Analyzing Tactics/Methods to play Black Jack to Determine Their Effectiveness

Abdulaziz Al-Said, Kalvin Xiong, Weston Hawes, Anthony Lopez
California State University, Fresno
April 14, 2023

Abstract

The purpose of this report is to introduce and analyze the card game black jack and tactics/methods to play it in order to determine their effectiveness. The method is through a Monte Carlo simulation in the scripting language Python. This report will breakdown the game's rules, different playing strategies, and the win/loss data collected. All data present in this report was directly collected through this simulation.

1 Introduction

Probability has been an important factor in many card games, including Black Jack.. There can be many tactics and methods utilized in playing a game like Black Jack to determine the best course of action, in order to calculate the effectiveness of these tactics and methods the monte carlo method was utilized to simulate a Black Jack game in Python. This simulation's purpose is to have the player use different tactics of playing in order to beat the dealer's static playstyle. The win ratio from these different tactics are then stored and we iterate many times in order to find the average win ratios for each playing policy. There is also an infinite deck, meaning cards are "replaced" after drawing, and a standard non-replacing deck.

2 Background Material

2.1 Game Rules

The goal of Black Jack is to beat the dealer by getting a count as close to 21 as possible, without going over 21. The value of an ace is up to each individual player, if it's worth 1 or 11. Face cards are 10 and any other card is its pip value

2.2 Simulation Model

2.3

This simulation model can be defined as a Monte Carlo and agent-based model with a primary discrete variable. The simulation is Monte Carlo since it has the stochastic element of random cards drawn and we are continuously repeating the simulation a certain amount of iterations to determine what the effectiveness of each playstyle is based on win/loss percentage. That win/loss value is the discrete variable of the simulation. The different play styles constitute the agent-based characteristic of this specific Black Jack simulation as well.

2.4 Player Playing Policy (The PPP)

The different playstyles of our agent are referred to as policies and are as follows. Policy 1 has the player stick if their hand totals 17 or more, and hit if less. Policy 2 has the player stick if their hand totals 17 or more and is hard, meaning they do not have an ace in hand, else hit unless their hand is equal to 21. Policy 3 was to always stick no matter the hand. Policy 4 was to always hit once no matter the hand. Policy 5 has the player check their opening hand for an ace, or soft, as well as having a total of 10 or less; when this is true we convert the ace to 11 points instead of its default 1 and stick. Else we will hit once and stick. These policies allow the player agent to function and behave. The dealer has the static policy of always hitting once if the hand is less than 21.

3 Approach

The code approach taken was to create a python script that will simulate all 5 playing policies and both deck types across 10, 100, 1000, and 10000 iterations. It starts with creating an array to represent the 52 card deck by taking the values [A,2,3,4,5,6,7,8,9,10,10,10,10] and multiplying the array by 4. We keep an ace as ‘A’ in order to classify it for the hard and soft rules, the Jack, Queen, and King are worth 10 points and that's the only necessary data. We multiply by 4 to compensate for all the suits in a deck, we don’t track suits since it is not pertinent to the scoring for the game. Once we have our 54 card deck array we will deal two cards to the player and two to the dealer by generating a random index and copying that index value to a card object, for the standard deck we will then delete it from the array. We then have the player's turn simulated by the agent with its policy, then the dealer plays using its universal policy. After it compares the dealer to the player's hand and determines if the player won or lost, ties count as losses for the purposes of these simulations. It then increments the win counter and creates graphs using the matlab plotting library, or matplotlib.

4 Results

Figure 1 is the data graph for the standard deck while Figure 2 represents the data for the infinite deck. Overall, it is apparent that the win ratios for the standard deck are higher than the infinite deck for Policies 1-3, Policies 4 and 5 both have consistent results across all iterations for the infinite deck. The only exception to these statements is policy 5 during the 10 iterations for the infinite deck. Overall, Policy 1 and 5 hold the best win ratio and Policy 5 is the most consistent among all the policies. Policy 4 is the absolute worst out of all the policies and logically so, always hitting even if you have 21 is a ridiculous strategy, but the data gathered

proves the overall strength of the simulation because it behaved as intended. The variation range displayed is wider on the lower amount of iterations and thinner the higher the iteration count. This is expected since the larger the sample size, the smaller the impact each loss has on the variation per policy. Out of all the policies, Policy 5 has the most consistent range of variation, further confirming how efficient the utilization of the ace mechanic is.

Figure 1:

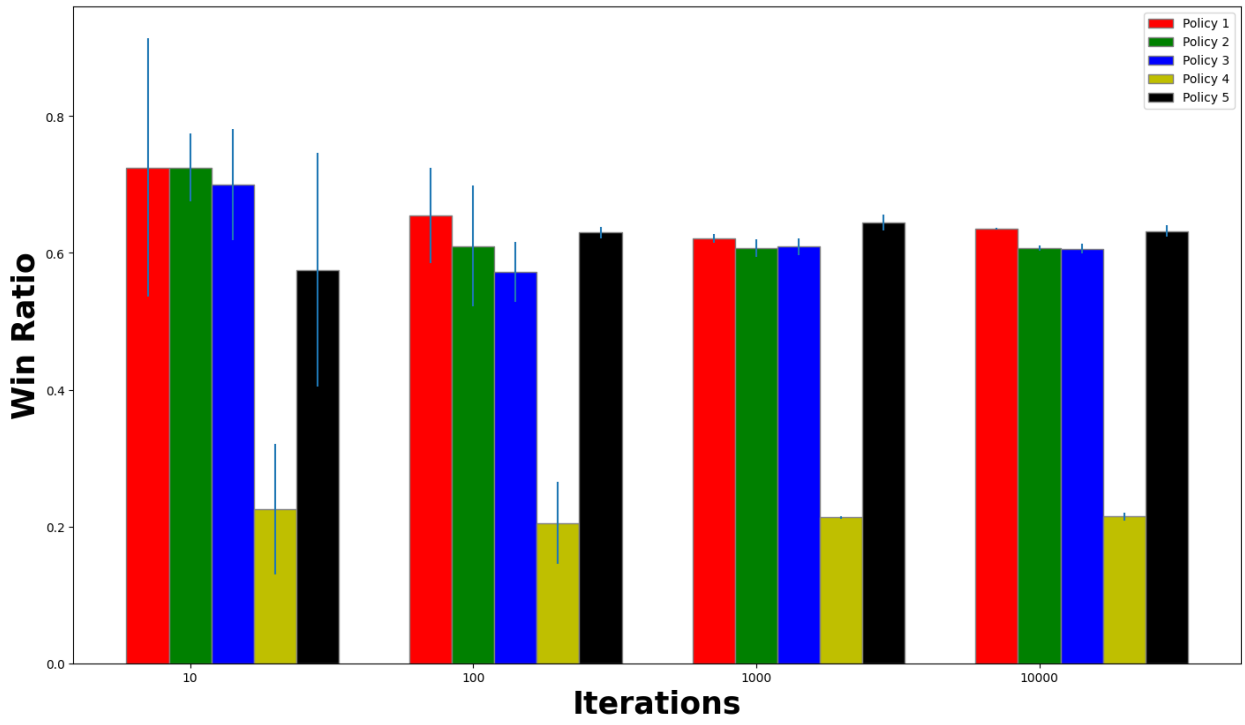
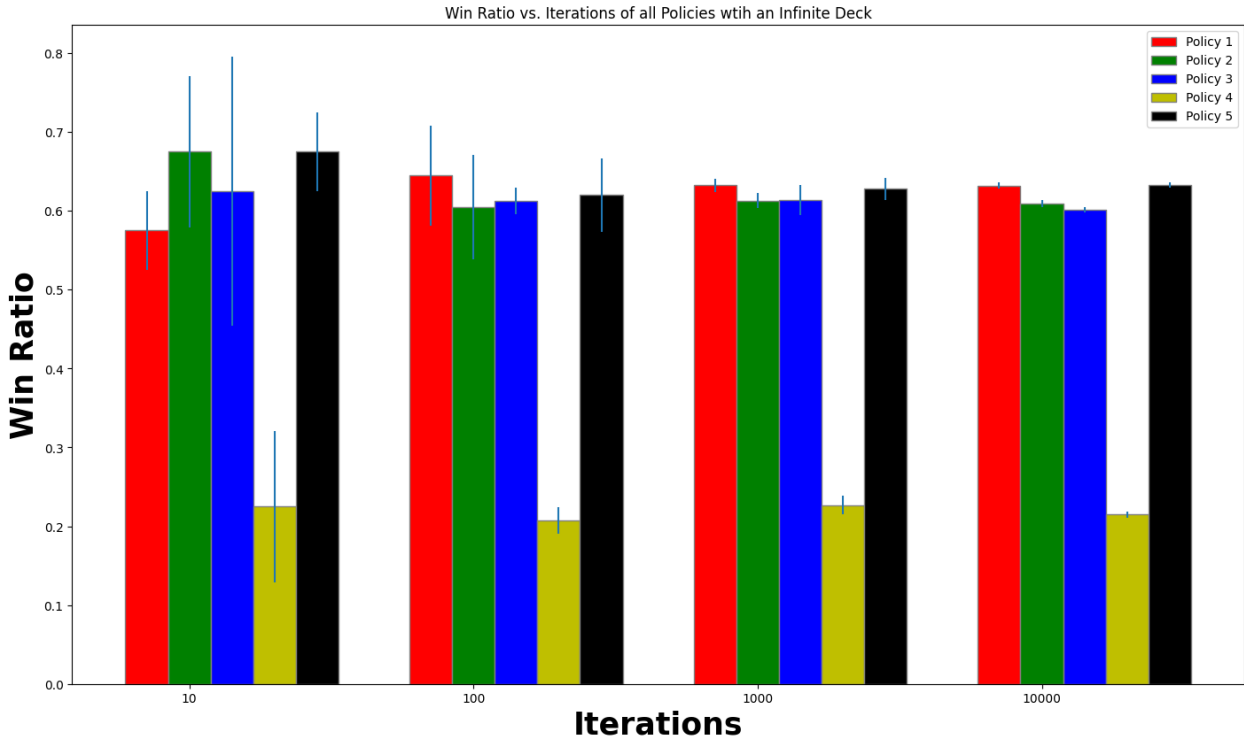


Figure 2:



5 Conclusion

In Conclusion, the playing policy numbers 1 & 5 performed the best and had the best results on average. Policy 4 was a terrible policy and displays the strength of the simulation for being so consistently bad, as should be expected. Overall, this data supports the effectiveness of logic and playing safe/smart rather than taking unnecessary risks.

6 References

N/A

7 Reflections

From the presentation we included a ‘whisker’ into our graphs to represent variation and it allowed for a better analysis of the data that supported the initial conclusions that were drawn about which policies were better and why.