Blackjack Single-Player Strategy Simulation

James Sinclair, jsinclair32@gatech.edu, Group Project 4

Abstract:

The game of blackjack was "solved" in 1956 and published in the Journal of the American Statistical Association. However, for the average player, the optimum strategy outlined in the paper is tough to completely memorize, so I wanted to explore a basic, easy to remember strategy with a few variations in order to perform well in a trip to the casino. Through simulating hundreds (and even thousands) of games I have found that my oversimplified strategies will result in one most likely losing money in the long run. However, this leaves room for future experiments to try other simplified strategies, that may include splitting and doubling down, to give the casual player hope for some winnings.

Background:

Blackjack is one of the most widely played card games for gambling across the world. The object of the game is for the player to beat the dealer, either by having a hand totaling 21 or by having a hand with a sum higher than the dealer's hand. In 1956, Roger Baldwin et al. published a paper titled The Optimum Strategy in Blackjack in the Journal of the American Statistical Association. This was the first mathematically sound optimal blackjack strategy (Baldwin et al). While this is the optimal strategy for the game, the average player who would like to have fun and hopefully win some money could not be bothered to memorize and perfect this strategy.

Methods:

This paper will lay out an oversimplified strategy, based on some of the key components of the optimal strategy, and simulate games of blackjack to see if it proves effective in serving as a go to strategy for the casual player. I also wanted to test three additional rules to this base strategy, also based upon significant rules laid out in the Baldwin et al. paper, in case they may add any significant statistical changes.

Upon some research on the most widely accepted "House" or dealer rules, I found these 4 rules that guided the dealer's strategy in the simulation:

1. If the total hand value is 16 or less, the dealer must hit
2. If the total hand value is 18 or more, the dealer must stand
3. If the total hand value is 17, and it is a hard 17 (no Ace in hand), then dealer must stand
4. If If the total hand value is 17, and it is a soft 17 (Ace in hand), then dealer must hit

Additionally, I saw through my research that some strategies are based off of the dealer's card that is played face up, so I will include the fact that one of the cards is visible to the player upon the initial dealing out of the cards.

The basic strategy I have chosen based off the "optimal strategy" was as follows:

1. Stand when your hand is sum 12 - 16 and the dealer's upcard is 2 - 6
2. Hit when your hand is sum 12 - 16 and the dealer's upcard is 7 - Ace
3. Stand when your hand is soft 18 (Ace in hand) and up
4. Stand when your hand is hard 17 and up

The 3 additional rules to build upon the control strategy to see if any make a statistical difference in the long term outcome were as follows:

1. Always hit a hard 12 against a dealer's 2 or 3 upcard
2. Always hit soft 18 (Ace, 7) when the dealer's upcard is 9, 10, or Ace
3. Always stand on a hard multi-card 16 if dealer's upcard is 10

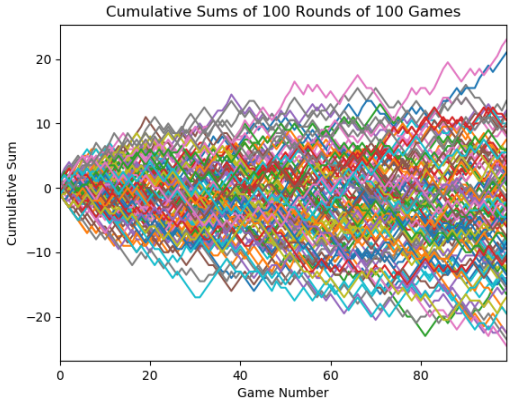
For simulating all of this, I used Python, specifically the PyDealer package was a great deal of help as it was developed specifically for use in simulating decks of standard playing cards. It allows you to create "Deck" and "Stack" (hand) classes that deal with "Cards" having their own name, value, suit, and abbreviation. The code I have written begins with defining functions that I use to convert each card to a numerical value and then calculate any hand's total sum with any number of cards. These functions also have special ways of handling Aces since they can be treated as numerical values 1 or 11. It picks the more advantageous way of counting the Ace. The next portion of the code handles the rules that guide how the dealer will play in each simulated game. These are the rules I have laid out above. Similarly, I have defined a function that dictates the decisions the player will make according to the base strategy I previously explained. I have included a python script for each additional strategy as well where I have modified this player strategy to include the three additional rules I mentioned.

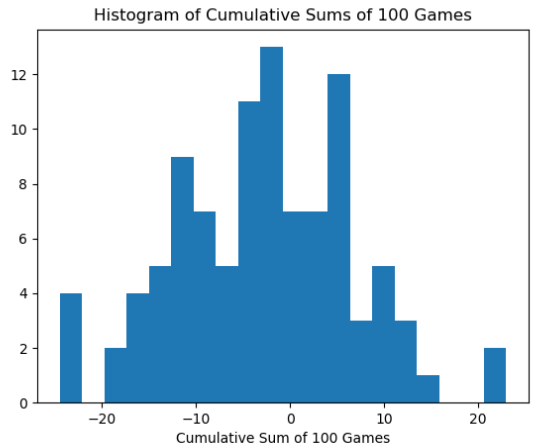
For simulating the gameplay, I have split it up between two functions. In the first function, I deal out the hands to both the player and the dealer, with one of the dealer's cards "face-up" so that the player can strategize off it. Then, the function handles the player's turn. The code assumes the player places a bet of $1. If the player wins by beating the dealer through multiple turns, the player is paid out $1, plus their initial bet. However, the code first checks to see if the player was dealt a "blackjack" or an automatic 21, which automatically pays out the player $1.5 in addition to their original $1. If the dealer also has "blackjack", it is a tie and no additional pay is made, so the player simply gets their bet back, which is represented as $0. Next, the code checks to see whether the player will stand, in which case it will proceed to the finish the game. Otherwise, the function will "hit" the player by dealing them another card, reassessing what they will do next, until they either stand or "bust", go over 21 and automatically lose.

If the player decides to stand, the code will then run the next function which completes the game by running the course of the dealer's hand. This function will first check if the dealer's strategy dictates they stand, and it will determine which hand sum is higher, player or dealer, and determine the outcome of the game. If the dealer does not stand, the function will "hit" the dealer by dealing them cards and then reassessing their next move once the new hand sum is determined. This goes on until the dealer decides to stand or they bust.

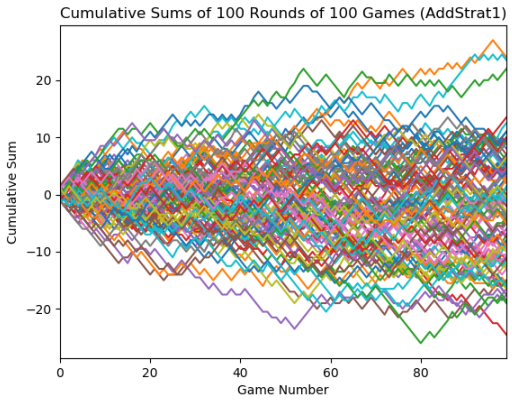
Results:

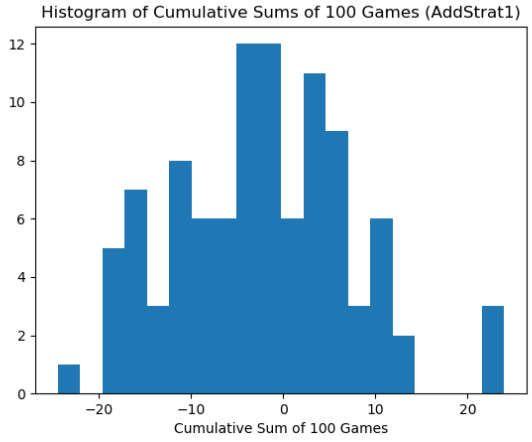
In order to test the strategies, I wrote code to execute 100 rounds of 100 games for each strategy. I then converted the data of all these outcomes and created histograms and line graphs for them, pictured below. I first began with the base strategy outcomes:

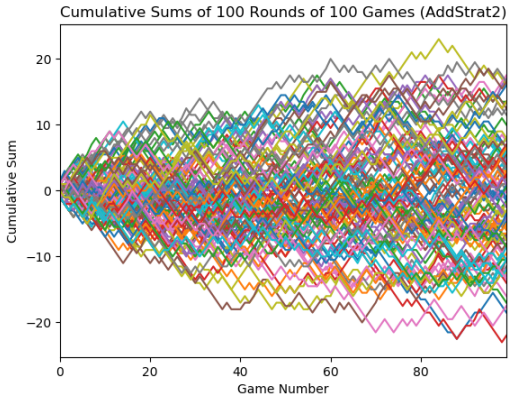


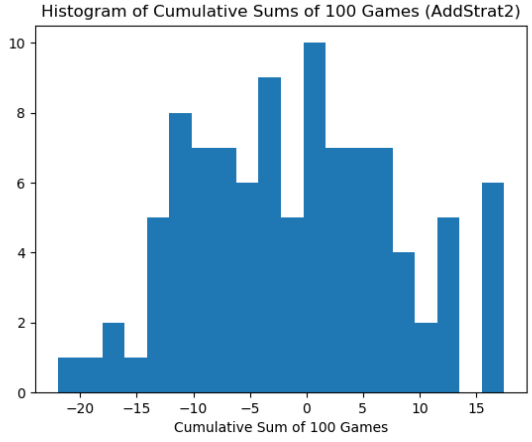


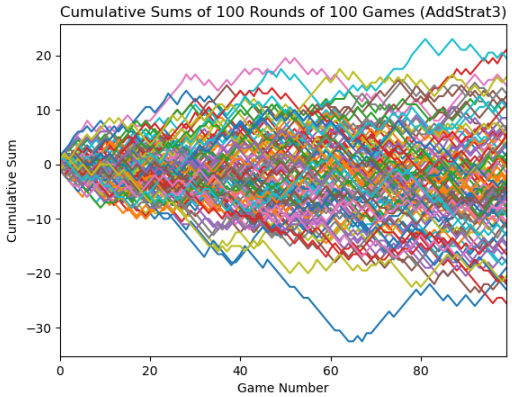
For the line graph, I have depicted the cumulative sum of only winnings (or losings) at the end of each game, beginning with game 1. As one can see, the outcomes of 100 games ranged from losing about $25 overall to gaining about $25 overall. However, as evidenced from the histogram of the final cumulative sums of each round of 100 games, one can see that the distribution is skewed left which means one is more likely to lose money at the end of 100 games using the base strategy. This is further supported by the mean which was -$2.96, with a standard deviation of $9.45.
I ran the same process for the three additional strategies:

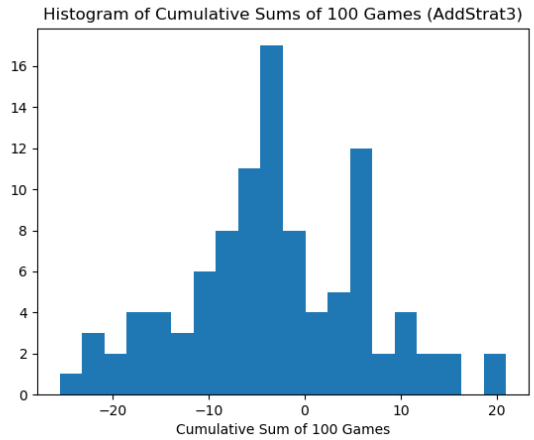












Starting with the first additional strategy or AddStrat1, it appears to have more or less performed the same as the base strategy. This is further supported by its mean which was -$2.21 with a standard deviation of $9.57. AddStrat2 is interesting in that it never reached a final cumulative sum as high as the others top range, but just visually looking at the histogram it appears to be skewed slightly more positive than the others. This is confirmed by a mean of -$1.03, standard deviation of $8.99. AddStrat3 is interesting in that it seemed to have a higher peak in the middle of the histogram than the others. Additionally, one game at one point experienced a cumulative sum more negative than any other game I had simulated. However, at the end of the day the mean was -$3.11 with a standard deviation of $9.54.

Running Scripts For Oneself:

For one to run the simulations and visualizations themselves, each python script corresponding to each of the four strategies can be opened in an IDE such as VSCode and run to produce the output visualizations. Each script is currently set to run 100 rounds of 100 games, but anyone can open the script and change the numbers to anything they desire. There are comments in the code indicating where someone can make these changes in order to simulate more or less games and/or rounds.

Conclusions:

Based upon my findings, I would not recommend using any of these strategies for the casual blackjack player. While the losses on average at least remained minimal, the fact remains that more often than not, one will lose money playing these strategies over long periods of time. However, I remain hopeful that there still may be a simple strategy out there that will at least average slight winnings or even breaking even. Were I to continue my exploration of strategies, I would perhaps try implementing strategies including “splitting”, where a player may choose to split their initial hand and play two hands, increasing their odds of beating the dealer. Or perhaps implementing strategies that include “doubling down” which means doubling the bet when you have a hand that is statistically more likely to beat the dealer.

References:

Baldwin, Roger R., et al. “The Optimum Strategy in Blackjack.” Journal of the American Statistical Association, vol. 51, no. 275, 1956, pp. 429–39. JSTOR, https://doi.org/10.2307/2281431. Accessed 4 Dec. 2022.

PyDealer Package:
https://pydealer.readthedocs.io/en/latest/license.html