The project that I did was on the Monte Carlo Simulation on a well known game, Black Jack, The motivation behind choosing this project is to see how the win probability changes when it given two different scenarios, a single deck and infinite deck. I applied three of the given conditions, while also giving two of my own conditions when it came to playing the game. I applied the five different conditions to each of the scenarios. I did the simulation 10,000 times for each policy and using the same scenario. All conditions were only applied to the player and not the dealer, since they normally have a condition set before the game, which mostly all casinos follow. The single deck and infinite deck had impressive results, the approach of the simulation consisted of for loops while using arrays in order to hold the deck which could have brought some wrong results, and since it brought questionable results I had to change the code in order to call each policy when a tied occurred.

During the first scenario, which was the single deck scenario, the first three given conditions offered good results. The first condition had the player “hit”, which means draw, if their hand was not equal or greater than 17, if they did they had to “stick”. This type of condition is one that a dealer follows as well, as mentioned before. Since there was five policies tested, this one came in second. This policy had a win probability of 57.29 percent, which was 5729 games won out of 10,000 games. The second policy had the player “stick”, which meant to stop drawing, If their hand was equal or greater than seventeen while their hand was “hard”. A hard hand is a hand that contains no Ace or does but it is valued at one because the player eventually went over 21 while having an Ace or drawing an Ace. This policy had a win probability of 56.20 percent. The third policy, which is the worst out of the original three that were given, had the player always stick after drawing a card that was not from the initial phase. This had a 46.31 percent win probability. Policy 4 had the player stick once his had was equal to 12 or greater, and policy 5 hand had the player stick once reaching 20 or greater. Policy had a surprising 62.54 win probability while policy 5 had the worst, which was 35.50 percent. The Infinite deck scenario had a few slight changes, such as policy 1 being the best, policy 2 being the second best, and the fourth policy being third. These results were too good, so I had there could have been an error with my approach.

In my approach, I created an array holding the 53 cards for the single deck and for the infinite deck I had made not to decrease the deck unlike the single deck. In my infinite deck, in order to ensure all cards had the same probability of being drawn, I had made sure to shuffle the deck after every card was dealt during and after the initial phase. It seems the issue came from the way I was handling the wins, losses, and ties. Wins and losses were pretty straightforward, if the player won, it would return 1, since the policy function is a int function. The losses simply returned to a zero since we are primarily focusing on the player and the current game, so returning a zero meant that it was a loss. Unfortunately, the way I approached ties was affecting my results since it would have it return a one, so it would be a considered a win. After reading a bit about casinos, seeing videos, and even going to the casino myself and playing blackjack, I realized that when ties occur, a push occurs. A push, in black jack, is when a tie occurs, so another new game is played, if the player wants to play again otherwise it won’t continue. So, the way I could fix was to either check the returned value from the policy functions in the main, or I could check for the tie and when it occurs, calls the policy function from the policy function. In order to not have it only wins and losses, I called the function if a tie occurred, until a win or loss occurred. This changed my results significantly. Policy 4 was still the best policy in the single deck scenario, but only decreased in the win probability by .20 percent. This could be due to an error in my code. All other policies decreased, except policy 3, and by a good amount. Policy 3 didn’t change because it only had the player draw one card after the initial phase in both scenarios.

When it came to my presentation, I could have added a few more images than what is right now. It was difficult to consider what to add since this Black Jack project that was proposed to the class was the most popular among the students, so there wasn’t much new information or slides that I could have done. I did add a graph in order to show my results compared to the single and infinite deck when it came to the five policies. The only issue I had throughout my project was policy 4, the one I made regarding if the player hand was 11 or greater we would stick. This win probability, in the single deck, was high, around 62%. I know this is wrong, but I could not find what was causing the issue in the policy, but the infinite deck did change. Something I could have done during the presentation is mentioned that I did try one of the policies, but it did not occur to me at the time.

In the Monte Carlo Black Jack simulation, most results at first were impressive with the 2 scenarios, single deck, and infinite deck, and 5 policies that I implemented. Although the game had 10,000 loops for each scenario, there were questionable results due to the ties in the game. Although my approach had dealt with arrays and for loops, when it came to how I handled the ties, it caused issues. Resolving this was to call the policy when it came to a tie with the dealer and player.