Analisis Pengaplikasian Algoritma *Greedy* pada *Battle Automation* dalam Permainan Honkai Star Rail

Louis Caesa Kesuma - 13521069
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (13521069@std.stei.itb.ac.id):

***Abstract*—Permainan *Honkai Star Rail* adalah permainan RPG (*Role Playing Game*) dengan *gameplay turn-based strategy*. Dalam *gameplay*-nya, *Honkai Star Rail* memiliki sebuah fitur otomasi yang dapat digunakan para pemain saat melakukan pertempuran, akan tetapi fitur tersebut bisa dibilang tidaklah optimal sehingga para pemain relatif sungkan untuk menggunakan fitur tersebut. Makalah ini akan menganalisis pengaplikasian algoritma *greedy* pada fitur otomasi tersebut yang menyebabkan hasil otomasinya tidak optimal.**

Keywords—Honkai Star Rail, turn-based strategy, otomasi, greedy

I. PENDAHULUAN

Honkai Star Rail (HSR) adalah permainan *online turn-based strategy* baru yang dibuat oleh *Hoyoverse*. HSR dirilis pada tanggal 26 April 2023. Selama kurang lebih 1 bulan setelah rilisnya secara global, HSR sudah memiliki kurang lebih 25 juta pemain global. Hal ini sangatlah wajar jika kita melihat permainan-permainan lain yang dikembangkan oleh *Hoyoverse*, seperti *Genshin Impact* dan *Honkai Impact 3rd* yang sangatlah sukses. HSR memiliki *gameplay* yang bisa dibilang mirip dengan *game-game* seperti *Persona*, *Final Fantasy*, dan sebagainya dimana pemain sebagai karakter utama dari *game* tersebut akan menjelajahi dunia *game* tersebut. Dalam proses penjelajahan tersebut kita akan bertemu dengan karakter-karakter lainnya, menjelajahi area-area yang tersedia, mengikuti cerita-cerita dari area yang kita kunjungi, serta mengalahkan musuh-musuh yang kita temukan di perjalanan kita. Musuh-musuh yang tersedia juga beragam, baik dari musuh biasa hingga *boss* yang berkaitan dengan cerita dari *game*-nya. Musuh-musuh tersebut juga dapat bertambah kuat seiring dengan pertumbuhan karakter kita, sehingga kita juga harus memperkuat karakter-karakter yang kita miliki agar dapat mengalahkan musuh-musuh tersebut.

Dalam proses pertempurannya, kita dapat memilih tim yang terdiri dari 4 karakter pilihan kita. Kita juga bebas untuk memilih karakter manapun dari 28 karakter yang sekarang tersedia di HSR. Akan tetapi, kita tidak secara langsung bisa mendapatkan seluruh karakter yang tersedia. Untuk mendapatkan karakter-karakter baru, kita harus melakukan pengundian (*gacha*). Para pemain bisa mengumpulkan mata uang yang ada di game tersebut, dan kemudian pemain bisa melakukan pengundian. Pemain mungkin mendapatkan karakter pada kali pertama melakukan pengundian, namun kesempatan tersebut sangatlah kecil. HSR memberikan jaminan bahwa dalam 10 kali pengundian, para pemain akan mendapat 1 karakter/senjata yang memiliki kelangkaan bintang 4. Setelah pemain melakukan 90 kali pengundian, maka pemain akan dijamin untuk mendapatkan 1 karakter/senjata yang memiliki kelangkaan bintang 5. Kita dapat meningkatkan kekuatan karakter-karakter yang kita miliki, baik dengan meningkatkan levelnya, meningkatkan kemampuan yang dimilikinya, dan sebagainya.

Tim yang telah dibentuk oleh pemain tentunya dapat disesuaikan lagi oleh pemainnya, namun tidak saat pemain sedang melakukan pertempuran dengan musuh. Setiap karakter memiliki kemampuan serta elemen yang dikuasainya. Elemen-elemen yang bisa dikuasai oleh karakter-karakter tersebut adalah satu diantara *Fire*, *Ice*, *Lightning*, *Wind*, *Imaginary*, dan *quantum*. Elemen-elemen tersebut dapat dimanfaatkan oleh pemain untuk mempermudah pertempurannya dengan musuh, karena setiap musuh memiliki kelemahan yang berkaitan dengan beberapa elemen tersebut. Setiap karakter juga memiliki peran yang umumnya dapat menjelaskan bagaimana karakter tersebut akan dimainkan. Peran-peran yang bisa dikuasai oleh karakter-karakter tersebut adalah salah satu diantara *Destruction* (*General Damage Dealer*), *Hunt* (*Single Target Damage Dealer*), *Erudition* (*Multi Target Damage Dealer*), *Harmony* (*Utility Support*), *Nihility* (*Debuffer*), *Preservation* (*Defensive Support*), dan *Abundance* (*Healer*). *Destruction* (*General Damage Dealer*). Sebagai pemain, kita dapat dengan mudah memenangkan pertempuran jika kita bisa menyusun tim yang memanfaatkan peran masing-masing karakter serta kelemahan musuh.

Karena HSR merupakan permainan *turn-based strategy*, akan lebih baik jika pemain menentukan aksi apa yang harus dilakukan pada giliran (*turn*) tersebut. Akan tetapi, HSR juga menyediakan fitur *auto-battle* yang dapat dihidupkan oleh pemain dalam pertempuran, pemain juga tentunya dapat mematikan fitur tersebut. Fitur *auto-battle* tersebut tentunya tidak akan berjalan sesuai dengan keinginan pemainnya. Fitur otomasi dalam segala permainan umumnya tidak efisien dikarenakan algoritma otomasi yang sangatlah canggih akan memakan waktu komputasi yang sangatlah banyak sehingga permainan tersebut tidak akan berjalan secara lancar. Bayangkan saja jika fitur otomasi tersebut harus memakan

waktu sebesar 30 menit untuk menghasilkan suatu aksi yang dianggap paling optimal, hal tersebut tentunya akan memperlambat keberlangsungan *game*-nya dan pemain akan merasa lebih sungkan untuk menggunakan fitur tersebut atau bahkan memainkan *game*-nya.

Umumnya untuk mengoptimasi waktu yang digunakan untuk menghasilkan aksi yang seoptimal mungkin, *game-game* dapat menggunakan algoritma *greedy*. Algoritma *greedy* tersebut biasanya seperti, mengutamakan menyerang musuh terlemah, selalu menggunakan *skill* jika tersedia, dan sebagainya. Strategi ini umumnya dapat mempermudah pekerjaan para pengembang *game* tersebut, serta sebagai strategi andalan yang dapat digunakan untuk mencari solusi dari masalah yang baru pertama kali ditemui.

II. TEORI DASAR

A. *Algoritma Greedy*

Algoritma *greedy* adalah algoritma yang identik dengan menghasilkan solusi yang terbaik pada saat itu tanpa memikirkan konsekuensi dari solusi tersebut. Umumnya algoritma *greedy* dapat digunakan sebagai pendekatan untuk menyelesaikan masalah baru yang belum pernah ditemui dengan cara-cara yang diketahui sebelumnya, hasilnya tentunya tidak dijamin optimal, namun akan lebih baik dari rata-rata karena solusi yang diambil merupakan yang terbaik dari kondisi awal.

Algoritma *greedy* dapat dibagi menjadi beberapa elemen penyusun:

- Himpunan kandidat, berisi kandidat yang dapat dipilih setiap langkahnya.
- Himpunan solusi, berisi kandidat yang telah dipilih (dianggap sebagai solusi yang telah diambil).
- Fungsi solusi, fungsi untuk menentukan apakah solusi yang telah diambil telah menghasilkan solusi atau tidak.
- Fungsi seleksi, fungsi untuk memilih solusi berdasarkan strategi.
- Fungsi kelayakan, fungsi untuk mengecek apakah kandidat solusi yang akan diambil dapat dimasukkan ke dalam himpunan solusi.
- Fungsi obyektif, fungsi yang digunakan untuk memaksimalkan atau meminimalkan solusi yang diinginkan.

Penyelesaian dengan algoritma *greedy*

- Strategi *greedy*: Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa.

```
function CoinExchange(C : himpunan_koin, A : integer) → himpunan_solusi
{ mengembalikan koin-koin yang total nilainya = A, tetapi jumlah koinnya minimum }
Deklarasi
  S : himpunan_solusi
  x : koin
Algoritma
S ← {}
while (∑(nilai semua koin di dalam S) ≠ A) and (C ≠ {} ) do
   x ← koin yang mempunyai nilai terbesar
   C ← C – {x}
   if (∑(nilai semua koin di dalam S) + nilai koin x ≤ A then
      S ← S ∪ {x}
   endif
endwhile
if (∑(nilai semua koin di dalam S) = A  then
   return S
else
   write('tidak ada solusi')
endif
```

Gambar 2.1. Contoh algoritma greedy dalam persoalan penukaran uang (sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

Tentunya komponen tersebut tidaklah harus dipenuhi secara eksplisit dalam pengimplementasian algoritma *greedy*, namun komponen-komponen tersebut umumnya terdapat pada algoritma *greedy*.

B. *Mekanisme Honkai Star Rail*

- Komposisi tim

  Dalam pertempuran, pemain hanya bisa menggunakan 1 tim yang terdiri dari maksimal 4 karakter dan minimum 1 karakter. Umumnya semakin banyak karakter dalam sebuah tim, maka pertempuran akan lebih mudah. Tidak ada Batasan untuk peran atau elemen maksimal dalam sebuah tim, sehingga pemain bisa membuat tim yang terdiri dari 4 karakter dengan peran *abundance* dengan elemen *physical* jika pemain memiliki karakter-karakternya.

- Posisi karakter

  Penempatan karakter-karakter dalam tim tersebut juga dapat berpengaruh secara tidak langsung pada *damage* yang diterima oleh karakter-karakter di tim kita selama pertempuran. Hal tersebut dikarenakan ada beberapa serangan atau kemampuan musuh yang menyerang 3 karakter sekaligus. Sehingga penempatan karakter-karakternya haruslah strategis, seperti tidak menempatkan karakter-karakter yang tidak memiliki HP (*health points*) berdekatan dengan satu sama lain, agar *damage* yang diterima bisa diminimalisasi.

- Urutan aksi

  Setiap karakter dan musuh memiliki atribut *speed*, yang menentukan kecepatan dari karakter tersebut dalam pertempuran. *Speed* tersebut menentukan berapa banyak aksi yang bisa dilakukan oleh karakter tersebut, serta urutan karakter tersebut dalam pertempuran. Terdapat beberapa kemampuan karakter maupun kemampuan musuh yang dapat memengaruhi

atribut ini, sehingga karakter/musuh yang dipengaruhi tersebut harus menunda atau mempercepat aksinya.

- Definisi *turn*

  Turn adalah tahapan dimana semua karakter dan semua musuh dalam pertempuran telah melakukan minimal 1 aksi. *Turn* ini penting karena beberapa kemampuan karakter didasari oleh *turn*, contohnya seperti memperkuat serangan karakter selama 2 *turn*, dan sebagainya.

- Kelemahan musuh

  Musuh selalu memiliki atribut *weakness* yang dapat digunakan pemain untuk memberikan *damage* yang lebih besar kepada musuh tersebut. Kelemahan ini dimanfaatkan di *break effect*.

- *Damage calculation*

  Damage yang dapat dihasilkan oleh karakter bergantung pada beberapa atribut seperti *base attack* karakter, *crit rate* karakter, *crit damage* karakter, *defense* musuh, kelemahan musuh, dan sebagainya.

- *Break effect*

  Musuh khususnya memiliki atribut tambahan, atribut tersebut bernama *toughness*. *Toughness* bisa dianggap sebagai pelindung yang dimiliki oleh musuh, namun juga bisa digunakan pemain untuk menghasilkan *damage* yang lebih besar. Jika *toughness* dari musuh tersebut habis, maka akan *break effect* akan terpicu, sehingga musuh tersebut akan menerima *damage* yang sangat besar berdasarkan serangan yang menguras *toughness* tersebut. Untuk menguras *toughness*, pemain dapat menyerang musuh tersebut dengan menggunakan karakter yang memiliki elemen yang sama dengan kelemahan musuh tersebut.

- *Skill points*

  Skill points adalah penanda berapa *skill* yang dapat digunakan. Umumnya saat memulai pertempuran, *skill points* dimulai dari 4. Jika *skill points* habis, maka karakter manapun tidak dapat menggunakan *skill*-nya. Untuk mengisi *skill points*, pemain dapat melakukan *basic attack* pada musuh manapun.

- *Basic attack*

  Basic attack adalah serangan biasa yang dapat dilakukan oleh karakter ke sebuah musuh. *Basic attack* dapat mengurangi *toughness*, menambah *skill points*, dan memberikan energi pada *ultimate* karakter tersebut. Jika karakter melakukan *basic attack*, maka giliran karakter pada *turn* tersebut akan terpakai.

- *Skill*

  Skill adalah kemampuan khusus yang dimiliki oleh karakter tersebut. *Skill* sangatlah beragam, mulai dari memberikan *damage* ke musuh, menyembuhkan *HP* karakter, memberikan *damage* tambahan, dan sebagainya. Untuk menggunakan *skill*, maka 1 *skill point* akan terpakai. *Skill* juga memberikan energi pada *ultimate* karakter tersebut serta mengurangi *toughness* musuh (jika *skill* tersebut menyerang musuh). Jika karakter menggunakan *skill*, maka giliran karakter pada *turn* tersebut akan terpakai.

- *Ultimate*

  Ultimate adalah kemampuan spesial yang harus di-*charge* terlebih dahulu oleh karakter tersebut. Karakter dapat mengisi energi *ultimate-nya* jika karakter tersebut diserang atau menyerang musuh. Uniknya *ultimate* ini dapat digunakan oleh karakter tersebut tanpa memikirkan gilirannya, bahkan karakter lain dapat menggunakan *ultimate*-nya saat giliran karakter lain. Tentunya pengisian energi tersebut tidaklah sama untuk setiap karakter, ada beberapa karakter yang cepat dalam mengisi energinya dan ada beberapa karakter yang sangatlah lama dalam mengisi energinya.

III. Strategi Greedy pada Pertempuran HSR

Pada pengimplementasiannya, strategi-strategi *greedy* yang dapat diterapkan oleh algoritmanya adalah:

- Jika ada satu atau lebih karakter yang bisa menggunakan *ultimate*, maka gunakanlah *ultimate* tersebut.
- Jika terdapat kesempatan untuk menyerang musuh, maka seranglah musuh yang memiliki persentase *HP* terkecil. Jika persentase *HP*-nya sama semua, maka seranglah yang memiliki *HP* terkecil.
- Jika ada kesempatan untuk memperkuat *teammate*, maka pilihlah *teammate* yang memiliki *attack* terbesar.
- Jika *skill* tersedia, maka selalu gunakan *skill*.
- Jika terdapat kesempatan untuk menyembuhkan karakter, maka lakukan jika terdapat karakter yang persentase *hp*-nya <= 80%.

Jika dipetakan ke komponen-komponen algoritma *greedy*, maka komponen-komponennya adalah sebagai berikut:

- Himpunan kandidat

  Berisi karakter-karakter yang dimiliki, aksi-aksi yang bisa dilakukan, dan mekanisme dasar HSR.

- Himpunan solusi

  Berisi keterangan aksi apa yang dilakukan oleh suatu karakter.

- Fungsi solusi

  Memeriksa apakah aksi-aksi yang diambil berujung pada semua musuh berhasil dikalahkan.

- Fungsi seleksi

  Memilih apakah *ultimate* dan *skill* tersedia.

- Fungsi kelayakan

  Memeriksa apakah aksi tersebut menyalahi strategi *greedy* sebelumnya, serta mengecek apakah *skill* dan *ultimate* yang digunakan memiliki target yang tepat.

- Fungsi obyektif

  Menghitung jumlah aksi yang dilakukan.

IV. Implementasi dan Eksperimen

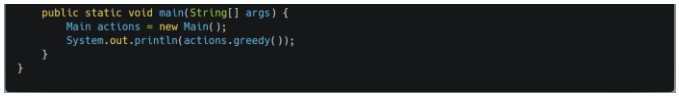
Berikut adalah cuplikan program sederhana yang digunakan untuk menyimulasikan proses:

```java
import java.util.ArrayList;

import Entity.Characters;
import Entity.Enemies;
import Entity.Healer;
import Entity.Karakter.Natasha;
import Entity.Karakter.Sushang;
import Entity.Musuh.SilvermaneGunner;

public class Main {
    private int skillPoints;
    private ArrayList<Characters> characters;
    private ArrayList<Enemies> enemies;

    public Main() {
        // banyak skill yang bisa digunakan
        this.skillPoints = 4;

        // inisialisasi karakter-karakternya
        characters = new ArrayList<>();
        enemies = new ArrayList<>();

        characters.add(new Sushang());
        characters.add(new Natasha());

        enemies.add(new SilvermaneGunner(1));
    }
```

```java
    public String greedy() {
        int actionstaken = 0;
        String actions = "";

        // greedy 1: jika ada satu atau lebih karakter yang bisa mengeluarkan ulti, maka keluarkan
        // greedy 2: selalu target musuh yang memiliki persentase HP terkecil
        // greedy 3: jika buffer, maka buff teammate yang memiliki attack terbesar
        // greedy 4: jika memiliki skill, maka selalu pakai
        // greedy 5: jika memiliki skill heal, hanya lakukan heal ketika ada teammate yang persentase HP-nya <= 80%

        while (enemies.size() > 0) {
            actions += "status tim:\n";
            for (Characters c: characters) {
                actions += c.getName() + " (HP: " + c.getHealth() + ", Attack: " + c.getAtt() + ")\n";
            }
            actions += "\n";

            actions += "status musuh:\n";
            for (Enemies c: enemies) {
                actions += c.getName() + " (HP: " + c.getHealth() + ")\n";
            }
            actions += "\n";
            // menerapkan greedy 1
            for (Characters c: characters) {
                if (c instanceof Healer) {
                    if (c.ultiAvailable()) {
                        // menerapkan greedy 5
                        int i = 0;
                        int idxMin = 0;
                        Double minRatio = (double) 100;
                        for (Characters e: characters) {
                            if (e.getHealthPercentage() < minRatio) {
                                minRatio = e.getHealthPercentage();
```

```java
                                idxMin = i;
                            }

                            i++;
                        }

                        if (minRatio <= 80) {
                            actionstaken++;
                            actions += c.getName() + " menyembuhkan " + characters.get(idxMin).getName() + " dengan ultimate\n";
                            (c).ulti(characters.get(idxMin));
                            c.resetEnergy();
                        }
                    }
                } else {
                    if (c.ultiAvailable()) {
                        // jika ulti menargetkan musuh
                        if (c.getUltiTarget()) {
                            // menerapkan greedy 2
                            int i = 0;
                            int idxMin = 0;
                            Double minRatio = (double) 100;
                            for (Enemies e: enemies) {
                                if (e.getHealthPercentage() < minRatio) {
                                    minRatio = e.getHealthPercentage();
                                    idxMin = i;
                                }

                                i++;
                            }

                            // jika ratio sama semua
                            i = 0;
                            int minHP = 999999999;
```

```java
                            if (minRatio == 100) {
                                for (Enemies e: enemies) {
                                    if (e.getHealth() < minHP) {
                                        minHP = e.getHealth();
                                        idxMin = i;
                                    }

                                    i++;
                                }
                            }

                            actionstaken++;
                            actions += c.getName() + " menyerang " + enemies.get(idxMin).getName() + " dengan menggunakan ultimate\n";
                            actions += c.ulti(enemies.get(idxMin));
                            c.resetEnergy();

                            if (enemies.get(idxMin).getHealthPercentage() <= 0) {
                                enemies.remove(idxMin);
                            }
                        } else { // jika tidak
                            // menerapkan greedy 3
                            int i = 0;
                            int idxMax = 0;
                            int maxAtt = 0;
                            for (Characters e: characters) {
                                if (e.getAtt() > maxAtt) {
                                    maxAtt = e.getAtt();
                                    idxMax = i;
                                }

                                i++;
                            }

                            actionstaken++;
                            actions += c.getName() + " memperkuat " + characters.get(idxMax).getName() + " dengan menggunakan ultimate\n";
                            c.ulti(characters.get(idxMax));
                            c.resetEnergy();
                        }
                    }
                }
            }

            Characters current = characters.remove(0);

            //  menerapkan greedy 4
            if (skillPoints > 0) {
                if (!(current instanceof Healer)) {
                    if (current.getSkillTarget()) { // jika skill menargetkan musuh
                        // menerapkan greedy 2
                        int i = 0;
                        int idxMin = 0;
                        Double minRatio = (double) 100;
                        for (Enemies e: enemies) {
                            if (e.getHealthPercentage() < minRatio) {
                                minRatio = e.getHealthPercentage();
                                idxMin = i;
                            }

                            i++;
                        }

                        // jika ratio sama semua
                        i = 0;
                        int minHP = 999999999;
                        if (minRatio == 100) {
                            for (Enemies e: enemies) {
```



Gambar 4.1. – 4.10. Cuplikan program sederhana untuk menyimulasikan solusi (sumber: arsip penulis)

Berikut adalah cuplikan kelas-kelas yang digunakan untuk menyimulasikan solusi:

- Beings.java

```java
package Entity;

public class Beings {
    protected int health;
    protected int basehealth;
    protected int speed;
    protected int attack;
    protected String name;

    public Beings(int health, int speed, int attack) {
        this.health = health;
        this.basehealth = health;
        this.speed = speed;
        this.attack = attack;
    }

    public void dealDamage(int x) {
        this.health -= x;
    }

    public String getName() {
        return this.name;
    }

    public double getHealthPercentage() {
        return ((double)this.health/(double)this.basehealth)*100;
    }

    public int getHealth() {
        return this.health;
    }

    public int getSpd() {
        return this.speed;
    }

    public int getAtt() {
        return this.attack;
    }

    public void addHP(int x) {
        this.health += x;
    }
}
```

Gambar 4.11. Cuplikan program sederhana untuk kelas dasar beings (sumber: arsip penulis)

- Characters.java

```java
package Entity;

public abstract class Characters extends Beings {
    protected int toughnessDamage;
    protected int energy;
    protected boolean skillTargetEnemies = true;
    protected boolean ultiTargetEnemies = true;

    public Characters(int toughnessDamage, int health, int speed, int attack) {
        super(health, speed, attack);
        this.toughnessDamage = toughnessDamage;
        this.energy = 0;
    }

    public void setTargetSkillToCharacters() {
        this.skillTargetEnemies = false;
    }

    public void rechargeEnergy() {
        this.energy++;
    }

    public void resetEnergy() {
        this.energy = 0;
    }

    public boolean ultiAvailable() {
        return this.energy == 4;
    }

    public boolean getSkillTarget () {
        return this.skillTargetEnemies;
    }

    public boolean getUltiTarget () {
        return this.ultiTargetEnemies;
    }

    public String skill (Enemies e) {
        return "";
    }

    public void skill (Characters e) {

    }

    public String ulti (Enemies e) {
        return "";
    }

    public void ulti (Characters e) {

    }

    public String attack(Enemies e) {
        return "";
    }
}
```

Gambar 4.12. Cuplikan program sederhana untuk kelas dasar characters (sumber: arsip penulis)

- Enemies.java

```java
package Entity;

public class Enemies extends Beings {
    private int toughness;
    private int baseToughness;

    public Enemies(int toughness, int health, int speed, int attack) {
        super(health, speed, attack);
        this.toughness = toughness;
        this.baseToughness = toughness;
    }

    public void dealDamageToToughness(int x) {
        if (this.toughness < x) {
            toughness = 0;
        } else {
            toughness -= x;
        }
    }

    public int getToughness() {
        return this.toughness;
    }

    public void refreshToughness() {
        this.toughness = this.baseToughness;
    }
}
```

Gambar 4.13. Cuplikan program sederhana untuk kelas dasar enemies (sumber: arsip penulis)

- Healer.java

```java
package Entity;

public interface Healer {
    public void heal (Characters e);
}
```

Gambar 4.14. Cuplikan program sederhana untuk kelas interface healer (sumber: arsip penulis)

Berikut adalah cuplikan beberapa karakter yang dibuat:

```java
package Entity.Karakter;

import Entity.Healer;
import Entity.Characters;
import Entity.Enemies;
import Entity.Elemen.Physical;

public class Natasha extends Characters implements Physical, Healer {
    public Natasha() {
        super(30, 1200, 120, 1300);
        this.name = "Natasha";
        this.setTargetSkillToCharacters();
    }

    public void heal (Characters e) {
        e.addHP((int) (basehealth*0.1225 + 375));
    }

    @Override
    public void ulti(Characters e) {
        e.addHP((int) (basehealth*0.1225 + 375));
    }

    @Override
    public String attack(Enemies e) {
        boolean breakeffect = false;
        if (e instanceof Physical) {
            e.dealDamage((int)(attack * 0.8));
            e.dealDamageToToughness(this.toughnessDamage);
            if (e.getToughness() <= 0) {
                e.dealDamage((int)(attack * 0.8 * 2)); // break effect
                breakeffect = true;
                e.refreshToughness();
            }
        } else {
            e.dealDamage((int)(attack * 0.8));
        }

        this.rechargeEnergy();

        if (breakeffect) {
            return "(break)\n";
        } else {
            return "";
        }
    }
}
```

Gambar 4.15. Cuplikan kode sederhana untuk karakter Natasha (sumber: arsip penulis)

```java
package Entity.Karakter;

import Entity.Characters;
import Entity.Enemies;
import Entity.Elemen.Physical;

public class Sushang extends Characters implements Physical {
    public Sushang() {
        super(30, 1200, 120, 1300);
        this.name = "Sushang";
    }

    @Override
    public String skill (Enemies e) {
        boolean breakeffect = false;
        if (e instanceof Physical) {
            e.dealDamage((int)(attack * 1.57));
            e.dealDamageToToughness(this.toughnessDamage*2);
            if (e.getToughness() <= 0) {
                e.dealDamage((int)(attack * 1.57 * 2)); // break effect
                breakeffect = true;
                e.refreshToughness();
            }
        } else {
            e.dealDamage((int)(attack * 1.57));
        }

        this.rechargeEnergy();

        if (breakeffect) {
            return "(break)\n";
        } else {
            return "";
        }
    }

    @Override
    public String ulti(Enemies e) {
        boolean breakeffect = false;
        if (e instanceof Physical) {
            e.dealDamage((int)(attack * 0.256));
            e.dealDamageToToughness(this.toughnessDamage);
            if (e.getToughness() <= 0) {
                e.dealDamage((int)(attack * 0.256 * 2)); // break effect
                breakeffect = true;
                e.refreshToughness();
            }
        } else {
            e.dealDamage((int)(attack * 0.256));
        }

        this.rechargeEnergy();

        if (breakeffect) {
            return "(break)\n";
        } else {
            return "";
        }
    }
```

```java
    @Override
    public String attack(Enemies e) {
        boolean breakeffect = false;
        if (e instanceof Physical) {
            e.dealDamage((int)(attack * 0.8));
            e.dealDamageToToughness(this.toughnessDamage);
            if (e.getToughness() <= 0) {
                e.dealDamage((int)(attack * 0.8 * 2)); // break effect
                breakeffect = true;
                e.refreshToughness();
            }
        } else {
            e.dealDamage((int)(attack * 0.8));
        }

        this.rechargeEnergy();

        if (breakeffect) {
            return "(break)\n";
        } else {
            return "";
        }
    }
}
```

Gambar 4.16.-4.17. Cuplikan kode sederhana untuk karakter Sushang (sumber: arsip penulis)

Berikut adalah cuplikan musuh yang dibuat:

```java
package Entity.Musuh;

import Entity.Enemies;
import Entity.Elemen.Ice;
import Entity.Elemen.Physical;

public class SilvermaneGunner extends Enemies implements Physical, Ice {
    public SilvermaneGunner(int x) {
        super(90, 15000, 90, 800);
        this.name = "Silvermane Gunner " + Integer.toString(x);
    }
}
```

Gambar 4.18. Cuplikan kode sederhana untuk musuh SilvermaneGunner (sumber: arsip penulis)

Berikut adalah contoh *output* yang didapat dari cuplikan program di atas:

```
status tim:
Sushang (HP: 1200, Attack: 1300)
Natasha (HP: 1200, Attack: 1300)

status musuh:
Silvermane Gunner 1 (HP: 15000)

Sushang menyerang Silvermane Gunner 1 dengan menggunakan skill
status tim:
Natasha (HP: 1200, Attack: 1300)
Sushang (HP: 1200, Attack: 1300)

status musuh:
Silvermane Gunner 1 (HP: 12959)

Natasha menyerang Silvermane Gunner 1
(break)
status tim:
Sushang (HP: 1200, Attack: 1300)
Natasha (HP: 1200, Attack: 1300)

status musuh:
Silvermane Gunner 1 (HP: 9839)

Sushang menyerang Silvermane Gunner 1 dengan menggunakan skill
status tim:
Natasha (HP: 1200, Attack: 1300)
Sushang (HP: 1200, Attack: 1300)

status musuh:
Silvermane Gunner 1 (HP: 7798)

Natasha menyerang Silvermane Gunner 1
(break)
status tim:
Sushang (HP: 1200, Attack: 1300)
Natasha (HP: 1200, Attack: 1300)

status musuh:
Silvermane Gunner 1 (HP: 4678)
```

Gambar 4.19.-4.20. Cuplikan solusi yang dihasilkan dari contoh program sederhana di atas (sumber: arsip penulis)

Berdasarkan hasil yang didapat, bisa dilihat bahwa jumlah langkah yang diambil adalah sebanyak 8 langkah. Karakter Sushang selalu menggunakan skill dikarenakan *greedy* yang telah dibuat sebelumnya, hal tersebut kemudian diperpanjang oleh karakter Natasha yang selalu mengisi *skill points* dengan melakukan *basic attack*.

Program diatas memiliki kompleksitas O(N^2), dengan N adalah jumlah karakter yang ada di pertempuran. Mengingat bahwa permainan ini maksimal hanya memiliki 4 buah karakter maksimal, dan musuh yang bisa dibilang sedikit, maka bisa dibilang algoritma *greedy* efisien jika diimplementasikan di fitur ini.

V. Kesimpulan

Algoritma *greedy* sangatlah bagus dalam segi efisiensi dalam optimasi fitur *battle automation* dalam *game* HSR. Tentunya solusi yang dihasilkan tidak efisien dan mungkin saja tidak tercapai. Namun kecepatan umumnya adalah aspek terpenting dalam fitur otomasi *battle automation*, namun akan lebih baik jika kita bisa menjamin langkah yang dihasilkan akan lebih baik dari rata-rata. Sehingga penggunaan algoritma *greedy* pada fitur otomasi *battle automation* adalah pilihan yang tepat. Jika dibandingkan dengan algoritma lain seperti misalnya BFS atau UCS, maka jelas algoritma contoh di atas jauh lebih cepat dalam memberikan solusinya, walaupun solusi tersebut belum terjamin tepat.

VI. Saran

Akan lebih baik jika strategi algoritma *greedy* yang digunakan didasari oleh banyak eksperimen yang dilakukan, strategi tersebut akan lebih efisien jika kita bisa menemukan batasan-batasan yang tepat.

Pranala Video Youtube

https://youtu.be/QFbrAlvmkIM

Pranala Link Github

https://github.com/Ainzw0rth/HSRBattleAutomationAnalysis.git

Ucapan Terima Kasih

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini dengan tepat waktu. Penulis juga berterima kasih sebesar-besarnya kepada para dosen pengampuh mata kuliah IF2211, terutama kepada Bapak Dr. Ir. Rinaldi Munir sebagai dosen pengampuh untuk kelas 01 yang telah memberikan pelajaran dan pengalaman yang berkesan pada semester ini. Tidak lupa, penulis juga berterima kasih kepada keluarga dan teman-teman penulis atas dukungannya selama ini.

References

[1] Munir, Rinaldi. 2021. "Algoritma *Greedy* (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf, diakses 20 Mei 2023.

[2] Munir, Rinaldi. 2021. "Algoritma *Greedy* (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf, diakses 20 Mei 2023.

[3] Munir, Rinaldi. 2021. "Algoritma *Greedy* (Bagian 3)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf, diakses 20 Mei 2023.

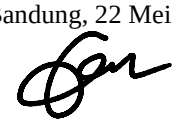
[4] Prydwen.gg. 2023. "Honkai: Star Rail Damage Formula". https://www.prydwen.gg/star-rail/guides/damage-formula/, diakses 20 Mei 2023

[5] https://honkai-star-rail.fandom.com/wiki/Toughness#Toughness_Value_and_Damage, diakses 20 Mei 2023

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023



Louis Caesa Kesuma 13521069