Monte-Carlo Simulation of Blackjack

Muneeb Azfar Nafees

February 2025

Contents

1 Abstract

My project sets up a simplified version of Blackjack using object-oriented programming in Java. The primary goal is to simulate multiple rounds of Blackjack and evaluate statistical outcomes such as player win rates, dealer win rates, and the percentage of games that end in a draw.

To achieve this, I designed several classes, Card, Hand, Deck, and Blackjack, that model the key elements of the game. This approach uses object-oriented design, where each class has its own data (fields) and behaviors (methods). For example, the Hand class uses ArrayList to dynamically manage the cards held by the player or dealer. This is crucial because the number of cards in a hand changes during gameplay, so using Arraylists allows us to save space. In addition, I implemented a Simulation class to run thousands of game iterations, employing Monte Carlo simulation techniques.

My key findings indicate that the dealer has a slight edge over the player, with win rates settling approximately at 49% for the dealer and 42% for the player in more than 10,000 games. The observed draw rate of 8% remains consistent across larger simulations. As anticipated, with the growing number of simulations, changes in win rates decrease, showing statistical convergence (law of large numbers).

2 Results

2.1 Experiments Conducted

I ran three sets of simulations:

1. **100 games**
2. **1,000 games**
3. **10,000 games**

For each experiment, I recorded:

1. **Percentage of Player Wins**
2. **Percentage of Dealer Wins**
3. **Percentage of Draws**

In each game, the player kept taking cards until their total was at least 16. After that, the dealer took cards until their total was at least 17. If either one went over 21, the other won immediately. If both had the same total, the game ended in a draw.

2.2 Results of the First Three Games

To illustrate how the game worked, I recorded the results of the first three individual games played. These results provide insight into how the simulation functions at a micro level.

Game #	Player's Final Hand	Player Total (End)	Dealer's Final Hand	Dealer Total (End)	Outcome
1	[11, 2, 10]	23 (Busts)	[10, 4]	14 (Stands)	Dealer Wins
2	[2, 10, 2, 10]	24 (Busts)	[6, 8]	14 (Stands)	Dealer Wins
3	[3, 4, 2, 7]	16 (Stands)	[5, 10, 10]	25 (Busts)	Player Wins

Table 1: Results of the first three simulated Blackjack games

These examples illustrate:

1. **The significance of busts** - In the first two games, the player lost due to exceeding 21, showing how the decision to hit can lead to unfavorable outcomes.
2. **Dealer's strategy** - The dealer only plays if the player does not bust, and they hit until reaching at least 17.
3. **Dealer bust scenario** - In the third game, the player stood at 16, forcing the dealer to draw another card and ultimately bust.

2.3 Key Metrics and Observations from Simulations

Simulation Results			
Number of Games	Player Win %	Dealer Win %	Draw %
100	41%	53%	6%
1,000	42%	49%	8%
10,000	42%	48%	8%

Table 2: Percentages for player's win, dealer's win, and draws over different game counts

Analysis of Results

1. Dealer's Advantage
   - The dealer consistently wins more games than the player, aligning with real-world Blackjack statistics where house rules favor the dealer.

- The player has a lower win percentage (42%), which reflects the disadvantage of playing first since they can bust before the dealer even takes action.

2. Stabilization of Win Rates

   - The variation in percentages is more noticeable in 100-game simulations but becomes more stable as we move to 1,000 and 10,000 games. This aligns with the Law of Large Numbers, showing that as the number of trials increases, the results converge toward expected probabilities.

3. Impact of Draws

   - The draw percentage increases slightly (from 6% in 100 games to 8% in larger runs), suggesting that as more games are played, the probability of tied hands evens out.

4. Game Fairness

   - Based on the results, the game is clearly not fair for the player, just as it is not in real casinos. The player's win percentage (42%) is lower than the dealer's (49%), reinforcing the house edge.
   - Since the player always acts first, they have a higher chance of busting before the dealer even plays, putting them at a disadvantage.
   - If a player were to play only 10 games, they would likely experience noticeable fluctuations in results. Short-term streaks of wins or losses may mislead them into thinking the game is either easier or harder than it actually is (the win/draw percentages at 100 simulations differ from percentages at 1,000 or 10,000 simulations)
   - Given the low sample size of 10 games, randomness plays a bigger role. A player may have one or two lucky wins, but over time, the dealer's advantage becomes apparent as more games are played.
   - The stabilization seen in 10,000 simulations (where the dealer consistently wins more) would not be obvious to a casual player, making it harder for them to recognize the statistical disadvantage in a short session.

3 Extensions

3.1 Ace Rule and Blackjack Check

3.1.1 What Did I Do and Why?

I improved my Blackjack rules to handle Aces more realistically. By default, the Ace was worth 11, but in a real Blackjack game, an Ace can count as either 1 or 11, whichever benefits the player or dealer most. Specifically, after the player or dealer busts, I check if there is at least one Ace valued at 11 in their hand. If so, I convert one Ace from 11 to 1 and let them continue drawing. I also implemented a rule to detect a real Blackjack: if a hand has exactly 21 with two cards, it beats any other hand totaling 21 with more than two cards. This brings the simulation closer to actual casino rules.

3.1.2 What Was the Outcome?

After introducing this Ace rule and real Blackjack detection, I ran the same Monte Carlo simulations (100, 1,000, and 10,000 games). The results are shown below:

Simulation Results (After Ace Rule Implementation)			
Number of Games	Player Win %	Dealer Win %	Draw %
100	34%	56%	10%
1,000	40%	49%	10%
10,000	42%	48%	8%

Table 3: Percentages for player's win, dealer's win, and draws over different game counts

Compared to the original rules, these changes did not drastically change the overall win/draw percentages in the large-scale (10,000-game) simulation. While the ability to switch Aces from 11 to 1 helps avoid some busts, the frequency of this scenario is relatively small when playing thousands of games. Consequently, the dealer still retains a similar edge, and the player's win rate remains close to the original figure. However, in smaller runs (e.g., 100 or 1,000 games), slight fluctuations may appear due to the short-term luck affecting outcomes.

3.1.3 How to Replicate This Extension?

1. Immediately after playerTurn() and dealerTurn() methods return false for a bust, check if the hand contains an Ace worth 11. If so, change its value to 1 and repeat the turn process.

2. During the final outcome check, if either the player or dealer has a total of exactly 21 with exactly two cards, declare a "Blackjack" that beats a three or more card 21.

3. Compile updated code and run the same sets of simulations (100, 1,000, and 10,000 games) via Simulation.java.

3.2 Betting Strategy

3.2.1 What Did I Do and Why?

I introduced a betting system in the simulation to see if the player could still make a profit despite having a lower overall win rate than the dealer. Both the player and the dealer start with $200. In each game, each side bets half of their current pot. Whoever wins that round takes both halves, while the loser's pot is halved. This means that if the player (or dealer) goes on a winning streak, even if they lose more games overall, they can still end up with more money by winning higher-stakes bets later on.

3.2.2 What Was the Outcome?

Below are three sample simulations showing the final pot sizes after running 100, 1,000, and 10,000 games each time:

Sim	**Games**	**Player Win %**	**Dealer Win %**	**Draw %**	**Player Pot**	**Dealer Pot**
1	100	42	49	9	347.57	52.42
2	100	43	50	7	250.97	149.03
3	100	34	56	10	131.38	268.62
1	1000	41	50	8	96.38	303.62
2	1000	43	47	8	15.73	384.27
3	1000	40	52	7	127.69	272.31
1	10000	41	48	9	244.94	155.06
2	10000	41	49	8	62.74	337.26
3	10000	42	49	8	46.10	353.90

Table 4: Betting Strategy Simulation Results Across Three Simulations

These results indicate that the overall win percentages are similar to those in the base game (the dealer still has an advantage). However, the final pot sizes vary significantly from one simulation to another, demonstrating that timing and streaks are very important. In some instances, the player's pot increases even with a lower win rate. In conclusion, a betting strategy can certainly enable the player to make a profit, even if they lose more games overall.

3.2.3 How to Replicate This Extension?

1. In Simulation.java, initialize two variables (i.e., double playerPot = 200.0; double dealerPot = 200.0;).
2. For each game, based on the outcome, perform the calculations on both pots (i.e., take half of the loser's pot and add it to half of the loser's pot in the winner's pot).

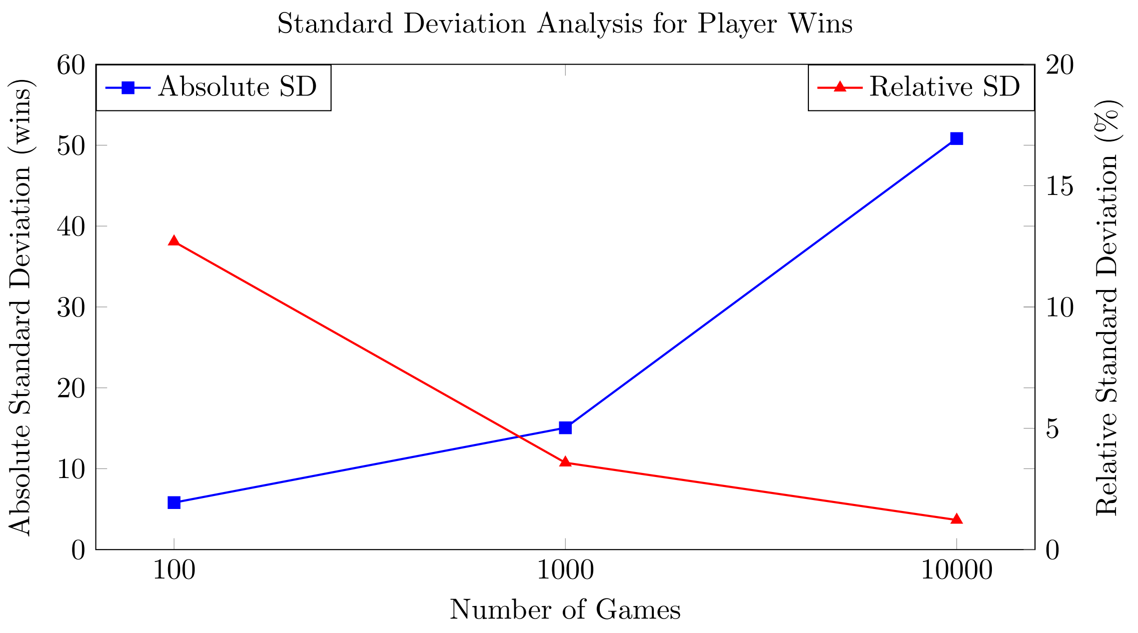
3. After simulating 100, 1,000, and 10,000 games, output both the final pot sizes and the win/loss/draw percentages to see how the strategy impacts final earnings.

3.3 Analysis of Simulation Variability Using Standard Deviation

3.3.1 What did I do and why?

To understand how consistent the simulation outcomes are as the number of games increases, I implemented a class StandardDeviation that runs 15 independent simulations for three different game counts: 100, 1,000, and 10,000 games of Blackjack. In each simulation, I recorded the number of games the player won. I then computed both the absolute standard deviation and the relative standard deviation of these win counts. The goal was to observe whether the standard deviation (when expressed as a percentage of the mean win count) decreases as more games are played, which would indicate that the simulation results become more stable with larger sample sizes.

3.3.2 What Was the Outcome?



While the absolute standard deviation increases as more games are played (which is expected because the number of wins increases), the relative standard deviation decreases significantly. This means that even though the raw count of wins has a larger spread in bigger simulations, the win rate (player wins as a proportion of total games) becomes much more consistent. This supports the Law of Large Numbers, showing that the simulation's win percentages stabilize with an increasing number of games.

3.3.3 How to Replicate This Extension?

1. Use the StandardDeviation class, which runs 15 simulations for each of the three game counts (100, 1,000, and 10,000 games).
2. For each simulation, record the number of wins for the player.
3. Implement the calculateStandardDeviation method to compute the mean and standard deviation from the recorded wins
4. Calculate the relative standard deviation using the formula:

$$\text{Relative Standard Deviation (\%)} = (\frac{\sigma}{\mu}) \times 100$$

5. Print both the absolute and relative standard deviations.

4 Acknowledgments

1. Law of Large Numbers - Wikipedia
2. PGFPlots Package - Overleaf
3. Relative Standard Deviation - Statistics How To