Winning Blackjack With Reinforcement Learning

Majd Zayyad[#1], Hamza Abuleil[#2], Ameen Darwish[#3] , Abd Shkerat[#4]

The Hebrew University of Jerusalem

***Abstract*— Blackjack is one of the most popular casino games in the world, and for a long time there have been many strategies devised to attempt to beat the game, or at least give the player a good chance of winning. In this paper we attempt to build reinforcement agents using Value Iterations, and Q-Learning that can develop good strategies for playing the game of Blackjack.**

I. INTRODUCTION

Blackjack is a card game between the player and the dealer. The cards are valued based on the number they display. The face cards (Jack, Queen and King) are counted as 10, and the Ace is a special card that can take the value of 1 or 11, depending on which option is more favourable for the card holder. In this game there are two possible actions: Hit or Stand.
A hit means that the agent chooses to draw a random card from the deck, and a stand means that the agent chooses to halt their turn.

The objective of the game is to get the accumulated value of the player's cards (or the player's hand) closer to 21 than the value of the dealer's cards. However, if the player or the dealer reaches a value greater than 21 then they lose. The dealer plays according to the following rule: keep drawing cards from the deck as long as the value of the hand is under 17, and stand otherwise.

The game of Blackjack is mainly a game of probability, but there are ways of gaining an edge by using information like the hand of the dealer, the hand of the player and the cards already drawn from the deck. Some of these methods of gaining an edge that we will use as a reference are:

Basic Strategy- a list of choices on whether to hit or stand that takes into account the cards the player holds and the dealer's upcard, which gives an average win rate of 42% [1].

Card Counting- a method of representing how favourable a deck is to the player at any given moment based on our knowledge of the cards that had been already dealt out of the deck. This strategy can be used to know which hand has a greater likelihood of winning, and coordinating bets accordingly, and typically provides expected returns of 1%-2.5% [2].

The objective of this project is to develop reinforcement learning agents that can create effective strategies for playing the game. We will compare between a model-based approach using Value Iterations and a model-free approach using Q-Learning, and how much of an improvement we could gain over basic strategy.

II. MODELING

In this section we will expand on our implementation of the algorithms used, and on the way we modeled the problem for the purposes of our research.

We represented the state of the game using the following parameters: 1- the sum of the cards in the player's possession, 2- the dealer's up card (the card that the dealer shows). We use these parameters to model the current state of our game, which we will assign a value to. For the purpose of simplicity we consider two possible actions: Hit and Stand. The game functionality is modeled after the modern casino Blackjack game, which contains the rule of the "soft 17" (meaning that the dealer hits on 17 if in possession of an Ace card), a rule that gives the dealer a 0.2% advantage over classic Blackjack.

The following is a detailed explanation of the algorithms we used and how we implemented them on our problem space:

A. *Value Iteration*

The Value Iteration algorithm uses the states, actions, and the probabilities of states occurring due to various actions in order to find the best policy of action for any given state. We can implement the algorithm with dynamic programming, where we

start with random value assignments for each state and then keep updating until convergence. We can use these values to then find the optimal policy for each state.

We update the value for each state by finding the maximum over all possible states, taking into the account the actions and probabilities, according to the following formula:

$$V(s) = \max_a \sum_{s',r'} p(s', r|s, a)[r + \gamma V(s')]$$

Fig. 1 the value iterations formula

For implementing the Value Iteration algorithm, we built a class to represent the MDP. the class contained methods to get possible states and actions (according to the state we are currently in), and also methods to get probabilities of each state occurring based on the remaining cards in the deck which is the same concept as card counting as we are deducing the probabilities for which cards remain in the deck, based on our knowledge on the cards that have already been seen. Using the MDP class we also built a class for the value iterations, which takes parameters for the number of iterations and the discount factor.

The most difficult challenge that we faced while testing and improving the Value Iteration agent was finding the perfect reward function, as we cannot simply build a reward function that takes into account whether the agent won or lost, as we need to run the formula multiple times per game, before we know the outcome of the game. The best reward function we found was to return -1 if our hand is busted (over 21), 1 if our hand is over 13, and 0 otherwise. With this reward function, we observed that our agent not only gave the highest, but also the most consistent results.

B. *Q-Learning*

Q-Learning is a model free approach to reinforcement learning, where the agent explores the environment, and through exploring random actions it can update the Q-values (which are values assigned to each state) accordingly to learn an optimal strategy. The way the Q-values are updated can be seen in the following formula:

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

Fig. 2 the formula for updating Q-Values

We represented the state of the game using the player's hand, the dealer's upcard, and another parameter that shows whether the state is terminal, which happens when we bust or choose to stand. The major difference between how a state is represented in the Q-Learning approach from the way a state is represented in the Value Iteration approach, is that in Value Iterations, we use our knowledge of the cards that have been dealt out of the deck in order to calculate the probability of drawing another card.

This is the way we calculate the probabilities of reaching the next state from the state we are currently in, so the Value Iteration agent requires some sort of card counting in order to keep track of the cards remaining in the deck.

However, when it comes to the model-free approach of the Q-Learning agent, we do not need to know the probabilities of the next state transitions, we allow the agent to learn to represent them on it's own, so the only information we need of the state of the game at any given time is the player's hand and the dealer's up-card.

The reward function for the Q-Learning agent is also different from that of the Value Iteration agent, in both the way it is calculated, and in when it is used. We observed that giving the agent a reward by the end of each game based on the outcome of the game gives the best results.

After multiple experiments on the most effective way to implement the reward function, we reached the following implementation: the reward function rewards the agent with +1 if it achieves 21 or beats the dealer and pinalizes the agent with a -1 if it busts or gets a result lower than the dealer. Otherwise, the agent gets a reward of 0.

Training the Q-Learning agent proved to be the most tricky part of this implementation, as we had to experiment with multiple different approaches of modeling the training data, and when to update the Q-Values.

The method that brought the most satisfactory results (with a winrate surpassing all other implementations by at least 13%), is to update the Q-Values (according to figure 2) with every new game, and with the reward function described above. Thus, we had to train the agent with live gameplay, which also gave us confidence in its consistency as it gave us similar results over completely randomized training data.

III. RESULTS

In this section we show the results of the testing and analysis we did on both approaches we considered. We tested our results on all possible parameters, and we provide analysis over which had a significant impact on the results and which had a negligible effect.

A. *Value Iteration Results Analysis*

After analysing the win rate of the Value Iteration agent, we discovered that it converges to a 46% win rate, which is higher than that of both card counting and basic strategy combined (being a maximum of 44.5%), this convergence can be observed in figure 3 below this paragraph. In figure 4 we can observe the convergence of the accumulated win rate more closely over the last 10% of gameplay.

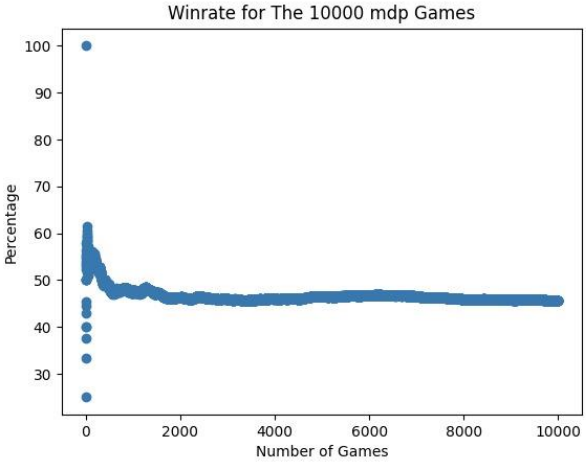

Fig. 3 the win rate of the Value Iteration agent over 10,000 games

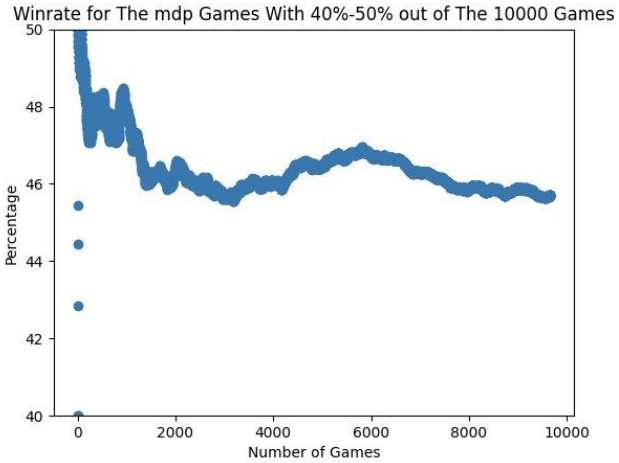

Fig. 4 the win rate of games in the range 40%-50%

We discover that a discount rate of 0.3 gives us a marginal increase in the win rate over the course of 1,000 games, as we can observe in figure 5. We also discovered (as shown in figure 6) that an increase in the number of iterations over 100 has no notable increase on the end results.

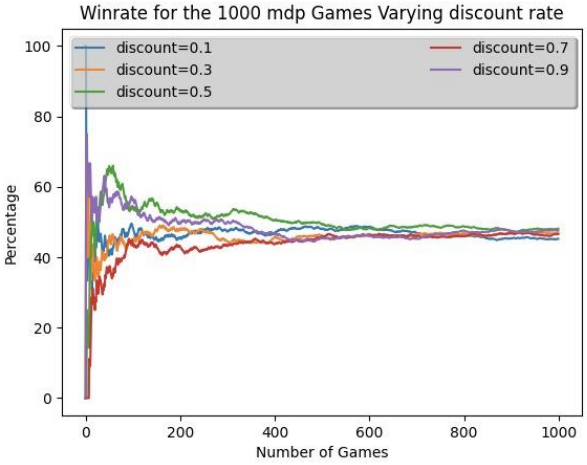

Fig. 5 the effect of the discount rate on the win rate

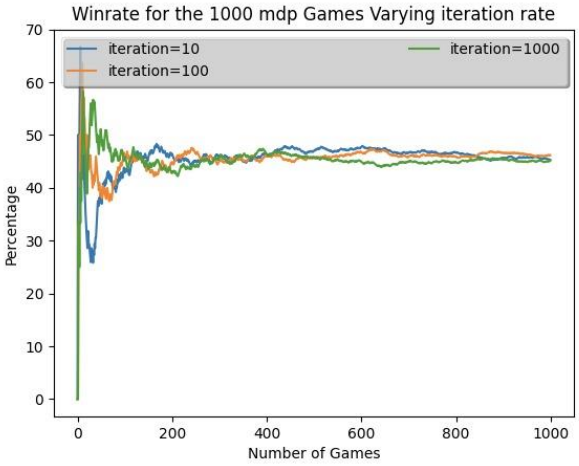

Fig. 6 how the number of iterations effect the win rate

B. *Q-Learning Results Analysis*

The Q-Learning agent converges to a win rate of 44% after testing it on 10,000 iterations as can be seen in figures 7 and 8 down below.

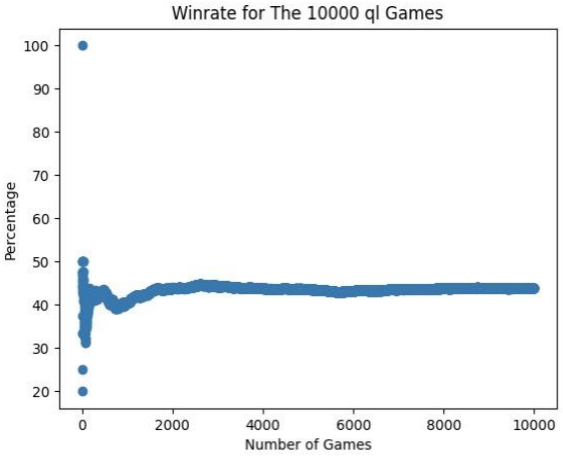

Fig. 7 the win rate of the Q-Learning agent over 10,000 games

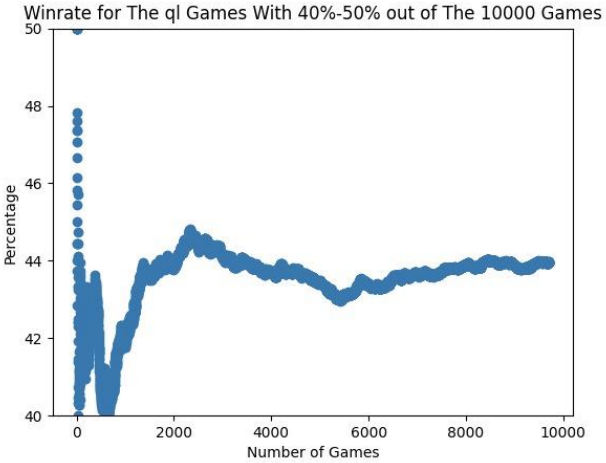

Fig. 8 the win rate of games played in the range 40%-50%

The amount of training contributed significantly to the performance of the agent. We tested the agent's performance over training data ranging from 1,000 games to 1,000,000 games, and we observed (as seen in figure 9) that the agent showed a steady increase in performance up to 100,000 games of training. However, when trained with 1,000,000 games the agent showed regressed results and became over-fitted for the game.

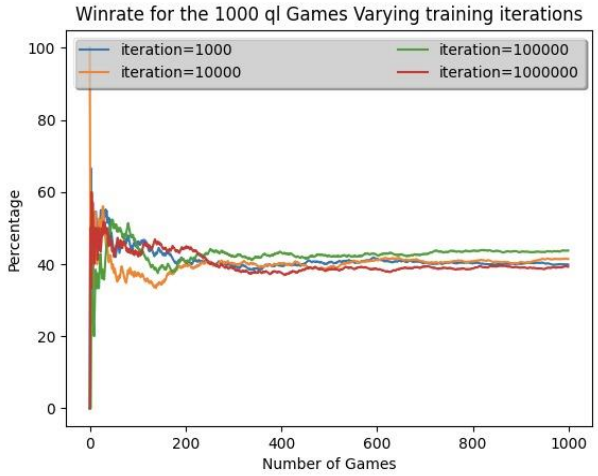

Fig. 9 impact of the amount of games the agent is trained with on win rate

Another significant contributor to the performance of the agent is the learning rate. We observed that the performance of the agent regresses consistently with every increase to the learning rate above 0.1, this can be observed in figure 10. Figure 11 on the other hand shows the opposite. With an increase in the discount rate comes a marginable increase in performance.

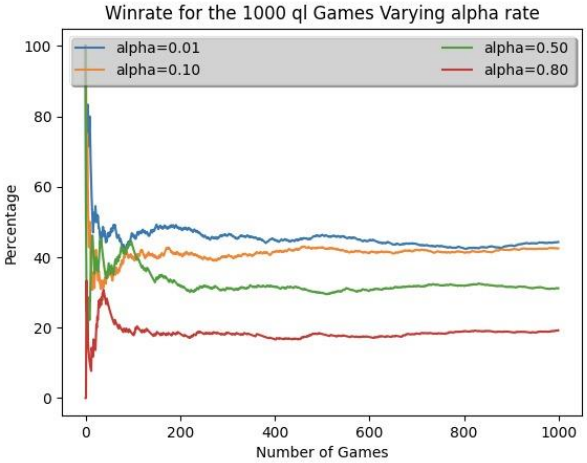

Fig. 10 the impact of the learning rate on the agent's performance

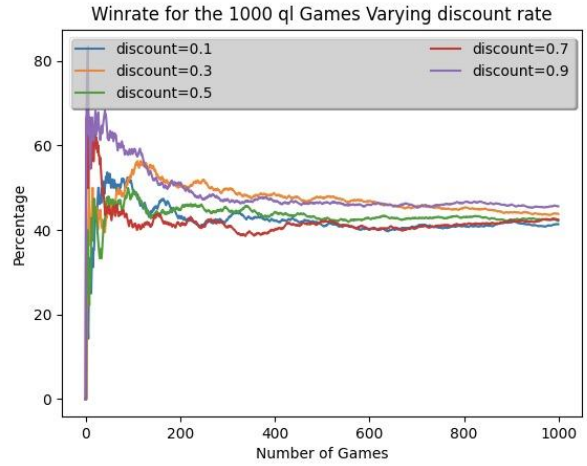

Fig. 11 the discount rate in correlation with the win rate

IV. CONCLUSION

We implemented two different methodologies in order to gain insight and develop strategies for beating the game of Blackjack. The highest results can be reached by using a Value Iteration agent (with a win rate of 46%). However, the Q-Learning agent achieved a win rate (44%) greater than that of basic strategy (42%) without having access to any information about the state of the deck, which allows it to be adopted as a general strategy for Blackjack.

REFERENCES

[1] https://towardsdatascience.com/how-much-will-100-000-win-you-at-blackjack-1c8344885b49

[2] https://en.wikipedia.org/wiki/Card_counting