Application of Binomial Distribution and Conditional Probability with Decision Tree to Calculate the Win Rate of Clash in the Game "Limbus Company"

Rayhan Ridhar Rahman - 13522160[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522160@std.stei.itb.ac.id

***Abstract*—*Limbus Company* is the third game developed by Project Moon, serving as the continuation of *Library of Ruina*. One of the most important mechanics in the battle system is clash. The clash system involves flipping coins and comparing the total power of two clashing skills. The coins being used are unfair coins, thus binomial distribution is being used to calculate the possibility of getting a specific number of heads and then finishing the touch with conditional probability. While the decision tree is being used to draw a timeline of the clash until win.**

***Keywords*— Binomial Distribution, Combinatorics, Conditional Probability, Decision Tree, Unfair Coin**

I. Introduction

The growth of technology brought a lot to the table in multiple aspects of human life. Entertainment is one of the most affected aspects. Video games are part of entertainment. Ever since the creation of the first ever video game in October of 1958 to recent years, it accompanies many adolescents to their adulthood. Video games are interactive digital media, a fusion of miraculous results between technology and imagination. Instead of strictly playing according to the script, the control rests in our hands, so that we may be immersed in the world that the creator created. Video games let us escape from this fleeting reality for a moment, to take a breather from stress and so on.

Limbus Company is a Turn-Based Strategy RPG Gacha Game developed and published by South Korean indie video game studio Project Moon, serving as a sequel to Library of Ruina and Lobotomy Corporation. This game is also their debut game on mobile platform.

Once again, we enter the world which Project Moon created called, "The City." It depicts a vastly different world than ours where humanity has already exhausted all of earth's natural resources and turns toward an alternative called "singularities" that advanced their technology.

In Limbus company, we take control of a character called Dante, the executive manager of Limbus Company Bus division. A person with a clock for a head. Dante is accompanied with twelve other 'sinners', each based of a character of famous literature like Ishmael from Moby-Dick by Herman Melville. As the clock hands in Dante's head rewind, the 'time' of each sinner also go backwards, thus they can be revived after dying. Limbus itself is a fusion of two words, *limbo*, and *bus*. It's a bus that traverses through the limbo to find salvation for its passengers.

Fig. 1.1. The Twelve Sinners of Limbus Company (Source: in-game screenshot)

The story follows the thirteen sinners travel around The City using a bus to retrieve branches of gold called "the golden boughs," the essence of energy which resulted by former L Corp. from the previous games. Each sinner may be distant from the others, caring only for themselves. But through the struggle in retrieving the boughs, they form a bond akin to a family, in an unforgiving world where cruelty is normality. They may find themselves lost sometimes, but Dante and the other sinners are there to direct them towards the right path.

The purpose of this paper is to try and calculate the more accurate win rate of clash mechanics, as the clash prediction in the game currently only calculates the win rate of the first instance of clash and not what happened after. More will be explained in the theoretical basis of clash.

II. Theoretical Basis

A. Combinatorics

Combinatorics is a branch of mathematics that focuses on the research of countable discrete problems. Combinatorics has many implementations in various fields, one of which is game theory. Combinatorics can help to count the number of ways an event could occur by every outcome. There are two basic counting principles in combinatorics, which are sum rule and product rule.

1) Sum Rule

If a task can be approached n ways **or** m ways, where none of the set of n ways is the same as the set of m ways, then there is (n + m) ways to do the task.

2) Product Rule
If a task can be divided to subset n ways **and** m ways, where none of the set of n ways is the same as the set of m ways, then there is (n × m) ways to do the task.

Beside those counting principle, combinatorics also has other principles derived from the basic counting principle such as:

1) Permutation
Permutation is a special case from product rule, it's the total arrangement of how to create a row of r things from a set with n things where n ≥ r. The formula is as follows:

$$P(n,r) = \frac{n!}{(n-r)!} \qquad (1)$$

With n being the number of things inside the set and r being the length of arrangement

2) Combination
Combination is a special case of permutation which counts different arrangements with same things as one and the same. The formula is as follows:

$$C(n,r) = \frac{n!}{(n-r)!r!} \qquad (2)$$

The logic behind dividing it with $r!$ is to divide it by the total arrangement of $P(r,r)$.

B. Binomial Distribution

In the event we need to calculate how many probabilities there are to get 1 heads from 3 coins, we might calculate it like this using the product rule and sum rule like this:

$$P(TTH) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$$
$$P(THT) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$$
$$P(HTT) = \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$$

So, the probability of getting 1 heads after flipping 3 coins is by using the sum rule of those three, resulting in $\frac{3}{8}$ chance. If the repetitions are being done even more, it's going to be much more tedious than it needs to be.

Binomial distribution is a way in statistics to simplify the tedious probability associated with independent and repeated trials. In a binomial distribution the probabilities of interest are those of receiving a certain number of successes **r**, in **n** independent trials each having only two possible outcomes and the same probability, **p**, of success. With this, we can calculate the probability of getting **r** amounts of heads in **n** amounts of unfair coins. The formula is as follows:

$$P(r|n,p) = \binom{n}{r} p^r (1-p)^{n-r} \qquad (3)$$

$$\binom{n}{r} = C(n,r) \qquad (4)$$

Equation (3) is the formula of binomial distribution, equation (4) shows the formula for binomial constant which is the combination of **r** things from **n** things. The result of binomial distribution can be drawn into a graph. For example:

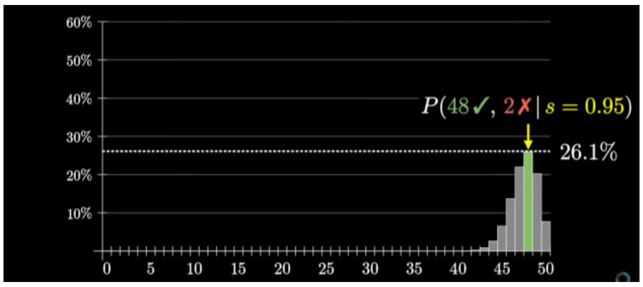


Fig. 2.0.1. Binomial Distribution from 3Blue1Brown (Source: https://www.youtube.com/watch?v=8idr1WZ1A7Q)

Since the coin mechanic of the game in question is mostly that of unfair coin, binomial distribution is the perfect way to count the probability of getting specific amounts of head. Combined with the principle of combinatorics, we can get the probability of winning in a clash.

C. Decision Tree

In many situations one needs to make a series of decisions. This leads naturally to a structure called a "decision tree." Decision trees provide a geometrical framework for organizing the decisions. Although it's not necessary to view all the theory in trees, they are powerful tools to truly understand the key concepts. It allows us to view the collection of all decisions in an organized manner.

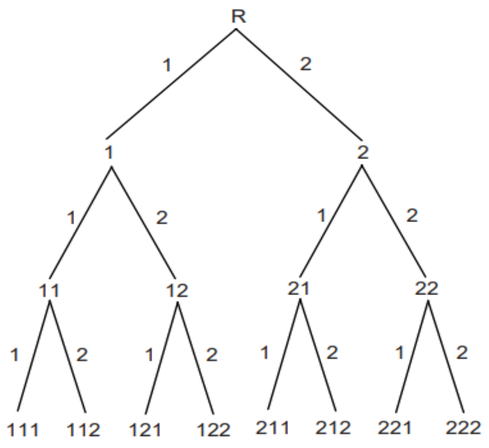


Fig. 2.2 Decision Tree Example for function in 2^3 (Source: [1])

Decision trees are a part of a more general subject in discrete mathematics called "graph theory". The decision tree of the previous example illustrates the various ways of generating a function in 2^3 sequentially. Instead of labelling the edges with 1 or 2, a decision tree should have used a terminology that fits the function.

In this terminology, a vertex v represents the partial function constructed so far, when the vertex v has been reached by starting at the root, following the edges to v, and making the decisions that label the edges on that unique path from the root to v.

In the shown example, each node has some leading out edges. Those edges are labeled with all possible continuation of the decision, a partial function. Meanwhile nodes that don't have leading out edges are called leaves, the results of the function.

D. Recursion

An object may be called recursive if the object is mentioned inside their own terminology. As with any recursive situation, when an algorithm refers to itself, it must be with "simpler" parameters so that it eventually reaches one of the "simplest" cases.

Recursive function can be divided into two parts, the basis and the recurrent. Basis is the part where the function is explicitly defined and is the part that stops recursive pattern. Meanwhile recurrent defined the function to its own terminology.

Recursion is used to traverse a binary tree structure. That's because every sample that is a branch, is also a tree. These are what are called subtrees. So, the basis happens when the tree is empty. The recurrent happens when the tree is not empty.

Here is a picture of recursion of the tree structure:

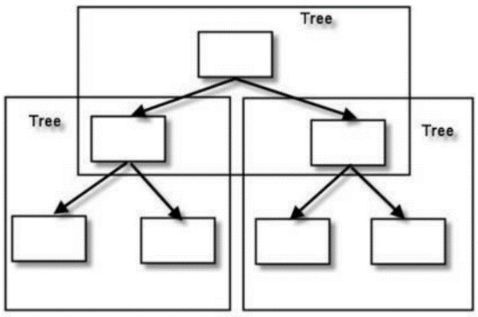


Fig. 2.3 Recursion of binary tree (Source: [2])

E. Conditional Probability

Conditional probability is the probability of a probability. Essentially, it's the probability of something occurring, given another probability already occurred. For clarification, let U be a sample space with probability function P. If A ⊆ U and B ⊆ U are subsets of U then the conditional probability of B given A, denoted by P(B|A) is as following:

$$P(B|A) = \begin{cases} \frac{P(A\cap B)}{P(A)}, & if\ P(A) \neq 0 \\ undefined, & if\ P(A) = 0 \end{cases} \qquad (5)$$

Let's interpret that in another way. In n times trials, we want to count the probability of A occurring, assume the count is a. Now with P(B|A), we would like to know the probability of B occurring within the confinement where A occurred, assume that the count is c. Hence, the probability of B occurs given that A occurred is $\frac{c}{a} = \frac{c}{n} \times \frac{n}{a}$ and as n increases, the result becomes approximately P(B|A).

Below are properties of conditional distribution:

1) $P(B|U) = P(B)$ and $P(B|A) = P(A \cap B | A)$.
2) A and B are independent events if and only if $P(B|A) = P(B)$.
3) $P(A|B) = P(B|A)\ P(A) / P(B)$ (This is what called as Bayes' Theorem).
4) $P(A1 \cap \cdots \cap An) = P(A1)\ P(A2 | A1)\ P(A3 | A1 \cap A2) \cdots P(An | A1 \cap \cdots \cap An-1)$.

F. Conditional Probability and Decision Tree

Now, after familiarizing ourselves with decision tree and conditional probability, let us try to combine them together. In the world of statistics, conditional probability is used to calculate the probability of something happening as the sample increases. Using the decision tree, it's going to be much easier to visualize the process.

Suppose that a research team is developing to see if a person has a disease called distortion. It's known that 1 out of 500 people can get distorted. To test the accuracy, they've asked many volunteers to participate. Out of 87 distorted people, the test passes with flying color. While testing it with normal people, the test got false-positive 3% of the times. Now, if they release it to the masses, what is the probability of a person that has the disease?

Assume W is the event of people get distorted and T is the event of the test came out positive. This information can be written as:

$$P(W) = \frac{1}{500} \quad P(T|W) = 1 \quad P(T|W^C) = \frac{3}{100}$$

Using the Bayes' theorem, we can tell that:

$$\mathrm{P(D|T)} = \frac{\mathrm{P(T|W)P(W)}}{\mathrm{P(T)}}$$

We haven't found what P(T) is yet, we can calculate that by:

$$\begin{aligned} P(T) &= P(T \cap W) + P(T \cap W^c) \\ &= P(T|W)P(W) + P(T|W^c)P(W^c) \\ &= 1 \times 0.002 + 0.03 \times (1 - 0.002) \approx 0.032 \end{aligned}$$

Thus P(W|T) ≈ 1 × 0.002/0.032 ≈ 6%. Which means that if we were diagnosed to have distortion, there is a 6% chance of having the disease, which is really misleading.

We can picture the case using a decision tree. We start out with the sample space U as the root. The first decision partition is going to be whether the person tested for having distortion or not. Which are $P(W)$ and $P(W^c)$ as the edges of U. Each edge is labeled with the form $P(B|A)$with A being the parent and B as the child. Consequently, it can be shown like this:

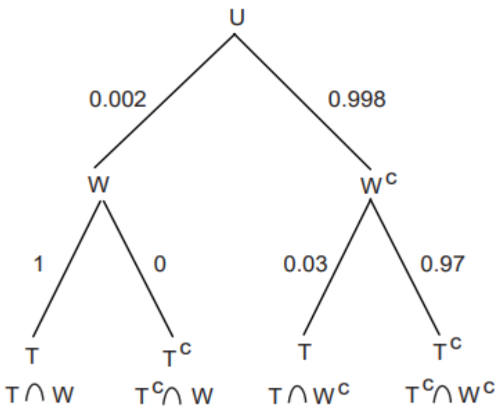


Fig. 2.4 Decision tree for conditional probability (Source: [1])

To calculate $P(T)$ this way is to find leaves with T in the end and use the sum rule.

$$P(T) = 0.002 \times 1 + 0.998 \times 0.03 \approx 0.032$$

We have the same probabilities of being tested positive. Hence, we still got P(W|T) ≈ 1 × 0.002/0.032 ≈ 6% as the answer.

G. Limbus Company: Character

As the executive manager of Limbus Company, we have the right to command our sinners, although they may not heed our words sometimes. To ease our decision-making, there are some combat UI, helping you identify the character status.

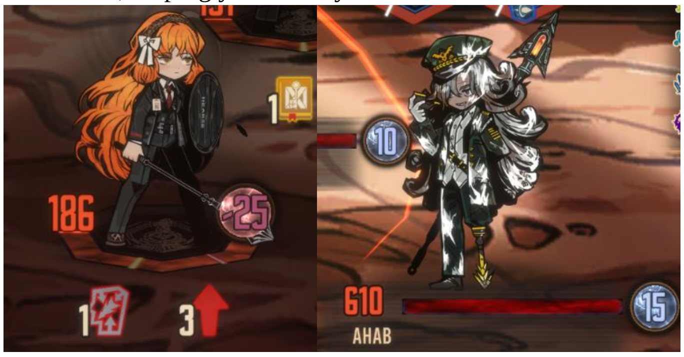


Fig. 2.5 Ishmael (sinner), Ahab (enemy)
(Source: in-game screenshot)

Below a character's sprite, there is a health bar. To the left of the health bar, is the character current health. To the right side of the health bar, exists a ball containing numbers for the character's sanity, ranging from -45 to 45. Finally, below the health bar are status effects, can change certain stats whether in making the sinner's stats better (buff) or making the sinner's stats worse (debuff).

Through seeing the mirror of the world, the sinners have multiple "identities" of what they could have been. These identities may have different stats and skills compared to the others. So, it's up to the executive manager to call for the right identity for varying conditions.

[Kurokumo Clan Wakashu] Ryoshu
[LCB Sinner] Ryoshu
[LCCB Assistant Manager] Ryoshu
[R.B. Chef de Cuisine] Ryoshu
[Seven Association South Section 6] Ryoshu
[W Corp. L3 Cleanup Agent] Ryoshu

Fig. 2.6 Ryoshu's identities
(Source: https://www.prydwen.gg/limbus-company/identities)

H. Limbus Company: Skills

As mentioned before, the sinners have different identities, each with different sets of skills. There are 3 skill 1s which are low-power setup skills, 2 skill 2s which are mid-power and usually are the main skill, and finally 1 skill 3 which is the unique skill. There is also defensive skill which can replace any skill.

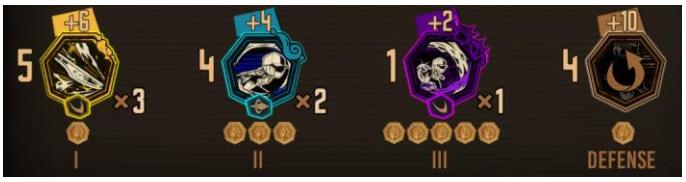


Fig. 2.7 [W Corp. L3 Cleanup Agent] Don Quixote skills
(Source: in-game screenshot)

Each skill has different base power (number beside the icon), coin power (number above the icon), and coin amount. These skills are going to be used in many of the mechanics in the game, not limited to battle. Each time a skill is being used all coins of the skill are flipped, if a coin lands on head, the power of the skill increases by the coin power. The probability of getting heads is influenced by the character's sanity by the following formula.

$$HeadOdds = \frac{50 + Sanity}{100} \times 100\% \qquad (6)$$

Skill at least has 1 coin and currently the highest number of coins possible is 5. There are also powerful skills that every sinner uniquely has, disregarding of which identity you have chosen.

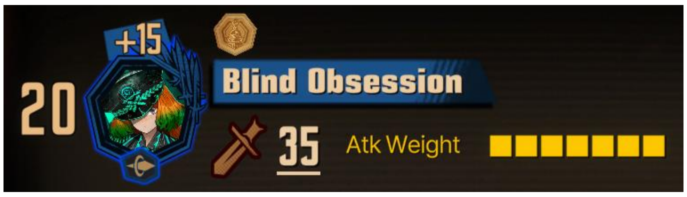


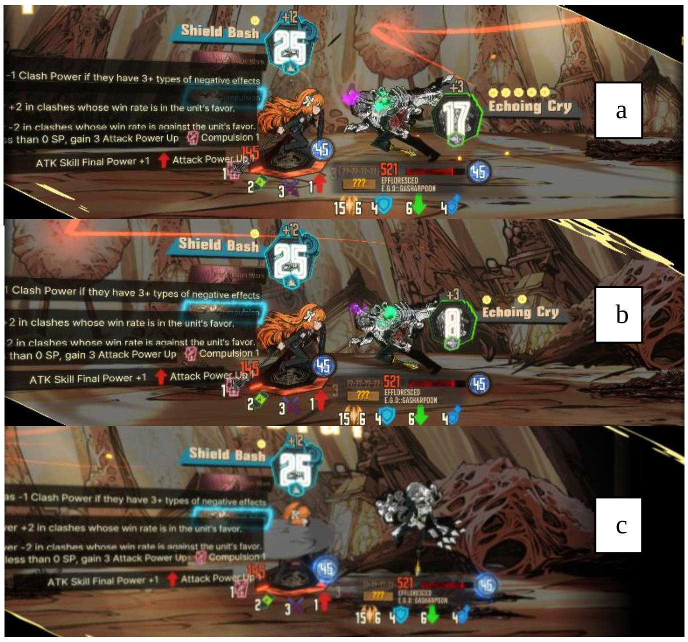
Fig. 2.8 Ishmael's Blind Obsession E.G.O stats
(Source: in-game screenshot)

This skill is called E.G.O skills and costs some resources and the sinner's sanity, as the sinner borrow the power of "abnormality". The 'Atk Weight' stat refers to how many targets it attacks.

I. Limbus Company: Clash Mechanics

A *clash* happens when two offensive skills become the target of each other. Clash begins with both sides flipping coins for their skill power. If the power of one skill overpowers the other, the losing side loses one coin. Then both sides flip their remaining coins again until one side runs out of coins.

If both skills flipped the same power, it's a clash draw, both sides don't lose coin. It should be noted that the maximum number of clashes can happen is 99. If reaching a stalemate after 99 clashes, both sides lose all coins resulting in no



-1 Clash Power if they have 3+ types of negative effects
+2 in clashes whose win rate is in the unit's favor.
-2 in clashes whose win rate is against the unit's favor.
s than 0 SP, gain 3 Attack Power Up Compulsion 1
ATK Skill Final Power +1 Attack Power Up

damage done to either side. Below is the example of clash:

Fig. 2.9 Ishmael and Ahab clash. (a) start. (b) continuation. (c) end.
(Source: in-game screenshot)

There are multiple things that affect the outcome of clashes, those are:

1. Skill's offensive level
   Besides base power and coin power, each skill also has an offense level, as seen in figure 6 beside the sword icon. The skill with higher offense level gain extra clash power by 1 for each 3 differences in offense level.
2. Clash-oriented status effects
   There are a lot of status effects that affect clashing directly or indirectly. Effects like clash power up/down, coin power up/down, base power up/down, and so on affect the skill's power directly. Meanwhile, there's also offense level up/down that change the offense level, affecting it indirectly.
3. Skill-specific conditions
   Some skills can get more power by reaching specific conditions. The best example of this is skill used by "charge identities" which get more power by consuming more charge.
4. Passives/support effects
   We can't bring every sinner to battle, only up to 5 sinners (some stages may vary) may be selected. But each sinner (on-field or off-field) can give support effects just by being in the bus. There are also passives granted by E.G.O skills that affect clash.

J. Limbus Company: Built-in Clash Prediction

In the game, there is a built-in win rate possibility, with different label for each range of probability. There are 5 labels, which are, *hopeless* within 0-10%, *struggling* within the range of 10-40%, *neutral* within the range of 40-60%, *favored* within the range of 60-90%, and *dominating* within the range of 90-100%.

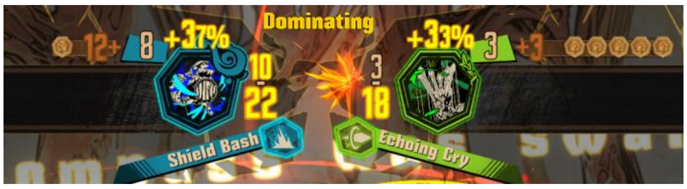


Fig. 2.10 Ishmael and Ahab clash from Fig. 2.8 prediction (Source: in-game screenshot)

But there is inaccuracies in the way the game calculates win rate. It only calculates the first instance clash without accounting for clashes after it. This results in a deceptively higher/lower win rate, especially when dealing with high base power with negative coin power skills.

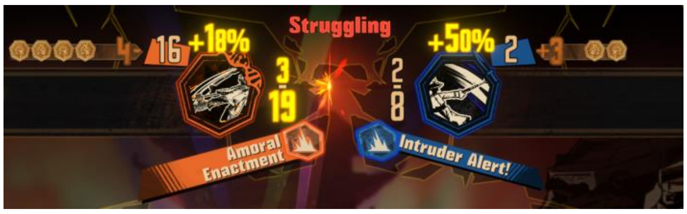


Fig. 2.11 Sinclair's negative coin clash prediction (Source: in-game screenshot)

If Amoral Enactment has only 1 coin left, with a probability of 95% getting heads, since the coin power is negative, the skill has its minimum value with heads, at minimum the power is $16 + (-4 \times 1) + \left\lfloor \frac{\Delta offenselevel}{3} \right\rfloor$ or 15 in this case, which is higher than the opposing skill maximum possible power of 8. The prediction should've been *dominating* (100%).

III. Methodology

A. Limitation

This paper will now try to calculate the win rate of the clash mechanics. With various factors affecting clash result in many ways, the method used won't account most status effects as they usually change sinner or skill power, so it can be calculated separately before calculating the win rate. The method used will also ignore draw chances, as it's infinitesimal to get to 99 clashes without some setup. We also will limit the skill that a character has to only one.

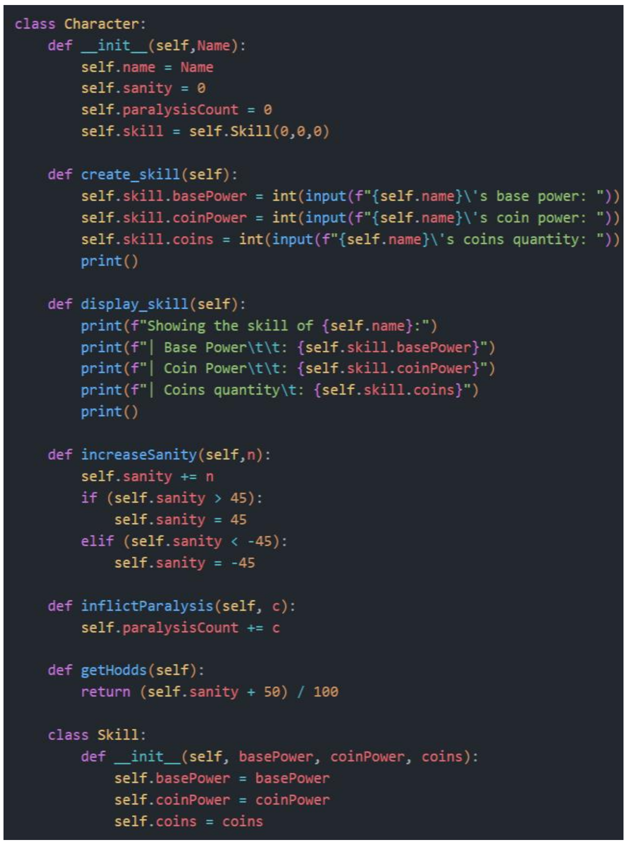
With these assumptions applied, the result might not be accurate, but it would at least be an introductory of combinatorics and probability to some.

B. Tools

The tools that will be used for this simulation include:

1. Python 3.10
2. Visual code studio
3. diagrams.net

C. Character Class

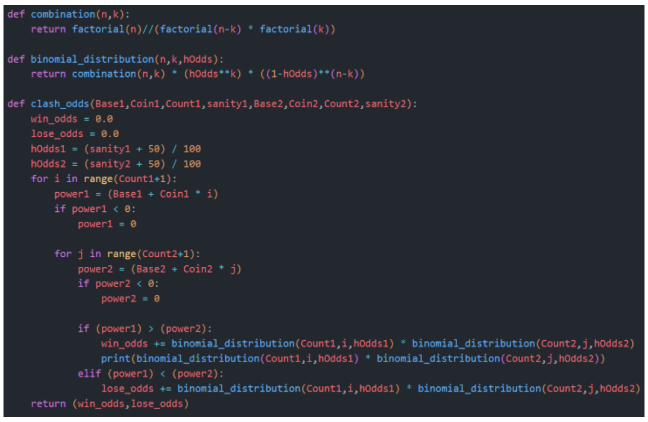


```python
class Character:
    def __init__(self,Name):
        self.name = Name
        self.sanity = 0
        self.paralysisCount = 0
        self.skill = self.Skill(0,0,0)

    def create_skill(self):
        self.skill.basePower = int(input(f"{self.name}\'s base power: "))
        self.skill.coinPower = int(input(f"{self.name}\'s coin power: "))
        self.skill.coins = int(input(f"{self.name}\'s coins quantity: "))
        print()

    def display_skill(self):
        print(f"Showing the skill of {self.name}:")
        print(f"| Base Power\t\t: {self.skill.basePower}")
        print(f"| Coin Power\t\t: {self.skill.coinPower}")
        print(f"| Coins quantity\t: {self.skill.coins}")
        print()

    def increaseSanity(self,n):
        self.sanity += n
        if (self.sanity > 45):
            self.sanity = 45
        elif (self.sanity < -45):
            self.sanity = -45

    def inflictParalysis(self, c):
        self.paralysisCount += c

    def getHodds(self):
        return (self.sanity + 50) / 100

    class Skill:
        def __init__(self, basePower, coinPower, coins):
            self.basePower = basePower
            self.coinPower = coinPower
            self.coins = coins
```

Fig. 3.1. The abstract for character class (Source: personal archive)

The character class has multiple variables, such as name, sanity, and skill. There is also an inner class called Skill that contains basePower, coinPower, and coins. These are classed together to have a better understanding of the object called character. The real Character class in-game is much more complex than what Fig. 3.1 shown.

D. Calculating Clash Odds of One Instance



```python
def combination(n,k):
    return factorial(n)//(factorial(n-k) * factorial(k))

def binomial_distribution(n,k,hOdds):
    return combination(n,k) * (hOdds**k) * ((1-hOdds)**(n-k))

def clash_odds(Base1,Coin1,Count1,sanity1,Base2,Coin2,Count2,sanity2):
    win_odds = 0.0
    lose_odds = 0.0
    hOdds1 = (sanity1 + 50) / 100
    hOdds2 = (sanity2 + 50) / 100
    for i in range(Count1+1):
        power1 = (Base1 + Coin1 * i)
        if power1 < 0:
            power1 = 0

        for j in range(Count2+1):
            power2 = (Base2 + Coin2 * j)
            if power2 < 0:
                power2 = 0

            if (power1) > (power2):
                win_odds += binomial_distribution(Count1,i,hOdds1) * binomial_distribution(Count2,j,hOdds2)
                print(binomial_distribution(Count1,i,hOdds1) * binomial_distribution(Count2,j,hOdds2))
            elif (power1) < (power2):
                lose_odds += binomial_distribution(Count1,i,hOdds1) * binomial_distribution(Count2,j,hOdds2)
    return (win_odds,lose_odds)
```

Fig. 3.2. Calculation of win odds and lose odds
(Source: personal archive)

Now we reach the first part of the program. First the function for combination is defined. With the combination function, now we will call it with another function for binomial distribution. These are going to be used later to calculate the combination of n amounts of coins in clash odds.

To help better understand what the clash odds do, we are reminded again about conditional probabilities. For the code, the probability function is P, assume that Hn is the how many n heads in combination with n being in the range of 0-coins. P(Hn) is calculated using the binomial distribution. Then we assume that W is win and L is lost. We can get $P(W|Hn)$ and $P(L|Hn)$ then we use rule of sum to either. Below is a decision tree to help in better understanding it:

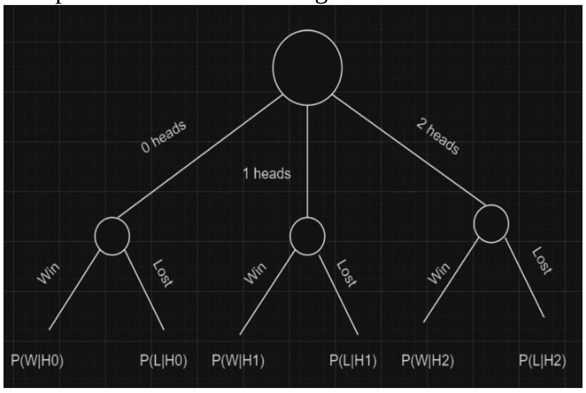
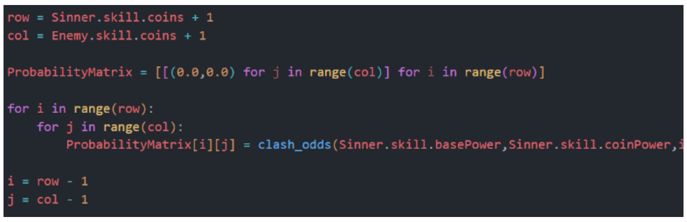


Fig. 3.3. Conditional probabilities for 2 coins first side
(Source: personal archive)

E. Calculating the Overall Win Rate

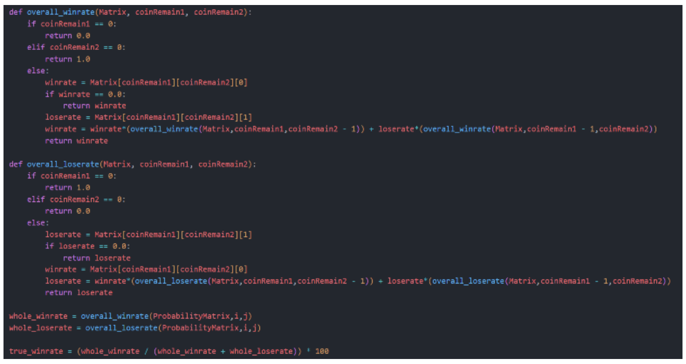
Calculating the win rate without minding draw rate, would need to also calculate the lose rate. Then the true win rate is $TrueWinrate = \frac{Winrate}{Winrate+Loserate}$. Keeping that in mind, let's represent the probability inside a matrix with a row length of coin amount of the first skill and column length of coin amount of the second skill. This is essentially a directed acyclic graph with each state being the remainder coin of both skills.



```python
row = Sinner.skill.coins + 1
col = Enemy.skill.coins + 1

ProbabilityMatrix = [[(0.0,0.0) for j in range(col)] for i in range(row)]

for i in range(row):
    for j in range(col):
        ProbabilityMatrix[i][j] = clash_odds(Sinner.skill.basePower,Sinner.skill.coinPower,i

i = row - 1
j = col - 1
```

Fig. 3.4. Algorithm to create probability matrix
(Source: personal archive)

After we get the matrix, thinking recursively will bring us the answer. Basically, we read it like tree with subtree. To get the win rate, we calculate the probability of a win after it or a win after loses. To calculate the lose rate, we just need to reverse our thinking resulting in this code.



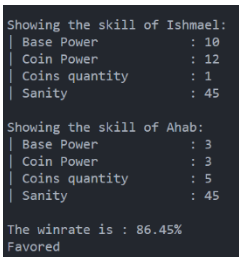
```python
def overall_winrate(Matrix, coinRemain1, coinRemain2):
    if coinRemain1 == 0:
        return 0.0
    elif coinRemain2 == 0:
        return 1.0
    else:
        winrate = Matrix[coinRemain1][coinRemain2][0]
        if winrate == 0.0:
            return winrate
        loserate = Matrix[coinRemain1][coinRemain2][1]
        winrate = winrate*(overall_winrate(Matrix,coinRemain1,coinRemain2 - 1)) + loserate*(overall_winrate(Matrix,coinRemain1 - 1,coinRemain2))
        return winrate

def overall_loserate(Matrix, coinRemain1, coinRemain2):
    if coinRemain1 == 0:
        return 1.0
    elif coinRemain2 == 0:
        return 0.0
    else:
        loserate = Matrix[coinRemain1][coinRemain2][1]
        if loserate == 0.0:
            return loserate
        winrate = Matrix[coinRemain1][coinRemain2][0]
        loserate = winrate*(overall_loserate(Matrix,coinRemain1,coinRemain2 - 1)) + loserate*(overall_loserate(Matrix,coinRemain1 - 1,coinRemain2))
        return loserate

whole_winrate = overall_winrate(ProbabilityMatrix,i,j)
whole_loserate = overall_loserate(ProbabilityMatrix,i,j)

true_winrate = (whole_winrate / (whole_winrate + whole_loserate)) * 100
```

Fig. 3.5. Conditional probabilities for 2 coins first side
(Source: personal archive)

V. Result and Analysis

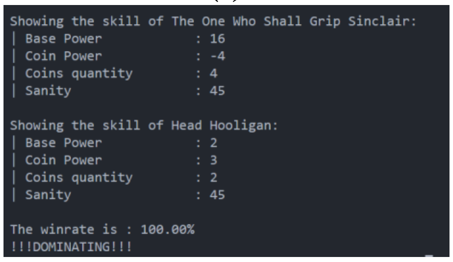
Now we can calculate the more accurate win rate of Fig. 2.10 and 2.11. Here are the results.

(a)



```
Showing the skill of Ishmael:
| Base Power              : 10
| Coin Power              : 12
| Coins quantity          : 1
| Sanity                  : 45

Showing the skill of Ahab:
| Base Power              : 3
| Coin Power              : 3
| Coins quantity          : 5
| Sanity                  : 45

The winrate is : 86.45%
Favored
```

(b)



```
Showing the skill of The One Who Shall Grip Sinclair:
| Base Power              : 16
| Coin Power              : -4
| Coins quantity          : 4
| Sanity                  : 45

Showing the skill of Head Hooligan:
| Base Power              : 2
| Coin Power              : 3
| Coins quantity          : 2
| Sanity                  : 45

The winrate is : 100.00%
!!!DOMINATING!!!
```

Fig. 3.6. Calculation result (a) Fig. 2.10. improvement. (b) Fig2.11 improvement
(Source: personal archive)

Looking at Fig. 3.6.(a) the win rate became *favored*. Ishmael would've lost 13,65% of the time against Ahab and possibly lost the whole fight after. However, this does not account for several status effects and passives applied on Ishmael so the win rate itself may be inaccurate. Looking at Fig. 3.6.(b), Sinclair now has the proper win rate against Head Hooligan.

Both results seem more accurate than what is in the game. However, using the new method might lead to performance issues on several devices especially alongside other mechanics like *stagger*, *one-sided attack*, *speed*, and many more I have not listed. Thus, my personal proposition is to calculate the win rate using the most significant combination of power possible of both skills.

IV. Conclusion

Calculating the win rate of clash may seem easy at first, but with the involvement of other mechanics the game has to offer, may directly or indirectly resulted in creating the simplified version of the calculation. This was done by the game Limbus Company, by only calculating the first instance of the clash rather than the sum of all the probabilities. This paper is meant to try finding a more accurate solution. But while it's maybe more accurate, in creating a game, the developers also need to think how it will run on most devices, so overcomplicating the calculations would bring unoptimized games that ruin the fun out of the players. We need to focus on the bigger picture, after all clash mechanics is not the only thing you need to watch out for in trying to win.

V. Acknowledgment

The author would like to deeply thank Mr. Dr. Ir. Rinaldi Munir, M.T., and Mr. Monterico Adrian, S.T., M.T. as the author's lecturers of Discrete Mathematics, and by extension, the entire Discrete Mathematics staff, consisting of lecturers and assistants, for giving the author a chance to not only deepen their knowledge of the field, but to conduct this study as well. Lastly, but certainly not least, the author would like to thank their friends and families, for always giving them support and always being present while going through every hardship experienced in the process of conducting their study as well as writing this academic paper.

References

[1] E. A. Bender and S. G. Williamson, "Lists, Decisions and Graphs," 2010. [Online]. Available: https://cseweb.ucsd.edu/~gill/BWLectSite/Resources/LDGbookCOV.pdf. [Accessed 9 12 2023].

[2] R. Munir, "Rekursi dan Relasi Rekurens (Bagian 1)," 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/09-Rekursi-dan-relasi-rekurens-(Bagian1)-2023.pdf. [Accessed 10 12 `2023].

[3] R. Munir, "Kombinatorial (Bagian 1)," 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/17-Kombinatorial-Bagian1-2023.pdf. [Accessed 9 12 2023].

[4] The University of Notre Dame, "The Binomial Distribution," [Online]. Available: https://www3.nd.edu/~rwilliam/stats1/x13.pdf. [Accessed 9 12 2023].

[5] Prydwen Institute, "Limbus Company database and wiki," 2023. [Online]. Available: https://www.prydwen.gg/limbus-company/. [Accessed 8 12 2023].

Statement

I hereby declare that what I have written in this paper is my own work, not an adaptation nor a translation of someone else's and not plagiarism as it's an insult to the integrity of a research.

Bandung, December 11th, 2023

[illegible]

Rayhan Ridhar Rahman
13522160