Optimizing Blackjack Strategies: A Reinforcement Learning Approach

Tharun Kumar Korine Palli
Master Artificial Intelligence, Faculty of Computer Science
Technical University of Applied Sciences Würzburg-Schweinfurt
tharunkumar.korinepalli@study.thws.de

***Abstract*—One of the famous casino games is blackjack, which involves players trying to attempt card values less than 21 while maintaining a total higher than that of the dealer. This paper presents optimal strategies for blackjack using reinforcement learning techniques. This study uses a simulated blackjack environment as a testing ground, where Q-learning and SARSA were implemented and trained. The agents will play many runs and learn progressively from their experience. The results show that reinforcement learning is quite promising in successfully finding ways of dealing with complex decision-making problems such as blackjack. This paper will compare two major techniques: the basic strategy and a complete point count system. A basic strategy agent makes decisions based on set of rules about standard blackjack, and agents using complete point count systems manipulate the play based on the count of high and low cards in the deck. Each agent was trained using both Q-learning and SARSA and then evaluated in the same environment. The results compare the agents winning rates using the complete point count system with those using the basic strategy, demonstrating the effectiveness of RL techniques in developing strategies for blackjack. This study shows the applicability and efficiency of the RL approach in mastering this game by providing the groundwork for future research related to strategic game-playing using artificial intelligence, especially reinforcement learning.**

***Index Terms*—Blackjack, Decision Making, Reinforcement Learning, Machine Learning, Q-learning, SARSA, Basic Strategy, Complete Point Count System, Rule Variations, Artificial Intelligence.**

I. Introduction

Blackjack is a famous game in casinos, specific to both luck and strategic decisions. Therefore, making it quite a good setting to experiment with AI techniques. The primary objective of this study is to apply reinforcement learning techniques known as Q-learning and SARSA. Using these techniques, the agents will learn how to choose actions to interact with their environment to achieve more rewards. Particularly, these techniques are utilized to compute optimal strategies for playing blackjack and maximize long-term expected profit.

A. What exactly is Blackjack and why is it complicated?

In blackjack, each card has a specific point value: cards from 2 to 10 are valued at the face value of the card, face cards (Jacks, Queens, Kings) are valued at 10 points each, and Aces can be worth either 1 or 11 points. A critical distinction in gameplay is between soft hand and hard hand, this depends on whether the Ace is counted as 11 or 1, respectively. Players start by placing bets and receiving two face-up cards, while the dealer gets one face-up card and another face-down card. Players must then decide to hit (draw a new card) or stand (keep their current hand) or split (if two of their cards consist of identical value) or double down (double their bets and draw one final card)[1]. These decisions should be strategic and involve complex probabilistic considerations, which makes blackjack both challenging and well-suited for machine learning approaches.

B. What are the Basic Strategy and Complete Point Count System in Blackjack?

The basic strategy comprises guidelines that advise the player on whether to hit, stand, double down, or split, based on cards in hand and the dealer's visible card, to obtain the best chances of winning[1]. A complete point count system utilizes positive(+), negative(-), and zero values of individual cards to determine or assign a future card. High cards such as 10s, Jacks, Queens, Kings, and Aces are valued at –1. Low cards (2 through 6) are +1, and neutrals (7 through 9) are 0. By monitoring the running total, the players are able to achieve a statistical advantage, betting more when counts are high and less when counts are low[1].

C. What is Reinforcement Learning and how is it helping in solving Blackjack?

Reinforcement Learning (RL) is a type of machine learning wherein agents learn to make decisions through interaction with an environment[2]. Unlike traditional methods of playing blackjack, such as basic strategy rules or counting card techniques, RL does not need computations on the side of the player that needs to be precise and timely. Instead, it dynamically enables an agent to learn and thus improve a strategy over experience. In RL, techniques such as Q-learning and SARSA (State-Action-Reward-State-Action) require a feedback signal to improve the decision-making process of an agent[2]. The environment of the game is very roughly represented as a Markov decision process composed of states, actions, rewards, and state transition rates (transition probabilities)[3]. In this environment, an agent makes decisions that impact rewards and future states. At the same time, the agent needs a policy that maximizes the total amount of rewards over time. Now, the main challenge is to balance the exploration of new actions with the exploitation of known strategies to ensure effective learning. The RL agent will learn optimal strategies through constant interaction with the environment, which adapts to the game dynamics and provides a solution to challenges in blackjack.

Figure (1) illustrates the flowchart for the blackjack environment using reinforcement learning, created by the author

using Lucidchart, detailing the interaction between the agent and the environment through actions, states, and rewards.

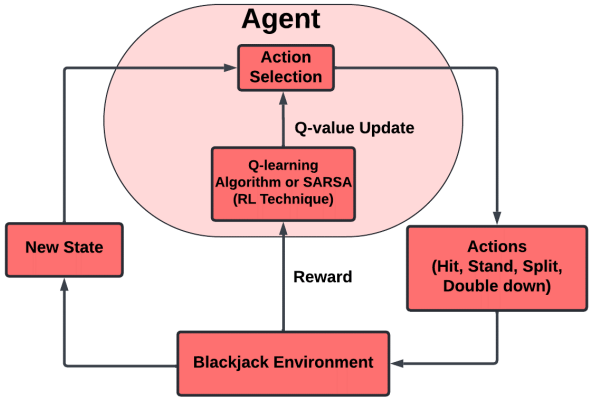

Fig. 1: Process diagram of reinforcement learning for training blackjack.

Despite its potential, using RL in blackjack poses several questions: How should the agent balance the trade-off between exploring new strategies and exploiting established ones in an uncertain environment? What reward structures and state representations are most effective for an agent learning to play blackjack? This paper addresses these questions by implementing Q-learning and SARSA techniques in a simulated blackjack environment, training the agents through extensive gameplay, and evaluating their performance.

D. Structural overview

The structure of this paper is as follows: Section II describes the design of the blackjack environment and details the learning algorithms used. Section III details the experimental design for different scenarios such as random selection, Q-learning, SARSA, with rule variations for two strategies (Basic and Card Counting) and evaluation of all cases showing how accurately those are working. Section IV presents results and analysis of the agent's performance and behavior, offering insights into their decision-making processes after training. Finally, the section V concludes the main findings, discusses their importance, and suggests future research in AI for strategic game-playing.

II. Methodology

This section explains how the basic strategy is implemented without reinforcement learning, how the environment for blackjack is coded, how actions are selected at random, and how the Q-learning and SARSA techniques are implemented to both the basic strategy and the complete point count system.

A. Blackjack Environment

An environment to play blackjack has been implemented with strategies and follows general game rules and rewards are assigned depending on the outcome of each game, aligned with the strategy being tested and the environment does not include the betting process. The environment is initialized using a 52-card standard deck, including four suits: hearts, diamonds, clubs, and spades. Each suit has cards numbered 2–10, face cards (Jack, Queen, and King) rated at 10, and an Ace, which can be worth 1 or 11. At the start of every game, the deck is shuffled to ensure randomness, and a single deck is used until all cards are drawn. At first, both the player and the dealer get two cards. The player's cards are both face-up, whereas the dealer has one face-up, known as the up card, and another face-down, known as the hole card. The player aims to reach a total of 21 or less while trying to win against the dealer. The player can choose different actions. If the player chooses to hit, then the player will get an additional card. If opting to stand, the player keeps the current hand and ends the turn. The double-down action allows the player to double their initial bet and pick up one more card. If the initial two cards have identical values, the player is allowed to separate them, and each will receive one more card[1].

Fixed rules define the dealer's actions. The dealer draws cards until reaching 17 or more. Aces count as 11 unless the total exceeds 21, otherwise, they count as 1. The game ends if the player or dealer busts. The player busts with a hand of 21 and loses immediately. If the dealer's hand exceeds 21, the dealer busts and the player wins. When both parties have an exact 21 on the first two cards, that is called Natural Blackjack. Otherwise, if neither has busted, then both hand values are compared to know the winner, and in the event of a draw, no one wins; this is known as Push[1].

The five essential aspects that would describe the status of the game's environment at any moment are: the value of the player's hand, the value of the dealer's up card, whether the player's hand is a soft hand involving an Ace counted as 11, whether the player's hand has been split, and whether a player has selected the option of doubling down. Actions could be taken on hitting, standing, doubling down, and splitting. Rewards from the environment are based on the outcome of each game. This means a win pays a reward of +1, a loss brings a reward of -1, while a drawing pays a reward of 0. Now, in doubling down to a win or loss, the reward becomes +2 and -2 respectively.

The environment provides three key elements after each action: the next state, the reward, and a boolean indicating if the game is over. The state contains the player's hand value, the dealer's up card value, and whether the player's hand is soft, split, or doubled down. The reward denotes the outcome of the action, while the boolean flag shows if the game has reached a terminal stage. The implemented environment for this detailed and extensive blackjack game provides a strong setting for implementing strategies and testing them using reinforcement learning techniques to see how well they improve gaming. The next subsection tells about playing blackjack without including any reinforcement learning techniques.

B. Playing Blackjack without Reinforcement Learning

Before using reinforcement learning, baseline techniques were developed for comparison. The initial technique involved selecting actions randomly, independent of the player's hand value or the dealer's up card. This provided a control measure to evaluate the improvement offered by more advanced strategies.

Another method was the basic strategy, which was developed from precomputed tables[5] that suggested the best move for each given hand based on the player's hand value and the dealer's up card. The rules were as follows: hit if the player's hand value was 11 or less, stand if the hand value was 17 or more, hit if the dealer's up card was 7 or higher and the player's hand value was between 12 and 16, and stand if the dealer's up card was 4 to 6 and the player's hand value was between 12 and 16. Since these rules are implemented directly without any learning, they serve as a useful benchmark for the effectiveness of reinforcement learning techniques. The next subsection provides an overview of the implementation of Q-learning for basic strategy.

C. Implementing Q-learning for Basic Strategy

Q-learning is an off-policy reinforcement learning technique that trains the agent learning to play blackjack based on Q-table, in which the expected rewards for each action are stored at all states. The agent then acts according to an epsilon-greedy policy to achieve the balance between exploration and exploitation. First, the agent explores by making random decisions on actions but gradually shifts to the well-known actions in the exploitation of what it has learned. While using different rule variations, the 4 actions (hit, stand, double down, and split) are considered. The Q-table serves as the agent's memory, and it will converge to optimal Q-values through iterative updates based on rewards for actions taken. The epsilon value (Exploration rate) is very high at the beginning to get more exploration, then decays with time to have more exploitation[2]. Refer to Figure (2) to know how epsilon values are decaying over time during training. One can observe that epsilon reaches 0.01 at the 550,000th episode after that exploration is confined throughout the remaining training.

Under the rules of basic strategy[5], there are many factors that an agent will consider in determining actions: the value of the player's current hand, the value of the dealer's face-up card, and whether the player's hand is hard or soft, meaning that it either does or does not, have an Ace counted as 11. Here, the value of the player's hand can be any value between 4 and 21, that of the dealer's up card from 2 to 11, and the status of the soft hand can be either true or false. Considering these aspects along with possible actions, the agent must navigate a comprehensive state-action space to determine the optimal strategy by updating the Q-values.

Initially, the simple epsilon greedy policy which is explained in this subsection's first paragraph II-C was used. Once an action is chosen to be executed, the Q-table will be updated based on the new state and reward. The update rule for Q-learning computes the difference between the current Q-value and the newly observed reward for a given state-action pair, plus the discounted future reward estimated by the maximum Q-value in the next state known as temporal difference[4]. To update the Q-table, this difference is multiplied by the alpha (learning rate) and summed to the current Q-value. The Q-values are updated using the following rule[2]:

$$\mathbf{Q}(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s},\mathbf{a}) + \alpha \left[\mathbf{r} + \gamma \max_{\mathbf{a}'} \mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a})\right] \quad (1)$$

where "$Q(s,a)$" is the current Q-value for state s and action a, "α" is the learning rate, "r" is the reward received after taking action a in state s, "γ" is the discount factor, "$\max_{a'} Q(s',a')$" is the maximum estimated future reward for the next state s', and "$[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$" is the temporal difference. The agent was trained over several episodes using this update rule. Equation (1) aims at the iterative process of updating the Q-values based on the observed rewards and transitions. This allows the agent to gradually learn the best policy for balancing exploration and exploitation.

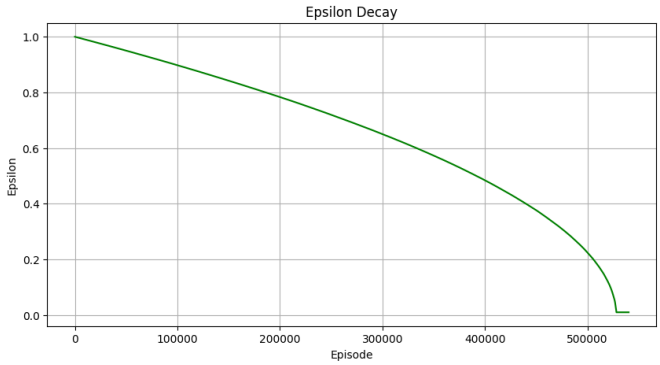

Fig. 2: The epsilon decay in the basic strategy training process using epsilon-greedy. Epsilon starts at 1.0 and decays to 0.01 over 1,500,000 episodes, with the decay occurring in the first 550,000 episodes. After this, exploration is limited to 1%.

In addition to the simple epsilon-greedy approach for the basic strategy, the specific rules for hard and soft hands are included. It means that the Q-learning class has been adjusted to follow these rules. These rules were derived from tables 3.1 and 3.2 in the book[1]. For hard hands (meaning with no Aces or with Aces counted as 1), the agent follows a specific strategy that hits if the player's hand value is 11 or less. When the hand value reaches 17 or more, the agent stands. For hand values between 12 and 16, the agent's action depends on the dealer's up card, the agent will hit if the dealer's up card is 7 or higher, and stand if the dealer's up card is between 4 and 6. In other specific cases, the agent applies the same decision rules[1]. For soft hands, where Aces are counted as 11, the agent's strategy differs slightly. The agent will hit if the player's hand value is 17 or less. If the hand value is 18 and the dealer's up card is either 9 or 10, the agent also chooses to hit. In all other cases, the agent will stand[1]. Then this agent is trained using these adjusted rules for hard and soft hands with the help of the same Q-learning approach. This is how the basic strategy with two different approaches is learning the rules and trying to maximize profits using the Q-learning technique. The next subsection will discuss about SARSA implementation for the basic strategy.

D. Implementing SARSA for Basic Strategy

The SARSA (State-Action-Reward-State-Action) algorithm[4] was implemented and trained which is an on-policy RL technique to enhance the basic strategy for blackjack. SARSA is on-policy because it adjusts its Q-values based on the next state's Q-value and the current policy action. It calculates the return for state-action pairings if the present

policy is followed[4]. The implementation of the SARSA technique for the basic strategy is the same except for the update rule (2). The SARSA update rule is given by[2]:

$$\mathbf{Q}(\mathbf{s},\mathbf{a}) \leftarrow \mathbf{Q}(\mathbf{s},\mathbf{a}) + \alpha\left[\mathbf{r} + \gamma\mathbf{Q}(\mathbf{s}',\mathbf{a}') - \mathbf{Q}(\mathbf{s},\mathbf{a})\right] \quad (2)$$

All variables in equation (2) are explained in the above equation (1). The key difference here is that $Q(s', a')$ represents the Q-value of the next state-action pair, where a' is the action taken in the next state s', instead of $\max_{a'} Q(s', a')$ as in Q-learning. This is how SARSA was implemented by updating Q-values using this rule (2) for the basic strategy. The next subsection will discuss the complete point count system and how the blackjack environment is modified to include it.

E. Integrating the Complete Point Count System into the Blackjack Environment

The environment is changed to include the complete point count system which improved the agent's ability to make strategic decisions. This modification is based on the Hi-Lo card counting method[1], where low cards (2-6) are assigned a count of +1, high cards (10-Ace) a count of -1, and neutral cards (7-9) a count of 0. The total count is updated when cards are dealt with, giving the agent more information for making decisions. In the modified environment (`BlackjackEnvCardCounting`), the `count_hand` function calculates the count for a given hand, while the `calculate_count` function accumulates the counts of the player's and dealer's hands while taking into consideration unseen cards in the deck. This improved state information, which includes the current count, enables the agent to consider deck composition while choosing actions such as hitting, standing, double down, and splitting.

The `take_step` function has been updated to include the card count in the decision-making process. By including the complete point count system, the agent can have access to more strategic information to improve its decision-making provided by card counting. The next subsection explains the Q-learning and SARSA implementation for the card counting technique.

F. Implementing Q-learning and SARSA for the Complete Point Count System

Implementing Q-learning for the complete point count system uses the same Q-learning method as the basic strategy. Both the epsilon-greedy policy and the approach that includes book rules for hard and soft hands[1] for the complete point count system are followed. The modifications to the environment, already explained in subsection II-E, include the card count in the state representation. Initially, the agent used Q-learning with an epsilon-greedy policy, incorporating the card counting information into the state representation. The state now included the player's hand value, the dealer's up card, and the current count and trained the agent using the same Q-learning update rule (1) over several episodes. Refer to Figure (3) to see that the reward changes throughout the training episodes for this complete point count system agent. One can observe that rewards are linearly increasing and fluctuating after reaching certain episodes.

For the next approach, the agent followed the specific rules for hard and soft hands from tables 3.1 and 3.2 of the book[1], similar to the basic strategy. Then the Q-learning agent has adjusted its actions based on the count and the specific rules for hard and soft hands, updating the Q-values accordingly. The agent was trained for this approach as well, ensuring it could use those defined rules effectively.

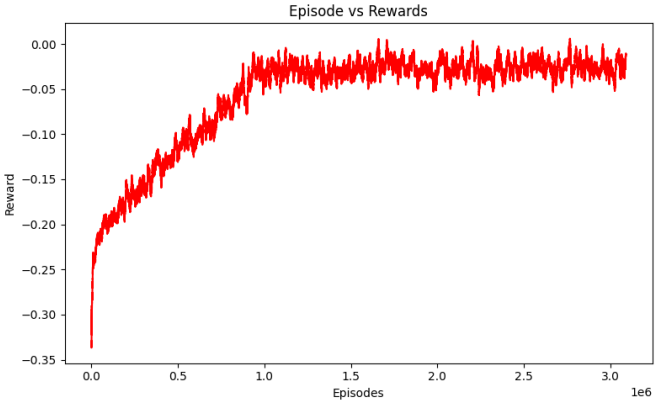

Fig. 3: The reward improvements throughout the training of the complete point count system using a Q-learning simple epsilon-greedy approach across 1,500,000 episodes (where $1e6$ equals 1,000,000 episodes).

For the implementation of SARSA for the complete point count system, Everything is the same except the update rule. This time, the adjusted hard and soft hand rules for the SARSA agent are not used because its on-policy nature needs a consistent policy to learn effectively. Changing these rules would complicate things, slow down learning, and make it less stable. This decision was based on the results obtained from the Q-learning method for card counting. Then again agent was trained over several episodes using the SARSA approach and then Q-values were updated using the SARSA update rule (2). By comparing these approaches, the effectiveness of card counting combined with reinforcement learning in improving the agent's performance in blackjack was evaluated. The experiments and evaluation procedure used will be discussed in detail in the next section.

III. Experiments and Evaluation

This section details the experimental procedure to evaluate the performance of various techniques for playing blackjack. Experiments include randomly selecting action approach and basic strategy rules without reinforcement learning as baselines. Agents were then trained using Q-learning and SARSA for basic and complete point count strategies.

Using Q-learning, the first two agents were trained using the basic strategy with a simple epsilon-greedy policy and then adjusted hard and soft hands rules[1]. Both agents followed the same reward structure: +1 for a win, -1 for a loss, and 0 for a draw. The next two agents were designed using the complete point count system with a straightforward epsilon-greedy strategy, integrating card counting into the state representation, and then for the next agent, the hard and soft hands rules are adjusted. For these agents, the reward system was adjusted to reflect the impact of card counting, giving additional rewards or penalties based on the card count as described clearly in the section II.

Then, using SARSA, two more agents were trained for both the basic and complete point count systems, without applying adjusted rules for hard and soft hands. All the agents were trained and experimented over different sets of episodes such as 100000, 1000000, and 1500000. Also, the experiments were conducted with different alpha values to understand how fast or slow the algorithm learns and how the number of episodes and learning rate affect the final winning rate. Through the training process, each agent adapted its strategy based on the rewards and the specific rules it followed. After the training, the agents' performance was evaluated over 10,000 games by tracking how many wins, losses, and draw counts to find the winning, losing, and draw rates. The next section will describe the results from these various strategies and insights gained from these experiments, focusing on the impact of a number of episodes on winning rates.

IV. Results and Discussion

In this section, the detailed results of each strategy are discussed. The experiments gave useful insights into the performance of different blackjack strategies, particularly the advantage of using reinforcement learning with strategic adjustments like card counting.

A. Outcomes without Reinforcement Learning

Initially, the randomly selected action approach was evaluated. The decisions are randomly made without prior knowledge. As expected, this performed poorly, with a winning rate of 28%, a losing rate of 68%, and a draw rate of 4%. This highlighted the necessity for other sophisticated methods to get good win rates. Then the basic strategy without reinforcement learning was evaluated. The basic strategy outperformed the random agent, demonstrating a winning rate of 42.5%, a losing rate of 49.5%, and a draw rate of 8%. However, unlike reinforcement learning methods, this basic strategy does not learn over time, making it less effective.

B. Outcomes with Reinforcement Learning

Here, the performance of the different reinforcement learning agents is evaluated. Firstly Q-learning, the different strategies with various numbers of episodes, such as 100000, 1000000, and 1500000 are trained, and evaluated in 10000 games to find the best winning rates for each strategy. Additionally, the different alpha values of 0.01, 0.1, and 0.5 are experimented. An alpha of 0.1 worked best. A lower value (0.01) made learning too slow and computationally inefficient, while an alpha of 0.5 gave poor results. Therefore, the values in Table(I) are based on an alpha of 0.1, showing the best win rates for the simple epsilon-greedy, and the values in Table(II) are for the epsilon-greedy with changed rules for hard and soft hands as per book[1].

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Basic	1,000,000	41.94%	48.83%	9.23%
Card Counting	1,500,000	**44.53%**	46.14%	9.86%

TABLE I: Accuracy of agents for playing blackjack using refined best policy with Q-learning - simple epsilon-greedy at best numbers of episodes for different strategies.

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Basic	1,500,000	43.52%	47.93%	8.55%
Card Counting	1,500,000	40.58%	54.07%	5.35%

TABLE II: Accuracy of agents for playing blackjack using refined best policy with Q-learning - adjusted rules for hard and soft hands at best numbers of episodes for different strategies.

Strategy	Episodes	Win Rate	Loss Rate	Draw Rate
Basic	1,000,000	42.33%	51.87%	5.8%
Card Counting	1,500,000	43.61%	46.72%	9.67%

TABLE III: Accuracy of agents for playing blackjack using refined best policy with SARSA at best numbers of episodes for different strategies.

Then for SARSA agent, the same sets of episodes as stated for Q-learning have been trained and evaluated for 10000 games. Therefore, the values in Table(III) are based on an alpha of 0.1, showing the best win rates for the SARSA agent.

However, all agents failed to win the game because none achieved more than a 50% winning rate. Each agent's outcomes will be discussed separately. Firstly, the basic strategy using Q-learning(simple epsilon-greedy) showed some improvement over random selection, balancing exploration, and exploitation. Although this strategy allowed the agent to learn and adapt to the game, it still fell short of achieving a winning rate above 50%. The agent adapted to some extent but could not overcome the inherent house edge in blackjack. Next, the basic strategy with epsilon-greedy, including rule adjustments for hard and soft hands, demonstrated a slight improvement over the first agent. By incorporating specific rules from the book[1], the agent made more refined decisions, leading to better performance but failed to achieve more than 50% win rate. The basic strategy with SARSA learning performed less effectively than the previous Q-learning and failed to achieve the desired win rate.

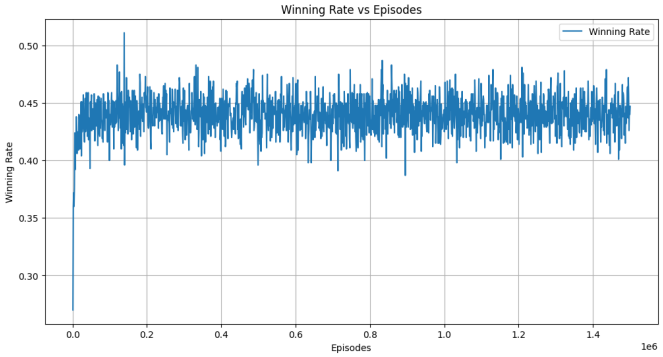

Fig. 4: The winning rate changes throughout the training of the complete point count system using a Q-learning (epsilon-greedy) approach across 1,500,000 episodes (where $1e6$ equals 1,000,000 episodes).

The complete point count system with a Q-learning (simple epsilon-greedy) showed a noticeable enhancement in performance. By comparing all other agents, this agent performed the best, achieving a **44.53%** winning rate. Figure (4) illustrates the winning rates against the number of episodes for this agent. One can observe that the winning rates fluctuate between 43% to 45% and are increasing over episodes.

Figures (5 and 6) illustrate the state values and policy grids using this agent. These plots help in understanding how the

agent makes decisions and adapts to the game dynamics. Figure (5) shows the results for soft hands, where the Ace is counted as 11. The 3D plot on the left displays the state values, representing the rewards for various combinations of player sums and dealer-showing cards. The policy grid on the right visualizes the optimal actions based on the player's hand value and the dealer's up card.

Figure (6) shows the results for hard hands, where the Ace is counted as 1. The 3D plot on the left displays the state values, and the policy grid on the right shows the optimal actions. These graphs show how the agent's approach is improved by adding card counting, making better decisions, and achieving higher performance. The agent adapted to some extent but could not overcome the inherent house edge in blackjack.

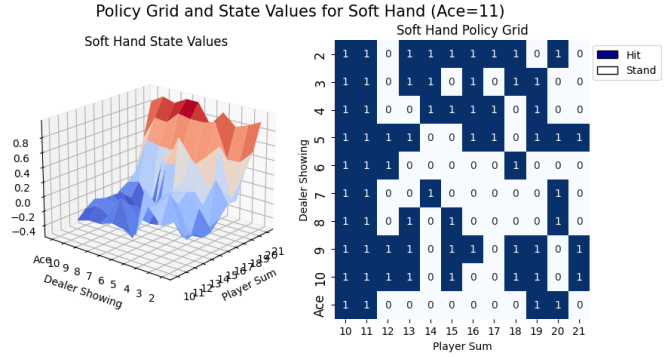

Fig. 5: The optimal policy grid and state values for a soft hand (Ace=11) for a complete point count system using a Q-learning (simple epsilon-greedy) approach on the z-axis. The left plot represents the state values, while the right plot displays the policy grid, indicating when to hit or stand based.

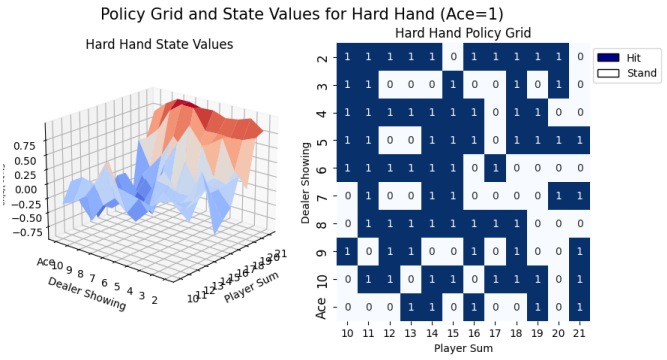

Fig. 6: The optimal policy grid and state values for a hard hand (Ace=1) for complete point count system using a Q-learning (simple epsilon-greedy) approach on the z-axis. The left plot represents the state values, while the right plot displays the policy grid, indicating when to hit or stand based.

Then, the complete point count system with epsilon-greedy and adjusted rules for hard and soft hands combined advanced strategies and all blackjack rules. But with this much effort, this agent performed worse than the previous agent (Simple Card Counting). This outcome shows that, while combining strategies can be beneficial, the increased complexity of the game might affect the overall performance. That is why the hard and soft special rules were not adjusted for the SARSA technique.

Lastly, the SARSA agent is benefiting the complete point count system because the win rates are improving as compared to the basic strategy but it has failed to outperform the Q-learning.

C. Agent Adaptation to Rule Variations

The agent's strategies change based on the rules of the environment. For example, when the complete point count system was added, the agent improved its decision-making and achieved a higher winning rate compared to using the basic strategy. Changes to specific rules, like the dealer standing on a 'soft 17', also affected the agent's behavior. Initially, the agent balanced its risks under the standard rules, but it adjusted its strategy when the rules changed. These rule variations showed that the agent could adapt its strategy to different conditions, using extra information like card counting to make better decisions and improve its overall performance. The next section concludes the whole work and suggestions for further research.

V. Conclusion

In this study, reinforcement learning techniques, specifically Q-learning and SARSA, were applied to both the basic strategy and complete point count system to improve the winning rate in blackjack under different rule variations. The complete point count system outperformed the basic strategy by a 2% winning rate, showing the advantage of including card counting in the agent's decision-making process. But also observed that adjusting hard and soft hand rules provided significant benefits for the basic strategy but did not for the complete point count system. Furthermore, Q-learning proved to be more effective than SARSA in optimizing the agent's performance. However, despite these advancements, none of the agents surpassed a 50% winning rate, indicating that there is still room for improvement and refinement in the strategies employed to win against the dealer.

Future work should use advanced RL techniques such as Actor-Critic methods and Deep Q-networks to enhance performance further. Incorporating Monte Carlo Tree Search (MCTS) methods could systematically explore the decision space and find more optimal strategies. Further research on the effects of real-time learning in which the agent continuously adjusts while playing may also result in notable advancements. The entire potential of RL to gain consistently better performance in blackjack and other difficult decision-making tasks will be revealed with the aid of this ongoing study.

References

[1] E. O. Thorp, *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. New York, NY, USA: Vintage, 1966.

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[3] R. Srinivasaiah, V. G. Biju, S. K. Jankatti, R. H. Channegowda, and N. S. Jinachandra, "Reinforcement learning strategies using Monte-Carlo to solve the blackjack problem," *Int. J. Elec. Comput. Eng.*, vol. 14, no. 1, pp. 904-910, Feb. 2024, doi: 10.11591/ijece.v14i1.pp904-910.

[4] A. Perez-Uribe and E. Sanchez, "Blackjack as a Test Bed for Learning Strategies in Neural Networks," in *Proc. IEEE Int. Joint Conf. Neural Networks (IJCNN)*, 1998, doi: 10.1109/IJCNN.1998.687170.

[5] X. Cai, "The House Edge and the Basic Strategy of Blackjack," *SHS Web of Conferences*, vol. 148, 2022, doi: 10.1051/shsconf/202214803038.

APPENDIX A - ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Prof. Dr. Frank Deinzer**. His lectures on the "Reasoning and Decision Making under Uncertainty" module have been instrumental in helping me complete this portfolio and write this scientific paper successfully.

I am also grateful to my university (**THWS**) for providing access to all the research resources, enabling me to read and reference various scholarly articles and papers for free.

I extend my thanks to the various tools that facilitated my work: **Lucidchart** for visualizing process diagrams; **Overleaf** for streamlining the writing and formatting of my paper in LaTeX; **Grammarly** for grammar checks; and **QuillBot** for paraphrasing my words. These tools have greatly enhanced the clarity, presentation, and professionalism of my paper.

APPENDIX B - GITHUB REPOSITORY

You can access the complete implementation and all the plots for every agent trained in the project "*Optimizing Blackjack Strategies: A Reinforcement Learning Approach*" through this repository: https://github.com/tharun-kumar-22/BlackJack_RL_methods.

APPENDIX C - COLOR CODING EXPLANATION

In this document, I have used specific color codes to enhance readability and reference different types of text:

1) **Blue colored text** indicates directions or references to other sections within this paper.
2) **Red colored brackets with numbers** indicate cited references, making it easier to locate them in the references section.