CSE 13S Assignment 3: I Have a Little Dreidel
Jaren Kawai - jkawai

play-dreidel.c

When implementing play-dreidel.c which would allow the user to input custom game settings, I used getopt() that would handle the data given an call play_game from dreidel.c accordingly. At the beginning of the file, an array is created that stores all the names of the players that could be playing the dreidel game. This array is to be referenced back to once play_game returns an integer that would represent the index of the name of the winner. Assuming that the user inputs arguments for all three parameters (players, coins, and seed) getopt() will check if each value is within the minimum requirements to play the game, hence the use of atoi to convert optarg into an integer that can be used to determine if valid choices were passed in. If the user's choices are valid, variables store the values which are ultimately used to call play_game. In addition, the third argument, seed, is used to call mtrand_seed() to set the seed for the random number generator used to spin the dreidel. If the user's choices were not valid, main returns a non-zero integer and exits the program. Finally, the optional argument, -v is used to determine if information regarding eliminated players is to be displayed.

dreidel.c

This is the main c function that contains the two functions necessary to play the game of dreidel. There was a similar array to the one in play-dreidel.c created here for reference. Global variable for display is used here to determine if eliminated players feed is to be displayed.

spin_dreidel

Spin_dreidel takes no arguments as mtrand.c already has the information needed to produce random numbers according to the value of the seed. With that in mind, mtrand.c is used to create one random number which is then modded by four to get a value between zero and three inclusive that will determine what side the dreidel stopped on. Switch cases were used to take action depending on the result from the random number. Given an array created at the beginning of the function that had each possible letter that can be returned, the switch cases referred to the array in order to return the appropriate character.

play_game

Play_game is the main function that simulates playing a single game of dreidel using either the default settings, or the settings provided by the user. The function begins by creating two arrays, one that keeps track of the amount of coins that each player has, and another that tracks whether or not a player has been eliminated from the game. Variables were also created to track the winner, pot amount, players remaining, the return value from spin_dreidel, and the amount subtracted from the pot should a player roll and H. Variable for the amount of players left

is set to the amount of players passed via the parameter that function accepts, and a variable for rounds is set to 1.

Before the game is actually simulated, the array for coins is filled with the amount of starting coins based on the parameter passed to the function, and the array that keeps track of player standing is filled with ones to indicate that they are all still in the game.

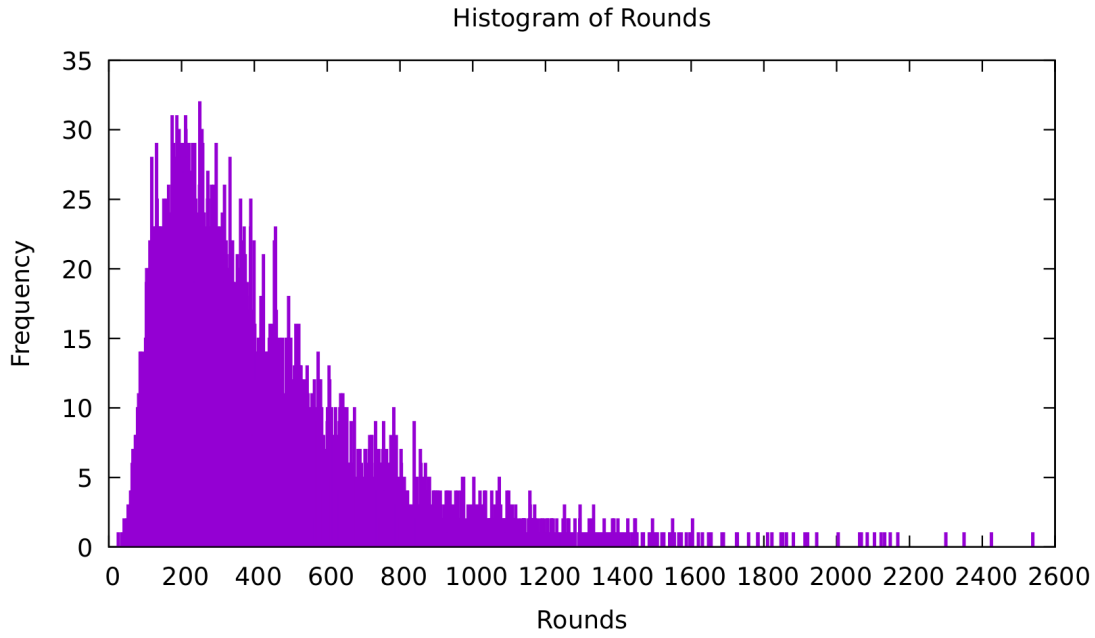
Then the game is carried out. The use of a while loop to play the game would continue as long as that amount of player left was greater than one. Within the while loop, another for loop was created to represent people taking turns. Within the for loop, an if statement determines whether or not a player is actually able to spin the dreidel as they must be in "good standing" to spin the dreidel. If a player is allowed to spin the dreidel, spin_dreidel() is called, and switch cases are created to handle each possible output of spin_dreidel().

The switch cases were broken down as follows. If a player were to roll a G, the amount of the pot would be added to that player's coins in the array and the value of pot would be reset to zero. If a player were to roll an H the subtraction variable would be equal to the pot divided by two. The value of sub would then be added to the players coins in the array and sub's value would be subtracted from pot. If a player were to roll an N, nothing would happen. Lastly, if a player were to roll an S, the amount of coins the current player has would first be checked, as if they have zero coins upon rolling an S, they are eliminated from the game. If the variable for display is set to one, the information about the eliminated player is displayed, and the standing of that character in the array is set to zero. The amount of players left is also decremented by one. If the number of players is equal to one, the loop breaks, and a for loop checks for the one player that still has a standing of one in the standing array. However, if a player does have coins upon rolling S on the dreidel, their coins are decremented by one, and one is added to the amount in the pot.

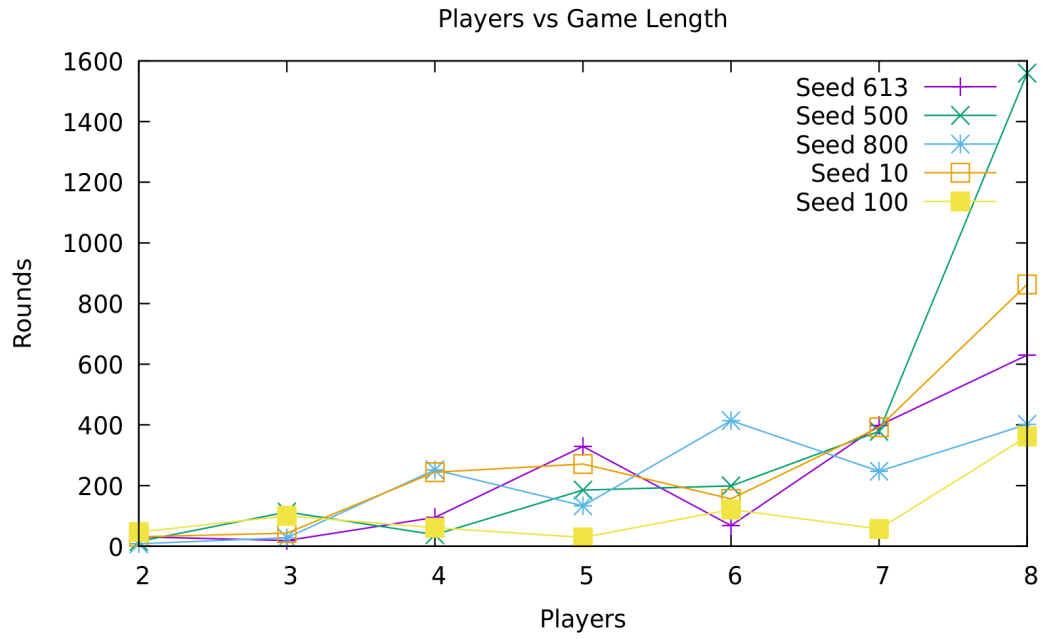
Once the for loop makes it through all of the current players, rounds are incremented by one and the process begins again until a winner is found.

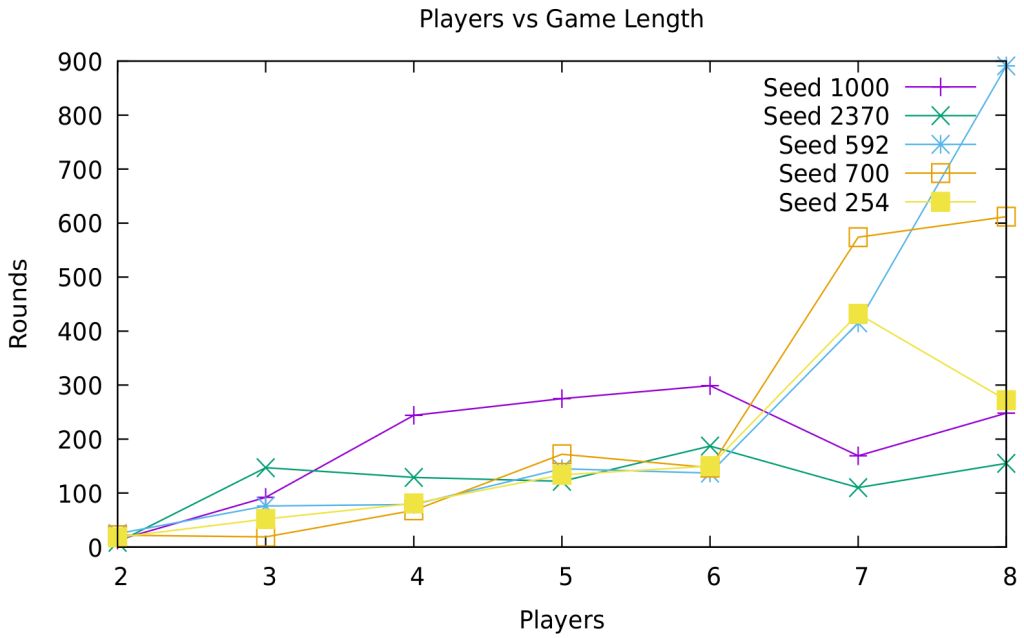
Further Analysis

When testing the play_game function through a various amount of seeds, player amounts, and coin amounts, more can be said about the game rather than who won and how long it took them to win. When six players and four coins were tested with 10000 different seeds, the average game length was around 401 rounds. Although the graph may not necessarily prove that the average is 401 from purely looking at it, it offers reason as to why the average is what it is. Although it seems that most of the values are around 200-300 rounds, there are many values that go outside of that range with many values in the 400-600 range. There are also values over 1000 that can have an effect on the average. The shortest game created within the bounds of the test was 26 rounds and the largest was 2539 rounds which means that there is a lot of variation in terms of outcome by only changing the sequence of what is rolled and when.



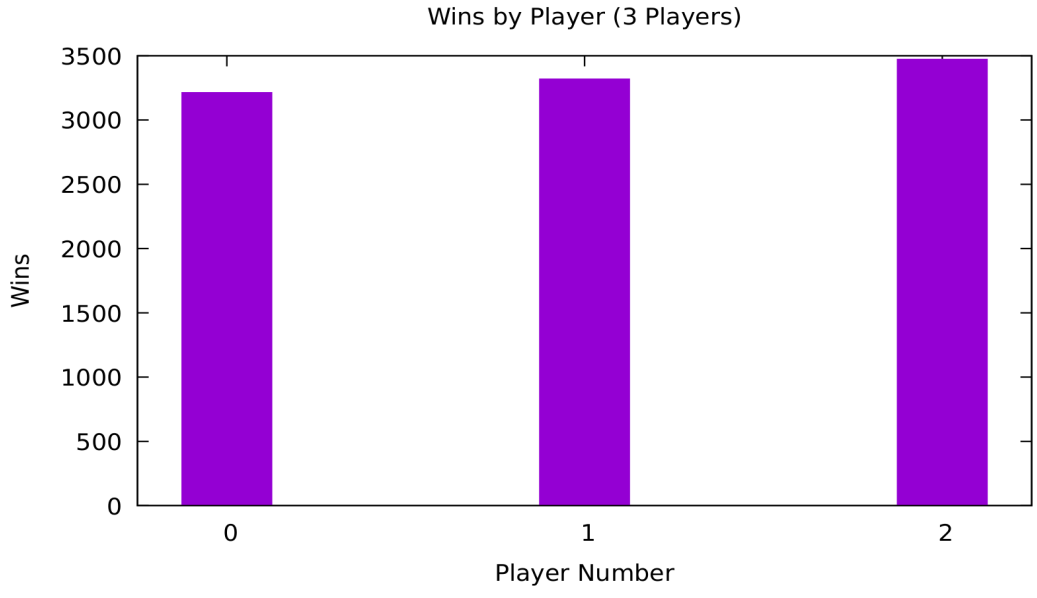
Having explored game length based on seed, question is raised as to whether or not the amount of players has an effect on game length. To test this, different seeds were used to determine if the amount of players in a game has a positive or negative effect on the amount of rounds the game lasts. From the two graphs below, 10 different seeds were used. Each seed was run with each of the valid player counts, and the amount of rounds were recorded. As shown from the graphs, from two players to eight players, games were almost always longer, however, when comparing players counts that are closer together such as three players to five players, the difference is not as significant, and game lengths vary because of the "randomness" of the dreidel that causes different event to happen that are unique to each seed.



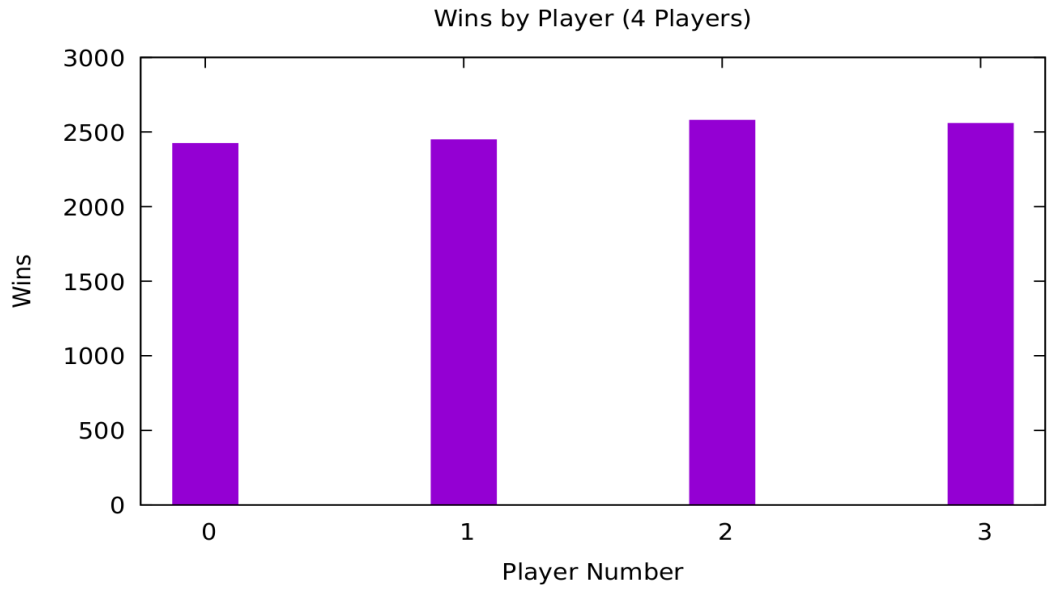


Ending, the last question that could be asked is whether or not your position in the game gives you an advantage or disadvantage. To gain data for this, the dreidel game was simulated with fixed player counts with the same amount of coins from before. Player counts started at three for this test because data with two players would be two close to an even split and it can be hard to tell if going first or second would truly matter. With the graphs, the x axis represents the number of the player zero through seven, and the y axis represents how many games that player won out of 10,000.

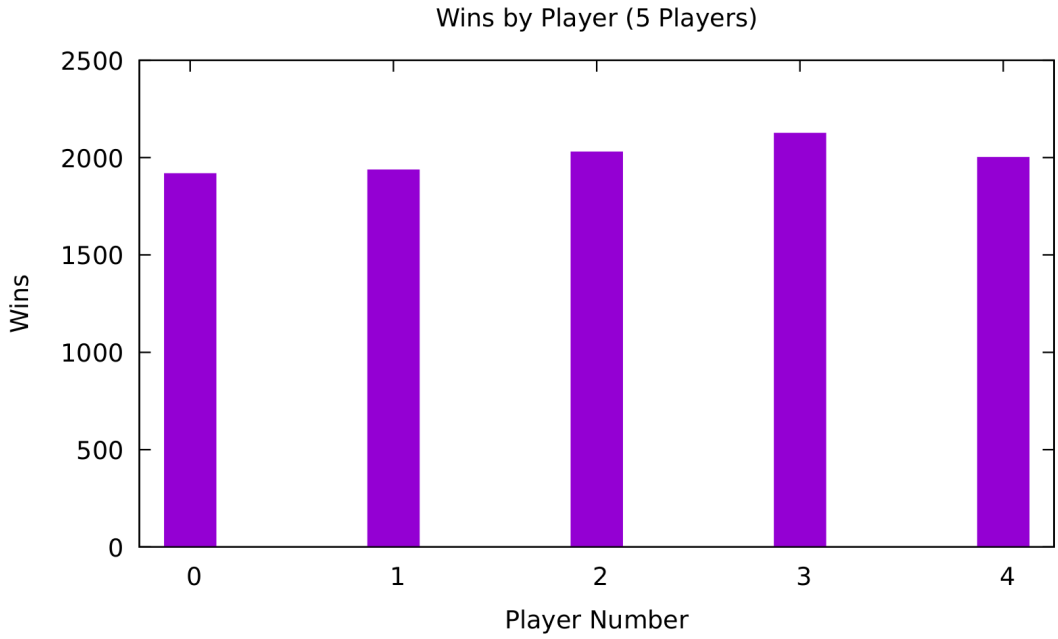
With three players, being first or second seemed reasonably close, and being third resulted in the most wins, however, it is not a clear advantage to going last in a three player game. It does seem to be more likely that you would win if you were to go third, but you are not at a clear disadvantage by going first or second.



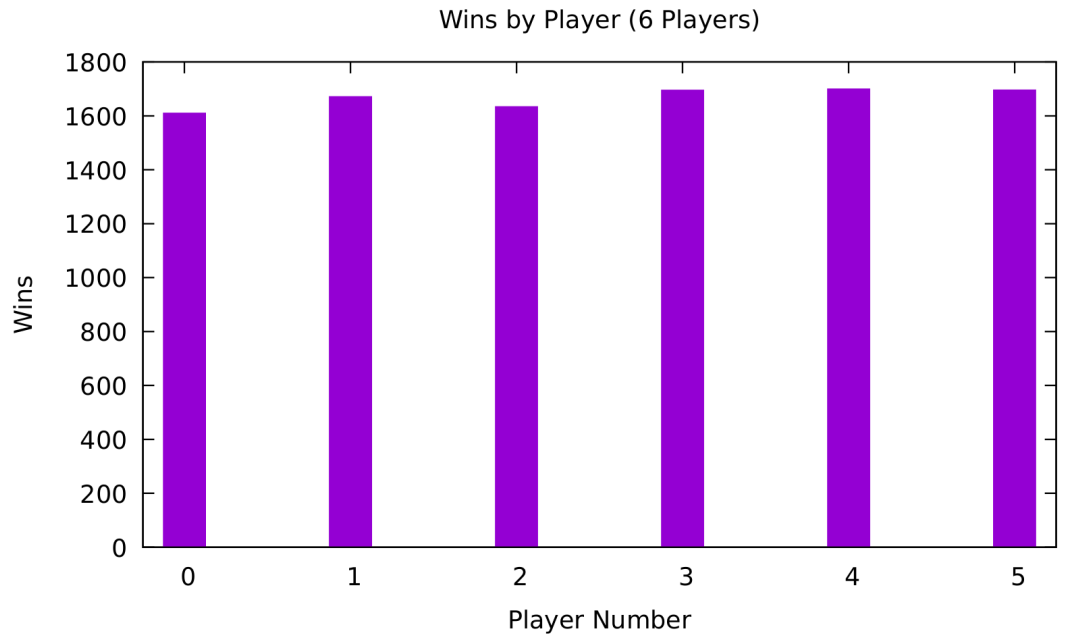
In a four player setting, almost all of the results were similar with almost all players getting similar amounts of wins. However, it appears that going third netting you the most wins in a 10,000 game test. Once again, there is no clear disadvantage or advantage to going in a certain position, but you are more likely to win if you go third according to this rest.



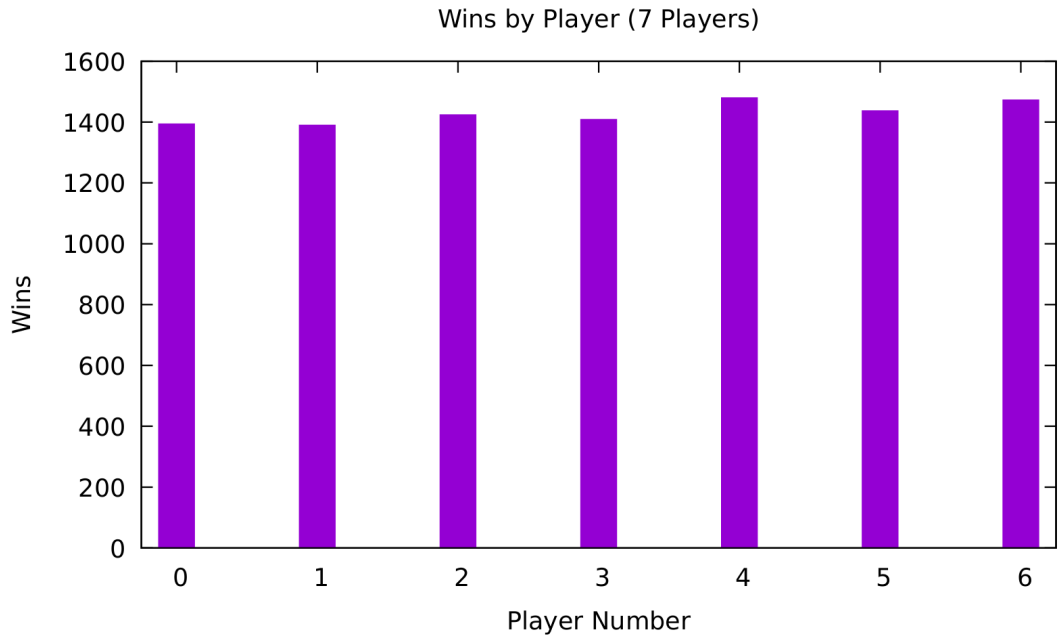
In a five player setting, the results were reasonably similar to before. However, going third or fourth seemed to slightly peak above the other players in this test. Also as before, there are no clear advantages or disadvantages to position here.



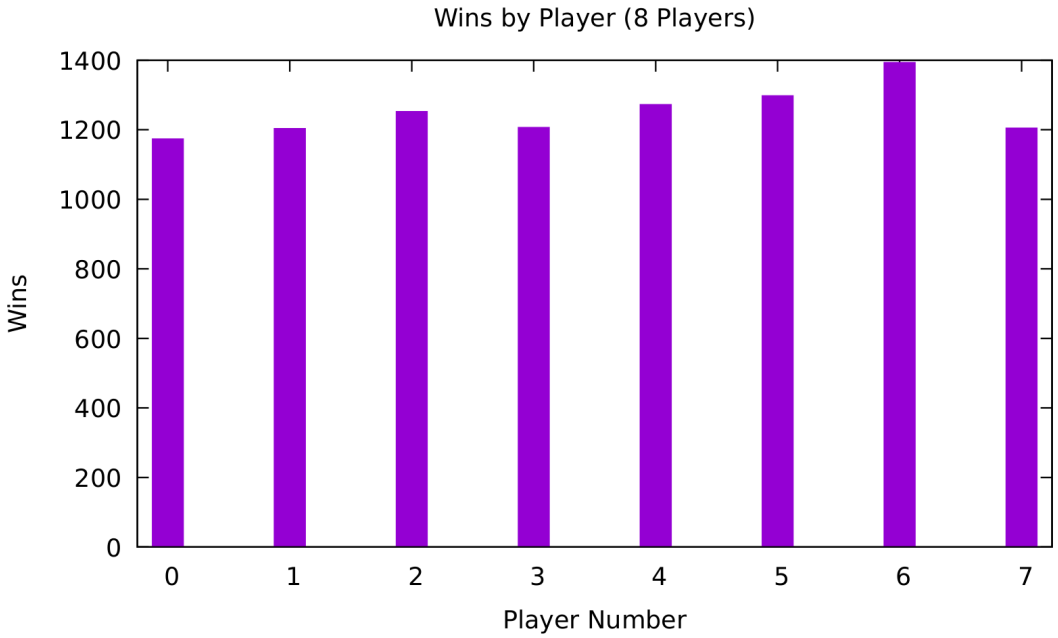
In a six player getting, results were close together, with going second, fourth, fifth, and sixth visually peaking above going first or third. Hence, going in these positions would be beneficial, but wouldn't result in significantly more wins compared to going in the less favorable positions.

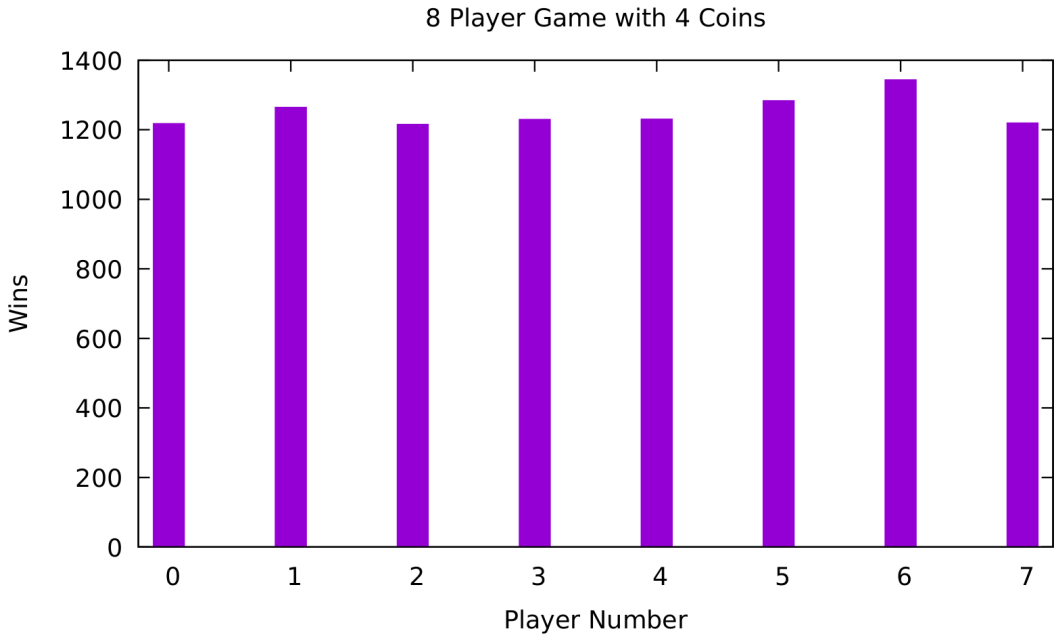


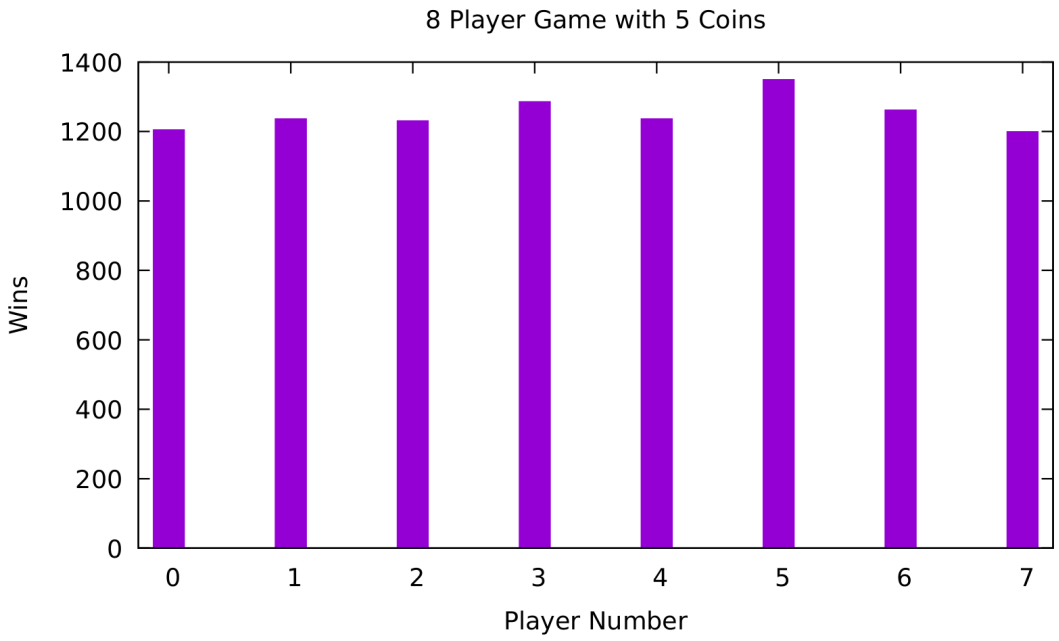
In a seven player setting, going fifth or last seemed to prove slightly more beneficial getting more wins than other positions. However, going in any of the other positions gives you a decent chance to win even though that likelihood isn't as high.



In an eight player setting, going in the back half of the players was visibly netting more wins than the other positions. However, in the big picture of position vs wins, there wasn't a clear winner as to what position is best. However, being last did net the least amount of wins amongst all the tests with eight players, and might serve as a slight disadvantage.







Overall, there were no clear advantages or disadvantages with regards to position in different player settings. However, the graphs did show that if you went later in the game (back half of the players) you were more likely to win compared to if you were to spin earlier in the game. As stated before, each game has events that drastically shape the outcome, and oftentimes you are at an advantage or disadvantage at different points in the game depending on its state. For example, you may be at an advantage in round 10, but are in a very unfavorable position in round 20 after a few spins put you on the edge of elimination.