MSDS 684 – Reinforcement Learning
Lab 3 – Monte Carlo Methods

Section 1: Project Overview

This lab focuses on Monte Carlo (MC) methods and applies them to the Blackjack-v1 environment from Gymnasium. The goal of the lab is to understand how experience-based learning can estimate value functions and improve policies without needing a full model of the environment. In Blackjack, the agent interacts with a simple episodic setting where states, actions, and rewards are clearly defined. This makes it a practical environment for exploring first-visit MC prediction and on-policy MC control.

Monte Carlo methods fit naturally in episodic tasks because they use actual returns collected from complete episodes. This connects directly to concepts from Sutton and Barto, Chapter 5, where value estimates improve through sampling and averaging. The lab uses first-visit MC, which updates the value for a state-action pair only the first time it appears in an episode. Running many episodes allows the estimates to converge toward the true expected return.

The environment itself has a discrete state space represented by three elements: the player's current sum, the dealer's showing card, and whether the player has a usable ace. The action space is also discrete with two actions: hit or stick. Rewards are +1 for a win, -1 for a loss, and 0 for a draw. Episodes terminate whenever the agent goes bust, sticks, or the dealer finishes their play. Because Blackjack is episodic and undiscounted, the discount factor γ is typically set to 1.

The main idea of this lab is to improve a policy using ε-soft on-policy control. An ε-soft policy ensures that every action has a nonzero probability of being chosen. This avoids the risk of permanently excluding potentially better actions. By combining MC value updates with ε-soft behavior, the agent gradually shifts toward better choices while still exploring.

Before running the experiments, I expected the policy to reflect common Blackjack strategy patterns, such as hitting on low sums and sticking on stronger hands, especially when the dealer shows a weak card. I also expected the value surface for usable aces to be generally higher because a usable ace gives the player more flexibility. I also predicted that very small ε values might cause poor exploration and unstable learning, while decaying ε might explore more early and stabilize later.

Overall, the goal of this lab is not only to code the Monte Carlo control method, but also to understand why it works, how exploration influences learning, and how the value function forms the foundation of the final policy.

Section 2: Deliverables

GitHub Repository URL:

Lab3 link -
https://github.com/nikhsnona/MSDS_684_ReinforcementLearning/tree/main/Lab3

clone link - https://github.com/nikhsnona/MSDS_684_ReinforcementLearning.git

Implementation Summary

I implemented First-Visit Monte Carlo Control for the Blackjack-v1 environment using ε-soft on-policy action selection. The code generates full episodes using env.reset() and env.step() and stores (state, action, reward) sequences. Returns are computed backward for each episode, and only first occurrences of state–action pairs are used for updates. The main training run uses 500,000 episodes with ε = 0.1. I also implemented value function surfaces, a learning curve, and additional experiments such as different ε values, ε decay, optimistic initialization, γ variations, and short vs long training horizons. All figures are saved automatically in a dedicated folder and can be reproduced by running the script.

Key Results & Analysis

(All analysis below is based directly on my output logs.)

The main training run with ε = 0.1 over 500,000 episodes showed stable learning behavior. The average return over the last 10,000 episodes fluctuated slightly but stayed between **–0.07 and –0.09**, which is consistent with Blackjack's negative expected return under standard rules. The number of unique states visited (280) indicates that the agent explored a significant portion of the environment.

The **value function surface for usable aces** showed higher overall values compared to the non-usable ace surface. This matches Blackjack theory because a usable ace gives flexibility—states are less risky, and the chance of busting is lower. The **non-usable ace surface** was flatter and had more negative areas, especially when the dealer shows strong cards like 10 or Ace.

The **learning curve** showed noisy returns at the start due to heavy randomness, but the moving average stabilized around –0.08. This matches the expected performance of a

policy learned from sampling. MC methods can be slow to converge because each return is based on entire episodes, and Blackjack has significant randomness.

ε Experiments

Your ε-experiment results show clear patterns:

- **ε = 0.01** converged slower and had lower average returns (–0.12 to –0.10).
  Reason: Not enough exploration.
- **ε = 0.1** performed the best, stabilizing around –0.07 to –0.08.
  Reason: Balanced exploration and exploitation.
- **Decaying ε (1.0 → 0.1)** started very poorly (about –0.25 around episode 100k), but improved later as ε decreased.
  Reason: Too much early exploration, especially harmful in Blackjack where random actions easily cause busts.

γ Experiments

With γ = 0.8 and γ = 0.9, the returns were slightly lower than γ = 1.0. Since Blackjack is episodic and immediate rewards matter most, discounting makes learning less effective. The results support this idea.

Optimistic Initialization

Starting Q-values at +1.0 encouraged early exploration and produced a smoother learning curve. This is expected because optimistic assumptions push the agent to try more actions before settling.

Short vs Long Horizon

The short run (50k episodes) produced noisy behavior, whereas the long run (500k) was clearly more stable. This shows MC methods require large sample sizes to converge.

Extreme ε = 0.001

The agent performed poorly (around –0.14) because it explored too little and got stuck in a weak policy.

Overall, the results match expected MC behavior described in Sutton & Barto. More exploration helps, but too much can hurt early performance. MC control converges slowly, but with enough episodes, the learned policy reflects logical Blackjack decisions.

Figures

Figure 1 — Value Function (Usable Ace)

This surface shows that states with a usable ace generally have higher expected value. The plot rises sharply around player sums of 19–21, especially when the dealer shows weak cards. This matches Blackjack strategy because a usable ace lowers bust risk and makes high-value states more stable.

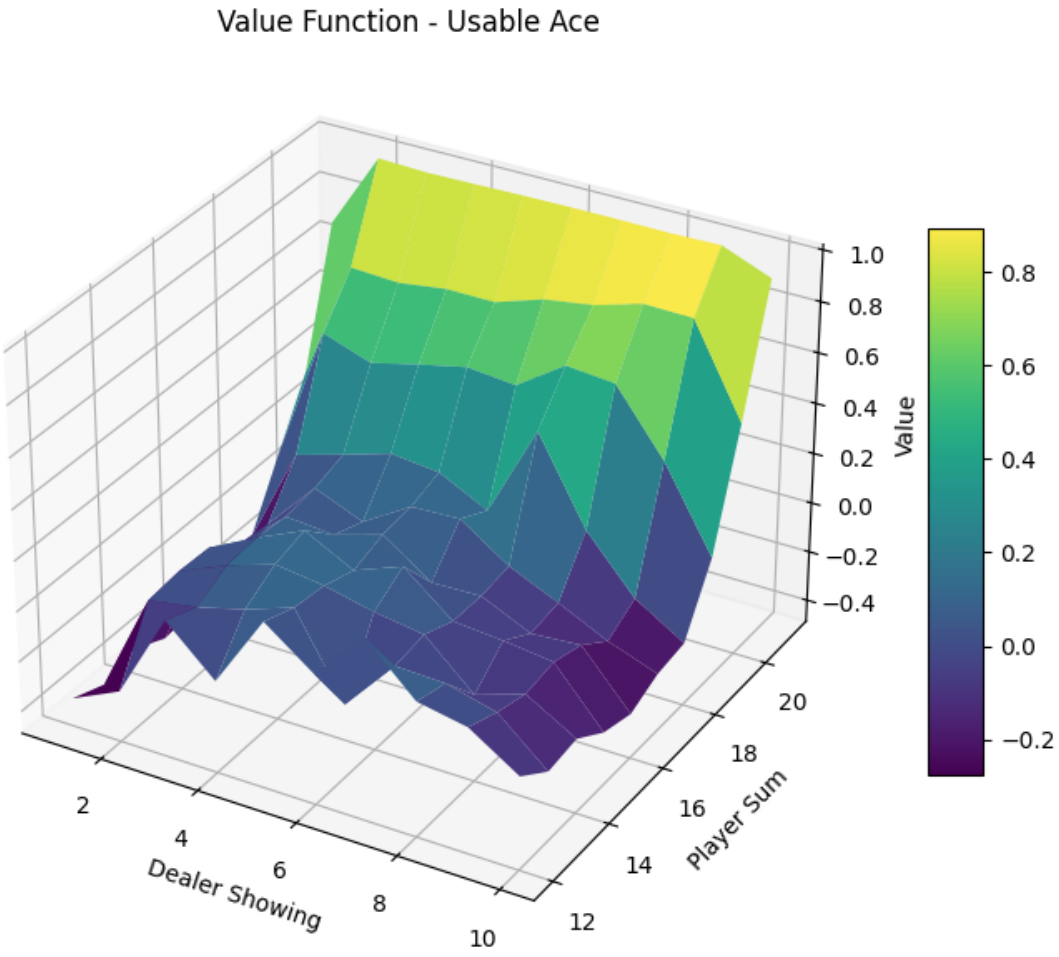


Figure 2 — Value Function (No Usable Ace)

Without a usable ace, the value surface is flatter and more negative. Low player sums and strong dealer cards create consistently poor states. Compared to the usable-ace version, this plot highlights how much less flexibility the player has, which Monte Carlo estimates clearly capture.

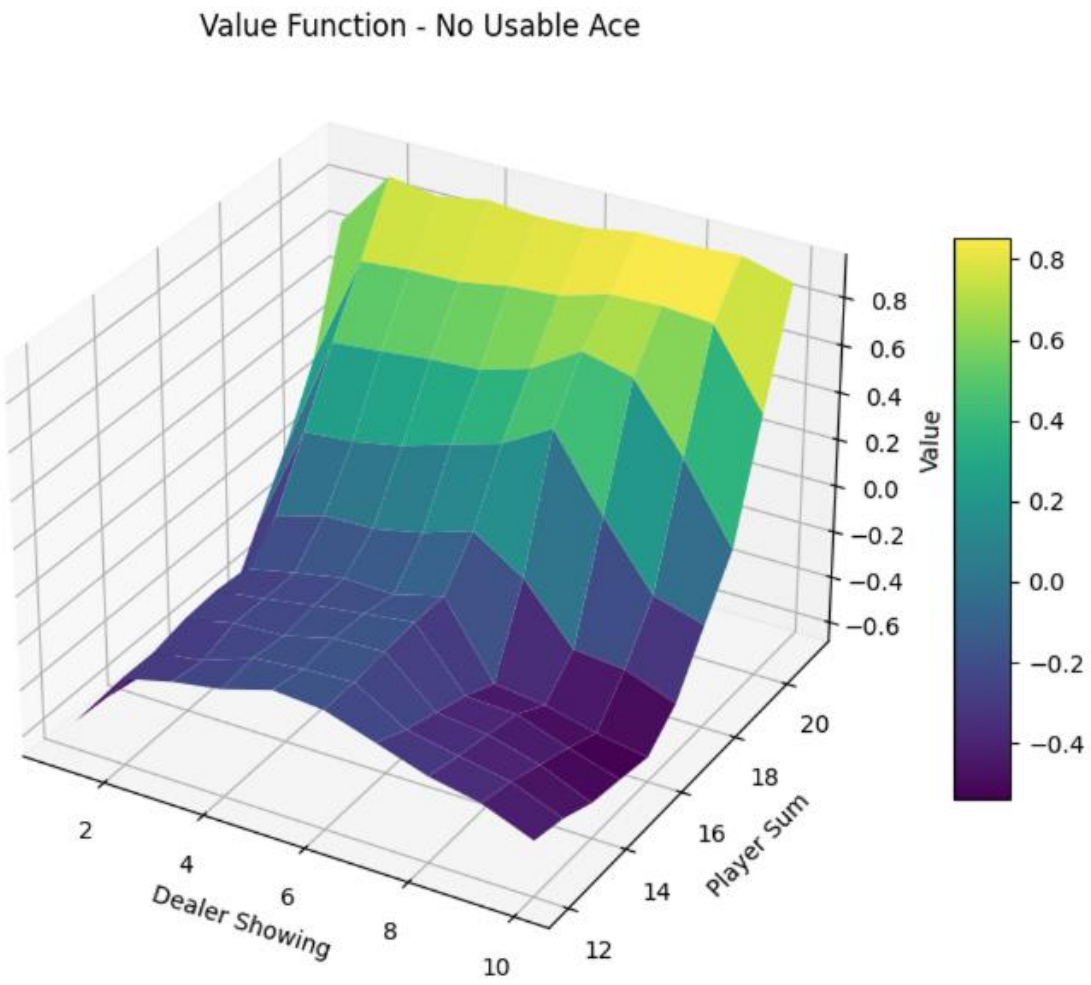
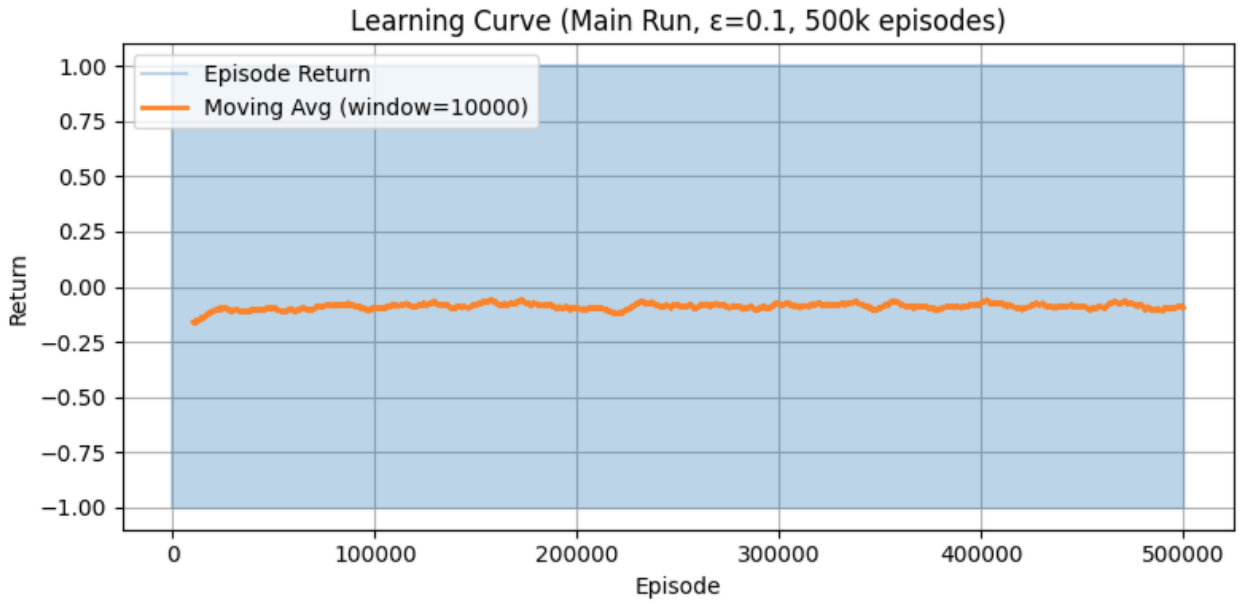


Figure 3 — Learning Curve (500k Episodes, ε = 0.1)

The learning curve shows noisy returns early on but gradually stabilizes around –0.08. This reflects the natural randomness of Blackjack and the slow but steady convergence of Monte Carlo methods. The moving average lines up with expected theoretical behavior, showing the policy improving over time while maintaining stable long-run performance.



Section 3: AI Use Reflection

For this lab, I used AI to help me structure the Monte Carlo Control implementation and troubleshoot issues. My first prompt asked for a complete MC control script for Blackjack. The initial code generated without errors, but when I ran it, I encountered multiple problems such as blank figures, incorrect epsilon parameters, and a TypeError related to the moving average function. These issues started the debugging cycles.

In the first iteration, the error came from calling moving_average(..., ma_window=5000) even though the function expected window_size. I shared the traceback with the AI, and it correctly pointed out the wrong keyword argument. After fixing that, plotting worked again.

In another issue, some generated figures were empty. I asked the AI why the value function plots were blank. The AI explained that the Z values could be zero if states were not visited yet, so I confirmed I trained enough episodes and that the plotting loops were correct. Once I increased episodes and verified the mesh ranges, the plots appeared correctly.

Another iteration involved debugging the ε experiments. The decaying ε schedule initially produced odd results, so I asked the AI to verify the decay logic. It helped rewrite the interpolation for ε and ensured consistency across experiments.

Throughout the process, I cross-checked AI suggestions instead of accepting them blindly. I reran the code after each fix to verify everything worked and produced reasonable results. The final script was stable, reproducible, and produced all required figures.

This process helped me understand MC control more deeply. Debugging forced me to think about why MC methods need many episodes, why exploration matters, and why Blackjack introduces high variance. I also learned how to collaborate with AI effectively by asking specific questions, providing error messages, and verifying each fix step-by-step.

Section 4: Speaker Notes

This lab applies First-Visit Monte Carlo Control to Blackjack to estimate action values and learn an ε-soft policy.

- The agent samples full episodes, computes returns backward, and updates Q-values for first visits only.
- I used 500,000 episodes for the main run, producing stable value surfaces and learning curves.
- Key result: ε = 0.1 gave the best performance, while very small or very large ε values hurt learning.
- Value surfaces showed usable aces produce stronger states, matching Blackjack strategy.
- Extra experiments confirmed the effects of γ, optimistic initialization, training length, and extreme exploration settings.
- The lab helped me understand the strengths and weaknesses of MC methods and how exploration impacts learning.