CS238 Project
Monte Carlo Blackjack

Tim Gianitsos, Maika Isogawa, and Daniel Mendoza
Stanford University
tgianit@stanford.edu, misogawa@stanford.edu, dmendo@stanford.edu

Abstract

Card games with multiple players provide an opportunity to experiment with reinforcement learning techniques. In card games played at casinos, the stakes are higher because real money is involved. Thus, 'best practices' have been outlined for humans to play these gambling games and have a greater chance of success. We investigate whether Monte Carlo Tree Search can perform better than the best practices for the casino game Blackjack. We also evaluate if having a history makes a difference on the success of the algorithm (allowing the agent to remember what cards have been played before or not). We find that there is a best policy specific to each state of the game, and this can be further detailed by keeping track of what cards are left in the deck. We also find that the history does not significantly affect the expected utility.

1 Introduction

Blackjack is an exemplary game of uncertainty and strategy. The decision space of the game, combined with the simplicity of available actions makes it a perfect problem to approach with reinforcement learning algorithms. Due to the popularity of gambling games, strategies and 'best practices' have developed to increase the chance that a player will win. In this project, we aimed to compare the success of our reinforcement learning techniques compared to the well known strategies to win Blackjack.

This paper depicts our effort to determine how common reinforcement learning algorithms stack up to the success of decades of human experience and knowledge. Throughout the history of casinos, people have found ways to beat the system. Groups around the world developed methods to 'count cards,' winning them large sums of money. These methods are now banned in casinos, but they cannot stop skilled players with good strategies. We implement a Monte Carlo Tree Search method to discover what the best strategy is for Blackjack.

2 Background

The rules of Blackjack are built into our reinforcement learning model. Thus, it is helpful to know the basic rules of the game. We mention some of the choices we have made to relax or constrict the problem in section 2.4.

2.1 Rules

Blackjack can be played with one dealer and multiple participants. For the scope of this project, we only consider the dealer and a single other player. Blackjack uses a standard deck of 52 cards, but it is common for casinos to use multiple decks. Here are some basic rules of the game:

- Objective: A player attempts to beat the dealer by getting as close to 21 as possible.

- A player loses if the value of their hand exceeds 21.
- Face cards are worth 10. Other cards are worth their value.
- An Ace can be worth 1 or 11. it is up to the player to decide how to value an Ace.
- At the beginning of the round, each player is dealt 2 cards. The dealer's second card is dealt face-down.
- If the value of the dealer's hand is less than 16, the dealer must take another card.

2.2 Actions

If a player does not have 21 during the initial deal (in which case they win automatically), they have two options during their turn. A player may 'hit' - add another card from the deck to their hand - or 'stay' - end their turn without adding another card to their hand.

2.3 Gameplay

A round of Blackjack is played as follows. This structure is built into our model.

- (SETUP) The player is dealt one card face-up, then the dealer is dealt one card face-up. The player is then dealt a second card face-up, then the dealer is dealt another card face-down. If the value of the player's hand is 21, they automatically win the round.
- (PLAYER TURN) The player makes a choice 'hit' or 'stand.' The player may choose to 'hit' as many times as they choose, until they choose to 'stand,' or until they 'bust' - the value of their hand exceeds 21.
- (DEALER TURN) The dealer now reveals their face-down card. If the value of the hand is 17 or more, they must 'stand.' If the value of the hand is less than 17, they must 'hit.' Thus, the dealer's actions are deterministic.
- (END) If neither the value of the hand of the dealer nor the player exceeds 21 and 'busts,' the hand with the higher value wins the round.

As a popular game with the opportunity for significant financial gain, players have developed a basic strategy for Blackjack. The strategy depends on the face-up card of the dealer's hand. If the dealer's face-up card is a good value (ex. Ace, 7, 8, 9, 10), the player should hit until the value of their hand is 17 or more. If the dealer's face-up card is a bad value (ex. 6 or less), the player should hit until the value of their hand is 12 or more.

2.4 Relaxations

For the scope of this project, we have chosen to forgo implementing some portions of Blackjack. These omitted rules include things like betting, splitting pairs, doubling-down, and reshuffling. In addition, we implement a naive strategy where the player will 'hit' until the value of their hand is 17 or more as our version of human performance. This simple best practice is more widely known than the slightly more complicated best practice described in section 2.3.

3 Literature Review

Multi-player card games are an excellent opportunity to test algorithms and techniques due to their straight-forward state and action spaces. Thus, many pieces of literature (both early and recent) regarding card games have made an impact on reinforcement learning. Kakvi [1] in 2009 specifically explores reinforcement learning for Blackjack. In this paper, a softmax selection agent is utilized to strike a balance between exploration and exploitation. The study showed that altering rewards changed the optimal policy of the agent. Furthermore, it was found that policy alone would not allow the agent to win the game. Seeing this result, we knew we needed to take a slightly more involved approach.

With Monte Carlo Tree Search as our chosen method, we searched for literature on prior work in this area. With the rising popularity of writing sites such as Medium, reinforcement learning techniques and machine learning has become more accessible compared to traditional article and journal papers.

In a recent article, Byrne [2] utilizes various Monte Carlo Tree Search methods to find optimal policies for Blackjack. Byrne explores various learning rates and policy updates, providing us with diverse inspiration for our approach beyond what we have learned in the course.

Furthermore, Monte Carlo Tree Search methods have been popularized for tackling problems with imperfect information or that are only partially observable. Delattre and Fournier [3] consider deterministic two-player games with incomplete information. While MCTS methods worked quite well for games that were totally observable, Delattre and Fournier came across disappointing results in situations that were only partially observable. Recognizing this difficulty regarding the performance of MCTS methods for certain problems, we adapted our model to better deal with situations with limited knowledge. In the case of Blackjack, the dealer's hand is only partially observable, making it the perfect problem to tackle.

4 Approach

We frame BlackJack as a Partially Observable Markov Decision Process (POMDP). We define our state space as a tuple of counts of each card currently in the deck, the agent's hand, and the dealer's hand: $(\text{cards in deck}, \text{cards in agent's hand}, \text{cards in dealer's hand})$. At each state, there are two possible actions: hit or stay. The observations are the cards that have been drawn from the deck that are visible to the agent. A reward of 1 is given if the agent wins and -1 if the agent loses, otherwise 0.

Our approach is based upon POMDP Monte Carlo Tree Search (MCTS) algorithm defined in [4]. MCTS is an online and sampling based method especially useful for POMDPs with many states for estimating the optimal value and policy of an POMDP. The MCTS algorithm first samples a state from the current belief and then simulates the outcome of the state based upon some initial policy.

4.1 POMDP MCTS

The POMDP MCTS approach assumes a general case that the sequence of previous actions and resulting observations impact the expected value of a state and action. Our implementation is outlined in Algorithm 1 and Algorithm 2. In Algorithm 1, SELECTACTION takes as argument the current belief and simulation depth. The sequence of actions and observations in the simulation is tracked in a history variable h and is initially an empty set at the start. The algorithm then samples a state from the current belief via SAMPLEBELIEF defined in Algorithm 2 and assumes that the sampled state is true. The assumed state is simulated according to POMDPSIMULATE. During the simulation variable T keeps a track of the histories that have already been simulated. Any history that has not been simulated is evaluated by ROLLOUT according to policy π_0 defined in Algorithm 2. Variable Q stores the value of a history-action pair that is updated throughout the simulation. Variable N keeps a track of the number of times the history-action pair has been visited. POMDP MCTS executes multiple simulations and updates the expected value of a history-action pair according to the outcome of each simulation. More details about MCTS with POMDPs can be found in [2].

4.2 Sampled MDP MCTS

The POMDP MCTS algorithm provides a general policy and value estimation algorthim for any POMDP. Most notably, it makes the generalization that the sequence of actions and observations leading up to a state affect the expected value of the state. However, In BlackJack the actions and observations of the previous game have no influence on the future games because at the end of each game, the true state of the game is revealed and thus there is no state uncertainty. Thus long sequences and actions and observations over multiple games of BlackJack have little influence on the outcome of the current game because the agent already knows what the deck looks like at the start of each game. Thus keeping track of long history sequences is redundant and likely will need more simulations to converge to the optimal policy. Thus to estimate the optimal policy, we propose that we can simply sample an MDP from the belief of the POMDP and simulate the MDP using the MDP MCTS implementation which is detailed in [2]. Thus we can simply redefine SELECTACTION shown in Algorithm 1 to use the MDP MCTS simulation method. Algorithm 3 outlines this change. In Algorithm 3, the variable Q now takes as argument a state-action pair instead of history-action pair. Throughout the simulation we update Q. To find the estimate optimal policy, we take the average of

Algorithm 1 POMDP MCTS

function SELECTACTION(b, d)
  $h \leftarrow \emptyset$
  while Convergence Condition not satisfied **do**
    $s \sim$ SAMPLEBELIEF(b)
    POMDPSIMULATE(s, h, d)
  end while
  return argmax$_a Q(h, a)$
end function
function SIMULATE(s, h, d)
  if $d = 0 \vee$ deckIsOutOfCards(s) **then return** 0
  end if
  if $h \notin T$ **then**
    for all $a \in A(s)$ **do**
      (N(h, a),Q(h, a)) $\leftarrow$ (N$_0(h, a)$,Q$_0(h, a)$)
    end for
    $T = T \cup \{h\}$
    return ROLLOUT(s, d, π_0)
  end if
  $a \leftarrow$ argmax$_a Q(h, a) + c\sqrt{\frac{\log N(h)}{N(h,a)}}$
  $(s', o, r) \sim G(s, a)$ ▷ next states are generated by drawing cards randomly in the deck
  $q \leftarrow r + \gamma$SIMULATE$(s', hao, d - 1)$
  $N(h, a) \leftarrow N(h, a) + 1$
  $Q(h, a) \leftarrow Q(h, a) + \frac{q - Q(h,a)}{N(h,a)}$
  return q
end function

Algorithm 2 Sampling Strategies

function SAMPLEBELIEF(b) ▷ We already know agent's hand, need to guess the dealer's hand by guessing cards that are in the deck
  dealerHand $\leftarrow$ getVisibleDealerCard(b)
  while getValue(dealerHand) < 17 **do** ▷ We know dealer must hit till atleast 17
    dealerHand $\leftarrow$ dealerHand $\cup$ RandomCardFromDeck(b)
  end while
end function

function $\pi_0(s)$
  dealerHand $\leftarrow$ getDealerHand(s) ▷ This is a guess of the dealer's sampled in SAMPLEBELIEF
  agentHand $\leftarrow$ getAgentHand(s)
  if getValue(dealerHand) > 21 **then**
    return stay
  end if
  if getValue(dealerHand) $\geq$ getValue(agentHand) **then**
    return hit ▷ The sampled value of dealer hand is greater, thus we hit
  else
    return stay ▷ The sampled value of dealer hand is smaller, thus we stay
  end if
end function

the estimated values of each sampled state given the action and choose the action associated with the highest average. This algorithm decreases the space complexity compared to POMDP MCTS since it no longer needs to maintain the history of the simulation.

Algorithm 3 Sampled MDP MCTS

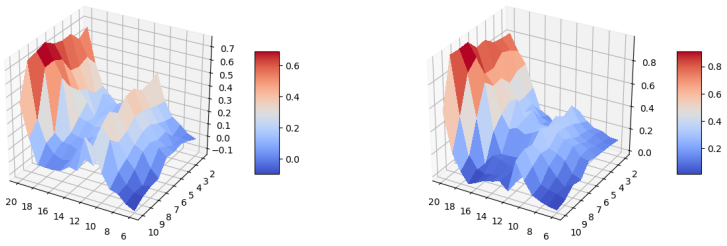
function SELECTACTION(b, d)
  $V \leftarrow \emptyset$ ▷ Keep track of the sampled states
  while Convergence Condition not satisfied **do**
    $s \sim$ SAMPLEBELIEF(b)
    $V \leftarrow V \cup \{s\}$
    SIMULATE(s, d) ▷ Only use the sampled state, do not need history
  end while
  return argmax$_a$(average($\{Q(v, a) : v \in V\}$)) ▷ Return action that maximizes the average estimate value of sampled states
end function

5 Results

We ran our MCTS implementations on different states of the game to extract the estimated optimal utility of policy of each state. In Figures 1,2, and 3, we show the estimated optimal expected utility for three different states: the deck of cards is full, the deck of cards has no more aces, and the deck of

cards has no more face cards. The axis with a scale from 4 to 21 indicates the agent's hand while the axis with a scale from 2 to 11 indicates the dealer's hand. The utility is drawn is scale from low (blue) to high (red). The hyper parameters of the MCTS algorithms were $\gamma = 0.5$ and $c = 5$. The Convergence Condition for SELECTACTION was set to 100 iterations of sampling and simulating.

Figure 1 shows the estimated optimal expected utility calculated by POMDP MCTS and sampled MDP MCTS with a full deck given the agent's and dealer's hand. We can observe that the sampled MDP MCTS graph and POMDP MCTS graph arrive at similar conclusion about the optimal utility for each state. This confirms our intuition that the history that is normally tracked in POMDP does not significantly affect the estimated expected utility since each game of BlackJack have no dependencies on one another.



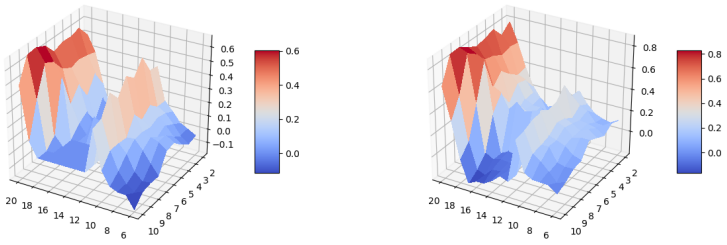
(a) POMDP MCTS (b) sampled MDP MCTS

Figure 1: Estimated Optimal Expected Utility with starting with full deck.

Figure 2 shows the estimated optimal expected utility given that there are no aces in the deck left. From a visual comparison of Figures 1 and 2, the peaks and valleys of each graph is the same, except in Figure 2, the slope of the peaks is more steep. This indicates that knowing that there are no aces in the deck, compared to a full deck, there are not as many advantageous hands. We can see that the expected value is much lower for agent's hands of about 14-18 for the graphs in Figure 2 than in Figure 1. This seems likely due to the fact that the if the dealer does not bust, it will always have hand of at least value 17. But since the agent only has 14-18, the agent has a high chance of busting in attempt to achieve a value greater than 17.

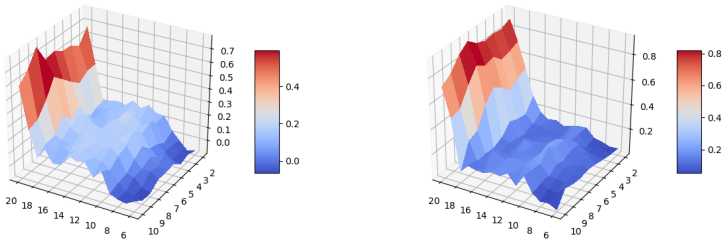
Figure 3 shows the estimated optimal expected utility given that there are no face cards (face cards have value of 10) in the deck left. Visually, the graphs in Figure 3 are quite difference to Figures 1 and 2 since the local maximum of the agent's hand at 10 is dramatically decreased. There is a low expected utility for most agent's hands except for values close to 21, indicating that there are very few advantageous hands available for the agent in this state.

Overall the graphs reflect expert human opinion about advantageous hands and disadvantageous hands. Clearly, the closer the dealer's hand is to 21, the higher expected utility. Further, one slightly less intuitive notion in Blackjack is that a starting hand of value 10 is considered advantageous



(a) POMDP MCTS (b) sampled MDP MCTS

Figure 2: Estimated Optimal Expected Utility knowing no aces in deck left



(a) POMDP MCTS (b) sampled MDP MCTS

Figure 3: Estimated Optimal Expected Utility knowing no face cards in deck left

because of the likelihood of hitting 21. Both graphs in Figures 1, 2, and 3 reflect this intuition as the estimated optimal utility reaches a local maximum when the agent's hand is close to a value of 10. Further, human experts claim that 16 is the worst hand [5]. Both graphs indicate that this is a valid claim since the estimated expected utility for an agent's hand of 16 for each dealer's hand is low. Further, observe that all graphs have a local maximum with respect to the dealer's hand as the dealer's hand approaches 6. Human experts suggest players to assume that the hidden card of the dealer is a value of 10, and thus when the dealer shows a 6, the player assumes the dealer has a hand of value 16. A hand of 16 is considered the worst hand for the dealer since the dealer must hit on a 16 and is likely to bust [5]. Thus the local maximum with respect to the dealer's hand approaching 6 reflects this human intuition.

Table 1 and Table 2 show the estimated optimal policy given the deck has no aces and no face card respectively. S indicates stay while H indicates hit. From observing Figures 2 and 3 we can see that the expected utility is quite different for some combinations of agent hands and dealer hands. However, a visual comparison of Tables 1 and 2 show a mostly similar policy overall.

Table 1: MCTS no aces

dealer\agent	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S	
3	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S	
4	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S	
5	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S	
6	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S	
7	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	
8	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	
9	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	
10	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	
11																

Table 2: MCTS no faces

dealer\agent	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S
3	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S
4	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S
5	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S
6	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S
7	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S
8	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S	S
9	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S
10	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S
11	H	H	H	H	H	H	H	H	H	H	H	H	H	S	S	S

6 Conclusion

In conclusion, we found that expert human Blackjack strategies are reflected by the estimated optimal policies and utilities extracted via Monte Carlo Tree Search. The similarity of sampled MDP MCTS and POMDP MCTS results suggests that the sequence of actions and observations has little influence on the current state due to the fact that at the end of each Blackjack game, the true state of the game is revealed. We also found that given a certain deck (i.e. deck with no face cards) significantly affects the expected utility. However, we saw that the optimal policy for different states of decks is not very different from one another.

7 Contributions

Tim Gianitsos - Development tools, repository maintenance, interpret results

Maika Isogawa - Developed an approach, programmed portions of the model, and wrote major sections of this paper.

Daniel Mendoza - Implementation and Experimentation of MCTS, Results Collection, Approach Paper section, Conclusions

References

[1] Kakvi S.A. (2009) Reinforcement Learning for Blackjack. In: Natkin S., Dupire J. (eds) Entertainment Computing – ICEC 2009. ICEC 2009. Lecture Notes in Computer Science, vol 5709. Springer, Berlin, Heidelberg

[2] Byrne, Donal. "Learning To Win Blackjack With Monte Carlo Methods." Medium, Towards Data Science, 7 Nov. 2018, towardsdatascience.com/learning-to-win-blackjack-with-monte-carlo-methods-61c90a52d53e.

[3] Delattre, Sylvain, and Nicolas Fournier. "On Monte-Carlo Tree Search for Deterministic Games with Alternate Moves and Complete Information." ESAIM: Probability and Statistics, vol. 23, 2019, pp. 176–216., doi:10.1051/ps/2018006.

[4] Kochenderfer, Mykel J. Decision Making Under Uncertainty: Theory and Application. MIT press, 2015.

[5] H. T. Ph.D, "How to Play Blackjack," How To Play Blackjack.