

# LAPORAN TUGAS BESAR

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan “*Worms*”

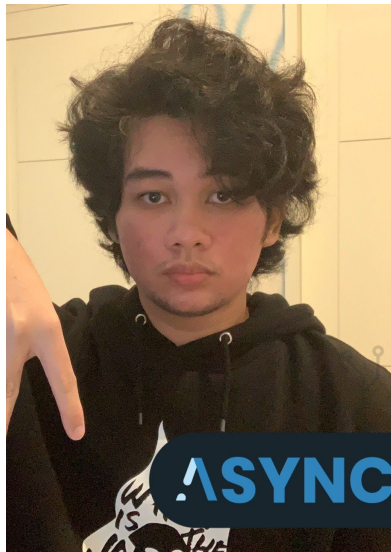
Dibuat dalam rangka:  
Tugas Besar 2 IF-2211 Strategi Algoritma

Oleh:  
Gemdev abal-abal



**Fabian Savero Diaz  
Pranoto**

13519140



**Fadel Ananda Dotty**

13519146



**Tanur Rizaldi Raharjo**

13519214

**PROGRAM STUDI TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2020**

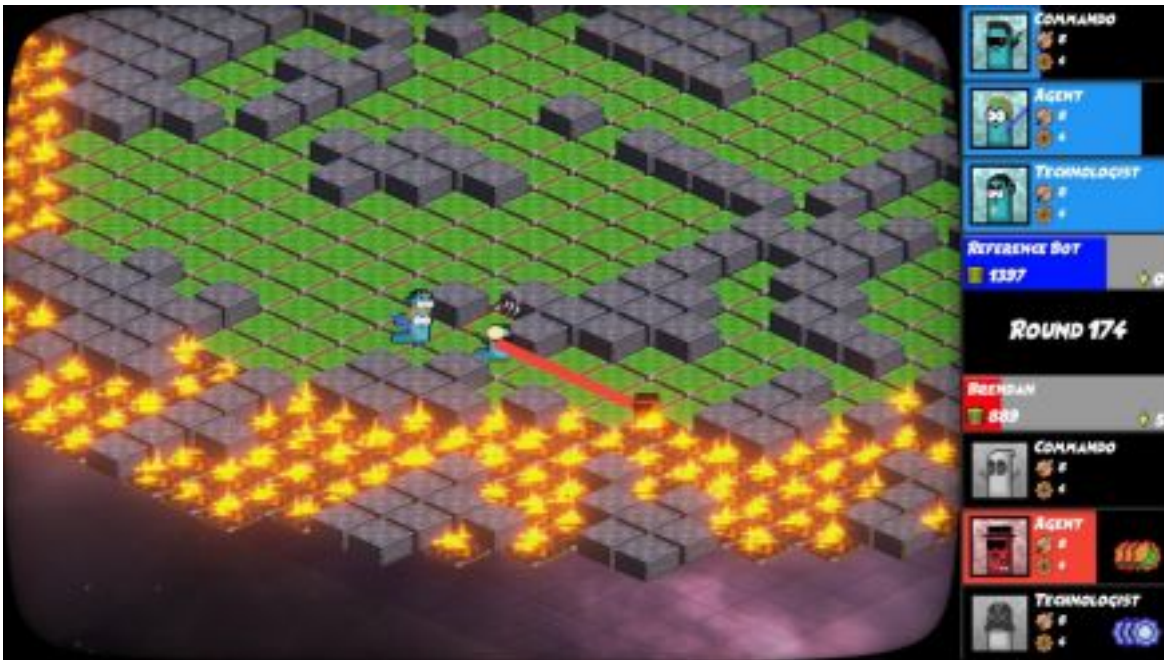
## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB I</b>	
<b>PENDAHULUAN</b>	<b>2</b>
Deskripsi Tugas	2
<b>BAB II</b>	
<b>LANDASAN TEORI</b>	<b>5</b>
Algoritma Greedy	5
<b>BAB III</b>	
<b>PEMANFAATAN STRATEGI GREEDY</b>	<b>8</b>
Elemen Algoritma Greedy pada Permainan	8
<b>BAB IV</b>	
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>15</b>
Implementasi	15
Struktur Data	17
Pengujian	18
<b>BAB V</b>	
<b>KESIMPULAN DAN SARAN</b>	<b>22</b>
KESIMPULAN	22
SARAN	22

## BAB I PENDAHULUAN

### 1.1 Deskripsi Tugas

Worms adalah sebuah *turned-based game* yang memerlukan strategi untuk memenangkannya. Setiap pemain akan memiliki 3 *worms* dengan perannya masing-masing. Pemain dinyatakan menang jika ia berhasil bertahan hingga akhir permainan dengan cara mengeliminasi pasukan *worms* lawan menggunakan strategi tertentu.



Gambar 1. Contoh tampilan permainan *Worms*

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* untuk mengimplementasikan permainan *Worms*. *Game engine* dapat diperoleh pada laman berikut <https://github.com/EntelectChallenge/2019-Worms>.

Tugas mahasiswa adalah mengimplementasikan seorang “pemain” *Worms*, dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan seorang “pemain” tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter bot* di dalam *starter pack* pada laman berikut ini:

(<https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2>)

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Worms* pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan berukuran 33x33 *cells*. Terdapat 4 tipe *cell*, yaitu *air*, *dirt*, *deep space*, dan *lava* yang masing-masing memiliki karakteristik berbeda. *Cell* dapat memuat

- powerups* yang bisa diambil oleh *worms* yang berada pada *cell* tersebut.
2. Di awal permainan, setiap pemain akan memiliki 3 pasukan *worms* dengan peran dan nilai *health points* yang berbeda, yaitu:
    - a. *Commando*
    - b. *Agent*
    - c. *Technologist*
  3. Pada setiap *round*, masing-masing pemain dapat memberikan satu buah *command* untuk pasukan *worm* mereka yang masih aktif (belum tereliminasi). Berikut jenis-jenis *command* yang ada pada permainan:
    - a. *Move*
    - b. *Dig*
    - c. *Shot*
    - d. *Do Nothing*
    - e. *Banana Bomb*
    - f. *Snowball*
    - g. *Select*
  4. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. *Command* juga akan dieksekusi sesuai urutan prioritas tertentu.
  5. Beberapa *command*, seperti *shot* dan *banana bomb* dapat memberikan *damage* pada *worms* target yang terkena serangan, sehingga mengurangi *health points*nya. Jika *health points* suatu *worm* sudah habis, maka *worm* tersebut dinyatakan tereliminasi dari permainan.
  6. Permainan akan berakhir ketika salah satu pemain berhasil mengeliminasi seluruh pasukan *worms* lawan atau permainan sudah mencapai jumlah *round* maksimum (400 rounds).

Adapun peraturan yang lebih lengkap dari permainan *Worms*, dapat dilihat pada laman <https://github.com/EntelectChallenge/2019-Worms/blob/develop/game-engine/game-rules.md>

### Spesifikasi tugas:

Pada tugas besar kali ini, anda diminta untuk membuat sebuah *bot* untuk bermain permainan *Worms* yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

1. *Download latest release starter pack.zip* dari tautan berikut. <https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2>. 2.
- Untuk menjalankan permainan, kalian butuh beberapa *requirement* dasar sebagai berikut.
  - a. Java (minimal Java 8): <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>
  - b. IntelliJ IDEA: <https://www.jetbrains.com/idea/>
  - c. NodeJS: <https://nodejs.org/en/download/>
3. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk *Windows/Mac* dapat buka dengan *double-click*, Untuk *Linux* dapat menjalankan *command* “make

run”).

4. Secara *default*, permainan akan dilakukan diantara *reference bot* (*default*-nya berbahasa *JavaScript*) dan starter bot yang disediakan. Untuk mengubah hal tersebut, silahkan edit file “game-runner-config.json”. Anda juga dapat mengubah file “bot.json” untuk mengatur informasi terkait bot anda.
5. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di *starter-bot*. Ingat bahwa bot kalian harus menggunakan bahasa **Java** dan di-*build* menggunakan **IntelliJ**. **Dilarang** menggunakan kode program tersebut untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan *visualizer* berikut <https://github.com/dlweatherhead/entelect-challenge-2019-visualiser/releases/tag/v1.0f1> 7. Untuk referensi lebih lanjut, silahkan eksplorasi di [tautan berikut](#).

Strategi *greedy* yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mengeliminasi seluruh worms lawan dengan senjata dan skill yang sudah disediakan dalam permainan. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menyerang pasukan lawan dengan senjata dengan *hitpoint* / *damage* terbesar. Buatlah strategi *greedy* terbaik, karena setiap “pemain” dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (secara daring).

Strategi *greedy* harus dituliskan secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi *greedy* untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

## BAB II LANDASAN TEORI

### 2.1 Algoritma Greedy

Algoritma merupakan sebuah sekuens aksi yang digunakan untuk menyelesaikan suatu permasalahan yang didapatkan dengan cara mengubah input menjadi output dengan waktu tertentu yang terbatas. Terdapat berbagai macam algoritma yang sudah ditemukan untuk menyelesaikan berbagai permasalahan. Algoritma greedy merupakan salah satu dari sekian banyak algoritma yang ditemukan yang dapat digunakan untuk menyelesaikan berbagai macam permasalahan. Algoritma greedy menggunakan pendekatan yaitu mencari sebuah solusi langkah per langkah yang terurut dan dari setiap langkah akan mendapat sebuah solusi parsial yang akan berkembang menjadi solusi yang lengkap dan solusi dari masalah dapat ditemukan. Dari setiap langkah-langkah yang ditinjau terdapat beberapa pilihan solusi. Akan tetapi, pilihan yang dipilih harus memenuhi beberapa kriteria, yaitu:

1. Solusi yang dipilih harus layak dan memenuhi kriteria permasalahan
2. Solusi yang dipilih harus merupakan solusi yang terbaik dari pilihan solusi lain yang ada (mengambil pilihan terbaik tanpa memperhatikan konsekuensi ke depan)
3. Solusi yang sudah dipilih tidak dapat diubah kedepannya

Dengan menggunakan algoritma greedy, permasalahan yang dipecahkan selalu mendapatkan solusi paling optimal lokal. Solusi optimal lokal ini kemudian diharapkan dapat menjadi solusi optimal global (*globally optimal solution*) dari permasalahan yang dipecahkan. Akan tetapi, algoritma ini tidak selalu menghasilkan solusi optimal global. Walaupun demikian, terdapat beberapa permasalahan yang cocok diselesaikan dengan algoritma ini karena dapat memberikan aproksimasi solusi yang optimal.

Terdapat beberapa cara yang dapat menunjukkan bahwa permasalahan yang diselesaikan dengan algoritma greedy menghasilkan solusi yang optimal, yaitu:

1. Menggunakan induksi matematika yang menunjukkan setiap solusi dari langkah-langkah yang dilakukan oleh algoritma ini akan berkembang menuju solusi paling optimal.
2. Menunjukkan setiap langkah yang digunakan oleh algoritma ini menggunakan pendekatan terbaik seperti algoritma lainnya.
3. Menunjukkan bahwa solusi yang didapat dari hasil akhir algoritma ini merupakan solusi yang optimal.

Dalam algoritma greedy, terdapat beberapa elemen yang harus diperhatikan agar dapat membentuk solusi dari suatu permasalahan. Elemen-elemen tersebut yaitu:

1. Himpunan kandidat ( $C$ )  
Himpunan kandidat merupakan himpunan yang berisi kandidat yang akan dipilih dari setiap langkah.
2. Himpunan solusi ( $S$ )

Himpunan solusi merupakan himpunan yang berisi kandidat yang sudah dipilih dan tidak dapat diubah.

3. Fungsi solusi

Fungsi solusi berfungsi untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.

4. Fungsi seleksi (*selection function*)

Fungsi yang digunakan untuk memilih kandidat berdasarkan suatu strategi *greedy* tertentu yang bersifat heuristik.

5. Fungsi kelayakan (*feasible*)

Fungsi yang digunakan untuk menentukan apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (sesuai dengan *problem constraint*).

6. Fungsi objektif

Fungsi yang dipilih untuk meminimumkan atau memaksimumkan.

Algoritma greedy biasanya lebih banyak dipilih untuk menyelesaikan suatu permasalahan apabila dibandingkan dengan algoritma brute force karena waktu yang diperlukan untuk menyelesaikan suatu permasalahan lebih sedikit apabila dibandingkan dengan algoritma brute force. Contohnya saja dalam permasalahan *activity selection problem*. Permasalahan ini akan menghasilkan algoritma dengan kompleksitas waktu sebesar  $O(n \cdot 2^n)$  apabila diselesaikan dengan *exhaustive search*. Akan tetapi, permasalahan ini akan menghasilkan algoritma dengan kompleksitas waktu sebesar  $O(n)$  apabila diselesaikan dengan menggunakan algoritma greedy (dengan waktu pengurutan tidak diperhitungkan).

**function** *Greedy-Activity-Selector*( $s_1, s_2, \dots, s_n : \text{integer}, f_1, f_2, \dots, f_n : \text{integer}$ )  $\rightarrow$  set of integer  
 { Asumsi: aktivitas sudah diurut terlebih dahulu berdasarkan waktu selesai:  $f_1 \leq f_2 \leq \dots \leq f_n$  }

**Deklarasi**

$i, j, n : \text{integer}$

**A** : set of integer

**Algoritma:**

$n \leftarrow \text{length}(s)$

$A \leftarrow \{1\}$  { aktivitas nomor 1 selalu terpilih }

$j \leftarrow 1$

**for**  $i \leftarrow 2$  **to**  $n$  **do**

**if**  $s_i \geq f_j$  **then**

$A \leftarrow A \cup \{i\}$

$j \leftarrow i$

**endif**

**endif**

Gambar 2.1.1 Contoh Penyelesaian Permasalahan *Activity Selection* dengan Menggunakan Algoritma Greedy



## 2.2 Pemanfaatan *Game Engine* pada *Game Worms*

Pada worms, permainan dapat dijalankan dengan menjalankan program bernama run.bat. Untuk dapat melihat keberjalanan permainan dengan lebih maksimal, penulis dapat menjalankan program entelect-visualiser.exe lalu kemudian memilih pertandingan yang ingin dilihat.

Pada worms, penulis bermain sebagai starter-bots. Dalam folder starter-bots, telah disediakan berbagai macam template program dengan berbagai macam bahasa pemrograman. penulis dapat memilih salah satu dari berbagai bahasa tersebut untuk dimodifikasi sesuai dengan keinginan penulis.

Setelah memilih bahasa pemrograman yang diinginkan, untuk menambahkan pemain, penulis dapat memodifikasi file bernama game-runner-config.json. Dalam file tersebut, terdapat objek bernama player-a dan player-b. Untuk menambahkan/mengganti pemain, penulis dapat mengubah objek tersebut sesuai dengan letak bot yang akan ditambahkan.

Dalam folder command yang terdapat pada folder starter bots, penulis menambahkan beberapa program java yang dapat membuat worms bertipe agent menggunakan skill banana, worm bertipe technologist menggunakan snowball, dan program yang memungkinkan penulis melakukan seleksi terhadap worm. Penulis menambahkan program bernama BananaCommand.java, SnowBallCommand.java, dan SelectCommand.java. Penambahan program ini mengacu kepada gamerules.md agar bot dapat berperilaku seperti yang terdapat pada game rules tersebut.

Penulis juga menambahkan beberapa atribut pada program Worm.java dan MyWorm.java. Atribut yang diberikan mengacu kepada file state.json yang berada pada folder rounds di reference bot. Penambahan atribut ini berfungsi agar penulis dapat mengakses atribut-atribut tersebut dalam mengimplementasikan strategi greedy yang dipilih.

Penulis dapat menerapkan algoritma greedy ke dalam program yang telah dipilih oleh penulis. Dalam kasus ini, penulis memilih bot yang ditulis dengan menggunakan bahasa java. Penerapan algoritma greedy dapat ditulis dalam file bot.java yang terdapat di dalam folder starter-bots. Dengan mengubah isi bot.java, penulis dapat mengubah perilaku starter-bots sesuai dengan algoritma greedy yang telah dipilih dengan cara menambah method-method yang akan dipanggil pada method run().

Setelah mengubah isi file java dan menambahkan perilaku-perilaku sesuai dengan gamerules worms, penulis melakukan build dengan IDE IntelliJ. Setelah di build, penulis menjalankan program dengan cara menjalankan run.bat lalu melihatnya di visualizer.



### **BAB III**

### **PEMANFAATAN STRATEGI GREEDY**

#### **3.1. Elemen Algoritma Greedy pada Permainan**

1. Himpunan kandidat  
Himpunan kandidat pada persoalan worms dapat berupa aksi-aksi yang mungkin dilakukan oleh bot, misalnya *move*, *dig*, *shoot*, *do nothing*, *banana bomb*, *snowball*, dan *select* per ronde.
2. Himpunan solusi  
Himpunan solusi pada persoalan worms merupakan aksi-aksi yang sudah dilakukan oleh worms pada saat melakukan pertandingan per ronde.
3. Fungsi solusi  
Fungsi solusi pada persoalan worms dapat berupa fungsi yang melakukan pengecekan apakah poin yang didapat lebih tinggi dari poin musuh atau fungsi yang melakukan pengecekan apakah semua musuh sudah dikalahkan.
4. Fungsi seleksi  
Fungsi seleksi pada persoalan worms dapat berupa memprioritaskan suatu aksi dari aksi-aksi yang ada ketika diberikan suatu kondisi tertentu.
5. Fungsi kelayakan  
Fungsi kelayakan dapat berupa pemeriksaan terhadap aksi yang diberikan kepada bot player. Fungsi dapat memeriksa apakah aksi tersebut valid dan sesuai dengan peraturan permainan.
6. Fungsi objektif  
Fungsi objektif dapat berupa memaksimalkan suatu aksi yang akan dilakukan bot player. Misalnya saja *dig* atau *shoot*.

#### **3.2. Alternatif Solusi Greedy yang Mungkin pada Permainan**

Dalam melakukan penentuan strategi greedy yang akan dipakai untuk mencari solusi dari permasalahan worms, terdapat beberapa alternatif pilihan strategi greedy yang dapat digunakan. Beberapa diantaranya yaitu:

1. Memaksimalkan pencarian terhadap bot musuh terdekat, kemudian memindahkan bot player agar mendekat ke bot musuh lalu berusaha memberikan damage kepada bot tersebut.
2. Memaksimalkan poin yang didapat dengan cara berjalan ke arah *cell* bertipe *dirt*, kemudian melakukan perintah *dig* untuk memaksimalkan poin.
3. Memaksimalkan poin yang didapat dengan cara mencari *cell* bertipe *air*, kemudian terus berjalan sepanjang *cell type* tersebut.
4. Memaksimalkan poin yang didapat dengan cara membekukan bot musuh.
5. Meminimalkan pengurangan poin dengan cara tidak melakukan aksi yang *invalid*.
6. Memaksimalkan poin yang didapat dengan cara menghindari bot musuh sambil melakukan aksi *dig* apabila terdapat *cell* bertipe *dirt* dan membekukan musuh apabila terdapat bot musuh dalam range tersebut sampai ronde ke 400.

7. Meminimalkan *health loss* dengan cara menghindari *cell type* bertipe lava.

### 3.3. Analisis Efisiensi dari Solusi

Terdapat 2 solusi greedy yang dibuat yaitu mencari musuh terdekat dan memaksimalkan jumlah worm musuh yang dilempari banana bomb atau snowball. Berikut analisis kompleksitasnya:

#### 1. Cari Musuh Terdekat

```
private Direction nearestEnemyDirection(Position
currentWorm) {
    int minimalDistance = 10000;
    int minimalIndex = -1;

    // Cari worm terdekat
    for (int i = 0; i < this.opponent.worms.length; i++)
    {
        // Cek enemy yang masih hidup, jika mati
        // lanjutkan iterasi
        if (this.opponent.worms[i].health <= 0)
            continue;

        int temp = euclideanDistance(currentWorm,
this.opponent.worms[i].position);
        if (temp < minimalDistance &&
this.opponent.worms[i].health > 0) {
            minimalIndex = i;
        }
    }

    return resolveDirection(currentWorm,
this.opponent.worms[minimalIndex].position);
}
```

Terlihat dari kode diatas, fungsi `nearestEnemyDirection` memiliki satu loop yang selalu mengiterasi sebesar panjang array worms musuh sehingga kompleksitas waktu algoritmanya adalah  $O(n)$ .

#### 2. Lempar Skill

```

// Fungsi lempar banana bomb atau snowball
private Command throwSkill(MyWorm selectedWorm) {
    int minimalDistance = 10000;
    int minimalIndex = -1;

    // Cari worm terdekat
    for (int i = 0; i < this.opponent.worms.length; i++)
    {
        int temp =
euclideanDistance(selectedWorm.position.x,
selectedWorm.position.y, this.opponent.worms[i].position.x,
this.opponent.worms[i].position.y);

        if (temp < minimalDistance &&
this.opponent.worms[i].health > 0) {
            minimalIndex = i;
        }
    }

    // Apabila health worm > 50 cek apakah musuh
bergerombolan. Jika iya
        // lempar banana bomb atau snowball. Apabila health
<= 50, langsung lempar tanpa memikirkan
        // musuh bergerombol atau tidak
        if (euclideanDistance(selectedWorm.position,
this.opponent.worms[minimalIndex].position) <= 5) {
            if (selectedWorm.snowball != null &&
selectedWorm.snowball.count > 0) {
                return Snowball(selectedWorm,
this.opponent.worms[minimalIndex]);
            } else if (selectedWorm.bananaBomb != null &&
selectedWorm.bananaBomb.count > 0) {
                return
BananaBomb(this.opponent.worms[minimalIndex].position);
            }
        }
        return null;
    }
}

```

```
}
```

Dari cuplikan kode diatas, fungsi melakukan loop melalui jumlah elemen di array worm musuh sebesar n sehingga memiliki kompleksitas algoritma  $O(n)$ .

### 3. Lempar Snowball

```
// Cek snowball
private Command Snowball(MyWorm selectedWorm, Worm
enemyWorm) {
    // Throwing snowball with conditional check
    List<Cell> surroundingBlocks =
getSurroundingCells(enemyWorm.position.x,
enemyWorm.position.y);
    for (int i = 0; i < surroundingBlocks.size(); i++) {
        Position targetPos = new Position();
        targetPos.x = surroundingBlocks.get(i).x;
        targetPos.y = surroundingBlocks.get(i).y;
        if (checkSnowballIsGroup(targetPos) &&
euclideanDistance(selectedWorm.position, targetPos) <= 5 &&
enemyWorm.roundsUntilUnfrozen < 2)
            return new SnowBallCommand(targetPos.x,
targetPos.y);
    }

    if (enemyWorm.roundsUntilUnfrozen < 2) {
        return new SnowBallCommand(enemyWorm.position.x,
enemyWorm.position.y);
    }

    return null;
}

// Greedy Snowball
private boolean checkSnowballIsGroup (Position enemyWorm)
{
    // Checking whether AOE centered on position contain
multiple worm or not
    List<Cell> surroundingBlocks =
getSurroundingCells(enemyWorm.x, enemyWorm.y);
```

```

        int enemyCount = 0;

        for (int i = 0; i < surroundingBlocks.size(); i++) {
            for (int j = 0; j < this.opponent.worms.length;
j++) {
                // if (this.opponent.worms[j].position !=
enemyWorm) {
                    if (surroundingBlocks.get(i).x ==
this.opponent.worms[j].position.x &&
                        surroundingBlocks.get(i).y ==
this.opponent.worms[j].position.y) {
                        enemyCount++;
                    }
                    if (enemyWorm.x ==
this.opponent.worms[j].position.x &&
                        enemyWorm.y ==
this.opponent.worms[j].position.y)
                        enemyCount++;
                // }
            }
        }

        if (enemyCount > 1)
            return true;
        else
            return false;
    }

```

Dari cuplikan kode diatas, fungsi melakukan loop melalui jumlah elemen di array worm musuh sebesar n sehingga memiliki kompleksitas algoritma  $O(n)$ .

#### 4. Lempar BananaBomb

```

// Cek banana
private Command BananaBomb(Position enemyPosition) {
    // Throwing banana with conditional check
    if (currentWorm.health > 50 &&
checkBananaIsGroup(enemyPosition)) {
        return new BananaCommand(enemyPosition.x,
enemyPosition.y);
    } else if (currentWorm.health <= 50) {
        return new BananaCommand(enemyPosition.x,
enemyPosition.y);
    }

    return null;
}

// Greedy Banana
private boolean checkBananaIsGroup (Position enemyWorm) {
    // Checking whether AOE centered on position contain
multiple worm or not
    for (int i = 0; i < this.opponent.worms.length; i++)
    {
        if (this.opponent.worms[i].position != enemyWorm)
        {
            if (euclideanDistance(enemyWorm,
this.opponent.worms[i].position) <= 2) {
                return true;
            }
        }
    }

    return false;
}

```

Dari cuplikan kode diatas, fungsi melakukan loop melalui jumlah elemen di array worm musuh sebesar n sehingga memiliki kompleksitas algoritma  $O(n)$ .

### 3.4. Analisis Efektivitas dari Solusi

Dengan strategi greedy yang dibuat oleh penulis, berikut efektivitas bot greedy melawan reference bot dalam 10 pertandingan:

No.	Menang/Kalah	Jumlah Ronde	Skor Akhir
1.	Menang	118	1356
2.	Menang	115	1380
3.	Menang	129	1376
4.	Kalah	123	1233
5.	Menang	115	1320
6.	Menang	119	1436
7.	Menang	134	1464
8.	Menang	118	1385
9.	Menang	117	1283
10.	Menang	110	1278

1. Win Percentage : 90%
2. Rata-rata Jumlah Ronde : 119.9
3. Rata-rata Skor Akhir : 1353.9

### 3.5. Strategi Greedy yang Dipilih

Dari beberapa kumpulan alternatif solusi yang terdapat pada bab 3.1., penulis memilih strategi greedy yaitu memaksimalkan pencarian terhadap bot musuh terdekat dari posisi worm player, lalu memindahkan worm player mendekat ke bot musuh lalu berusaha memberikan damage kepada bot tersebut. Untuk implementasi lebih lanjut, worm player memaksimalkan jumlah bot musuh yang akan dilempar *banana bomb* dan *snowball*. Jika nyawa di atas 50, worm player akan melempar *banana bomb* dan *snowball* apabila jumlah musuh di dalam radius kedua *skill* sama dengan atau lebih dari 2. Apabila nyawa sudah 50 ke bawah, *worm* akan secara otomatis melemparkan bom atau *snowball* ke musuh terdekat. Jika ada dua musuh di jarak tembak *worm* player, *worm player* akan menembak musuh yang memiliki nyawa terendah. Jika darah *worm* ada yang darahnya kurang dari sama dengan 15, maka *worm* akan berusaha mencari apakah masih terdapat powerup health pack yang tersisa. Dan jika masih ada, *worm* akan berusaha bergerak ke arah powerup tersebut. Jika darah *worm* kurang dari sama dengan 25, maka *worm player* akan diseleksi secara paksa lalu akan menembak musuh terdekat.



## BAB IV IMPLEMENTASI DAN PENGUJIAN

### 4.1. Implementasi

#### Pseudocode Fungsi Musuh Terdekat

```
function nearestEnemyDirection(input currentWorm : Position) -> Direction  
{Mencari musuh terdekat dan mengembalikan arah musuh}
```

##### **KAMUS LOKAL**

minimalIndex, minimalDistance, temp : integer  
worm : Array of Worm

```
function euclideanDistance(input currentWorm : Position, input enemyWorm : Position)  
-> integer  
{Fungsi menghitung jarak euclidean antara 2 titik di map}
```

```
Function resolveDirection(input currentWorm : Position, input enemyWorm : Position)  
-> Direction  
{Fungsi menentukan arah di map}
```

##### **ALGORITMA**

minimalDistance <- 10000  
minimalIndex <- -1

```
{Iterasi sepanjang array worm musuh}  
i traversal [0..2]  
    {Jika worm musuh tidak hidup, lompat}  
    if (health_musuh <= 0) then  
        continue  
  
    {Cari jarak minimal}  
    temp = euclideanDistance(currentWorm, worm[i].position)  
  
    if (temp < minimalDistance) then  
        minimalIndex <- i  
  
{Kembalikan arah seperti E, NE, W, dsb.}  
-> resolveDirection(currentWorm, opponent.worm[minimalIndex].position)
```

#### Pseudocode Lempar Skill

```
function throwSkill(input selectedWorm : MyWorm) -> Command  
{Mengembalikan lempar banana bomb atau snowball}
```

##### **KAMUS LOKAL**

minimalIndex, minimalDistance, temp : integer

```

worm : Array of Worm
selectedWorm : MyWorm

function euclideanDistance(input currentWorm : Position, input enemyWorm : Position)
-> integer
{Fungsi menghitung jarak euclidean antara 2 titik di map}

function BananaBomb(input WormPosition : Position) -> Command
{Fungsi memanggil fungsi lempar Bomb}

function Snowball(input SelectedWorm : MyWorm, input WormPosition : Position) ->
Command
{Fungsi memanggil fungsi lempar Bomb}

ALGORITMA
minimalDistance <- 10000
minimalIndex <- -1

{Iterasi sepanjang array worm musuh}
i traversal [0..2]
    temp <- euclideanDistance(selectedWorm.position, worm[i].position)

    {Musuh terdekat}
    if (temp < minimalDistance and worm.health > 0) then
        minimalIndex <- i

    if (euclideanDistance(selectedWorm.position, worm[i].position)) then

        {Jika worm saat ini bisa snowball, snowball masih ada dan musuh tidak
        frozen}
        if (selectedWorm bisa snowball and snowball > 0 and musuh unfrozen)
        then
            -> Snowball(SelectedWorm, worm[i].position)

        {Jika worm dapat banana bomb dan banana masih ada}
        else if (selectedWorm bisa banana bomb and banana > 0) then
            -> BananaBomb(worm[i].position)

```

### Pseudocode Banana

```

function BananaBomb(input enemyPosition : Position) -> Command
{Mengembalikan command BananaCommand}

```

### KAMUS LOKAL

### ALGORITMA

```

if (health_player > 50 and checkBananasGroup(enemyPosition)) {mengecek apakah
ada 2 musuh atau lebih yang berada pada jarak tembak bananaBomb}
-> BananaCommand(enemyPosition.x, enemyPosition.y)

```

```
else if (health_player <= 50) {apabila health player kurang dari 50, maka
menembakkan bananaBomb walaupun musuh tidak berkumpul}
-> BananaCommand(enemyPosition.x, enemyPosition.y)
-> null
```

### Pseudocode Snowball

```
function Snowball(input SelectedWorm : MyWorm, input WormPosition : Position) ->
Command
{Fungsi memanggil fungsi lempar SnowBall}
```

#### KAMUS LOKAL

surroundingBlocks : Array of Cell  
targetPos : Array of Position

#### ALGORITMA

```
surroundingBlocks ← getSurroundingCells(enemyWorm.position.x,
enemyWorm.position.y)
i traversal[0..surroundingBlocks.size()] {traversal di array surroundingBlocks}
    targetPos.x ← surroundingBlocks.get(i).x
    targetPos.y ← surroundingBlocks.get(i).y
    If (checkSnowBallsGroup(targetPos) and euclideanDistance
(selectedWorm.position, targetPos) <= 5 and enemyWorm.roundsUntilUnfrozen < 2)
then {mengecek apakah musuh berkumpul}
    → SnowBallCommand(targetPos.x, targetPos.y)
if (enemyWorm.roundsUntilUnfrozen < 2)
    → SnowBallCommand(enemyWorm.position.x, enemyWorm.position.y)
→ null
```

## 4.2. Struktur Data

Terdapat beberapa objek yang terdapat pada permainan worms. Objek-objek yang dipakai dalam mengimplementasikan strategi greedy pada permainan worm ini diantaranya yaitu:

1. myPlayer  
Terdapat beberapa atribut yang terdapat pada objek myPlayer. Atribut tersebut yaitu id, score, health, dan remainingWormsSelection. Kemudian terdapat juga array objek bertipe worms.
2. opponents  
Terdapat beberapa atribut yang terdapat pada objek opponents, yaitu id dan score. Kemudian terdapat juga array objek bertipe worms.
3. cellType  
CellType berisi jenis tipe cell yang ada dalam map permainan. Terdapat tiga jenis type, yaitu DIRT, DEEP\_SPACE, dan AIR.
4. Worms

Terdapat beberapa atribut yang terdapat pada objek worm. Atribut-atribut tersebut yaitu id, health, position, diggingRange, movementRange, roundsUntilUnfrozen, dan profession.

#### 5. myWorms

Objek myWorms merupakan anak dari objek Worms. Objek ini memiliki atribut weapon, bananaBombs, dan snowballs.

Struktur data objek secara keseluruhan adalah Bot sebagai parent class. Di dalam kelas ini dapat didefinisikan method-method yang akan digunakan untuk mengimplementasikan algoritma greedy yang sudah dipilih. Beberapa method yang didefinisikan penulis adalah forceMoveToCell, forceMoveToNearestEnemy, isOccupied, powerUpPosition, nearestEnemyDirection, throwSkill, Snowball, BananaBomb, checkBananaIsGroup dan checkSnowballsGroup. Method-method ini kemudian dipanggil dalam method run yang akan dipanggil di Main.java ketika sedang menjalankan program.

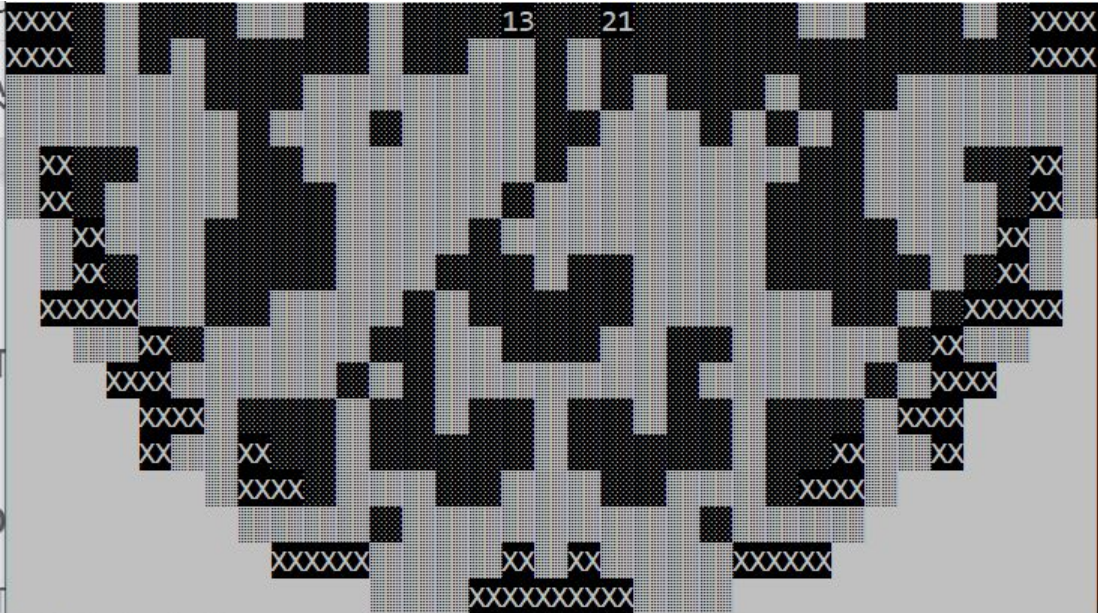
### 4.3. Pengujian

#### Proses Kompilasi

```
[INFO] Building jar: H:\ITB Semester 4\Tubes\Stima1-IF2211\starter-bots\target\java-sample-bot.jar
[INFO]
[INFO] --- maven-assembly-plugin:2.2-beta-5:single (make-assembly) @ java-sample-bot ---
[INFO] Building jar: H:\ITB Semester 4\Tubes\Stima1-IF2211\starter-bots\target\java-sample-bot-jar-with-d
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ java-sample-bot ---
[INFO] Installing H:\ITB Semester 4\Tubes\Stima1-IF2211\starter-bots\target\java-sample-bot.jar to C:\Use
[INFO] Installing H:\ITB Semester 4\Tubes\Stima1-IF2211\starter-bots\pom.xml to C:\Users\TES\.m2\reposito
[INFO] Installing H:\ITB Semester 4\Tubes\Stima1-IF2211\starter-bots\target\java-sample-bot-jar-with-depe
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.672 s
[INFO] Finished at: 2021-02-19T21:05:22+07:00
[INFO] -----

Process finished with exit code 0
```

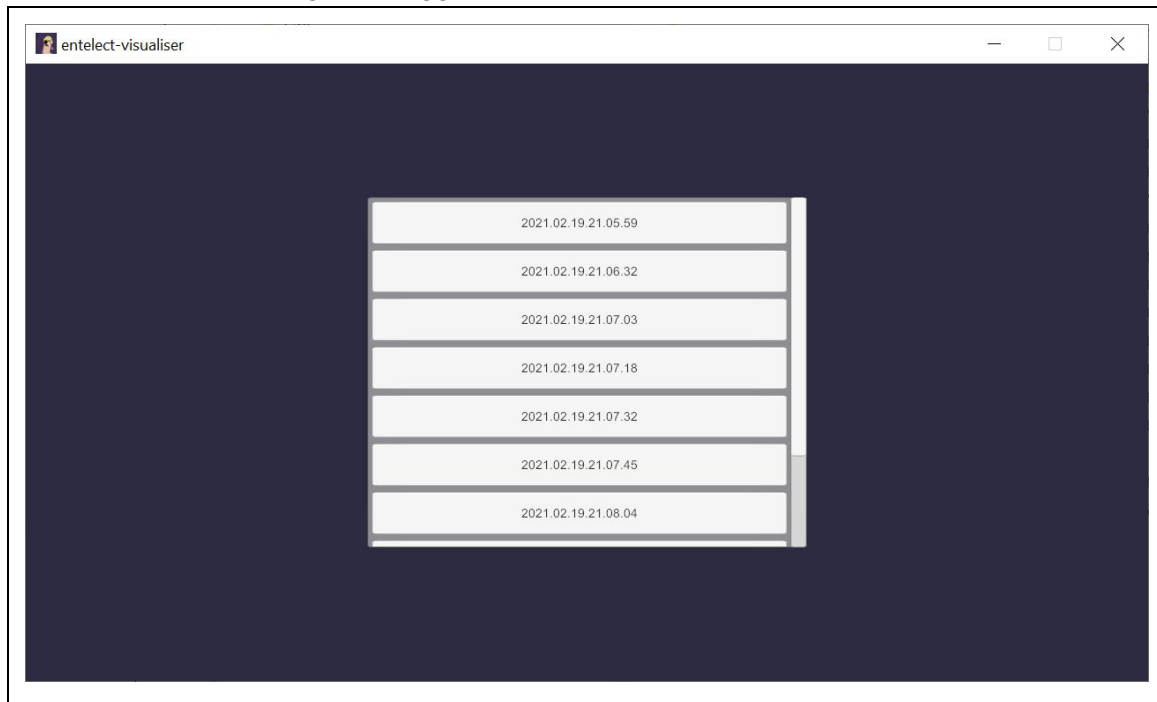
Run Program Utama



```
C;118;shoot E
C;118;shoot W
=====
Completed round: 118
=====

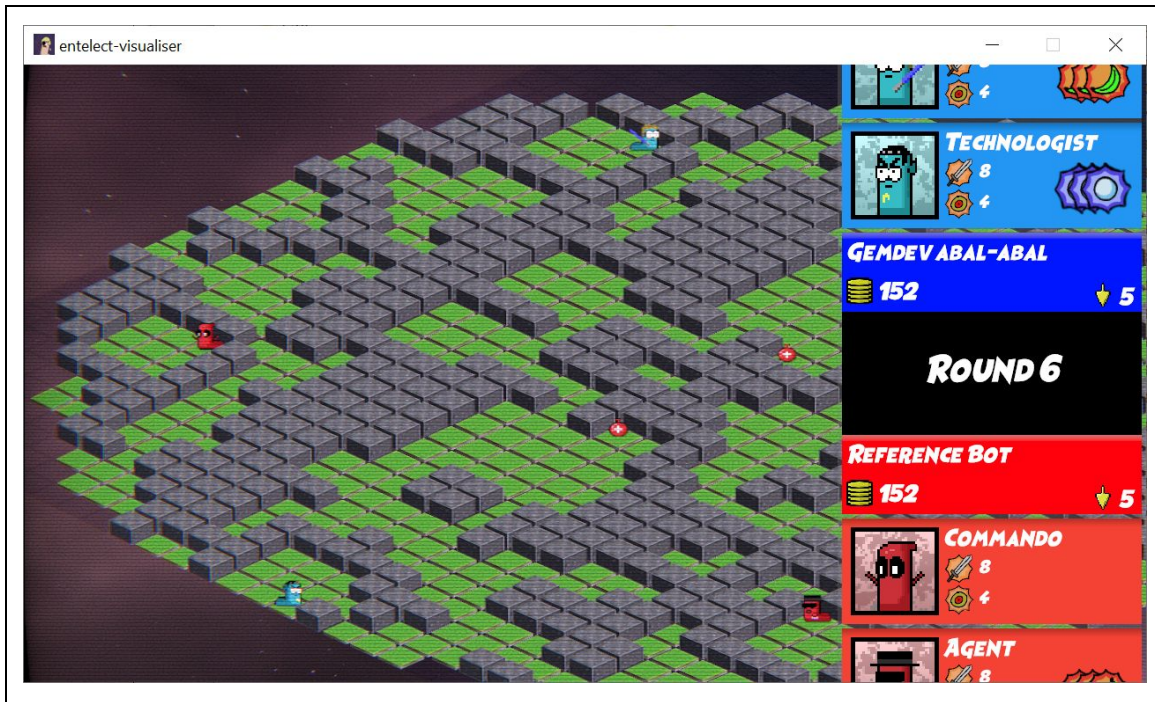
Check if match is valid
=====
The winner is: A - Gemdev abal-abal
=====
```

Melihat permainan dengan menggunakan visualizer

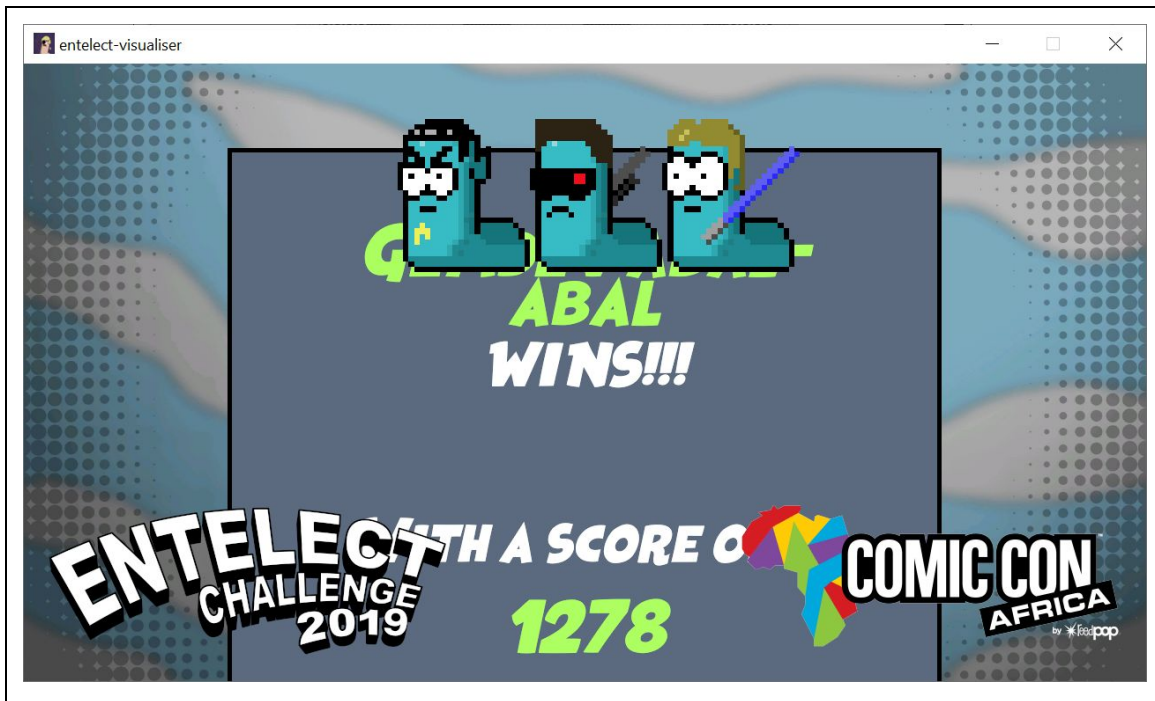




## Cuplikan Permainan



## Hasil Permainan





## BAB V

### KESIMPULAN DAN SARAN

#### 5.1. KESIMPULAN

Algoritma *greedy* merupakan salah satu algoritma dari berbagai macam algoritma yang berfungsi untuk mencari solusi dari sebuah permasalahan yang ada dengan melakukan pendekatan langkah per langkah dengan cara meminimumkan atau memaksimalkan sesuatu. Algoritma ini dapat memberikan solusi optimal lokal dengan harapan solusi optimal lokal ini dapat menjadi solusi optimal global. Algoritma ini biasanya lebih sering dipakai untuk menyelesaikan permasalahan apabila dibandingkan dengan algoritma *brute force* karena terbukti lebih efisien secara besarnya kompleksitas waktu yang dihasilkan.

Salah satu penerapan algoritma *greedy* adalah dalam menyelesaikan permasalahan dalam permainan *worms*. Strategi *greedy* yang digunakan dalam permainan ini harus dapat memberikan sebuah solusi akhir yaitu bot yang dijalankan oleh pemain harus dapat menang apabila ditandingkan dengan bot musuh. Dalam usaha untuk mencari solusi dari permainan ini, terdapat beberapa alternatif solusi *greedy* yang mungkin diterapkan untuk menyelesaikan permasalahan. Akan tetapi pada akhirnya penulis memilih satu solusi yang menurut penulis paling cocok digunakan untuk permainan ini, yaitu strategi *greedy* untuk mendekati musuh yang berada pada jarak paling dekat dari bot pemain kemudian menyerang bot musuh tersebut dengan syarat lebih lanjut yang dijelaskan pada bab 3.5.

Penulis menerapkan strategi *greedy* yang dipilih ke dalam bot *player* yang dituliskan dalam bahasa pemrograman java. Dalam implementasinya, penulis mengubah beberapa file yang ada dalam folder *entities*. Penulis juga menambahkan file ke dalam folder *command* agar bot dapat melakukan command sesuai dengan *game rules* yang diberikan. Implementasi algoritma *greedy* yang dipilih dituliskan dalam program bot.java. Hasil dari pengujian menunjukkan bahwa bot dapat menerapkan strategi *greedy* yang dipilih oleh pengguna.

Hasil dari pengerjaan tugas besar I IF2211 Strategi Algoritma ini adalah penulis dapat memahami lebih mendalam tentang algoritma *greedy* dan penerapannya dalam program di dunia nyata (dalam kasus ini permainan *worms*). Penulis juga mendapatkan ilmu yang lebih dalam tentang paradigma pemrograman berorientasi objek dengan menggunakan bahasa Java.

#### 5.2. SARAN

Saran untuk pengembangan lebih lanjut dari implementasi algoritma *greedy* pada bot adalah dapat berupa mencari kemungkinan-kemungkinan lain yang dapat dijadikan solusi namun dengan efisiensi dan efektivitas yang lebih baik daripada solusi yang diterapkan pada saat ini. Saran untuk keberlanjutan program ini adalah penulisan kode

dapat lebih digeneralisasi agar dalam pemanggilannya lebih efektif dan efisien serta membutuhkan waktu kompilasi yang lebih sedikit. Kode yang dibuat juga harus menggunakan nama yang jelas agar lebih mudah digunakan pada saat menerapkan algoritma greedy. Saran untuk tugas besar ini adalah sebaiknya memberikan lebih referensi kode yang dapat dijadikan referensi belajar.

## DAFTAR PUSTAKA

- EntelectChallenge. (2019, August 16). EntelectChallenge/2019-Worms. Retrieved from <https://github.com/EntelectChallenge/2019-Worms/blob/2019.3.2/game-engine/game-rules.md>
- Greedy Algorithms. (n.d.). Retrieved from <https://brilliant.org/wiki/greedy-algorithm/>
- Levitin, A. (2012). *Introduction to the design & analysis of algorithms*. Pearson.
- Munir, R. (2021, Januari). *Algoritma Greedy*. Homepage Rinaldi Munir. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)