Monte Carlo Methods and Random Processes: Report on Implementation and Analysis

---

1. Introduction

Monte Carlo methods are a powerful class of computational algorithms that utilize random sampling to solve complex mathematical problems and estimate numerical results. By simulating large numbers of random trials, these methods provide a robust approach to modeling stochastic systems, optimizing decisions, and evaluating mathematical functions. Their versatility has led to widespread application in fields such as finance, physics, artificial intelligence, and operations research.

In this project, Monte Carlo methods are applied to the game of Blackjack, a quintessential example of a stochastic process where randomness governs the game's progression. The player's and dealer's decisions introduce layers of uncertainty, making it an ideal setting to showcase the effectiveness of Monte Carlo techniques. By leveraging episodic sampling and reinforcement learning principles, this project aims to develop an optimal policy for the player.

The primary objective of this report is to present a comprehensive exploration of the theoretical foundations, practical implementation, and critical insights gained from employing Monte Carlo methods in the context of Blackjack. Through visualizations, policy evaluation, and analysis of decision-making strategies, this report highlights how randomness can be harnessed to develop intelligent agents capable of navigating uncertain environments.

---

2. Objectives

1. **Understand Random Processes**: Explore how randomness influences decision-making in stochastic systems.
2. **Apply Monte Carlo Methods**: Use Monte Carlo prediction and control to optimize decision-making strategies in the Blackjack environment.
3. **Develop a Blackjack Agent**: Build an agent capable of learning an optimal policy using episodic reinforcement learning techniques.
4. **Visualize Results**: Present the state-value function and policy visually for better understanding and analysis.

---

3. Theoretical Background

3.1 Random Processes

A random process is a collection of random variables that evolve over time, where each variable represents a possible state of the system at a specific point in time. These processes are fundamental in modeling dynamic and unpredictable systems across various domains, such as natural phenomena, financial markets, and decision-based games. Unlike deterministic systems, random processes account

for uncertainty and randomness, making them essential for scenarios where future states are influenced by probabilistic events.

Random processes play a significant role in fields such as meteorology (to predict weather patterns), economics (to model stock price fluctuations), biology (to track population growth or genetic mutations), and queueing theory (to analyze customer arrival times in service systems). In the context of Blackjack, randomness is introduced through several factors, including card shuffling, player decisions (Hit or Stick), and dealer behavior. Each of these factors introduces variability in the game's progression, making Blackjack a classic example of a stochastic process.

By modeling Blackjack as a random process, it becomes possible to employ probabilistic techniques, such as Monte Carlo methods, to analyze and optimize decision-making. Through repeated sampling, these methods estimate the expected outcomes of different strategies, allowing for the development of an optimal policy that can adapt to the inherent uncertainty in the system. This concept extends to numerous real-world applications, demonstrating the versatility and power of random processes in navigating unpredictable environments.

3.2 Monte Carlo Methods

Monte Carlo methods are computational techniques that use repeated random sampling to compute the expected value of a function or to solve optimization problems. Unlike deterministic methods, Monte Carlo methods leverage randomness to discover optimal solutions. Key ideas include:

- **Prediction**: Estimating the value of states or actions.
- **Control**: Optimizing decisions to maximize cumulative rewards.
- **Exploration vs. Exploitation**: Balancing exploration of unknown actions with exploitation of known optimal actions using epsilon-greedy policies.

---

4. Implementation

The implementation of Monte Carlo methods involves several interconnected components, each playing a critical role in modeling, learning, and optimizing decision-making within a stochastic environment. At its core, the implementation focuses on simulating a dynamic system where uncertainty is inherent, allowing for the development of intelligent agents that can learn optimal strategies over time.

The project utilizes the Python programming language due to its flexibility and extensive support for scientific computing. Key libraries include **NumPy** for numerical computations, **Matplotlib** for data visualization, and **OpenAI Gym** for environment simulation. These libraries enable efficient data manipulation, graphical insights, and seamless interaction with pre-built stochastic environments like the Blackjack game.

Beyond Blackjack, the design of this implementation serves as a blueprint for tackling a wide range of real-world stochastic problems. Key aspects of the implementation include:

- **Environment Setup**: The environment defines the state-action space, transition dynamics, and reward system. In Blackjack, the player and dealer operate under probabilistic rules that create an unpredictable trajectory of the game.
- **Data Generation**: Using random sampling, the system generates episodes of state-action-reward sequences, which form the basis for learning.
- **Policy Representation**: Policies dictate the agent's behavior, balancing exploration (trying new actions) and exploitation (choosing the best-known action). This balance is achieved using epsilon-greedy strategies, which can be generalized to other decision-making problems.
- **State-Value and Action-Value Estimation**: To evaluate the effectiveness of decisions, the implementation computes the state-value function V(s)V(s)V(s) and the action-value function Q(s,a)Q(s, a)Q(s,a). These functions provide critical insights into the long-term benefits of being in a particular state or taking a specific action.
- **Policy Improvement and Control**: As the agent learns from experience, it updates its policy to achieve higher rewards. This is achieved using Monte Carlo control, where the agent refines its decisions after observing the results of its actions.
- **Visualization and Analysis**: Visualization of the policy and state-value function helps identify patterns in decision-making. This step can be applied to various applications, from finance to healthcare, where interpretability is crucial for understanding optimal decisions.

By structuring the implementation in this way, the Monte Carlo approach can be adapted to broader problems beyond Blackjack. From stock market predictions to resource allocation in logistics, this modular design offers a scalable framework for developing data-driven, decision-making agents that thrive in stochastic and uncertain environments.

---

5. Blackjack Environment

The Blackjack game environment is created using OpenAI's Gym library (gym.make('Blackjack-v0')). The key elements of this environment are:

- **State**: Consists of the player’s current sum, the dealer's showing card, and the presence or absence of a usable ace.
- **Actions**: The player can either "Hit" (draw another card) or "Stick" (keep the current total).
- **Reward**: The player receives +1 for winning, -1 for losing, and 0 for a draw.

---

6. Monte Carlo Implementation

6.1 Policy Visualization

Policy visualization plays a crucial role in understanding and interpreting the decisions made by a learning agent in a stochastic environment. It provides an intuitive way to observe how the agent's

strategy evolves and how optimal decisions change based on different states of the system. In the context of Blackjack, policy visualization offers insights into when the player should "Hit" (draw another card) or "Stick" (keep the current hand) to maximize the chance of winning.

Key Scenarios

Two key scenarios are visualized to illustrate how the player's decisions are influenced by the availability of a usable ace:

- **Usable Ace**: This scenario occurs when the player has an ace in their hand that can be counted as 11 without causing the player to "bust" (exceed a total of 21). This is a significant advantage since the ace introduces flexibility, allowing the player to toggle its value between 1 and 11 as needed.
- **No Usable Ace**: In this scenario, the player does not have an ace that can be counted as 11. The absence of this flexibility restricts the player's options, increasing the risk of busting if another card is drawn.

These two scenarios illustrate the impact of the ace on decision-making, highlighting how strategies can shift depending on the player's current hand.

Visualization Approach

The policy is visualized using **2D heatmaps**, which display the player's optimal decision ("Hit" or "Stick") for each possible state of the game. Each heatmap plots the player's current total (x-axis) against the dealer's visible card (y-axis). The color of each grid cell indicates the player's decision for that state, where:

- **"Hit" (1)** is represented by one color (e.g., green or blue), meaning the player should draw another card.
- **"Stick" (0)** is represented by a different color (e.g., red or orange), meaning the player should hold their current total and wait for the dealer's move.

This visual representation allows for a clear and immediate understanding of the strategy. States where "Hit" is optimal are easily distinguishable from those where "Stick" is preferred.

Policy Insights

1. **Impact of Player's Current Sum**:
    - For smaller sums (e.g., 11 to 16), the player is more likely to "Hit" since the probability of improving the hand without busting is high.
    - As the player's total approaches 21, the agent's optimal strategy tends to "Stick" more frequently, as the risk of busting outweighs the potential benefits of drawing another card.

2. **Impact of Dealer's Showing Card**:
    - If the dealer's visible card is strong (e.g., 10 or face cards), the player is more likely to "Hit" to improve their hand, anticipating the dealer might achieve a high total.
    - If the dealer's visible card is weak (e.g., 2 to 6), the player is more likely to "Stick," as the dealer is at greater risk of busting due to the rules of Blackjack (dealers must hit until they reach at least 17).
3. **Effect of a Usable Ace**:
    - The presence of a usable ace significantly alters the strategy. For example, with a usable ace, a player holding a total of 17 may be more inclined to "Hit" since they can reduce the ace's value from 11 to 1 if needed.
    - Without a usable ace, a total of 17 typically results in a "Stick" action since drawing another card carries a high risk of busting.

Broader Applications of Policy Visualization

While this visualization technique is applied to Blackjack, the approach can be generalized to other stochastic and decision-making problems. Examples include:

- **Robotics**: Visualizing an agent's movement strategy in a grid world or maze environment.
- **Healthcare**: Analyzing decision pathways for patient treatment strategies, showing when to "continue treatment" or "switch to a new intervention."
- **Finance**: Displaying optimal portfolio rebalancing strategies as a function of current market conditions.
- **Logistics**: Visualizing optimal routing policies for dynamic vehicle routing problems based on traffic conditions and package demands.

6.2 State-Value Function Visualization

What is the State-Value Function V(s)V(s)V(s)?

The state-value function V(s)V(s)V(s) is a measure of how "good" it is to be in a particular state sss. In simpler terms, it tells us the expected reward (or outcome) that can be achieved from that state if the player follows an optimal strategy. The equation for V(s)V(s)V(s) is: $V(s)=E[G_t|S_t=s]$

Where:

- V(s)V(s)V(s): The value of state s.
- G_t: The total discounted return starting from time t onward.
- S_t: The state at time t. For Blackjack, the state sis typically defined as the player's current hand value, the dealer's visible card, and the presence or absence of a usable ace.

Visualization Approach

To visualize V(s)V(s)V(s), **3D surface plots** are used, which provide a graphical representation of how the value of a state changes with respect to two key parameters:

- **Player's Current Total**: Represented on the x-axis, it shows the player's current sum (e.g., 11 to 21).
- **Dealer's Showing Card**: Represented on the y-axis, it shows the card that the dealer has revealed (e.g., 1 to 10, where 1 represents an Ace).

The z-axis represents V(s)V(s)V(s), the expected value of being in that state. Colors and elevation of the 3D surface highlight how favorable (or unfavorable) certain states are for the player.

How the Visualization Works

1. **3D Surface Plot**:
    - The player's current total (x-axis) ranges from 11 to 21 (since anything below 11 isn't a playable hand, and anything above 21 results in a "bust").
    - The dealer's visible card (y-axis) ranges from 1 (Ace) to 10 (face cards are treated as 10).
    - The state value V(s)V(s)V(s) (z-axis) is the agent's estimate of the total reward expected from being in that state.
2. **Two Separate Plots**:
    - **Usable Ace**: When the player has an ace that can be counted as 11.
    - **No Usable Ace**: When no usable ace is present in the player's hand. These two plots allow for a side-by-side comparison of how having a usable ace impacts player decisions and expected outcomes.
3. **Color Mapping**:
    - Warmer colors (e.g., red) may represent states with negative expected value, indicating that they are undesirable or risky.
    - Cooler colors (e.g., blue) may represent states with positive expected value, suggesting that the player is in a favorable position.

Key Insights from Visualization

1. **Impact of Player's Total**:
    - **Higher Totals Are More Desirable**: States where the player has a higher total (e.g., 19, 20, or 21) tend to have higher values, as these totals are closer to a win.

- **Risk of Busting**: For player totals near 21, the player is likely to "Stick" rather than "Hit" since the risk of busting is high. This is reflected in higher V(s)V(s)V(s) values for states near 21.

2. **Impact of Dealer's Showing Card**:
    - **Dealer's Weak Cards Are Beneficial**: When the dealer's card is low (e.g., 2 to 6), the player's state-value function tends to be higher. This is because the dealer is more likely to bust, giving the player an advantage.
    - **Dealer's Strong Cards Are Riskier**: When the dealer shows a 10 or an Ace, the player's state-value tends to be lower since the dealer is more likely to end up with a high total. The player may be forced to "Hit" more frequently, increasing the risk of busting.
3. **Effect of Usable Ace**:
    - **Increased Flexibility**: When the player has a usable ace, the value of states increases significantly because the player can avoid busting by treating the ace as a 1 instead of 11.
    - **Safety Net**: This "safety net" allows the player to take more risks when the player's total is lower (e.g., 13 to 17), as they can always reduce the ace's value. This leads to higher expected values in states where the player has a usable ace compared to when they do not.

---

7. Key Functions in the Code

7.1 generate_episode_from_limit_stochastic(bj_env)

This function generates episodes of the Blackjack game using a simple policy:

- If the player's sum is greater than 18, they stick with 80% probability.
- Otherwise, the player hits with 80% probability.

This policy ensures a balance of exploration and exploitation, allowing the agent to explore new actions while following a basic strategy.

7.2 mc_prediction_q(env, num_episodes, generate_episode, gamma=1.0)

This function estimates the action-value function Q(s,a)Q(s,a) using Monte Carlo prediction:

- **Input**: Number of episodes, discount factor γ\gamma, and a function for generating episodes.
- **Output**: Q-value function Q(s,a)Q(s,a) for each state-action pair.
- **Logic**: It calculates returns for each state-action pair in the episode and updates the average return to form the Q-value.

7.3 generate_episode_from_Q(env, Q, epsilon, nA)

This function generates an episode by following an epsilon-greedy policy:

- With probability ϵ\epsilon, a random action is chosen.
- With probability 1−ϵ1 - \epsilon, the action with the highest Q-value is chosen. This approach enables exploration of unseen actions while preferring optimal actions.

7.4 mc_control(env, num_episodes, alpha, gamma=1.0, eps_start=1.0, eps_decay=.99999, eps_min=0.05)

This function applies Monte Carlo control to find the optimal policy:

- **Input**: Number of episodes, learning rate α\alpha, epsilon decay parameters, and discount factor γ\gamma.
- **Output**: Optimal policy and Q-values.
- **Logic**: It iteratively updates the Q-value using the episode data and adjusts epsilon to balance exploration and exploitation.

---

8. Results and Visualization

8.1 State-Value Function

- **Visualization**: A 3D plot of the value function V(s)V(s) is presented, showing the player's current sum (x-axis), the dealer's showing card (y-axis), and the value of the state (z-axis).
- **Insight**: The optimal strategy emerges from the visualization, showing which states are favorable for the player.

8.2 Policy Heatmaps

- **Visualization**: Two heatmaps display the policy for "Hit" (1) or "Stick" (0) decisions.
- **Insight**: The player's decision changes depending on the dealer's visible card, the player's current total, and the presence of a usable ace.

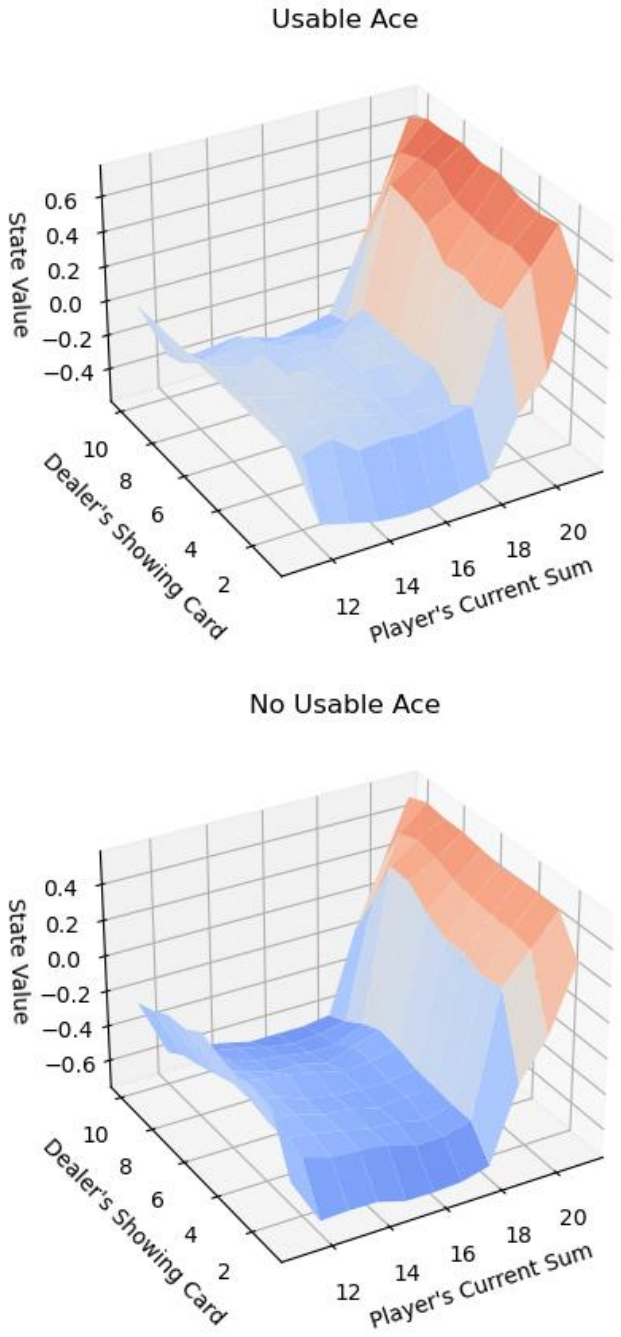

Figure 1: 3D Surface Plot of State-Value Function

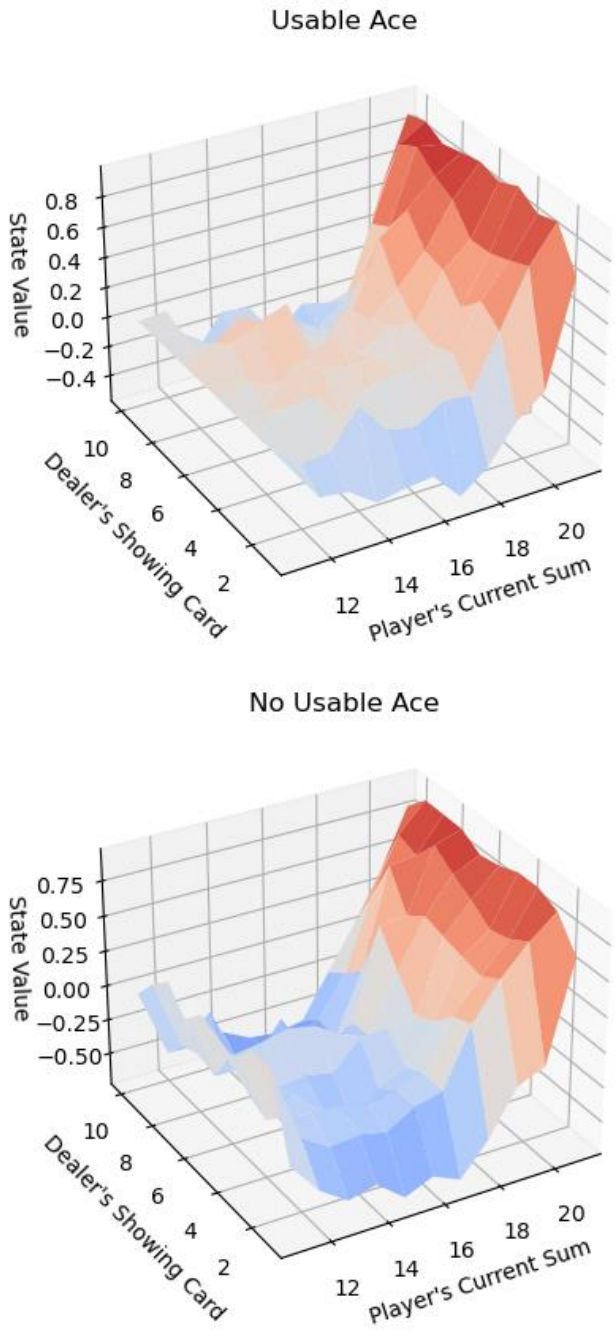

Figure 2: 3D Surface Plot of State-Value Function

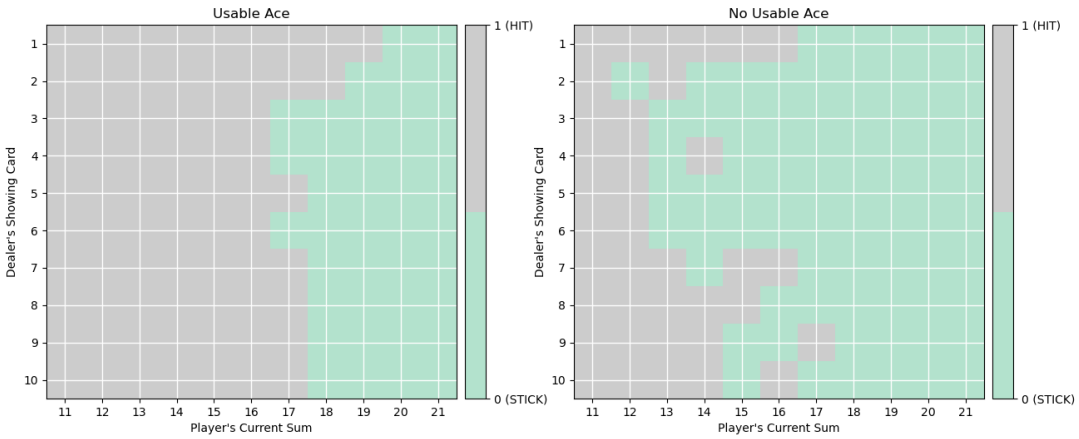

Figure 3: 2D Heatmap of the Optimal Policy

---

9. Insights and Discussion

- **Exploration vs. Exploitation**: The epsilon-greedy strategy enabled efficient learning by balancing the exploration of new actions with the exploitation of optimal actions.
- **Monte Carlo Control**: The iterative policy update gradually improved the performance of the agent, leading to an optimal policy.
- **Visualization**: 3D plots of V(s)V(s) and 2D policy heatmaps reveal the player's optimal strategy for different game states.
- **Stochastic Nature**: The randomness in Blackjack (card shuffling, player decisions) was effectively modeled using random processes, and randomness-driven exploration ensured the learning of robust policies.

---

10. Challenges and Limitations

- **High Variance in Estimates**: Since Monte Carlo methods rely on random sampling, the value estimates can have high variance.

- **Convergence Speed**: Convergence to the optimal policy may require a large number of episodes, as randomness plays a major role in learning.
- **Exploration vs. Exploitation Dilemma**: The challenge of balancing exploration and exploitation is critical in reinforcement learning. Epsilon decay helps achieve this balance.

---

11. Applications of Monte Carlo Methods

Beyond Blackjack, Monte Carlo methods are widely used in:

- **Finance**: To model stock price movements.
- **Physics**: To simulate complex systems like particle interactions.
- **Queueing Theory**: To model and optimize queue management.
- **Artificial Intelligence**: In reinforcement learning to train intelligent agents.

---

12. Conclusion

This project highlights the power and versatility of Monte Carlo methods and random processes in solving complex stochastic problems, using Blackjack as a case study. By harnessing the principles of exploration, randomness, and episodic learning, the agent was able to develop an optimal policy that maximizes cumulative rewards. The use of randomness was not merely a tool for simulation but a driving force for effective exploration, allowing the agent to discover superior strategies that may have been overlooked through deterministic approaches.

Monte Carlo methods proved to be a robust framework for both policy evaluation and policy improvement, offering a structured way to estimate state and action values without requiring explicit knowledge of the environment's transition dynamics. This flexibility makes Monte Carlo techniques applicable to a wide range of real-world problems where exact models are difficult to obtain.

The visualization of state-value functions and policy heatmaps provided valuable insights into how the agent's strategy evolves over time. Key decision points were identified, and the influence of critical variables—like the presence of a usable ace or the dealer's showing card—was made explicit. This form of interpretability is vital in applications where understanding the logic behind an agent's decisions is essential.

Beyond Blackjack, the lessons learned from this project are applicable to broader areas such as reinforcement learning, robotics, healthcare, and financial modeling. By employing randomness to explore potential solutions, agents can develop more adaptable, flexible, and optimal decision-making strategies in dynamic environments.

In summary, this project demonstrates that randomness, when properly managed, is not a hindrance but a catalyst for discovery. Monte Carlo methods, with their capacity for exploration and adaptation, remain a cornerstone of modern reinforcement learning, enabling intelligent agents to thrive in complex, uncertain, and ever-changing environments.

---

13. References

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.).
2. OpenAI Gym Documentation: https://www.gymlibrary.dev/
3. Matplotlib Visualization Library: https://matplotlib.org/
4. NumPy: The fundamental package for scientific computing with Python: https://numpy.org/
5. Pease, C. (2018, September 6). An Overview of Monte Carlo Methods. Medium; Towards Data Science. https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694
6. Agent, T. P. (2020, April 25). Monte Carlo Methods — Learning from experience - Analytics Vidhya - Medium. Medium; Analytics Vidhya. https://medium.com/analytics-vidhya/monte-carlo-methods-learning-from-experience-1bfce2d53313
7. Bajaj, A. (2022, July 21). Monte Carlo Simulation: A Hands-On Guide. Neptune.ai. https://neptune.ai/blog/monte-carlo-simulation
8. Pelin Okutan. (2024, January 12). Monte Carlo Simulation: A Comprehensive Guide - Pelin Okutan - Medium. Medium. https://medium.com/@pelinokutan/monte-carlo-simulation-a-comprehensive-guide-67ba39028831
9. Byrne, D. (2018, November 7). Learning To Win Blackjack With Monte Carlo Methods - Towards Data Science. Medium; Towards Data Science. https://towardsdatascience.com/learning-to-win-blackjack-with-monte-carlo-methods-61c90a52d53e
10. Yijie, A. (2019, November 18). Fundamentals of Reinforcement Learning: Understanding Blackjack Strategy through Monte Carlo Methods. Medium; GradientCrescent. https://medium.com/gradientcrescent/fundamentals-of-reinforcement-learning-understanding-blackjack-strategy-through-monte-carlo-88c9b85194ed
11. Adelson, A. (2017, June 22). Using Monte Carlo to Answer a Blackjack Question - Andrew Adelson - Medium. Medium. https://medium.com/@andrewadelson/using-monte-carlo-to-answer-a-blackjack-question-part-1-b5714e4592cb
12. Vyacheslav Efimov. (2024, May 23). Reinforcement Learning, Part 3: Monte Carlo Methods. Medium; Towards Data Science. https://towardsdatascience.com/reinforcement-learning-part-3-monte-carlo-methods-7ce2828a1fdb

13. Gianitsos, T., Isogawa, M., & Mendoza, D. (n.d.). CS238 Project Monte Carlo Blackjack. https://web.stanford.edu/class/aa228/reports/2019/final69.pdf

14. Chiswick, M. (2020, September 21). Monte Carlo RL and Blackjack. Max Chiswick. https://chisness.github.io/2020-09-21/monte-carlo-rl-and-blackjack

---

Video link : https://youtu.be/K8Mxj1uDIGM
N.b – if you can not access the video link please email me at kazi-mushfiq.rafid.358@my.csun.edu or rafid109@gmail.com