Utilizing Greedy Algorithm to Automate Turn in The Game “Library of Ruina”

Rayhan Ridhar Rahman - 13522160
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522160@std.stei.itb.ac.id

***Abstract*—This paper presents the usage of greedy algorithm by different factors to implement automation of turn in Library of Ruina. Library of Ruina is a deck builder game, where the main gameplay is assigning cards from our team against the enemies. Greedy algorithm is an algorithm that follows problem-solving heuristic of making the locally optimal choice, which most of the time results in suboptimal solution. The gameplay relies on throwing dice, so the greedy algorithm will use win rate and average damage as its base.**

***Keywords*—greedy algorithm; Library of Ruina; cards; dice; win rate; damage**

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 until recent years, it has accompanied many adolescents to their adulthood. Video games are interactive digital media, a fusion of miraculous results between technology and imagination.

Library of Ruina is a deck-building turn-based RPG game developed by South Korean indie video game studio, Project Moon and served as a direct sequel to their debut game Lobotomy Corporation. This game was officially released for PC and Xbox One in August of 2021 and later released for Nintendo Switch and PlayStation 5 in April of 2024.

We enter the world known as “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. The rich stayed in the nest of a district, meanwhile the majority of the populus stayed in the backstreet where crime is rampant, and violence is a normality.

Library of Ruina mainly took place in *the library*, a supernatural place where guests fight against librarians to get any book they wanted. The place itself is inspired by *the library of babel*, where all possible books with all possible letters are contained. The library holds a mythical power to turn anyone that died in the place into book with their whole life imprinted on the pages.

Fig. I.1 Official Artwork for PS4/Switch Port
(Source: https://www.arcsystemworks.jp/lor/en/gallery.php)

The story mainly focuses on two of our main characters. Angela, the library’s director of her role namesake’s, which was first introduced as a side-character on Lobotomy Corporation. She is an AI that’s capable of human emotions. Now she wants to find the one true book which will solve all her problems. Through receptions, each defeated guest brings her closer to that book, serving as a compass.

The other is Roland, a washed-up lowest grade *fixer*, which stumbled upon the library while trying to buy a meal. After hitting the bottom of the barrel, he didn’t know what else to do and so he kept waiting by Angela’s side and assisted her toward reaching her goal. He is knowledgeable of *the city*. Through him, both Angela and the player will learn more about the city and its inhabitants.

The purpose of this paper is to try adapting Limbus Company’s (This game sequel) win rate and damage button and put it into Library of Ruina. Given how those buttons worked in Limbus Company, the algorithm should be of greedy algorithm. Assigning the most optimal for the fastest skill and followed by the rest

II. Theoretical Basis

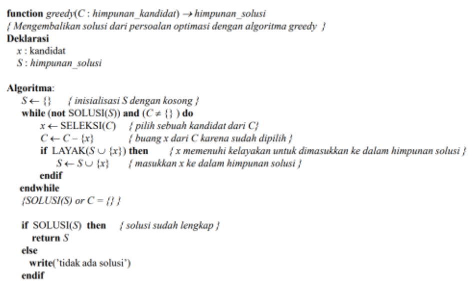
A. *Greedy Algorithm*

The greedy algorithm is a popular algorithm to solve the problem for optimization. This algorithm follows the principle of taking what you can now. After taking the local solution, there is no turning back. So, it hopes to search for the optimal solution through locally optimal solution.

Greedy algorithm consisted of multiple elements; those are:

1. Candidates set, C: List of candidates that will be chosen in every step (e.g. node in graph, task, coins, things, etc.)
2. Solutions set, S: List of chosen candidates
3. Solution function: determine that list of candidates have already given solution
4. Selection function: choose candidate based on specific greedy heuristic strategy
5. Feasibility function: check if the candidate that has been chosen can be put into solutions set
6. Objective function: maximize or minimize

The global optimum from this algorithm may not be the actual optimal solution. This happens because it doesn't exhaustively analyze all the possibilities. There is also a factor from the selection function, which can be the different maker. But because of their simplicity, greedy algorithms are frequently straightforward and efficient.



```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
  x : kandidat
  S : himpunan_solusi

Algoritma:
   S ← {}      { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
      x ← SELEKSI(C)    { pilih sebuah kandidat dari C}
      C ← C − {x}       { buang x dari C karena sudah dipilih }
      if LAYAK(S ∪ {x}) then     { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
         S ← S ∪ {x}    { masukkan x ke dalam himpunan solusi }
      endif
  endwhile
   {SOLUSI(S) or C = {} }

   if SOLUSI(S) then    { solusi sudah lengkap }
     return S
   else
    write('tidak ada solusi')
   endif
```

Fig. II.1 General Schema for Greedy Algorithm
(Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

In the case where the absolute optimal solution is not needed, this algorithm can serve well as approximation solution. Then to prove the optimality of solution given by greedy algorithm is another challenge. It is easier to serve counterexample proving the solution is not optimal.

B. Discrete Uniform Distribution Probability

Discrete means in a way analogous to discrete variables, having a bijection with the set of natural numbers. A uniform distribution is a type of symmetric probability distribution in which all the outcomes have an equal likelihood of occurrence. A discrete uniform distribution is one that has a finite (or countably finite) number of random variables that have an equally likely chance of occurring.

Let x be a discrete random variable having n values over the interval [a, b]; x has a discrete uniform distribution if its probability mass function (pmf) is defined by:

$$f(x) = \frac{1}{b-a+1}, \text{for } x = a, a+1, \dots, b \tag{1}$$

This pmf will be used to later calculate the probability of an X discrete random variable within one range being higher than a Y discrete random variable within another range both completely unrelated to each other. To calculate that probability, combinatorics can be used. Let nx be the number of possible values for X within [ax,bx] and ny for Y between [ay,by]:

$$n_x = b_x - a_x + 1 \tag{2}$$

$$n_y = b_y - a_y + 1 \tag{3}$$

After we got both size, we can calculate their pmf:

$$P(X = x) = \frac{1}{n_x} \tag{4}$$

$$P(Y = y) = \frac{1}{n_y} \tag{5}$$

Then we denote $I(x > y)$ as the summation in the case of y being lower than x. The we put both pmf into a general formula:

$$P(X > Y) = \sum_{x=a_x}^{b_x} P(X = x) \sum_{y=a_y}^{b_y} P(Y = y).I(X > Y) \tag{6}$$

$$P(X > Y) = \frac{1}{n_x n_y} \sum_{x=a_x}^{b_x} \sum_{y=a_y}^{b_y} I(X > Y) \tag{7}$$

Finally, we simplify the inner sum of (7), creating the final formula:

$$P(X > Y) = \frac{1}{n_x n_y} \sum_{x=a_x}^{b_x} \max\left(0, \min\left(b_y, x-1\right) - a_y + 1\right) \quad (7)$$

C. Library of Ruina : Combat Pages (Cards) and Dice

Combat pages are what the cards are called in the game. Each can be assigned to a speed die of a character. It can also interact with other combat pages if there is a pair of speed die targeting each other.

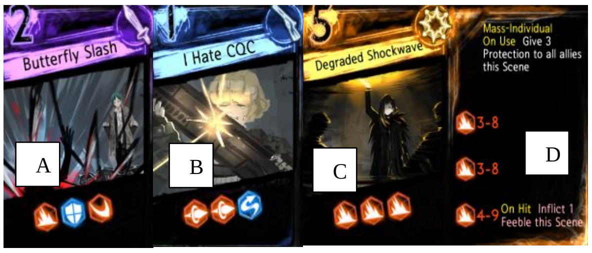

2
Butterfly Slash
1
I Hate CQC
5
Degraded Shockwave
Mass-Individual
On Use Give 3 Protection to all allies this Scene
3-8
3-8
4-9 On Hit Inflict 1 Feeble this Scene
A
B
C
D

Fig. II.2 Card Example (A: melee; B: ranged; C: mass-attack; D: card description)
(Source: Personal Library)

There are three types of combat page. Melee, ranged, and mass attack. Each has different interactions with the others. Every combat page contains a queue of dice that hold a range of values, used for clashing. Below are the properties of each card type.

1. **Melee** pages can only target one speed die. Has the lowest priority in combat.
2. **Ranged** pages can only target one speed die. Has higher priority than melee but lower than mass attack.
3. **Mass attack** pages can target one main speed die and target every other character on a random speed die. This page can't trigger passive effects.

Normally the interaction between cards is each die in both queues clashing with each other. Both die roll within their range. Then if one die is higher, the losing die is removed, meanwhile the winner attacks the target and removes itself. If it's a draw, both dice are removed.

Then there is also melee vs ranged. If a die from melee won against the ranged page, the winner die won't attack and being moved to the back of the queue. The melee page can only attack after the queue for the ranged page is empty.

Finally, there is mass attack vs other type. Mass attack can't interact with other mass attack. Mass attack cannot be redirected, only the speed die targeted by the mass attack may answer, these dice don't need to target the mass attack related speed die. There are two types of mass attack which are summation and individual. Combat pages directed by summation will add up all the rolled dice value and compare it meanwhile if targeted by individual, each die being compared to each die of the mass attack. It will destroy every dice that rolled lower and damage them.

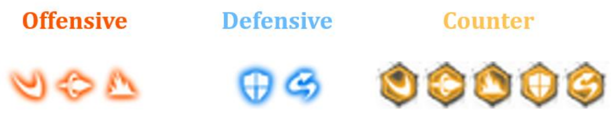


Fig. II.3 Dice Types
(Source: Personal Library)

Dice also has their own typing, offensive dice which consists of slash, pierce, and blunt and defensive dice which consists of block and evade. There is also counter type that can possess any type but only clashes against one-sided attacks. Below is the table for clashing interactions between dice.

TABLE I. "ON CLASH" TABLE GRID

vs	*Offensive dice*	*Block dice*	*Evade dice*
Offensive dice	Win: Deals damage equal to roll value, if melee vs ranged, recycled to the back of queue Draw: Negate both dice Lose: Negate dice	Attack Win: Deal damage equal to the die roll subtract the roll of the Block die. Draw: Damage nullified Block Win: Deals stagger damage equal to roll value	Win: Deals damage equal to roll value Draw: Damage nullified Evade Win: Damage nullified, recycle evade die, recover stagger equal to roll value
Block dice	Block Win: Deals stagger damage equal to roll value Draw: Damage nullified Attack Win: Deal damage equal to the die roll subtract the roll of the Block die.	Win: Deal Stagger damage equal to the winner's roll. Draw: Both dice are negated. Lose: Take Stagger damage equal to the winner's roll.	Block Win: Deal Stagger damage equal to the die roll. Draw: Both Dice are negated. Evade Win: Recover Stagger equal to the winner's roll.
Evade dice	Evade Win: Damage nullified, recycle evade die, recover stagger equal to roll value Draw: Damage nullified Win: Deals damage equal to roll value	Evade Win: Recover Stagger equal to the winner's roll. Draw: Both Dice are negated. Block Win: Deal Stagger damage equal to the die roll.	Both dice are always negated

D. *Library of Ruina : Stageplay Phase*

Reception in Library of Ruina consisted of three phases, not counting cutscenes. Consisted of preparation phase, stageplay phase, and post-reception phase. The combat took place in stageplay phase which also consisted of multiple scenes (turns). Every turn is divided into *scene start*, *rolling speed dice*, *choosing combat pages*, and finally *combat stage*.

Fig. II.4 Example of The Start of a Scene
(Source: Personal Library)

Scene start is the first phase. Showing the result of previous scene such as status effects, light gained, and their emotion level. If a character staggers an enemy, they gain more light, and if a character is staggered, they won't receive any light and won't roll the speed dice in the next phase.

Every character may have multiple speed dice, due to emotion level or passive status. Speed dice are independent of one another even with the same character. After the starting scene, every character will roll their speed dice. The value gained will be within the range of the character's speed, which will be important when selecting combat pages.

Selecting combat pages is the scene where the player took the most part in. When selecting combat pages, it is important to understand the combat mechanic of this game.

Each speed die can be assigned a combat page from their deck and directed to other speed die. After rolling for speed, every enemy will direct attacks on our librarians with all their speed dice. Some speed dice might be empty due to combat pages cost or ran out of card in their active deck. After that, we can start assigning our own combat pages.

The speed die that has more speed can redirect attack from other slower speed die. A speed die may be targeted by multiple speed dice, so it's very important to redirect strong attack from struggling librarian. At a higher emotional level, our librarian may use powerful E.G.O pages that are shared among all librarians.

After assigning our librarians combat pages, we can move on to combat stage which will test the player strategy and show the result. The stage will begin with mass attack pages, then ranged pages, and finally, melee pages each being played in descending order of their related speed dice.

E. *Limbus Company : Automation of Skill Assignment*

Limbus Company already has its own win rate and damage button to automate the process. Library of Ruina also has it byt pressing "P" on the keyboard, but the result suggests that it is randomized and not a strategical approach unlike Limbus

Company's. This can be seen by clicking it multiple times, resulting in an entirely different set of combat pages.

Limbus Company, as mentioned before, has two different buttons to automate the turn. In normal battles, it doesn't differ much, so this paper won't cover the normal battles algorithm. Instead, this paper will focus on the algorithm in focused encounter. It should be much more like combat mechanic in Library of Ruina while also having a clear distinct algorithm between both buttons.

From what was observed of the win rate button, the target itself is not clear cut as one would have thought. But it remains consistent whenever we retry the process. So here is what was observed.

One of the most promising hypotheses that can be pulled is the fastest slot will pick the clash among the selection which has the highest win rate. If the win rate is about the same, then pick the fastest slot. Then slower slots will also try to do the same until all enemy's slots have been targeted, whether it was a clash or a one-sided attack. Once all of them are covered, the rest will follow the same algorithm as the damage button.

Then the damage button is simpler. It will focus all attack into one slot, guaranteeing most of the slot will do a one-sided attack while the fastest one clashes with said slot. This is less optimal than the win rate button, as it will lead to overkilling one enemy, while other enemies can get free attacks on our team.

But there is also a special case if the targeted enemy has multiple body parts, the fastest will target the leftmost slot of the limbs, then the leftmost slot of the next limb, until the main body get targeted, it will loop back.

F. *Differences in The Combat Mechanics*

Limbus Company mostly retained the mechanics from Library of Ruina, such as how the turn works. Beginning with selecting skills / cards against the enemies and clashing mechanics. But both games have different directions they are going for other than those base mechanics. Especially because Limbus Company must account for players that may not have all the resources since it's a gacha game. Let's abbreviate those name into LC (Limbus Company) and LoR (Library of Ruina)

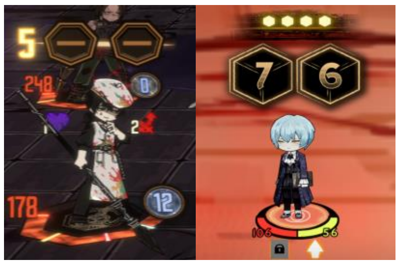


Fig. II.5 LC and LoR Character Comparison
(Source: Personal Library)

The main one is the difference between speed dice and skill slots. Speed dice from LoR works independently of each other, meanwhile in LC, skill slots of the same character are treated as the same slot. In the algorithm that was observed in LC, it was founded that, the targeting start by each character, so even if a skill slot might show up on the right or regarded as slower, if it is by the same character then it will first calculate the higher win rate with whichever skill slot.

In both games, each character has health points represented in health bar. The difference lies in how stagger works. LoR has a separate bar for stagger. Meanwhile in LC, stagger is based on how much health point a character had left represented by the yellowish line in their health bar.

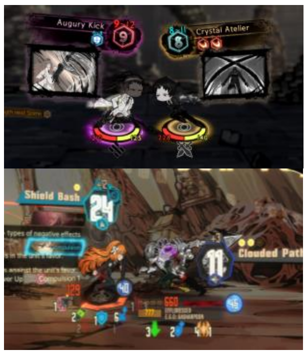


Fig. II.6 LoR and LC Clash Comparison
(Source: Personal Library)

The other major difference lies in how clashing works. In LoR, each dice of opposing cards will clash only once then move on to the next. But in LC, a clash will happen until one side loses all their coins or reaches the maximum limit of clash. After randomizing the whole value of both skills

Finally, in Library of Ruina, it centers around the emotion level of characters, giving the character more light (currency to use card), more speed die and access to abnormality pages. Meanwhile Limbus Company has sanity which raises the chance of getting heads coin and sin resources that can be used for E.G.O skills.

III. METHODOLOGY

A. *Constraint*

To make it easier, we will create an algorithm based on just the numbers within each die and not accounting for special conditional. Then, status effects will also be ignored. Then the deck used may not contain mass attack pages. Finally, it will assume that every die in a combat page will be used, so counter dice will not be present and any unused defensive dice will be used for stagger damage.

B. *Tools*

Used tools for the simulation:

1. java 17.0.8
2. IntelliJ IDEA
3. Maven

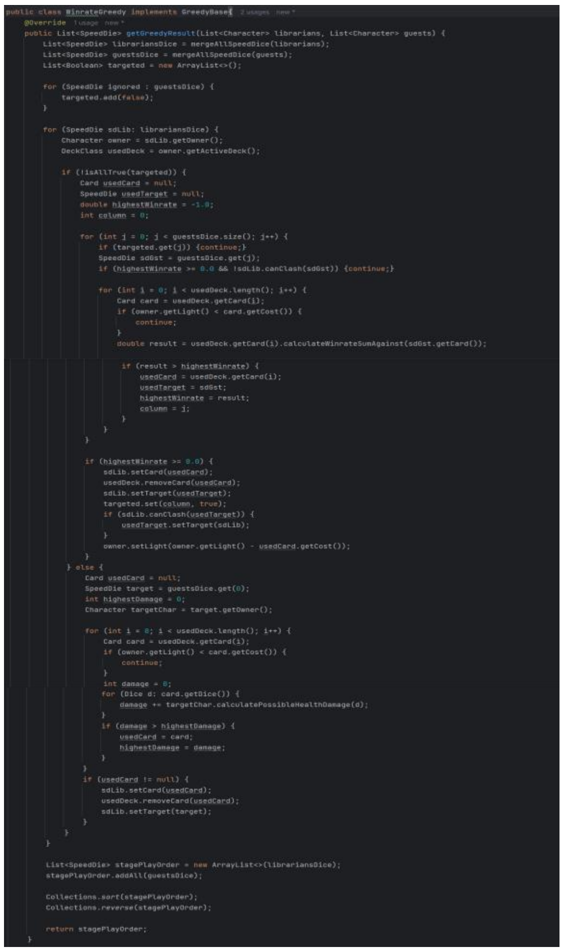
C. *Simulation Process*

The simulation will start with the creation of characters. Then add a good amount of speed dice to those characters. Then, create some of the cards that will be used. Then add those cards to the active deck of each character. Then assign each character to team librarians (allies) and team guests (enemies).

To start the algorithm, we will first roll the speed dice of all the characters involved. Then we get a separate priority queue between both teams. Before trying out the algorithm, each character from the guests team will start to set their speed die into the librarians. Finally, we can start using the algorithm.

D. Implementation of Greedy Algortihm by Win Rate

The greedy by win rate algorithm is mainly used to negate the most damage in Limbus Company. The reason is healing is very scarce and so we must try to negate every damage while also attacking the enemies.



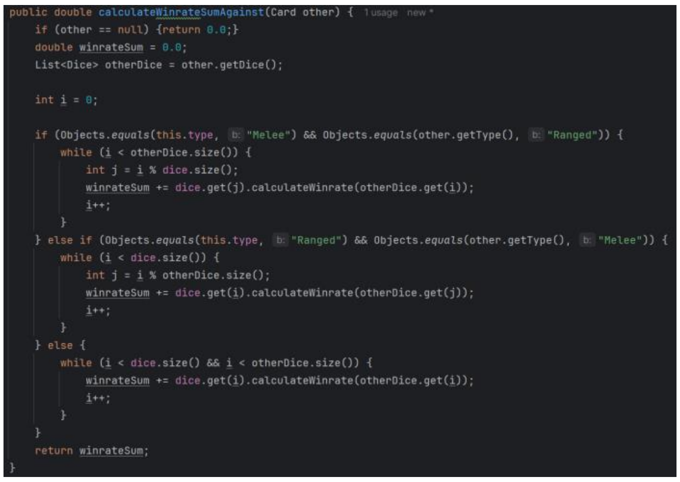
```java
public class WinrateGreedy implements GreedyBase{
    @Override
    public List<SpeedDie> getGreedyResult(List<Character> librarians, List<Character> guests) {
        List<SpeedDie> librariansDice = mergeAllSpeedDice(librarians);
        List<SpeedDie> guestsDice = mergeAllSpeedDice(guests);
        List<Boolean> targeted = new ArrayList<>();

        for (SpeedDie ignored : guestsDice) {
            targeted.add(false);
        }

        for (SpeedDie sdLib: librariansDice) {
            Character owner = sdLib.getOwner();
            DeckClass usedDeck = owner.getActiveDeck();

            if (!isAllTrue(targeted)) {
                Card usedCard = null;
                SpeedDie usedTarget = null;
                double highestWinrate = -1.0;
                int column = 0;

                for (int j = 0; j < guestsDice.size(); j++) {
                    if (targeted.get(j)) {continue;}
                    SpeedDie sdGst = guestsDice.get(j);
                    if (highestWinrate >= 0.0 && !sdLib.canClash(sdGst)) {continue;}

                    for (int i = 0; i < usedDeck.length(); i++) {
                        Card card = usedDeck.getCard(i);
                        if (owner.getLight() < card.getCost()) {
                            continue;
                        }
                        double result = usedDeck.getCard(i).calculateWinrateSumAgainst(sdGst.getCard());

                        if (result > highestWinrate) {
                            usedCard = usedDeck.getCard(i);
                            usedTarget = sdGst;
                            highestWinrate = result;
                            column = j;
                        }
                    }
                }

                if (highestWinrate >= 0.0) {
                    sdLib.setCard(usedCard);
                    usedDeck.removeCard(usedCard);
                    sdLib.setTarget(usedTarget);
                    targeted.set(column, true);
                    if (sdLib.canClash(usedTarget)) {
                        usedTarget.setTarget(sdLib);
                    }
                    owner.setLight(owner.getLight() - usedCard.getCost());
                }
            } else {
                Card usedCard = null;
                SpeedDie target = guestsDice.get(0);
                int highestDamage = 0;
                Character targetChar = target.getOwner();

                for (int i = 0; i < usedDeck.length(); i++) {
                    Card card = usedDeck.getCard(i);
                    if (owner.getLight() < card.getCost()) {
                        continue;
                    }
                    int damage = 0;
                    for (Dice d: card.getDice()) {
                        damage += targetChar.calculatePossibleHealthDamage(d);
                    }
                    if (damage > highestDamage) {
                        usedCard = card;
                        highestDamage = damage;
                    }
                }
                if (usedCard != null) {
                    sdLib.setCard(usedCard);
                    usedDeck.removeCard(usedCard);
                    sdLib.setTarget(target);
                }
            }
        }

        List<SpeedDie> stagePlayOrder = new ArrayList<>(librariansDice);
        stagePlayOrder.addAll(guestsDice);

        Collections.sort(stagePlayOrder);
        Collections.reverse(stagePlayOrder);

        return stagePlayOrder;
    }
```

Fig. III.1 Proposed Win Rate Implementation
(Source: Personal Library)

First, we can think of the whole stage as a matrix. The rows represent the queue for librarians and the columns represent the queue for quests. Each box then contains a card already being set by guests on the right and librarians later will take the left. In the algorithm created, there is also a targeted queue with the same length as the queue for guests.

We start iterating from the first row until the end. In the first row, we will start by searching for columns that are clash able. If not clash able, then set the points to 0. If it is clash able, try to calculate the win rate of every possible card from the associated librarian's deck. After every column has been analyzed, it will pick the column which has the highest win rate that appeared first. Then the targeted queue will be set true for that column.

Then starting the next row, it will ignore any column that has already clashed with previous rows. This process will be repeated until each librarian has used their speed dice or all the guests' speed dice has been targeted. Then it will follow the same algorithm as greedy by health damage. Below is the implementation of win rate calculation of cards.

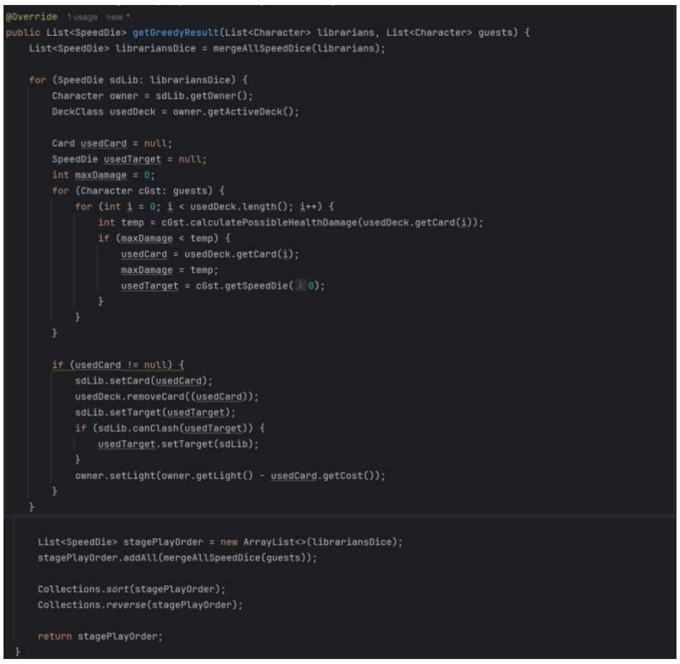


```java
public double calculateWinrateSumAgainst(Card other) {
    if (other == null) {return 0.0;}
    double winrateSum = 0.0;
    List<Dice> otherDice = other.getDice();

    int i = 0;

    if (Objects.equals(this.type, "Melee") && Objects.equals(other.getType(), "Ranged")) {
        while (i < otherDice.size()) {
            int j = i % dice.size();
            winrateSum += dice.get(j).calculateWinrate(otherDice.get(i));
            i++;
        }
    } else if (Objects.equals(this.type, "Ranged") && Objects.equals(other.getType(), "Melee")) {
        while (i < dice.size()) {
            int j = i % otherDice.size();
            winrateSum += dice.get(i).calculateWinrate(otherDice.get(j));
            i++;
        }
    } else {
        while (i < dice.size() && i < otherDice.size()) {
            winrateSum += dice.get(i).calculateWinrate(otherDice.get(i));
            i++;
        }
    }
    return winrateSum;
}
```

Fig. III.2 Proposed Win Rate Calculation
(Source: Personal Library)

E. Implementation of Greedy Algortihm by Damage

The greedy by health damage algorithm will be based on the average possible damage a card can do to an enemy. In Limbus Company, this algorithm is considered bad, since overkilling an enemy can happen often, especially since most stages have the same type of enemy.



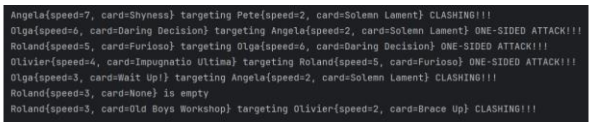
```java
@Override
public List<SpeedDie> getGreedyResult(List<Character> librarians, List<Character> guests) {
    List<SpeedDie> librariansDice = mergeAllSpeedDice(librarians);

    for (SpeedDie sdLib: librariansDice) {
        Character owner = sdLib.getOwner();
        DeckClass usedDeck = owner.getActiveDeck();

        Card usedCard = null;
        SpeedDie usedTarget = null;
        int maxDamage = 0;
        for (Character cGst: guests) {
            for (int i = 0; i < usedDeck.length(); i++) {
                int temp = cGst.calculatePossibleHealthDamage(usedDeck.getCard(i));
                if (maxDamage < temp) {
                    usedCard = usedDeck.getCard(i);
                    maxDamage = temp;
                    usedTarget = cGst.getSpeedDie(0);
                }
            }
        }

        if (usedCard != null) {
            sdLib.setCard(usedCard);
            usedDeck.removeCard((usedCard));
            sdLib.setTarget(usedTarget);
            if (sdLib.canClash(usedTarget)) {
                usedTarget.setTarget(sdLib);
            }
            owner.setLight(owner.getLight() - usedCard.getCost());
        }
    }

    List<SpeedDie> stagePlayOrder = new ArrayList<>(librariansDice);
    stagePlayOrder.addAll(mergeAllSpeedDice(guests));

    Collections.sort(stagePlayOrder);
    Collections.reverse(stagePlayOrder);

    return stagePlayOrder;
}
```

Fig. III.3 Proposed Greedy by Damage Algorithm
(Source: Personal Library)

Once again, we can imagine it as a matrix. This time the columns are representing the guests, since we only need their fastest skill die. After calculating the potential damage against all of the column, the next row doesn't have to mind about which character has already been targeted. Since we would want our total damage to be as high as possible.

There is also another variation of this algorithm being the greedy by stagger damage. The difference with the former is that greedy by damage prefer staggered enemy, since their resistance are switched to fatal. Meanwhile greedy by stagger damage will ignore the enemies that are already staggered. Block dice also contributes as flat value for stagger damage.

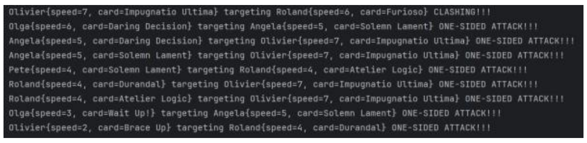
IV. Testing and Analysis



```
Angela{speed=7, card=Shyness} targeting Pete{speed=2, card=Solemn Lament} CLASHING!!!
Olga{speed=6, card=Daring Decision} targeting Angela{speed=2, card=Solemn Lament} ONE-SIDED ATTACK!!!
Roland{speed=5, card=Furioso} targeting Olga{speed=6, card=Daring Decision} ONE-SIDED ATTACK!!!
Olivier{speed=4, card=Impugnatio Ultima} targeting Roland{speed=5, card=Furioso} ONE-SIDED ATTACK!!!
Olga{speed=3, card=Wait Up!} targeting Angela{speed=2, card=Solemn Lament} CLASHING!!!
Roland{speed=3, card=None} is empty
Roland{speed=3, card=Old Boys Workshop} targeting Olivier{speed=2, card=Brace Up} CLASHING!!!
```

Fig. IV.1 Win Rate Result
(Source: Personal Library)

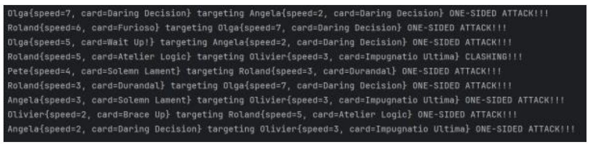
The result for the win rate algorithm is suboptimal than what was hoped. Both of our librarians (Roland and Angela) clash against the weaker dice (Solemn Lament has 8 weak dice), meanwhile some powerful combat pages get a one-sided attack on the librarians. This suggests that using the summation of the win rate of each dice is not the correct approach in solving the problem. The problem arises from the clash against different types of combat page (melee vs ranged).



```
Olivier{speed=7, card=Impugnatio Ultima} targeting Roland{speed=6, card=Furioso} CLASHING!!!
Olga{speed=6, card=Daring Decision} targeting Angela{speed=5, card=Solemn Lament} ONE-SIDED ATTACK!!!
Angela{speed=5, card=Daring Decision} targeting Olivier{speed=7, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
Angela{speed=5, card=Solemn Lament} targeting Olivier{speed=7, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
Pete{speed=4, card=Solemn Lament} targeting Roland{speed=4, card=Atelier Logic} ONE-SIDED ATTACK!!!
Roland{speed=4, card=Durandal} targeting Olivier{speed=7, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
Roland{speed=4, card=Atelier Logic} targeting Olivier{speed=7, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
Olga{speed=3, card=Wait Up!} targeting Angela{speed=5, card=Solemn Lament} ONE-SIDED ATTACK!!!
Olivier{speed=2, card=Brace Up} targeting Roland{speed=4, card=Durandal} ONE-SIDED ATTACK!!!
```

Fig. IV.2 Greedy by Health Damage Result
(Source: Personal Library)

The result for greedy by health damage is about what was expected, all the librarians only target character based on the damage they can deal based on resistance. This time, the use of the card "Shyness", a pure defensive combat page cannot be seen. The number of one-sided attacks also increases compared to greedy by win rate.



```
Olga{speed=7, card=Daring Decision} targeting Angela{speed=2, card=Daring Decision} ONE-SIDED ATTACK!!!
Roland{speed=6, card=Furioso} targeting Olga{speed=7, card=Daring Decision} ONE-SIDED ATTACK!!!
Olga{speed=5, card=Wait Up!} targeting Angela{speed=2, card=Daring Decision} ONE-SIDED ATTACK!!!
Roland{speed=5, card=Atelier Logic} targeting Olivier{speed=3, card=Impugnatio Ultima} CLASHING!!!
Pete{speed=4, card=Solemn Lament} targeting Roland{speed=3, card=Durandal} ONE-SIDED ATTACK!!!
Roland{speed=3, card=Durandal} targeting Olga{speed=7, card=Daring Decision} ONE-SIDED ATTACK!!!
Angela{speed=3, card=Solemn Lament} targeting Olivier{speed=3, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
Olivier{speed=2, card=Brace Up} targeting Roland{speed=5, card=Atelier Logic} ONE-SIDED ATTACK!!!
Angela{speed=2, card=Daring Decision} targeting Olivier{speed=3, card=Impugnatio Ultima} ONE-SIDED ATTACK!!!
```

Fig. IV.3 Greedy by Stagger Damage Result
(Source: Personal Library)

The result for greedy by stagger damage is about the same as greedy by health damage, all the librarians only target character based on the damage they can deal based on resistance. Pure defensive combat page did not get used again, since having a weakness again certain offensive type could lead to more stagger damage than just the value from block dice.

The more optimal solution for win rate might lies in greedy by stagger damage. As previously mentioned, the main purpose of the win rate algorithm is to negate the most damage. Combining it with greedy by stagger damage ensure facorable clash may happen more.

V. Conclusion

The greedy algorithm is popular because of its efficient and straightforward nature. Leading towards a faster algorithm that can be used to solve a problem fast while being in the approximate region of the optimal solution.

In video games where performance is needed for an enjoyable experience, this couldn't be further from the truth. There are a lot of games with many variations, many strategies, and many playstyles so getting the most optimal solution in a blitz is an impossible goal. We don't necessarily need the optimal solution to have fun.

From this paper, what I have learned is to appreciate the video games alongside their developer. Maybe LoR is not the cup of tea for everyone, but it certainly is for me. There are many unnoticed details when you just play normally. But once you try to change your perspective, it enhances the fun you get from playing it.

The algorithms found are unoptimized for the game. But they are very efficient. But to implement it into real gameplay is going to be hard without knowing the specification of the game engine. That's why most of the interesting mechanics are being left by this paper.

Below is link to the GitHub repository to understand more about the classes that were made.

Repository Link

https://github.com/Rinnearu/13522160_Makalah_Stima

Acknowledgment

The author would like to deeply thank Mr. Dr. Ir. Rinaldi Munir, M.T., Dr. Nur Ulfa Maulidevi, and Dr. Ir. Rila Mandala, as the author's lecturers of Algorithm Strategies, and by extension, the entire Algorithm Strategies 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] Munir, R. (2024). Algoritma Greedt (Bagian 1). Accessed at : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (Accessed: 12th June 2024)

[2] Munir, R. (2024). Algoritma Greedt (Bagian 2). Accessed at : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf (Accessed: 12th June 2024)

[3] S. Sinharay, Discrete Probability Distributions, Editor(s): Penelope Peterson, Eva Baker, Barry McGaw, International Encyclopedia of Education (Third Edition), Elsevier, 2010, Pages 132-134, ISBN

9780080448947, https://doi.org/10.1016/B978-0-08-044894-7.01721-8. (https://www.sciencedirect.com/science/article/pii/B9780080448947017218) (Accessed: 12th June 2024)

PERNYATAAN

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, 12 Juni 2024

[illegible]

Rayhan Ridhar Rahman
13522160