# ISYE6644 – Team191 Final Project
# Distribution of Turns to Complete a Coin and Dice Game

**Alexandra Keamy and Aaron Adams**

## Abstract

Simulation is a useful tool to understand how a system that relies on random variable input works. In this report, we will analyze a game (detailed in the introduction) involving two players rolling a die in turns, adding or removing coins from a central pot, based on their roll. The key factors in this game are the die used, the starting number of coins each player has, the starting number of coins in the pot, and the magnitude of the penalty for a failed roll. A Python program was developed to simulate the game and using Monte Carlo simulation with 1,000,000 runs we were able to produce a distribution that appears geometric, for the number of turns it takes to complete the game, with the most common outcome being 6 turns, and a median of 14 turns. Using the initial parameters, no player seems to have a distinct advantage in going first or second. However, altering the starting parameters changes this distribution and we explore the effects of a few permutations of starting parameters.

## Introduction

This project is centered around a game involving two players, a die, and some coins. Players start with 4 coins each and a central pot is initialized with 2 coins. Players take turns rolling a six-sided die, and based on their roll perform one of four actions:

- If a player rolls a 1, no action is taken.
- If a player rolls a 2, they take all coins from the pot.
- If a player rolls a 3, they take half of the coins from the pot rounded down.
- If a player rolls a 4, 5, or 6 they pay a penalty and place 1 coin into the pot.

The game is conducted in turns, where both players roll a die. Player A always goes first, and Player B always goes second. The game ends when a player rolls a 1 and doesn't have enough coins to pay the penalty.

Based on the game's rules, a few things of interest become obvious. Player A seems to have an advantage in that they have a chance to take from the central pot first. However, Player A also has a disadvantage in that they can lose the game the roll before Player B might have also lost. As with any game of chance, we will want to see if the game is fair or if either player has a distinct advantage.

The goal of this project is to use simulation to determine if going first or second offers an advantage, the expected number of turns for the game to complete, and the distribution of the number of turns the game takes.  A Python script and Monte Carlo simulation with 1,000,000 runs will be used to answer these three questions. While we trust the built-in random functionality of a well-adopted language like Python, we also perform a visual, goodness of fit, and test of independence on our die. After we have analyzed the game under the base conditions, we can tweak some of the starting parameters to understand how they affect the game's outcomes.

## Methodology

Before coding the logic for the game, we wanted to validate the randomness of our die roll, as it is central to the random variation from game to game. While we trust the robustness of the Python random library, it is best practice to go ahead and check. We used the following three tests:

1. Visually via a Monte Carlo simulation of 100,000 rolls and a resulting histogram (Fig. 1).
2. A Chi-Squared goodness of fit test with an alpha level of 0.05.
3. A Runs test of independence with an alpha level of 0.05 using the statsmodels library.

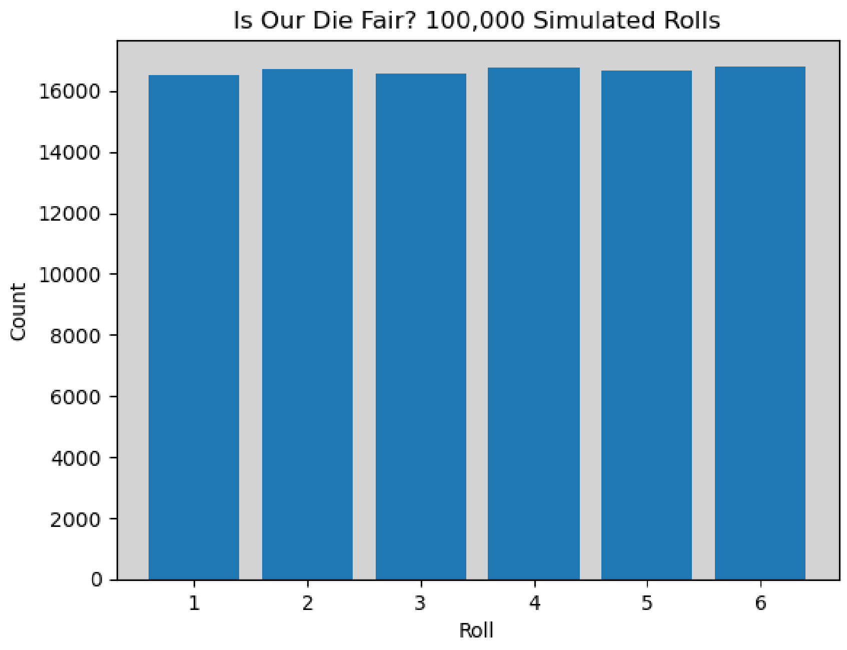

Fig. 1 - Simulated die roll 100,000 times.

Test 1 is shown above, where the die rolls seem to be evenly distributed. The Chi-Squared test statistic was 4.017 and the critical value was 11.07, meaning the goodness of fit test passed. Similarly, the test of independence passed, producing a test statistic of 1.12 and a p-value of 0.26 meaning. Unsurprisingly, our random number generator has good statistical properties and is good to use.

Next, the logic of the game was coded using a Jupyter notebook environment. After verification that the game was running properly, Monte Carlo simulation was once again used, this time with 1,000,000 runs to simulate game results. This number was chosen as it was large enough to appeal to the law of large numbers for statistical rigor, while balancing run time to generate results. After this, we can graph the distribution, provide an expected value, and determine if either player has an advantage.

## Results

Over 1,000,000 simulations of the base game, player A won the game 502,278 times (50.23%) and player B won 497722 (49.77%) of the time. This indicates that the game is very balanced. The distribution of all 1,000,000 turns until completion is graphed in Fig. 2 below:

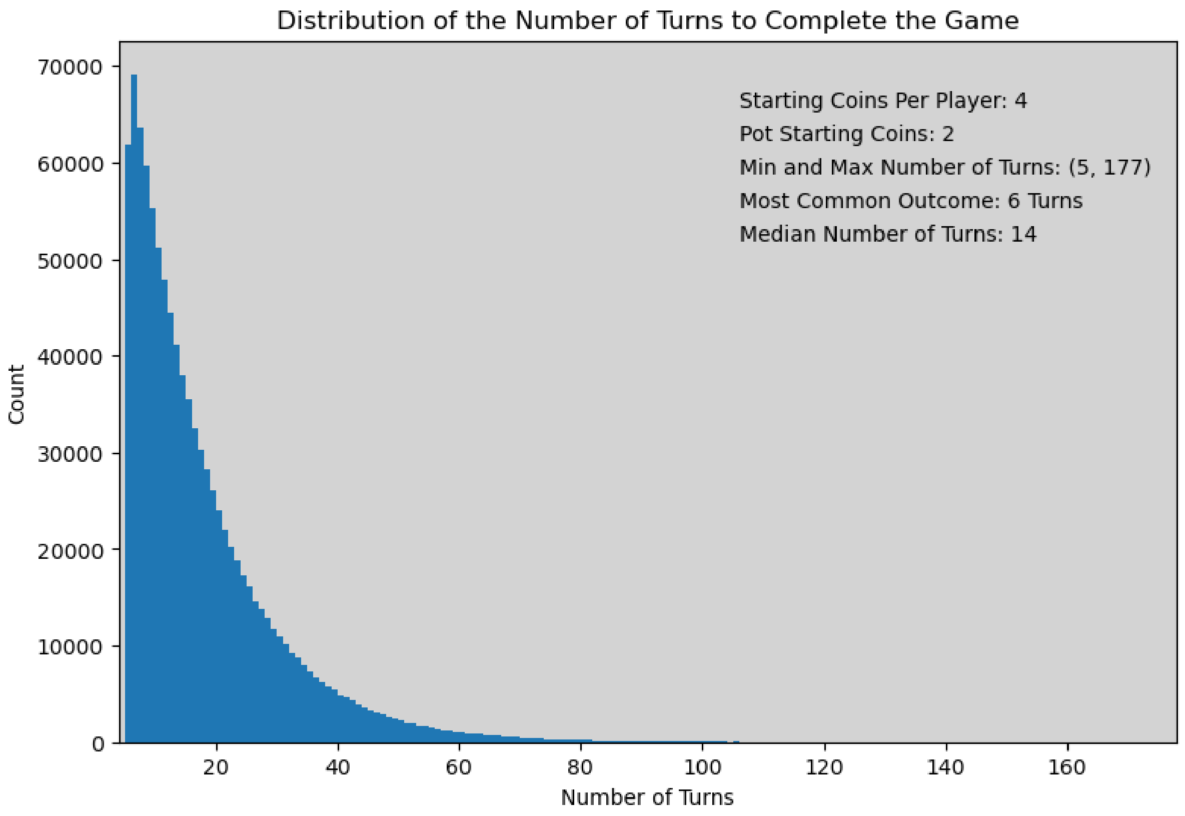

Fig. 2 - Number of turns to finish the game simulated 1,000,000 times.

The distribution above appears to be geometric with 6 turns being the most common result, a median of 14 turns, and a mean of 17.51 turns. There is one key exception to the geometric distribution and that is the case when the game ends in 5 turns. This gives the distribution a small left shoulder. This special case is the result of either player rolling a 4, 5, or 6 on each of the first 5 rolls. This happens with a probability of (1/2)^5 for each player.

Next, we wanted to see the effect of changing the number of coins each player started with on the distribution of the length of the game. Choosing values of 8 (Fig. 3) and 16 (Fig. 4) starting coins each, we re-ran the simulation 1 million times for each condition:

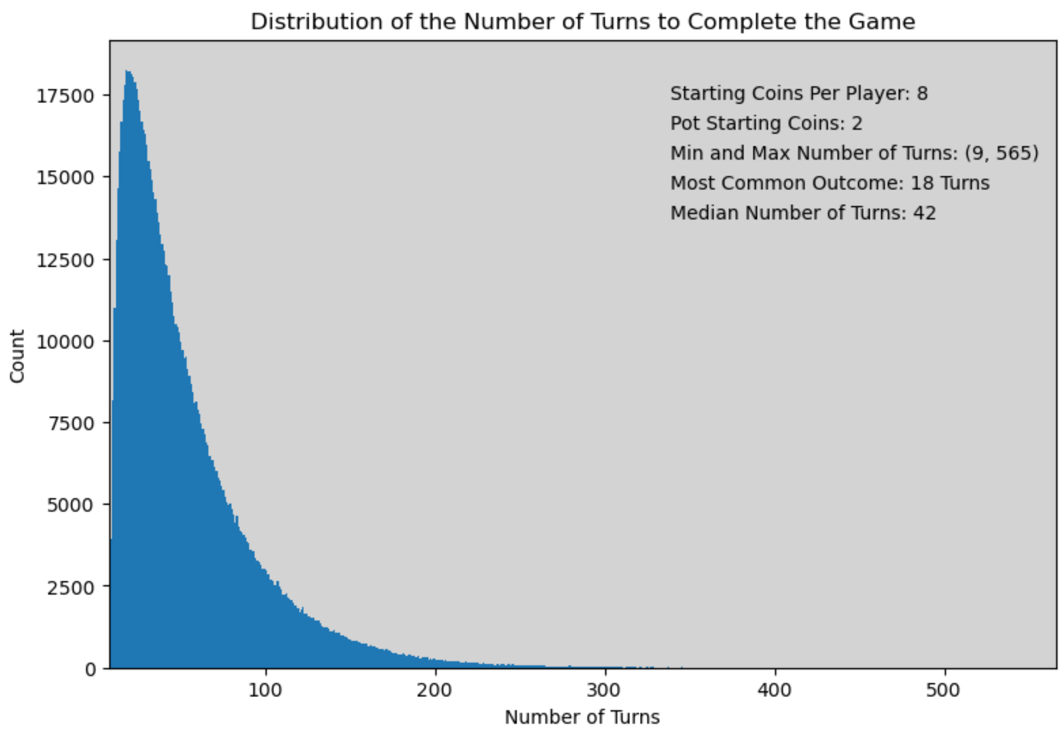

Fig. 3 - Number of turns to complete the game, increasing player starting coins to 8, 1,000,000 simulations.

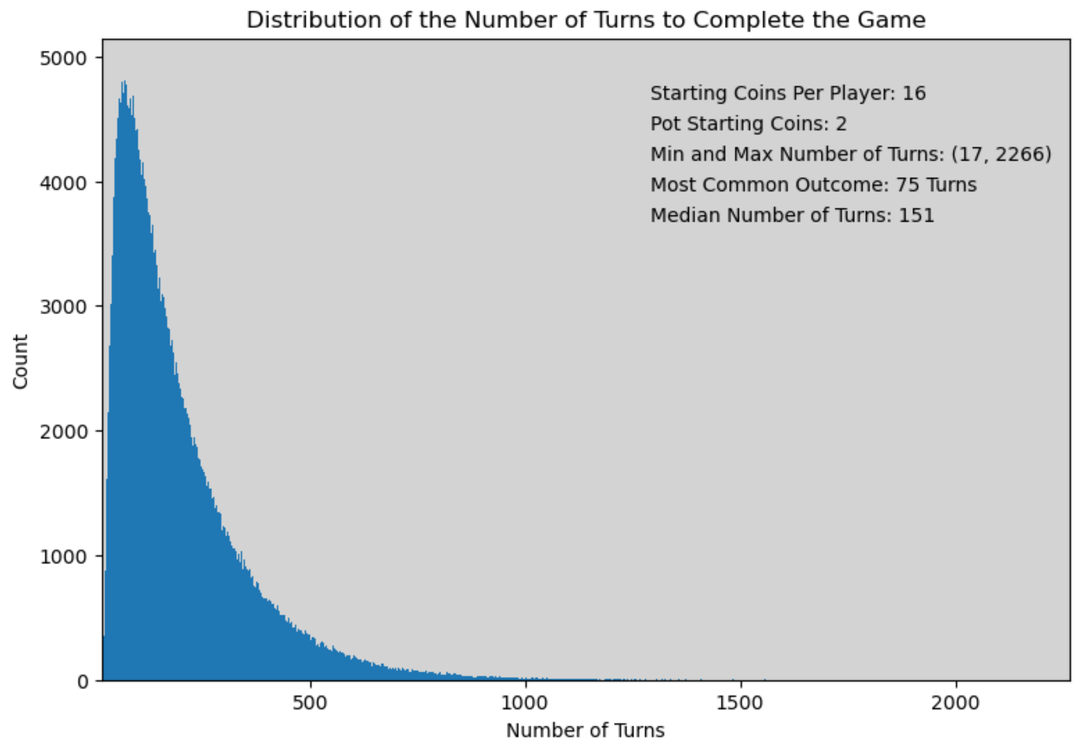

Fig. 4 - Number of turns to complete the game, increasing player starting coins to 16, 1,000,000 simulations.

By increasing the number of coins each player starts with, and keeping the starting pot constant, a few things jump out. First, the distribution shape appears less geometric, and the left-hand shoulder is more apparent. Now the distribution looks like a negative binomial distribution with a heavy right tail. The minimum number of turns to complete the game is bound by the number of starting coins of the player plus the penalty (1). The gap between the minimum number of observed turns to complete the game and the most common outcome is increasing. The range of values in game length also increases as we increase the starting coins.

Next, we wanted to see the impact of changing the number of coins initialized in the pot. We tested values of 4 (Fig. 5) and 8 (Fig. 6) and again ran the simulation one million times.

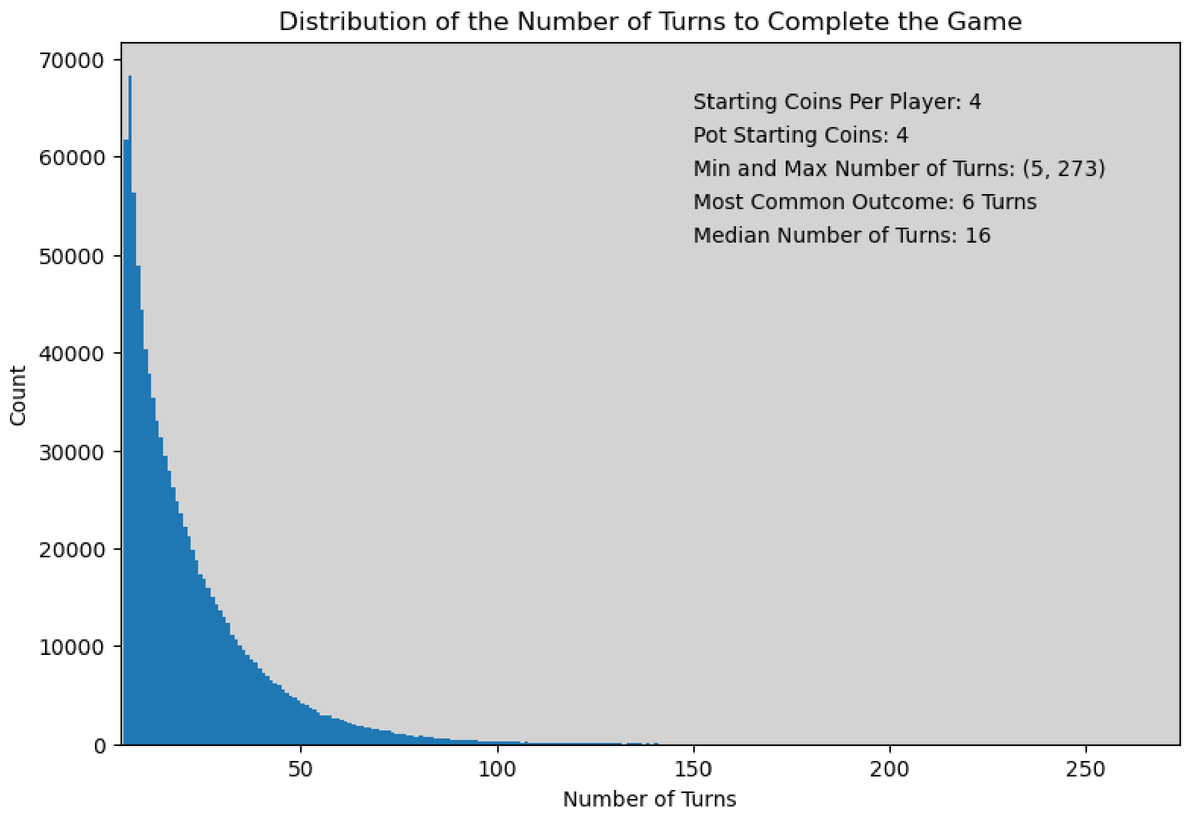

Fig. 5 - Number of turns to complete the game, increasing pot start size to 4, 1,000,000 simulations.

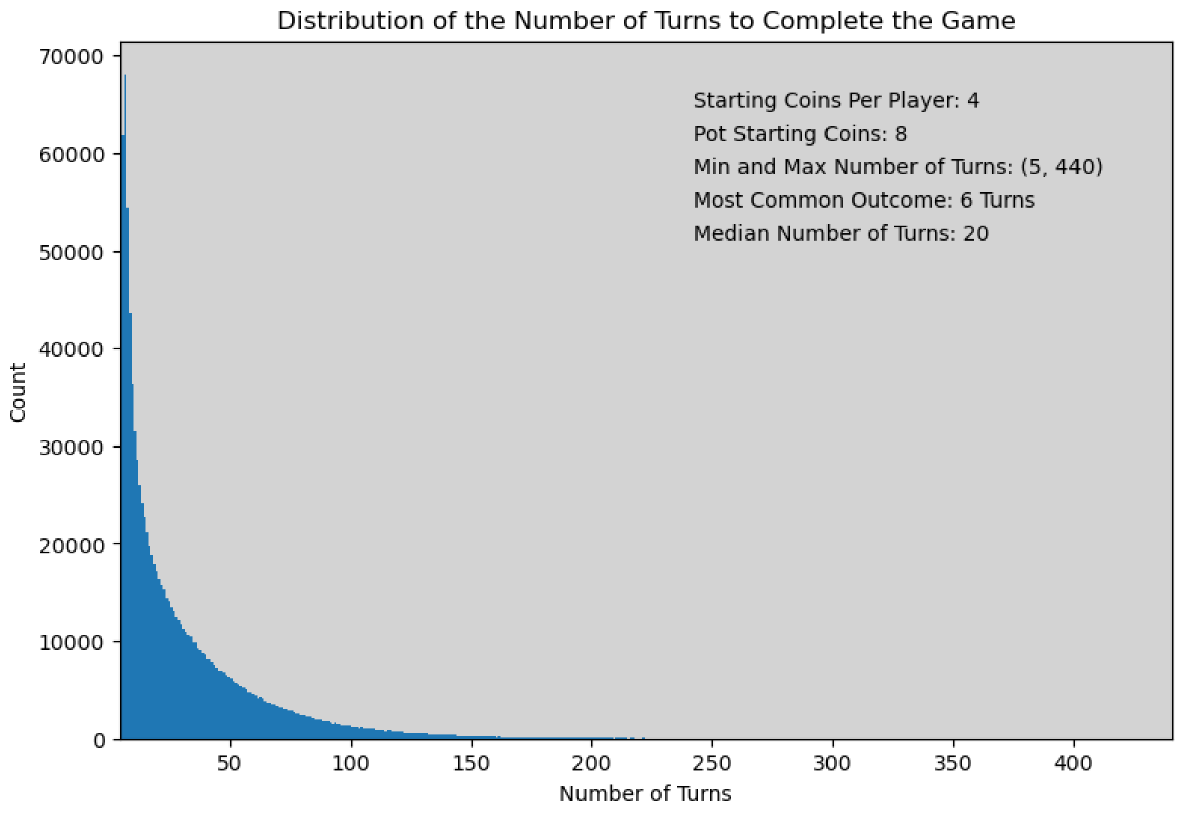

Fig 6. Number of turns to complete the game, increasing pot start size to 8, 1,000,000 simulations.

Interestingly, changing the pot size, while keeping the player starting coins constant didn't change the distribution shape as much. It appears geometric, just like the base game. In fact, the minimum value of 5 and the most common outcome of 6 did not change. However, the median value and maximum number of turns did increase.

Finally, we wanted to see what would happen if we changed both variables simultaneously. Selecting a value of 16 coins per player and an initial pot size of 8 (Fig. 7), we fired up the simulation once again for one million iterations.

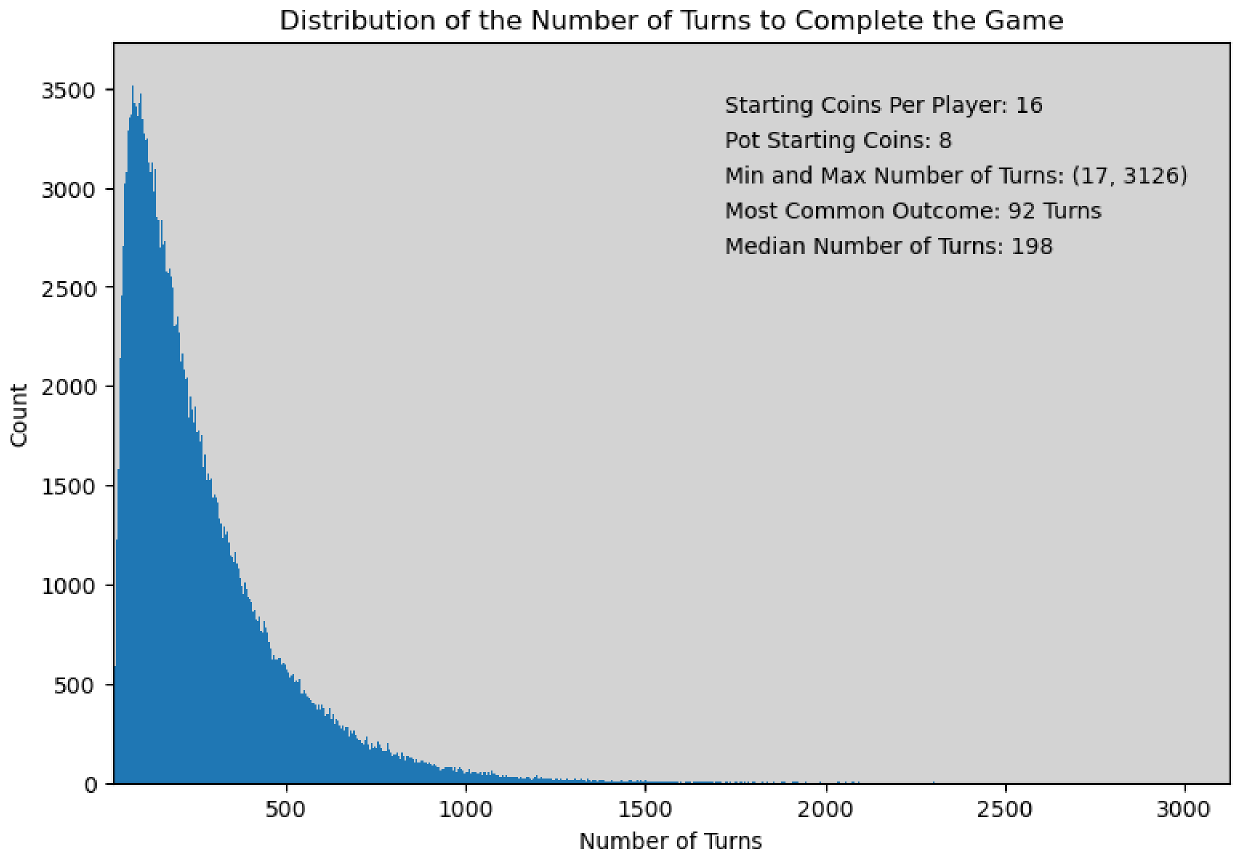

Fig 7. Number of turns to complete the game, increasing pot start size to 8 and 16 coins per player, 1,000,000 simulations.

By changing both the starting coins per player and the starting pot size, we see the widest range in outcomes in the number of turns to complete the game. The distribution in this configuration of the game has the highest median, most common outcome, and range of values. The distribution appears to be a negative binomial with a small left-hand shoulder present similar to those in Figures 3 & 4.

## Conclusions

Through this project we were able to use simulation to determine the fairness of a game, the expected number of turns it takes to complete it, as well as the distribution of the number of turns to complete it. While a closed-form solution for this relatively simple game most likely exists, building the simulation in a programmatic way allowed us to test an array of different inputs and get results in a relatively time-efficient manner. Throughout this project, we learned the importance of verifying all assumptions and validation of randomization to ensure the trustworthiness of results. For example, it was vital for us when starting to confirm the dice is fair. This is to ensure that the simulation represents the random process that was intended. It was also vital to perform a goodness of fit test and check for independence, as it helps us build confidence in our simulated model. Future work for this project should look at a wider range of values for the starting number of coins for each player and the starting pot size.

It would also be interesting to change the die from a standard six sided one, to one with either fewer or more sides. How would the game be different if we added a third player? With our simulation framework already built, only small adaptations to the code are needed so we can address questions like these and many more.