Variations on the Reinforcement Learning performance of Blackjack

Avish Buramdoyal

Department of Statistical Sciences
University of Cape Town,
Rondebosch 7701, South Africa
orcid.org/0009-0004-3877-8704
brmavi002@myuct.ac.za

Tim Gebbie

Department of Statistical Sciences
University of Cape Town,
Rondebosch 7701, South Africa
orcid.org/0000-0002-4061-2621
tim.gebbie@uct.ac.za

Abstract—**Blackjack or "21" is a popular card-based game of chance and skill. The objective of the game is to win by obtaining a hand total higher than the dealer's without exceeding 21. The ideal blackjack strategy will maximize financial return in the long run while avoiding gambler's ruin. The stochastic environment and inherent reward structure of blackjack presents an appealing problem to better understand reinforcement learning agents in the presence of environment variations. Here we consider a q-learning solution for optimal play and investigate the rate of learning convergence of the algorithm as a function of deck size. A blackjack simulator allowing for universal blackjack rules is also implemented to demonstrate the extent to which a card counter perfectly using the basic strategy and hi-lo system can bring the house to bankruptcy and how environment variations impact this outcome. The novelty of our work is to place this conceptual understanding of the impact of deck size in the context of learning agent convergence.**

Index Terms—**Blackjack, multi-agent dynamics, Q-learning**

I. INTRODUCTION

The origin of blackjack is still debated today. It seems that blackjack started around the 1700's in France [1] and was called *vingt-et-un*, translating to 21 [2]. The game evolved extensively in North America where a house-banked blackjack system was first introduced in Nevada in 1931 [3]. During the 20th century the game evolved to offer bonus payouts. This included one that paid an extra if a jack of spades or clubs *i.e.* a "blackjack", was dealt together with an ace of spades [1]. It was from then the game changed its name to blackjack [2]. The rise in prevalence of legalised games in the casinos of Las Vegas inspired a number of players to develop optimal strategies for play [2].

Roger Baldwin, Wilbert Cantey, Herbert Maisel and James McDermott, known to blackjack insiders as the "Four Horse-men", were the first in determining modern optimal blackjack strategies [4]. The basic strategy to play blackjack [5] was published in 1956 and this spawned numerous attempts aimed at improvement.

A seminal strategy success became widely known in the early 1960's when mathematician Edward Thorp published his book "Beat the dealer: a winning strategy for the game of Twenty-One" [1]. This caught the attention of many players. Casinos then started to increase the deck size in an attempt to overcome the effectiveness of counting systems [6]. Despite this, blackjack is one of today's most popular gambling games, and is played in practically every casino worldwide.

II. BLACKJACK PROBLEM

A. Game setting

The objective of blackjack is to get a hand total higher than the dealer without "busting". Busting occurs when the players hand exceeds a score of 21. Blackjack is a casino banked game allowing players to compete against the house rather than each other [7]. The game of blackjack consists of a dealer and 1 to 7 players. A standard deck of 52 cards was initially used, but after the announcement of the first winning strategies [8], casinos implemented countermeasures. One countermeasure was to vary the deck size. Cards 2 to 10 are worth their face values. Jacks (J), queens (Q) and kings (K) are counted as 10 and an ace (A) is worth 1 or 11, whichever is the most favourable to the player. A hand with an ace valued at 11 is a "soft hand", and all other hands are considered "hard hands". The distinction between soft and hard hands is important as this defines a key difference in the strategy optimal to the player who is holding either hands. This distinction is understood by analysing the strategy tables provided in Section IX-A of the Appendix.

B. Playing blackjack

Before the start of any hand, each player places an initial bet based on their own bankroll and the minimum-maximum bet set by the casino. The players are then each dealt with two cards face up, and the dealer then also gets two cards; one face up and one face down – the face down card is called a "hole card". The dealing and drawing of cards occur in a clockwise fashion starting from the player left of the dealer [7].

The player will look at his two cards and take actions requested from the dealer. After all players have finished their actions on a given round, the dealer then turns his face down card up and decides which action to take. The challenge posed to the player is the choice of optimal action at each hand, given his current hand total and the dealer's face up card. The

possible actions at play include: standing, hitting, splitting, doubling down, surrendering and insurance.

C. Available actions

A player can choose to "stand", *i.e.* take no additional cards, if doing so is unattractive for his current hand total. Hitting is when a player requests another card from the dealer. A player is allowed to "hit" as many times as he chooses, as long as he does not bust. A player can choose to separate two cards of the same face value and make another bet of equal size as the initial bet made and play each card as a separate hand. This is known as splitting. Here we do not consider the case where the player is allowed to re-split when they have cards of the same value after the first split. Another possible action to a player is to toss his cards away and give up half of his bet, i.e. surrender when he feels that he has a weak hand, based on his first two cards against the dealer's potential hand. A further choice available to the player is to double his bet in the middle of a hand. The player has to then draw only one card from the dealer. A further available choice is "insurance". Under insurance, an additional wager by the player is allowed on the condition that the dealer has an ace as the face up card, and the player does not have a blackjack [6]. The player expects the dealer to have a blackjack. If the dealer does have a blackjack, the player wins twice his side bet.

D. Why not take insurance?

It has been recommended that a player should never take insurance [9]. Using 6 packs of cards, making a total of 312 cards with indices +2 and -1 being assigned for winning and losing respectively, it is shown that irrespective of whether the player has a blackjack or not, and the dealer's face up is an ace, the expected winnings of the player is always less for taking insurance [9]. We therefore do not incorporate insurance as a necessary allowable action in this study.

E. Natural

A starting hand of an ace and a 10-valued card is a "blackjack" or "natural" and beats any other hands – even a hand total of 21 which is not a blackjack.

F. Dealer stands or hits on soft 17

A "soft 17" hand is one when the ace is counted as 11 and the hand total is 17 *e.g.* A-6, A-2-4, A-3-3. In the realm of blackjack, certain casinos require dealers to stand on a soft 17 hand while others necessitate dealers to hit on a soft 17 hand [10].

G. Settlement

If the player and dealer have the same hand total, it is a "tie" or "draw", and no money is exchanged. If neither the player nor the dealer busts, the one with the higher and more favourable hand value wins the bet. If the player busts, the player loses the bet, irrespective of the dealer's score. Assuming blackjack pays 3:2, the player receives 1.5 times his initial bet for having a blackjack. For cases other than for a blackjack and insurance, the amount paid to the dealer by the player is always equal the total bet made by the player for that round.

III. BASIC STRATEGY

For this project, we follow the blackjack "basic strategy" as initially determined by Baldwin et al. [5]. This is a strategy maximizing a player's mathematical expectation given his strategic problem. A basic strategy is simply a proper playing decision for every possible hand against the dealer [8]. Every possible combination of the player hands' total and dealer's face up card has a mathematically correct play and these can be summarised in a strategy table framework adapted from [11], as provided in Section IX-A of the Appendix. The optimal action per round therefore depends on the player's current hand total and the dealer's face up card [5].

The player's basic decisions when playing blackjack are: when to draw or stand, when to double down, and when to split pairs [5].

IV. COUNTING CARDS

The optimal action per each round depends on the player's current hand total and the dealer's face up card [12]. A good player can take advantage of the basic strategy by keeping track of cards dealt in previous plays [8]. The track, or sequence of plays realised at the table, is used as information for the next round and is known as "card counting". Generally, a card counter looks for hands when more high cards are left to be played than a standard deck would have allowed [13]. This was the key to Edward Thorp's Ten Count system which attempts to determine the ratio of high to low cards in a deck [8].

A. Ten count and Hi-lo system

In determining the ratio of high to low cards for a one-deck blackjack game, values are assigned to each card. The values for the Ten count system is given by Table I. The player can now count cards. The player starts with a count of 0. Then, based on the indices, he will add or subtract for every single card revealed. This is known as the "running count".

Card	Value Assigned
A-9	+4
10,K,Q,J	-9

TABLE I: This table is the Ten count system proposed by [8]. This system assigns a value of +4 to cards (A-9) and a value of -9 to (10,J,K). The intuition the player follows is to then bet a higher proportion of his bankroll for higher counts and lower proportion for lower counts.

The next step is to compute the "true count":

$$\text{true count} = \frac{\text{running count}}{\text{decks remaining}}. \tag{1}$$

The higher the count, the more the player should bet given his higher advantage. The general idea is to bet little or nothing when player advantage is low and to bet proportionately higher when player advantage is high [8], *i.e.* in multiples of the player's betting-unit[1] [8]. The mathematical theory of Edward

---

[1]The size of player's bet.

Thorp behind sizing bets from a count was based on the Kelly criterion, an intermediate strategy between maximising one's expected return growth rate while minimising the probability of ruin [14].

Harvey Dubner introduced a simplified variation of Thorp's strategy, called the "Hi-Lo card counting strategy" (or the "point count system"), at the Fall Joint Computer Conference in Las Vegas in 1963 [15]. The Hi-Low system is the most commonly used card counting system nowadays [16] and represented in Table II.

Card	Value Assigned
2,3,4,5,6	+1
7,8,9	0
10,J,Q,K,A	-1

TABLE II: Under the Hi-Lo system, each card is assigned a value of either -1,+1 or 0. The lower cards (2-6) in the deck are given a value of +1. The "neutral cards" (7-9) are given a zero value and the high cards (10,K,Q,J,A) are all given a -1 value. Using the true count, this system allows a player to know his edge and helps him to size his bet at the start of every round.

The value of +1 means that as low cards are depleted from the game, chances of busting are low and so fewer cards can hurt you in the future while the value of -1 means that as high cards are depleted from the game, player advantage falls. The value of 0 represents neutral cards and favours neither the player nor the house.

B. Other counting systems

In assessing the impact that a variation in deck size may have on the performance of a learning agent, we adapt the comparison made by [17] on "The Best Card Counting System: A Comparison of the Top 100" for blackjack. The methodology adopted by [17] in searching for the "best card counting system" involved ranking the different schemes by profit potential.

The counting systems were tested for both single-deck and multi-deck games but ranged from level 1 to 4. It has been found that higher level systems perform at a rate of profit of 0.1% better than level one systems [17]. We consider 2 other counting systems: Zen Count and Uston APC system. To use these two systems, the same counting principle is applied, meaning a player would still bet a higher proportion of his bankroll for having a high count and bet proportionately lower for lower counts. The Zen count framework is presented in Table III.

Card	Value Assigned
2,3	+1
4,5,6	2
7	1
8,9	0
10, A	-2

TABLE III: For this counting method, the player uses these indices assigned and bets accordingly.

The Zen count as suggested from a Blackjack formula used by [17], would yield a Rate of Profit of 2% for a player for a single deck game, with a betting efficiency of 0.97.

The Uston APC system is a balanced count one, implying the running count will always end up 0 once the whole deck has been dealt out. Uston APC is the second best counting scheme [17] and a player would use Table IV to count cards using the Uston APC method.

Card	Value Assigned
2,8	+1
3,4,6,7	2
5	3
9	-1
10	-3
A	0

TABLE IV: Uston APC method indices

Compared to the Zen count, the Uston APC yields a Rate of Profit of 1.98% for a player [17].

V. PERFORMANCE EXPECTATIONS

As suggested in "Professional blackjack" by [12], a player's advantage or disadvantage varies with the rules and number of decks played. Commonly, a player's advantage is around 0.5% with the basic strategy [12].

A. Deck size variation

It turns out that increasing the deck size only slightly cuts the player's advantage [8]. An analysis was conducted by [18] using similar blackjack rules adapted from [8]. The analysis performed was to determine the player's disadvantage % for using the basic strategy. The only difference in simulation was the absence of a cut card in the pack. It was found that a player could have an advantage for a single-deck game but player disadvantage increases when playing with more decks. In addition, it was also found that the fewer decks, the more blackjack you could win relative to a game with more decks [19].

B. Bet size and strategy variation

Wong [12] suggests that a player's expected win % is proportional to his bet size and the amount of time available for play. The more a player bets, the more he will win on lucky hands and the more he loses on unlucky hands. For every unit increase in the true count, given by Equation 1, player's advantage goes up by 0.5% [12].

The variance of possible outcomes to the player depends on the specific rules set [12]. For example, a player allowed to double down after splitting generates higher variance (bigger ups and downs). Alternatively, allowing a player to only double down on 10 or 11 means lower variability. Variance of outcomes also depends on the number of simultaneous hands being played [12].

VI. REINFORCEMENT LEARNING

A. The Reinforcement Learning problem

Reinforcement learning (RL) is often considered to be a third machine learning paradigm alongside supervised and unsupervised learning [20]. RL problems involve relying on

responses from an environment to learn [21], and more specifically to map situations to actions [20]. The responses under RL take the form of rewards to guide the agent in developing his policy [21]. The aim is to maximise a numerical reward signal.

The environment allows for interactions among agents. An agent is an entity making decisions in the environment and must have goal(s) relating to the environment. To encode these goals, we first need to specify how the agent behaves relative to states of the environment. A policy is used to define the agent's behaviour at a given time and is a mapping of states to actions when in those states [20]. A reward signal is then used to define the goal of the problem. At each time step, the agent is sent a single number, a reward. The agent's objective is to maximize his total rewards in the long run while avoiding gambler's ruin. The agent can influence the reward signal only through his actions and thus the reward sent to the agent at any given time depends on the agent's current action and state of the environment. Values advise the long-term worthiness of states after taking into account, the states to follow by the agent and the rewards attached to each of those states. In addition, the model mimics the behaviour of the environment allowing inferences to be drawn about the environment's future behaviour.

For the blackjack case, the environment is the blackjack setup simulation allowing for play between the agent and dealer. The agent is the player playing against the dealer. An intuitive policy would be one allowing the player to get a score as close to 21 without busting. The reward to be maximized in the long run is the difference between the reward the agent receives or pays out at the end of each round. Value states here relates to the value to the agent of being in any particular state from the lookup table in Section IX-A of the Appendix.

Blackjack can be formulated as an episodic finite Markov Decision Process (MDP) where each game is an episode [20]. Rewards +1, -1 and 0 are given for winning, losing and drawing respectively. We note that rewards γ are not discounted, i.e. $\gamma = 1$ and so terminal rewards represent returns. The state depends on the player's hand total and the dealer's face up card.

B. Value functions

We aim to estimate "how good" it might be for an agent to be in a given state, or to take an action in any particular state, *i.e.* "how good" it might be for an agent to take an action given his hand total and the dealer's face up card. The estimates of "how good" it might be for an agent to either be in a state or take an action is derived from experience using Monte Carlo methods [20].

1) State-value function: The idea of "how good" is captured through the state-value function, the expected returns of future rewards for the agent, and is represented by the value function $v_\pi(s)$. The value function represents the value of a state s under policy π and is given by:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]. \tag{2}$$

Here G_t is the reward sequence of the agent when in state $S_t = s$. $v_\pi(s)$ can also be given by the sum of all rewards R_t for each time step t where γ^k represents the discounting factor:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]. \tag{3}$$

2) Action-value function: We can also compute the value of performing action a, being in state s under policy π, *i.e.* the action-value function $q_\pi(s, a)$. The action-valye function represents the expected return of starting from state s, performing action a and subsequently following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]. \tag{4}$$

We can again represent the reward sequence in terms of the sum of all rewards:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]. \tag{5}$$

C. Optimal value function and policy

1) Optimal Value function: The optimal state-value function and optimal action-value function across state space $\mathcal{S}$ are then defined as $v_*(s)$ and $q_*(s, a)$ respectively, and are given by:

$$v_*(s) = \max_\pi \left\{ v_\pi(s) \right\} \text{ for all } s \in \mathcal{S}, \tag{6}$$

$$q_*(s, a) = \max_\pi \left\{ q_\pi(s, a) \right\} \text{ for all } s \in \mathcal{S}. \tag{7}$$

2) Optimal policy: Using the optimal state-value function v_*, actions appearing best after a one-step search will be the sub optimal ones. With optimal action-value function q_*, no one-step ahead search is required from the agent. The action-value function effectively stores results of all one-step ahead searches and the agent can simply find that action maximizing action-value function $q_\pi(s, a)$ for any given states.

VII. METHODOLOGY AND IMPLEMENTATION

A. Environment used

The platform module, OpenGym AI [22], containing a number of simulated environments for which RL algorithms can be compared is used in the first part of this work. We make use of the environment "Blackjack-v0", allowing one to simulate a blackjack game with the MC on-policy, MC off-policy, and the one-step q learning (hitting and standing game only). We provide the known settings of the simulated blackjack environment.

The game is played with an infinite deck. The allowable actions to the player defining the state space are to hit (hit = 1) or stand (stand = 0). After the player stops hitting or stands, the dealer reveals his face-down card and draws until his sum is greater than or equal to 17. If the player busts, the player loses and if the dealer busts, the player wins. If neither the player nor the dealer busts, the (win, lose, draw) outcome is determined by whose sum is more favourable and closer

to 21. The reward returned at each step, is 1 if the player wins, -1 if the player loses, and 0 if there is a tie. In the "Blackjack-v0" setting, we view an observation as a 3-tuple of the player's current hand total, the dealer's face up card and whether the player holds a usable ace, i.e. an ace that can be counted as an 11 without going bust. Extended environments are also implemented, allowing for more realistic assumptions to the game of blackjack.

B. Monte Carlo policy

We first evaluate the state-value function of a particular policy and performance of the player as a function of the number of simulations using a MC approach. Monte Carlo methods only requirement is experience, i.e. sample sequences of actions, states and rewards from an actual or simulated environment. Value estimates and policies are only changed upon the completion of an episode. To estimate the value states, we could adopt either the Monte Carlo on-policy or Monte Carlo off-policy. Any state s can be visited multiple times for the same episode. We adopt the first-visit MC, which averages returns obtained from first visits to states in each episode. to formulate the blackjack problem.

A major problem here is that many state-action pairs may never be visited and learning is thus affected as all states cannot be explored. Episodes are therefore generated with exploring starts. This means that each episode begins with a randomly chosen state and action and then follows the current policy. The agent has to therefore make sure to continue selecting states over time. We therefore adopt the MC on-policy and MC off-policy approach to learn the value states. MC on-policy attempts to evaluate or improve an existing policy while MC off-policy try to evaluate or improve one of two policies.

C. Monte Carlo improvement

Policy improvement is done by making the policy greedy with respect to the current value function such that:

$$\pi(s) = \arg\max_a q(s, a). \tag{8}$$

D. Q-Learning

One of the most important breakthroughs in RL was the development of an off-policy TD control algorithm known as q-Learning [20]. The simplest form of q-learning is a one-step q-Learning as given by:

$$
\begin{aligned}
Q_t(s_t, a_t) &\Leftarrow Q_t(s_t, a_t) \\
&+ \alpha \left[R_t + \gamma \max_{a'} Q_{t+1}(s_{t+1}', a') - Q_{t+1}(s_{t+1}, a_t) \right].
\end{aligned} \tag{9}
$$

α is the learning rate, allowing for the determination of the update made on each time-step t and γ is the discount rate, allowing for the determination of the value of future rewards.

The learned action-value function, Q, directly approximates q_*, independent of the policy being followed which dramatically simplifies our analysis and fasten convergence.

E. Random agent

We compute the average payoff achieved by a random agent using no strategy but simply randomly hitting and standing at any state, i.e. at any round against the dealer.
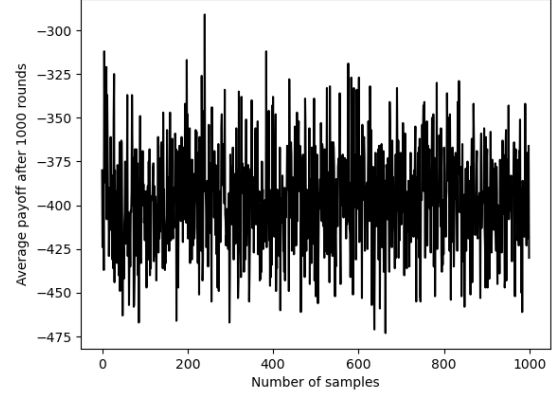

Fig. 1: Here, we provide the average payoff achieved by the agent randomly hitting and standing over 1000 episodes. The figure can be recovered using python script random_agent.py on GitHub resource [23].

The average payoff of -396.29 achieved by the random agent is to be compared to the payoff achieved by the q-learning agent for following q-learning policy.

F. Q-Learning agent: Hit and stand only

For the base q-learning model, we implement a decaying epsilon q-learning algorithm, using the one step q-learning given by Equation 9 for the blackjack environment assumed by "blackjack-v0".

To ensure an effective learning process, appropriate values for ϵ. ϵ is a hyperparameter used to control the balance between exploration and exploitation in many reinforcement learning algorithms. It represents the probability of selecting a random action (exploration) versus the probability of selecting the best-known action (exploitation) at a given state. ϵ allows the agent to explore more states by forcing him to take random actions with probability ϵ [20]. To overcome this problem of exploration and exploitation, a decaying value of ϵ is adopted to ensure the agent minimizes exploring and exploits other states once he has learnt enough about the environment.

We initiate ϵ to be 1 and formulated it in a way that ϵ decreases to 0.9 of its initial value for the first 30% of the training episodes, decreases to 0.2 for the next 40% of the training rounds and reaches 0 in the next 30% of the training rounds. The q-learning model is simulated 1000 times for 1000 rounds between the dealer and the player, using a discount factor γ of 0.1 to keep the agent short-sighted and an arbitrary learning rate α of 0.05.

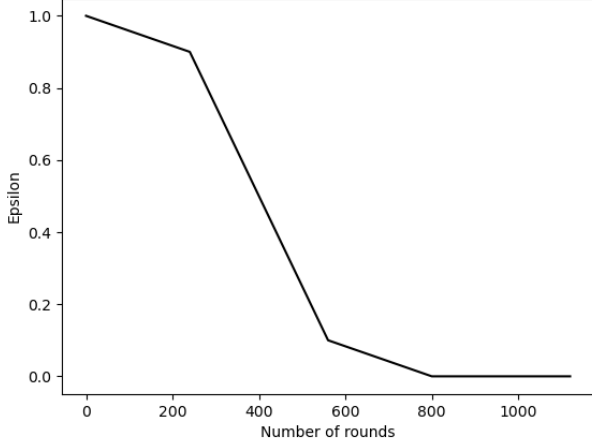We perform a validation analysis of our base q-Learning model to evaluate the distribution of epsilon.

Fig. 2: Here, we provide the ϵ decay plot for the ϵ-greedy polcy used to generate actions of the q-learning agent. The figure can be recovered using python script validation_analysis.py on GitHub resource [23].

Figure 2 demonstrates the rate of decay of epsilon as the number of rounds is increased. Once ϵ reaches a tolerance value, which happens around a total of 800 rounds, the agent has explored enough and the value of ϵ converges to 0.
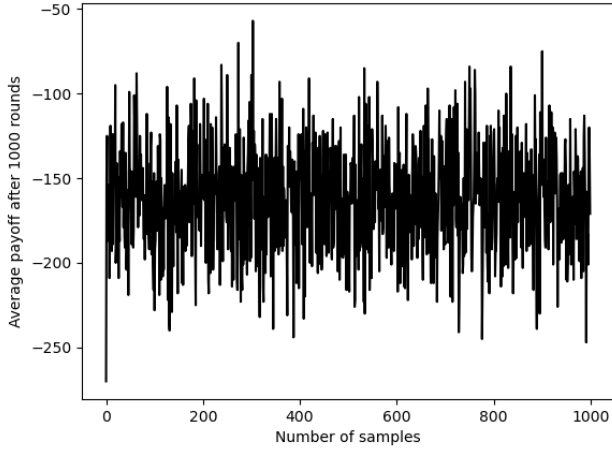

Fig. 3: Here, we provide the average payoff achieved by the q-learning agent over 1000 episodes. The figure can be recovered using python script q_agent.py on GitHub resource [23].

The trained agent, using the chosen hyperparameters achieves an average payoff of -127.677, representing an increase in average payoff of 62.74% to the random agent. This is indicative, of a better and more rewarding policy learnt by the agent following a decaying epsilon q-learning policy.

It is also particularly interesting to analyze the strategy chart learnt by the agent. Figure 4 is a tabular representation of the strategy learned by the agent. The left part shows the strategy for which the player has no usable ace and the right half indicates the strategy for which the player has a usable ace. There are 3 cases where the q-learning agent either hits (H), stands (S) or doesn't reach a state.

(a) No usable ace
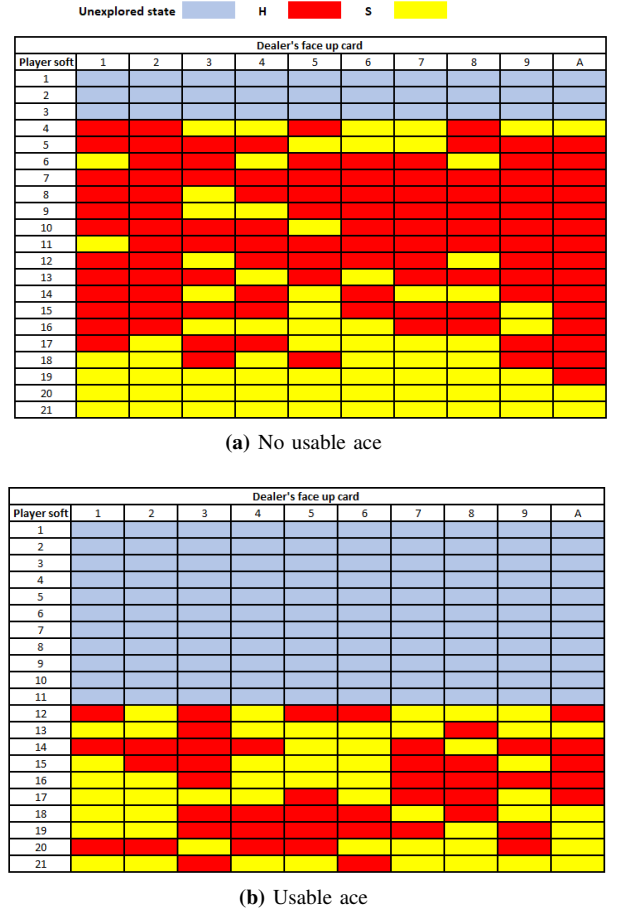
(b) Usable ace

Fig. 4: The strategy learnt by the q-learning agent can be recovered using python script strategy_learnt.py on GitHub resource [23].

In the second part of of Figure 4, 110 states are not explored, given the player has a usable ace, meaning the player's hand total will always be more than 11.

G. Q-Learning policy (all actions implemented correctly)

Two further extended q-Learning models are implemented making the game settings of blackjack more realistic. We assess the impact that a variation in deck size may have on the learning ability of the agent. In addition, we analyze the strategy learnt by the q-learning agent.

1) Impact of deck-size Q-Learning Blackjack: We analyze whether a variation in deck size influences the learning rate of the q-learning agent for a game between one player and the dealer. We assume a standard blackjack game allowing for all actions: hitting, standing, doubling down, splitting and surrendering. The metric used to analyze the learning rate of the agent is winning odds % against the dealer for using the q-learning strategy. The one-step q-learning as given by Equation 9 in Section VII-D is adapted.

We compare the performance of the trained agent over 4 to 8 decks played across three different counting systems. We choose an arbitrary learning rate α of 0.05 and a discount factor γ of 0.1. We train the agent for 500000 episodes and backtest the game 50000 times based on training results.

Deck size	Hi-Lo	Zen count	Uston APC
4	42.24%	41.83%	41.45%
5	42.59%	41.88%	41.43%
6	42.01%	41.96%	42.21%
7	42.14%	42.08%	41.90%
8	41.13%	41.91%	42.11%

TABLE V: The winning odds % can be recovered using python script main.py for which the variation in counting system is implemented from python script env_state.py and variation in deck size from Python script deck_state.py on GitHub resource [23].

We observe that for a blackjack game between a single player and a dealer, played with 4 to 8 decks, the discrepancy in winning odds % across varying deck size is not significant across the 3 counting systems: Hi-Lo, Zen count and Uston APC. The Hi-Lo system might seem to be the most rewarding system for this setting.

We evaluate the learning performance of the trained agent for the 3 counting systems across varied deck size, beyond the standard 4-8 decks used in casinos.
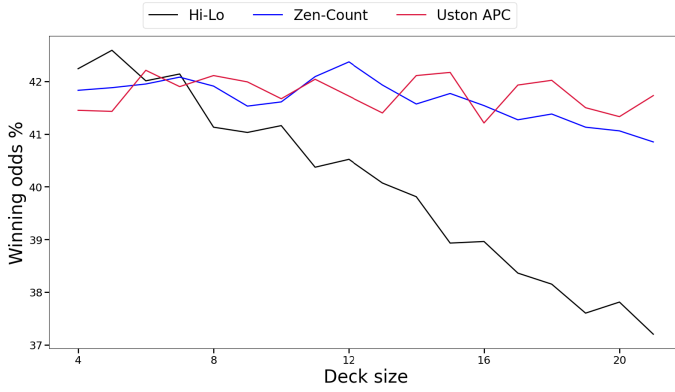
Fig. 5: Figure 5 can be recovered using python script Impact of deck size Q-Learning.py, to which all average payoffs were calculated using, main.py for which the deck size and counting rule variation are implemented using python scripts deck_state.py and env_state.py respectively, available on GitHub resource [23].

From Figure 5 we observe little discrepancy in terms of winning odds % across deck size (4-21) under the Zen Count and Uston APC system and a steeper downward trend of the winning odds % of the trained agent for using the Hi-Lo method.

We conduct a comparative analysis of the winning odds % between the Zen-count and Uston APC methods, with a particular emphasis on a game with more than 21 decks. Empirical data reveals that both Zen-count and Uston APC consistently demonstrate superior performance compared to the widely-used Hi-Lo method. Additionally, we observe a significant downward trend in the Hi-Lo method's winning odds% as the number of decks increases, thus highlighting the need to explore more effective counting strategies in games with a larger deck count. Using multiple decks also makes it more challenging to predict the probability of certain cards being dealt, thus increasing the house edge.
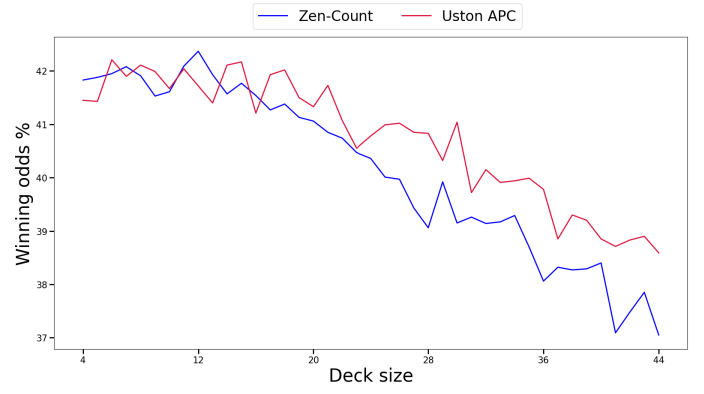

Fig. 6: Figure 6 can be recovered using python script Impact of deck size Q-Learning.py, to which all average payoffs are calculated using, main.py for which the deck size and counting rule variation are implemented using python scripts deck_state.py and env_state.py respectively, available on GitHub resource [23].

2) Policy learnt:

The strategy table for the extended q-learning model[2] is also analyzed.

(a) Player hard total

(b) Player soft total

(c) Player splits

Fig. 7: The policy learnt is generated in a csv format using python script Q Learning - states mapping.py after the Q-scores are calculated using python script Q Learning - qscores.py on GitHub resource [23].

Comparing the strategy learnt for which the player may have hard totals, soft totals and pairs, we observe similarities

---

[2]We use an epsilon greedy q-learning algorithm with a learning rate α of 0.01, a discount factor γ of 0.1 and allow the dealer to hit on soft 17 across 10000000 blackjack games simulated.

to the strategy table adapted from [11] in Section IX-A of the Appendix in 64.78% of the cases. This is a satisfactory result but we note that the agent has not learnt when to surrender even the action is allowable.

H. Bringing down the house

ect thWe implement a blackjack simulator allowing a player to perfectly use the basic strategy table, outlined in Section IX-A of the Appendix and the Hi-Lo system in Section IV-A. This player is the card counter. We compare the performance rates of the card counter to that of other players (called random agents) taking randomly intuitive actions.

The game starts off with a fixed number of decks. After each round, each player places an initial bet. The dealer then deals two cards to each player and himself. Each player chooses their actions followed by the dealer choosing his. Based on favourable scores, money between the dealer and each player is exchanged. This is when one round ends. Once the number of decks used is nearly depleted, there are not enough cards to be dealt to allow play to happen. This is when one simulation ends. A second simulation begins when a fresh number of decks is again placed on the table after which another game with multiple rounds resume. A simulation comes to a halt when we have less than or equal to 25 cards.

We allow the use multiple decks and simulate the game for 4-8 decks. We allow for a multiplayer game and simulate the game for 1-7 players. Actions are based off the strategy table, indicated in Section II-C with the exception of insurance. We impose an intuitive condition where the random agent hits 3 times for a hand total less than 6 and hit 2 times otherwise. The choice of hitting 2 times is chosen minimizing the risk of player busting. A condition where the player splits for a hand total of greater or equal to 4 and less than 18 is also imposed. The card counter uses the Hi-Low method and the other players randomly decide on the % of their bankroll to bet for each round. We allow the dealer the choice to either stand or hit on having a soft 17. We set a fixed bankroll for all players and a fixed one (proportionately higher) for the dealer. It is assumed that, when the dealer goes bankrupt, the house still pays the winning agent and goes negative. The dealer and players all stays in the game and their bankroll stay updated with respect to positive and negative rewards and payments.

Case 1: 4 Players, 6 Decks, Dealer stands on soft 17 and 10000 simulations

Players	Win %	Draw %	Loss %	Bankroll
Card counter	14.55 %	1.35 %	3.21 %	R14, 647, 394
Random agent 1	11.03 %	1.43 %	7.52 %	-R6, 835, 650
Random agent 2	11.01 %	1.44 %	7.52 %	-R6, 989, 950
Random agent 3	11.02 %	1.47 %	7.42 %	-R6, 949, 200
Dealer	3.21 %	1.35 %	14.55 %	-R14, 622, 385

**TABLE VI:** The win, draw, loss % and bankroll balance can be recovered using Python script Strategy Table + Hi-low.py on GitHub resource [23].

Case 2: 6 Players, 8 Decks, Dealer hits on soft 17 and 10000 simulations

Players	Win %	Draw %	Loss %	Bankroll
Card counter	8.58 %	0.81 %	2.06 %	R14, 061, 791
Random agent 1	6.58 %	0.84 %	4.53 %	-R6, 975, 850
Random agent 2	6.56 %	0.87 %	4.53 %	-R7, 012, 000
Random agent 3	6.56 %	0.85 %	4.54 %	-R6, 980, 700
Random agent 4	6.56 %	0.85 %	4.55 %	-R7, 177, 450
Random agent 5	6.61 %	0.88 %	4.47 %	-R6, 961, 650
Dealer	2.06 %	0.81 %	8.58 %	-R18, 691, 233

TABLE VII: The win, draw, loss % and bankroll balance can be recovered using Python script Strategy Table + Hi-low.pyon GitHub resource [23].

Tables VI and VII indicate the outperformance of the card counter against the dealer in terms of win % relative to the random agents whether the dealer hits or stands on soft a soft 17 hand. We also observe a large discrepancy between the bankroll of the card counter compared to the other players and dealer. Even for an initial bankroll ratio of 1:200 in favour of the house, the dealer goes bankrupt for having a card counter perfectly using the basic strategy table and the Hi-Lo system.

Next, we investigate the performance of the card counter relative to another random agent against the dealer as a function of simulation size, number of players and deck size.

1) Performance across simulation size:

Figure 8 shows a clear outperformance of the card counter to the random agent against the dealer across both variations in the number of players, deck size and whether the dealer hits or stands on a soft 17 hand.
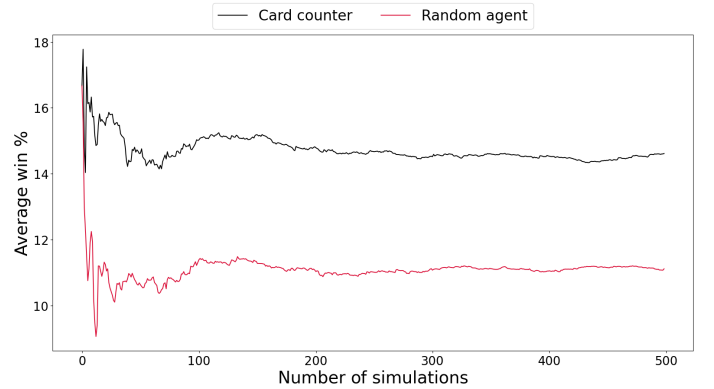
(a) Simulation: 500, Players: 4, Decks: 6, Dealer: Stands on soft 17
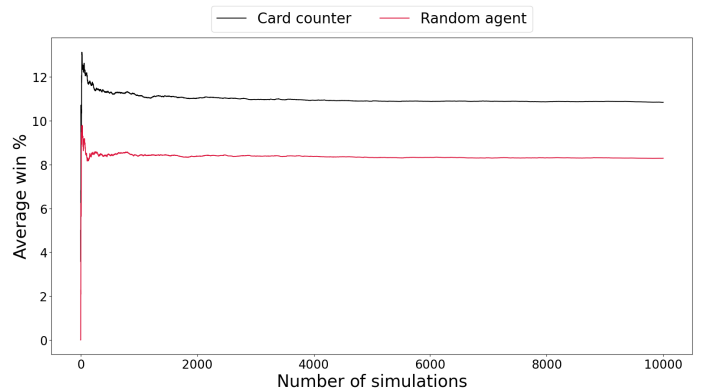
(b) Simulation: 10000, Players: 6, Decks: 8, Dealer: Hits on soft 17

Fig. 8: Figure 8 can be recovered using Python script Performance vs simulations.py on GitHub resource [23].

2) Performance across number of players:

Figure 9 also indicates the outperformance of the card counter to the random agent across increased number of players and varied deck size.
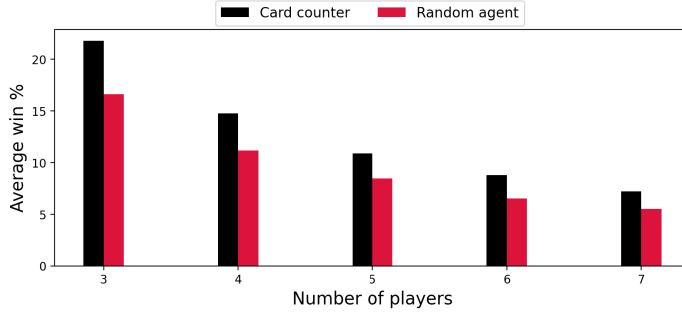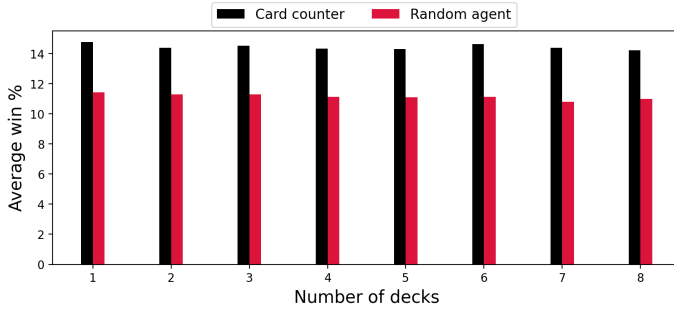

Fig. 9: Simulation: 2000, Players: 3-7, Decks: 8, Dealer: Stands on soft 17. Figure 9 can be recovered using python script Performance vs number of players.py on GitHub resource [23].

3) Performance across deck size:

Figure 10 indicates that across all variations in the number of players, simulation size and whether the dealer stands or hits on soft 17, the card counter always outperforms the random agent. The number of decks in play do not seem to influence the discrepancy in win% against the dealer between the card counter and the random agent.

Fig. 10: Simulation: 1000, Players: 6, Decks: 1-8, Dealer: Hits on soft 17. Figure 10 can be recovered using python script Performance vs deck size.py on GitHub resource [23].

VIII. CONCLUSION

This paper set out to better understand the performance of the decaying epsilon greedy algorithm to blackjack and whether variations in blackjack settings may affect performance of the q-learning agent, particularly across a variation in deck size.. In particular when the learning agent is faced with combinations of increasing deck size, that represents the available states faced by the learning algorithm, and the number of players. The change in number of players moves the game into a simple multi-agent game. Here it is still found that a traditional q-learning approach is preferred relative to the other reinforcement learning algorithms that were implemented. The reasons seems to be that this still allows for realistically learning to happen through the use of the decaying epsilon greedy parameter while requiring fewer simulations for reasonable convergence to the right policy with improved payoffs.

We demonstrate that for deck sizes 4 to 8, learning blackjack using a decaying epsilon greedy q-learning approach, and adopting one of: Hi-Lo system, Zen count or Uston APC to count cards, does not significantly affect learning performance of the agent. As the deck size is extended to a maximum of 21 decks, we observe that across all deck sizes, the agent's average winning odds % remain roughly the same and no apparent trend is noticed for the Zen count and Uston APC. Under the Hi-Lo method however, extending the deck size to a maximum of 21 clearly indicates a steeper downward trend in winning odds % of the learning agent. We would therefore can recommend the use of either the Zen count or Uston APC – two more seemingly rewarding counting systems.

It has also been shown that the decaying epsilon greedy q-learning provides satisfactory results in learning the optimal policy *i.e.* the optimal strategy table, but note that the agent never learns to surrender.

We also investigate the performance of a player perfectly sizing his bet using the Hi-Lo system, and perfectly requesting actions following the basic strategy table as outlined in Section IX of the Appendix. It has been found that the card counter always outperforms the house irrespective of the deck size, number of players or simulations. It is, however, observed that across an increased number of players, the average winnings % of both the card counter and other players decrease exponentially. This unsurprisingly suggests that the algorithm should play on a table with fewer players. Increasing the deck size for a non q-learning process should not significantly affect the winning % of the card counter when using the Hi-lo system. In future work we could move away from the 7 player limit to try find out when learning is saturated as one moves towards an infinite player and decks size problem.

IX. APPENDIX

A. Strategy Table

Fig. 11: The above two tables are the strategy tables adapted from [11]. To use the basic strategy, a player looks up his hand along the left vertical edge and the dealer's up card along the top. In both cases an A stands for ace. From top to bottom are the hard totals, soft totals, and splittable hands. There are two charts depending on whether the dealer hits or stands on soft 17.

B. Algorithm

The q-Learning algorithm procedural form for a decaying greedy epsilon policy case is shown below:

Q-Learning off-policy TD control algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, **arbitrarily, and** $Q(\text{ terminal-state}, \cdot) = 0$

Repeat (for each episode):
   Initialize S
   Repeat (for each step of episode):
     Choose A from S using policy derived from Q
     Take action A, observe R, S′
       $Q(S_t, A_t) \leftarrow Q(S_t, A_t) +$
       $\alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$
   until S is terminal

Algorithm: Here we provide the procedural form of the one-step q-learning, adapted from [20]. This algorithm is fundamental to the project for purpose of analyzing the strategy learnt, average payoff achieved by a trained agent and the impact that a variation in deck size might have on learning performance. The Python implementation of the one-step q-learning can be found in Qbase on Github resource [23].

C. Modules for extended Q-Learning model

Module set 1: Impact of deck size learning:

The Python implementation in assessing the impact of deck size is done using 5 modules: `deck_state`, `env_state`, `training`, `testing` and `main`.

1) `deck_state`: A class `Deck` for which functions allowing for: a variation in the number of decks, shuffling of pack, drawing of cards and returning the number of remaining cards in deck, to be used to size bets, are defined.

2) `env_state`: A class `handState`. is defined. We make the distinction between the dealer's and the player's hand. Distinction about "special type hands": hard hands, soft hands and pair hands are also considered. The state of the environment is definted as a tuple of the player's hand total and the dealer's face up card. A function `update_deck_state` is created to update the deck state based off card counting and a function `update_card_state` which updates the card states (total) upon actions being requested.

3) `training`: A class `Train_agent` is defined, allowing an agent to be trained for particular settings. The allowable actions includes: hitting, standing, doubling down and splitting. A function `row_index` is created which returns the row index of the Q-matrix for a given card state. Two additional functions, `stand_reward` and `doubledown_reward` are also created to return immediate rewards for a player either standing (no reward) and for a player doubling down (getting double his initial bet for placing a bet of equal size in between game). The rewards are modelled as $0, 1$ and 2 for the player losing, drawing and winning respectively against the dealer for standing in a round. Also, rewards are also modelled as $-1, 1$ and 3 for the player losing, drawing and winning against the dealer for doubling down in a round.

4) `blackjack_game`: The `training` module is created which takes as argument the `card_state`, i.e. the player's total and dealer's face up card, allowing for the blackjack play and returns profit made by the learning agent. The agent is then trained for a specified number of times. We note that a new card set is presented to the table when the remaining cards in the deck is less than 30.

5) `testing`: A further module adapted from [24]. Under `testing`, we have a class `Backtest_rl_agent` taking into account the game settings adapted for the training phase. An additional function, `backtest_model` is then created which returns profit made by the learning agent by backtesting the game based off the training results.

Module set 2: Strategy chart:

In this case, we work with fewer modules. More specifically, we have 3 modules: **main**, **Q Learning - qscores** and **Q Learning - states mapping**.

1) **main**: Represents a blackjack simulator allowing for play to happen between a dealer and one player for which an infinite deck/deck with replacement is assumed on each round.

2) **Q Learning**: Allows the agent/player to q-learn the blackjack game, as explained in Section VII-D and returns a csv file: **Qsa_values** of the q-values to each states.

3) **Qsa_values**: Maps states of the strategy table to the respective actions based on the q-values returned by module **Q Learning - qscores**.

REFERENCES

[1] J. Grant, "Why does deck size matter in blackjack?" 2018, https://www.onlinecasinoselite.org/post/origins-history-of-blackjack.

[2] L. Ofton, "The history of blackjack and card counting," 1998, https://www.blackjackapprenticeship.com/the-history-of-blackjack-and-card-counting/.

[3] L. Clark, "Blackjack: The beginning," 2020, http://helipoker.org/poker/blackjack-the-beginning.htm.

[4] J. Haney, "They invented basic strategy," 2008, https://lasvegassun.com/news/2008/jan/04/the-inside-straight-they-invented-basic-strategy-j/.

[5] R. R. Baldwin, W. E. Cantey, H. Maisel, and J. P. McDermott, "The optimum strategy in blackjack," *Journal of the American Statistical Association*, vol. 51, no. 275, pp. 429–439, 1956. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1956.10501334

[6] H. Hellemons, "Can you still beat the dealer?" Master's thesis, Vrij Universiteit Amsterdam, Amsterdam, 1996.

[7] D. Garvie, "Blackjack," 2017, https://www.pagat.com/banking/blackjack.html#:~:text=Blackjack%20is%20a%20casino%20banked,going%20over%2021%20(busting).&text=The%20house%20advantage%20of%20this,rules%20that%20favour%20the%20dealer.

[8] E. O. Thorp, *BEAT THE DEALER. A WINNING STRATEGY FOR THE GAME OF TWENTY ONE*. Vintage; Revised edition, 1966.

[9] M. Shackleford, "Insurance in blackjack - just say no!" 2018, https://wizardofodds.com/video/blackjack-insurance/.

[10] H. Tamburin, "The soft 17 rule in blackjack," 2006, http://www.readybetgo.com/blackjack/strategy/soft-17-rule-2496.html.

[11] M. Shackleford, "4-deck to 8-deck blackjack strategy," 2019, http://wizardofodds.com/games/blackjack/strategy/4-decks/.

[12] S. Wong, *Professiona Blackjack*, 1st ed. Pi Yi Press, 1994.

[13] Pagat, "Blackjack," 2019, https://www.pagat.com/banking/blackjack.html.

[14] E. O. Thorp, "The kelly criterion in blackjack sports betting, and the stock market," *Handbook of Asset and Liability Management*, pp. 385–428, 2006. [Online]. Available: http://www.eecs.harvard.edu/cs286r/courses/fall12/papers/Thorpe_KellyCriterion2007.pdf

[15] L. Patterson, "Harvey dubner: the forgotten man of blackjack. gambling times,," 2002, http://www.gamblingtimes.com/writers/jpatterson/jpattersonsummer2002.htm.

[16] M. Shackleford, "Introduction to the high-low card counting strategy," 2019, https://wizardofodds.com/games/blackjack/card-counting/high-low/.

[17] A. Snyder, "The best card counting system: A comparison of the top 100," 1981, http://www.blackjackforumonline.com/content/hundred.htm.

[18] K. A. Conrad and B. Smith, "Blackjack betting systems and strategies: the mathematics behind the game," Master's thesis, Ball State University, Muncie Indiana, 2002.

[19] K. Smith, "Why does the number of decks matter in blackjack?" 2015, https://www.blackjackinfo.com/why-does-the-number-of-decks-matter-in-blackjack/.

[20] R. S.Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2014.

[21] C. D. Granville, "Applying reinforcement learning to blackjack using q-learning," Master's thesis, University of Oklahoma, United States, 2005.

[22] "Gym is a standard api for reinforcement learning, and a diverse collection of reference environments," https://www.gymlibrary.dev/index.html.

[23] A. Buramdoyal, "Impact of deck size q-learning blackjack," 2020, https://github.com/avishburamdoyal/The-Impact-of-deck-size-Q-Learning-Blackjack.

[24] Y. Jung, "Playing blackjack with reinforcement learning," 2018, https://github.com/youngho92/Blackjack_Project-ReinforcementLearning/blob/master/codes/backtest.py.