Optimizing Blackjack Strategy with Reinforcement Learning

Bradford Lin, Enoch Li, Nash Luxsuwong
CS 238: Decision Making Under Uncertainty
Stanford University

Abstract

Blackjack provides an outlet to test traditional reinforcement learning algorithms such as Q-learning. Existing literature already dictates optimal strategies for playing blackjack given different environments, such as finite deck environments and infinite deck environments. We seek to see if we can train a computer to arrive at the same optimal solution using Q-learning. By using OpenAI's infinite deck blackjack environment, we run Q-learning against the basic strategy of blackjack as defined in literature. After 1,000,000 training iterations, we see that our Q-learning algorithm has essentially converged with the basic strategy, and when we compare policy maps, we see that the actions at each given state are nearly identical. Had even more training iterations been run, we could expect to see even more similarity. In the future, we hope to expand our work to run in finite deck environments and compare our Q-learning algorithm against additional strategies such as card counting, which do not work in infinite deck environments.

Blackjack is the most commonly played casino banking game in the world. With one of the lowest house edges of any casino game, Blackjack is very attractive for the player. It can even have positive expected value with advantage play, such as card counting. In this paper, we will use reinforcement learning to find the most optimal Blackjack strategy.

Blackjack Rules

The object of the game is to attempt to beat the dealer by getting a count as close to 21 as possible, without going over 21 (which is known as a "bust"). At the start of the game, each player is dealt two cards face up, while the dealer is dealt one card face up and the other face down. Players only compete against the dealer and not among themselves. Regarding card values, the value of cards 2 through 10 is their numerical value, while face cards (Jack, Queen, and King) are all worth 10. Aces can be worth 1 or 11, depending on what would lead to a better hand. A hand with an ace valued as 11 is called "soft", meaning that the hand will not bust by taking an additional card, since the value of the ace would become 1 to prevent the hand from exceeding 21. Otherwise, the hand is called "hard". A hand's value is the sum of the card values of the player. Players are allowed to draw additional cards to improve their hands by choosing to "hit", or can keep their hand by choosing to "stay". If the player has 21 immediately (an ace and a 10-card), it is called a natural blackjack. The player then wins and receives a bonus unless the dealer also has a natural blackjack, in which case the game is a draw.

Once all the players have completed their hands, it is the dealer's turn, in which the dealer then reveals the hidden card. If all players have either busted or received blackjacks, the dealer's hand will not be completed (since the player automatically loses in the case of a bust, regardless of the dealer's hand). Otherwise, the dealer must hit until the cards total up to 17. At 17 or higher the dealer must stay. Then, the dealer and the player's hand are compared. If the player's hand is higher than the dealer's or if the dealer busts, the player wins. If the dealer's hand is higher, then the dealer wins. If both hands are tied, it is a draw.

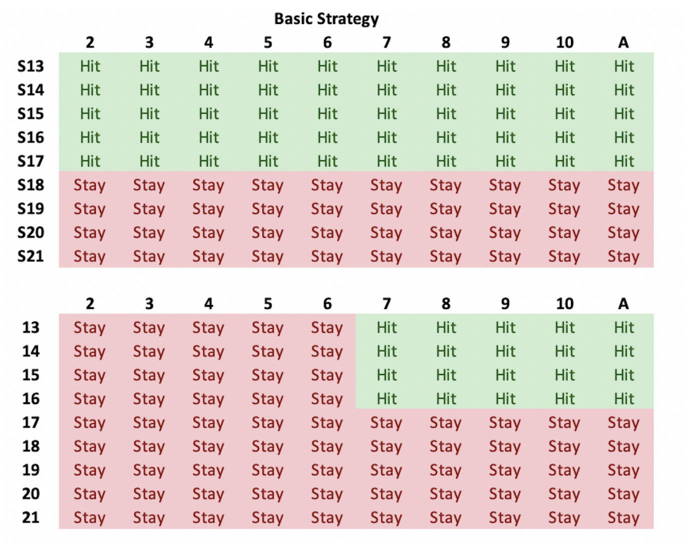
There are over 100 variations of Blackjack, many of which involve additional player actions such as splitting and doubling. However, for simplicity, we will use the aforementioned rules and assume there is just a single player playing against the dealer.

Literature Review

Reinforcement learning has been explored extensively in Blackjack because the game can easily be modeled into an Markov decision process. Due to the probabilistic nature of the game, an optimal strategy, also known as basic strategy, has already been solved for in Blackjack (Figure 1).

Basic strategy is defined as the optimal way for a player to play every hand dealt based off the player's hand and the dealer's upcard. While this is the mathematically correct way to play and lowers the house edge from 8% to 2.2% (when excluding the ability to split or double), the house still maintains its edge due to the fact that players must draw first and that they lose if they bust, regardless of whether the dealer busts or not later (Tamburin, 2020).

This finding is confirmed in a paper from the University of London, where Kakvi (2009) implements a softmax selection agent to play Blackjack. He experiments with different rewards to see how the policy of the agent would change, finding that "altering negative rewards has a larger effect than altering positive rewards" (p. 2). Most importantly though, when looking at the net wins of the different



Basic Strategy

	2	3	4	5	6	7	8	9	10	A
S13	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S14	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S15	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S16	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S17	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S18	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

	2	3	4	5	6	7	8	9	10	A
13	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
14	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
15	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
16	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
17	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
18	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

Figure 1: Basic Strategy

policies, Kakvi (2009) finds that "policy alone cannot win" (p. 2), reflecting the house edge in Blackjack.

Other reinforcement learning approaches have also been explored as well. In a paper from the University of Massachusetts Lowell, Reilly (2012) implements Q-learning and feature extraction to solve for the optimal policy in Blackjack. He finds that both implementations performed relatively equally and achieved an average success rate of 43-44%, comparable to the 44% "average success rate for a competent Blackjack player" (Reilly, 2012, p. 3). However, he notes that Q-learning takes 10 times as many trials to peak compared to feature extraction (10,000 vs. 1000 trials).

Additionally, Granville (2005) from the University of Oklahoma also implements Q-learning in Blackjack. Taking into account additional aspects of the game, such as splitting and doubling down, Granville's implementation is able to handle much more than the simplified version of the game presented earlier. Comparing Q-learning with the basic strategy and a random policy, Granville (2005) finds that Q-learning's performance is initially similar to the random policy's, then asymptotically "approaches the performance of the basic strategy" (p. 3) as the number of training iterations increases to ten million. This convergence is to be expected since "the Q-learning algorithm directly approximates the optimal action-value function $Q^*(s, a)$" (Granville, 2005, p. 3). However, as found in other papers, even the optimal policy still has a negative expected reward and causes players to lose money, albeit the least amount of any policy.

Overall, the findings from these papers, particularly the robustness of the house edge, the number of training iterations needed in Q-learning, and Q-learning's expected convergence to the basic strategy, were instrumental in helping us design our study and know what to expect.

Methodology

We implement a Q-learning algorithm to determine an optimal policy for blackjack. After training, we write up a simulation function that allows us to test different policies and strategies against our blackjack environment. Specifically, we compare our Q-learning policy to a random policy and a policy determined by the basic strategy (Simplified Blackjack Basic Strategy, n.d.). In our basic strategy, if the player has a usable ace, then the player will hit if the player sum is less than 18 and stay if above. If the player does not have a usable ace, the player will hit when the player sum is less than 17 and when the dealer shows an ace or a card that is greater than or equal to 7. In our analysis, we run a simulation on the basic strategy, random strategy, and Q-learning strategy and calculate average score over training iterations from 0 through 100,000. We also create a heat map of our Q-matrix and create a policy map that shows which action to be taken at each state given our Q-matrix.

Environment

We use OpenAI's blackjack environment for the purposes of our testing and algorithmic implementation. Face cards are given a point value of 10. Aces can count as 11 or 1, and if counted as 11, the ace is called usable. OpenAI's blackjack game is played using an infinite deck, meaning cards are drawn with replacement.

In OpenAI's blackjack environment, the reward for winning is +1, the reward for losing is -1, and the reward for a draw is 0. A natural blackjack win, when a player's first two cards are an ace and a ten-card, gains a reward of +1.5, similar to standard casino rules.

Algorithm

We implement a Q-learning algorithm to determine an optimal blackjack strategy. We define our action set as either hitting or staying, and our state array to be a 32 by 11 by 2 array, with 32 corresponding to the number of possibilities for a player's current sum, 11 corresponding to the number of possibilities for the dealer's face up card, and 2 corresponding to whether or not the player has a usable ace. These parameters are defined by the OpenAI blackjack environment.

In our Q-learning algorithm, we choose a learning rate (α) of 0.01 and a discount factor (γ) of 0.15. We chose a small learning rate to allow the model to learn a more optimal solution, even if it meant that training time of the model was longer. Through testing, we found that our learning rate, combined with the rest of our parameters, allowed for our model to converge upon the basic blackjack strategy. We set a discount factor of 0.15 so that we can put more weighting on recent states. For each iteration of learning, our algorithm can take a maximum of 100 steps of exploration.

In our initial training, we implement an epsilon-greedy approach, where we set our initial epsilon to 1 and have it decay over time down to a lower-bound of 0.15. We start at 1 because in the beginning, our Q-matrix is essentially empty, so we instruct the computer to explore. Over time, as the algorithm builds up the Q-matrix, we instruct the computer to exploit the current best option rather than to keep exploring. Rather than keep a constant epsilon, we start at 1 (instructing the computer to explore) and decay our epsilon by 1% each iteration, until hitting a lower bound of 0.15, where the computer will exploit most of the time and do minimal exploring.

We step through our games of Blackjack by calling upon the OpenAI environment, and we update our Q-matrix at each iteration using the following equation:

$$New\ Q = Q + \alpha[r + \gamma max Q' - Q]$$

Here, Q represents the Q-values from the Q-matrix as a function of state s and action a. Furthermore, r represents the reward as a function state s and action a. Finally, Q' represents the maximum expected future reward as a function of the next state from taking the given action.

Results and Discussion

Our first analysis was to determine the performance of the Q-learning policy in relation to a random policy and the basic strategy policy. We run Q-learning on iterations ranging from 1 through 100,000 and we see that our policy's performance quickly converges to that of the basic strategy (Figure 2). This is expected, as the basic strategy is the set of optimal rules such that the player can maximize the amount of money won. In the long run, if we run even more training iterations, our Q-learning strategy should replicate that of the basic strategy (Figure 2).

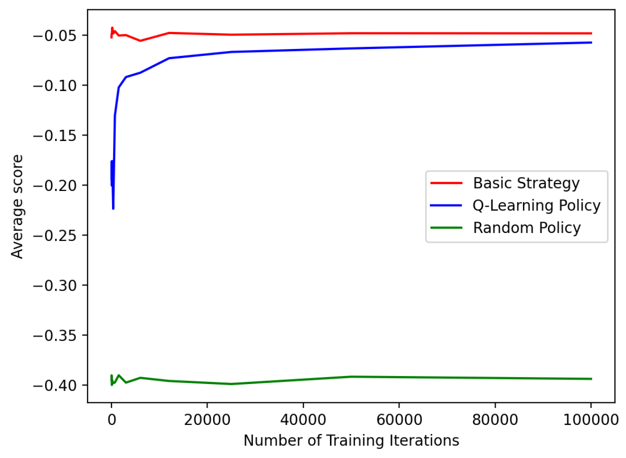


Figure 2: Average Score vs. Number of Training Iterations

In the beginning, we see that Q-learning significantly under-performs the basic strategy because the computer is not yet trained and must do much more exploration. However, by the time the computer runs 20,000 training iterations, it has already improved significantly and begins to converge upon the basic strategy. We expect basic strategy to perform similarly regardless of training iterations because the strategy has already been determined. Moreover, we see that both the Q-learning policy and basic strategy policy significantly outperform the random policy. These conclusions are all in line with what we expect.

After graphing the convergence of our Q-learning policy to that of the basic strategy, we illustrate the actual Q-matrix and heat map of decisions being made at each state. To do so, we run Q-learning for 20,000,000 training iterations and output two heat maps, one for when the player has a usable ace, and one for when the player does not have a usable ace (Figure 3).

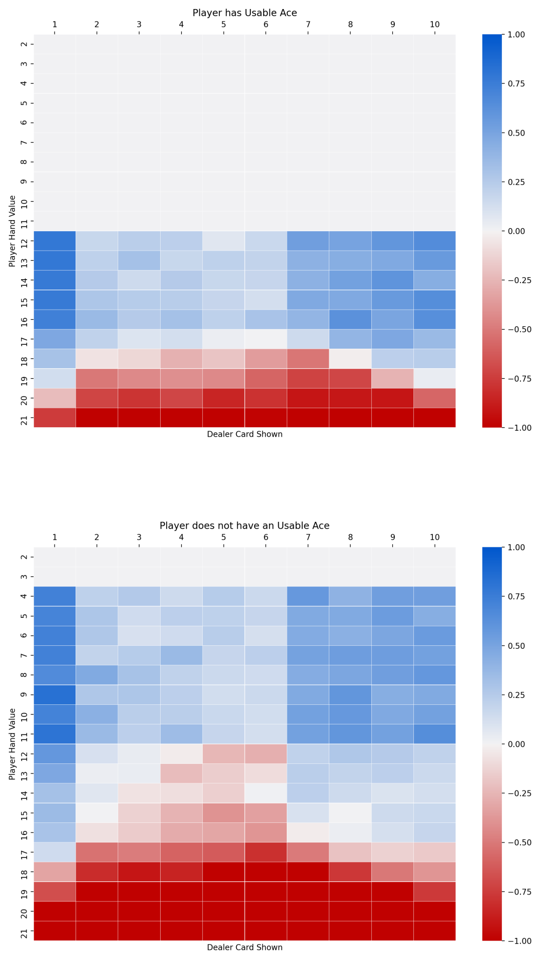


Figure 3: Q-learning Heat Maps

The reason why we choose to run Q-learning with a higher iteration count for the heat map is so that we get more clarity on the computer's decision-making process. That is to say, with more training iterations, the computer is more sure of which action to take at each state. This is depicted visually by having darker colors in the heat map (Figure 3).

Finally, based on our Q-learning policy heat map, we create a policy map that instructs the player what to do in each given state (Figure 4).

From our policy maps, we can really see the convergence

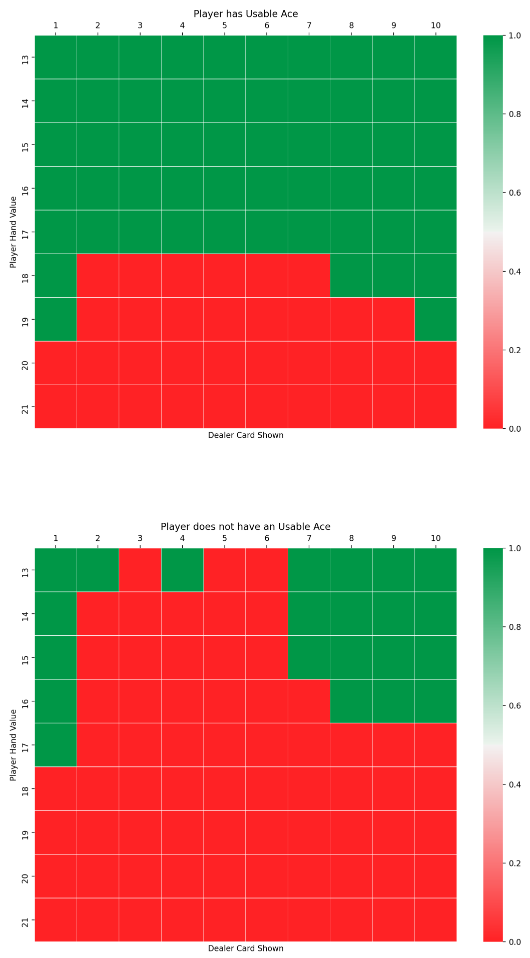
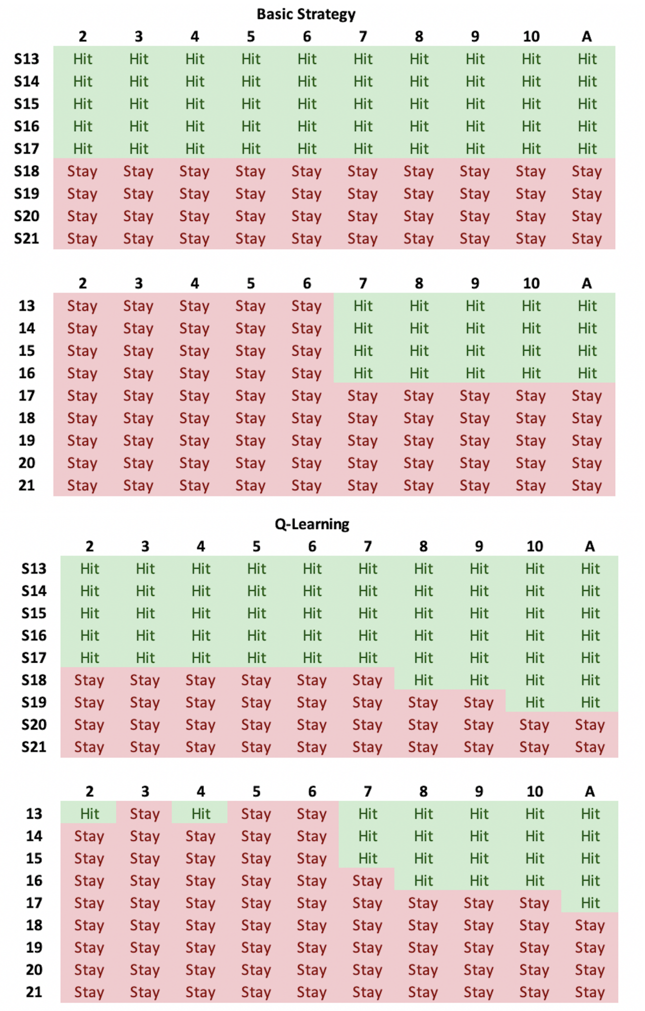


Figure 4: Q-learning Policy Maps

of our Q-learning strategy to the basic strategy. Our final policy map instructs the player to hit whenever the player hand value is less than or equal to 17, assuming the player has a usable ace. In a few instances, when the dealer shows an Ace or an 8, 9, or 10-card, the player is also instructed to hit. These instances slightly stray from the true basic strategy, but with more training iterations, this convergence would be even more obvious (Figure 4).

When the player does not have a usable ace, the Q-learning policy instructs the player to hit whenever the player's hand value is less than or equal to 16 and when the dealer shows a card greater than or equal to seven. There is one anomaly, however, when the player's hand value is 16 and the dealer shows a 7, where the player is instructed to stay (Figure 4). However, aside from this one anomaly, our Q-learning policy instructs the player to play essentially as the basic strategy would. There are a few more edge cases, such as when the dealer shows an ace, that the player is instructed to hit, where the basic strategy would instruct the player to stay, but overall our Q-learning policy's convergence to the basic strategy is striking, as can be seen from our side-by-side comparison of the two policies (Figure 5).



Basic Strategy

	2	3	4	5	6	7	8	9	10	A
S13	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S14	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S15	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S16	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S17	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S18	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

	2	3	4	5	6	7	8	9	10	A
13	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
14	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
15	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
16	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
17	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
18	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

Q-Learning

	2	3	4	5	6	7	8	9	10	A
S13	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S14	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S15	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S16	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S17	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit	Hit
S18	Stay	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit
S19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Hit	Hit
S20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
S21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

	2	3	4	5	6	7	8	9	10	A
13	Hit	Stay	Hit	Stay	Stay	Hit	Hit	Hit	Hit	Hit
14	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
15	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit	Hit
16	Stay	Stay	Stay	Stay	Stay	Stay	Hit	Hit	Hit	Hit
17	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Hit
18	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
19	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
20	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay
21	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay	Stay

Figure 5: Policy Map Comparisons

Future Work

In our current blackjack environment, we use an infinite deck, so positive expected value strategies such as card counting do not work. Since blackjack inherently favors the dealer, we can see that for an infinite deck, the optimal strategy will still yield a negative expected reward, albeit close to the break-even point at 0.

In the future, it would be interesting to run our algorithm on a singular deck, or on four to five decks as is standard in casinos, and compare our Q-learning policy against card counting and the basic strategy. In these situations, it would be interesting to see if our Q-learning policy outperforms the basic strategy and begins to converge upon card counting, or if some other result is obtained. If we were to experiment with card counting and non-infinite decks, we would have to write our own custom blackjack environment, as the OpenAI environment would not support such experimentation.

Summary

After running Q-learning on OpenAI's blackjack environment, our algorithm quickly converges to the optimal strategy given an infinite deck, that of the basic strategy. From our results, we see that our Q-learning algorithm begins converging after around 20,000 training iterations, and by 1,000,000 training iterations, Q-learning and the basic strategy begin to score quite similarly, with a slightly negative expected reward of approximately -0.05. Comparing policy maps from the Q-learning matrix heat map shows almost identical actions between Q-learning and the basic strategy, except for a few discrepancies. With even more training iterations, we would expect the policy maps to become even more similar, and even identical in the long run. In the future, we hope to extend our work on a limited deck, where we can introduce positive expected value strategies, such as card counting, and see how our Q-learning algorithm fares in a scenario with drawing cards without replacement.

Author Contributions

Each group member contributed evenly. Bradford helped write the Q-learning algorithm and contributed to the final report. Nash ran the analyses and created the convergence graphs as well as the heat maps and policy maps. Enoch helped write the blackjack testing simulations and contributed to the final report.

References

Granville, C. d. (2005). *Applying reinforcement learning to blackjack using Q-learning.* University of Oklahoma.

Kakvi, S. A. (2009). *Reinforcement learning for blackjack.* University of London

Reilly, B. (2012). *An MDP blackjack agent.* University of Massachusetts Lowell.

Super-easy simplified blackjack basic strategy chart. http://blackjackcalculation.com/blackjack-super-easy-basic-strat.html

Tamburin, H. (2020). *Blackjack basic strategy.* https://www.888casino.com/blog/blackjack-strategy-guide/basic-blackjack-strategy