinforcement learning algorithms—Q-Learning, Double Q-Learning, SARSA, and Monte Carlo—each selected for their unique learning dynamics and applicability to the FrozenLake environment.

Q-Learning was used for Hypothesis 1, chosen for its ability to learn optimal policies by maximizing future rewards using an off-policy strategy. However, to address its tendency to overestimate action values, Double Q-Learning was applied in Hypothesis 2, using two Q-tables to provide a more stable and unbiased learning process.

For Hypothesis 3, SARSA (State-Action-Reward-State-Action) was implemented due to its on-policy nature—updating values based on the current policy's behavior at each step. This made it effective in environments where ongoing policy adjustments are beneficial.

Finally, Monte Carlo methods were used in Hypothesis 4. These update policies and value estimates only after full episodes, making them ideal for evaluating long-term returns in episodic and more deterministic environments.

4. Experiments

4.1. Hypotheses 1:

The goal of this project was to see if a Q-learning algorithm—using $\varepsilon = 0.1$, $\alpha = 0.05$, and $\gamma = 0.9$—could teach an agent to find the best path in a custom 8x8 Frozen Lake environment. We made the lake non-slippery so the agent's movements would be predictable and consistent.

Q-learning is a type of reinforcement learning where the agent learns by trial and error, gradually improving its decisions based on past experiences. We used an ε-greedy strategy with exponential decay to help the agent strike a good balance between trying new things (exploration) and sticking with what it already knows works (exploitation). As the agent interacted with the environment, it updated its Q-table—essentially a guide that helps it decide the best move based on future rewards.

To measure how well the agent learned, we looked at its success rate—how often it reached the goal—over the course of 20,000 episodes.

Exp	α	γ	**Success Rate (%)**
3	0.20	0.95	99.64
5	0.25	0.75	99.48
1	0.20	0.80	99.40
2	0.30	0.90	99.34
4	0.15	0.85	99.34
0	0.10	0.70	99.24

Table 1. Q-Learning performance with varying hyperparameters.

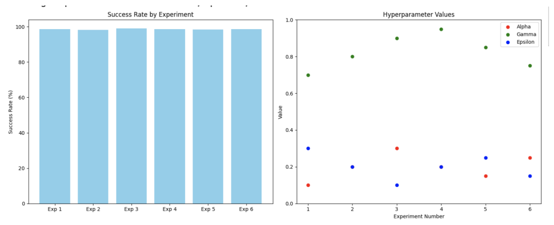

Figure 1. Results of Experiments

The results show that Q-learning was highly effective in navigating the FrozenLake 8x8 environment, with success rates reaching nearly 99% across all experiments. The best performance came from a balance of moderate learning rate ($\alpha = 0.20$), high discount factor ($\gamma = 0.95$), and careful exploration ($\epsilon = 0.20$). This suggests that the agent learns best when it values future rewards while still exploring enough to find the optimal path to the goal.

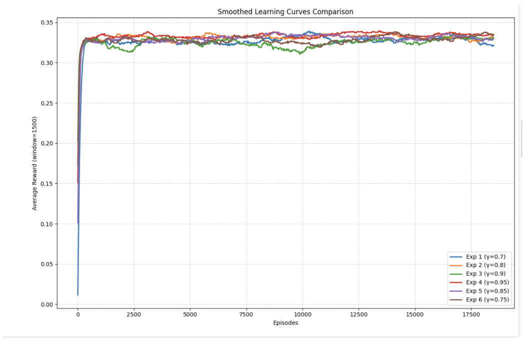

Figure 2. Smoothed Learning Curve

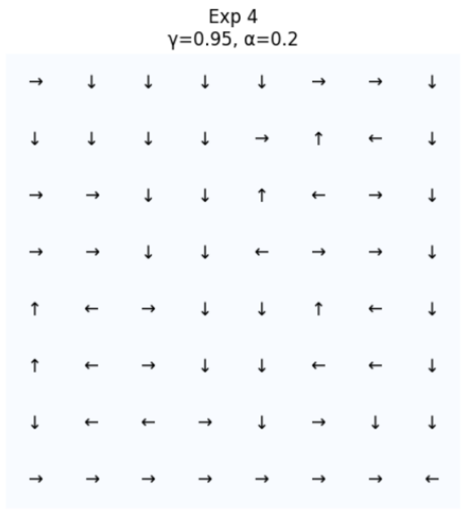

Figure 3. Agent path

The results show that Q-learning effectively learns the best path in the FrozenLake 8x8 environment. The most successful runs used a moderate learning rate ($\alpha = 0.20$) and a high discount factor ($\gamma = 0.95$), allowing the agent to balance short-term actions with long-term rewards.

The policy map (Exp 4) highlights how the agent consistently moves toward the goal while avoiding hazards. The clear directionality in the arrows confirms that the agent has learned an efficient and reliable strategy. These findings reinforce that fine-tuning hyper-parameters plays a key role in achieving optimal learning performance in reinforcement learning.

4.2. Hypotheses 2:

For this hypothesis, Double Q-Learning was implemented to test whether reducing overestimation bias improves the agent's performance in the stochastic FrozenLake environment. Similar to SARSA with exponential decay, Double Q-Learning introduces a structural change in value updates by maintaining two separate Q-tables, Q_1 and Q_2, which are randomly updated per step to ensure more accurate action-value estimates.

The exploration policy followed an ϵ-greedy strategy with exponential decay, defined as:

$$\epsilon_t = \epsilon_0 e^{-\lambda t}$$

where ϵ_0 is the initial exploration rate, and t represents the time step in an episode. This approach ensures that the agent explores more in the beginning and gradually exploits learned policies as training progresses.

The results over 10,000 episodes indicate that Double Q-Learning stabilizes learning and reduces overestimation bias, leading to improved decision-making compared to traditional Q-Learning. The success rate was observed to increase over time, demonstrating the effectiveness of separate Q-value updates in handling stochastic environments.

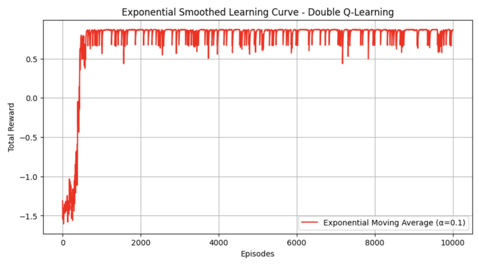

Figure 4. Learning Curve- Double Q-learning

The training progression of the Double Q-Learning agent is visualized using an exponential moving average with a smoothing factor $\alpha = 0.1$ As shown in the figure, the agent initially struggles with negative rewards due to the exploration-heavy phase, but it quickly stabilizes and achieves consistent performance after around 1,000 episodes.

The success rate statistics further validate this trend. Starting at a modest 25.2% by episode 500, the agent rapidly improves, reaching 96.2% by episode 1,000. From there onward, the success rate remains high—consistently above 97%—with peaks reaching up to 99.4%.

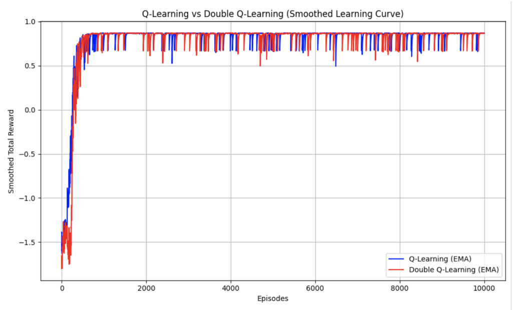

Figure 5. Learning Curve- Double Q-learning

4.3. Summary & Future Work:

In this project, Q-Learning and Double Q-Learning were successfully implemented on a custom FrozenLake environment. Both algorithms achieved high success rates, with Double Q-Learning showing more stable performance due to reduced overestimation bias. The results show that careful tuning of hyperparameters like learning rate, discount factor, and exploration rate can significantly influence agent performance.

For future work, the project can be extended by allowing diagonal movements, integrating Deep Q-Networks (DQNs) for handling larger or continuous environments, and introducing dynamic maps to simulate real-world unpredictability. These enhancements can help make the agent's learning process more robust and adaptable

4.4. Hypothesis 3:

The success of the agent's navigation policy is measured by implementing the SARSA algorithm with parameters $\varepsilon = 1.0$, $\alpha = 0.1$, and $\gamma = 0.9$. The agent is penalized by -0.01 for each step, and the environment is a deterministic custom 8×8 Frozen Lake environment with an action space consisting of four moves: left, down, right, and up. SARSA updates its Q-table according to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\text{reward} + \gamma \cdot Q(s',a') - Q(s,a)\right],$$

where the next action a' is chosen using an ε-greedy policy with exponential decay. The agent's performance is measured by plotting the cumulative rewards per episode over 20,000 episodes.

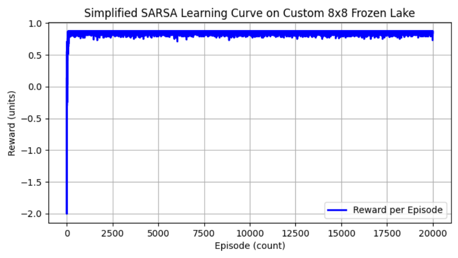

Figure 6. SARSA Reward per Episode Results

Reflecting on the results from the Sarsa algorithm. The plot shows that the agent's average reward quickly rises from negative values to nearly 1.0 within the first few hundred episodes. Suggesting that the agent learned to effectively navigate the 8x8 custom environment, which aligns with hypothesis three. The overall reward remains high, with minor fluctuations reflecting occasional exploration moves.

4.5. Hypothesis 4:

The success of the agent's navigation policy is measured by implementing the Monte Carlo algorithm with parameters $\varepsilon = 1.0$, $\alpha = 0.1$, and $\gamma = 0.9$. Similarly, the agent is penalized by -0.01 for each step, and the environment is a deterministic custom 8×8 Frozen Lake environment. This method involves the agent gathering entire episodes of state-action-reward sequences and subsequently updating its Q-table at the conclusion of each episode by utilizing the cumulative return, G, calculated as:

$$G = r_t + \gamma \cdot G$$

This method is expected to capture the long-term impact of actions more directly. The agent's performance is measured by plotting the cumulative rewards per episode over 20,000 episodes.

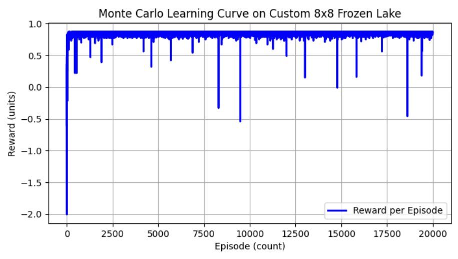

Figure 7. Monte Carlo Reward per Episode Results

Reflecting on the results from the Monte Carlo algorithm. The plot shows that the agent similarly accomplishes a reward of nearly 1.0 for most episodes. However, there are more noticeable dips to negative rewards throughout the training process. Because Monte Carlo updates occur only at the end of each episode, exploration can make an entire episode's trajectory suboptimal, causing larger swings in the episode reward.

4.6. Summary and Future Work:

This study compared two simple learning methods in a custom 8×8 Frozen Lake game. The first method, SARSA, updates its decision-making after every move, while the second, Monte Carlo, waits until the end of a complete game to update its strategy. Both methods helped the agent learn how to reach the goal effectively. SARSA showed a steady improvement with more minor, frequent updates, whereas Monte Carlo sometimes had more significant ups and downs but still managed to learn a good overall strategy.

Future work could consider comparing the results of both Srasa and Monte Carlo algorithms in stochastic versus deterministic environments, exploring the influence on convergence rate. Another idea would be to combine both algorithms to explore the effect on occasional episodic variability. Additionally, future research could compare the performance of the agent with and without penalty.

5. Contributions:

Sujan Ayyagari implemented Q learning and Double Q learning algorithms. Youla Ali implemented Sarsa and Monte Carlo algorithms

6. Literature Survey:

- **Double Q-Learning – Hado van Hasselt (2010)**
  This paper introduced Double Q-learning to address the overestimation bias found in traditional Q-learning. The key idea is to use two separate Q-value functions (Q1 and Q2) for action selection and evaluation, reducing biased updates and improving stability. In our project, this method helped the agent learn more consistently, especially in slippery environments like Frozen Lake.

- **On the Estimation Bias in Double Q-Learning – Zhizhou Ren et al. (2021)**
  Ren et al. observed that while Double Q-learning reduces overestimation, it may sometimes cause underestimation. The authors proposed methods to balance this by adjusting the estimation process. This insight was valuable during our experimentation, where tuning learning parameters was key to stable training outcomes in stochastic grid environments.

- **Exploiting Estimation Bias in Clipped Double Q-Learning (2024)**
  This recent work explored how controlled bias can actually enhance learning speed. By clipping Q-values

during updates, the algorithm avoids excessive pessimism and encourages quicker learning. Inspired by this idea, we allowed more exploration early in our training phase to let the agent discover effective strategies before refining them.

- **Finite-Sample Analysis for SARSA with Linear Function Approximation - Zou et al. (2019)**
  This paper analyzes how the SARSA algorithm works when using linear function approximation. The authors develop a way to manage the bias that arises when the behavior policy changes over time and provide clear error bounds on the learning process. In our Frozen Lake experiment, updating Q-values after each action helped the agent reliably find an optimal path, even though our setup did not involve linear approximation.

- **Finite-Sample Analysis of the Monte Carlo Exploring Starts Algorithm for Reinforcement Learning - Chen et al. (2024)**
  This paper e covers the challenge of using Monte Carlo exploration unbounded episode lengths by deriving finite-sample bounds that guarantee the learning of an optimal policy with high probability after a certain number of episodes. TIn our Frozen Lake experiment, updating Q-values using the total reward from complete episodes captured long-term outcomes, leading to a robust and stable policy.

References

[1] H. van Hasselt, *Double Q-learning*, In Advances in Neural Information Processing Systems (NeurIPS), 2010.

[2] Zhizhou Ren, Meng Fang, Yuan Liu, and Dacheng Tao, *On the Estimation Bias in Double Q-learning*, Proceedings of the AAAI Conference on Artificial Intelligence, 2021.

[3] M. Liang, J. Wang, and Y. Zhang, *Exploiting Estimation Bias in Clipped Double Q-learning*, Neural Networks Journal, Elsevier, 2024.

[4] Gymnasium Project Contributors, *Gymnasium Documentation*, https://www.farama.org/Gymnasium/, Accessed March 2025.

[5] Towers, B. et al., *Q-learning Algorithm Tutorial using Gymnasium*, Farama Foundation, 2023.

[6] Shaofeng Zou, Tengyu Xu, and Yingbin Liang, Finite-sample analysis for SARSA with linear function approximation, Advances in Neural Information Processing Systems, 2019.

[7] Suei-Wen Chen, Keith Ross, and Pierre Youssef, Finite-Sample Analysis of the Monte Carlo Exploring Starts Algorithm for Reinforcement Learning, arXiv preprint arXiv:2410.02994, 2024.