Modeling Flip-cup With Markov Chains and Probabilistic Distributions

Rain Juhl[1]

CS109, Stanford University, Stanford CA, United States[1]
rjuhl@stanford.edu

Abstract. This paper explores and models the game Flipcup defined by 3 parameters: number of checkpoints, rate of player 1, and rate of player 2. It investigates the probability of winning and the expected time of a game using several different methods. Firstly, it will approximate these questions using the power of simulation. For each question, it will use Markov Chains to model the game and dive into the resulting exponential distributions, in order to derive precise answers. Additionally, it will analyze how changing any of the parameters affects someone's chances of winning the game.

Keywords: Exponential Distribution · Markov Chain · Joint Distributions · Expected Value · Flipcup

1 Introduction

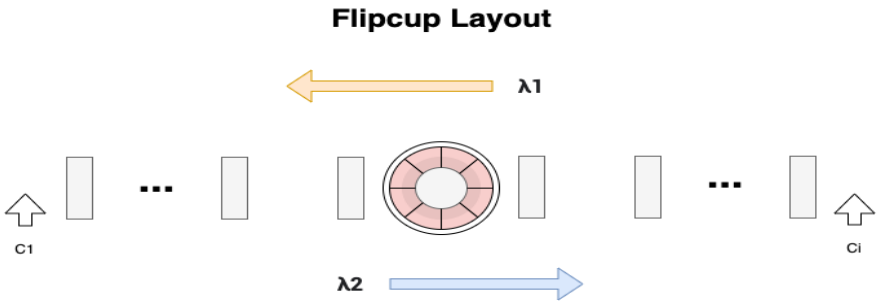
Flipcup is a common game that can be set up and played in a variety of ways. The core of the game is based around moving a cup past certain checkpoints until it reaches one of two endpoints. Two players work against each other to move the cup to the opposing endpoints. The catch is that a cup must be "flipped" before you may move it onto the next checkpoint. There are many variables to the game and different adaptions to explore. The goal of this paper is to model the game and to answer key questions. How long should the game take? This question is crucial to players because the game is often played before going to an event. Therefore, understanding the length of the game will help players plan when they should start playing. What is someone's chance of winning the game? The motivation behind this question is simple. People like to win and want to know how likely they are to win. Again, this helps one understands the stakes if there are bets or punishments for losing.

2 Game Mechanics and Defining Variables

2.1 The Basics

To analyze and modeling Flipcup, it is essential to think about it in its simplest form. Say there are i checkpoints such that i is odd and $i \geq 3$ (this means to

play $i-1$ dividers are necessary) so that we have $c_1, c_2, ..., c_i$. Next, for simplicity's sake, assume that each team is composed of one player who has a known "rate" of success for flipping a cup. Call these rates λ_1 and λ_2 for each team respectively. Below is a diagram of the setup:



To illustrate this example, run a few simulations. The code is at the end of the document. Play around with the number of checkpoints and the two rates to see how this affects the predicted time. The simulation also keeps track of which player is winning each game in order to predict the chances of winning based on a certain set of a parameters (i, λ_1, and λ_2). For the first simulation, see how the parameters effect the time and winning chances. Look at the two tables below (Number are from average over 10,000 simulations). Table (1):

Parameters Results

i	λ1	λ2	Time	P1 win %
5	1 per/min	1 per/min	1.9730min	0.5048
11	1 per/min	1 per/min	12.3925min	0.4981
23	1 per/min	1 per/min	61.4230min	0.5058

Parameters Results

i	λ1	λ2	Time	P1 win %
5	5 per/min	2 per/min	0.4755min	0.8636
11	5 per/min	2 per/min	1.6209min	0.9910
23	5 per/min	2 per/min	3.6312min	1.0000

Clearly, the number of checkpoints will affect both the time it takes to finish and the win percentage. Increasing the number of check points results in an increase of time to finish. This makes sense because there are more checkpoints to get through. An increase in checkpoints also increases the chances the player with the higher rate will win. This is also to be expected as there are more checkpoints to move through and the slower player needs more upsets to win. Finally, the data shows that the larger the difference in rate between the two players, the larger the difference in player win percentage and the shorter the game. These

observations can be defined mathematically. Specifically, how can we calculate the expected time that a game will take as well as find the chance of player 1 winning given i, λ_1, and λ_2?

2.2 Probability that Player 1 Wins

For simplicity sake, start by calculating the probability player 1 wins when $i = 5$. To do this let λ_1 be the average number of successes per minute of player one and λ_2 be the same but for player 2. Define random variables E_1 and E_2 to be the time until a successful cup flip for players 1 and 2 respectively. This can be modeled as a exponential distribution such that $E_1 = exp(\lambda_1)$ and $E_2 = exp(\lambda_2)$. Note that an exponential is consider a memoryless distribution. This means that $P(T > s+t|T > s) = P(T > t), \forall s, t \geq 0$. Ergo at any checkpoint and at anytime in the game, the probability that player 1 succeeds before player 2 is the same. Next find $P(E_2 > E_1)$ which can be re-written as $P(E_2 - E_1 > 0)$. Below is the PDF for the distribution $E_2 - E_1$ where $a_1 = \frac{1}{\lambda_1}$ and $a_2 = \frac{1}{\lambda_2}$:

$$f_{E_2-E_1}(x) = \begin{cases} \frac{e^{x/a_2}}{a_1+a_2} & x \leq 0 \\ \frac{e^{-x/a_1}}{a_1+a_2} & x > 0 \end{cases}$$

The PDF can then be integrated from 0 to infinity to find $P(E_2 - E_1 > 0)$.

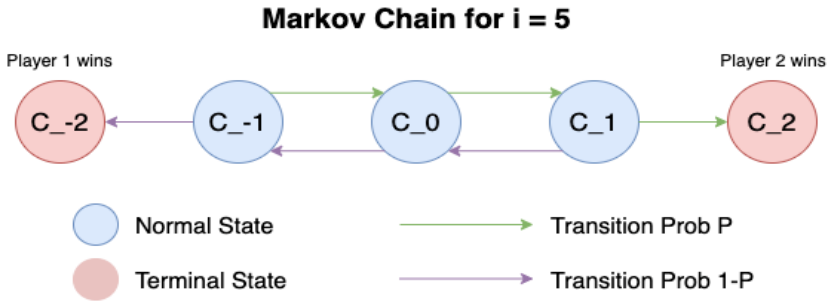
$$\int_0^\infty f_{E_2-E_1}(x)dx$$

$$= \frac{1}{a_1 + a_2} \int_0^\infty e^{-x/a_1} dx$$
$$= \frac{a_1}{a_1 + a_2}$$

Finally, substitute the lambdas back in.

$$\frac{1/\lambda_1}{1/\lambda_1 + 1/\lambda_2} = \frac{1}{1 + \frac{\lambda_1}{\lambda_2}}$$

Let this final result be q and the probability that player 2 wins be $p = 1 - q$ and let the checkpoints be labeled $c_{-k}, ..., c_k$ where $k = \frac{i-1}{2}$ and for any checkpoint C_j such that $j \neq -k$ and $j \neq k$ the chance the cup is moved to the check point $C_{j+1} = p$ and the chance it is moved to $C_{j-1} = 1 - p$. This principle can used to create a Markov Chain for $i = 5$.



Using this model, equations can be written for the probability of ending at a state given your current state!

First Consider

$$P(C_{-2}|C_0) = 1 - P(C_2|C_0) \text{ The Goal}$$

Now create a system of equations

$$1\text{: } P(C_{-2}|C_0) = p(P(C_{-2}|C_1)) + (1-p)(P(C_{-2}|C_{-1})$$
$$2\text{: } P(C_{-2}|C_1) = (1-p)P(C_{-2}|C_0)$$
$$3\text{: } P(C_{-2}|C_{-1}) = (1-p) + pP(C_{-2}|C_0)$$

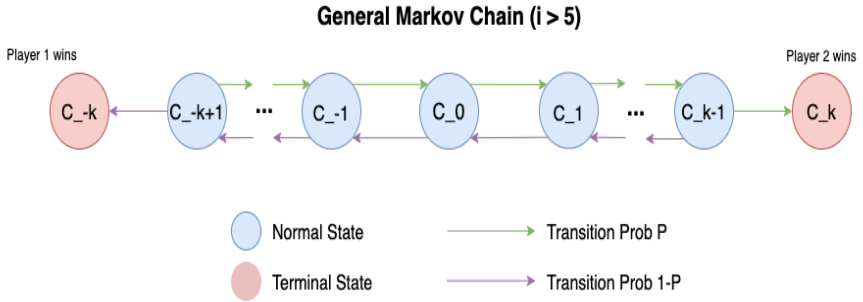
Next, solve the system of equations for $P(C_{-2}|C_0)$.

$$\begin{aligned} P(C_{-2}|C_0) &= p(P(C_{-2}|C_1)) + (1-p)(P(C_{-2}|C_{-1}) \\ &= p(1-p)P(C_{-2}|C_0) + (1-p)((1-p) + pP(C_{-2}|C_0)) \\ &= (p-p^2)P(C_{-2}|C_0) + (1-p)^2 + (p-p^2)P(C_{-2}|C_0) \end{aligned}$$

$$P(C_{-2}|C_0)(1-2(p-p^2)) = (1-p)^2$$

$$P(C_{-2}|C_0) = \frac{(1-p)^2}{1-2(p-p^2)}$$

Now the probability of winning is in terms of λ_1 and λ_2 for $i = 5$ lets generalize it to all values of i. To do so, consider the Markov matrix below:



This can be solved similarly as before by recognizing the recursive relationship between states.

The base cases:

$$P(C_{-k}|C_{-k}) = 1$$
$$P(C_{-k}|C_k) = 0$$

Otherwise:

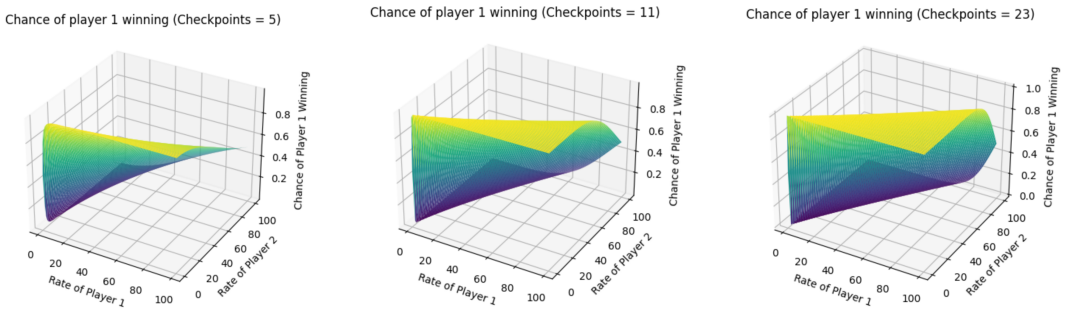
$$P(C_{-k}|C_j) = pP(C_{-k}|C_{j+1}) + (1-p)(P(C_{-k}|C_{j-1}))$$

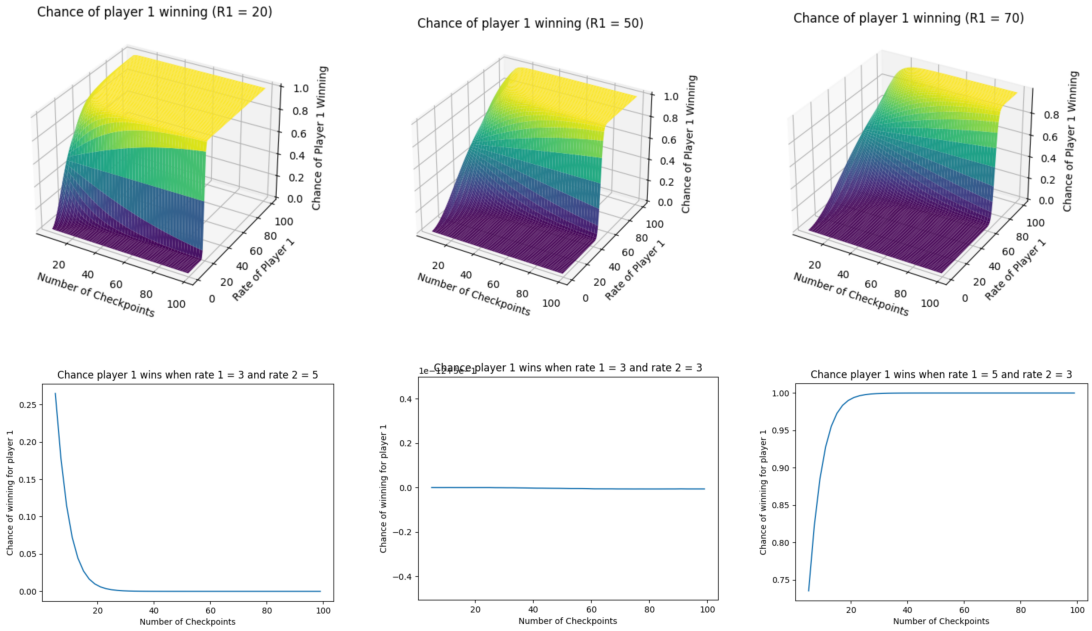
Such that $-k < j < k$

Although this strategy works to solve the general Markov Chain linear algebra offers tricks that will help to solve it computationally. The key is to form a matrix of transition probabilities. Then order the rows so that terminal states (c_k and c_{-k}) come first. To get an idea of what this would look like, consider the matrix created for $i = 5$ such that rows are $[c_{-2}, c_{-1}, c_0, c_1, c_2]$ and the columns are $[c_{-2}, c_{-1}, c_0, c_1, c_2]$.

$$M = \begin{bmatrix} 1 & 0 & 1-p & 0 & 0 \\ 0 & 1 & 0 & 0 & p \\ 0 & 0 & 0 & 1-p & 0 \\ 0 & 0 & p & 0 & 1-p \\ 0 & 0 & 0 & p & 0 \end{bmatrix}$$

Engel algorithm for absorbing Markov chains can be applied to this matrix to solve for the chance of ending up at either c_{-2} or c_2 given the starting checkpoints. See a generic implementation of the algorithm in the code attached at the end. Next, consider the following Graph some results and observe what happens as each parameter is changed.





Observe the first row of graphs. It appears that as rates of each player get closer the chance of player 1 winning approaches 0.5. Additionally, as 1 rate increases and the other decreases, the increasing player's chances go to 1 while the decreasing player's chances go to 0. Additionally, as i increases the switch from one player having a greater chance of winning to the other player happens more quickly. The second row of graphs shows that as the rate of player 1 increased compared to the fix rate of player 2, the chance that player 1 wins also increases. Additionally, these graphs show that as i increases, the change in one player's chances becoming better than the other's happens at much quicker. This effect is apparent again in row 3. Furthermore, row 3 highlights what is intuitively obvious: with equal rates both players have an equal chance of winning no matter the number of checkpoints.

2.3 Expected Time of Game

Next, find the $E[T|C = c_0]$ where T is time of game and C is a RV for the current checkpoint $(c_{-k}, ..., c_k \in C)$ with the same parameters as before $(i, \lambda_1$, and λ_2 where i is the number of checkpoints and λ_1/λ_2 are the rates of player 1 and player 2 respectively. Again, borrow the General Markov Model that was used to calculate the probability of player 1 winning. Based on this, find $E[T|C = c_0]$. This again writes a recursive relation.

The base cases:

$$E[T|C = c_{-k}] = 0$$
$$E[T|C = c_k] = 0$$

Otherwise:

$$E[T|C = c_j] = p(E[T|C = c_{j+1}] + E[e_1|e_1 < e2]) \\ + (1-p)(E[T|C = c_{j-1}] + E[e_2|e_2 < e1])$$

Such that $-k < j < k$ and $e_1 \sim exp(\lambda_1)$ and $e_2 \sim exp(\lambda_2)$.

This is similar to the recursive relationship for finding winning chances except for one crucial difference: this time a $E[e_1|e_1 < e2]$ term or $E[e_2|e_2 < e1]$ term appears each time a state changes. Note that replacing these terms with $E[e_1]$ and $E[e_2]$ gives a reasonable approximation for the answer but not the true expected time because when traveling from one state to another, the player that flipped the cup faster is known. This means that expected time is based on the fact that either $e_2 > e_1$ or $e_1 > e_2$. This is why $E[e_1|e_1 < e2]$ and $E[e_2|e_2 < e1]$ is used instead. This can be checked numerically with the simulation code. To move forward in this problem $E[e_1|e_1 < e2]$ and $E[e_2|e_2 < e1]$ must be solved. First solve $E[e_1|e_1 < e2]$.

$$E[e_1|e_1 < e_2] = E[e_1|e_2 - e_1 > 0]$$

Let $Y = e_2 - e_1$ then

$$\begin{aligned} E[e_1|e_2 - e_1 > 0] &= \int_0^\infty x \cdot f_{E_1|Y}(E_1 = x|Y = y)dx \\ &= \int_0^\infty x \cdot \frac{f_{e_1,e_2}(x,y)}{f_Y(y)}dx \\ &= \int_0^\infty x \cdot \frac{\lambda_1 e^{-\lambda_1 x} \cdot \lambda_2 e^{-\lambda_2(x+y)}}{\frac{\lambda_1 \lambda 2 e^{-y\lambda_2}}{\lambda_1+\lambda_2}}dx \end{aligned}$$

The numerator was formulated from the PDF's of e_1 and e_2 multiplied together to get the joint. Since the PDF of exponential is well known, this is straightforward. The denominator came from the PDF of $e_2 - e_1$ which was used earlier; however, this time it is terms of λ_1 and λ_2 instead of $a_1 = \frac{1}{\lambda_1}$ and $a_1 = \frac{1}{\lambda_1}$.

Simplify further:

$$
\begin{aligned}
&= \int_0^\infty x \cdot \frac{(\lambda_1+\lambda_2)e^{-\lambda_1 x+\lambda_2 x-\lambda_2 y}}{e^{-\lambda_2 y}}dx \\
&= \int_0^\infty x \cdot (\lambda_1+\lambda_2)e^{-(\lambda_1+\lambda_2)x}dx \\
&= (\lambda_1+\lambda_2)\int_0^\infty x \cdot e^{-(\lambda_1+\lambda_2)x}dx \\
&= (\lambda_1+\lambda_2)(\frac{-((\lambda_1+\lambda_2)+1)e^{-(\lambda_1+\lambda_2)x}}{(\lambda_1+\lambda_2)^2}|_0^\infty) \\
&= \frac{\lambda_1+\lambda_2}{(\lambda_1+\lambda_2)^2} \\
&= \frac{1}{\lambda_1+\lambda_2}
\end{aligned}
$$

Therefore $E[e_1|e_1 < e_2] = \frac{1}{\lambda_1+\lambda_2}$. The same calculation can be done for $E[e_2|e_2 < e_1]$ and doing so will yield that $E[e_1|e_1 < e_2] = E[e_2|e_2 < e_1] = \frac{1}{\lambda_1+\lambda_2}$. Using these values and the recursive formula, a system of equations can be obtained. The resulting system is most easily solved by a computer in matrix form. The resulting matrices follow a general pattern that is demonstrated below:

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & \cdots & 0 \\
-1+p & 1 & -p & 0 & \cdots & 0 \\
0 & -1+p & 1 & -p & \cdots & 0 \\
0 & 0 & -1+p & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & \cdots & -p \\
0 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
E[T|c_{-k}] \\ E[T|c_{-k+1}] \\ E[T|c_{-k+2}] \\ E[T|c_{-k+3}] \\ \vdots \\ E[T|c_{k-1}] \\ E[T|c_k]
\end{bmatrix}
=
\begin{bmatrix}
0 \\
pE[e_1|e_1 < e_2] + (1-p)E[e_2|e_2 < e_1] \\
pE[e_1|e_1 < e_2] + (1-p)E[e_2|e_2 < e_1] \\
pE[e_1|e_1 < e_2] + (1-p)E[e_2|e_2 < e_1] \\
\vdots \\
pE[e_1|e_1 < e_2] + (1-p)E[e_2|e_2 < e_1] \\
0
\end{bmatrix}
$$

Where p is the probability that player 1 wins, and $E[T|c_j]$ where $-k \leq j \leq k$ are the variables being solved for. Once solved, just pick out $E[T|C = c_0]$. Now check the simulations with the correct computed probability. Below is a table of expected times with the same inputs as table 1. Table (2):

Parameters E[T | C=c0]

i	λ1	λ2	Time
5	1 per/min	1 per/min	2min
11	1 per/min	1 per/min	12.5min
23	1 per/min	1 per/min	60.5min

Parameters E[T | C=c0]

i	λ1	λ2	Time
5	5 per/min	2 per/min	0.4828min
11	5 per/min	2 per/min	1.6320min
23	5 per/min	2 per/min	3.6664min

Based on the results, simulating the game 10,000 times and taking the average of those times gets close to the actual expected time. This is an efficacious sanity check for both the simulation and the precise calculation. Using either, one can figure out how long one should expect to play Flipcup given the number of checkpoints and both players' rate.

3 Conclusion

Overall, there are two solutions for how to find the probability of winning given the number of checkpoints and both players' rates and two solutions for guessing the time a game is going to take given the same parameters. To answer the first problem either a simulation can be run keeping track of the winners to find a reasonable approximate of the true probability or the exact probability can be calculated with the program derived from modeling the problem as a Markov Chain. The second problem can be solved by again running a simulation which will give an reasonable approximation of the expected time or by solving the system of equations derived again from the Markov Chain Model.

In the future, this work can be expanded to model Flipcup with two teams instead of two different players. Additionally, I would like to work on a program to visualize how the parameters affect expected time.

4 References

[1] Kennedy, T. (n.d.). Conditional expectation. Retrieved November 27, 2021, from http://www.math.wm.edu/~leemis/chart/UDR/PDFs/ExponentialLaplace.pdf.

[2] Lemis, L. (n.d.). Exponential Laplace. Retrieved November 27, 2021, from http://www.math.wm.edu/~leemis/chart/UDR/PDFs/ExponentialLaplace.pdf.

4.1 Special Thanks

Special Thanks to William Xi and Kim Swensen for peer reviewing the paper!

5 Code

5.1 Sampling Code

```
import numpy as np


num_sims = 10000


def get_exp_sample(beta):
    return np.random.exponential(scale=beta)


def run_simple_exp(i, r1, r2):
    start = 0
    ends = (i - 1) / 2
    assert ends.is_integer() and ends > 0

    s1 = get_exp_sample(1 / r1)
    s2 = get_exp_sample(1 / r2)

    while start != ends and start != ends * -1:
        if s2 > s1:
            start -= 1
            if start != ends * -1:
                s1 += get_exp_sample(b1)
            else:
                break

        elif s1 > s2:
            start += 1
            if start != ends:
                s2 += get_exp_sample(b2)
            else:
                break
        else:
            s1 += get_exp_sample(s1)
            s2 += get_exp_sample(s2)

    player_win = 1 if start < 0 else 2
    time = s1 if player_win == 1 else s2

    return player_win, time
```

```
def printOutcome(cp, r1, r2):
    times = []
    p1_wins = 0
    p2_wins = 0
    for i in range(num_sims):
        winner, time = run_simple_exp(cp, r1, r2)
        if winner == 1:
            p1_wins += 1
        else:
            p2_wins += 1
        times.append(time)
    print(sum(times) / len(times))
    print(p1_wins / (p1_wins + p2_wins))
```

5.2 Winning Probability and Graphing

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

def findWinningChances(c, r1, r2):
    p = 1 / (1 + (r1 / r2))
    ends = (c - 1) / 2
    assert ends.is_integer() and ends > 0

    trans_matrix = np.zeros((c - 2, c - 2))
    s_matrix = np.zeros((2, c-2))
    s_matrix[0][0] = 1 - p
    s_matrix[1][c - 3] = p
    for i in range(c - 2):
        trans_matrix[i][i] = 1
        if i < c - 3:
            trans_matrix[i + 1][i] = -p
            trans_matrix[i][i + 1] = -1 + p
    inverse_matrix = np.linalg.inv(trans_matrix)
    ans_matrix = np.matmul(s_matrix, inverse_matrix)
    p1_win_prob = ans_matrix[0][int((c - 3) / 2)]
    return p1_win_prob

def graph3D(ranges, const_dim, c=None, r1=None, r2=None):
    if const_dim == 0:
        r1vals = [i for i in range(1, ranges[0])]
        r2vals = [i for i in range(1, ranges[1])]
        X, Y = np.meshgrid(r1vals, r2vals)
        Z = np.zeros(X.shape)
```

```
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i][j] = findWinningChances(c, X[i][j], Y[i][j])
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                    cmap='viridis', edgecolor='none')
    ax.set_title(f'Chance of player 1 winning (Checkpoints = {c})')
    ax.set_xlabel('Rate of Player 1')
    ax.set_ylabel('Rate of Player 2')
    ax.set_zlabel('Chance of Player 1 Winning')
    plt.show()
elif const_dim == 1:
    c_vals = [i for i in range(5, ranges[0], 2)]
    r2_vals = [i for i in range(1, ranges[1])]
    X, Y = np.meshgrid(c_vals, r2_vals)
    Z = np.zeros(X.shape)
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i][j] = findWinningChances(X[i][j], r1, Y[i][j])
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                    cmap='viridis', edgecolor='none')
    ax.set_title(f'Chance of player 1 winning (R1 = {r1})')
    ax.set_xlabel('Number of Checkpoints')
    ax.set_ylabel('Rate of Player 2')
    ax.set_zlabel('Chance of Player 1 Winning')
    plt.show()
elif const_dim == 2:
    c_vals = [i for i in range(5, ranges[0], 2)]
    r1_vals = [i for i in range(1, ranges[1])]
    X, Y = np.meshgrid(c_vals, r1_vals)
    Z = np.zeros(X.shape)
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Z[i][j] = findWinningChances(X[i][j], Y[i][j], r2)
    ax = plt.axes(projection='3d')
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                    cmap='viridis', edgecolor='none')
    ax.set_title(f'Chance of player 1 winning (R1 = {r2})')
    ax.set_xlabel('Number of Checkpoints')
    ax.set_ylabel('Rate of Player 1')
    ax.set_zlabel('Chance of Player 1 Winning')
    plt.show()
else:
    print(f"Error const_dim = {const_dim} not a # between 1 and 2 inclusive"
```

```
def graph2D(r1, r2, ran):
    c_vals = [i for i in range(5, ran, 2)]
    results = []
    for num in c_vals:
        results.append(findWinningChances(num, r1, r2))
    plt.plot(c_vals, results)
    plt.title(f'Chance player 1 wins when rate 1 = {r1} and rate 2 = {r2}')
    plt.xlabel('Number of Checkpoints')
    plt.ylabel('Chance of winning for player 1')
    plt.show()
```

5.3 Expected Time

```
import numpy as np


def findExpTime(c, r1, r2):
    p = 1 / (1 + (r1 / r2)) #Prob of success
    e1 = 1 / (r1 + r2)  #E[e_1 | e_1 < e_2] where e_1 ~ exp(r1)
    e2 = 1 / (r1 + r2) #E[e_2 | e_2 < e_1] where e_2 ~ exp(r2)

    start = (c - 1) / 2
    assert start.is_integer() and start > 0

    var_matrix = np.zeros((c, c))
    intercept_matrix = np.zeros(c)

    for i in range(c):
        var_matrix[i][i] = 1
        if i < c - 2:
            var_matrix[i + 1][i] = -1 + p
            var_matrix[i + 1][i + 2] = -p

    for i in range(1, c - 1):
        intercept_matrix[i] = (p * e1) + ((1 - p) * e2)

    ans_arr = np.linalg.solve(var_matrix, intercept_matrix)
    return ans_arr[int(start)]
```