Finding the Optimal Blackjack Strategy Using Markov Decision Process Monte Carlo

Manuel Lopez-Mejia
College of Engineering, North Carolina State University, Raleigh, USA, manuel.lopezme05@gmail.com, 92855

Keywords: Blackjack, Monte Carlo Tree Search, Markov Decision Process, Markov Chains, Dynamic Programming.

Abstract. This paper investigates the application of Bayesian probability to predict subsequent card draws in Blackjack. Leveraging these probabilistic predictions, we calculate the expected value for key player decisions (Hit, Stand, Double Down, Split) to identify strategies that maximize expected returns. The player's sequential decision-making problem is modeled as a Markov Decision Process (MDP). A Monte Carlo Tree Search (MCTS) algorithm is implemented to estimate the optimal action-value functions within this MDP framework by averaging returns over numerous simulated game episodes. This approach aims to derive a near-optimal playing strategy based on probabilistic inference and reinforcement learning techniques.

1 Introduction

Blackjack is a popular card game where you try to reach a hand as close to 21 as possible without going over (busting). If you are able to have a higher hand than the dealer's hand, you win. Cards are ranked with values from 1 to 11, with Ace being worth 1 or 11 (whichever is optimal) and face cards (Kings, Queens, and Jacks) being valued at 10.

Players and the dealer are initially dealt a pair of cards which are shown to everyone except for one of the dealer's cards. Players move first, choosing to either stand, hit (draw one more card), double down (double one's wager by drawing one card and then stand), and if dealt a hand with a pair of the exact same cards, split where you can split your current hand into two new hands and wager the same amount on each. After a player stands, the dealer's hidden card is shown and he must hit until he reaches at least 17. For every round of Blackjack played, cards are drawn without replacement until they're reshuffled, typically once 15 to 50 percent of the game deck has been drawn.

Given these rules, particularly the sequential nature of decisions, the dealer's fixed strategy, and the changing composition of the deck due to drawing without replacement, determining the optimal action at any given point presents a complex challenge. The player must constantly assess the risk of busting against the potential reward of improving their hand relative to the dealer's likely outcome, making Blackjack a fascinating domain for applying strategic analysis and Bayesian reasoning.

Contemporary computing experiments simulate games that utilize infinite decks (thus ignoring card composition) [1] [2] [3] and are thus fairly tractable games that can be solved using traditional linear programming methods. Other limitations of these experiments are that they only handle hitting and standing while ignoring doubling down and splitting (the latter which requires recursive programming) and are not very applicable to most contemporary casino versions of the game.

Khurna (2020) uses an MDP approach following Sutton and Barto (2015) but, in addition to the limitations above, uses a Q-learning method for reinforcement learning rather than Monte Carlo as prescribed by Sutton and Barto. Q-learning requires an agent to explore every possible game state which may be reasonable in a tractable environment. But in an environment where we want our agent to keep a measure of the shoe composition (e.g., ”the true count”), accurately learning Q-values across this vast, dynamic space is extremely challenging and data-intensive. On the other hand, MCTS does not need to explicitly learn values for every possible state. Instead, it performs simulations from the current state (which includes the current shoe composition). It builds a search tree on the fly, exploring relevant future states based on the current probabilities derived from the shoe. This makes it much more scalable to large state spaces driven by factors such as card counting.

2 Background

2.1 Bayesian Probability

Since the game is played with a finite set of cards (a shoe can consist of any number set of cards, typically 6 decks in casino games) and we can have some information of what cards have been played, we can estimate the probability of what the next card in the shoe will be and what the final dealer hand’s will be so that we know what actions will give us the best chance at beating the dealer and maximize our winnings. As the game continues and we continue to draw more cards, we gain more information of what the shoe composition looks like and are able to predict with more certainty the likelihood of what cards will be drawn next. This is what Bayesian Probability is about, updating our beliefs as new information comes to light.

2.1.1 Formulation

Let:

- H: the player’s current hand (a multiset of card ranks)
- U: the dealer’s upcard
- P: the multiset of all cards dealt so far (including player’s cards, dealer’s cards, and previous rounds)
- $R = \{\mathrm{A}, 2, 3, \ldots, 10, \mathrm{J}, \mathrm{Q}, \mathrm{K}\}$: the set of card ranks
- $m_r = 4$: the number of cards of rank r per deck, for each $r \in R$
- $D \in \mathbb{N}$: the number of decks in the shoe

The initial count of cards of rank r in the shoe is

$$x_r^{(0)} = D \cdot m_r, \quad r \in R,$$

and the initial total number of cards is

$$N^{(0)} = \sum_{r \in R} x_r^{(0)} = 52D.$$

Let

$$O = H \cup \{U\} \cup P$$

be the multiset of all observed cards. For each rank r, define

$$\Delta_r = |\{c \in O : \mathrm{rank}(c) = r\}|,$$

i.e., the number of cards of rank r removed from the shoe. Then the remaining count of rank r is

$$X_r = x_r^{(0)} - \Delta_r,$$

and the total remaining cards is

$$N = \sum_{r \in R} X_r = 52D - |O|.$$

2.1.2 Probability of Drawing Rank Value

Since draws are without replacement and the remaining cards are identically distributed, the probability that the next card has rank r given the current state (H, U, P) is

$$\mathbb{P}(\mathrm{rank}(\text{next card}) = r \mid H, U, P) = \frac{X_r}{N}.$$

2.1.3 Proof

By symmetry, each of the N remaining cards is equally likely. Of these, exactly X_r have rank r. Thus the probability of drawing rank r is the fraction of remaining cards of that rank.

2.2 Expected Value of Player's Hand

For our model, we want to select the action that maximizes the player's expected value (AKA "expected utility"). In this case, expected value is the payoff we receive from our wager based on whether we choose to Hit, Stand, Double Down, or Split (if possible).

2.2.1 Formulation

Let:

- $R = \{A, 2, 3, \ldots, 10, J, Q, K\}$: Set of card ranks.
- X_r: Number of cards of rank r remaining in the deck(s).
- $N = \sum_{r \in R} X_r$: Total number of cards remaining.
- $\Pr(r) = \frac{X_r}{N}$: Probability of drawing a card of rank r.
- H: The player's current hand (a multiset).
- U: The dealer's upcard.
- P: The multiset of all cards drawn in previous rounds (observable history).
- $v(H)$: The value of hand H, treating Aces optimally.
- $\sigma \in \{0, 1, 2, 3\}$: Number of splits remaining.
- $P_D(y \mid U, P)$: Probability that the dealer ends with value y, given upcard U and card history P.

2.2.2 Payoff Function

For simplicity denote *x* as $V(H_F)$ (the final value of the hand of the player) and *y* denote the final hand value of the dealer, The payoff function is represented by

$$g(x,y) = \begin{cases} +1, & x \leq 21 \text{ and } (y > 21 \text{ or } x > y) \\ 0, & x \leq 21,\ y \leq 21,\ x = y \\ -1, & \text{otherwise} \end{cases}$$

2.2.3 Expected Value Formulas

Expected Values for Each Action

1. **Standing**: Take no more cards

$$\mathrm{EV}_{\text{stand}}(H) = \sum_{y=17}^{22} P_D(y \mid U, P) \cdot g(v(H), y) \tag{1}$$

2. **Hitting**: Draw 1 card

$$\mathrm{EV}_{\text{hit}}(H, \sigma) = \sum_{r \in R} \frac{X_r}{N} \cdot \mathrm{EV}^*(H \cup \{r\}, \sigma) \tag{2}$$

   You draw one card rank r) with probability x_r/N. Update your hand to $H \cup \{r\}$. Choose the best action given the same number of splits remaining

3. **Doubling Down**: Double the wager, draw once and stand

$$\mathrm{EV}_{\text{dd}}(H) = \sum_{r \in R} \frac{X_r}{N} \cdot 2 \cdot \mathrm{EV}_{\text{stand}}(H \cup \{r\}) \tag{3}$$

4. **Splitting** (if $H = \{r, r\}$ and $\sigma \geq 1$)

   Let $X_{r_2}^{(-r_1)}$ denote the updated count of r_2 after removing r_1 from the deck.

$$\mathrm{EV}_{\text{split}}(H = \{r, r\}, \sigma) = \sum_{r_1 \in R} \sum_{r_2 \in R} \frac{X_{r_1}}{N} \cdot \frac{X_{r_2}^{(-r_1)}}{N-1} \cdot [\mathrm{EV}^*(\{r, r_1\}, \sigma - 1) + \mathrm{EV}^*(\{r, r_2\}, \sigma - 1)] \tag{4}$$

   $X_{r_2}^{(-r_1)}$ is the count of r_2 after removing the r_1 drawn from the first split. You draw 2 new cards, which couple with r_1 and r_2 separately to form 2 new hands: a right hand $\{r_1, r\}$ and left hand $\{r_2, r\}$. Each hand is played optimally with $\sigma - 1$ remaining splits. Since you now have 2 hands, their expected values are added together.

2.2.4 Optimal Action Value

At every game state we choose the expected value that maximizes our return

$$\mathrm{EV}^*(H, \sigma) = \max \left\{ \mathrm{EV}_{\text{stand}}(H), \mathrm{EV}_{\text{hit}}(H, \sigma), \mathrm{EV}_{\text{dd}}(H), [\mathrm{EV}_{\text{split}}(H, \sigma)]_{H=\{r,r\},\, \sigma \geq 1} \right\} \tag{5}$$

The objective function defines a recursive dynamic program that can be solved backwards from terminal hands (bust or stand) upwards.

3 Method: Markov Decision Process

A Markov Process refers to a process where the possibility of arriving at a certain state depends on the conditions of the current state. For example, the probability of us being able to split in the next state depends on whether or not we've exhausted all our possible splits in the current state.

However since the probability of the next state depends on the players decision and we want to maximize the probability obtaining the state with the highest expected value, we will be using a special case known as the Markov Decision Process (MDP) where outcomes are partly random and partly under the agents (e.g, the player) control

Let

States: $s = (H, U, X, \sigma)$, where:

- H: Player's current hand (multiset of ranks)
- U: Dealer's upcard
- $X = (X_r)_{r \in R}$: Remaining counts of each rank
- $\sigma \in \{0, 1, 2, 3\}$: Number of splits remaining

Actions: $A(s) = \{\text{Stand, Hit, Double, Split if } H = \{r, r\} \text{ and } \sigma \geq 1\}$

The probability of reaching a particular next state s' from $A(s)$ can be formulated by the following transition functions

Transitions:

- **Stand:** Dealer plays out. Transition to terminal state.
  $P(s' \mid s, \text{Stand}) = P_D(y \mid U, X)$, where s' encodes terminal outcome with dealer total y
- **Hit:** Draw one card r with probability $\frac{X_r}{N}$.
  $s' = (H \cup \{r\}, U, X^{(-r)}, \sigma)$
- **Double:** Draw one card r, then stand. Terminal transition.
  $s' =$ terminal with doubled reward, dealer plays out
- **Split:** If allowed, draw two cards r_1, r_2 without replacement.
  Create two hands: $(\{r, r_1\}, U, X^{(-\{r, r_1\})}, \sigma - 1)$ and $(\{r, r_2\}, U, X^{(-\{r, r_2\})}, \sigma - 1)$

Rewards: 0 for non-terminal steps. Terminal reward:

$$R = g(\text{player total, dealer total})$$

Doubled if action = Double. Intermediate transitions have reward 0. Terminal transitions deliver $R = g$(Player Total, Dealer Total) or doubled if action a = Double Down

Discount: $\gamma = 1$ Assume a risk factor of 1 since we care only about the total undiscounted payoff.[4]

3.1 Bellman Optimality Equations

Our goal can therefore be written as maximizing the expected value of state S

$$V^*(s) = \max_{a \in A(s)} Q(s, a)$$

where

$$Q(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + V^*(s') \right]$$

The expected value for each action can be given as:

- **Stand:**

$$Q(s, \mathrm{Stand}) = \sum_{y=17}^{22} P_D(y \mid U, X)\; g(v(H), y)$$

- **Hit:**

$$Q(s, \mathrm{Hit}) = \sum_{r \in R} \frac{X_r}{N}\; V^*(H \cup \{r\}, U, X^{(-r)}, \sigma)$$

- **Double:**

$$Q(s, \mathrm{Double}) = \sum_{r \in R} \frac{X_r}{N} \cdot 2 \sum_{y=17}^{22} P_D(y \mid U, X)\; g(v(H \cup \{r\}), y)$$

- **Split:**

$$Q(s, \mathrm{Split}) = \sum_{r_1, r_2 \in R} \frac{X_{r_1}}{N} \cdot \frac{X_{r_2}^{(-r_1)}}{N-1} \times \left[V^*(\{r, r_1\}, U, X^{(-\{r, r_1\})}, \sigma - 1) + V^*(\{r, r_2\}, U, X^{(-\{r, r_2\})}, \sigma - 1)\right]$$

3.2 Steps to Solve with Dynamic Programming

1. Initialize $V_0(s) = 0$ for all s, or to known terminal values.

2. Repeat until convergence:

$$V_{k+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + V_k(s')]$$

3. Policy extraction:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)[R(s, a, s') + V^*(s')]$$

3.3 Limitations

This requires we pre-compute $P_d(y|U, x)$ for each dealer upcard U and card-count x, iterate over all reachable states of (H, U, x, σ) and store their values. So that in a given state we lookup the optimal policy to take.

However, this is gravely computationally intensive because there can be $\binom{52D}{k}$ possible states for a given hand of size K. For example, suppose a game with a shoe of 6 decks, the number of possible combinations for the players initial hand is $\binom{312}{2}$=$48,516$. This count skyrockets as you allow k to grow as the player increases his hand count.

4 Method: Monte Carlo Tree Search for Blackjack MDP

Solving a Blackjack Markov Decision Process (MDP) optimally involves evaluating a vast number of game states and action sequences, which becomes computationally intractable under realistic card-counting assumptions. To approximate optimal play efficiently, we adopt a Monte Carlo Tree Search (MCTS) approach, which avoids exhaustive enumeration by simulating representative games and focusing computational effort on the most promising branches.

Monte Carlo methods have a long history in game analysis, originally popularized by Stanislaw Ulam to estimate the odds of winning solitaire through repeated simulation rather than exact computation. MCTS builds upon this philosophy by incrementally constructing a search tree guided by statistics collected through simulated rollouts.

In blackjack, MCTS approximates the action value function $Q(s, a)$ - the expected reward of taking action a in state s - by simulating the full game trajectories. This includes player decisions, as well as the dealer's play, conditioned on the current deck state. The dealer's terminal outcome distribution $P_d(y \mid U, x)$ is estimated dynamically using Monte Carlo rollouts: after each player-finalizing action (e.g., Stand or Double), the dealer's play is simulated from the current reduced shoe.

After sufficient simulations, the MCTS policy in each state converges toward an ϵ-optimal policy, effectively approximating the optimal strategy without solving the full MDP analytically.

Here's the basic loop MCTS runs over and over to build its search tree:

- **Selection:** Starting from the root node (the current game state), the algorithm traverses the tree by selecting child nodes that optimize the Upper Confidence Bounds for Trees (UCT) criterion:
$$a^* = \arg\max_a \left[Q(s, a) + c\sqrt{\frac{\ln n(s)}{n(s, a)}} \right],$$
where $n(s)$ is the visit count of state s, $n(s, a)$ is the visit count of action a from state s, and $c = \sqrt{2}$ controls the exploration-exploitation tradeoff [5]. Selection continues until a leaf node is reached.

- **Expansion:** If the selected leaf node is non-terminal, one or more child nodes are added to represent unexplored legal actions. The tree is allowed to grow to full game depth, which typically spans 4–6 steps depending on the sequence of player choices such as hits, splits, and doubles. No artificial depth limit is imposed.

- **Rollout**: From the expanded node, a complete hand is simulated using a heuristic policy (π_{rollout}), and the dealer plays according to standard house rules. Simulations terminate when the game ends in a win, loss, or push. To model the dealer's behavior accurately, we simulate N dealer rollouts using the current deck state, capturing card-removal effects and ensuring realistic outcome estimation. The heuristic policy, π_{rollout}, is defined by the basic strategy explained in Table 1.

Player Hand	**Dealer Upcard**	**Action**
Splitting Pairs		
Pair of 2s, 3s, 4s, 5s, 6s, 7s	2-7	Split
Pair of 8s	Any	Split
Pair of 9s	3-6, 8-9	Split
Pair of Aces	Any	Split
Standing		
Hard 21	Any	Stand
Hard 17-20	Any	Stand
Hard 12-16	3-6	Stand
Doubling Down		
Hard 9, 10, 11	2-9	Double
Hitting		
Hard ≤ 11	Any	Hit
Hard 12-16	2, 7-Ace	Hit

Table 1: Blackjack Rollout Strategy

- **Backpropagation:** The terminal result is propagated back through all nodes visited during the current iteration. Each node updates its visit count and cumulative reward, refining the estimated value $Q(s, a)$ used in subsequent simulations.

We let MCTS run this loop many times. Once we're out of time or have run enough iterations, we look at the children of the very first node (our starting state) and pick the move that leads to the child with the best stats (usually the highest win rate or the most visited). This is the move MCTS recommends for our current situation in the Blackjack game.

Let

- State $s = (H, U, X, \sigma)$, where:
-
  - H: Player hand (multiset of ranks)
  - U: Dealer upcard
  - $X = (X_r)_{r \in R}$: Remaining card counts
  - $\sigma \in \{0, 1, 2, 3\}$: Splits remaining
- Actions $A(s) = \{\mathrm{Stand}, \mathrm{Hit}, \mathrm{Double}\}$ plus Split if allowed.

4.1 Node Statistics

For each node s and action $a \in A(s)$:

- $n(s)$: visits of state s
- $n(s, a)$: times action a taken
- $W(s, a)$: total reward from those simulations
- $Q(s, a) = W(s, a)/N(s, a)$: average action value

4.2 Deck Integration

Use exact draw probabilities:

$$\Pr(r) = \frac{X_r}{N}, \quad X \to X^{(-r)}, \quad N \to N - 1.$$

Actions map to MDP transitions: Hit = sample one card, Stand = terminal + dealer simulation, Double = one draw then stand, Split = two subtrees.

4.3 Decision Rule

After M iterations,

$$\hat{Q}(s_0, a) = \frac{W(s_0, a)}{n(s_0, a)}, \quad a^* = \arg\max_a \hat{Q}(s_0, a).$$

This action approximates one that maximizes the true $Q^*(s_0, a)$ in the full MDP

4.4 Limitations

- Memory grows with visited nodes (much smaller than full state space).
- Time per iteration is tree depth + rollout length.
- With $M \to \infty$ and UCT, MCTS converges to optimal action.

5 Model Results

We ran 25 independent Blackjack simulations with $M = 1000$ MCTS iterations per hand. The agent begins with an initial bankroll of 100.00 dollars and bets a flat 10 dollars for every hand for up to 50 possible hands. Figures 1–3 summarize:

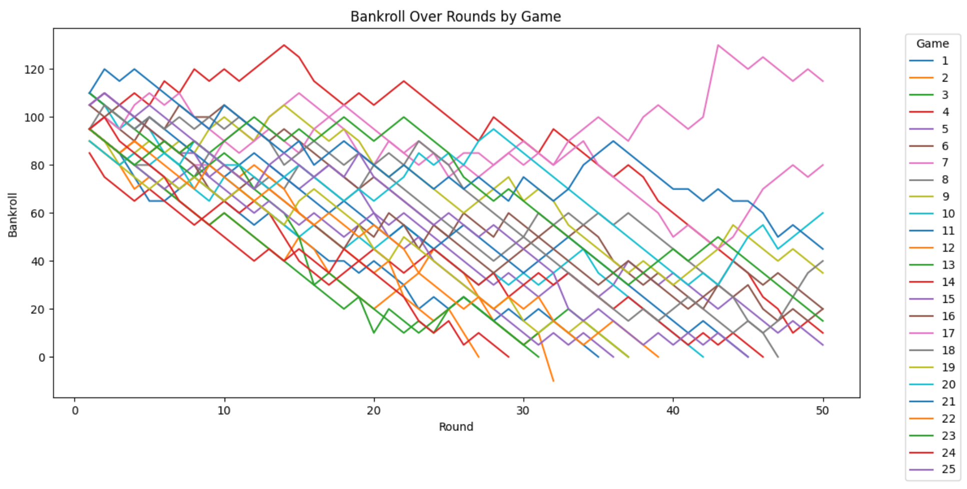


Figure 1: Bankroll over 25 simulated games.

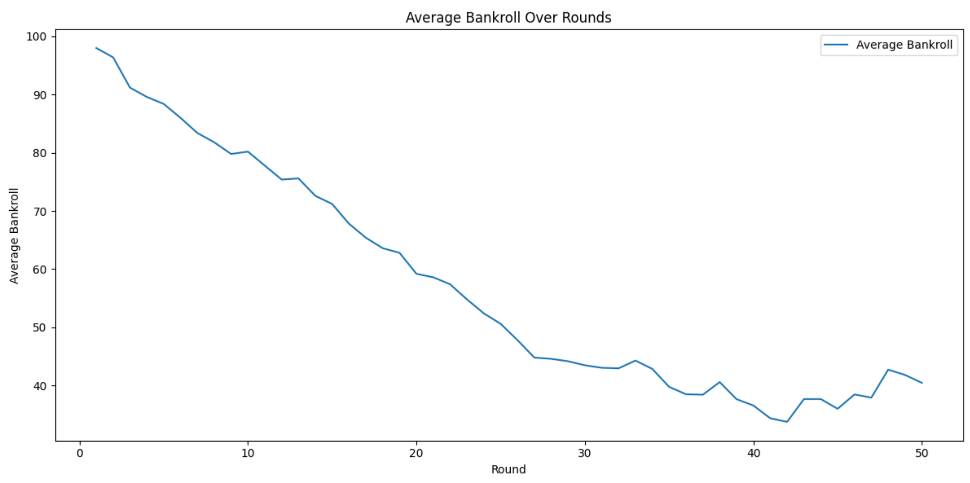
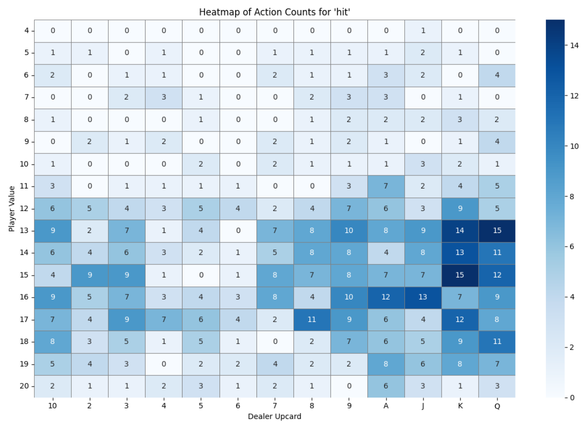
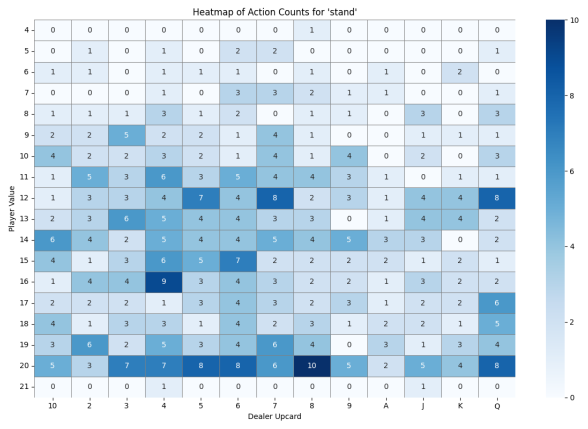


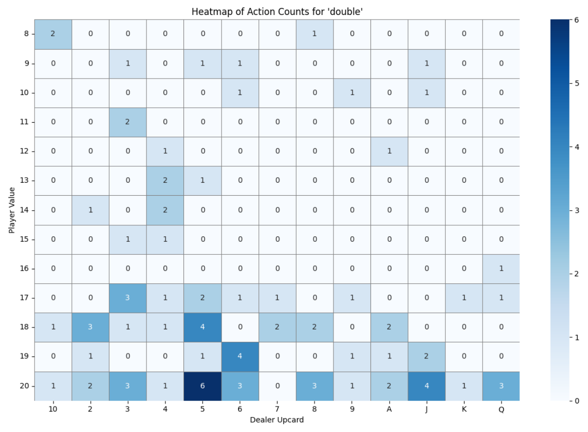
Figure 2: Average bankroll across all games.



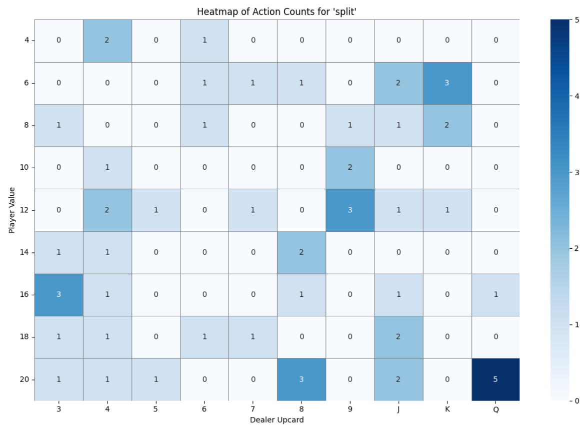
(a) Hit



(b) Stand



(c) Double



(d) Split

Figure 3: Heat-maps of action frequencies by player hand (y-axis) and dealer up-card (x-axis).

6 Recommendations

In this experiment, bet sizing is fixed and no surrender is allowed. Future work should let the agent vary its wager dynamically based on card counts and implement surrender when, given a certain risk appetite, the expected value is negative. Furthermore, the model should be evaluated against standard baselines such as basic strategy and Q-learning agents to contextualize its performance. Incorporating options like insurance and common side bets (e.g. Perfect Pairs, 21 + 3) would also make the simulation more comprehensive and aligned with real-world Blackjack scenarios, allowing the agent to learn richer policies under realistic casino conditions.

7 Conclusion

We presented a Monte Carlo Tree Search framework for solving a finite-deck Blackjack MDP, integrating Bayesian probability to model the evolving composition of the shoe. By maintaining and updating card draw probabilities after each action, the agent incorporates a form of Bayesian reasoning to condition future expectations on observed outcomes. This probabilistic foundation enables accurate dealer simulations, realistic player rollouts, and more informed action selection within the MCTS framework. Our results show that this approach can approximate near-optimal strategies without exhaustively evaluating the entire state space. Future work should extend this framework with dynamic bet sizing, surrender, insurance, side bets, and comparative evaluation against basic-strategy and Q-learning baselines to further assess its practical value in casino-style Blackjack.

A Codebase

Code and data are open source on GitHub:

- `https://github.com/ManuelLopezMe/Lets-Go-Gambling`
- Data: `https://github.com/ManuelLopezMe/Lets-Go-Gambling/data/main-run-1.json`

B Notation Summary

Symbol	Description
s	Current game state (including player's hand, dealer upcard, and shoe composition)
a	Action taken by the player (e.g., Hit, Stand, Double, Split)
$Q(s, a)$	Estimated expected return (EV) of taking action a in state s
c	UCT exploration constant, balancing exploration and exploitation ($c = \sqrt{2}$)
$n(s)$	Number of times state s has been visited
$n(s, a)$	Number of times action a has been taken from state s
U	Dealer's visible upcard
x	Current composition of the shoe (card counts or probability vector)
$P_d(y \mid U, x)$	Probability that the dealer ends with value y, given upcard U and shoe x
N	Total number of cards remaining in the shoe
π	Policy used to select actions during simulation

Table 2: Summary of notation used in the MCTS-based Blackjack model.

References

[1] Rob Pratt. Computing an Optimal Blackjack Strategy with SAS/OR. https://blogs.sas.com/content/operations/2016/06/20/computing-an-optimal-blackjack-strategy-with-sasor/, 2016.

[2] Connie Trojan. How to Lose Blackjack (Optimally). https://www.lancaster.ac.uk/stor-i-student-sites/connie-trojan/2022/05/05/how-to-lose-blackjack-optimally/, 2022.

[3] Aman Khurana. Blackjack. https://github.com/aman-khurana/blackjack, 2020.

[4] Andrew G. Barto ČRichard S. Sutton. Monte carlo methods. In *Reinforcement Learning: An Introduction*, pages 114–117. MIT Press, 2015.

[5] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006.