Reinforcement Learning Homework3

Project 1: Grid World Wind Simulation Task
Project 2: Brick Breaking Game Task

Pang Liu

March 30, 2025

1 Introduction

This report documents the development and analysis of two distinct projects implemented in Python. The first project focuses on a grid world environment with stochastic wind dynamics, where multiple reinforcement learning algorithms (SARSA, Q-learning, and Monte Carlo) are employed to learn an optimal policy for reaching a designated goal. The second project is a discrete version of the classic Breakout (brick breaking) game, where a Q-learning agent is trained to control a paddle and clear bricks. The experiment details the problem setups, state-action spaces, reward settings, iterative update rules, implementation details, and experimental results for both tasks.

2 Grid World Wind Simulation Task

2.1 Problem Description and Environment Setup

In this task, a grid world environment with dimensions 10×10 is constructed. The environment is adapted from *Exercise 6.9* (Windy Gridworld with King's Moves) and *Exercise 6.10* (Stochastic Windy Gridworld) in *Reinforcement Learning: An Introduction* (Sutton & Barto). Specifically, the experiment combines:

- **King's Moves (Exercise 6.9):** The agent can move in 8 possible directions plus a "stay" action, allowing for diagonal movements.
- **Stochastic Wind (Exercise 6.10):** Each column has a base wind strength, which can increase by 1, remain unchanged, or decrease by 1 with equal probability.

The agent starts at position $(5, 3)$ and must reach the goal at $(5, 8)$ while contending with wind pushing it upward.

2.2 State and Action Space

2.2.1 State Space

The state is represented by the agent's position on the grid:

$$S = (r, c) \quad \text{with } r \in \{0, 1, \ldots, 9\},\ c \in \{0, 1, \ldots, 9\}.$$

2.2.2 Action Space

Due to the King's Moves, the action set consists of 9 possible moves:

$$\text{Actions} = \{(0,0),\ (-1,0),\ (-1,1),\ (0,1),\ (1,1),\ (1,0),\ (1,-1),\ (0,-1),\ (-1,-1)\},$$

where each tuple represents the change in row and column. After applying the action, the stochastic wind further adjusts the row value by pushing the agent upward.

2.3 Reward Setting

The reward is defined as:

$$r_t = \begin{cases} -1, & \text{if the agent moves without reaching the goal,} \\ 0, & \text{if the agent reaches the goal.} \end{cases}$$

Thus, the agent is penalized by -1 per step, which encourages the experiment to reach the goal in as few steps as possible.

2.4 Reinforcement Learning Algorithms and Iterative Updates

The implementation in `1_primary.py` uses three reinforcement learning methods:

- **SARSA:** The Q-value update is

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[r + \gamma Q(s',a') - Q(s,a) \Big],$$

  where (s,a) is the current state-action pair and (s',a') is the next state and action.

- **Q-learning:** The off-policy update rule is

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \Big],$$

  which uses the maximum Q-value in the next state.

- **Monte Carlo:** The Q-values are updated based on the averaged returns from complete episodes.

2.5 Experimental Testbed and Results

The experiment was carried out over multiple episodes (e.g., 500 episodes per run) and several runs (e.g., 3 runs) to ensure robustness of the performance metrics. The testbed includes a well-defined grid world, a wind simulation based on column-specific base wind with stochastic variations, and an implementation of the three RL algorithms. Figure 1 shows the learning curves (average reward and success rate vs. episodes) obtained from the experiment. Here, the orange curve represents Q-learning, the blue curve represents SARSA, and the green curve represents Monte Carlo.

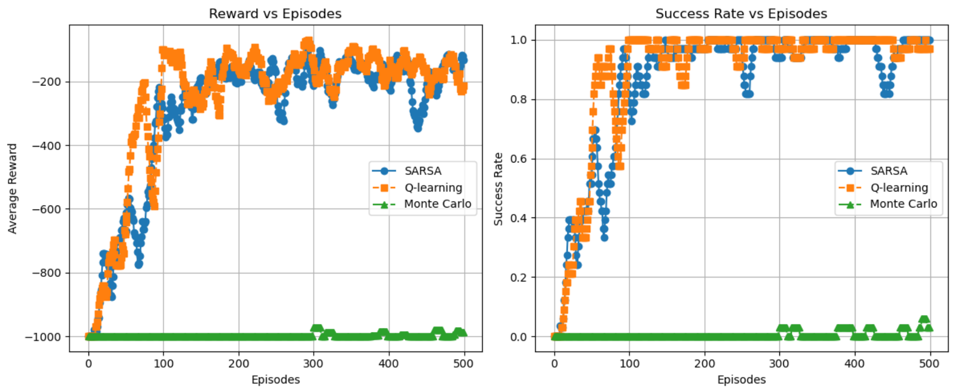

Figure 1: Learning curves for the grid world wind simulation task (from `1_primary.py`). Left: Average reward vs. episodes. Right: Success rate vs. episodes.

Analysis of the Curves:

- **Q-learning (orange):**
  The average reward increases rapidly, indicating that the experiment quickly learns to minimize step penalties by reaching the goal in fewer steps. The success rate climbs sharply and remains close to 1.0, which suggests that once Q-learning finds an efficient path, the goal is consistently reached. The off-policy nature of Q-learning allows value estimates to be propagated more aggressively from the goal backwards, thereby accelerating convergence.

- **SARSA (blue):**
  Although the success rate converges to a high value, it does so more gradually compared to Q-learning. The average reward initially lags behind that of Q-learning because SARSA updates using the actual action taken (on-policy), which leads to more cautious exploration under stochastic wind conditions. Over time, SARSA's performance approaches that of Q-learning, demonstrating that near-optimal paths are eventually learned.

- **Monte Carlo (green):**
  The method remains at a very low success rate and yields highly negative average rewards over the limited number of episodes. This outcome is attributed to the need for complete episodes to update the action values; in a large or stochastic environment, successful episodes are rare early in the experiment. With a larger number of episodes or an alternative exploration strategy, the Monte Carlo method might eventually converge, but under the current settings it struggles.

3 Brick Breaking Game Task

3.1 Problem Description and Environment Setup

The second project implements a discrete version of the Breakout game as defined in `2_secondary.py`. The testbed for the experiment includes:

- A 10×10 grid.
- A paddle fixed at the bottom row that can move left or right.
- Bricks arranged in the top row, represented by a bit mask.
- A ball that moves with fixed velocity, bouncing off walls, bricks, and the paddle.

The experiment is conducted in a simulated environment where the paddle is controlled by a Q-learning agent, and game dynamics (ball movement, collision detection, and reward distribution) are implemented following the predefined rules. This experiment is related to the final project, which will address a more complicated breakout game with a larger state space. To evaluate Q-learning on a simpler testbed, the breakout game is abstracted to a much smaller state space. An important improvement in this testbed is the careful design of the state space to achieve efficient learning.

3.2 Environment Visualization

To help the reader understand the experimental testbed, Figures 2 and 3 below show screenshots of the breakout game environment. Figure 2 displays the game in progress, while Figure 3 shows the game state after clearing all bricks.

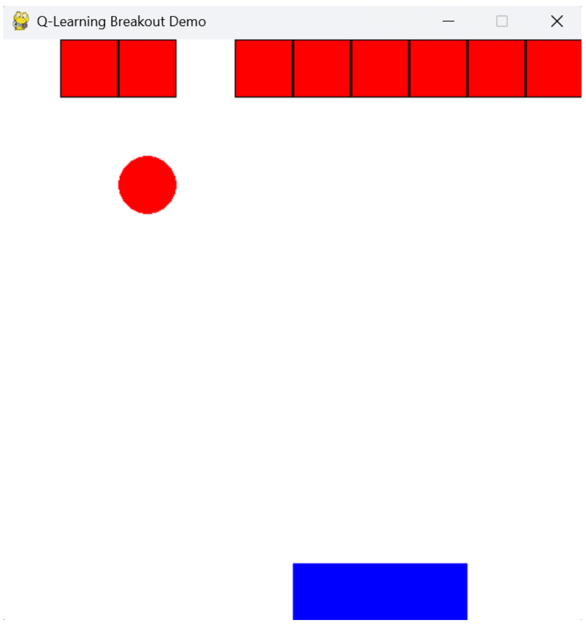

Figure 2: Breakout game in progress.

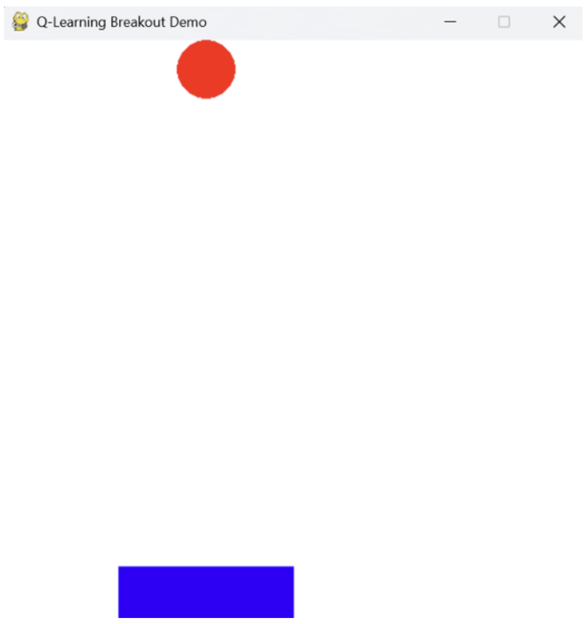

Figure 3: Breakout game after clearing all bricks.

3.3 State and Action Space

3.3.1 State Space

The state is represented by a tuple that captures the key elements of the game:

$$S = (\text{ball}_x, \text{ball}_y, v_x, v_y, \text{paddle}_x, \text{brick\_count}),$$

where:

- $\text{ball}_x, \text{ball}_y$: The position of the ball.
- v_x, v_y: The ball's velocity components.
- paddle_x: The leftmost position of the paddle.
- brick_count: The number of bricks remaining in the top row.

3.3.2 Action Space

There are three possible actions for the paddle:

- **0:** No movement.
- **1:** Move paddle left.
- **2:** Move paddle right.

3.4 Reward Setting

The reward design in the Breakout game is as follows:

- A small step penalty of -0.1 per move.
- A reward of $+5$ when the ball hits a brick (and the brick is removed).
- A reward of $+1$ when the ball hits the paddle.
- A final reward of $+10$ if all bricks are cleared (victory) or a penalty of -10 if the ball is lost (failure).

3.5 Q-learning Iterative Update

The Q-learning algorithm is implemented with an ϵ-greedy strategy (with ϵ decaying linearly) and uses the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big].$$

Here,

- α is the learning rate.
- γ is the discount factor.
- r is the reward received after taking action a in state s.
- s' is the next state, and $\max_{a'} Q(s',a')$ represents the maximum Q-value for the next state.

3.6 Experimental Testbed and Results

The experiment was conducted over a large number of episodes (e.g., 100,000 episodes) with the Q-learning agent, and performance metrics (episode reward and success rate) were recorded. Figure 4 displays the learning curves for the Breakout game, showing the evolution of episode rewards and success rate (defined as clearing all bricks) with a moving average applied for smoother visualization.

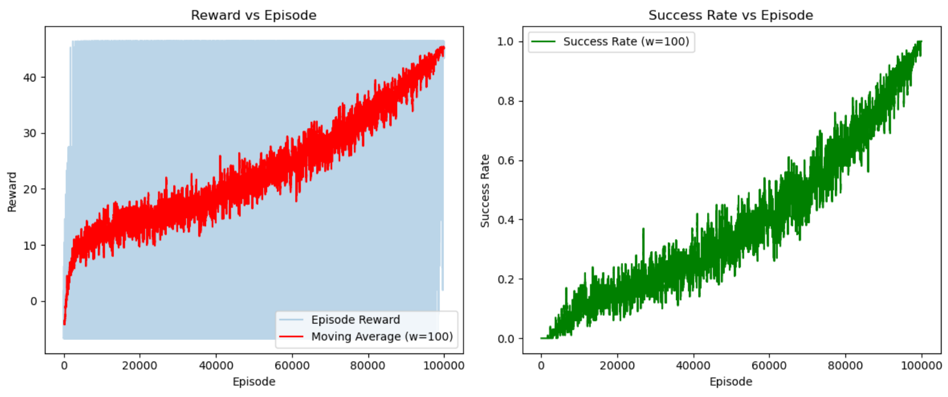

Figure 4: Learning curves for the Breakout game task (from `2_secondary.py`).

The experimental results indicate that the Q-learning agent effectively learns to control the paddle. The learning curves reflect an increase in average reward and success rate over episodes, while the game demonstration confirms that the agent can clear all bricks.

3.7 Future Extensions toward a Full Breakout Game

In future work, the insights gained from this simplified Breakout environment will be used to design a more complete version of the game with a larger state space and additional features (e.g., multiple rows of bricks, advanced collision physics, power-ups). The next step is to extend Q-learning to more powerful function approximators such as Deep Q-Networks (DQN), which can handle high-dimensional inputs effectively. This approach will allow the policy to generalize better in complex environments, paving the way for a robust solution to the full Breakout game.

4 Conclusion

This report presents two reinforcement learning applications developed as part of Homework3. The grid world wind simulation task incorporates a stochastic wind environment and King's Moves, combining ideas from Exercises 6.9 and 6.10 in *Reinforcement Learning: An Introduction*. The experiment employs SARSA, Q-learning, and Monte Carlo methods to guide the agent toward a target, with detailed descriptions of state-action space, reward design, and iterative update rules. The learning curves reveal that

Q-learning demonstrates the fastest convergence, SARSA converges more cautiously, and Monte Carlo struggles under the given episode limit.

The Breakout game task uses a Q-learning agent to control a paddle in a discrete grid environment, with clearly defined state space, action space, reward settings, and update formulas. The experimental results and visualizations confirm that the agent can successfully learn to clear all bricks. In addition, the simplified environment design offers valuable insights for scaling to a full Breakout game and transitioning from tabular Q-learning to deep reinforcement learning methods such as DQN.

5 References

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd Edition).
- Additional online resources and documentation for Python, NumPy, Matplotlib, and Pygame.