# Game Simulation: An Application of Simulation Techniques

Anish Thomas

*Abstract*—Simulation study has many applications. One such application is in the area of game development. In this work a program oriented simulation of a two player game is conducted with the goal of finding out the duration to which it would last. While a simulation provided empirical results on a fixed set of runs of the game, an application of analytical method derived the expected value for the duration of each game. Distribution of frequencies for the number of cycles of a game set in the simulations were computed and visualized as histograms.

## 1 PROBLEM DESCRIPTION

The goal of this work is to determine the expected number and the distribution of cycles a game will last for. A simulation of the game is used to find the entire distribution. First step analysis is used to arrive at the expected value[1] of the cycle count.

## 2 GAME DESIGN

### 2.1 Rules of Game

Below is an explanation of the game as provided in the projects sheet. There are two players, A and B. At the beginning of the game, each starts with 4 coins, and there are 2 coins in the pot. A goes first, then B, then A, and so on. During a particular player's turn, the player tosses a 6-sided die. If the player rolls a:

- "1", then the player does nothing.
- "2", then the player takes all coins in the pot.
- "3", then the player takes half of the coins in the pot (rounded down).
- "4" or "5" or "6", then the player puts a coin back in the pot.

A player loses and the game is over if they are unable to perform the task i.e., if they have zero coins and need to place one in the pot. We define a cycle as A and then B completing their turns. The exception is if a player goes out; that is the final cycle, but it still counts as the last cycle.

### 2.2 Game Flow Logic

There is a total of ten coins. At the start of the game the both the players are given four coins each and two coins are placed in the pot. The cycle count is initialized to zero. The moves of the game are determined by the results of throwing a fair die, which is similar to randomly choosing a number from 1 to 6, representing each face of a six sided die. Most of the simulation software and programming languages provides some type of function to generate random numbers.

Player A is given the first chance to throw the die i.e. pick a number between 1 and 6. Based on the throw A gains or loses coins or takes no action. At this time the count of coins for A and that in the pot either changes or remains the same. Next is player B's turn to throw the die. Based on the throw B also gains or loses a coin or takes no action. The count of coins with B and that in the pot are adjusted to reflect B's action. As soon as both the players chances are completed, the cycle count is incremented by one. At each step it is evaluated whether the end of game is reached. Usually, at some point Player A or player B is unable to take an action, when they have zero coins with them and they need to put a coin to the pot. This ends the game, and the cycle count is incremented by one.

## 3 SIMULATION

To perform the simulations, there exists multiple tools and applications. In this project work, a program in Python language has been written to simulate the game. There are two functions created: `play_game()` and `run_simulations(n)`. The first function is used to play a single game and return the total number of cycles the game lasted for. The second function is used to simulate the game i.e. run the game as many times as needed for the simulation. For example, if the input parameter n=100, there will be a hundred individual games played and the frequency distributions of the cycles counts of those games will be calculated and displayed. *(Code available in Appendix Section 8.1).*

## 4 SIMULATION RESULTS

A total of two sets of simulation experiments were performed. The first experiment set had a total of hundred games played and then calculated the average cycle times per game. The second experiment set had a total of two thousand and five hundred games played and the average cycle times were calculated. Below is a sample result from running the simulation in Python.

```
>python anothergame.py

100 Simulations: 18.01
2500 Simulations: 17.60
```

### 4.1 Distributions

The distribution of cycle counts for each of the simulations sets is provided below (Figure 1 and Figure 2). The frequencies are automatically divided into different bins. The lowest value for any bin is 5, which is the lowest number of cycles a game has to be played for either of the players to lose and thus reaching the end of game. The horizontal line indicates the average cycles count for an entire simulation set.

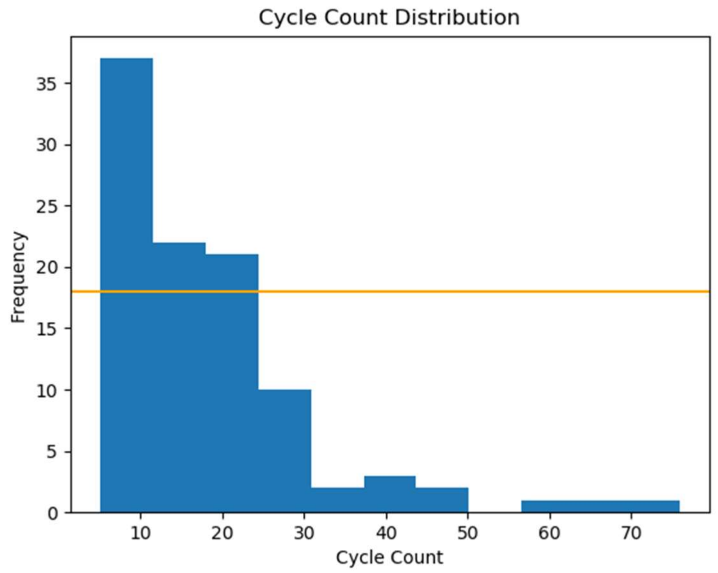

***Figure 1—*** Distribution of cycle counts for a simulation set of 100 games with an average cycle count of 18.01.

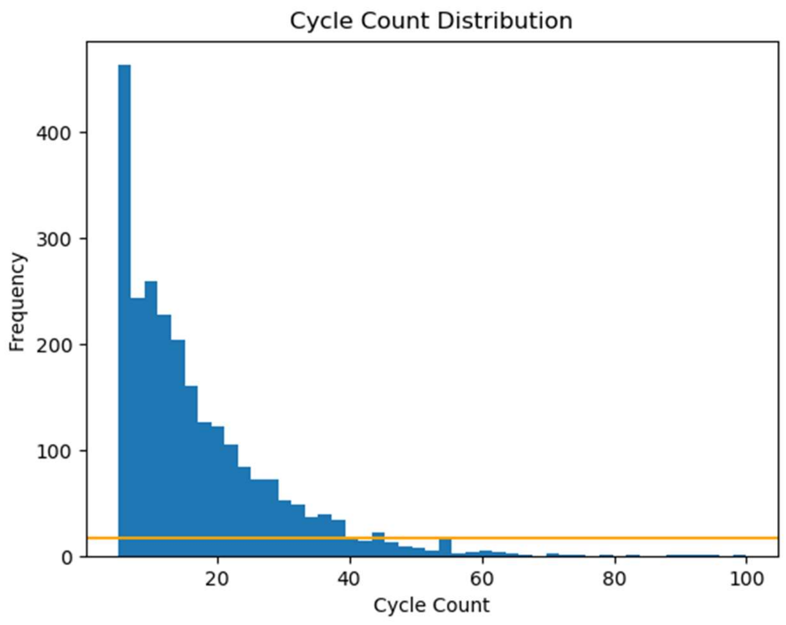

*Figure 2*— Distribution of cycle counts for a simulation set of 2500 games with an average cycle count of 17.60.

## 5 EXPECTATION

The expected number of cycles for the game will be calculated next. Before doing the calculations, the concepts of Markov Chains and First step analysis are briefly introduced.

### 5.1 Markov Chain

A discrete stochastic processes that possess the following property - the future state of the process depends only upon the present state and not on the past states leading up to the present state. This property is called the Markov property. Any stochastic process that possesses the Markov property is called a Markov chain [3]. In this game a player's future amount is dependent only on the current amounts of both the players and that in the pot. It is not dependent upon any of the previous amounts. Thus here the states of the game [2] forms a Markov chain.

If there exists a Markov chain with transition probability matrix P, then the elements of P are the one-step transition probabilities and can be denoted by $P_{ij}$. As the Markov process moves through the states over a time, the probabilities in the matrix P shows how likely the process will transition to state j in the next time period if the process is currently in state i [3]. The transition probabilities in P are independent of the time period n and are called stationary transition probabilities.

### 5.2 First Step Analysis

Using a technique called First Step Analysis, it is possible to arrive at the n-step transition probabilities $P_{ij}^{n}$ given the one-step transition probabilities $P_{ij}$ for the states of the game. This method breaks down the possibilities resulting from the first step or first transition in the Markov chain. Then it uses the law of total probability and Markov property to derive a set of relationship among the unknown variables[3]. The approach used here are based on the fundamental matrix.

The transition probability matrix P is deducted based on the possible number of coins for the Player A at any given step. At any given moment there can be a total of zero to ten coins with Player A. For the Player A, at each step, the result of a die's throw and the number of coins with Player B and that in the pot determines the probability of reaching a specific next step. The transition probability matrix P for the game is shown in "*Appendix 8.2: Matrix Calculations.*"

Using the transition probability matrix P, it is possible to construct a matrix Q which is the matrix of transition probabilities for the transient states. The matrix I is an identity matrix describing the absorbing states which has the same dimensions as matrix Q. Using matrix Q and I, matrix $W=(I-Q)^{-1}$, the inverse of (I-Q), called the fundamental matrix for the absorbing chain can be calculated. The fundamental matrix can be used to find the probabilities of absorption and the mean times to absorption. In this project work we have calculated the mean time to absorption which is the end of game. The expected value (E) of total number of cycles for the game is found to be 17.76. The "*Appendix 8.2: Matrix Calculations*" shows the matrix calculations. This shows that, on an average, a duration of about seventeen cycles needs to be played before either of the player loses and the game reaches its end.

## 6 CONCLUSION

This project work was built on the foundational concepts of simulation learned through the academic course. An in depth study on how to understand

and design a game, how to effectively simulate it, how to apply the concepts such as first step analysis etc. were accomplished and the results were presented. The results of the simulation and the expected values were found to be closely aligned with each other. Based on the strong foundation built in this work, it is possible to take up simulation work on different types of games in the future.

## 7 REFERENCES

1. A First Course in Probability and Statistics
   David Goldsman, Ph.D.
2. Probability and Random Processes With Applications to Signal Processing and Communications
   Scott L. Miller and Donald Childers
3. Topics in probability concepts in applied probability
   www.probabilitytopics.wordpress.com

## 8 APPENDICES

### 8.1 Python Code

Program File Name: anothergame.py

To run the program, go to the file directory via a Python console and run the following command.

```
>python anothergame.py
```

Note: The Numpy library function `random.randint()` is used to generate integer random numbers from 1 to 6, to mimic the dice throw.

```
import numpy as np
from matplotlib import pyplot as plt

def play_game():
    """
    Returns: cycle_count
    """

    cycle_count = 0
    a_coins = 4
    b_coins = 4
    pot_coins = 2

 while True:
        # Player A
        throw = np.random.randint(1,7,1)
        if throw!= 1:

            if throw == 2:
                a_coins = a_coins + pot_coins
                pot_coins = 0
            elif throw == 3:
                extra = np.floor_divide(pot_coins,2)
                a_coins = a_coins + extra
                pot_coins = pot_coins - extra
            else: # 4,5,6
                if a_coins == 0:
                    cycle_count += 1
                    break
                else:
                    a_coins -= 1
                    pot_coins += 1
        else: # if 1, do nothing
            pass

        # Player B
        throw = np.random.randint(1,7,1)
        # print("Throw2:",throw)
        if throw!= 1:
            if throw == 2:
                b_coins = b_coins + pot_coins
                pot_coins = 0
            elif throw == 3:
                extra = np.floor_divide(pot_coins,2)
                b_coins = b_coins + extra
                pot_coins = pot_coins - extra
            else: # 4,5,6
                if b_coins == 0:
                    cycle_count += 1
                    break
                else:
                    b_coins -= 1
                    pot_coins += 1
        else: # if 1, do nothing
            pass
        cycle_count += 1
    return cycle_count

def run_simulations(game_count):
    """
    Args:
        game_count:
    Returns:
        cycle_counts[]
    """

    total_count = 0
    all_cycles = np.array([], dtype=np.int64)
    for id in range(0, game_count):
        count = play_game()
        total_count += count
        all_cycles = np.append(all_cycles,count)
    cycle_counts = total_count / game_count

    # Display Histogram
    plt.hist(all_cycles, bins='auto')
    plt.title("Cycle Count Distribution")
    plt.ylabel("Frequency")
    plt.xlabel("Cycle Count")
    plt.axhline(cycle_counts,color='orange')
    plt.savefig("Graph.png")
    return cycle_counts

if __name__ == "__main__":
    cycle_100 = run_simulations(100)
    cycle_2500 = run_simulations(2500)

    print("100 Simulations:",cycle_100)
    print("2500 Simulations:",cycle_2500)
```

## 8.2 Matrix Calculations

The starting state of the game is (4,4,2) where Player A has 4 coins, Player B has 4 coins and there are 2 coins in the pot. Thus the starting step for Player A is four, from which it will reach the end of game (an absorbing state). The expected value E of 17.76 was arrived via the following matrix calculations and then summing up the entire transition probability values from matrix W on the row of starting state for the Player A. Using matrix W it is possible to find the expected value for any of the possible starting states for the Player A.

$$P = \begin{pmatrix}
\frac{47}{66} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{22} & \frac{1}{33} & \frac{1}{66} & \frac{1}{66} & \frac{1}{66} & \frac{1}{66} & \frac{1}{66} \\
\frac{1}{2} & \frac{13}{60} & \frac{1}{20} & \frac{1}{20} & \frac{1}{20} & \frac{1}{20} & \frac{1}{60} & \frac{1}{60} & \frac{1}{60} & \frac{1}{60} & \frac{1}{60} \\
0 & \frac{1}{2} & \frac{2}{9} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{27} & \frac{1}{54} & \frac{1}{54} & \frac{1}{54} & \frac{1}{54} \\
0 & 0 & \frac{1}{2} & \frac{11}{48} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{48} & \frac{1}{48} & \frac{1}{48} & \frac{1}{48} \\
0 & 0 & 0 & \frac{1}{2} & \frac{5}{21} & \frac{1}{14} & \frac{1}{14} & \frac{1}{21} & \frac{1}{42} & \frac{1}{42} & \frac{1}{42} \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{36} & \frac{1}{36} & \frac{1}{36} \\
0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{4}{15} & \frac{1}{10} & \frac{1}{15} & \frac{1}{30} & \frac{1}{30} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{7}{24} & \frac{1}{8} & \frac{1}{24} & \frac{1}{24} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{9} & \frac{1}{18} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{5}{12} & \frac{1}{12} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2}
\end{pmatrix}$$

$$Q = \begin{pmatrix} \frac{13}{60} & \frac{1}{20} & \frac{1}{20} & \frac{1}{20} & \frac{1}{20} & \frac{1}{60} & \frac{1}{60} & \frac{1}{60} & \frac{1}{60} \\ \frac{1}{2} & \frac{2}{9} & \frac{1}{18} & \frac{1}{18} & \frac{1}{18} & \frac{1}{27} & \frac{1}{54} & \frac{1}{54} & \frac{1}{54} \\ 0 & \frac{1}{2} & \frac{11}{48} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{48} & \frac{1}{48} & \frac{1}{48} \\ 0 & 0 & \frac{1}{2} & \frac{5}{21} & \frac{1}{14} & \frac{1}{14} & \frac{1}{21} & \frac{1}{42} & \frac{1}{42} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{36} & \frac{1}{36} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{4}{15} & \frac{1}{10} & \frac{1}{15} & \frac{1}{30} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{7}{24} & \frac{1}{8} & \frac{1}{24} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{3} & \frac{1}{9} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{5}{12} \end{pmatrix}$$

$$I = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$W = (I - Q)^{-1} = \begin{pmatrix} 1.58 & 0.481 & 0.590 & 0.698 & 0.778 & 0.781 & 0.755 & 0.621 & 0.364 \\ 1.29 & 2.02 & 1.01 & 1.20 & 1.35 & 1.37 & 1.30 & 1.07 & 0.624 \\ 1.07 & 1.68 & 2.50 & 1.56 & 1.78 & 1.84 & 1.70 & 1.39 & 0.813 \\ 0.909 & 1.42 & 2.12 & 3.03 & 2.10 & 2.20 & 2.04 & 1.64 & 0.955 \\ 0.786 & 1.23 & 1.84 & 2.62 & 3.54 & 2.50 & 2.35 & 1.84 & 1.06 \\ 0.687 & 1.08 & 1.61 & 2.29 & 3.10 & 3.93 & 2.62 & 2.06 & 1.16 \\ 0.605 & 0.949 & 1.42 & 2.02 & 2.73 & 3.46 & 4.07 & 2.31 & 1.24 \\ 0.530 & 0.830 & 1.24 & 1.76 & 2.39 & 3.03 & 3.56 & 3.77 & 1.42 \\ 0.454 & 0.711 & 1.06 & 1.51 & 2.05 & 2.60 & 3.05 & 3.23 & 2.93 \end{pmatrix}$$