October 10th 2024

To:
Professor Dieu
Phenikaa University
Ha Dong, Ha Noi

Dear Professor,

We are pleased to present our Capstone Senior Design project titled "Comparison of Reinforcement Learning Algorithms: Q-Learning vs. SARSA in Tic-Tac-Toe." This project explores the performance of two reinforcement learning algorithms in a game environment. We hope the findings from this project will contribute to the study of AI in games.

Sincerely,

Quan

ABSTRACT

This project investigates the performance of two reinforcement learning algorithms, Q-Learning and SARSA, in the game environment of Tic-Tac-Toe. Both algorithms were implemented and the agents were trained over multiple games. The performance of each agent was measured based on their win rate in head-to-head matches. Results showed that Q-Learning generally outperformed SARSA. The report provides insight into the learning mechanisms of both algorithms.

Contents

List of Figures

Chapter 1

Introduction

1.1 Project Definition

Consider the familiar child's game of tic-tac-toe. Two players take turns playing on a three-by-three board. One player plays Xs and the other Os until one player wins by placing three marks in a row, horizontally, vertically, or diagonally. Although this is a simple problem, it cannot readily be solved in a satisfactory way through classical techniques. [1]

This project aims to compare the performance of two reinforcement learning algorithms, Q-Learning and SARSA, in the game of Tic-Tac-Toe. Both agents are trained to learn optimal strategies through interaction with the environment, and their performance is evaluated in a competitive setting.

1.2 Project Objective

The objective of this project is to build agents using Q-Learning and SARSA algorithms and compare their effectiveness. This involves training the agents to play Tic-Tac-Toe and evaluating their performance in a series of matches.

1.3 Project Specifications

- A 3x3 Tic-Tac-Toe grid is used.
- Agents play 2000 episodes, each episode containing 10 games.
- State: current state of the board, can be a list or array of 9 elements representing all the cells on the board (1:X, -1:O, 0:none)
- Action: the move the agent will take in that state (represented by an index from 0 to 8).
- Rewards: +5 for a win, -5 for a loss, and 0 for a draw.

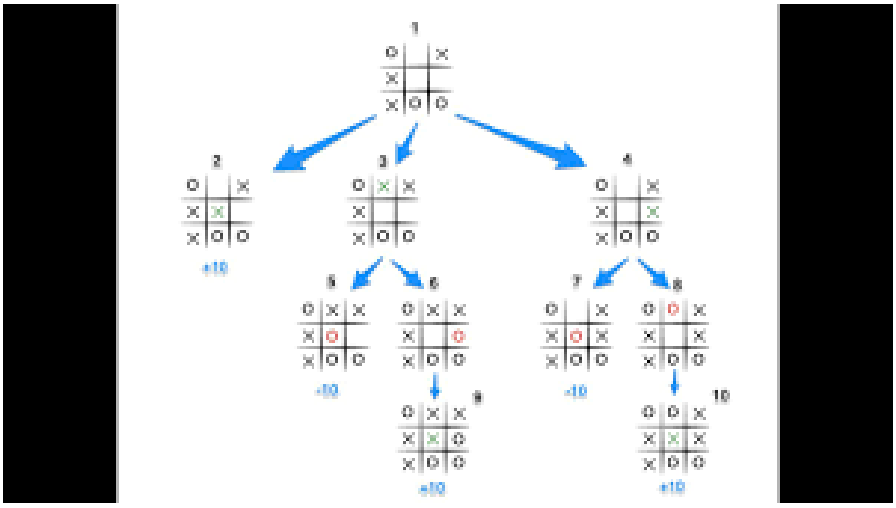


Figure 1.1: Example of different actions lead to diffferent states.

1.4 Project Applications

The algorithms used in this project have applications in game AI and decision-making systems, where reinforcement learning is critical for optimizing strategies in complex environments.

Chapter 2

Literature review

2.1 Project Background

Tic-Tac-Toe is a well-known game that has been used to study AI. Reinforcement learning, specifically Q-Learning and SARSA, are popular algorithms for training agents in such environments. This project builds on previous studies by comparing the two algorithms head-to-head in Tic-Tac-Toe.

2.2 Previous Work

Past research has explored the use of Q-Learning in game environments, demonstrating its efficiency in learning optimal strategies.

SARSA, an on-policy algorithm, has also been studied for its ability to follow learned policies in real-time decision-making.

2.3 Comparative Work

While past projects focus on training agents against random opponents, this project compares Q-Learning and SARSA agents directly, providing a practical analysis of both algorithms' learning capabilities.

Chapter 3

Standards and Methods

3.1 Design Constraints

One limitation of Tic-Tac-Toe is the limited number of game states. The computational complexity is also constrained by the number of training episodes required for convergence.

3.2 Engineering Design Standards

This project follows standard AI design principles, including modular code and reusable environments. Python was used to implement the Tic-Tac-Toe environment and the agents.

3.3 Theory and Theoretical Calculations

The Q-Learning update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max Q(s',a') - Q(s,a)\right)$$

SARSA's update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma Q(s',a') - Q(s,a)\right)$$

.

Both algorithms will follow similar steps:

- State Transition: From current state s to next state s'.

- Action Selection: Select an action based on the current state.

- Reward: Receive a reward after taking the action.

The difference lies in how they update the Q-values:

- Q-Learning: Uses the maximum possible future reward (max Q(s', a')).

- SARSA: Uses the reward for the actual action the agent chooses (Q(s', a')).

3.4 Product Subsystems and Components

- **Tic-Tac-Toe Environment**: Implements the game logic.
- **Q-Learning Agent**: Trains the agent using the Q-learning algorithm.
- **SARSA Agent**: Trains the agent using SARSA.

3.5 Manufacturing and Assembling (Implementation):

The implementation was done using Python. The Tic-Tac-Toe environment was created as a class, while both Q-Learning and SARSA agents were implemented using standard reinforcement learning algorithms. Both agents were trained over 2000 episodes and compared in terms of performance.

Chapter 4

Results and Discussions

4.1 Results and Analysis:

After training the Q-Learning and SARSA agents for 2000 episodes, the performance of both agents was measured by calculating the cumulative rewards after each episode (10 games). However, due to the stochastic nature of reinforcement learning, there were significant fluctuations in the rewards for individual episodes. To mitigate this issue and observe a clearer trend in the learning process, we averaged the rewards over every 10 episodes.

4.1.1 Averaging for Smoother Results

In reinforcement learning, the exploration-exploitation trade-off can lead to erratic short-term rewards, especially in early episodes when

the agent is still learning the environment. By averaging the rewards every 10 episodes, we reduce the noise caused by this inherent randomness and can better observe the long-term trends in the learning process. This technique helps to smooth out the curve, making it easier to identify whether the agent is truly converging or if there are persistent fluctuations.

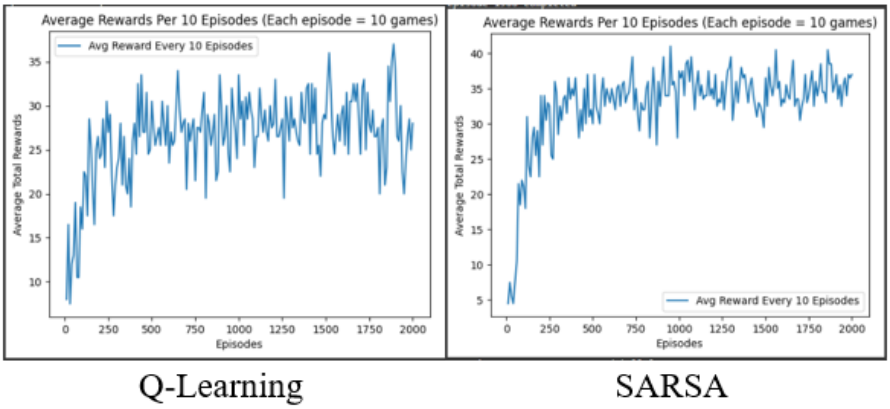
4.1.2 Benefits of Epsilon Decay

Epsilon decay is an important strategy in the reinforcement learning process to reduce exploration over time. Gradually, epsilon is decreased to shift toward exploitation, meaning the agent relies more on what it has learned to make the best decisions.

Main Benefits of Epsilon Decay:

- Balance Between Exploration and Exploitation:

When epsilon is high, the agent has the opportunity to explore many different strategies and discover more advantageous actions. However, if epsilon remains too high for too long, the agent will not be able to exploit what it has learned. As epsilon decreases, the agent transitions from exploration to exploitation, focusing on applying the optimal decisions it has learned, thereby improving performance.

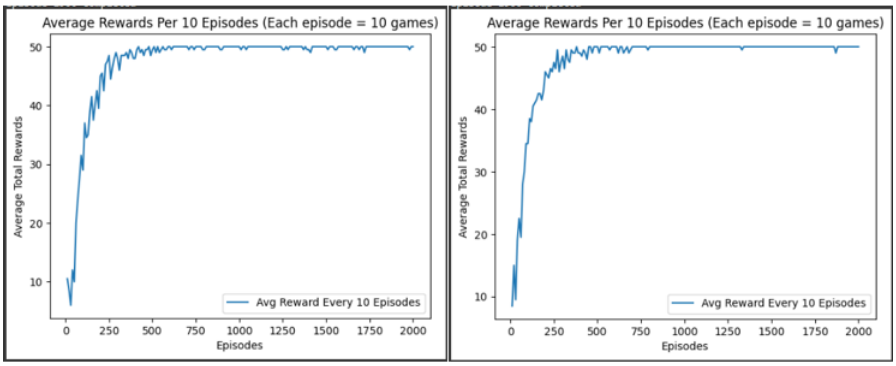


Q-Learning SARSA

Figure 4.1: Without decaying epsilon.

- Reducing Noise from Random Exploration:

In the early stages, random actions can cause significant fluctuations in results. As epsilon decreases, the agent will choose fewer random actions, leading to more systematic actions and more stable performance. This helps reduce variance in the learning process, supporting better convergence.



Q-Learning SARSA

Figure 4.2: Decaying epsilon (Converge).

- Optimizing the Learning Process:

An agent that starts with a high epsilon will learn more from trial-and-error actions, and as epsilon decays, it will focus on the optimal decisions it has learned, thus improving learning efficiency.

So I use "agent.epsilon *= 0.995" to reduce epsilon over each episode. The following graph shows the convergence of both Q-Learning and SARSA after training for 2000 episodes. As can be seen, both methods stabilize towards the end.

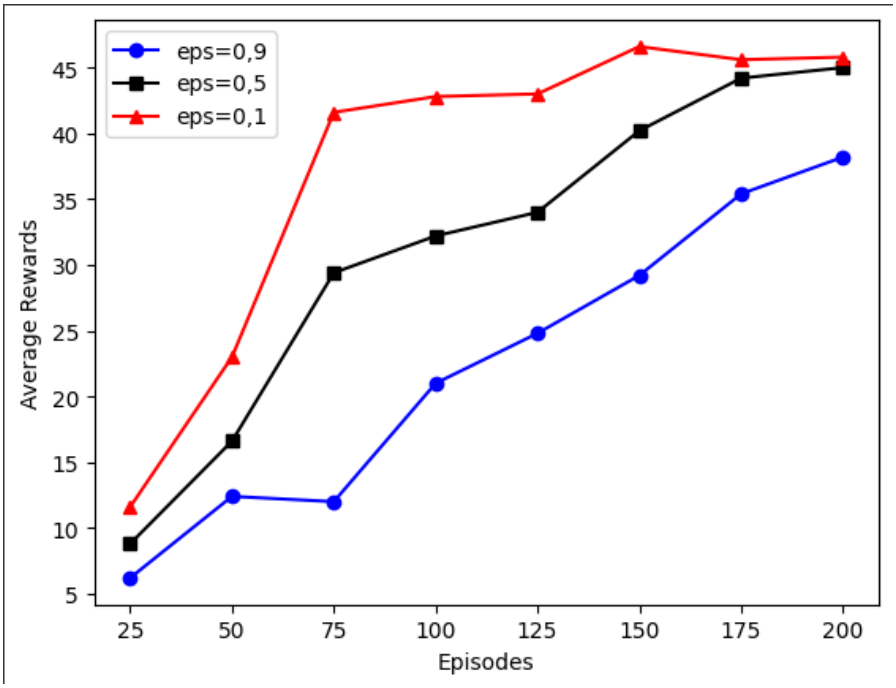


Figure 4.3: Comparision of 3 values of epsilon in Q-learning.

4.1.3 Choosing value of epsilon.

In selecting an epsilon value, I compared initial values of 0.1, 0.5, and 0.9, observing each over the first 200 episodes and then through the full training period of 2000 episodes. In the initial comparison, the reward for epsilon = 0.9 was notably lower than for other values. This result is due to the agent's high exploration rate at epsilon = 0.9, which leads to excessive exploration and less focus on maximizing rewards in a stable way. Conversely, epsilon = 0.1 achieved the highest rewards initially, indicating that the agent was exploiting learned strategies effectively from early on.

$\varepsilon = 0{,}5$ $\varepsilon = 0{,}1$

Figure 4.4: Compare the convergence in Q-learning.

However, examining the entire 2000 episodes separately showed that, over time, an epsilon of 0.1 tended to converge more slowly than an epsilon of 0.5. This is likely because a low epsilon value (0.1) limits the agent's exploratory capacity, causing it to settle on safer moves and miss opportunities to explore optimal strategies for handling more complex states. The epsilon = 0.5 option, by comparison, balanced exploration and exploitation, allowing the agent to adapt efficiently to both early and late-game scenarios. Thus, epsilon = 0.5 emerged as the most effective balance, promoting steady convergence over time.

Similar to Q-learning, SARSA's choice of epsilon follows a similar approach in both method and results.

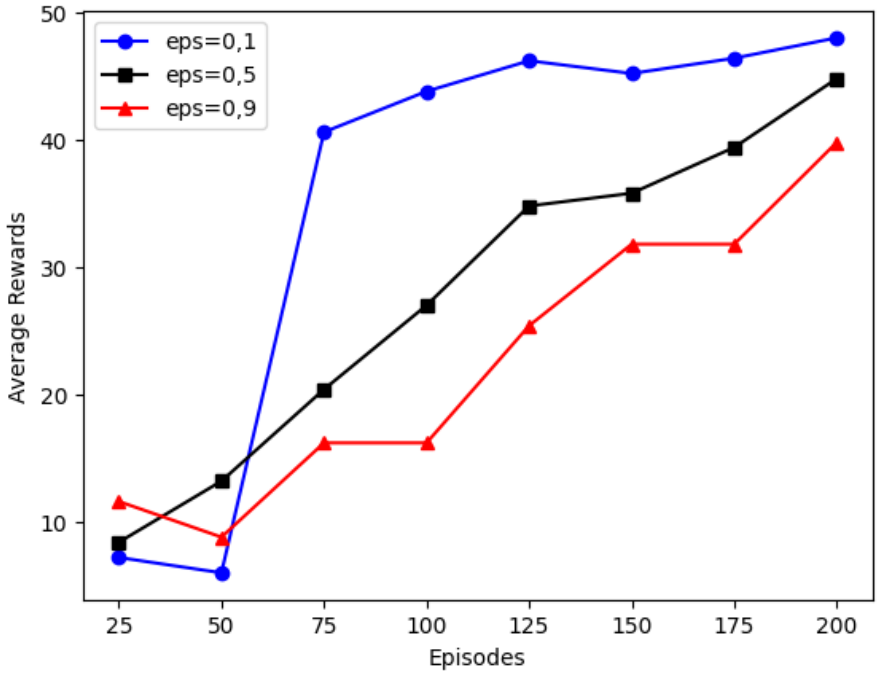
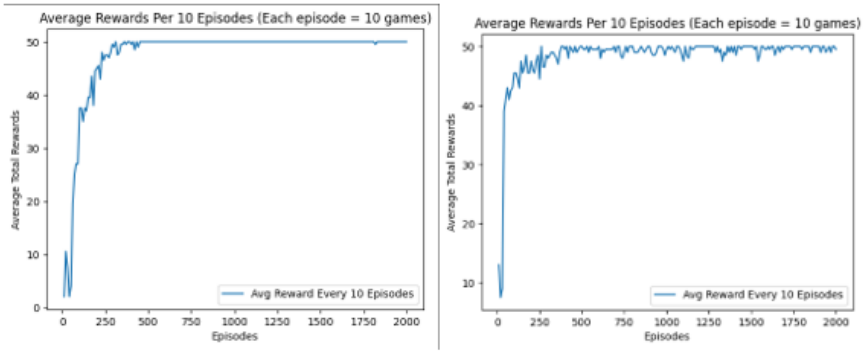


Figure 4.5: Comparision of 3 values of epsilon in SARSA.



$\varepsilon = 0{,}5$ $\varepsilon = 0{,}1$

Figure 4.6: Compare the convergence in SARSA.

4.2 Discussions:

Q-Learning's off-policy nature allowed it to explore more optimal moves even when it was not directly following the learned policy. This gave it an advantage in exploiting weaker moves made by the SARSA agent. However, SARSA's on-policy learning produced more consistent strategies, which is evident from the fewer mistakes in its decision-making process.

After training both agents, we can have them compete directly to evaluate the effectiveness of the two algorithms.

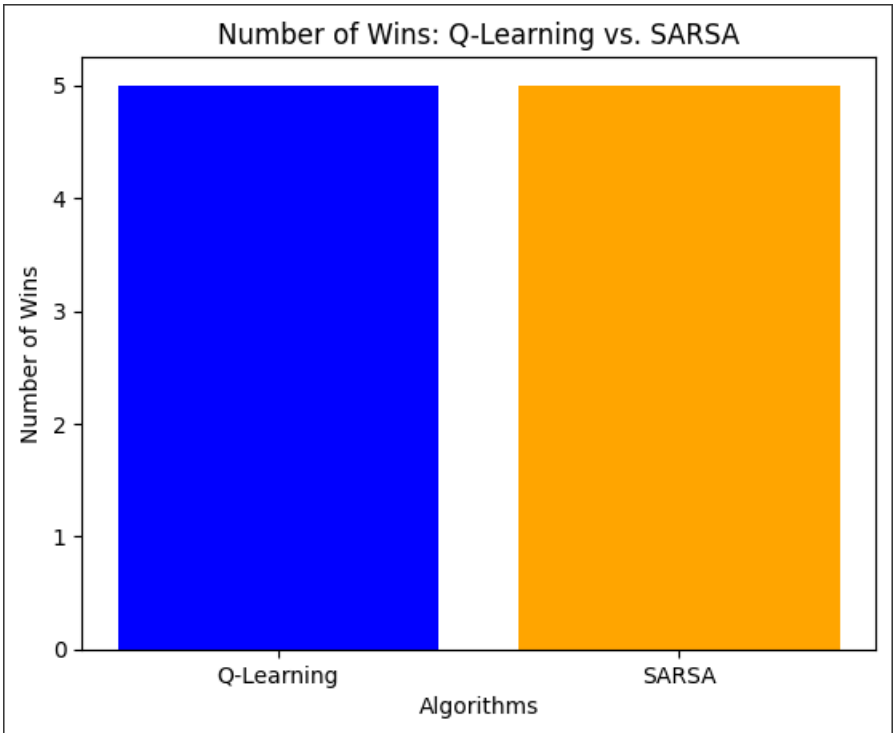


Figure 4.7: Bar charts of Win/Loss Comparison.

After 10 head-to-head matches, the Q-Learning agent and the SARSA agent each won 5 games, with no draws. The balanced outcome highlights that while Q-Learning may explore and exploit more efficiently, SARSA's stability through its policy-driven learning helps it maintain consistent performance. Both algorithms proved effective in learning the game of Tic-Tac-Toe, and their strengths and weaknesses largely balance out in this specific environment.

Chapter 5

Project management

5.1 Designing and implementing the Tic-Tac-Toe environment.

In the project, I implemented a specific reward system to guide the learning process of both Q-Learning and SARSA agents. The rewards were assigned as follows:

+5 for a win: This encourages the agent to prioritize moves that lead to victory.

0 for a draw: This neutral reward prevents the agent from treating a draw as either a success or a failure.

-5 for a loss: This penalty discourages the agent from taking actions that result in losing.

-1 for invalid moves: This small penalty was introduced to discour-

age the agent from attempting illegal or repetitive moves, pushing it to explore valid and strategic actions instead.

This reward system helped the agents focus on optimizing their strategies while avoiding inefficient or incorrect actions, leading to more effective learning and game performance.

5.2 Implementing and training the Q-Learning and SARSA agents.

During the project, I implemented a custom function called *check-danger*. This function helps the agent detect dangerous moves—when the opponent is about to win—and block them effectively. By adding this feature, the agents became more strategic, preventing simple losses and improving overall performance. This enhancement was particularly useful in both Q-Learning and SARSA, as it allowed the agents to respond to immediate threats during the game.

5.3 Design interface.

To visualize the learning performance of both Q-Learning and SARSA agents, I utilized the matplotlib library in Python, specifically em-

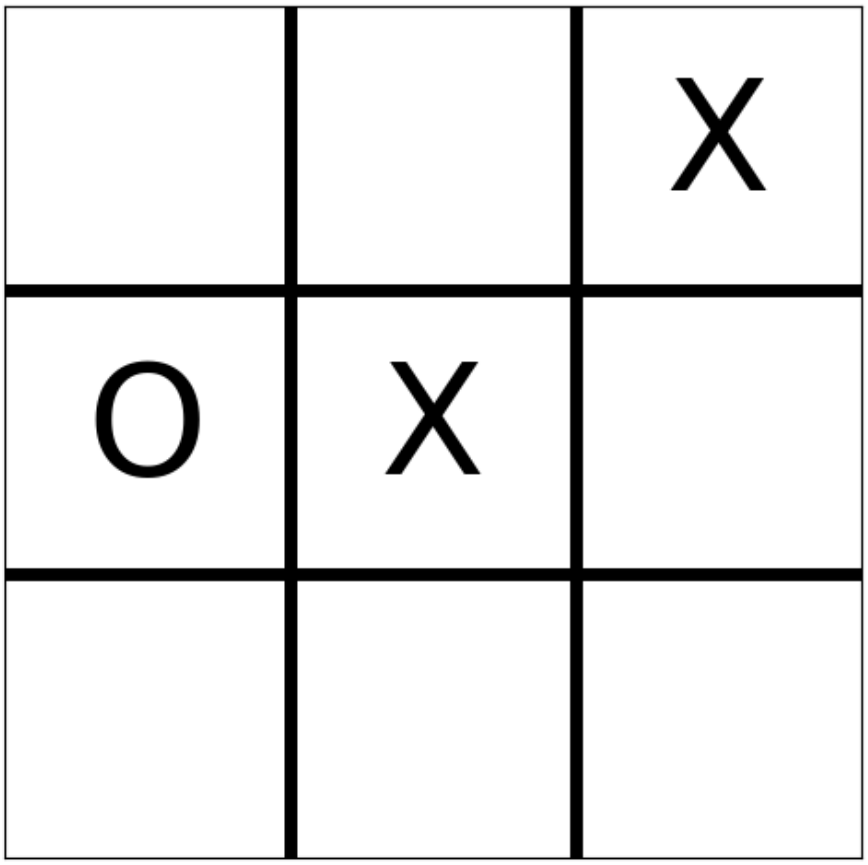


Figure 5.1: Main interface.

ploying the plot function to generate line charts. This function enabled tracking and comparison of the average rewards over episodes, making it easier to analyze trends, detect convergence, and observe the impact of different parameters (e.g., epsilon decay) on learning efficiency.

5.4 Evaluating the performance of both agents and documenting the results.

One of the biggest challenges was managing the exploration-exploitation balance in both Q-Learning and SARSA. Initially, high epsilon values caused the agents to explore excessively, which delayed the convergence process. After several iterations, I implemented epsilon decay to gradually reduce exploration, allowing both agents to focus on exploiting learned strategies. Another challenge was managing the training time. As I conducted 2000 episodes for each agent, the time required to process each match was substantial.

Chapter 6

Conclusion and Future Recommendations

6.1 Conclusion

The main objective of this project was to demonstrate the effectiveness of Q-Learning and SARSA algorithms in a Tic-Tac-Toe environment. Both algorithms successfully learned to make strategic decisions through interactions with the environment, showing their practical effectiveness in reinforcement learning tasks.

Q-Learning excelled in exploring the game space, using its off-policy nature to evaluate potential future actions and find more aggressive strategies. This led to higher performance in certain situations, proving its strength in finding optimal solutions.

SARSA, on the other hand, prioritized stability and consistency. As

an on-policy algorithm, it followed the actions it actually performed, which resulted in fewer mistakes and a more reliable decision-making process, though it didn't always achieve as many wins.

While the comparison between the two algorithms was part of the study, the results primarily highlight the effectiveness of both algorithms in adapting to and optimizing their strategies within the game environment. Epsilon decay was critical in guiding both agents toward convergence, helping them balance exploration and exploitation.

Ultimately, the project demonstrates that both Q-Learning and SARSA are effective reinforcement learning techniques, with each excelling in different aspects of the learning process.

6.2 Future Recommendations

Future work could involve testing these algorithms in more complex games or environments, such as Connect Four or Chess. Additionally, tuning hyperparameters and increasing training episodes could provide further insights.

Bibliography

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 2018.