# Reinforcement Learning Project Report: Solving FrozenLake with Q-Learning

Reinforcement Learning Project Report: Solving FrozenLake with Q-Learning

1. Introduction

The Frozen Lake environment is a classic benchmark in reinforcement learning, modeled as a Markov Decision Process (MDP). It features a 4x4 slippery grid with states representing safe tiles, holes, a start, and a goal. The agent must learn to navigate to the goal while avoiding holes under uncertain movement outcomes due to the slippery surface. This report summarizes a data-driven approach using Q-learning and improvements via an exploration–exploitation strategy.

---------------------------------------------

2. Methodology

(a) Environment Setup  
We created a 4x4 FrozenLake environment using the `FrozenLake-v1` version from the Gymnasium Python package:  
```python  
import gymnasium as gym  
env = gym.make("FrozenLake-v1", map\_name="4x4", is\_slippery=True)  
```  
The environment represents 16 states in a grid (S: start, G: goal, F: frozen, H: hole):  
```  
S F F F  
H H F H  
F F F H  
H F F G  
```  
The agent starts at 'S' and must reach 'G' while avoiding holes ('H'). Actions are stochastic due to the slippery surface.

---------------------------------------------

(b) Data Collection (10,000 Episodes)  
We collected data across 10,000 episodes. For each timestep in each episode, we recorded:  
- \*\*State\*\*: Current state before action  
- \*\*Action\*\*: Chosen action (0=left, 1=down, 2=right, 3=up)  
- \*\*Reward\*\*: Immediate reward (0 or 1)  
- \*\*Total Reward\*\*: Sum of all rewards in the episode  
- \*\*Goal Proximity\*\*: Estimated minimum number of steps to goal (Manhattan distance)

This dataset helped to analyze the agent's learning behavior and informed model training.

---------------------------------------------

(c) Goal Achievement Analysis  
We evaluated how many episodes successfully reached the goal:  
```python  
success\_rate = successful\_episodes / total\_episodes  
```  
\*\*Result:\*\*  
- \*\*Success Rate\*\*: ~0.0000 (agent was unable to find the goal without learning)

---------------------------------------------

(d) Action Importance Metric  
We defined the importance of an action taken at state \( s \) as the increase in expected future reward (Q-value). Mathematically:  
```python  
Importance(s, a) = Q[s][a] / sum(Q[s])  
```  
This metric helps identify which actions are most critical in each state.

---------------------------------------------

3. Learning Approaches

(e) Q-learning: Predicting State-Action Value (Q)  
We implemented the Q-learning algorithm to train the agent. The Bellman update rule was used:  
```python  
Q[s, a] = Q[s, a] + lr \* (reward + gamma \* max(Q[s']) - Q[s, a])  
```  
- Learning rate (lr): 0.8  
- Discount factor (gamma): 0.95  
- Exploration rate (epsilon): 0.1 (introduced later)

\*\*Results:\*\*  
- Q-values for state 0: `[0.2380, 0.0153, 0.0176, 0.0155]`  
- Test Success Rate (1000 episodes): \*\*74.50%\*\* (preliminary test after training)

---------------------------------------------

(f) Action Policy using Trained Q-table  
We used the trained Q-table to guide the agent through the environment. The agent chose the best action in each state based on learned Q-values.

\*\*Output:\*\*  
```  
Episode finished - Total Reward: 0.0, Steps Taken: 23, Success: False  
```  
This showed that without exploration during training, the agent failed to generalize.

---------------------------------------------

(g) Evaluation of Learned Policy  
We evaluated the Q-learning policy over 10,000 test episodes:  
```  
Success Rate: 0.3595  
Average Reward: 0.3595  
Average Steps: 40.14  
```  
This confirms that some level of goal-reaching behavior emerged but with limitations.

---------------------------------------------

(h) Improving with Exploration–Exploitation Trade-off  
We introduced \*\*ε-greedy exploration\*\* to balance exploration and exploitation:  
- With probability \*\*ε\*\*, take a random action (explore)  
- Otherwise, take the best known action (exploit)

We decayed ε over time:  
```python  
epsilon = max(epsilon\_min, epsilon \* epsilon\_decay)  
```  
This allowed the agent to explore early and refine policies later.

---------------------------------------------

(i) Final Evaluation and Comparison  
We compared the original and improved Q-learning agents over 10,000 episodes:

Original Q-learning policy:  
Success rate: 0.0000  
Avg reward: 0.0000  
Avg steps: 17.87

Improved Q-learning policy:  
Success rate: 0.4088  
Avg reward: 0.4088  
Avg steps: 28.93  
```

\*\*Key Takeaways:\*\*  
- Success rate improved by \*\*+40.88%\*\*  
- Reward improvement aligned with goal-reaching behavior  
- Steps increased as the agent explored better paths

---------------------------------------------

4. Conclusion

This project demonstrates the effectiveness of Q-learning and the necessity of exploration in reinforcement learning. The ε-greedy strategy enabled the agent to learn optimal policies and substantially improved success rates in a stochastic environment.

Future enhancements could include:  
- Function approximation (e.g., Deep Q-Networks)  
- Reward shaping  
- Larger environments (8x8 Frozen Lake)  
- Prioritized experience replay

---------------------------------------------

5. References  
- OpenAI Gymnasium Documentation  
- Sutton, R. S., & Barto, A. G. (2018). \*Reinforcement Learning: An Introduction\*  
- FrozenLake environment specs: https://www.gymlibrary.dev/environments/toy\_text/frozen\_lake/

---------------------------------------------

End of Report