Monte Carlo Simulation of Blackjack

Abstract

Blackjack is one of the most widely played casino card games, making it a natural subject for studying probability and game fairness. In this project, I implemented a simplified version of Blackjack in Java using object-oriented programming concepts, including custom classes for Card, Deck, Hand, and Blackjack, with ArrayLists to manage collections of cards. I then used Monte Carlo simulations to run 100, 1000, and 10000 games, tracking the frequency of player wins, dealer wins, and ties. My results show that the dealer consistently has a statistical advantage over the player, but outcome percentages stabilize as the number of simulated games increases, demonstrating both the house edge and the power of large-scale simulation for analyzing game systems.

Results:

- Experiment

To evaluate the behavior of my Blackjack implementation, I ran a series of Monte Carlo simulations with different numbers of trials. Each simulation calls the game(false) method, which plays out a complete round of Blackjack without verbose output. I recorded the outcomes as player wins, dealer wins, or draws. I also inspected sample verbose logs of individual games to confirm that the game flow, turn-taking logic, and bust conditions worked correctly. These logs showed realistic scenarios, such as the dealer busting with a hand value of 22, the player busting with a hand value of 23, and the dealer winning by achieving a higher final total than the player.

- Hypothesis

Based on the rules of Blackjack, I expected that the dealer would win slightly more often than the player, since the dealer benefits from going second and only has to play until reaching a hand value of at least 17. I also expected the draw rate to be relatively low compared to wins and losses, since it requires both the player and dealer to end with the same total without busting.

- Table of results

The following tables show the results of 100, 1000, and 10000 simulated Blackjack games.

Table 1. Results of 100 simulated Blackjack games.

Outcome	Count	Percentage
Dealer wins	50	50.0%
Player wins	42	42.0%
Draws	8	8.0%

This table shows the results of 100 simulated Blackjack games. The dealer wins slightly more often than the player (50.0% vs. 42.0%), and draws are relatively rare (8.0%). This indicates the dealer's structural advantage, even in a small number of trials.

Table 2. Results of 1000 simulated Blackjack games.

Outcome	Count	Percentage
Dealer wins	499	49.9%
Player wins	419	41.9%
Draws	82	8.2%

This table shows the results of 1000 simulated Blackjack games. After simulating 1,000 games, the dealer won 499 times (49.9%), the player won 419 times (41.9%), and 82 games ended in a draw (8.2%). These percentages align closely with the expected ranges outlined in the project specifications (dealer ≈ 49%, player ≈ 41%, draws ≈ 8%). The percentages align closely with expected theoretical ranges, indicating correct implementation of the game logic.

Table 3. Results of 10000 simulated Blackjack games.

Outcome	Count	Percentage
Dealer wins	499	49.9%
Player wins	419	41.9%
Draws	82	8.2%

This table shows the results of 10,000 simulated games. With a larger number of trials, the outcome percentages converge even closer to theoretical expectations: the dealer wins roughly 49.7%, the player 41.3%, and draws 9%. This confirms the fairness of the simulation, with the dealer maintaining a slight advantage due to acting last.

- Synthesis

Across all three simulations, the dealer consistently wins slightly more often than the player, reflecting the inherent structural advantage of acting last in Blackjack. The player wins approximately 41–42% of games, and draws occur around 8–9% of the time. As the number of simulated games increases, the outcome percentages stabilize and converge toward expected theoretical probabilities.

These results indicate that:

- The turn-taking logic, hand evaluation, and win conditions in the Blackjack class are implemented correctly.
- The simulation accurately reflects the real-world dynamics of Blackjack.
- The dealer's advantage is consistently captured, while the player still has a substantial chance of winning.

Thus, the game behaves realistically, and the simulation results are robust across small (100 games), medium (1000 games), and large (10000 games) sample sizes. However, if a player played 10 games, it's unlikely they would notice this small dealer advantage. With such a small sample size, the outcomes are highly variable: the player might win 4–6 games, the dealer might win 3–5, and there might be 0–1 draws.

Acknowledgements:

I would like to thank the course staff for providing the project template and guidance site. I consulted Professor Harper during the lab and the W3Schools Java Tutorial to check the documentation of java.util.Scanner class.

Extensions:

Extension 1:

What did you do and why?

For the extension 1, I implemented a new method in the Blackjack class called playerTurnInteractive(), which allows a human player to control their moves via terminal input. I also created a separate class, Interactive, that lets a user play a single hand of Blackjack against the dealer on the terminal. The goal of this extension was to explore how the game behaves when a real human makes decisions, rather than relying solely on the automated player strategy used in the simulation.

What was the outcome?

The extension successfully allows a human player to hit or stay at each turn. The program displays the current hands, total values, and whether the player or dealer wins. This interactive mode provides a realistic experience of playing Blackjack, demonstrating how player decisions affect outcomes differently than an automated strategy.

How can we replicate your outcome in your code-base?

Compile the Interactive.java class and run the program, optionally entering a custom reshuffle cutoff when prompted.

Extension 2:

What did you do and why?

For the extension, I updated the Hand class so that Aces can count as either 11 or 1, depending on which is more advantageous to the player, thereby preventing unnecessary busts. I also added a method Hand.isBlackjack() to detect when a hand is a natural two-card 21, and updated the game logic to handle Blackjacks before any additional drawing. The rules implemented are:

- Both player and dealer have Blackjack -> tie (push)
- Only player has Blackjack -> player wins
- Only dealer has Blackjack -> dealer wins

Then, I modified the game to optionally use 6 decks and allow the user to specify a reshuffle cutoff. In the simulation, the game reshuffles when fewer than one deck remains. I modified the deal() method to provide each participant with two initial cards in the standard sequence: player-dealer-player-dealer. These changes enable a more accurate simulation of real Blackjack play, allowing us to explore the impact of deck size and reshuffling on outcomes.

What was the outcome?

Table 4. Results of 10000 simulated Blackjack games after applying extension 2.

Outcome	Count	Percentage
Dealer wins	4959	49.59%
Player wins	4126	41.26%
Draws	915	9.15%

These outcomes are consistent with theoretical expectations: the dealer maintains a slight advantage due to acting last, while the player still has a substantial chance of winning. Compared to a single-deck simulation, the results are similar, though using multiple decks reduces the effect of card counting and stabilizes outcome percentages over many games. Overall, the simulation confirms that the extended game behaves fairly.

How can we replicate your outcome in your code-base?

The updated code with extension 2 is already in the ext folder. Run the Simulation class in the ext folder, and you can adjust the simulation time variable in the class to get a higher precision of the expectation.