Battleship with Reinforcement Learning

Applying Q-Learning and SARSA Algorithms to Reinforcement Learning in Strategic Decision-Making Games

Nicholas Wong

Student at Northeastern University

Abstract

This paper explores the application of reinforcement learning algorithms, Q-Learning and SARSA, in the strategic game of Battleship. By simulating controlled and random environments, the performance of these algorithms was evaluated based on accuracy metrics. The results demonstrate the strengths and limitations of each approach, offering insights into the applicability of reinforcement learning to dynamic decision-making problems.

Introduction

Battleship is a strategic, two-player game in which the players are challenged to sink another fleet of ships hidden in a given grid. This game mixes unknown information, decision-making, and strategic planning. These game variables make it ideal to utilize artificial intelligence techniques. One technique is reinforcement learning which “can applied to make game characters (agents) learn decisions and behaviors based on their environment” (Tian). Using reinforcement learning will allow me to create an intelligent battleship-playing agent to maximize the chances of winning a game of Battleship

I used the SARSA (State-Action-Reward-State-Action) and Q-learning algorithms in this project. Essentially, the agent was rewarded for hitting a target which allowed the agent to strategically target spots to maximize hits and minimize misses. The main objective was to compare SARSA's and Q-learning's effectiveness in decision-making in a game of Battleship.

Methods

The approach I used for the project was reinforcement learning. Reinforcement learning is a method where agents learn over time based on feedback on their actions. For this project, the feedback was represented as positive rewards for landing a hit on the opponent's ship. The reinforcement learning algorithms that were used for the Battleship agents were Q-learning and SARSA. Q-learning is a model-free, off-policy algorithm that uses a value function to determine the most optimal action to take and a given probability of choosing a random action. SARSA is similar to Q-learning as they are both model-free algorithms that use a value function, however, SARSA is an on-policy algorithm. The policy used involves choosing moves based on whether a ship is been found. The agent defines a found ship as landing two consecutive, adjacent hits.

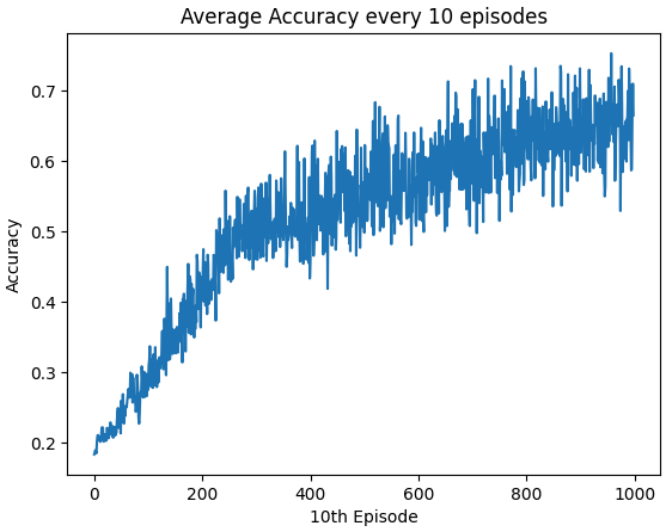
To train the agents with each algorithm, a controlled and random simulation was run. The controlled simulation used a set board placement for each episode of a total of 10,000 episodes. This approach allows the agents to form a strong understanding of the locations of the opponent's ships since these locations don't change. Measuring the performance of the controlled simulation creates a basis for judging the random simulation. Unlike the controlled simulation, the random simulation changes the board placements after each episode of a total of 50,000

episodes. Therefore, the agents cannot rely on their memory from the last board placement. Instead, the agents must learn consistent patterns between states and actions.

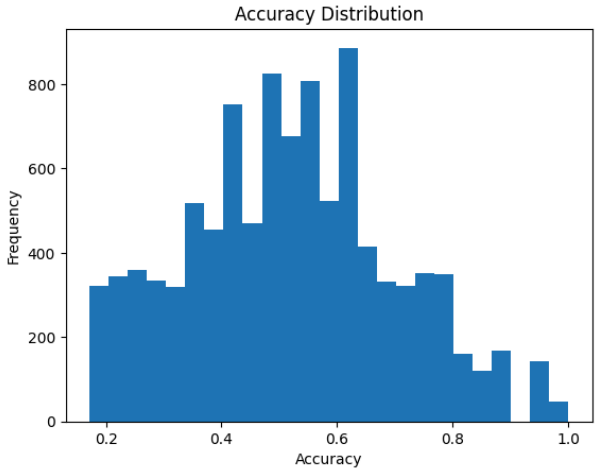
To measure the performance of these simulations, the accuracy of each episode was recorded. Accuracy for these agents is calculated by the ratio between the number of hits, 17, and the total number of rounds. Using this metric, the performance of the agents can be evaluated based on accuracy per episode, average accuracy, maximum average, and the distribution of accuracy across all episodes. Measuring the accuracy per episode shows how the accuracy changes after each episode. Average accuracy describes the agent's overall performance, maximum average describes the agent's strongest performance, and the distribution of the accuracy shows the consistency of the agent.

Results

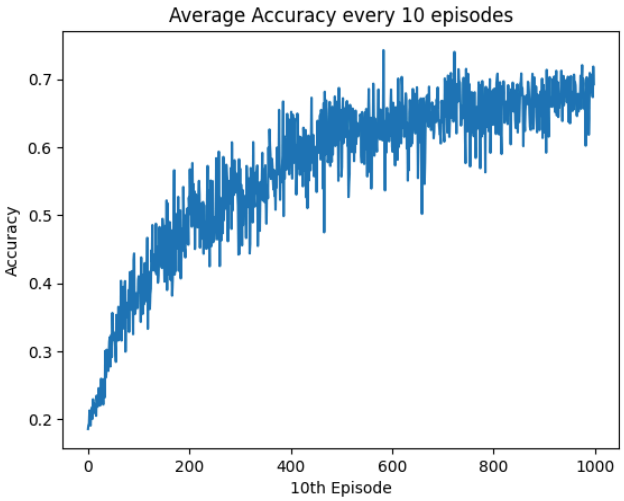
The Q-learning and SARSA algorithms both successfully learned how to play Battleship. Both algorithms performed very well in the controlled simulation. The Q-learning algorithm had the highest accuracy of 100%, but the SARSA algorithm had a higher average accuracy of 57% compared to Q-learning's 52%.



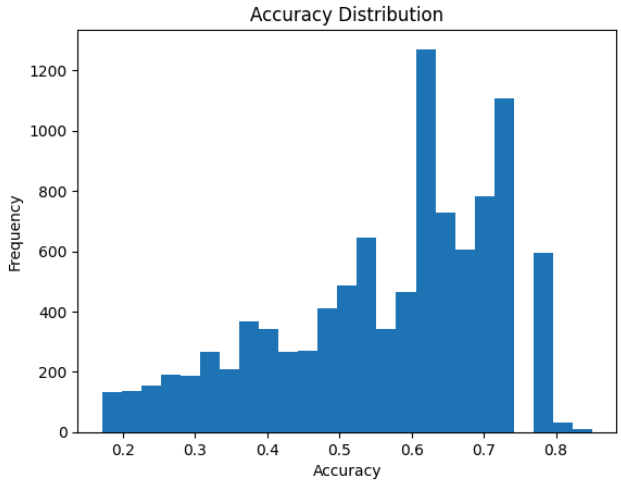
Average accuracy every 10 episodes for Q-learning Controlled Simulation



Distribution of accuracy for Q-learning Controlled Simulation



Average accuracy every 10 episodes for SARSA Controlled Simulation

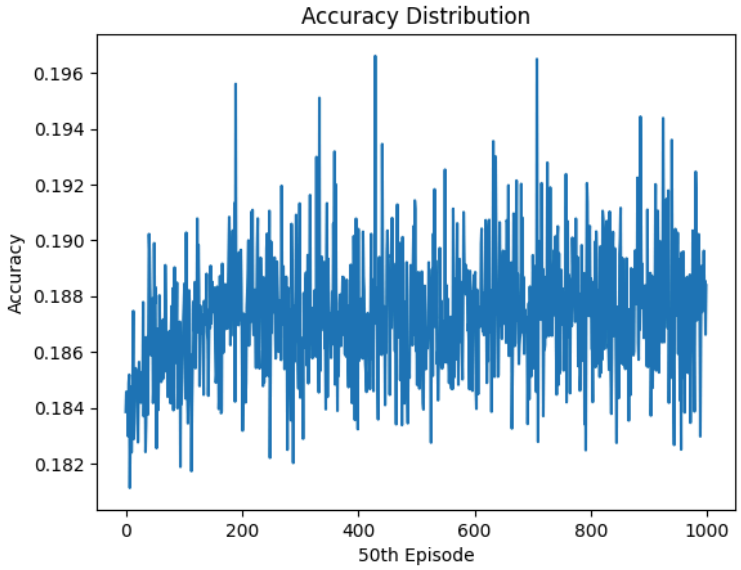


Distribution of accuracy for SARSA Controlled Simulation

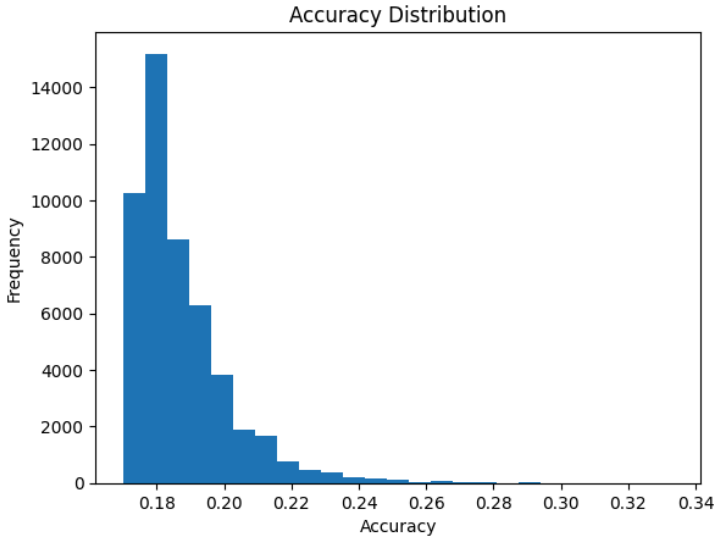
Q-learning and SARSA had a similar relationship between accuracy and the number of episodes in the controlled simulation. Both have the accuracy increasing after each episode and level off at around 65 % accuracy. Although the change accuracy over episodes was similar, the distributions were different. The Q-learning algorithm's distribution is more symmetrical with the most common accuracy being around 50% and the surrounding accuracies are less common. In the

SARSA algorithm's distribution, as the accuracy increased, the frequency increased. Until 60% where the frequency slowly decreased.

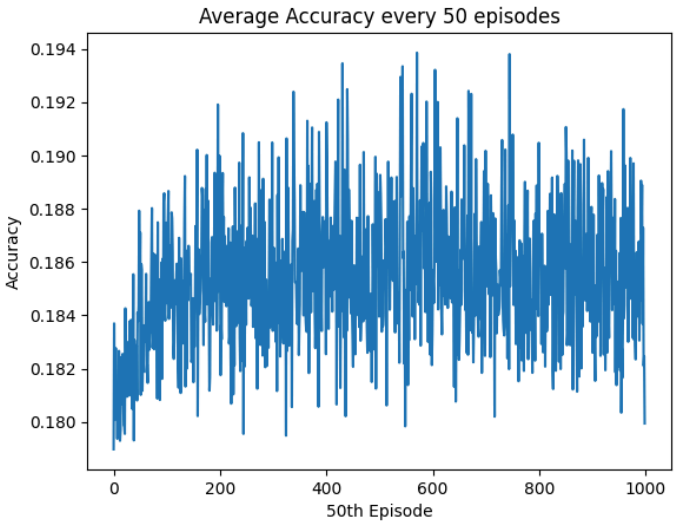
Using the results from the controlled simulation, the performance of the random simulation can be evaluated. In this simulation, the SARSA algorithm had a higher maximum accuracy of 40% compared to Q-learning's 33%. Both algorithms had the same average accuracy of 19%.



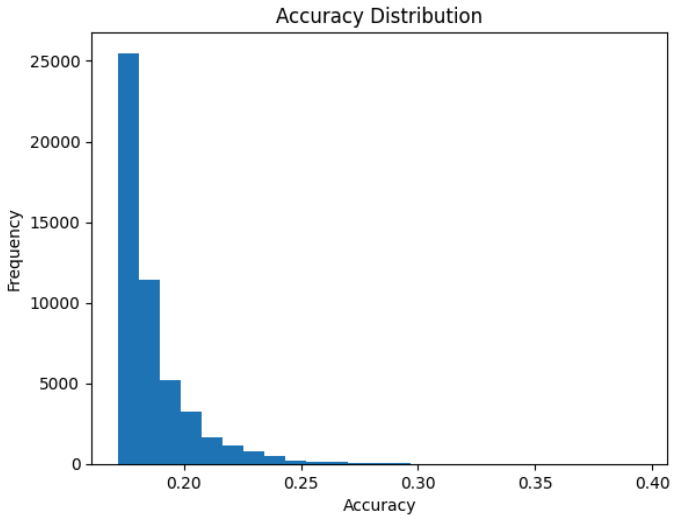
Average accuracy every 50 episodes for Q-learning Random Simulation



Distribution of Accuracy for Q-learning Random Simulation



Average accuracy every 50 episodes for SARSA Random Simulation



Distribution of Accuracy for Q-learning Random Simulation

The distribution of the accuracy and the accuracy over episodes was very similar between the SARSA and Q-learning. Both algorithms improve their accuracy in the first 10,000 episodes,

then the average accuracy remains stable. In both distributions, there is a peak in accuracies under 20% and a sharp decline moving forward.

Conclusion

Based on the results, the SARSA algorithm was slightly better than the Q-learning algorithm at playing Battleship. The SARSA algorithm had a higher average accuracy in the controlled simulation by 5% and a higher maximum accuracy in the random simulation by 7%. Since the SARSA outperformed the Q-learning on more metrics, the SARSA algorithm was better for the problem.

The reason SARSA outperformed Q-learning is its policy. The policy used in the algorithm allows for a more strategic way of landing hits than Q-learning's off-policy approach. The policy used in the algorithm helps the agent completely sink a ship by following the direction of the hits already landed on said ship. Although this strategy can lead to shooting one too many shots, it greatly helped the agent land hits in the random simulation because the policy doesn't rely on the value function. Even though the policy approach in the SARSA algorithm helps the agent land more hits, the Q-learning algorithm was more accurate in the controlled simulation.

The Q-learning algorithm was able to outperform the SARSA algorithm in the controlled simulation because of its off-policy approach. Since the algorithm does not utilize a policy to select actions, actions are chosen either at random based on the epsilon probability or optimally from the value function. The epsilon greedy approach allows the agent to explore the outcomes of random actions and update the Q matrix accordingly. After several explorations, the agent can

confidently locate the ships using the information from the Q matrix. Because Q-learning can let the agent choose the best action for each turn, the agent can play a round of Battleship with zero misses when all the ships are located. However, the agent can struggle to hit unlocated ships without a policy.

Overall, the Q-learning and SARSA algorithms were a good approach for playing the game of Battleship. Although a large amount of training episodes were needed, the algorithms were able to learn and improve at playing the game. Keeping track of the rewards of actions provided success in later episodes for the agents. Since the information the agent holds refers to previous actions, the algorithms do not handle changes to the environment well. However, when the environment is stable, the algorithm can be very accurate.

For future work, I could extend this project in many ways. An example of something that I can work on further is incorporating domain-specific knowledge such as a probabilistic model of ship placements to figure out what the best arrangement of your fleet of ships is to give you the greatest chance to win against your opponent. Another avenue that could be explored is to use the knowledge of reinforcement learning that I developed for this project and work on another game with a more intricate game environment and then compare the differences. This will allow me to have another perspective on reinforcement learning and expand my knowledge of the algorithms used.

This project not only showcased the effectiveness of reinforcement learning techniques but also offered valuable insights into the application of AI for solving real-world challenges. This project allowed me to deepen my understanding of the principles and obstacles in designing intelligent agents. Additionally, this project laid a solid groundwork for future endeavors,

providing me with the knowledge and tools necessary to create more advanced and complex AI-driven systems that can tackle a variety of practical applications.

Works Cited

Tian, Xinhe. "AI Applications in Video Games and Future Expectations." Proceedings of the 4th International Conference on Signal Processing and Machine Learning. DOI: 10.54254/2755-2721/54/20241484