Q-learning in Blackjack: Evaluating Basic Strategy and Complete Point Counting with Modified Rules

Preeti Venkataraman Hegde
Faculty of Computer Science
Technical University of Applied Sciences Würzburg-Schweinfurt
Würzburg, Germany
preetivenkataraman.hegde@study.thws.de

***Abstract*—Adapting to the dynamic rules of Blackjack, a popular classic card game played in casinos, often requires advanced strategies. This paper focuses on one such strategy in reinforcement learning that can significantly impact strategic play. The key objective of this paper is to introduce the reinforcement learning algorithm, the Q-learning algorithm to play Blackjack efficiently by adapting itself to the rules. A Blackjack environment is implemented with Basic Strategies, and Complete Point Counting techniques embedded with specific Rule Variations, such as 6-to-5 Blackjack Payout and Prohibition of double down when hard 11. The performance of Q-learning is tested and compared against multiple scenarios like a random selection of an action, and a traditional method of game. The study also examines, how the changes in the rules can impact a player's chances of winning or losing the game. The experimental results showcase the remarkable improvement in its strategy, flexibility, and efficiency achieving higher win rates and better decision-making under varying conditions. The core findings demonstrate the potential feature of Q-learning its ability to flexibly adapt to the environment.**

***Index Terms*—Reinforcement learning, Q-learning, Blackjack, Basic Strategy, Complete Point Counting, 6-to-5 Blackjack Payout, Prohibition of double down when hard 11.**

I. Introduction

Blackjack is one of the most widely played card games in casinos. Due to its probabilistic nature and a wide range of possibilities and policies, Blackjack always remains an extensive research topic for any Artificial Intelligence (AI) researcher. Sequential decision-making skills are key for Blackjack players to gain more rewards. With this in mind, reinforcement learning becomes a more sensible choice. The foremost objective of this research is to integrate a Q-learning algorithm to learn the most advantageous moves of the Blackjack game.

A. The Blackjack Game

A Blackjack game can typically have one to seven players playing along with a dealer. The game's purpose is to get a card value of 21 for a player to win, or more than the dealer's card value. The notable point is if the player exceeds value 21 then it is called busted and the player loses the game. Each player competes with the dealer and not with the other players present in the game [2].

The game can typically contain a single deck of 52 cards, or double deck of 104 cards, or multiple decks namely 6 or 8. When there are multiple decks house edge increases (a statistical advantage for casinos) as the probability of getting natural Blackjack for the players decreases, in addition, it increases the complexity of maintaining true and running count. The values of the card are the same as the numerical values on the card, the face cards 'Jack', 'Queen', and 'King' are valued at 10 and an 'Ace' can have either 1 or 11 depends upon the player. It is called a hard hand if the player has no 'Ace' or if the value of 'Ace' is 1. If 'Ace' is considered to be 11 it is called a soft hand. At the start of the game, each player bets a certain amount with the dealer. The dealer shuffles the card and distributes exactly two cards for each player and himself. The dealer card is one face up called an upcard, and another face down called a hole card. When the sum of the two cards of the player at the start of the game is 21, that is if a player gets an 'Ace' and 10 valued cards either of '10', 'Jack', 'Queen', or 'King' it is considered as Blackjack and player wins 3:2 times of his bet. For Instance, if the player's initial bet was $10 and has a Blackjack, the player receives $15 as a winning reward.

Once the player gets the initial two cards, the player can perform various actions such as 'hit', 'stand', 'double down', and 'split'. If the player chooses 'hit', the player gets another card which will alter the total card values. The player can hit until better card values are observed to win the bet. Once the player thinks not to have any more cards the action 'stand' is chosen. The player can also choose to 'stand' at the initial stage. Once the player picks to 'stand', it is not allowed to draw more cards from the deck. Now it is the dealer's turn to 'hit' or 'stand'. Once they both finish their moves, the winner is decided. If the total card value of the player is more than the dealer without exceeding 21 then the player wins the bet and receives the 1:1 amount that is, the original bet back, plus the winning bet which is the same as the original bet. For example, if the player's initial bet was $10 and if they win they receive $10 + $10 total of $20 as the winning bet. If the player loses in the game, they lose the full bet which is in this example, the player will receive $0. If it is a tie then the player receives the original bet back, as per the above example, they receive $10 back.

The player can also decide to double down at the start of the game. When double down is preferred, the player places a double down bet which is equal to the original bet and they can draw exactly one card from the deck and should stand.

If the player wins then they receive a 1:1 winning reward. If the player's initial bet was \$10 and the double down bet is \$10 after winning, the received bet is \$20 (original + double down bet) + \$20 (winning reward) total of \$40. If the player loses in the game, both winning and double-down bet is lost which in this example, the player will receive \$0. If it is a tie then the player receives the original and doubles down the bet back as per the above example, \$20 is got back.

The player can also opt to split at the beginning of the game, if the player has two identical cards, like '8 of Spades' and '8 of Hearts'. When cards are split an additional bet equal to the original bet is placed and each card becomes an independent start of a new hand. The player receives 2 new cards from the deck one for each hand, the each hand is played against the normal rules. It is important to know that both hands are not dependent on each other. The resulting outcomes are as follows: if the initial bet and split bet combined are \$20, if one hand wins, one loses and receives \$20 for the winning hand and \$0 for the losing hand, if both hands lose \$0 if both the hands win \$20 for each hand is received. If both hands are tie, a combined bet of \$20 is received back.

The Complete Point Counting System is a powerful strategy that determines whether the next hand is likely to favor the dealer or the player. The main idea of card counting is to keep track of visible cards this is done by a high-low system where low cards ('2', '3', '4', '5', '6') are assigned a positive value 1, neutral cards ('7', '8', '9') are assigned a value 0, and high cards ('10' and 'face cards') are assigned a negative 1. Whenever the player sees a visible card they assign these numbers and count the total, called the running count. The positive running count indicates high values are remaining in the deck which is favorable for the player, whereas the negative running count favors the dealer. The track of the true count, which is the running count divided by the number of remaining decks is kept. This data will give an idea of what kind of cards are remaining in the deck. These running and true counts have a major influence on figuring out the most efficient move which impacts the player's decision-making and winning or losing of a bet.

The strategies and rules discussed above are only a portion of the extensive content found from [2]. There are additional methods and rules which can be found in [2]. This paper highlights the basic rules, strategies, and a few advanced techniques. This knowledge is sufficient for any player or a reinforcement learning agent to step into the journey of Blackjack.

B. Q-learning

Reinforcement learning is a type of machine learning where a decision is made by an agent to maximize the cumulative rewards. The agent uses the feedback from the actions and rewards gained for that particular action to improve its performance. Q-learning is an off-policy Temporal Difference (TD) control algorithm that learns based on state-action pairs. It enables the agent to learn the best and optimal moves for any given finite Markov Decision Process (MDP) even when it does not have a prior knowledge of system dynamics. This is because of its model-free nature which is versatile and robust capable of handling complex and stochastic environments [1] [3].

Algorithm 1 Q-learning (off-policy TD control) [1]

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
for each episode **do**
  Initialize S
  for each step of episode **do**
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe reward R, next state S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$
    $S \leftarrow S'$
  end for
end for

The Q-learning algorithm defined above [1] provides a brief picture of how Q-learning works. It starts by assigning Q-values arbitrarily for all the state-action pairs. At every step, the agent picks an action using the greedy-ϵ method. Greedy-ϵ policy is often used in reinforcement learning to balance exploration and exploitation. Based on the action chosen by the agent rewards are acquired and transit to a new state. The obtained current state, reward, and new states are updated in the Q-table using the Bellman equation. This process continues until the agent converges to the optimal Q-values with which it forms optimal strategies.

The layout of the paper is divided into the following sections: The Introduction section briefly introduces the Blackjack game and an understanding of Q-learning. The Methodology section talks about, how the environment of the game is simulated and the interaction between the agent and the game. This section enumerates sufficient details for any other researcher to replicate the same. The Experiment section focuses on the experimental setup and parameters used for testing the system. The Results and Discussion section gives insights about the outcomes of the experiment. The Conclusion section offers a concise summary of the paper and suggests the areas of research for the future.

II. Methodology

This section focuses on the implementation of a Blackjack environment, induction Q-learning algorithm, and integration of various rules.

A. Random Selection of an Action

This approach serves as a benchmark to point out the efficacy of more strategic approaches namely the Q-learning agent, Basic Strategy, and Complete Point Counting System. For this approach, the agent chooses actions randomly without considering any information about the current state or values

learned from the Q-table. Later, this method is compared against a more sophisticated decision-making model which includes other aspects of decision-making rules.

B. The Blackjack Environment

The Blackjack environment is implemented using the basic Object-Oriented Programming concept in Python. A class *Blackjack(object)* is defined which encapsulates all the functions and logic that are critical for building an environment equivalent to traditional rules of Blackjack. The Actual game is mimicked in the environment as in the real game of Blackjack the players place a bet and the deck of cards will be shuffled. The dealer distributes 2 cards to each player and himself. Then, players and dealers start playing as per the rules and decide the winning or losing bet. Similarly, in the Blackjack environment, the function *create_card()* is used for creating a deck of 52 cards, and shuffling the cards, and the *get_card()* function is used for allocating the topmost card to the player and the dealer. The *reset()* function resets to a new game where it re-initializes the game state and distributes exactly 2 cards to the player and the dealer. The Environment is handled to maintain one deck until all the cards in the deck are empty. However, it is important to note that here, no bets are placed instead, the system receives a series of rewards for every win or loss. It receives positive rewards for winning situations, negative rewards if the players lose the game and if it is a draw the reward will be 0.

Basic Strategy: The Environment has a function that aims to understand the rules of basic strategies which support actions like 'hit', 'stand', 'double down', and 'split'. Functions like *hit_or_stand()*, *double_down()*, *split_pairs()*, and *general_rules()* are defined to handle all the rules of Basic Strategy such as when there are the identical cards the agent can either split or continue. If the agent chooses to split the system performs the split operation based on the rules defined in book [2]. Similarly, if the agent chooses to double down it will draw a new card and then stand and wait for the dealer to finish its move based on the card values of the dealer and agent the winner is decided and rewarded accordingly. The action is hit when the agent has lower cards, and the agent will draw another card from the deck to reach to get the card value 21 or have a higher value card than the dealer without busting. The agent will stand when it thinks it might bust out if it draws another card from the deck. For example, if the player has a soft 'Ace' (Ace considered as 11) and '8' which is equal to 19 then, the agent chooses to 'stand' as one more card can cause a bust whereas when the agent has a hard hand 'Ace', 2, 5 the player will choose to hit as the probability of an agent getting busted is nearly zero.

Complete Point Counting System: The function *calculate_running_count()* implements the functionality of a high-low system where the lower cards like 2 to 6 are assigned the running count as 1, and the neutral cards such as 7 to 9 are assigned the running count as 0, and the high cards 10, Jack, Queen, King, and Ace are assigned running count -1. These running counts are updated whenever an agent sees a new card which will eventually impact the decision on the agent's action.

Rule Variations: Along with a Basic Strategy and a Complete Point Counting System, two Rule Variations are added. These Rule Variations are sent as a flag. The program users can choose whether or not they want to use the rules. The first rule is 6 to 5 Blackjack payout - when the agent has Blackjack the reward they receive in general is 3:2, with this rule the reward received will be 6:5. The second rule is a Prohibition of double down on hard 11 - This prevents users from using double down when player card values are hard 11. These rules can impact the player's winning or losing rate.

C. The State and Action Representation

In a Blackjack environment, a properly defined state and action representation plays a significant role because the informed decisions of reinforcement learning are taken based on these values. This subsection emphasizes the various states and actions defined and represented within the environment.

State Representation: Player's hand values - This state defines the current values of the player's cards. The player's hand values can range from 4 to 21 leading to 18 distinct values. Dealer's visible hand - Dealer's face-up card value is stored. This information primarily helps Q-learning agents understand the likelihood of a dealer's hand-busting or calculating high total which can influence an agent's strategy. The dealer's face-up card values can range from 2 to 11 leading to 10 distinct values. Contains Ace - This is a boolean indicator with values either True or False. This binary value helps the agent get better visuals and connections about hard and soft hands. It can have 2 distinct values either True or False. Hand Type - In a classic Blackjack game, the player may or may not have an 'Ace'. If the player does not have an 'Ace' or chooses the value of an 'Ace' to be 1 it is called hard hands, if the player chooses the value of an 'Ace' to be 11 it is called soft hands. With this indicator, the agent can understand what hand it has. This state can have 2 distinct values either hard or soft. Running count - This stores the count of the Hi-Lo system. It helps the agent in adjusting the strategies based on the remaining card compositions in the deck. In multi-deck for practical purposes the running count ranges between -100 to 100 whereas in a single deck, the running count can range from -20 to +20, resulting in 41 distinct values. Therefore, after evaluating the above states the Q-learning agent should ideally explore 29,520 states in order to fully interpret and optimize the strategy.

Actions Representation: Hit - The agent draws another card from the deck. Stand - It is the end of the agent's turn now dealer can hit or stand before showing the cards. Double Down - The agent will be able to double the initial reward and draw exactly one more card from the deck. Split - If an agent has two cards of the same values the agent can choose to split, if the cards are split then, the agent plays the game as usual but for 2 different hands. There are 4 actions leading to 4 distinct actions. To summarize, the total number of state-action pairs

for Q-learning agents to strategize and learn from different scenarios is 118,080 scenarios.

Navigating through the diverse set of state-action space in the Blackjack environment not only showcases its complexity and exploration of countless possible scenarios that should be handled but also, the intricate nature it utilizes for decision-making.

D. The Q-learning Agent

This section discusses about the implementation of the Blackjack Q-learning agent class. It tells about methods used for selecting an action, updating the Q-table, and decaying of epsilon. The agent is created by defining a class *BlackjackAgent*. This class consists of methods *select_action()* used for selecting an action for an agent to learn from the given set of actions, *update_q_table()* used for updating the states, action, and rewards into the Q-table, and *decay_epsilon()* used to adjust the ϵ value.

Selecting an action: The agent is given to select different actions from the list of actions such as 'hit', 'stand', 'double down', and 'split'. The agent selects an action based on the greedy-ϵ algorithm. The greedy-ϵ algorithm [1] basically consists of two different steps: Exploration and Exploitation. During exploration, the random action is chosen with probability epsilon (ϵ), at this stage the agent tries to explore new actions which it has never tried before. This technique helps the agent discover valuable strategies that it would not have figured out if it had chosen only exploitation. when the random number is greater than the probability epsilon (ϵ), the agent decides to exploit at this stage the agent selects the action with the highest Q-values from the Q-table, this helps the agent to maximize the reward as well as understand its current knowledge based on the observed Q-values.

Updating an action: Once the action is selected and executed the environment returns the next state of the system along with the reward. All these data such as current action. current state, reward, and next state are stored in the Q-table. Initially, it checks if the current and next state exist in the Q-table if not then initializes them. It uses the Bellman equation to calculate the predicted Q-value based on the maximum Q-value of the next state and the reward it received. Finally, it combines the calculated information along with state-action space and updates the Q-values along with the weighted learning rate. This is a repetitive process, allowing the agent to modify and refine its policies and strategies by learning through immediate and future rewards which in turn ensures optimal moves for determining best moves over time.

Decaying Epsilon (ϵ): The main intention of using decaying epsilon is to balance exploration and exploitation rates. ϵ variable determines the agent's ability to explore or exploit. During the early stage of the episode, the value of ϵ is set high encouraging the agent to explore a wide range of possibilities, eventually, as the number of episodes increases the rate of ϵ decreases which indeed signals the agent to choose the best-known actions based on its current learning which are stored in the Q-tables. The rate of decaying epsilon can affect the agent's performance therefore careful selection of the value is vital.

III. Experiments

This section focuses on the experimental setup for the training process for four different scenarios Random Selection of an Action, Basic Strategy, Complete Point Counting System, and impact of Rule Variations.

A. Random selection of an Action

For the training of agents for Random selection of an action approach, the learning rate (α) was set to 0 because learning is not applicable when actions are chosen randomly. Although the initial epsilon (ϵ) was set to 1, it is considered irrelevant in this context since the greedy-ϵ policy does not influence random action selection. The agent was trained for over 1,000,000 episodes to keep consistency among other strategies. This setup marks as a baseline performance measure against more powerful strategic approaches.

B. Basic Strategy and Complete Point Count System

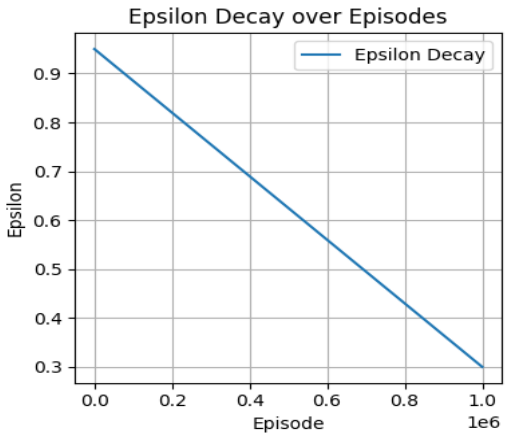


Fig. 1. The plot illustrates the epsilon decay process over 1,000,000 episodes. The y-axis represents the epsilon (ϵ) value, while the x-axis represents the number of episodes. It is a linear epsilon decay, gradually decreasing the value of epsilon (ϵ) over the episodes.

The Q-learning agent was trained with fine-tuning of different hyper-parameters. The hyper-parameters for Basic Strategy and Complete Point Counting were kept the same. The learning rate (α) was set to 0.01 which enables the agent's better learning experience. A smaller learning rate ensures that the agent properly marks the Q-tables which are updated gradually which helps the agent to decide more stable and reliable policies over time. The discount factor gamma (γ) is set to 0.8 indicates, future rewards are valued more than immediate rewards this helps the agent to consider long-term advantage in the period of decision-making. A decaying epsilon is used with an initial epsilon (ϵ) of 0.95 and a final epsilon (ϵ) of 0.3 guiding the agent to focus more on exploration at the initial stages of training, gradually decreasing the exploration rate till it reaches final exploration rate. This allows the agent

to maintain the balance between exploration and exploitation allowing the agent to understand diverse states of the game and sufficiently discover effective strategies.

C. Rule Variations

The hyper-parameters for Rule Variations are set the same as in the Basic strategy. To understand the impact of Rule Variations two flags are set one for each rule which can take boolean values True or False. The agent can decide if the rule must be enabled or disabled.

IV. Results and Discussion

This section includes the findings from the experiment, which contains the visualization of different policies, performance metrics used for Random Selection of an Action, Basic Strategy, Complete Point Counting System, and Rule Variations.

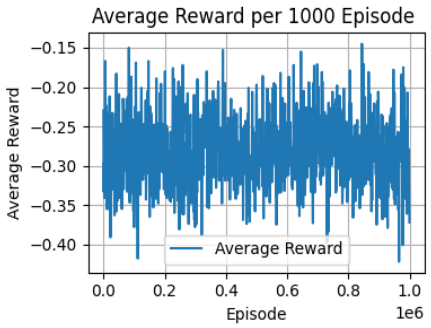
Performance metrics and Rewards

Fig 2. Shows the variations in average rewards progressed over episodes for different methods. The performance of the Q-learning agent was compared with the Random Selection of an Action, Basic Strategy, and Complete Point Counting. Table I below, depicts win, lose, and draw rates of these methods.

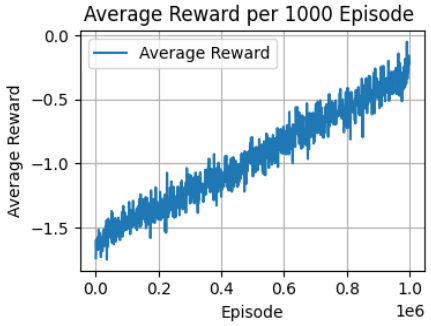
TABLE I
THE TABLE SUMMARISES THE WIN, LOSS, AND DRAW RATE FOR VARIOUS METHODS TESTED FOR 100,000 EPISODES

Strategy	Win rate (%)	Loss rate (%)	Draw rate (%)
Random Selection of an Action	26.384%	64.657%	8.959%
Basic Strategy	43.31%	47.62%	9.06%
Basic Strategy with Q-learning	44.276%	47.848%	7.876%
Complete Point Count System	43.88%	46.53%	9.59%
Complete Point Count System with Q-learning	45.067%	46.737%	8.196%

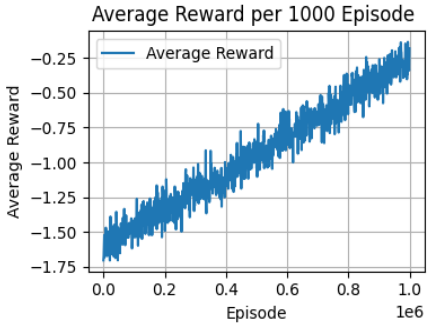
From Table I, a clear conclusion can be drawn that the Basic Strategy and Complete Point Count System have significantly improved their performance compared to the Random Selection of an Action. The second observation that can be drawn is that integrating Basic Strategy and a Complete Point Count System with Q-learning shows a noticeable improvement in their win rates, implying that Q-learning can understand and develop an optimal policy to increase its win rate. The combination of the Complete Point Count System with Q-learning yields the highest performance this signifies the importance of the High-Low system which alters the probability of the winning rate of an agent. Despite of these strategies performing better, the players or the agents are losing games. This means the agents are losing irrespective of any methods they try to apply.



(a) Average Reward per 1000 episode for Random Selection of an action



(b) Average Reward per 1000 episode for Basic Strategy



(c) Average Reward per 1000 episode for Complete Point Counting System

Fig. 2. These figures depict the average reward structure of a Q-learning agent for different methods. The average reward for (a) is almost constant indicating that the agent not learning effectively. The average reward for (b) and (c) increases over the episodes indicating the agent's effective learning and adaptability of the policies.

State values and Policies

Fig 3. highlights the state and policy decisions made by Q-learning agents for soft hands. Fig 3 ascertains that when the player's sum is between 16 to 20 the state values are higher especially when the dealer has lower cards. From the policy heatmap it is inferred that the agent prefers to hit when the agent's sum is low and the dealer shows higher cards, standing is favored for sums ranging between 18 to 21 and the dealer's low cards. The use of double down and split is more specific and strategically placed.

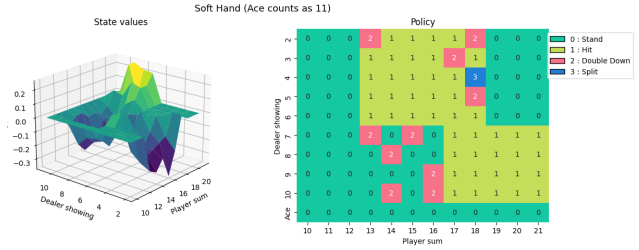


Fig. 3. The plot illustrates the state values and policy decisions of Q-learning agent's strategies for soft hands. The agent adapts the Basic Strategy and Complete Point Counting and achieves an average win rate of around 45% for over 100,000 Episodes

Similarly, Fig 4. highlights the state and policy decisions made by Q-learning agents for hard hands. Fig 4 displays various peaks which indicate the higher risks associated with the hands. The policy heatmaps reveal that the agent prefers to hit when the sum is lower and stands when the sum is high. Double down and split are carefully chosen, due to the lack of flexibility in hard hands.

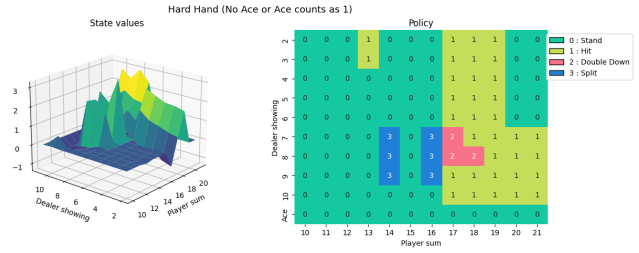


Fig. 4. The plot illustrates the state values and policy decisions of Q-learning agent's strategies for hard hands. The agent adapts the Basic Strategy and Complete Point Counting and achieves an average win rate of around 45% for over 100,000 Episodes

Impact of Rule Variations

The two rules from [2] are analyzed and tested to understand the effects and behavior of an agent when rules such as the Prohibition of double down on hard 11 and 6 to 5 Blackjack payout are imposed on it.

Table II reveals several vital insights regarding the impact of different Rule Variations on the Q-learning model's performance in Blackjack. The average reward structure for win, loss, and draw is noted for the 'No rule variation' method, considering this as a baseline for differentiating the effects of Rule Variations. It is seen that when 'Prohibition of double down for hard 11' is applied the performance of the agent slightly decreases, this limits the flexibility for double down restricting the agent's probability of getting higher rewards. The '6 to 5 Blackjack payout' rule significantly reduces the rewards to slightly more than half of the original and increases the draw reward, which is unfavorable for the agent as it decreases the benefits of achieving a higher reward during a Blackjack. When both rules are applied together, there is a substantial decrease in the average reward distribution.

TABLE II
THE TABLE SUMMARISES THE AVERAGE WIN, LOSS, AND DRAW REWARD DIFFERENT RULES WHICH ARE TESTED FOR 100,000 EPISODES

Rules	Average win reward	Average loss reward	Average draw reward
No Rules applied	2.12	1.09	0.51
Only No Double down for hard 11	2.07	1.05	0.47
Only 6 to 5 Blackjack Payout	1.19	1.09	4.30
Both rules applied	1.09	1.05	4.27

These findings showcase the restrictive nature of rules when modified, particularly when strategy optimization plays a major factor. Q-learning's ability to adapt to these kinds of challenging conditions demonstrates the flexibility and robustness of reinforcement learning in developing adaptive strategies for complex environments of Blackjack. To summarise, the results emphasize the influence of game rules on efficiency and the necessity for adaptive learning approaches to mitigate the effects.

V. CONCLUSION

This study explored the optimal strategy for developing a stochastic environment of Blackjack with Q-learning. The agent which was integrated, effectively learnt the rules and policies for Basic Strategy, Complete Point Counting System, and Rule Variations. The results indicated that the Q-learning agent was able to outperform compared to other methods discussed in this paper. However, the agent was slightly impacted negatively by the introduction of Rule Variations. On the other hand, this study lacks in capturing the intricate details of live casino plays such as betting, usage of multiple decks, or relying on the specific rules of casinos.

The findings of this paper can equally benefit both casino owners and players. The owners can understand how minute changes in the rule can be useful for earning higher profits, while players can make informed decisions to achieve a high success rate. In the future, this work can be leveraged to support the real-time betting system or focus on finding an adaptive method to diminish the negative effects of rules and building a more sophisticated reinforcement learning algorithm to further enhance the gameplay.

REFERENCES

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.
[2] Edward O. Thorp. *Beat the Dealer: A Winning Strategy for the Game of Twenty-One*. Vintage Books, 1966.
[3] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.