Monte Carlo Simulations

Student Name: Kratin Gupta
Roll No: 24B2208

WiDS 5.0 Project Report

Febuary 2026

Contents

1 Week 1: Basic Game Engine Development

The foundation of the project was a robust, object-oriented implementation of Blackjack. The goal was to create a playable game that could later double as a training environment for an AI agent.

1.1 Implementation

The game logic was split into modular classes to ensure maintainability:

- **Card & Deck Classes:** Implemented utilizing Python's `Enum` for suits to prevent data errors. The deck logic includes a `reset()` method to refill and shuffle, supporting the "Infinite Deck" assumption required for Markov properties.
- **Hand Class:** Contains the critical logic for calculating scores. It dynamically adjusts the value of Aces (11 to 1) to prevent busting, a crucial edge case in Blackjack rules.
- **BlackjackEnv Class:** Adapted specifically for Reinforcement Learning. Unlike a standard game loop, this class utilizes a `step(action)` method that returns a standard RL tuple `(state, reward, done)`.

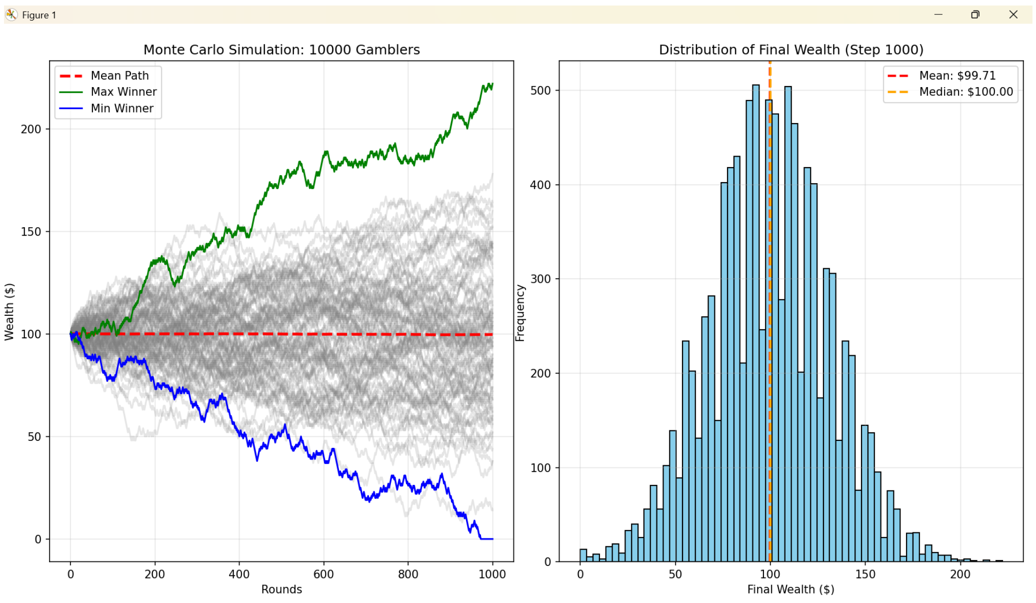


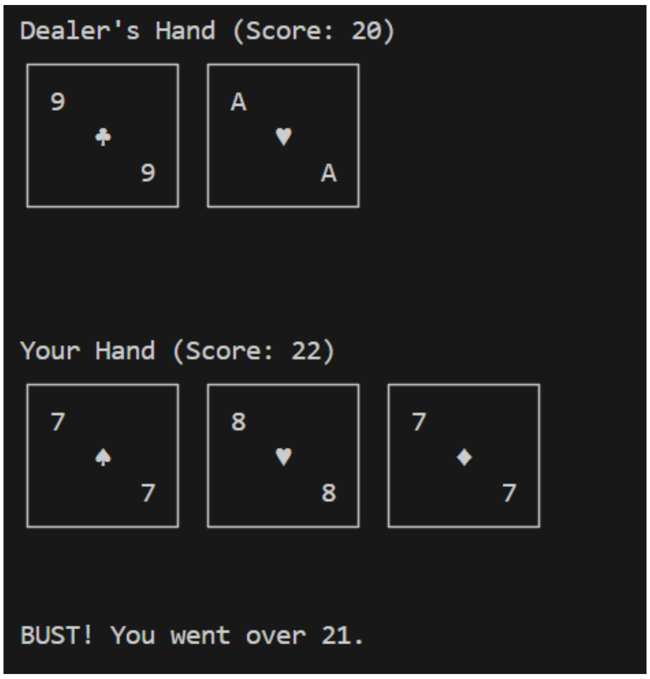
Figure 1: Gambler's ruin

1.2 State Representation

To make the game compatible with RL algorithms, the state was distilled into a compact tuple:

$$s = (\text{Player Sum}, \text{Dealer Show Card}, \text{Usable Ace})$$

This representation captures all necessary information for decision-making under standard casino rules.



```
Dealer's Hand (Score: 20)
┌─────────┐ ┌─────────┐
│ 9       │ │ A       │
│    ♣    │ │    ♥    │
│       9 │ │       A │
└─────────┘ └─────────┘


Your Hand (Score: 22)
┌─────────┐ ┌─────────┐ ┌─────────┐
│ 7       │ │ 8       │ │ 7       │
│    ♠    │ │    ♥    │ │    ♦    │
│       7 │ │       8 │ │       7 │
└─────────┘ └─────────┘ └─────────┘


BUST! You went over 21.
```

Figure 2: Example Hand

2 Week 2: Monte Carlo Estimation

Before tackling Blackjack, I explored the mathematical foundations of Monte Carlo methods—using random sampling to solve deterministic problems.

2.1 Geometric Integration

I implemented a modular simulation engine `run_simulation` capable of integrating various shapes by defining predicate functions:

- **Circle (π):** Estimated by sampling points within $x^2 + y^2 \leq 1$.
- **Parabola:** Integrated the area under $y = x^2$.
- **Gaussian:** Estimated the area under e^{-x^2} and compared it against the analytical Error Function (`erf`).

2.2 Euler's Number Estimation

I compared two distinct Monte Carlo approaches for estimating e:

1. **Area Method:** Integration of $1/x$.
2. **Forsythe's "Magic" Method:** A probabilistic approach based on the sequence lengths of sorted random numbers.

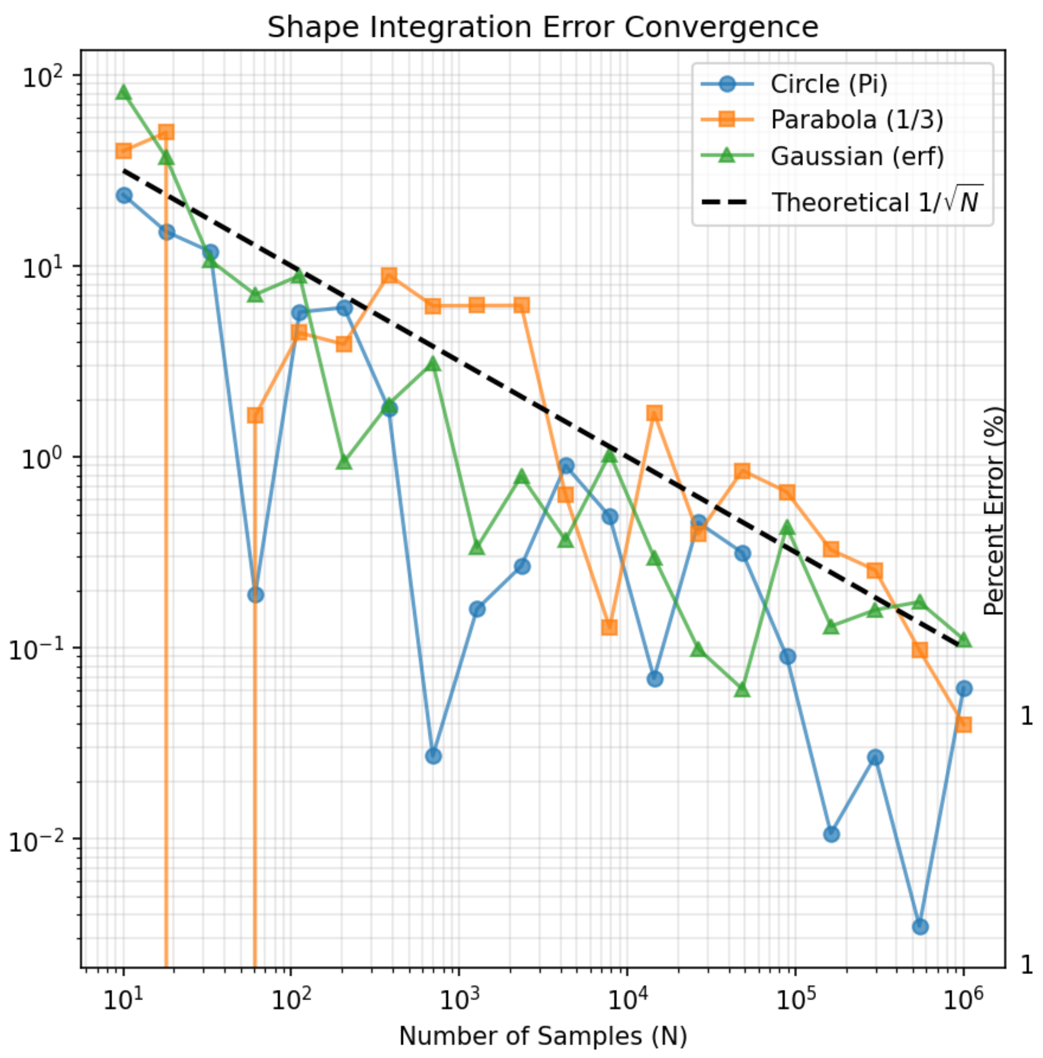


Figure 3: Convergence of π estimate over logarithmic scale.

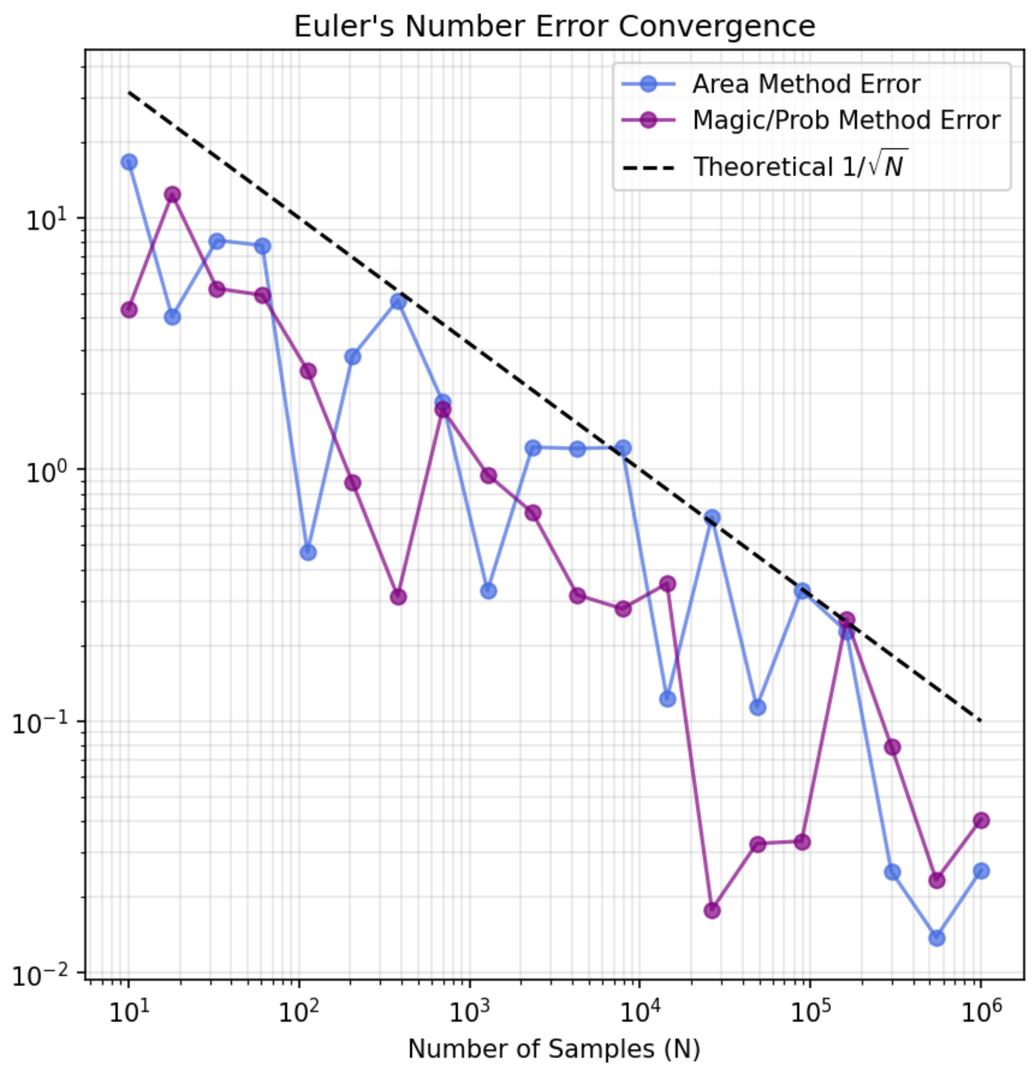


Figure 4: Error rate decreasing following theoretical $1/\sqrt{N}$ law.

3 Week 3: Reinforcement Learning Foundations

The third week focused on the theoretical framework of Markov Decision Processes (MDPs).

3.1 The Reality Check: Non-Stationarity

A key theoretical hurdle in applying RL to Blackjack is the **Markov Property**: *"The future depends only on the present, not on the past."*

- **Infinite Deck (Simulated):** Probabilities remain constant (e.g., drawing a King is always 1/13). The environment is stationary and Markovian.
- **Real Casino (Shoe):** Probabilities change as cards are removed. A state of $(12, 10)$ in a "hot deck" is mathematically different from the same state in a "cold deck."
- **Solution:** To apply RL in a real casino, the state space must be expanded to include the **Running Count**. However, this introduces the **Curse of Dimensionality**, expanding the state space by roughly $40\times$ and requiring significantly more training data.

3.2 Manual Monte Carlo Calculation

To verify understanding of the update rules, I performed a manual calculation for the trace $A \to B \to A \to$ Terminate with rewards $(+1, -1, +10)$:

- **First-Visit MC:** The return G for state A is calculated from its *first* appearance ($t = 0$).

$$G = 1 - 1 + 10 = 10$$

- **Every-Visit MC:** State A is updated twice.
    1. First instance ($t = 0$): $G = 10$
    2. Second instance ($t = 2$): $G = 10$ (from the final step).

4 Week 4: Prediction and Control

The final week integrated all components to train the agent.

4.1 Task 1: Monte Carlo Prediction

I evaluated a "High Risk" fixed policy: *"Stick only on 20 or 21, Hit on everything else."* Running 10,000 episodes with First-Visit Monte Carlo yielded intuitive results:

- **State 21:** High positive value (approx +0.9), confirming it is a strong winning state.
- **State 5:** High negative value. While usually a safe starting hand, the rigid policy forced the agent to hit until at least 20, leading to frequent busts.

4.2 Task 2: Monte Carlo Control

I implemented an **On-Policy Epsilon-Greedy Control** algorithm to learn the optimal policy π_*.

The Learning Algorithm

1. **Exploration:** Used $\epsilon = 0.05$ to ensure the agent explored random actions 5% of the time, preventing it from getting stuck in suboptimal loops.
2. **Q-Value Update:**
$$Q(s, a) \leftarrow Q(s, a) + \alpha[G - Q(s, a)]$$
Using a learning rate $\alpha = 0.01$ allowed for stable updates amidst the high variance of Blackjack rewards.

4.3 Results & Convergence

The agent's average reward started low (random play) and gradually climbed, stabilizing around a break-even point (approx -0.05), which is optimal for a casino game with a house edge.

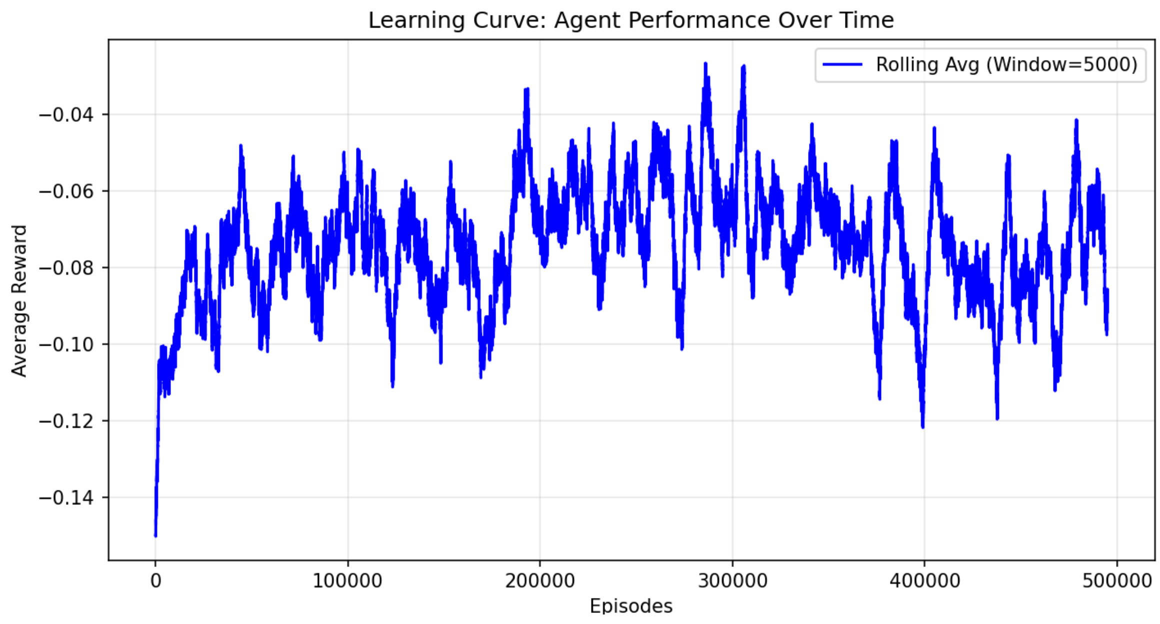


Figure 5: The agent's performance started at random (−0.3 reward) and improved to optimal play.

4.4 Learned Strategy

The final policy (Figure 6) reproduced "Basic Strategy":

- **Hard Hands:** The agent learned to *Stick* on 12–16 if the dealer showed a weak card (2–6), but *Hit* if the dealer showed a strong card (7–Ace).
- **Soft Hands:** The agent learned to be aggressive with Usable Aces, hitting on soft 17s and 18s to maximize the potential of the hand.

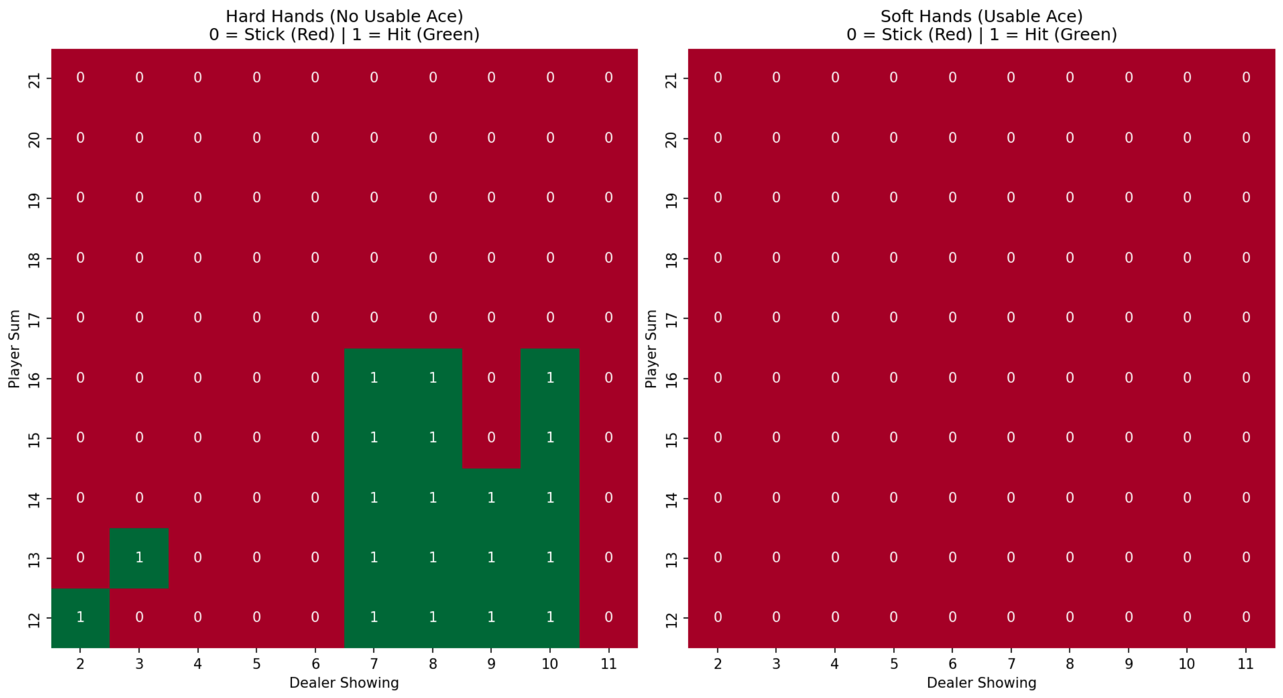


Figure 6: The learned policy matches 'Basic Strategy'. Red = Stick, Green = Hit.

5 Conclusion

This project successfully demonstrated the power of Monte Carlo methods. By observing raw experience—wins, losses, and draws—the agent derived complex, mathematically optimal behavior without ever being explicitly programmed with the rules of Blackjack. This reinforces the core tenet of Reinforcement Learning: intelligent behavior can emerge from simple trial-and-error interaction with an environment.