Probability Report:
Determining the Probability of a Theoretical Game

Author: Srihari Vishnu
Instructor: Carly Funk
Course: MDM4UI
Date: May 1st, 2020

Introduction

The purpose of the report is to design a theoretical game, and determine the probabilities of a certain player winning. The instructions of the game are outlined below.

Materials:

- 3 6-sided die (Numbered from 1-6)
- 30cm radius, circular, bordered playing area (so die cannot bounce outside of playing area)
- A 30cm Ruler
- A random number generator (between 1-30)

Goal:

The last player standing wins (in other words, the other player has run out of lives).

Setup:

Before starting the game, designate one of the dice as the *central die*, and the other two as *roller dice*.

Roll the *central die* and then record its value. The number shown on the central die is the number of lives each player begins with. Then place the *central die* at the **center** of the container; this marks the central point of the board and it serves as a marker for the duration of the game. While playing, you must toss the other two dice 30cm directly above its position. The central dice will not be touched/rolled for the remainder of the game.

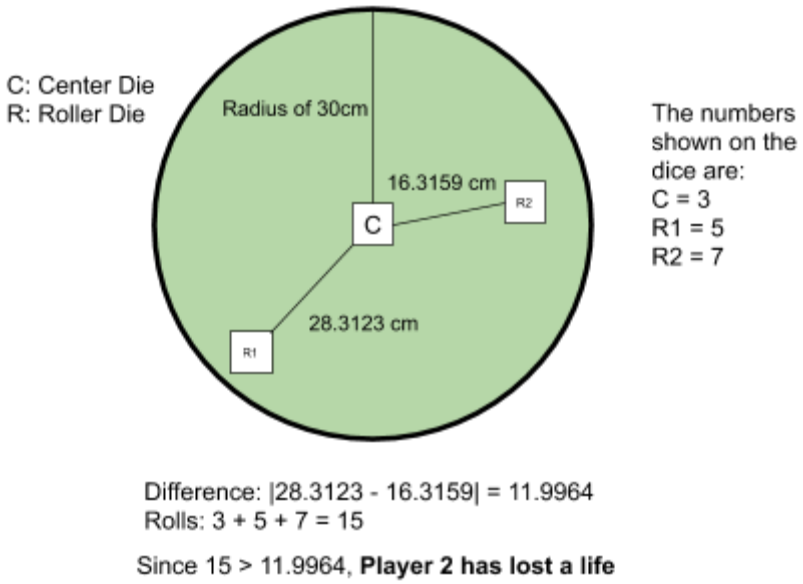
GamePlay:

Each round of the game consists of the following:

1. Each player drops one of the roller dice from directly above the central die.
2. After each player drops their roller die, measure the distance in centimetres from the center of each roller die to the central die using the ruler as precise as possible, and then record the difference between the two measurements.
3. Record the sum of the value of the faces shown on all three dice (including the central die as well).
4. If the **difference** of the measurements is **greater** than the **value** shown on all three dice, **player 1 loses a life**.
5. Otherwise, if the **difference** of the measurements is **less** than the **value** shown on all three dice, **player 2 loses a life**.
6. If the sum of the rolls is the exact same, measure the lengths again to another decimal place.
7. Start the next round with each picking up the roller dice and tossing them again; make sure the central die is still in the central position of the playing area

8. Continue playing rounds until one of the players runs out of lives, at which point the other player wins the game.

Diagram 1: A Sample Round



NOTE: Please note that although the distances are used throughout this text, for purposes of a consistent probability, it is assumed that the dice have equal probability of landing anywhere within the playing area. To simulate this, instead of measuring the distances, please use a random real number generator to generate a number between 0 and 30.

Theoretical Probability

It is important to note that it is sufficient to calculate the probability of Player B winning, since probability of Player A winning is just the complement of the probability of Player B. To calculate the theoretical probability of this game, the calculations can be divided into two components for a certain round, given the value of the central die.

1. Probability of obtaining a certain sum for the other two die (whose value would be added to the central die)
2. Probability that the value of the measured distances is greater than the value of the dice.

The three above components calculate the probability that player B wins a certain round of a game. Since rounds are independent from each other, the game tree has equal probability for each player to win or lose at each round, an important fact used later on in calculating the probability of the overall game.

Calculating Probability of Each Component

Calculating the probability of Component 1 can be calculated by counting all the possible sums of 2 dice rolling and dividing it by 36 (Sample Space as there are 6x6 outcomes). The possible sums of rolling two dice are:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 *Possible sums of rolling 2 dice*

Tallying up the results, each sum obtained the following probabilities respectively:

S = [1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, 1/36]

Since Component is more involved, so the following notation is used for brevity:

- M - represents the difference in the measured distances between the two roller dice
- α - Sum of the values of the three dice (including the central die)
- X - Distance of one of the roller dice to the central die
- Y - Distance of the other roller dice to the central die
- c - Value of the central die

Component 3 involves calculating the probability that M is greater than α. When dealing with uniform probability distributions, to find the probability that a certain condition is met, the continuous variables are expressed as ranges on each of the axes of a graph. X and Y are both bounded by the edge of the playing area; this implies a possible distance of being 0 cm away from the center of the board, to being at the very edge of the board 30 cm away from the central die.

$$X \in \{X \in R,\ 0 \leq X \leq 30\} \qquad Y \in \{Y \in R,\ 0 \leq Y \leq 30\} \qquad \textit{Domains}$$

Since we are interested in the absolute difference of the two values, the relationship of the two variables is defined in the following inequality:

$$|X - Y| > \alpha \qquad \textit{Inequality 1}$$

Figure 1 shows *Inequality 1* and the domains of the two continuous variables for when α = 10.

Figure 1: Graph of |X-Y| > 10

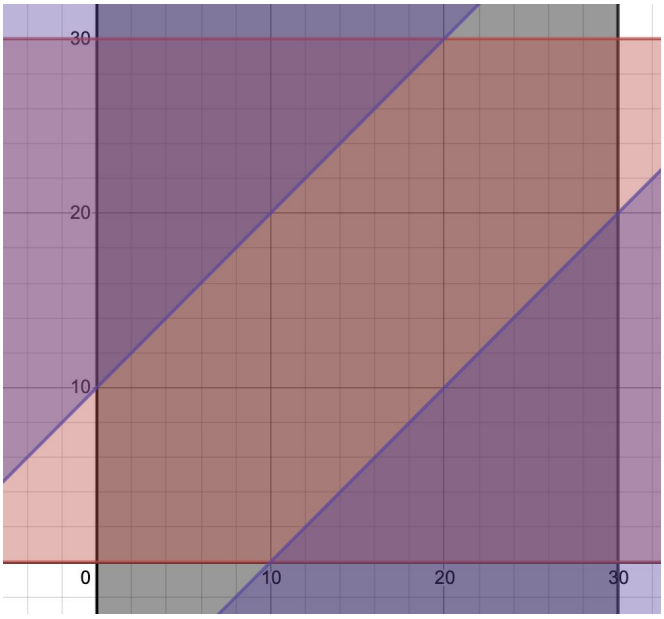


Figure 1: A plot of |X-Y| > α, where α is 10. The purple region is the significant area (bounded by x = 0, x = 30, y = 0, y = 30.

The probability of the two continuous variables meeting the conditions listed above is calculated by finding the ratio of the desired area to the figure as a whole. Since $|X - Y| > \alpha$ always produces two symmetrical triangles in the range $3 \leq \alpha \leq 18$, the probability can be calculated as the area of the two triangles divided by 900 (which is 30^2, or the area of the entire square).

$$P(M > \alpha) = \frac{(30-\alpha)^2}{900}$$

Equation 2

Putting The Components together

By leveraging components 1, 2 and 3, it is possible to calculate the probability of player B winning given the value of the central die with the following equation, where $P_{2D}(x)$ is the probability of rolling a sum of x with the two roller dice.

$$\sum_{\alpha = 2+c}^{12+c} P_{2D}(\alpha - c) \cdot \frac{(30-\alpha)^2}{900}$$

Equation 3

Equation 3 sums over all the possible sums of dice rolls (since its value will always be in the interval $2+c \leq S \leq 12+c$ (since the interval of two dice rolling is $2 \leq 2D \leq 12$). Similarly, the probability of player A winning can be calculated by taking the complement of the area component. The result is shown in *Equation 4* below.

$$\sum_{\alpha = 2+c}^{12+c} P_{2D}(\alpha - c) \cdot \left(1 - \frac{(30-\alpha)^2}{900}\right) \qquad \textit{Equation 4}$$

Calculating the Probability of the Entire Game

Calculating the probability that player B wins the game is quite straightforward when $c = 1$ (the center die is equal to 1).

Possible Game Sequences (**where A = Player A loses a life and B = Player B loses a life**):

- ➔ *A*
- ➔ *B*

With one life, the maximum amount of rounds that the game can last is one. Therefore, calculating the probability that B wins the game is as simple as using *Equation 3*.

$$\sum_{\alpha = 2+1}^{12+1} P_{2D}\,(\alpha - 1) \cdot \frac{(30-\alpha)^2}{900} = 0.5442592592592593$$

Therefore, $P(B\ wins \mid c = 1) = 0.544259...$

However, as the game tree grows, it becomes more difficult to calculate the probabilities.

Possible Game Sequences when c = 2:

- ➔ *AA*
- ➔ *BB*
- ➔ *ABA*
- ➔ *ABB*
- ➔ *BAA*
- ➔ *BAB*

Since each round is independent, the probability that player A loses a life in the first round is no different than player A losing a life in a later round. Therefore, the probability of a sequence can be calculated by multiplying the probabilities of a player losing a point each round by each other (using *Equation 3* and *Equation 4).*

To calculate the entire game tree for the 6 possible values of the central die, a Python program was written to compute the probabilities of the different games. The algorithm is described in Appendix A.

After going through all possible game trees, the following results were obtained:

Player A wins with probability: 0.6555751685064313
Player B wins with probability: 0.3444248314935687

With an error of *5.551115123125783e-16* due to rounding and floating-point error.

Experimental Probability

To quickly play the game, a simulation was written in Python to play the game multiple times. The results were quickly converging to the expected values calculated in the theoretical probability section for both the number of times each player won, and the number of turns each game would take. A sample output of the probabilities is shown below.

Player A won 131186 times for a total probability of 0.65593
Player B won 68814 times for a total probability of 0.34407
NUMBER OF TURN PROBABILITIES:
[0.0, 0.16678, 0.08388, 0.12582, 0.08768, 0.1235, 0.10016, 0.11918, 0.07645, 0.06765, 0.027215, 0.021685]

The ith turn probabilities represents the probability of the game ending in the i turns. For example, there is 0% chance of the game ending after 0 turns, but there is 8.388% chance of the game ending after 2 turns.

A comparison between the theoretical calculated probabilities vs the experimental probabilities is shown graphically in the figures below. The experimental probabilities were calculated after 1 000 000 rounds of playing.

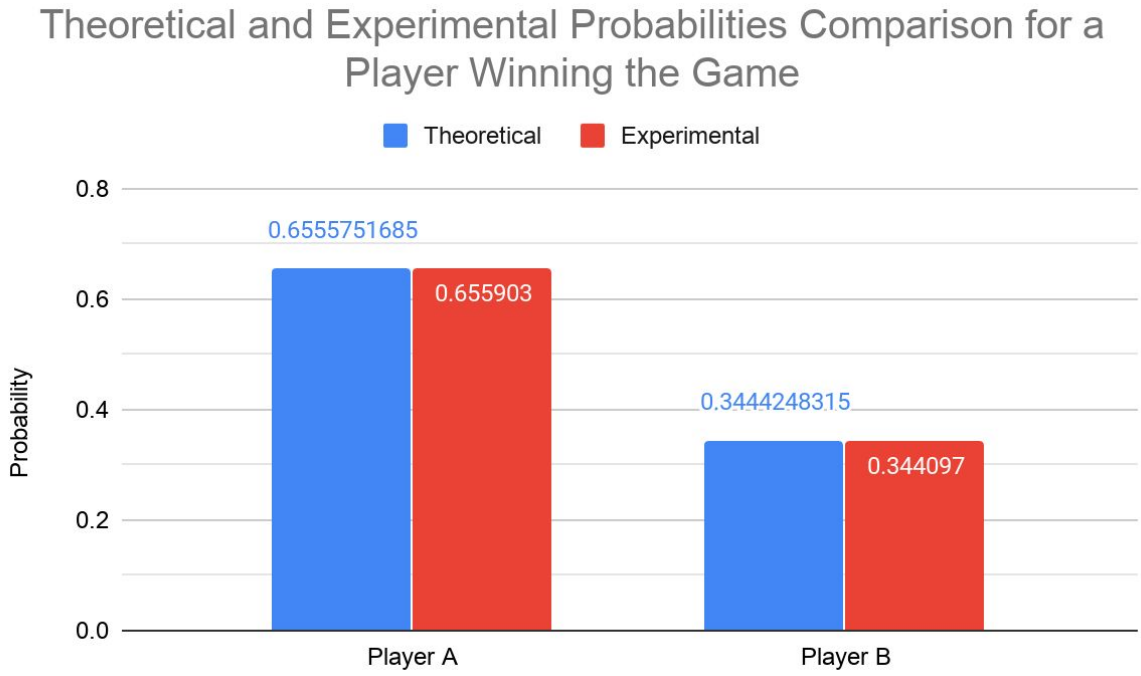


Figure 2: A comparison of the probability that a certain player wins the game. The probabilities are very close, with only minor discrepancy.

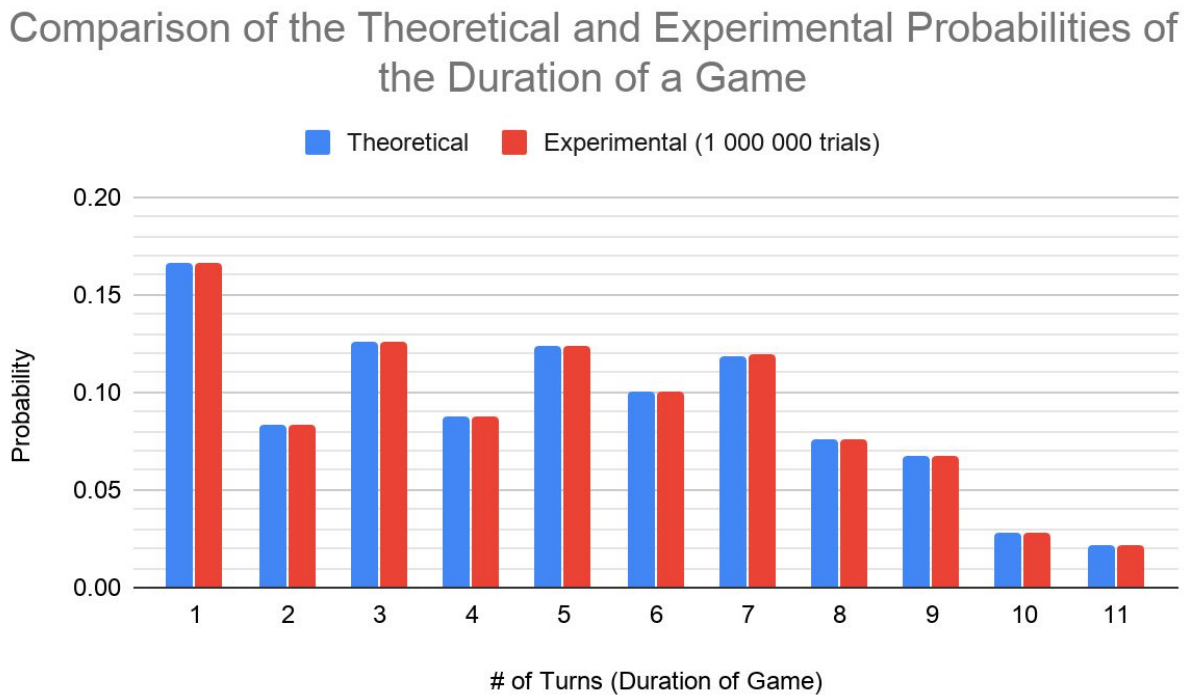


Figure 3: The probabilities of a game ending after a certain amount of turns closely matches the theoretical probability as well.

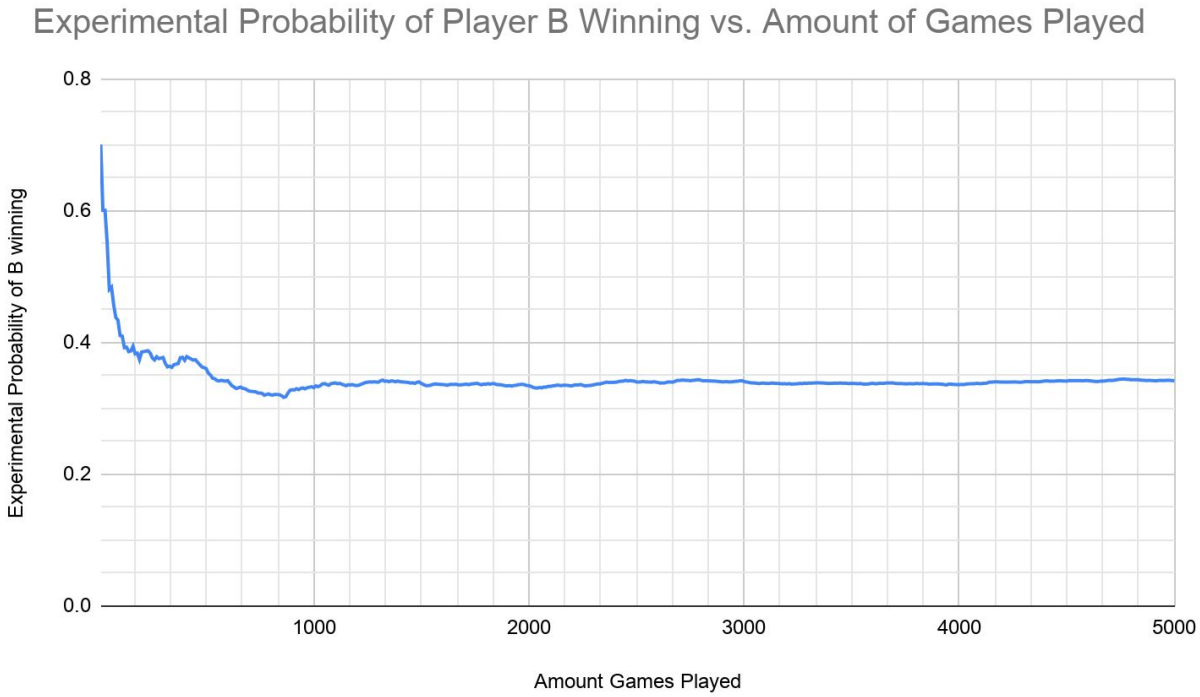


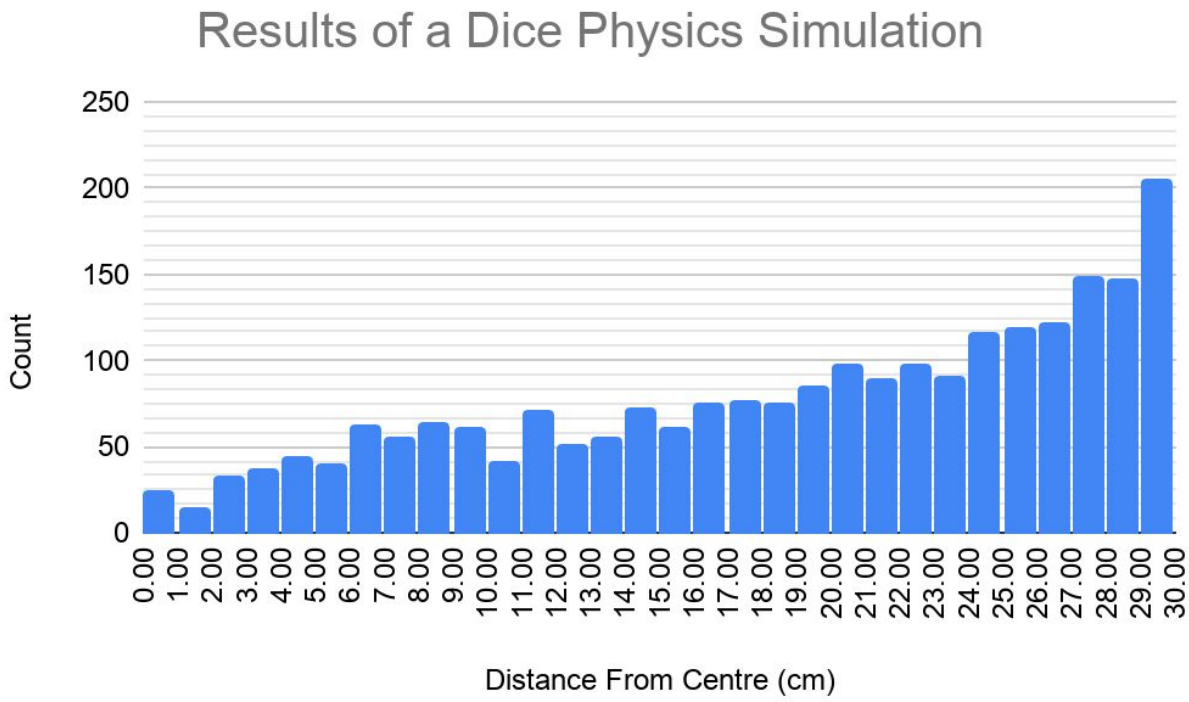
Figure 4: This graph shows the calculated probability of Player B winning as the simulation progresses. The probabilities were calculated every 10 games.

Conclusion

The results derived from the experiment closely match the expected probabilities. *Figure 2, Figure 3, and Figure 4* show the relationship between the expected values, and the actual values. Possible sources of the small margin of error came from floating-point precision error, which details that it is impossible to perfectly store an irrational number on a computer (there must be some rounding). If this game were to be played in real-life, without the use of a simulator, the error margin would be significantly higher, since there are many variables that disrupt the motion of the dice. That is why, for the purpose of this report, a random number generator was used instead, to allow for a uniform probability density.

Extension

Although much more difficult, it is possible to generate an approximate probability function based on trials of throwing dice. However, each environment would produce dramatically different results.The following demonstrates the results of physics simulation, and shows that the probability of the distance of the die to be farther away from the center much larger.



The trend seems to be roughly linear, however, there is a lot of variance. The simulation was run only 2000 times, due to the amount of time it took to run per toss of the die, but could be run more times to normalize the results.

Works Cited

"Continuous Probability Densities." Dartmouth Edu, www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/Chapter2.pdf.

Appendix

Appendix A: Code for Calculating the Probability of Each Player Winning

```
#Counter to count the total number of possible games
counter = 0
#The total probability (Should equal 1 once divided by 6. The difference is the error
margin due to rounding)
totalprob = 0.0
# Probabilities of sums of 2,3,4,5,6,7,8,10,11,12
P = [1/36, 2/36, 3/36, 4/36, 5/36, 6/36, 5/36, 4/36, 3/36, 2/36, 1/36]
#An array to count the number of times a game ended after x amount of turns
turns = [0 for x in range(13)]

# Calulating the probability that B wins with the formula
# c is the value of the central die
def PBWin(c):
   s = 0.0
   for a in range(2+c, 12+c+1):
       s += P[a-c-1-1]*(30-a)**2/900
   return s

#Calculating the probability that A wins with the complement of the formula for B
def PAWin(c):
   s = 0.0
   for a in range(2+c, 12+c+1):
       s += P[a-c-1-1]*(1-(30-a)**2/900)
   return s
"""
Since the game tree is a tree, you can recurse on each node. Return when the lives of
a player equals 0. Return the probability of B winning in that state.
If Player A has 0 lives left, increase the counter for number of possible games by 1,
increase the total probability by the amount to get to that point in the tree.
Also, increase the count of the total turns it took to get there. Same for Player B.
If neither player has 0 lives, recurse to the next level of the tree and add the
probability of B winning from each branch.

NOTE: prev stores the specific order of players who have lost lives. Therefore adding
A onto the path means finding the probability that B wins.
```

```
"""
def recurse(prev, startlives, livesa, livesb, prob):
   global counter
   global totalprob
   if (livesa == 0):
       # print ("B wins with: " + " ".join(prev) + " and with probability " +
str(prob))
       counter += 1
       totalprob += prob
       turns[len(prev)]+=prob/6
       return prob #Return the probability of the current branch since the B won this
branch
   elif (livesb == 0):
       # print ("A wins with: " + " ".join(prev) + " and with probability " +
str(prob))
       counter += 1
       totalprob += prob
       turns[len(prev)]+=prob/6
       return 0
   else:
       return recurse(prev + ["A"], startlives, livesa-1, livesb,
prob*PBWin(startlives)) + recurse(prev + ["B"], startlives, livesa, livesb-1,
prob*PAWin(startlives))

#Start the process of recursing by considering the two branches (Player A losing
first, or Player B losing first)
def start(lives):
   #P(B winning | center dice is lives)
   return recurse(["A"], lives, lives-1, lives, PBWin(lives)) + recurse(["B"], lives,
lives, lives-1, PAWin(lives))

p = []
probability = 0.0
#Go through all the possible centre dice rolls (1-6)
for x in range(1,7):
   #P(A|rolled x as centre die)
   probability += (1/6)*start(x)

print (counter)
```

```
print (1-totalprob/6)
print ("A wins with probability: ", 1-probability)
print ("B wins with probability: ", probability)

print (turns, sum(turns))
```

Appendix B: Simulation Code

```
import random

awin = 0
bwin = 0
turncount = [0 for x in range(12)]

#Roll a singular 6-sided die
def roll_die():
   return float(random.randint(1,6))

#Obtain a random real-number measurement ranging from 0 to 30
def get_random_value():
   return 30*random.random()



print ("\n\n__________STARTING PROCESS__________\n\n")

#Play the game this many times
EPOCHS = 5001
for x in range(EPOCHS):
   central_dice_val = roll_die()
   a_lives = central_dice_val
   b_lives = central_dice_val
   turns = 0
   while (a_lives > 0 and b_lives > 0):
       roller_dice_sum = roll_die() + roll_die()
       random_measured_value = abs(get_random_value() - get_random_value())
       turns += 1
       if (random_measured_value > roller_dice_sum + central_dice_val):
```

```
            # print ("Player A Lost a Life since", random_measured_value, "is greater
than", roller_dice_sum + central_dice_val)
            a_lives -= 1
        else:
            # print ("Player B Lost a Life since", random_measured_value, "is less
than", roller_dice_sum + central_dice_val)
            b_lives -= 1

    if (a_lives == 0):
        # print ("B won with " + str(b_lives) +" lives left")
        bwin += 1

    if (b_lives == 0):
        # print ("A won with " + str(a_lives) +" lives left")
        awin += 1

    turncount[turns] += 1

    if (x % 10 == 0 and x != 0):
        print (bwin/x)

print ("\n\n__________FINISHED PROCESS__________\n\n")

print ("Player A won", awin, "times for a total probability of", awin/EPOCHS)
print ("Player B won", bwin, "times for a total probability of", bwin/EPOCHS)

print ("NUMBER OF TURN PROBABILITIES: ")
for x in list(map(lambda x: x/EPOCHS, turncount)):
    print (x)
```

Appendix C: Rough Calculations

https://docs.google.com/spreadsheets/d/1OxdgMQgYVwZwbzQvsnOWFgFb0ZWq2PYje3lDurEGFqs/edit?usp=sharing