# CS7641 – Machine Learning

Assignment 4 – Markov Decision Processes and Reinforcement Learning

Mason J. Kelchner
mason.kelchner@gatech.edu

## 1 INTRODUCTION

This analysis explores the application of reinforcement learning techniques to solve Markov Decision Process (MDP) type machine learning problems. An MDP is a process described by states, actions, and rewards. For example, consider the game Connect 4 except in this scenario when playing there is a small probability that when you attempt to play in a column, that you instead accidentally place your coin in one of the adjacent columns (perhaps this has happened to you). The game follows an MDP in that after every action taken by each player, the Connect 4 board changes i.e. the state of the game is completely defined as the board state and which players' turn it is. Each player at any point in the game (at any state) has a certain choice of actions to take. Sometimes the columns or rows fill up in which case some actions are not available to a player given a state. Likewise, the game ends when a player is able to align 4 or more coins in a column, row, or diagonal. This outcome from an agent's perspective would achieve a reward or +1 for winning and -1 for losing. According to the Markov property, an MDP's future evolution is independent of its history and thus we can work to optimally solve MDPs using a variety of algorithms. While the Connect 4 example is easily explained, its implementation and solution as a reinforcement learning problem is difficult and exponentially large as the game contains a total of 4,531,985,219,092 possible board states [1]. For this analysis, two MDP problems are described and solved using three reinforcement learning methods. The first is the game of blackjack which contains far fewer states than Connect 4 with only 290 states. The second is a simple grid-world style game where an agent stochastically explores a frozen lake with the goal of reaching a treasure but potentially falling through the ice in certain states. For this problem the game was a 20x20 grid world with 400 states forming a larger state space to explore. Both of these games were implemented using the open source repository in Python called *gym* and solved using the reinforcement algorithm learning module *bettermdptools* implementations of the three algorithms [3] [4] with a few alterations for Q-learning described later. The three algorithms used to solve these problems are value iteration, policy iteration, and Q-learning. For both games, a random seed was used to initialize both games for reproducibility for training. Only with Blackjack, when the optimal policy from each algorithm is tested were initializations of the game allowed to vary to evaluate general performance. For the Frozen Lake game, the policy was specific to the initialized game environment, thus it would be inappropriate to reinitialize a different random game state.

## 2 GAME DESCRIPTIONS

Blackjack is a popular game of cards where a dealer and player aim to build a hand as close as possible to a value of 21 without going over 21 with the winner having the highest value. Going over 21 immediately 'busts' and the player or dealer loses, respectively. The numbered cards having their face value, face cards having a value of 10 and an Ace able to be either 1 or 11. This implementation of the game only considered two actions for an agent: HIT or STAND. A more sophisticated implementation would consider additional actions available in many styles of Blackjack including DOUBLING DOWN, SPLITTING, and SURRENDERING. Each of these actions have their own outcomes and potentially change the rewards from playing since new states would be available from a particular starting state. Additionally, playing an INSURANCE bet which is a side bet that the dealer has blackjack is another more sophisticated betting action. Allowing these additional actions would affect the reward structure of a reinforcement learning agent in the game. However, for simplicity, only the actions STICK (take no more cards) and HIT (take one more card) are allowed in this analysis. This is an interesting MDP in that the states encountered by the agent are stochastic and a finite number of outcomes are available from a starting state. By extension, many reinforcement learning problems are formulated as games and Blackjack can be used to understand outcomes of choices i.e. actions given a state as it relates to the MDP.

The second game for this analysis was the popular Frozen Lake environment available in the open source repository *gymnasium*. "Frozen lake involves crossing a frozen lake from Start (S) to Goal (G) without falling into any Holes (H) by walking over the Frozen (F) Lake. The agent may not always move in the intended direction due to the slippery nature of the frozen lake" [4]. The action space available to an agent is to move left, down, right, or up; however, as previously stated, this implementation includes a probability that the resulting movement of a commanded action moves perpendicular to the intended direction. This probability is a fixed 1/3 for all moves and directions. For example, commanding an action of right has the probability of moving up or down with a probability of 1/3 for each and 1/3 for moving in the intended direction. This randomness drives significant variation in a global policy for a fixed frozen lake configuration compared with a game where this randomness is not included and the actions are completely deterministic. Since actions can have more than one state outcome from the randomness, a learning agent explores additional states. Like Blackjack, this game is interesting and relevant to reinforcement learning as larger grid worlds of the frozen lake game contain far more possible states than blackjack and its states resulting from actions are not deterministic. However, all states are accessible from the starting state in the Frozen Lake game, granted with small probability, while in Blackjack only a certain subset of states are accessible given a particular initial state. For example, an agent in Frozen Lake can explore all states by making successive move throughout the board, even when the action results are random, there remains a non-zero probability of reaching all other board states. While in Blackjack, being dealt an initial hand of a 4 and a 5 (i.e. a hand total of 9), and not considering the ability to SPLIT (only when the starting two cards are the same value) an agent cannot reach a state where the hand total is less than the initial hand value of 9 during this game. Along with the size difference in states in Frozen Lake, this difference is valuable for understanding the effect of sparse rewards in a large environment on the performance of reinforcement learning techniques. For this analysis, a grid size of 20x20 was used for the Frozen Lake problem, resulting in 400 states. For comparison with later results, the initialized game state for Frozen Lake is shown in Figure 2-1.

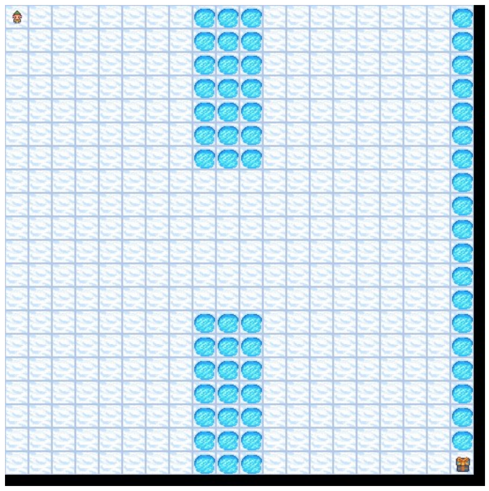

*Figure 2-1 Frozen Lake 20x20 initialize game state*

## 3 Policy and Value Iteration

Two algorithms applied to both MDPs in this analysis were policy iteration (PI) and value iteration (VI). Both involve iteratively updating a state-value-action mapping of an environment described by an MDP to find the optimal policy function (for PI) and optimal value function (VI). Given enough time and iterations, both policy iteration and value iteration will converge to the optimal solution for an MDP. Convergence for a PI and VI algorithm occurs when the value function for state transitions no longer increases during iterations. During each update step for PI and VI, the next policy and next value for a given state may not be known but it is guaranteed that the new policy and new value function is greater than or equal to the previous step's policy and value function at every state. The optimal value fulfills the Bellman optimality equation. The PI algorithm stops once the policy improvement doesn't change π. That is the exact case when v_π fulfills the Bellman optimality equation. Thus the policy π must be equal to an optimal policy, once policy iteration stops. In order to use PI and VI algorithms on an MDP problem, the transition and reward matrices must be known apriori. This is not always possible and deviates from some reinforcement learning problems where these matrices are not or cannot be known in advance and must be "discovered" through advancing through the MDP by making actions determining state transition probabilities. Additionally, PI and VI algorithms are limited to discrete state space MDPs as they are tabular algorithms with finite bounds. Continuous state space MDPs can be discretized to use PI and VI; however, uncertainty is introduced as chunking a continuous space into discrete spaces removes intermediate states which may or may not have a significant impact on the optimal policy and value functions. Both algorithms were applied to the Frozen Lake and Blackjack MDP problems. For the Blackjack problem, with a smaller state size, shorter evolution loops and quicker reward feedback than the Frozen Lake problem, both PI and VI algorithms converged to an optimal policy/value function quickly, converging within 0.01 seconds of wall clock run time. Likewise, even with a large state space and stochastic actions in the Frozen Lake MDP, both PI and VI converged within a few seconds of wall clock runtime. Value iteration converged in approximately 500 iterations while policy iteration converged within 10 iterations; however, both algorithms took similar amounts of runtime to converge.

## 4 Q-Learning

Q-Learning is a model free reinforcement learning class of algorithms which doesn't require the state and reward matrices apriori and instead builds a model through interacting with the environment, receiving rewards, and selecting new actions. Additionally, Q-Learning algorithms allow making "off-policy" actions allowing a reinforcement learning agent to explore the environment and move into states that may have not been previously explored with the goal of discovering potentially better actions in each state based on the discounted rewards received. The Q-learning process balances this off-policy exploration with the on-policy selected action of exploitation using a simple probability during a training episode which can be changed from episode to episode. Initial training episodes typically start with a high probability of selecting a random action in a given state, this is a high epsilon resulting in a high probability of exploration. As the agent receives rewards, it builds a Q matrix containing the Q values for taking a particular action, a, in a particular state, s. This matrix is updating according to the Bellman equation including the learning rate alpha and the discounted reward rate gamma. As the agent continuously re-explores the environment, the exploration parameter, epsilon, is reduced to increase the probability that the agent will select the on-policy best action in a state by selecting the action with the highest Q-value for that state-action pair. Q-learning is more flexible than PI and VI algorithms in that it is model free and doesn't require knowing the state-value functions in advance which in some MDPs can be impossible to recover. However, Q-learning requires significant computation especially for large state space MDPs since the agent must explore the environment for many iterations to build a Q matrix which accurately represents the true state-action-state transitions. Additionally, Q-learning is highly sensitive to a particular MDP's state space and reward function and requires significant parameter tuning to improve the performance of the learning algorithm and for the reward function to potentially altering the reward outcomes. Sparse rewards in a large state space MDP are particularly difficult for Q-learning in that many state-action-state transitions must occur during an iteration before a reward is obtain and propagated through the Q-table through discounting. This results in a slow learning process and increases the number of iterations necessary to converge to an optimal policy. Reward shaping can be implemented to produce intermediate rewards for actions not resulting in ending an iteration, this can improve the balance of exploration and exploitation and benefit the converge of the Q-learning algorithm. Reward shaping and parameter tuning were implemented for Q-learning for both of the MDPs in this analysis and particularly for the large state space Frozen Lake problem. For Q-Learning, convergence was determined as a minimal change in average q-value for all states resulting in no improvement in policy or state transition function.

## 5 Result Comparison

### *5.1 Blackjack*

For the Blackjack game, each algorithm was tested by simulating 10,000 games. A parameter grid search was performed using the Q-learning algorithm with the best criteria being determined after training by evaluating the percent of wins and losses of the algorithm versus the theoretical optimal percentages in Blackjack without counting cards. For Blackjack, the optimal policy wins and loss percentages are approximately 43% and 48%, respectively, with the remainder of games resulting in ties. The grid search of hyper parameters for Q-Learning varied the learning and epsilon initial, minimum and decay rates. Additionally, Q-Learning was allowed to iterate for 100,000 episodes. Convergence for Q-Learning was determined by evaluating the max q-value for each iteration and the average q-value in the q-table to determine when the q-table was no longer being updated. The performance of the final tuned Q-learning algorithm after simulating 10,000 games is shown in Table 5-1 along with the performance of PI and VI. Q-Learning did not require reward shaping for Blackjack nor did it require altering the parameter decay functions for either alpha or epsilon.

*Table 5-1 Policy Performance for Blackjack*

| Algorithm | Wins | Losses | Ties |
|---|---|---|---|
| Policy Iteration | 4280 (42.8%) | 4753 (47.53%) | 967 (9.67%) |
| Value Iteration | 4187 (41.87%) | 4902 (49.02%) | 911 (9.11%) |
| Q-Learning | 4316 (43.16%) | 4787 (47.87%) | 897 (8.97%) |

All three algorithms were able to effectively find a near optimal policy for playing Blackjack as shown by the win and loss percentages near the theoretical values. No algorithm performed statistically significantly better than the others for the 10,000 simulated games; however, PI and VI train in much shorter time than Q-Learning which solved in approximately 6 seconds. For Q-Learning, the convergence criteria of average and maximum q-value versus iteration is shown in Figure 5-1.

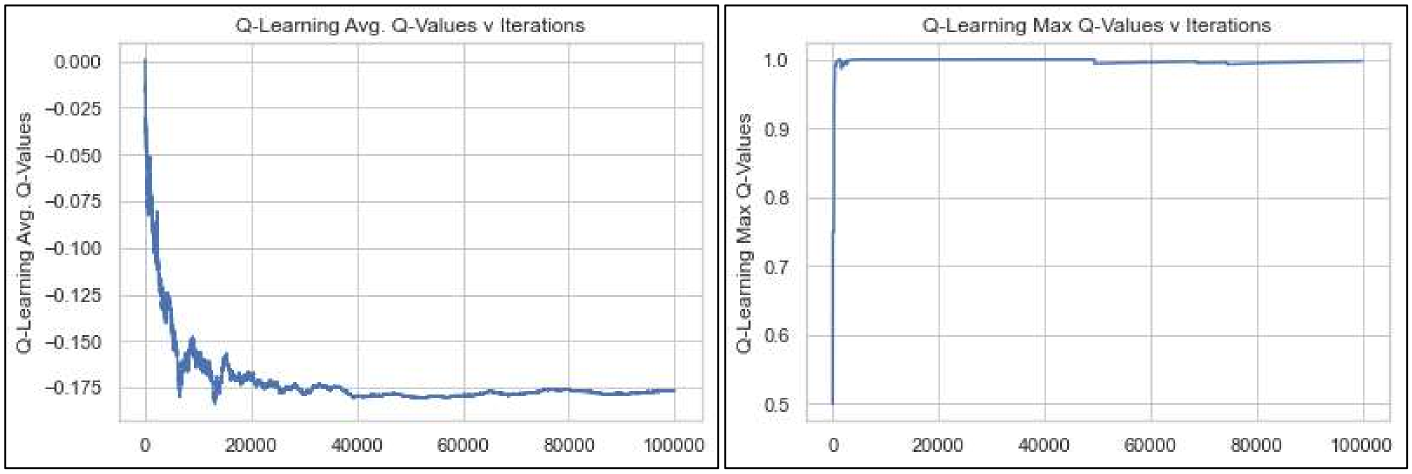

*Figure 5-1 Q-Learning Convergence Criteria Average and Max Q-Value versus iteration for Blackjack*

As shown in Figure 5-1, while the average q-value reached steady state after 80,000 iterations, the maximum q-value reached steady state within a few hundred iterations. However, stopping the Q-Learning algorithm early at 10,000 iterations proved to be inappropriate as the policy after 10,000 iterations given the same inputs won 50% fewer games than shown in Table 5-1 in the simulation than the algorithm's policy after training for 100,000 iterations. Thus, while maximum q-value is important given the reward structure for determining a policy that achieves the goal, an effective policy is found when all state-action pairs in the Q-table are converged. Likewise, testing the resulting policy from both model free and model based algorithms on simulated stochastic game play is important for evaluating the policy effectivity.

## 5.2 Frozen Lake

For the Frozen Lake game, various randomly generated grid worlds were explored for testing these algorithms. As shown in Figure 2-1, a simple large game was initialized to more easily compare resulting policies to the game world to determine effective solutions to the game. This also allowed reproducibility by supplying the algorithms with a random seed. The PI and VI algorithms converged to a policy within a few seconds. Value iteration converged in approximately 1,000 iterations while policy iteration converged within 10 iterations; however, both algorithms took similar amounts of runtime to converge. Both produced policies as shown in Figure 5-2. PI and VI converged to identical policies for this MDP. Minimal parameter tuning was utilized in PI and VI, only the gamma parameter controlling the discount of future rewards was varied to improve converge and policy performance for this game.

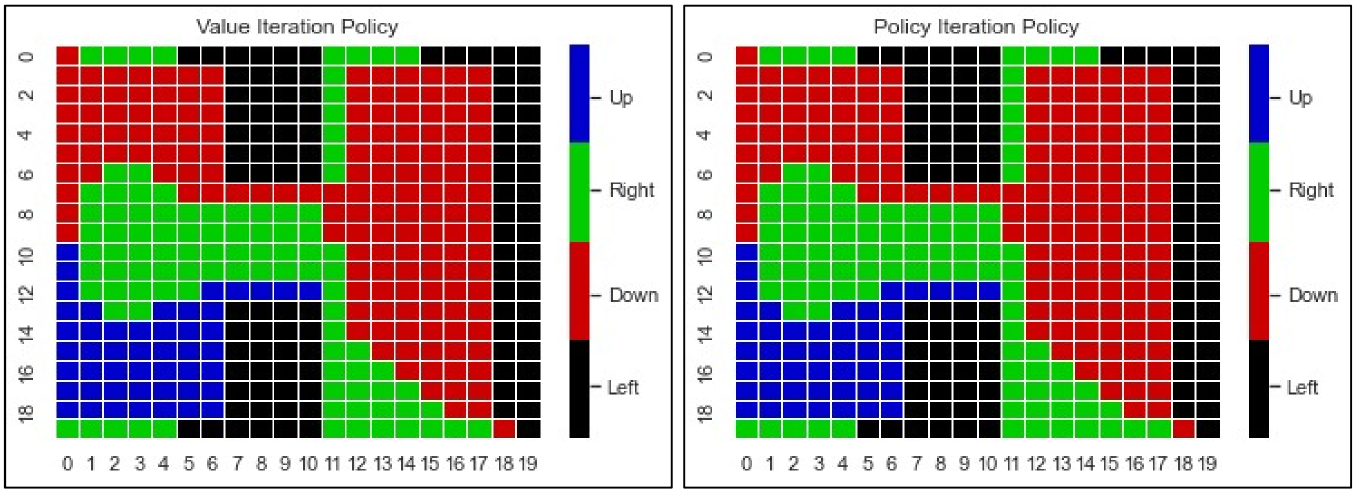

*Figure 5-2 Policy and Value iteration policies for Frozen Lake on 20x20 size world*

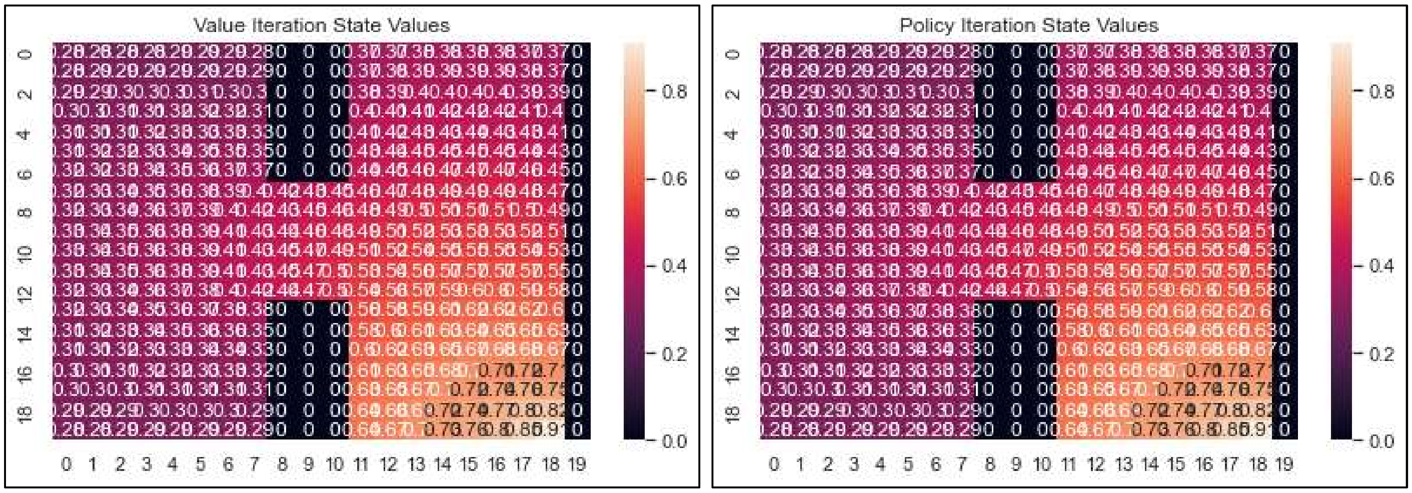

*Figure 5-3 Policy and Value iteration state values for Frozen Lake 20x20*

While PI and VI performed well and converge to policies quickly, these algorithms know the state transition probabilities in advance due to being model based methods. Q-Learning required significant parameter tuning, reward shaping, and decay function research to promote convergence in this large state space with sparse rewards. The Q-Learning implementation for this game required a large grid search of hyper-parameters including varying the initial learning and epsilon rates, the minimum learning and epsilon rates, the function governing their decay, and the discount factor, gamma, applied to future rewards. Additionally, reward shaping was implemented by adding an immediate reward of -0.05 for all moves not resulting in falling through the lake i.e. into a hole or obtaining the goal which promoted fewer steps to the goal. If intermediate rewards were positive, the agent would be encourage to entire an infinite loop gathering rewards and never finishing the game. For those actions resulting in falling in a hole, a reward of -1.0 was given while +1.0 was given for obtaining the goal. This allowed the Q-table state-action q-values to update more frequently during iteration and allowed the agent to progress through the game more effectively in fewer iterations. Additionally, instead of using the default exponential decay function for both the learning parameter, alpha, and random action exploration parameter, epsilon, and varying the exponential ratio, the function used to decay these value was changed to an inverted parabolic function which allowed the epsilon and alpha parameters to remain relatively higher, for more iterations versus exponential decay functions. This exploration-exploitation strategy utilizing a different decay function for epsilon and alpha was found to be more effective for the Frozen Lake MDP than the exponential decay function. The decay functions with initial and final values for both parameters versus each episode for Q-Learning is shown in Figure 5-4.

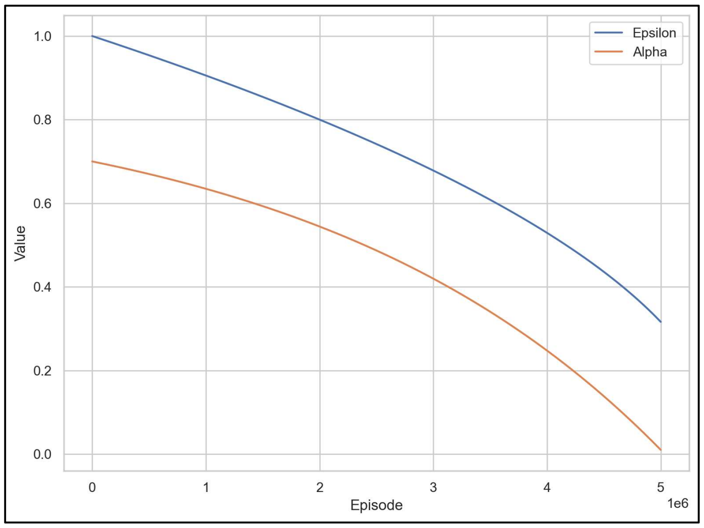

*Figure 5-4 Alpha and Epsilon decay schedule for Q-learning for Frozen Lake 20x20*

While running the grid search, including reward shaping, and changing the decay function for alpha and epsilon all incrementally improved the performance and progress of the Q-learning algorithm, these changes eventually promoted convergence in a sufficiently low number of iterations for this problem. Various Q-table initializations were investigated, including randomly generating small initial q-values for all state-actions as well as assigning a uniform initial q-value. It was determined that using a uniform initialized q-table was effective. Additionally, initializing the q-table with a function to encourage exploration toward the goal would be an introduction of domain knowledge into a model free reinforcement learning algorithm which would no longer be a model free solution. Such an initialization for the Frozen Lake problem could be using a heuristic like from A* search algorithm where initial values are inversely proportional to their Euclidean distance from the goal regardless of whether or not they were holes (H) or frozen (F). Thus, a uniform initialization was used for this analysis. Including the aforementioned adjustments, the Q-Learning algorithm was able to converge to a policy within 5e6 iterations as shown in Figure 5-5 in approximately 30 minutes of runtime. Q-Learning thus took significantly more iterations and longer wall clock runtime than both PI and VI methods. Additionally, the average q-value versus iterations is shown in Figure 5-6. The average q-value versus iterations was used as convergence criteria as the reward structure for the game was altered after reward shaping to include two additional negative rewards during the Q-learning algorithm evolution.

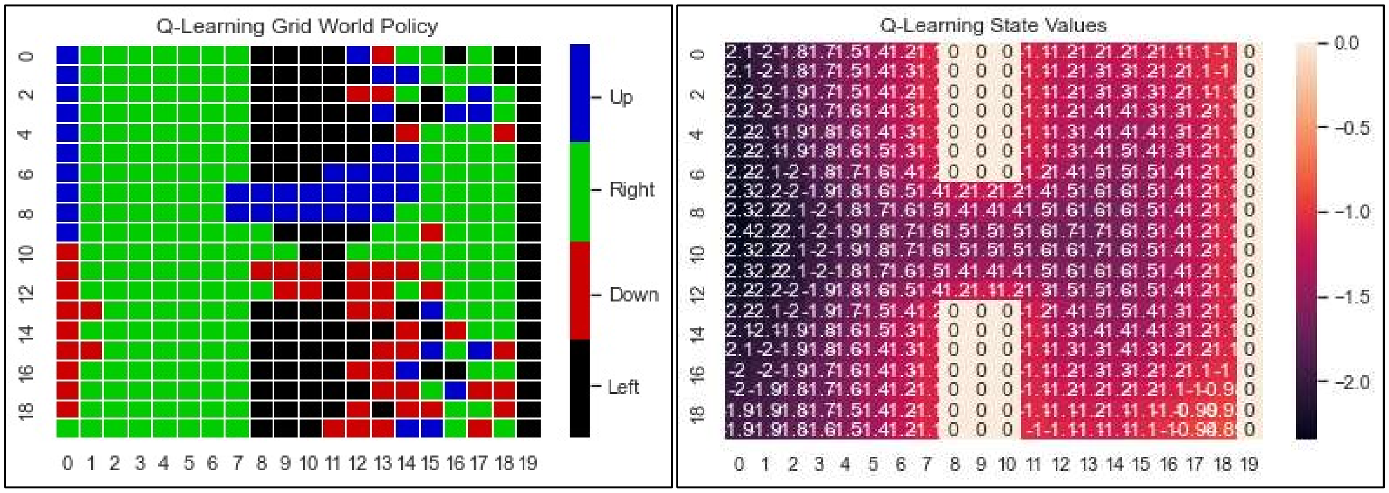

*Figure 5-5 Q-Learning policy and grid values for Frozen Lake 20x20*

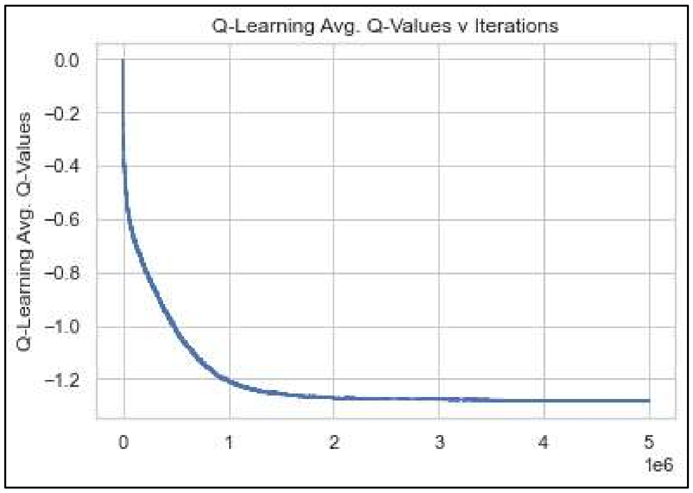

*Figure 5-6 Q-Learning convergence criteria of average q-value in q-table for each iteration for Frozen Lake 20x20*

Similarly to the Blackjack game, 10,000 simulated games of Frozen Lake were run for each policy determined by PI, VI, and Q-learning to determine how effect the policy was at achieving the goal despite the stochastic actions of the game, the results of these simulations are shown in Table 5-2.

*Table 5-2 Policy Performance for Frozen Lake*

| Algorithm | Wins | Losses |
|---|---|---|
| PI | 1882 (18.82%) | 8118 (81.18%) |
| VI | 1882 (18.82%) | 8118 (81.18%) |
| Q-Learning | 0 (0.0%) | 10000 (100.0%) |

Again, PI and VI optimal policies performed substantially better than the Q-learning policy despite Q-learning converging to a steady state q-table. Unfortunately, while the Q-learning algorithm converged, the resulting policy determined by Q-learning was very poor when applied to the stochastic game simulations resulting in no wins. While reward shaping, decay function and parameter tuning aided in the Q-learning algorithm convergence, the policy determined by the model free algorithm deviated significantly from the model based PI and VI algorithms for the Frozen Lake MDP. The size of the state space and stochastic actions of Frozen Lake contributed to poor performance of the Q-learning algorithm relative to PI and VI. Additionally, tuning and optimization could be explored to allow Q-learning to perform well on the large state space of the Frozen Lake game. While Q-Learning performed poorly for the 20x20 grid world, smaller randomly generated game sizes from 4x4 and up were analyzed to determine if the algorithm or game size contribute more to the poor performance of the Q-learning policy. Q-Learning was found to be able to find effective policies that resulted in game win percentages like that of PI and VI when simulating game play for smaller grid worlds. This indicates that while the Q-Learning algorithm converged in the large 20x20 grid, additional tuning is still necessary to obtain an effective policy at convergence and care should be taken to test policies from model free reinforcement learning algorithms on MDPs.

## 6 Summary

This report explored reinforcement learning techniques applied to Markov Decision Processes by comparing the performance of model based algorithms (Policy and Value Iteration) and a model free algorithm (Q-Learning) in determining game playing policies for two stochastic games: Blackjack and Frozen Lake. For both games, the model based algorithms, PI and VI, determined policies significantly faster than Q-Learning. All three algorithms determined near optimal policies for Blackjack; while for Frozen Lake, PI and VI determined policies that performed significantly better than the policy determined by Q-Learning. The PI and VI algorithms, however, require knowing state-action transition probabilities in advance while Q-Learning creates estimates of these transitions through iteration. All algorithms presented in this analysis are confined to discrete state space MDPs, though they can be applied to continuous state space MDPs if effective discretization of the state space is utilized. It was found that while the model free Q-Learning algorithm converges to a policy, it is important to evaluate this policy by simulating game play to determine if they are effective.

## 7 References

[1] E. a. Kissmann, "Number of legal 7 X 6 Connect-Four positions after n plies".

[2] "Playing Blackjack with Machine Learning".

[3] "Bettermdptools".

[4] M. Towers, J. K. Terry and e. al, "Gymnasium: An API standard for single-agent reinforcement learning environments, with popular reference environments and related utilities (formerly Gym)," 2023.

[5] F. Pedregosa and e. al, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[6] G. B. e. al, "OpenAI Gym," 2016.