ISYE 6644: Simulation

Another Game: Using First Step Analysis and Simulation to Determine the Expected Number of Game Cycles

Group 192: Frances DePree, Julia Lee

Abstract

Developers and players of chance-based games want to know statistics, such as how likely they are to win and how long the game will last. This paper examines a specific dice and coin game by modeling it as a memoryless Markov Chain. We then determine how long the game is likely to last using First Step Analysis as well as Monte Carlo simulation and compare the results. Both of these methods proved useful in determining the expected length of the game, and using Monte Carlo simulation we are also able to determine the distribution of the expected length of the game.

Background and Description of Problem

The topic of focus for the project is demonstrating the use of First Step Analysis and Monte Carlo simulation on a dice game. While the rules of the game used in this project are specific, the application of First Step Analysis and Monte Carlo simulation is far reaching and will apply to any other game dependent on purely statistical outcomes. The game used in this project is called simply, "Another Game."

The game is played by two players, Player A and Player B, who each start with 4 coins. Another 2 coins remain in the pot for a total of 10 coins in play. The players take turns rolling a 6-sided die and take an action depending on the number they roll.
If the player rolls a 1, no action is taken. If the player rolls a 2, they gain all the coins in the pot. If the player rolls a 3, the player takes half the coins in the pot, rounded down. If the player rolls a 4, 5, or 6, then they have to put one coin in the pot.

A player loses when they are not able to perform an action. For example, if a player rolls a 4 but has no coins left to put in the pot. A full cycle is complete when both Player A and Player B have taken their turn. We further defined these rules by assuming that if a player rolls a 3 and the pot only has one coin, then they do not take the coin but continue the game. Additionally, if the pot has zero coins and a player rolls a 2 or a 3, the game will continue. Essentially, the only way that a player can lose the game is if they roll a 4, 5, or 6 but have no remaining coins to contribute to the pot.

The objective of the project is to simulate the expected number of cycles a game will last for, as well as determine the distribution of the number of cycles over multiple iterations of the game.

Theory

Markov Chains

A stochastic process is a family of random variables X_n where $n \in \tau$ and τ is the index set which can be discrete for $\tau = \{0,1,2,3,\ldots\}$, a discrete time stochastic process, or $\tau = [0\,,\infty)$ for a continuous time stochastic process [1]. Focusing on discrete time stochastic processes, let the state space S be the set where X_n takes all its values $\{X_n : \mathrm{n} = 0, 1, 2, 3\ldots\}$ and so the probability model then is determined by $P(X_0 = x_0, X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ for all values of n and where $x_0, x_1, x_2, \ldots, x_n \in S$ (all possible values of X_i) [1]. A discrete time stochastic process is also known as a Markov Chain if for every $x_0, x_1, x_2, \ldots, x_n \in \mathrm{S}$, $n \geq 0$,

$$P(X_n = i|X_{n-1} = j, \ldots, X_0 = x_0) = P(X_n = i|X_{n-1} = j)$$

(Known as the Markov Property)

or

$$P(X_{n+1} = i|X_n = j) \text{ [2]}$$

which are called transition probabilities. Transition probabilities can be better represented as a transition matrix of the Markov Chain X_n, where p_{ij} is the probability of state *i* to state *j* [4].

$$P = (p_{ij}) = \begin{pmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mn} \end{pmatrix}$$

Markov Chains are "memoryless" or have "one-step memory" which means that the probability of transitioning into a state only depends on the current state and not on any other previous historical information [1].

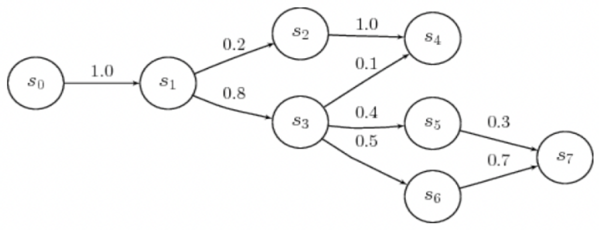


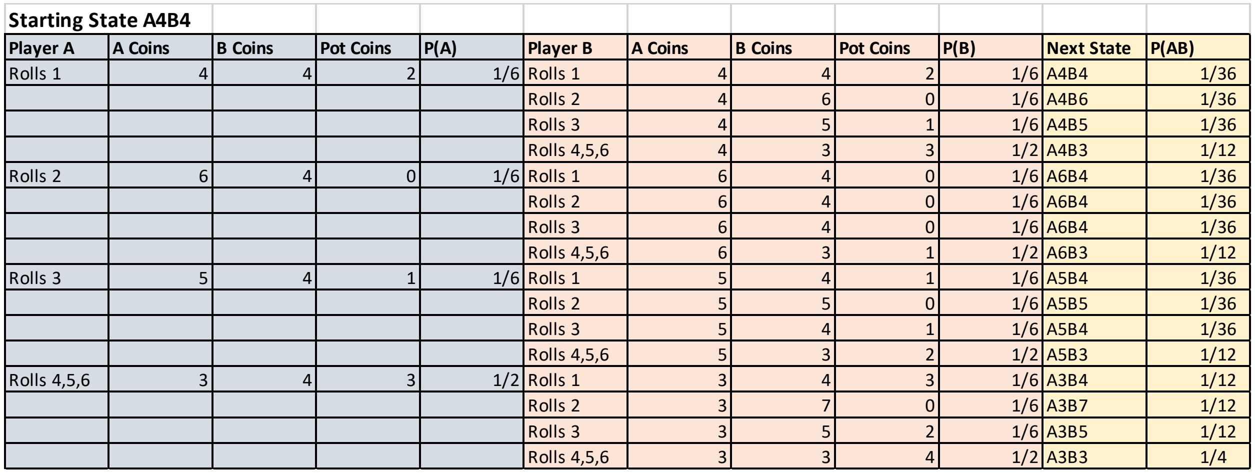
Figure 1: Example of a simple directed graph representing a Markov Chain [3]

We can think of a "state" as a set of variables that fully describe the system at any point in time. In our example of "Another Game," the amount of coins Player A and B have, say Player A = 3, Player B = 3, represents the system state at that time. Note that we know there are a total of 10 coins in the game, therefore, if both players have 3 coins, then we know that the pot would have 4 coins [2]. It is important to note that a state is said to be 'absorbing' if $p_{ij} = 1$, and where $i = j$, which means that every other element in that row in the transition matrix would be 0. With this, we can rewrite the matrix P above like

$$P = \begin{pmatrix} I & O \\ R & Q \end{pmatrix}$$

If we say that there are s transient states and $r - s$ absorbing states, then I is an identity matrix $(r - s)$ X $(r - s)$, O is a submatrix consisting of all 0's, R is a submatrix of probabilities that a transient state will transition to an absorbing state, and lastly Q is the submatrix of the probabilities that a transient state transitions to another transient state [6]. For any absorbing Markov chain, $I - Q$ has an inverse $(I - Q)^{-1} = I + Q + Q^2 + \ldots = \sum_{k=0}^{\infty} Q^k$, where k is the number of steps. This inverse $(I - Q)^{-1}$ is defined as the fundamental matrix N [6].

Furthermore, the described game can be represented by a system of states that transition from one state to the next state. Taking a state to be the number of coins each player has; we can predict the number of cycles each game lasts for, starting from the state Player A and Player B having 4 coins (2 in the pot). Starting with this state A_4B_4, say Player A rolls the die and it lands on a 2. The die has equal probability for all 6 sides, so $P(Player\ A\ rolls\ 2) = \frac{1}{6}$. Next let us say that Player B rolls a 4, 5, or 6 $P(Player\ B\ rolls\ 4, 5, 6) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$. This full cycle will give Player A 2 coins for a total of 6 coins, and Player B would give 1 coin to the pot leaving them with 3 coins, which can be represented by state A_6B_3. This transition from A_4B_4 to A_6B_3 has the probability $P(Player\ A\ rolls\ 2) * P(Player\ B\ rolls\ 4, 5, 6) = \frac{1}{6} * \frac{1}{2} = \frac{1}{12}$. Given this, we can find the probability of transition states in the table below with the starting state of A_4B_4.



Starting State A4B4											
Player A	**A Coins**	**B Coins**	**Pot Coins**	**P(A)**	**Player B**	**A Coins**	**B Coins**	**Pot Coins**	**P(B)**	**Next State**	**P(AB)**
Rolls 1	4	4	2	1/6	Rolls 1	4	4	2	1/6	A4B4	1/36
					Rolls 2	4	6	0	1/6	A4B6	1/36
					Rolls 3	4	5	1	1/6	A4B5	1/36
					Rolls 4,5,6	4	3	3	1/2	A4B3	1/12
Rolls 2	6	4	0	1/6	Rolls 1	6	4	0	1/6	A6B4	1/36
					Rolls 2	6	4	0	1/6	A6B4	1/36
					Rolls 3	6	4	0	1/6	A6B4	1/36
					Rolls 4,5,6	6	3	1	1/2	A6B3	1/12
Rolls 3	5	4	1	1/6	Rolls 1	5	4	1	1/6	A5B4	1/36
					Rolls 2	5	5	0	1/6	A5B5	1/36
					Rolls 3	5	4	1	1/6	A5B4	1/36
					Rolls 4,5,6	5	3	2	1/2	A5B3	1/12
Rolls 4,5,6	3	4	3	1/2	Rolls 1	3	4	3	1/6	A3B4	1/12
					Rolls 2	3	7	0	1/6	A3B7	1/12
					Rolls 3	3	5	2	1/6	A3B5	1/12
					Rolls 4,5,6	3	3	4	1/2	A3B3	1/4

Figure 2: Probabilities of next state given the starting state A_4B_4

Starting State AnBm where n and m > 0. All variables are defined at the start of the cycle.									
p represents the number of coins in the pot at the beginning of the cycle, defined as p = 10-n-m									
Player A	**A Coins**	**B Coins**	**P(A)**	**Player B**	**A Coins**	**B Coins**	**P(B)**	**Next State**	**P(AB)**
Rolls 1	n	m	1/6	Rolls 1	n	m	1/6	A_nB_m	1/36
				Rolls 2	n	m+p	1/6	A_nB_{m+p}	1/36
				Rolls 3	n	m+floor[p/2]	1/6	$A_nB_{m+floor[p/2]}$	1/36
				Rolls 4,5,6	n	m-1	1/2	A_nB_{m-1}	1/12
Rolls 2	n+p	m	1/6	Rolls 1	n+p	m	1/6	$A_{n+p}B_m$	1/36
				Rolls 2	n+p	m	1/6	$A_{n+p}B_m$	1/36
				Rolls 3	n+p	m	1/6	$A_{n+p}B_m$	1/36
				Rolls 4,5,6	n+p	m-1	1/2	$A_{n+p}B_{m-1}$	1/12
Rolls 3	n+floor[p/2]	m	1/6	Rolls 1	n+floor[p/2]	m	1/6	$A_{n+floor[p/2]}B_m$	1/36
				Rolls 2	n+floor[p/2]	m+ceil[p/2]	1/6	$A_{n+floor[p/2]}B_{m+ceil[p/2]}$	1/36
				Rolls 3	n+floor[p/2]	m+floor[ceil[p/2]/2]	1/6	$A_{n+floor[p/2]}B_{m+floor[ceil[p/2]/2]}$	1/36
				Rolls 4,5,6	n+floor[p/2]	m-1	1/2	$A_{n+floor[p/2]}B_{m-1}$	1/12
Rolls 4,5,6	n-1	m	1/2	Rolls 1	n-1	m	1/6	$A_{n-1}B_m$	1/12
				Rolls 2	n-1	m+p+1	1/6	$A_{n-1}B_{m+p+1}$	1/12
				Rolls 3	n-1	m+floor[(p+1)/2]	1/6	$A_{n-1}B_{m+floor[(p+1)/2]}$	1/12
				Rolls 4,5,6	n-1	m-1	1/2	$A_{n-1}B_{m-1}$	1/4

Figure 3: Probabilities of next state given generic starting state $A_\mathrm{n}B_\mathrm{m}$

It is important to note that A_nB_m where n and m > 0 have potentially different logic then if n and/or m = 0. We say this because if a player has 0 coins, we are $P = \frac{1}{2}$ that the game would end in that turn, whereas if n and m > 0, and the player rolled a 4, 5, or 6, the game would continue.

This theory of Markov Chains and transition state probabilities leads to First Step Analysis, which is a technique used by breaking down the total possibilities after the first transition, the Markov property and the law of total probability to derive the expected time to reach a certain state, along with the probability of reaching that state [4]. In Figure 2, we have a starting state of A_4B_4, but we can describe the game based on any of the starting states, $A_{10}B_0$ (Player A has all coins, Player B and the pot have none), all the way to A_0B_{10} (Player B has all the coins, Player A and the pot have none).

Simulation

As this is a game of probabilities, there are many outcomes that may occur. Since the outcome of the game is left only to chance rather than the skill of the players, it is well suited to simulation. Steady-state simulation is the technique best suited for Markov Chains. Discrete events are defined by each player taking a turn, which will then determine the next state that the system is in. The simulation will end when a player loses the game.

A Monte Carlo analysis of the game using multiple repetitions, “r”, of the simulation can be performed to determine the expected value of the number of cycles the game will last for and to find out the distribution of the number of cycles the game lasts for across the repetitions. We can also find out other information, such as the likelihood of the game ending in a particular state. We can determine a good estimate for the likelihood that the game lasts for the expected number of cycles and the likelihood that it will continue longer.

Application

First Step Analysis

Given what we know about the theory behind First Step Analysis, we can put that theory into code to estimate the number of expected cycles each game will last depending on what the starting state is. We started out with creating a list of all the possible states of starting coins; again from A_0B_0 to $A_{10}B_{10}$ which would be the keys to our key-value pairs in our state probabilities dictionary.

```
['loss', 'A00B00', 'A00B01', 'A00B02', 'A00B03', 'A00B04', 'A00B05', 'A00B06', 'A00B07', 'A00B08', 'A00B09', 'A00B1
0', 'A01B00', 'A01B01', 'A01B02', 'A01B03', 'A01B04', 'A01B05', 'A01B06', 'A01B07', 'A01B08', 'A01B09', 'A02B00', 'A0
2B01', 'A02B02', 'A02B03', 'A02B04', 'A02B05', 'A02B06', 'A02B07', 'A02B08', 'A03B00', 'A03B01', 'A03B02', 'A03B03',
'A03B04', 'A03B05', 'A03B06', 'A03B07', 'A04B00', 'A04B01', 'A04B02', 'A04B03', 'A04B04', 'A04B05', 'A04B06', 'A05B0
0', 'A05B01', 'A05B02', 'A05B03', 'A05B04', 'A05B05', 'A06B00', 'A06B01', 'A06B02', 'A06B03', 'A06B04', 'A07B00', 'A0
7B01', 'A07B02', 'A07B03', 'A08B00', 'A08B01', 'A08B02', 'A09B00', 'A09B01', 'A10B00']
```

After populating the state probabilities dictionary, where each key : value pair was a starting state: probability of that state, we moved onto creating our transition matrix P.

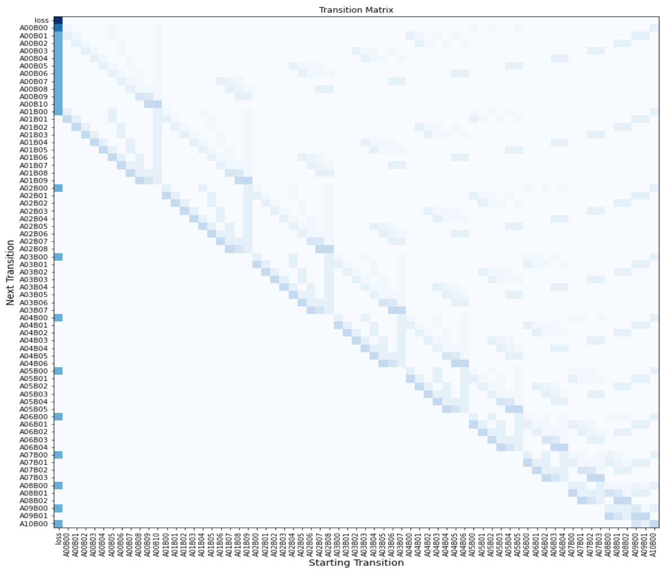


Figure 4: Transition Matrix from Starting Transition to Next Transition

Here we have 1 absorbing state and 66 transient states (total states – 1 absorbing state), and after calculating the fundamental matrix N, we return the expected number of cycles per game graphed along its starting state in Figure 5. Remembering our game rules, each player starts with 4 coins each, with 2 in the pot A_4B_4, we expected that a game would last for 17.3 cycles. Below we can see that the starting state of A_5B_5 where each player has 5 coins and none in the pot would return the highest expected number of cycles of 19.2 per game. Notice the starting states with the highest number of expected cycles start with an even split of coins between each player with zero starting coins in the pot and decreases as the pot increases coins.

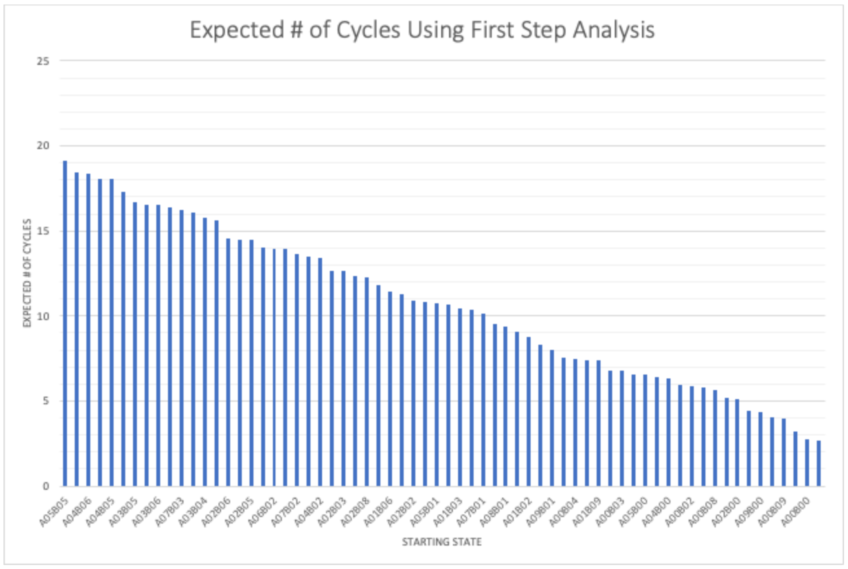
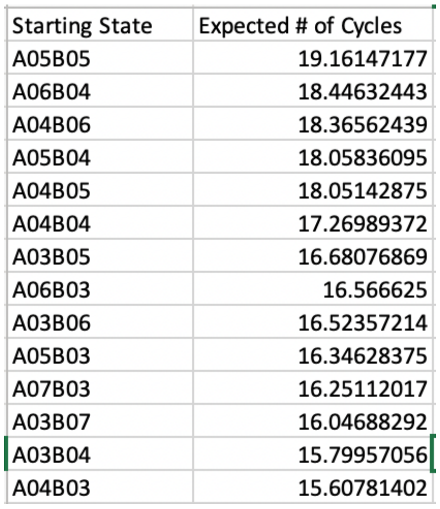


Figure 5: Plot of the Expected Number of Cycles per Game Given its Starting State



Starting State	Expected # of Cycles
A05B05	19.16147177
A06B04	18.44632443
A04B06	18.36562439
A05B04	18.05836095
A04B05	18.05142875
A04B04	17.26989372
A03B05	16.68076869
A06B03	16.566625
A03B06	16.52357214
A05B03	16.34628375
A07B03	16.25112017
A03B07	16.04688292
A03B04	15.79957056
A04B03	15.60781402

Figure 6: 15 Longest Expected # of Cycles with respective Starting States

Simulation

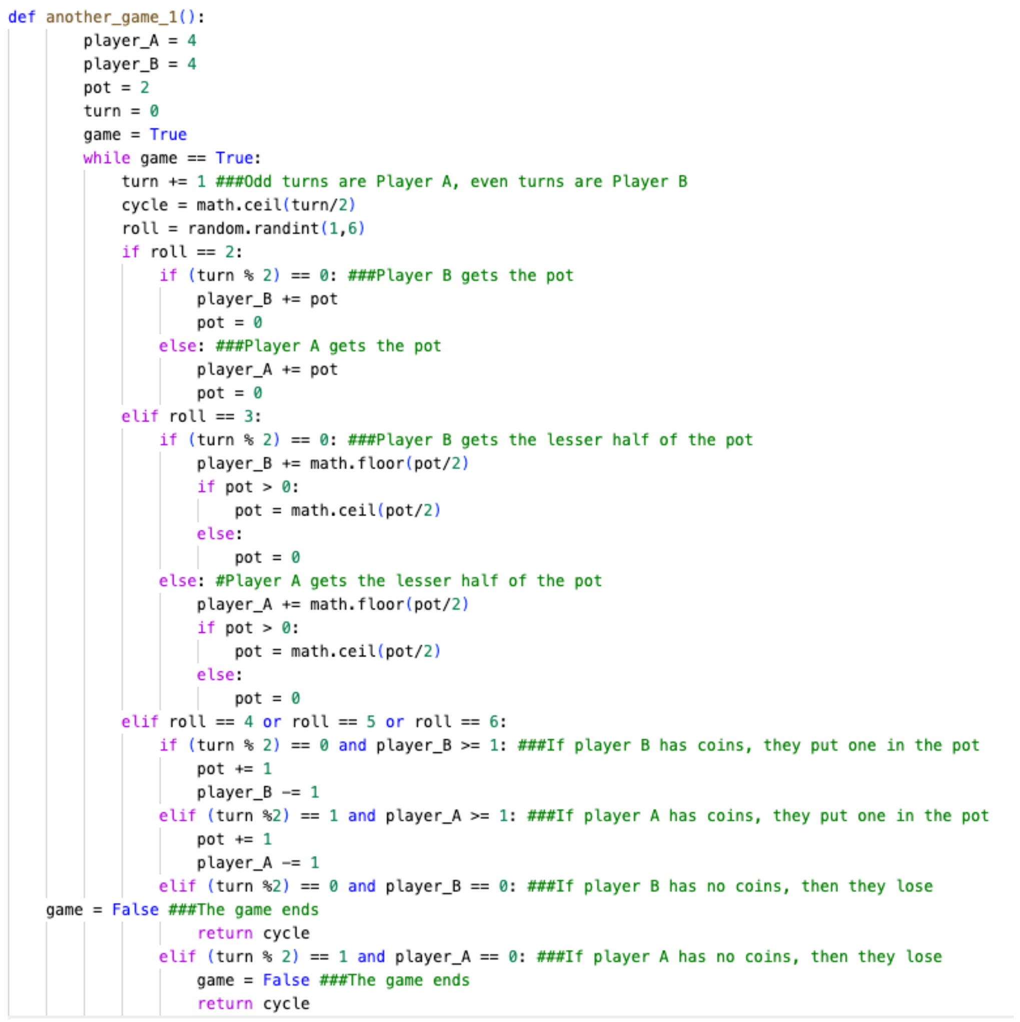
To ensure the outcomes were correct, we programmed two different functions in Python to simulate the play of the game until either player A or B loses. The only two Python packages used were the pseudo-random number generation package, "random.py", and the mathematical functions package, "math.py". The analysis of the quality of the pseudo-random numbers generated by random.py is outside of the scope of this project. Pseudo random numbers between one and six to simulate a die roll were generated by the function, "random.randint(1,6)." The ceiling function in the mathematical functions package was useful since we were dealing only in integers.

The two Python functions simulate the same game. The first version uses a "while" loop to go through each roll of the dice and counts turns. It also keeps track of the number of coins in the pot and in each player's possession. Each roll of the dice is a turn; and odd turns are player A's while even turns are player B's. Conditional "if, elif, else" statements are used to define the outcomes of a specific roll for a specific player, based on the current coin counts.

The second version also uses a while statement and sets the same initial conditions. However, it goes logically through A's turn and then B's turn for each cycle while the game is still being played, following the conditional statements that apply depending on the current coin counts in possession of each player and in the pot.

While this is a relatively simple game, two versions of the game assisted with debugging and defining key assumptions about the rules of the game. Returning the correct cycle count is important. In the description of the problem, a cycle is defined as player A and then player B completing their turns. However, if a player goes out, that counts as the final and last cycle. For instance, if player A were to lose on their sixth turn, then it would still count as a six-cycle game even though player B never rolled the dice.

Version 1 of “Another Game”



```python
def another_game_1():
    player_A = 4
    player_B = 4
    pot = 2
    turn = 0
    game = True
    while game == True:
        turn += 1 ###Odd turns are Player A, even turns are Player B
        cycle = math.ceil(turn/2)
        roll = random.randint(1,6)
        if roll == 2:
            if (turn % 2) == 0: ###Player B gets the pot
                player_B += pot
                pot = 0
            else: ###Player A gets the pot
                player_A += pot
                pot = 0
        elif roll == 3:
            if (turn % 2) == 0: ###Player B gets the lesser half of the pot
                player_B += math.floor(pot/2)
                if pot > 0:
                    pot = math.ceil(pot/2)
                else:
                    pot = 0
            else: #Player A gets the lesser half of the pot
                player_A += math.floor(pot/2)
                if pot > 0:
                    pot = math.ceil(pot/2)
                else:
                    pot = 0
        elif roll == 4 or roll == 5 or roll == 6:
            if (turn % 2) == 0 and player_B >= 1: ###If player B has coins, they put one in the pot
                pot += 1
                player_B -= 1
            elif (turn %2) == 1 and player_A >= 1: ###If player A has coins, they put one in the pot
                pot += 1
                player_A -= 1
            elif (turn %2) == 0 and player_B == 0: ###If player B has no coins, then they lose
    game = False ###The game ends
                return cycle
            elif (turn % 2) == 1 and player_A == 0: ###If player A has no coins, then they lose
                game = False ###The game ends
                return cycle
```

Version 2 of “Another Game”

```
def roll(): ###This version defines a function to use for the dice roll
    return random.randint(1, 6)

def another_game_2():
    a_coins = 4
    b_coins = 4
    pot = 2
    cycles = 0
    a_turn = 0
    b_turn = 0

    while True:
        # A's turn
        if a_coins == 0:
            a_roll = roll()
            a_turn += 1
            if a_roll == 4 or a_roll == 5 or a_roll == 6: ###Player A loses the game
                cycles = math.ceil((a_turn + b_turn)/2)
                return cycles, a_coins, b_coins
            elif a_roll == 2: ###Player A gets the pot
                a_coins += pot
                pot = 0
            elif a_roll == 3: ###Player A gets the lesser half of the pot
                a_coins += pot//2
                pot -= pot//2
            else: ###If player A rolls a 1, nothing happens
                pass
        else:
            a_roll = roll()
            a_turn += 1
            if a_roll == 1: ###If player A rolls a 1, nothing happens
                pass
            elif a_roll == 2: ###Player A gets the pot
                a_coins += pot
                pot = 0
            elif a_roll == 3: ###Player A gets the lesser half of the pot
                a_coins += pot // 2
                pot -= pot // 2
            else: ###If Player A rolls a 4, 5, or 6, they put one coin in the pot
                a_coins -= 1
                pot += 1
### Repeats the same statements above for player B’s turn
```

In order to conduct the Monte Carlo analysis, we defined the number of replications and counted the average number of cycles that the game lasted for. We also created a dictionary of the states in which the game ended and recorded the frequency at which the game ended in each state. While there are 66 total states, it is important to recognize that the game can only be lost from one of 21 states, where either A and/or B = 0. This is also demonstrated in the transition matrix.

```
r = 100000
total_cycles = 0
for i in range(r):
    total_cycles += another_game_1()
avg_cycles = total_cycles / r
print("Average number of cycles:", avg_cycles)
```

Simulation Results

Using these Python functions to simulate the game, counting the number of cycles, and running 100,000 replications, the average number of cycles is approximately 17.55. While this is slightly higher than the expected value of 17.3 from First Step Analysis, both versions of the game above produced similar results. Five different trials of 100,000 replications of Version 1 of the game produced the following average number of cycles: 17.572, 17.523, 17.531, 17.495, and 17.588. Five different trials of 100,000 replications of Version 2 of the game produced the following average number of cycles: 17.547, 17.551, 17.556, 17.554, and 17.533. Variation from First Step Analysis may be due to the method used to simulate the dice roll.

When the games are lost, the losing states that more frequently occur are those where the winning player has a higher number of coins. Roughly 20% of games are lost from the state where the winning player had 10 coins. For example, if Player A = 10 and Player B = 0, and Player B rolls a 4, 5, or 6 and loses the game, then Player A would win. These percentages decrease steadily, for example, 17% of games are lost with the winning player having 8 coins, 8% of games are lost with the winning player having 5 coins, and 2% of games are lost when the winning player has 1 coin. An additional 1.5% of games are lost when neither player has any coins. This outcome is significant, especially considering that if a player wins, they are more than 80% likely to win more than the 4 coins that they started with.

While the sample means listed above categorize the expected number of cycles that a game will last, the overall distribution of the game, shown in Figure 6, is close to an exponential distribution. Figure 7 shows an exponential distribution modeled using the scipy.stats package, with a mean of 17.55 and shifted by +5. This would be the expected histogram if the distribution of game lengths were truly exponential, based on the expected length and the fact that the game will last at least 5 cycles. The modeled data does not pass a chi-squared Goodness of Fit Test for this exponential distribution. It is slightly shorter and skewed left. From the simulation, a slightly greater percentage of games lasted to the 6th cycle than to the 5th, but from the 7th cycle and onward the percentages dropped steadily. It is rare that a game will last past 40 cycles, and the highest number of cycles observed in any simulated game was 230.

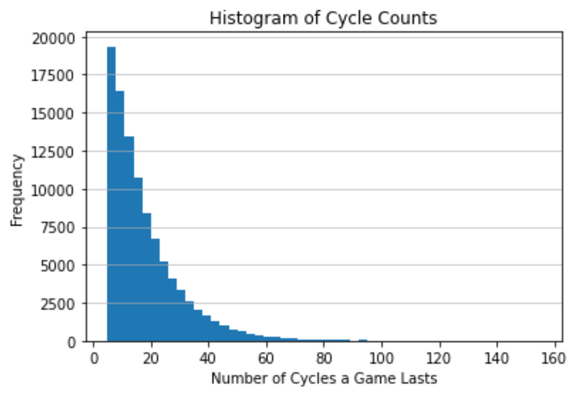


Figure 7: Distribution of Simulated Game Length

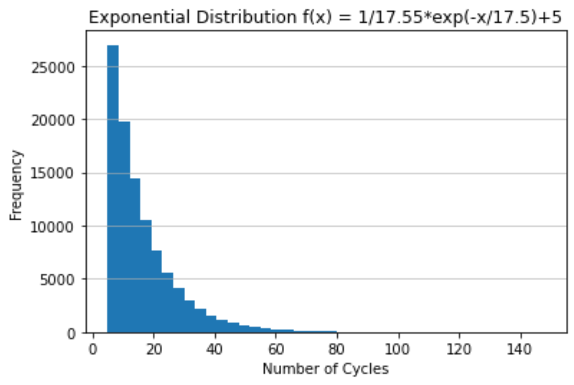


Figure 8: Exponential Distribution

Future Work

This game itself is rudimentary, as it only takes into account two players, and a one 6-sided die. When we look at our results, our game lasts about 17.5 cycles, but how long would the game last if we were using a 10-sided die, or if we had more than 2 players? The next steps in a project to improve upon this game would be to create rules for a 10-sided die, for example if we were to keep the same rules for a 1, 2, or 3 roll, but changed the rolls 4-10 to have variations with interchanging coins with the pot and with each of the players. It would be interesting to see how long the games would be expected to last, and at which starting point of coin distributions among the players and pot would return the highest number of expected cycle lengths. With the results from First Step Analysis, we could estimate that an even split of coins among all the players and none in the pot would also return the longest number of expected cycles.

Furthermore, this game is a basis for many other games that could be simulated. Much like how we took inspiration from the classic game "Chutes and Ladders", our game could be extended to many other dice games, or even poker games [9][10]. This simulation can be an important potential area of research especially for game developers that want to balance the time it takes to play a game and the players' amusement.

Conclusion

Using Markov Theory, First Step Analysis and Monte Carlo simulation, we were able to calculate the expected number of game cycles of a simple 2 player dice and coin game. Markov Theory is based on the conditional probability that the next system state/step is dependent only on the most current state and does not consider any states previous. This theory perfectly emulates our game simulation as to determine when the game would end would be again only dependent on the previous turn. We were able to use First Step Analysis to consider all the starting states and create a transition matrix that would return the probability of one state going to the next, until we finally reached the "loss" or end of game state – when a player is not able to perform an action. The expected number of game cycles a First Step Analysis returned was 17.3 cycles. Knowing this, we then created a function in Python that would simulate this game and all the expected outcomes of coin allocations upon a dice roll. Once we produced a working program, we then replicated the simulation in trials of 100,000 where we obtained a result of about 17.55 expected number of cycles. Additionally, the data generated from the simulation showed that the expected cycle counts are very close to an exponential distribution. We learned that you simulate most games of chance by understanding the fundamental probabilities and theory that create these games.

References

[1] Chen, Yen-Chi. "Discrete-Time Markov Chain – Part 1". University of Washington, Autumn 2018, Lecture.

[2] Privault, Nicolas., Understanding Markov Chains. Springer, 2018.

[3] Columbia University Irving Medical Center. “Markov Chain Monte Carlo.” *Columbia Mailman School of Public Health,* https://www.publichealth.columbia.edu/research/population-health-methods/markov-chain-monte-carlo. Accessed 17 March 2023.

[4] Queen Mary University of London. “The First Step Analysis.” https://webspace.maths.qmul.ac.uk/i.goldsheid/MAS338/FSAnalysis.pdf

[5] Tsidulko, Maya. “A Markov Chain Approach to a Game of Chance.” Montana State University, May 2015, Department of Mathematical Sciences Montana State University. https://math.montana.edu/grad_students/writing-projects/2015/15tsidulko.pdf

[6] Kemeny, John G., Snell, J.Laurie., Finite Markov Chains. D. Van Nostrand Company, Inc., 1960

[7] (2023, April 16). *Python 3.11.3 Documentation*. Python Documentation. htt[ps://docs.python.org/

[8] Biker, Helton. (2023, March 24). *Colorplot of 2D array matplotlib.* Stackoverflow. https://stackoverflow.com/questions/16492830/colorplot-of-2d-array-matplotlib

[9] Hochman, Michael. “Chutes and Ladders.” https://math.uchicago.edu/~may/REU2014/REUPapers/Hochman.pdf. Accessed 16 April 2023.

[10] Tae, Jake. “Markov Chain and Chutes and Ladders.” https://jaketae.github.io/study/markov-chain/. Accessed 16 April 2023.