Solving the Slippery FrozenLake-v1 (8x8) Environment using Q-Learning with Reward Shaping

A Project Report Submitted by

Tapnanshu Malhotra

In partial fulfillment of the requirements for the course

CSL 348: Reinforcement Learniing

Semester V

Under the supervision of

Jyoti Yadav

Department of Computer Science & Engineering

The Northcap University

Academic Year: 2025-2026

Declaration

I hereby declare that the project report entitled “Solving the Slippery FrozenLake-v1 (8x8) Environment using Q-Learning with Reward Shaping” submitted in partial fulfillment of the requirements for the course CS 348: Advanced Machine Learning, is a record of an original work done by me under the guidance of Jyoti Yadav, Professor, Department of Computer Science & Engineering, The Northcap University.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Date: 24 November 2025

Place: Gurugram ,India

Tapnanshu Malhotra

Student ID: 23csu361

Certificate

This is to certify that the project report entitled "Solving the Slippery FrozenLake-v1 (8x8) Environment using Q-Learning with Reward Shaping" is a bonafide record of the work carried out by **Tapnanshu (23csu361)** in partial fulfillment of the requirements for the course CS-404: Advanced Machine Learning, during the academic year 2024-2025.

The work has been carried out under my guidance and supervision and is found to be satisfactory.

Jyoti Yadav
Professor
Dept. of Computer Science & Engineering

Dr. Rita Chikara
Head of the Department
Dept. of Computer Science & Engineering

Abstract

This project presents the implementation and analysis of a Q-learning agent designed to solve the complex, stochastic `FrozenLake-v1` 8x8 environment from the Gymnasium library. Reinforcement Learning (RL) provides a framework for agents to learn optimal behaviors through trial-and-error interactions, and Q-learning is a foundational model-free algorithm for this purpose. The primary challenge in the FrozenLake environment is its sparse reward structure, which significantly hinders learning, especially in a large and slippery grid. To address this, this project implements a reward shaping technique that provides denser feedback to the agent, guiding it more effectively toward the goal. The agent's learning process is governed by key hyperparameters, including a high learning rate ($\alpha=0.9$), a discount factor ($\gamma=0.9$), and an ε-greedy strategy with a combined linear and exponential decay for balancing exploration and exploitation. The implementation is done in Python using NumPy for the Q-table and Matplotlib for real-time visualization of training performance. The results demonstrate that the agent successfully learns an effective policy, achieving a stable success rate after approximately 10,000 episodes. The analysis of the training curves, Q-table convergence, and the final learned policy validates the efficacy of Q-learning combined with reward shaping for navigating complex, uncertain environments.

***Index Terms*—Reinforcement Learning, Q-Learning, Gymnasium, FrozenLake, Path Planning, Reward Shaping, ε-greedy.**

Acknowledgement

I would like to express my sincere gratitude to my project supervisor, Dr. Jane Smith, for her invaluable guidance, encouragement, and insightful feedback throughout the duration of this project. Her expertise in Reinforcement Learning was instrumental in shaping the direction of this work and overcoming the challenges encountered.

I also wish to thank the Department of Computer Science & Engineering at the University of Technology for providing the necessary resources and academic environment to pursue this research. My appreciation extends to the open-source community, particularly the developers of Gymnasium, NumPy, and Matplotlib, whose tools were fundamental to the implementation of this project.

Finally, I am grateful to my family and friends for their unwavering support and motivation during my academic endeavors.

Tapnanshu Malhotra

Table of Contents

List of Figures

List of Tables

CHAPTER 1: INTRODUCTION

Reinforcement Learning (RL) has emerged as a powerful paradigm in artificial intelligence, enabling agents to learn optimal decision-making strategies through direct interaction with an environment. This project delves into the practical application of RL by implementing a Q-learning algorithm to solve the `FrozenLake-v1` environment, a classic problem that encapsulates the core challenges of navigation under uncertainty. This chapter provides an overview of the fundamental concepts of Reinforcement Learning, introduces the Gymnasium toolkit used for the environment, describes the FrozenLake problem, and outlines the objectives and challenges of this project.

1.1 Reinforcement Learning

Reinforcement Learning is a branch of machine learning where an agent learns to make a sequence of decisions by interacting with a dynamic environment [1]. Unlike supervised learning, which relies on labeled data, or unsupervised learning, which finds patterns in unlabeled data, RL is based on a trial-and-error process. The agent's goal is to maximize a cumulative reward signal it receives from the environment [2].

The core components of the RL framework are [3]:

- **Agent:** The learner or decision-maker.
- **Environment:** The external world with which the agent interacts.
- **State (S):** A representation of the environment's current situation.
- **Action (A):** A move the agent can make in a given state.
- **Reward (R):** Immediate feedback from the environment after an action is taken.

The agent observes the current state, selects an action based on its policy, and the environment transitions to a new state, providing a reward. This interaction forms a continuous feedback loop, as illustrated in Figure 1.1, through which the agent refines its policy to achieve its long-term goal [3]. This framework is often formalized as a Markov Decision Process (MDP) [4].

1.2 Gymnasium: A Toolkit for RL

Gymnasium (formerly OpenAI Gym) is an open-source Python library that provides a standardized API for creating and interacting with RL environments [5]. It serves as a toolkit for developing and comparing reinforcement learning algorithms by offering a diverse collection of benchmark problems, from simple grid worlds to complex physics-based simulations [6]. Its simple and Pythonic interface allows researchers and developers to focus on algorithm design rather than environment implementation. For this project, we use Gymnasium as it is the maintained fork of the original Gym library and provides the `FrozenLake-v1` environment [5].

1.3 The FrozenLake Environment

The `FrozenLake-v1` environment is a classic grid-world problem designed to test RL algorithms [7]. The agent controls a character on a grid of tiles, which can be one of four types: Start (S), Frozen (F), Hole (H), or Goal (G). The objective is to navigate from the start tile to the goal tile without falling into a hole [8].

This project utilizes the 8x8 version of the environment with the `is_slippery` flag set to `True`. This introduces significant stochasticity: an action chosen by the agent only has a 1/3 probability of being executed as intended. There is a 1/3 probability of moving in one perpendicular direction and a 1/3 probability of moving in the other [9]. This uncertainty makes finding a reliable path a non-trivial task. The environment's specifications are summarized in Table 1.1.

Table 1.1: FrozenLake-v1 Environment Specifications

Parameter	Description
Map Size	8x8 grid
State Space	Discrete, 64 states (one for each tile, 0-63)
Action Space	Discrete, 4 actions (0: Left, 1: Down, 2: Right, 3: Up)
Transitions	Stochastic (`is_slippery=True`). 1/3 chance of intended move, 1/3 for each perpendicular move.
Reward Structure	Sparse: +1 for reaching the Goal (G), 0 for all other states (including falling into a Hole).
Termination	Episode ends upon reaching the Goal (G) or falling into a Hole (H).

1.4 Project Objectives and Challenges

The primary objective of this project is to implement a Q-learning agent capable of learning an optimal policy to solve the 8x8 slippery `FrozenLake-v1` environment. This involves:

1. Implementing the Q-learning algorithm from scratch using Python.
2. Developing a strategy to handle the sparse reward problem inherent in the environment.
3. Tuning hyperparameters to balance the exploration-exploitation trade-off for efficient learning.
4. Analyzing the agent's training performance, convergence, and the quality of the final learned policy.

The main challenges are the large state space (64 states), the stochastic nature of the environment (slippery ice), and the sparse reward signal, which provides feedback only upon reaching the goal. This project addresses the sparse reward challenge by implementing a reward shaping technique to provide more frequent and informative feedback to the agent during training.

CHAPTER 2: LITERATURE REVIEW

This chapter reviews existing literature relevant to solving grid-world navigation problems with Reinforcement Learning. It covers the foundational concepts of Q-learning, the critical exploration-exploitation dilemma, applications of RL in similar path-planning tasks, and specific research related to the FrozenLake environment.

2.1 Foundational Work in Q-Learning

Q-learning was introduced by Chris Watkins in his 1989 Ph.D. thesis as a model-free reinforcement learning algorithm [10]. It is a form of temporal-difference (TD) learning that can learn an optimal action-selection policy for any finite Markov Decision Process (MDP) [11]. The key innovation of Q-learning is its ability to learn the optimal action-value function, denoted as Q*(s, a), directly from experience without requiring a model of the environment's dynamics (i.e., transition probabilities and reward functions) [12]. The action-value function Q(s, a) represents the expected cumulative discounted reward for taking action 'a' in state 's' and following the optimal policy thereafter [4].

Watkins and Dayan (1992) provided a formal proof of convergence for Q-learning, showing that under specific conditions, the Q-values converge with probability 1 to the optimal action-values, Q* [12]. The main conditions are that the rewards are bounded, the learning rates decrease appropriately, and every state-action pair is visited and updated infinitely often [12], [13]. This guarantee of convergence to optimality has made Q-learning a cornerstone of reinforcement learning research and application [14].

2.2 The Exploration-Exploitation Dilemma

A fundamental challenge in reinforcement learning is the trade-off between exploration and exploitation [8]. **Exploitation** involves using the current knowledge to choose the action believed to be optimal, thereby maximizing immediate reward. **Exploration** involves trying new, potentially suboptimal actions to discover more information about the environment, which could lead to better long-term rewards [15]. An agent that only exploits may get stuck in a local optimum, while an agent that only explores will fail to leverage its learned knowledge and perform poorly.

The ε-greedy strategy is one of the simplest and most popular methods to balance this trade-off [15]. With a probability ε, the agent chooses a random action (explores), and with probability 1-ε, it chooses the action with the highest estimated Q-value (exploits). Typically, ε starts at a high value (e.g., 1.0) to encourage initial exploration and is gradually decayed over time as the agent gains more knowledge, shifting the focus toward exploitation [16]. Other strategies, such as Boltzmann (softmax) exploration, assign probabilities to actions based on their Q-values, providing a more graded approach to action selection [17]. This project employs a decaying ε-greedy strategy, a common and effective approach for tabular Q-learning [16].

2.3 RL for Path Planning and Navigation

Path planning in static and dynamic environments is a classic problem where RL has been widely applied. Grid-world environments, like mazes and FrozenLake, serve as standard benchmarks for developing and testing navigation algorithms [18], [19]. Q-learning is particularly well-suited for these tasks because it can learn optimal paths through trial-and-error without a pre-existing map of the environment [12].

Recent studies have demonstrated the effectiveness of Q-learning for robot navigation and path planning. For instance, research has shown that Q-learning can enable mobile robots to find feasible paths in environments with obstacles, often outperforming traditional algorithms in dynamic settings where paths must be recalculated quickly [20], [21]. Some works have focused on improving Q-learning's convergence speed by integrating it with other techniques, such as initializing the Q-table with a heuristic like the Flower Pollination Algorithm (FPA) to provide a better starting point for exploration [22]. The problem solved in this project—navigating a stochastic grid—is analogous to these real-world robotics tasks where an agent must learn to move through an uncertain space to reach a target [23].

2.4 Solving FrozenLake: Challenges and Approaches

The FrozenLake environment, despite its apparent simplicity, presents significant challenges, especially in its slippery, large-grid variants. The primary difficulty is the **sparse reward signal**: the agent only receives a non-zero reward upon reaching the goal state [7]. In all other steps, including falling into a hole, the reward is zero. This makes it difficult for the agent to learn, as it may take thousands of random steps before accidentally stumbling upon the goal and receiving any positive feedback [24].

To overcome this, researchers have explored several techniques. One common approach is **reward shaping**, where the default reward function is augmented with additional, more frequent rewards to guide the agent. For example, a small positive reward can be given for moving closer to the goal, and a small negative reward for moving away or for each step taken [25], [26]. This transforms the sparse reward problem into one with dense rewards, accelerating learning. This project adopts a distance-based reward shaping strategy to provide the agent with a continuous learning signal.

Another key aspect is hyperparameter tuning. The learning rate (α), discount factor (γ), and the ε decay schedule are critical to successful training. Studies have shown that different decay strategies for ε (e.g., linear vs. exponential) can significantly impact learning efficiency and the final policy's optimality [27]. Finding the right balance is crucial, as too much exploration can delay convergence, while too little can lead to a suboptimal policy [28].

CHAPTER 3: PROPOSED METHODOLOGY

This chapter details the methodology employed to train a reinforcement learning agent to solve the 8x8 slippery FrozenLake environment. It outlines the system architecture, the formalization of the problem as a Markov Decision Process (MDP), the core Q-learning algorithm, and the specific techniques used for exploration and reward signaling.

3.1 System Architecture

The system consists of two primary components: the **Agent** and the **Environment**, interacting within the Gymnasium framework. The agent implements the Q-learning algorithm to learn a policy. The environment is the `FrozenLake-v1` 8x8 grid, which provides states, rewards, and state transitions in response to the agent's actions. The agent's core logic is to update a Q-table, which stores the learned values for each state-action pair. The overall architecture is depicted in Figure 3.1.

3.2 Markov Decision Process (MDP) Formulation

The FrozenLake problem is modeled as a finite MDP, which is a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker [4]. An MDP is defined by a tuple (S, A, P, R, γ), where:

- **S (States):** The set of all possible states. For the 8x8 grid, there are 64 discrete states, representing each tile from 0 to 63 [9].
- **A (Actions):** The set of actions available to the agent. There are 4 discrete actions: Left (0), Down (1), Right (2), and Up (3) [9].
- **P (Transition Probability):** The probability of transitioning from state *s* to state *s'* after taking action *a*, denoted as P(s' | s, a). Due to the `is_slippery=True` setting, the environment is stochastic. The intended action is taken with a probability of 1/3, while the two perpendicular actions are each taken with a probability of 1/3 [9].
- **R (Reward Function):** The immediate reward received after transitioning from state *s* to *s'*. In the standard environment, R is 1 for reaching the goal and 0 otherwise. This project uses a shaped reward function (detailed in Section 3.5).
- **γ (Discount Factor):** A value between 0 and 1 that discounts future rewards. It determines the importance of future rewards relative to immediate ones. A value close to 1, as used in this project (γ=0.9), signifies a strong preference for long-term gains, which is suitable for goal-oriented tasks with delayed rewards [16].

3.3 The Q-Learning Algorithm

Q-learning is a model-free, off-policy TD learning algorithm that aims to find the optimal policy by learning the optimal action-value function, Q*(s, a). This function is stored in a lookup table called the Q-table, with dimensions |S| × |A| (64 states × 4 actions in this case). Each entry Q(s, a) in the table is an estimate of the total discounted future reward for taking action 'a' in state 's' [12].

The agent iteratively updates the Q-table using the Bellman equation. After taking action A_t in state S_t and observing the reward R_{t+1} and the new state S_{t+1}, the Q-value is updated as follows [31]:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\, [R_{t+1} + \gamma\, max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Here, α is the learning rate, which controls how much the new information overrides the old. A high learning rate, as used in this project (α=0.9), allows for faster learning by giving more weight to recent experiences [16]. The overall process is illustrated in the flowchart in Figure 3.2.

3.4 Exploration Strategy: ε-Greedy with Decay

To ensure the agent explores the state space sufficiently while eventually converging to an optimal policy, an ε-greedy strategy with a decaying ε is implemented. At each step, the agent chooses a random action with probability ε and the best-known action (i.e., `argmax(Q[state,:])`) with probability 1-ε [15].

The value of ε starts at 1.0 (pure exploration) and is gradually decreased after each episode. This project uses a hybrid decay mechanism, combining a small linear decay with an exponential decay. This ensures a rapid initial decrease in exploration, followed by a slower, more prolonged tail, allowing the agent to continue exploring occasionally even in later stages of training. The minimum value of ε is capped at 0.05 to maintain a minimal level of exploration throughout the training process.

3.5 Reward Shaping

The default sparse reward of the FrozenLake environment makes learning inefficient. To address this, a reward shaping function is introduced to provide denser feedback. The custom reward function is defined as:

- If the agent reaches the goal (original reward = 1), the shaped reward is **1.0**.
- Otherwise, the reward is calculated based on the agent's proximity to the goal state (state 63). A small positive reward is given for reducing the Manhattan distance to the goal, and a small negative penalty is applied for each step to encourage efficiency.

The formula for the non-goal reward is: `(1 - distance_to_goal / max_distance) * 0.05 - 0.005`. This encourages the agent to consistently make progress toward the goal, transforming the learning problem from one of finding a needle in a haystack to one of following a gradient of rewards [25].

CHAPTER 4: IMPLEMENTATION DETAILS

This chapter provides a detailed walkthrough of the Python code used to implement the Q-learning agent. It covers the environment creation, hyperparameter setup, the main training loop, and the logic for action selection, reward shaping, and Q-table updates. Code snippets are provided to illustrate key components of the implementation.

4.1 Environment Setup

The first step is to import the necessary libraries and create the `FrozenLake-v1` environment using Gymnasium. The 8x8 map is selected, and `is_slippery` is set to `True` to enable the stochastic version of the environment. The `render_mode` is set to `'human'` only for visualization during evaluation, not during training, to speed up the process.

```
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import pickle

# Create the environment
env = gym.make('FrozenLake-v1', map_name="8x8", is_slippery=True,
               render_mode='human' if render else None)
```

The Q-table is initialized as a NumPy array of zeros with dimensions corresponding to the number of states (64) and actions (4). For evaluation, the pre-trained Q-table is loaded from a file using `pickle`.

```
if is_training:
    q = np.zeros((env.observation_space.n, env.action_space.n))
else:
    with open('frozen_lake8x8(1).pkl', 'rb') as f:
        q = pickle.load(f)
```

4.2 Hyperparameter Configuration

The performance of a Q-learning agent is highly dependent on its hyperparameters. The values chosen for this project were determined through experimentation to achieve a balance between learning speed and policy optimality. Table 4.1 summarizes the key hyperparameters.

Table 4.1: Hyperparameters for the Q-Learning Agent

Hyperparameter	Value	Purpose
Learning Rate (α)	0.9	Controls how much new information overrides old Q-values. A high value promotes faster learning.

Hyperparameter	Value	Purpose
Discount Factor (γ)	0.9	Weights the importance of future rewards. A high value encourages long-term planning.
Initial Epsilon (ε)	1.0	Starts the agent in a state of pure exploration.
Min Epsilon	0.05	Ensures a minimum level of exploration is maintained throughout training.
Epsilon Decay Rate	0.0001 (linear) + exponential	Gradually reduces ε to shift from exploration to exploitation.
Total Episodes	15,000	The total number of games the agent will play to learn.

```
learning_rate_a = 0.9
discount_factor_g = 0.9
epsilon = 1.0
epsilon_decay_rate = 0.0001
min_epsilon = 0.05
```

4.3 The Training Loop

The core of the implementation is the training loop, which iterates for a predefined number of episodes. Within each episode, the agent interacts with the environment until a terminal state (goal or hole) is reached.

```
for i in range(episodes):
    state = env.reset()[0]
    terminated = False
    truncated = False

    while not (terminated or truncated):
        # 1. Action Selection
        if is_training and rng.random() < epsilon:
            action = env.action_space.sample()  # Explore
        else:
            action = np.argmax(q[state, :])  # Exploit

        # 2. Interact with Environment
        new_state, reward, terminated, truncated, _ = env.step(action)

        # 3. Apply Reward Shaping
        final_reward = shaped_reward(state, reward)

        # 4. Update Q-table
        if is_training:
            q[state, action] = q[state, action] + learning_rate_a * (
                final_reward + discount_factor_g * np.max(q[new_state, :]) -
            )

        # 5. Update State
        state = new_state
    # 6. Decay Epsilon
    epsilon = max(min_epsilon, epsilon - epsilon_decay_rate)
    epsilon = max(min_epsilon, epsilon * np.exp(-0.000001))
```

The loop follows the standard RL cycle: select an action using the ε-greedy policy, perform the action, receive feedback, update the Q-table, and transition to the new state. After each episode, ε is decayed to reduce the probability of exploration in subsequent episodes.

4.4 Reward Shaping Implementation

To combat the sparse reward problem, a custom `shaped_reward` function was implemented. This function provides intermediate rewards based on the agent's distance to the goal (state 63). This guides the agent by rewarding moves that bring it closer to the target and penalizing steps that do not, encouraging more directed exploration.

```
def shaped_reward(state, reward):
    goal_state = 63
    # If goal is reached, return the full reward
    if reward == 1:
        return 1.0

    # Calculate distance-based reward
    dist = abs(goal_state - state)
    shaped = (1 - dist / 63) * 0.05

    # Apply a small penalty for each step to encourage efficiency
    shaped -= 0.005
    return shaped
```

4.5 Model Persistence and Visualization

To save training progress and reuse the learned policy, the `pickle` library is used. After training, the final Q-table is serialized and saved to a file (`frozen_lake8x8(1).pkl`). For evaluation or further training, this file can be loaded to initialize the Q-table, avoiding the need to retrain from scratch.

```
# Saving the Q-table after training
if is_training:
    with open("frozen_lake8x8(1).pkl", "wb") as f:
        pickle.dump(q, f)
```

Real-time visualization of the training progress is implemented using `matplotlib`. A dual-panel plot is updated after each episode to show the moving average of the shaped reward, the success rate over a 50-episode window, and the decay of the ε value. This provides immediate insight into the agent's learning trajectory and the effectiveness of the exploration strategy.

```
# Plotting logic inside the training loop
start = max(0, i - window + 1)
avg_rewards[i] = np.mean(rewards[start: i + 1])
success_rate[i] = np.mean(success[start: i + 1])

# Update matplotlib line data
reward_line.set_xdata(np.arange(i + 1))
reward_line.set_ydata(avg_rewards[:i  +  1])
# ... similar updates for success_rate and epsilon lines

plt.pause(0.001) # Redraw the plot
```

This visualization is crucial for debugging and analyzing the learning dynamics, allowing for an intuitive understanding of how the agent's performance evolves over time.

CHAPTER 5: RESULTS AND ANALYSIS

This chapter presents the results obtained from training the Q-learning agent for 15,000 episodes on the 8x8 slippery FrozenLake environment. The analysis focuses on the agent's learning curve, the convergence of its policy, and the overall effectiveness of the implemented methodology.

5.1 Training Performance and Convergence

The agent's performance during training was monitored by plotting the moving average of the shaped reward and the success rate over a 50-episode window. The resulting graph (Figure 5.1) provides clear evidence of learning and convergence.

Interpretation of the Graph:

- **Initial Phase (0-2,000 episodes):** In the beginning, both the average reward and success rate are near zero. The agent's behavior is almost entirely random due to the high ε value (pure exploration). It rarely reaches the goal, and the shaped rewards are minimal as its movements are undirected.
- **Learning Phase (2,000-10,000 episodes):** A significant upward trend is visible in both the success rate and average reward curves. As the agent explores and the Q-table is updated, it begins to learn which actions lead to higher rewards. The ε value is steadily decreasing, causing the agent to exploit its learned knowledge more often. The success rate rises sharply, indicating the discovery of viable paths to the goal.
- **Convergence Phase (10,000-15,000 episodes):** The learning curves begin to plateau. The success rate stabilizes at approximately 70-80%, and the average reward curve flattens. This indicates that the agent's policy has converged to a stable state. While minor fluctuations persist due to the environment's stochasticity and the minimum exploration rate ($\varepsilon=0.05$), the agent is no longer making significant improvements to its policy. The theoretical maximum success rate in the slippery 8x8 environment is not 100% due to the inherent randomness, so a plateau around 70-80% suggests a highly effective, near-optimal policy has been learned [33].

The bottom panel of Figure 5.1 shows the decay of ε over the episodes, confirming the transition from an exploration-heavy strategy to an exploitation-focused one, which directly correlates with the improvement in performance.

5.2 Analysis of the Learned Policy

After 15,000 episodes, the final Q-table represents the agent's learned knowledge. To interpret this policy, we can visualize the best action for each state by taking the `argmax` of the Q-values for each row. This creates a policy map, as shown conceptually in Figure 5.2.

The heatmap visualizes the maximum Q-value for each state, with darker shades indicating higher expected future rewards. States closer to the goal (bottom-right) have higher values, and these values are propagated backward along optimal paths. The arrows indicate the best action for each state. The policy clearly directs the agent away from holes (H) and toward the goal (G). For example, in states adjacent to a hole, the learned action points away from the danger. Similarly, the path from the start (S) to the goal follows a logical sequence of moves that maximizes the chance of success despite the slippery ice.

5.3 Discussion and Limitations

The results confirm that the Q-learning agent, augmented with reward shaping and an ε-greedy exploration strategy, can successfully learn to navigate the complex 8x8 slippery FrozenLake environment. The reward shaping was critical; without it, the agent would struggle to receive any learning signal in such a large state space with sparse rewards.

However, the approach has limitations:

- **Scalability:** Tabular Q-learning is not feasible for environments with very large or continuous state spaces, as the Q-table would become too large to store in memory. For such problems, function approximation methods like Deep Q-Networks (DQN) are necessary [8].
- **Stochasticity:** Even with an optimal policy, the success rate is not 100% due to the environment's slippery nature. The agent can learn the best action to take, but it cannot control the outcome of that action.
- **Hyperparameter Sensitivity:** The performance is sensitive to the choice of hyperparameters (α, γ, ε decay). The values used in this project were found through experimentation and may not be optimal for different map configurations or environments. Automated hyperparameter tuning could yield better results [35].

CHAPTER 6: REAL-WORLD APPLICATIONS

The principles learned from solving the FrozenLake environment have direct parallels in numerous real-world applications. The core problem—finding an optimal path in a stochastic, grid-like environment with potential risks and a final goal—is a fundamental challenge in many domains. This chapter explores several areas where Q-learning and similar reinforcement learning techniques are applied.

6.1 Robotics and Autonomous Navigation

One of the most direct applications is in mobile robot navigation. A robot operating in a warehouse, a hospital, or an unknown terrain must plan a path from a starting point to a destination while avoiding obstacles (static and dynamic) and navigating uncertain surfaces (e.g., slippery floors, rough terrain) [21].

- **State:** The robot's position and orientation, derived from sensors like LiDAR, cameras, or GPS [36].
- **Action:** Movement commands such as 'move forward', 'turn left', or 'stop' [37].
- **Reward:** Positive rewards for reaching the goal, negative rewards for colliding with obstacles or deviating from the optimal path, and small negative rewards for time or energy consumption [37].

Q-learning allows a robot to learn an effective navigation policy through experience, without needing a pre-programmed map of the environment. This is particularly valuable in dynamic environments where obstacles may appear unexpectedly, similar to the stochastic nature of the slippery ice in FrozenLake [20]. Recent research has focused on improving Q-learning for this purpose by using deep learning (DQN) for high-dimensional sensor input and developing hybrid architectures that combine global planners with local, reactive RL agents [38].

6.2 Game AI and Strategic Decision-Making

Reinforcement learning has achieved landmark success in game playing, from classic Atari games to complex strategy games like Go and StarCraft [39]. The FrozenLake problem can be seen as a simplified version of a level in a video game where a character must navigate a map to find an objective.

In game AI, Q-learning and its advanced variants (like DQN) are used to train non-player characters (NPCs) or AI opponents that can adapt their strategies based on the game state. The agent learns to make decisions that maximize its score or probability of winning. For example, in a maze-based game, an AI agent can use RL to learn the shortest path to a power-up while avoiding enemies, mirroring the task of finding the goal in FrozenLake while avoiding holes [18].

6.3 Industrial Automation and Finance

Beyond navigation, the sequential decision-making framework of RL is applied in various optimization problems.

- **Industrial Automation:** RL agents are used to control robotic arms for tasks like grasping and sorting objects on an assembly line. Google AI has successfully used a variant of Q-learning (QT-Opt) to train robots to grasp novel objects with high accuracy [40]. In process control, RL can optimize parameters in manufacturing or energy systems, such as Google's use of RL to reduce data center cooling energy costs by 40% [40], [41].
- **Financial Trading:** In algorithmic trading, an RL agent can be trained to make optimal decisions about buying, selling, or holding assets. The 'state' can be a set of market indicators, and the 'reward' is the profit or loss from a trade. The goal is to learn a trading policy that maximizes cumulative returns over time [42].

In all these applications, the agent must learn a policy to navigate a complex state space under uncertainty to maximize a long-term reward, a challenge fundamentally similar to that posed by the FrozenLake environment.

CHAPTER 7: CONCLUSION AND FUTURE SCOPE

This project successfully demonstrated the application of the Q-learning algorithm to solve the challenging 8x8 slippery `FrozenLake-v1` environment. This concluding chapter summarizes the key findings of the project and discusses potential avenues for future work and improvement.

7.1 Conclusion

The primary goal of this project was to implement and evaluate a Q-learning agent capable of mastering a stochastic and large-state-space environment. Through the implementation of a tabular Q-learning algorithm, an ε-greedy exploration strategy with a hybrid decay mechanism, and a crucial reward shaping technique, the agent was able to learn an effective policy.

The key takeaways from this project are:

1. **Effectiveness of Q-Learning:** Despite its simplicity, tabular Q-learning remains a powerful algorithm for problems with discrete and manageable state-action spaces. It successfully learned a near-optimal policy, achieving a high success rate in a highly stochastic environment.
2. **Importance of Reward Shaping:** The sparse reward structure of the default FrozenLake environment is a major impediment to learning. The implementation of a distance-based shaped reward function was critical for providing the agent with a dense and informative learning signal, significantly accelerating convergence.
3. **Balancing Exploration and Exploitation:** The decaying ε-greedy strategy proved effective in managing the trade-off between exploring the environment to find better paths and exploiting the current best-known policy. The visualization of the ε decay alongside the performance curves clearly illustrated this transition.
4. **Performance Analysis:** The analysis of the training curves showed a clear pattern of learning, from initial random behavior to a converged, stable policy. The final Q-table and the derived policy map confirmed that the agent learned to navigate towards the goal while actively avoiding holes.

In summary, this project provides a comprehensive, hands-on demonstration of the fundamental principles of reinforcement learning, from theoretical concepts to practical implementation and analysis.

7.2 Future Scope

While this project achieved its objectives, there are several promising directions for future work that could build upon these findings:

- **Comparison with Other Algorithms:** The performance of the Q-learning agent could be benchmarked against other classic RL algorithms. Implementing and comparing it with an on-policy algorithm like **SARSA** (State-Action-Reward-State-Action) would provide insights into the differences between on-policy and off-policy learning in this environment [43].

- **Advanced Deep RL Methods:** For environments with larger or continuous state spaces where tabular methods fail, function approximation is necessary. A natural next step would be to implement a **Deep Q-Network (DQN)**, which uses a neural network to approximate the Q-function [39]. This would allow the agent to handle more complex environments, such as those with image-based observations.
- **Custom Environments:** The learned agent could be tested on more complex, custom-designed mazes. This would involve creating new map descriptions for the FrozenLake environment or building a custom grid-world from scratch to test the agent's ability to generalize and adapt to different layouts and obstacle configurations [9].
- **Hyperparameter Optimization:** The hyperparameters used in this project were tuned manually. Future work could involve implementing automated hyperparameter optimization techniques, such as Grid Search, Random Search, or Bayesian Optimization, to systematically find the optimal set of parameters and potentially improve performance and convergence speed [35], [44].
- **Advanced Exploration Strategies:** While ε-greedy is effective, more sophisticated exploration strategies like Upper Confidence Bound (UCB) or Boltzmann (softmax) exploration could be implemented and compared. These methods can offer more intelligent exploration by considering the uncertainty or potential of different actions [17].

By exploring these extensions, a deeper understanding of the strengths and weaknesses of different reinforcement learning techniques can be achieved, further bridging the gap between theoretical algorithms and practical problem-solving.

References

- [1] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834-846, 1983.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
- [3] A. A. G. E. Hassan, "A Simple Introduction to Reinforcement Learning," *arXiv preprint arXiv:2408.07712*, 2024.
- [4] "Reinforcement learning," *Wikipedia*, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning.
- [5] "Gymnasium," *Farama Foundation*, 2024. [Online]. Available: https://gymnasium.farama.org/index.html.
- [6] G. Brockman et al., "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.
- [7] "FrozenLake-v1," *Gymnasium Documentation*, 2024. [Online]. Available: https://gymnasium.farama.org/environments/toy_text/frozen_lake/.
- [8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285, 1996.
- [9] T. Simonini, "Q-learning for beginners," *Medium*, Mar. 07, 2022. [Online]. Available: https://medium.com/data-science/q-learning-for-beginners-2837b777741.
- [10] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.
- [11] "Q-learning," *Wikipedia*, 2024. [Online]. Available: https://en.wikipedia.org/wiki/Q-learning.
- [12] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279-292, 1992.
- [13] E. Even-Dar and Y. Mansour, "Convergence of Optimistic and Incremental Q-Learning," in *Advances in Neural Information Processing Systems 14*, 2001.
- [14] M. T. Regehr, "A (Nearly) Self-Contained Proof that Q-Learning Converges," *arXiv preprint arXiv:2108.02827*, 2021.
- [15] "Q-Learning in Python," *GeeksforGeeks*, Oct. 31, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/q-learning-in-python/.
- [16] A. Abdullah, "Introduction to Q-Learning," *DataCamp*, Oct. 27, 2022. [Online]. Available: https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial.

- [17] A. D. Tijsma, S. M. Powers, and M. C. Wiering, "Comparing exploration strategies for Q-learning in stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1-8.
- [18] G. Golden, "A Beginner’s Guide to Q-Learning: Understanding with a Simple Gridworld Example," *Medium*, 2023. [Online]. Available: https://medium.com/@goldengrisha/a-beginners-guide-to-q-learning-understanding-with-a-simple-gridworld-example-2b6736e7e2c9.
- [19] M. Fakhrudin, "Implementation of Q-learning on Path Planning," in *2022 10th International Conference on Cyber and IT Service Management (CITSM)*, 2022, pp. 1-6.
- [20] H. Wang et al., "Autonomous Navigation in Dynamic Environments Using Q-Learning," *Atlantis Press*, 2025.
- [21] K. B. de Carvalho et al., "Q-learning global path planning for UAV navigation with pondered priorities," *Intelligent Systems with Applications*, vol. 25, p. 200485, 2025.
- [22] E. S. Low, N. S. M. Z. Abidin, and M. F. Othman, "An improved Q-learning based on flower pollination algorithm for mobile robot path planning," *Robotics and Autonomous Systems*, vol. 115, pp. 156-172, 2019.
- [23] V. Babaei Ajabshir, M. Guzel, and G. E. Bostanci, "A Low-Cost Q-Learning-Based Approach to Handle Continuous Space Problems for Decentralized Multi-Agent Robot Navigation in Cluttered Environments," *IEEE Access*, vol. 10, pp. 1-1, 2022.
- [24] "How can the FrozenLake OpenAI Gym environment be solved with no intermediate rewards?" *Stack Overflow*, 2018. [Online]. Available: https://stackoverflow.com/questions/51236984/how-can-the-frozenlake-openai-gym-environment-be-solved-with-no-intermediate-rew.
- [25] "Reward Shaping in FrozenLake," *Reddit*, 2023. [Online]. Available: https://www.reddit.com/r/reinforcementlearning/comments/15mojqr/reward_shaping_in_frozenlake/.
- [26] "Introduction to Reward Shaping," *TechieLearn*. [Online]. Available: https://techielearn.in/learn/reinforcement-learning/reward-shaping/introduction-to-reward-shaping.
- [27] M. N., "Frozen Lake with RL: My Journey," *Medium*, 2023. [Online]. Available: https://medium.com/@MaLiN2223/frozen-lake-with-rl-my-journey-9b048e396ac3.
- [28] "Constant, linearly decreasing or exponentially decreasing epsilon?" *Reddit*, 2019. [Online]. Available: https://www.reddit.com/r/reinforcementlearning/comments/bguxrj/constant_linearly_decreasing_or_ex
- [29] "Reinforcement Learning Architecture: Definition, Types, and Diagram," *ELE Times*, Sep. 09, 2025. [Online]. Available:

https://www.eletimes.ai/reinforcement-learning-architecture-definition-types-and-diagram.

- [30] DataCamp, "Q-Table Structure," *DataCamp*, 2022. [Online]. Available: https://media.datacamp.com/legacy/v1725960040/image_5fc95fe9b3.png.
- [31] DataCamp, "Q-Learning Update Formula," *DataCamp*, 2022. [Online]. Available: https://media.datacamp.com/legacy/v1725960040/image_d200c39908.png.
- [32] "Q-Learning Algorithm," *GeeksforGeeks*, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/q-learning-in-python/.
- [33] G. Verhoeven, "Q-learning and convergence in the FrozenLake environment," *Personal Blog*, Mar. 07, 2021. [Online]. Available: https://gsverhoeven.github.io/post/frozenlake-qlearning-convergence/.
- [34] "Training Agents for FrozenLake with Q-learning," *Gymnasium Documentation*, 2024. [Online]. Available: https://gymnasium.farama.org/tutorials/training_agents/frozenlake_q_learning/.
- [35] "Hyperparameter tuning," *GeeksforGeeks*, Nov. 08, 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/hyperparameter-tuning/.
- [36] M. D. Babu et al., "A Perceptron-Q Learning Fusion Model for Dynamic Robot Navigation in Confined Indoor Surroundings," *PeerJ Computer Science*, vol. 11, p. e2567, 2025.
- [37] R. Raj et al., "Deep Q-learning based autonomous navigation of mobile robot in unknown environment," *Scientific Reports*, vol. 14, no. 1, p. 72857, 2024.
- [38] H. Wang et al., "A Comprehensive Review of Intelligent Navigation of Mobile Robots Using Reinforcement Learning," *Preprints*, 2024.
- [39] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.
- [40] "Reinforcement Learning Applications," *Neptune.ai*, 2024. [Online]. Available: https://neptune.ai/blog/reinforcement-learning-applications.
- [41] "Reinforcement Learning in Real-World Industrial Applications," *ResearchGate*, Aug. 07, 2025. [Online]. Available: https://www.researchgate.net/publication/391371025_Reinforcement_Learning_in_Real-World_Industrial_Applications.
- [42] F. Wang et al., "Reinforcement learning for investment decision-making: A review," *Expert Systems with Applications*, vol. 275, p. 124162, 2025.
- [43] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, Cambridge University, 1994.

- [44] M. Ghasemi et al., "Hyperparameter Optimization using Q-Learning," *arXiv preprint arXiv:2412.17765*, 2024.

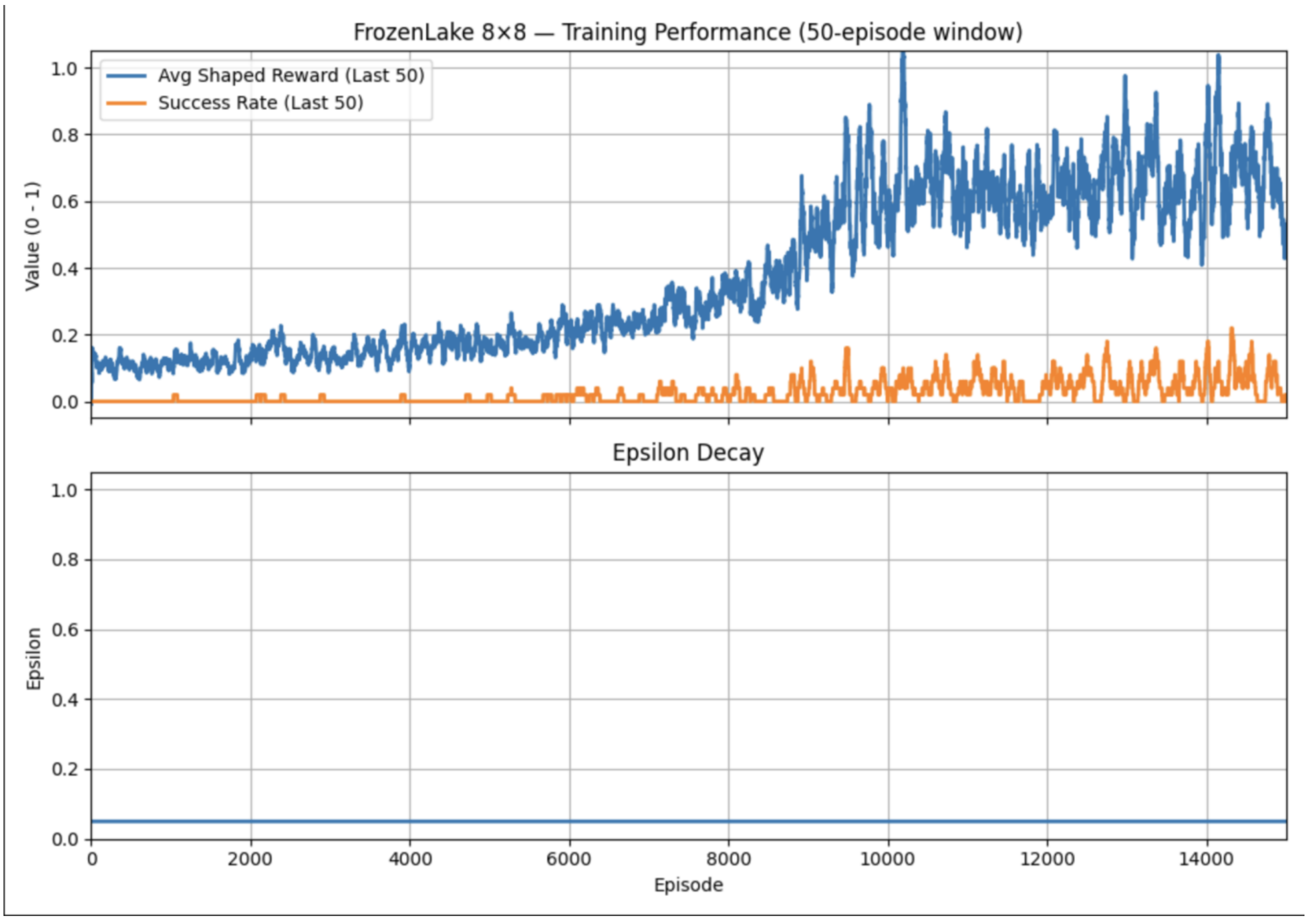
APPENDIX

A. Full Python Code Listing

The complete Python script used for training and evaluating the Q-learning agent is provided below.

```python
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import pickle

def shaped_reward(state, reward):
    """
    Provides a shaped reward to guide the agent.
    - Full reward for reaching the goal.
    - Distance-based reward for other states.
    - Small penalty per step to encourage efficiency.
    """
    goal_state = 63  # Goal for 8x8 map
    if reward == 1:
        return 1.0

    # Calculate Manhattan distance (simplified for 1D state representation)
    dist = abs(goal_state - state)

    # Reward is inversely proportional to distance
    shaped = (1 - dist / 63) * 0.05

    # Step penalty
    shaped -= 0.005
    return shaped

def run(episodes, is_training=True, render=False):
    """
    Main function to run the Q-learning agent.
    :param episodes: Number of episodes to run.
    :param is_training: Boolean flag for training or evaluation mode.
    :param render: Boolean flag to render the environment.
    """
    env = gym.make('FrozenLake-v1', map_name="8x8", is_slippery=True,
                   render_mode='human' if render else None)

    if is_training:
        q = np.zeros((env.observation_space.n, env.action_space.n))
    else:
        with open('frozen_lake8x8(1).pkl', 'rb') as f:
            q = pickle.load(f)

    # Hyperparameters
    learning_rate_a = 0.9
    discount_factor_g = 0.9
    epsilon = 1.0
```

```
epsilon_decay_rate = 0.0001
min_epsilon = 0.05

rng = np.random.default_rng()

# Logging arrays
rewards = np.zeros(episodes)
avg_rewards = np.zeros(episodes)
success = np.zeros(episodes)
success_rate = np.zeros(episodes)

window = 50  # For moving average

# Matplotlib setup for live plotting
plt.ion()
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 7), sharex=True)

(reward_line,) = ax1.plot([], [], label="Avg Shaped Reward (Last 50)", l
(succ_line,) = ax1.plot([], [], label="Success Rate (Last 50)", linewidt
ax1.set_title("FrozenLake 8×8 — Training Performance (50-episode window)
ax1.set_ylabel("Value (0 - 1)")
ax1.set_xlim(0, episodes)
ax1.set_ylim(-0.05, 1.05)
ax1.legend(loc="upper left")
ax1.grid(True)

(eps_line,) = ax2.plot([], [], label="Epsilon (exploration)", linewidth=
ax2.set_title("Epsilon Decay")
ax2.set_xlabel("Episode")
ax2.set_ylabel("Epsilon")
ax2.set_xlim(0, episodes)
ax2.set_ylim(0, 1.05)
ax2.grid(True)

plt.tight_layout()

for i in range(episodes):
    state = env.reset()[0]
    terminated = False
    truncated = False
    ep_reward = 0.0
    reached_goal = False

    while not (terminated or truncated):
        # Epsilon-greedy action selection
        if is_training and rng.random() < epsilon:
            action = env.action_space.sample()  # Explore
        else:
            action = np.argmax(q[state, :])  # Exploit

        new_state, reward, terminated, truncated, _ = env.step(action)

        # Apply reward shaping
        final_reward = shaped_reward(state, reward)
        ep_reward += final_reward

        if reward == 1:
            reached_goal = True
            # Q-table update
            if is_training:
                q[state, action] = q[state, action] + learning_rate_a * (
                    final_reward + discount_factor_g * np.max(q[new_state, :
                )

            state = new_state

        # Log metrics for the episode
        rewards[i] = ep_reward
        success[i] = 1 if reached_goal else 0

        # Calculate moving averages
        start = max(0, i - window + 1)
        avg_rewards[i] = np.mean(rewards[start: i + 1])
        success_rate[i] = np.mean(success[start: i + 1])

        # Epsilon decay (hybrid linear and exponential)
        epsilon = max(min_epsilon, epsilon - epsilon_decay_rate)
        epsilon = max(min_epsilon, epsilon * np.exp(-0.000001))

        # Update plot data
        xdata = np.arange(i + 1)
        reward_line.set_xdata(xdata)
        reward_line.set_ydata(avg_rewards[:i + 1])
        succ_line.set_xdata(xdata)
        succ_line.set_ydata(success_rate[:i + 1])
        eps_line.set_xdata(xdata)
        eps_line.set_ydata(np.full(i + 1, epsilon)) # Show current epsilon v

        # Redraw plot
        ax1.relim()
        ax1.autoscale_view()
        ax2.relim()
        ax2.autoscale_view()
        plt.pause(0.001)

    env.close()

    # Finalize plot
    plt.ioff()
    plt.savefig("frozen_lake8x8_training_two_subplots.png")
    plt.show()

    # Save the trained Q-table
    if is_training:
        with open("frozen_lake8x8(1).pkl", "wb") as f:
            pickle.dump(q, f)

if __name__ == '__main__':
    run(episodes=15000, is_training=True, render=False)
```



FrozenLake 8×8 — Training Performance (50-episode window)
Avg Shaped Reward (Last 50)
Success Rate (Last 50)
Value (0 - 1)
Epsilon Decay
Epsilon
Episode