

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Worms”

LAPORAN TUGAS BESAR

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma
Semester II Tahun 2020/2021



Disusun oleh

Gde Anantha Priharsena	(13519026)
Reihan Andhika Putra	(13519043)
Reyhan Emyr Arrosyid	(13519167)

**TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2020**

BAB I

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> .

BAB II

LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma greedy adalah algoritma yang *digunakan* untuk membentuk solusi langkah perlangkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya.

2.1.1 Prinsip Utama Algoritma Greedy

Prinsip utama algoritma greedy adalah “*take what you can get now*”. Maksud dari prinsip tersebut adalah pada setiap langkah dalam algoritma greedy, diambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya. Solusi tersebut disebut dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.

2.1.2 Elemen Algoritma Greedy

Elemen-elemen yang *digunakan* dalam penerapan algoritma greedy antara lain:

1. Himpunan Kandidat, adalah himpunan yang berisi elemen pembentuk solusi.
2. Himpunan Solusi, adalah himpunan yang terpilih sebagai solusi persoalan.
3. Fungsi Seleksi, adalah fungsi memilih kandidat yang paling mungkin mencapai solusi optimal
4. Fungsi Kelayakan, adalah fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.
5. Fungsi Solusi, adalah fungsi yang mengembalikan nilai boolean, True jika himpunan solusi yang sudah terbentuk merupakan solusi yang lengkap; False jika himpunan solusi belum lengkap
6. Fungsi Objektif, adalah fungsi yang mengoptimalkan Solusi

2.2 Game Engine

Game engine adalah perangkat lunak yang membantu dalam pembuatan dan pengembangan permainan. Fungsi utama *game engine* adalah membuat proses pembuatan dan pengembangan permainan lebih ekonomis, karena cukup dengan satu *game engine* mampu membuat banyak permainan dengan jenis yang berbeda-beda. Salah satu *game engine* yang populer digunakan adalah Unity.

2.3 Game Engine Entellect Worms

Dalam *game Entellect 2019 – Worms* terdapat beberapa komponen yang diperlukan agar game dapat berjalan:






1. *Game engine interface: Interface*(antarmuka) yang digunakan *game runner* untuk dapat menjalankan atau mengembangkan *game* dengan *game engine* yang berbeda.
2. *Game engine: Game engine* bertanggung jawab untuk memaksa *bot* yang dibuat pemain agar mematuhi aturan yang didefinisikan. *Game engine* akan menyalurkan perintah yang dikirimkan oleh *bot* kedalam game untuk diproses jika perintah yang diberikan valid.
3. *Game runner: Game runner* bertanggung jawab untuk menjalankan pertandingan antar pemain. *Game runner* menerima perintah dari *bot* (dengan berbagai bahasa) dan mengirimkannya ke *game engine* untuk dieksekusi
4. *Reference bot: bot* yang disediakan dengan beberapa strategi agar dapat diujikan dengan *bot* yang akan dibuat. Tersedia dalam bahasa javascript.
5. *Starter bot: bot* awal dengan logika dasar yang bisa digunakan sebagai referensi dalam pengembangan *bot*.

2.3.1 Mulai Membangun Starter Bot Dalam Bahasa Java

Hal pertama yang harus dilakukan untuk membangun *bot* adalah melakukan *clone* terhadap program entellect starter-pack di (<https://github.com/EntellectChallenge/2019-Worms>). Dalam membangun *bot* terdapat banyak bahasa pilihan yang dapat digunakan. Pada tugas besar ini akan dibangun sebuah *bot* dengan bahasa pemrograman Java. Pada *folder* Java terdapat tiga *folder* dan dua *file* awal yang disediakan, penjelasannya adalah sebagai berikut:

1. *Folder command: Folder* ini berisikan kelas dari perintah yang dapat digunakan oleh pemain. Perintah yang disediakan adalah *dig*, *do nothing*, *move*, dan *shoot*. Semua kelas tersebut mengimplementasikan interface *command*.

2. *Folder entities*: *Folder* ini berisikan kelas entitas (mahkluk) yang ada pada game, contohnya adalah *MyPlayer*, *MyWorm*, *Cell* dan lain-lain.
3. *Folder enums*: *Folder* ini berisikan entitas yang memiliki nilai berupa konstanta. Misalnya fakta bahwa ada empat jenis *cell* yaitu *DIRT*, *DEEP_SPACE*, *AIR*, dan *LAVA*.
4. *Bot.Java*: *File* ini berisikan logika dari *bot* yang dibangun. Pada *file* ini akan diimplementasikan strategi greedy dengan memanfaatkan kelas lain yang ada. Semua strategi akan ditulis pada *method* *run()* di kelas *bot*.
5. *Main.Java*: *File* ini adalah *file* yang bertanggung jawab untuk menjalankan *Bot.Java* dengan menginstansiasi *bot* pada setiap ronde dan menerima perintah dari *bot* tersebut untuk dikirimkan ke game engine untuk kemudian diproses.

 command	02/09/2019 17:38	File folder	
 entities	09/02/2021 23:10	File folder	
 enums	02/09/2019 17:38	File folder	
 Bot.java	09/02/2021 23:51	JAVA File	6 KB
 Main.java	02/09/2019 17:38	JAVA File	2 KB

Gambar 2. Struktur *Folder Java*

Untuk dapat menjalankan bot nya maka dibutuhkan beberapa aplikasi yang harus di-*install* yaitu Java JDK 8, *code editor* IntelliJ IDEA, dan NodeJs. Untuk mendapatkan *file executable* java (.jar), *file* yang disediakan harus di-*build* menggunakan IntelliJ IDEA. Setelah itu data lokasi “player-a” diubah menjadi “./starter-bots/java” pada *file* game-runner-config.json. Setelah itu, jalankan run.bat untuk menjalankan *game-engine*. Berikut ini adalah informasi mengenai beberapa kegunaan *file* konfigurasi yang ada di game:

1. game-runner-config.json: Letaknya ada di dalam *folder* starter pack, *digunakan* untuk mengatur lokasi *file bot* pemain satu, pemain dua, *file* hasil pertandingan, konfigurasi *game*, dan *game engine*.
2. game-config.json: Letaknya ada di dalam *folder* starter-pack, *digunakan* untuk mengatur konfigurasi dasar dari game *Worms* seperti atribut dari setiap jenis *worm*, akumulasi poin dari setiap perintah dalam game, dan lain-lain.
3. bot.json : Letaknya ada di dalam *folder* java dan *digunakan* untuk mengganti nama *bot* dan lokasi *executable* (.jar) dari bot yang dibuat.

Untuk dapat menjalankan *visualizer*, harus mendownload *visualizernya* di (<https://github.com/dlweatherhead/entelect-challenge-2019-visualiser>). Setelah menjalankan pertandingan melalui cmd maka hasilnya secara *default* akan tersimpan di *folder* match-logs.

Kemudian *folder* pertandingan yang ingin divisualisasikan harus dipindahkan ke dalam *folder* matches di *visualizer*. *Folder* pertandingan memiliki format nama seperti *timestamp* pertandingannya sehingga hanya perlu mengingat kapan pertandingan tersebut dijalankan untuk mendapatkan *folder* yang tepat. Setelah memindahkan *folder* pertandingan ke *visualizer*, jalankan aplikasi *visualizer* dan pertandingan sudah siap ditonton dengan menggunakan tampilan GUI.

2.3.2 Mengembangkan Starter Bot

Meskipun *starter bot* sudah menyediakan berbagai macam kelas beserta *method* dan *atributenya*, kelas tersebut masih bisa disempurnakan lagi dan ditambah jumlahnya. Contohnya *starter-bot* sudah menyediakan kelas untuk perintah *dig*, *move*, *do nothing*, dan *shot* namun belum menyediakan perintah untuk *select*, *banana bomb*, dan *snowball*. Selain kelas *command*, kelas *entity* dan yang lainnya juga masih bisa ditambahkan lagi, misalnya *entity* Banana Bomb yang tidak disediakan oleh *starter bot*. Kelas tersebut dapat ditambahkan dengan cara di bawah ini

BananaCommand.java	BananaBomb.java
<pre>package za.co.entelect.challenge.command; public class BananaCommand implements Command { // Attribute private final int x; private final int y; private int selectedWorm; public BananaCommand(int x, int y) { // Constructor } public BananaCommand(...Parameter...) { // Constructor user defined } @Override public String render() { // Render string command untuk dibaca game engine } }</pre>	<pre>package za.co.entelect.challenge.entities; import com.google.gson.annotations.SerializedName; public class BananaBomb { @SerializedName("damage") public int damage; @SerializedName("range") public int range; @SerializedName("count") public int count; @SerializedName("damageRadius") public int damageRadius; }</pre>

Dalam membuat kelas baru, *method* dan *attribute* yang dibutuhkan dapat ditambahkan ke kelas tersebut. Pada kasus *BananaCommand.java*, dapat didefinisikan *constructor default* dan *user-defined constructor* seperlunya. Perhatikan bahwa *method* *render* perlu dituliskan supaya *command* bisa dibaca oleh *game engine* melalui *main.java*. *Method* *render* bisa dicontoh dari kelas *command* lain yang sudah disediakan.

Untuk membuat kelas *BananaBomb*, manfaatkan data yang disediakan oleh *game engine* ke program *java*. Hal itu diperlukan karena tidak diketahuinya jumlah, jarak, *damage*, dan atribut lain dari *BananaBomb* dan hal tersebut tidak dikodekan secara manual melainkan diatur oleh *game*

engine. Dalam pertukaran data dengan *game engine*, *worms* memanfaatkan *file json* yaitu *state.json*. Penjelasannya tentang *state* yang disediakan, namanya, dan isinya dapat dibaca di (<https://github.com/EntelectChallenge/2019-Worms/blob/master/game-engine/state-files.md>).

Untuk menggunakan *file* dengan ekstensi *.json* di Java maka diperlukan modul *json* yang harus di-*import*. Sintaks penggunaannya adalah sebagai berikut:

```
com.google.gson.annotations.SerializedName;
@SerializedName(<nama_value_di_file_state.json>)
    <keyword_modifier> <type_varibel> <nama_variabel>;
Contohnya
com.google.gson.annotations.SerializedName;
@SerializedName("damageRadius")
    public int damageRadius;
```

Setelah mempersiapkan semua kelas dan elemen baru yang diperlukan, langkah selanjutnya adalah memasukkan strategi greedy yang diinginkan pada *file bot.java*. *Starter bot* sudah menyediakan beberapa *method* dan *attribute* awal yang bisa digunakan dalam pengembangan *bot*. *Method* dan *attribute* yang disediakan juga bisa diubah dan ditambah lagi untuk melengkapi strategi greedy. Semua strategi greedy yang ditulis akan diimplementasikan pada *method run* dan pastikan *method run* harus memberikan *return* sebuah perintah yang valid agar dideteksi oleh *game engine*. Manfaatkan fitur OOP yang disediakan Java untuk membuat strategi greedy. Isi dari *bot.java* adalah sebagai berikut:

```
package za.co.entelect.challenge;

import za.co.entelect.challenge.command.*;
import za.co.entelect.challenge.entities.*;
import java.util.*;
import java.util.stream.Collectors;
// Import semua hal yang dibutuhkan

public class Bot {
    private Random random;
    private GameState gameState;
    private Opponent opponent;
    // Attribute bawaan dan tambahaan

    public class A{ Nested Class baru }
    // Nested Class jika dibutuhkan

    private Direction resolveDirection(Position a, Position b){ Isi method }
    private int euclideanDistance(int aX, int aY, int bX, int bY) { .... }
    private List<CellandFreezeCount> getFreezedLocation() {
        // Method bawaan dan tambahan sendiri untuk membuat strategi greedy
    }

    public Command run() {Strategi Greedy diletakkan disini}
}
```


BAB III

PEMANFAATAN STRATEGI GREEDY

3.1 Solusi Algoritma Greedy yang Mungkin Dipilih

3.1.1. *Greedy by Health Point*

Greedy by Health Point adalah strategi penargetan *worm* lawan berdasarkan *health point*. Pada setiap ronde, pilihlah *worm* lawan yang masih hidup dan memiliki *health point* terkecil dari seluruh *worm* lawan yang tersisa. Kemudian lakukan *command* yang “menargetkan” *worm* lawan tersebut.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *worm* lawan.
- Himpunan Solusi: *Worm* lawan yang terpilih.
- Fungsi Solusi: Memeriksa apakah *worm* lawan yang dipilih masih hidup dan memiliki *health point* terkecil.
- Fungsi Seleksi: Pilih *worm* lawan dengan *health point* terkecil.
- Fungsi Kelayakan: Memeriksa apakah *worm* lawan masih hidup (memiliki *health point* lebih dari 0).
- Fungsi Objektif: Minimumkan *health point* dari *worm* lawan yang dipilih.

b. Analisis Efisiensi Solusi

Terdapat tiga *worm* yang harus diseleksi(3) lalu mengecek cell yang mengarah ke *worm* musuh(1) supaya dapat melakukan (*move/dig*) atau *shoot* (3) sehingga kompleksitasnya adalah

$$T(n) = 3*3*1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Jumlah *worm* lawan pada radius *shoot worm* kita minimal sama dengan jumlah *worm* kita disekitar *worm* yang sedang aktif (inklusif), sehingga jika diperlukan pertarungan jarak dekat dengan *command shoot* secara intens ada kemungkinan untuk mengurangi jumlah *worm* lawan sehingga mengurangi *damage* yang diterima *worm* kita.

Strategi ini tidak efektif apabila:

- Jumlah *worm* lawan pada radius *shoot worm* kita lebih dari jumlah *worm* kita disekitar *worm* yang sedang aktif (inklusif), sehingga jika harus dilakukan pertarungan jarak dekat dengan *command shoot* ada kemungkinan untuk kalah dari segi jumlah dan membuat pertarungan sia-sia.

3.1.2. Greedy by Enemy Location

Greedy by enemy location adalah strategi penargetan *worm* lawan berdasarkan jarak *worm* lawan dengan *worm* kita. Pada setiap ronde, cari *worm* lawan terdekat dengan *worm* kita yang sedang aktif, apabila *worm* lawan bisa diserang maka lakukan *command shoot*, apabila tidak maka dekati *worm* lawan.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *worm* lawan.
- Himpunan Solusi: *Worm* lawan yang terpilih.
- Fungsi Solusi: Memeriksa apakah *worm* lawan yang dipilih masih hidup dan memiliki jarak terdekat dengan *worm* kita yang sedang aktif.
- Fungsi Seleksi: Pilihlah *worm* lawan yang memiliki jarak terdekat dengan *worm* kita yang sedang aktif.
- Fungsi Kelayakan: Memeriksa apakah *worm* lawan yang ditarget masih hidup.
- Fungsi Objektif: Meminimumkan jarak *worm* kita dengan *worm* lawan supaya dapat ditembak.

b. Analisis Efisiensi Solusi

Terdapat tiga *worm* yang harus diseleksi(3) lalu mengecek cell yang mengarah ke *worm* musuh(1) supaya dapat melakukan (*move/dig*) atau *shoot* (3) sehingga kompleksitasnya adalah

$$T(n) = 3*3*1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Beberapa *worm* kita dekat dengan tepat satu *worm* lawan sehingga dapat dilakukan pengepungan dengan cepat.
- *Worm* lawan terdekat memiliki *damage* dan *health* lebih kecil sehingga *worm* kita menang dalam adu tembak saat jarak antar *worm* sudah dekat.

Strategi ini tidak efektif apabila:

- Beberapa *worm* lawan sedang berkumpul sehingga *worm* kita kalah dalam adu tembak saat jarak antar *worm* sudah dekat.
- *Worm* lawan terdekat memiliki *damage* dan *health* lebih besar sehingga *worm* kita kalah dalam adu tembak saat jarak antar *worm* sudah dekat.

3.1.3. Greedy by Enemy Location V2

Greedy by enemy location adalah strategi pelarian diri dari *worm* lawan dengan mempertimbangkan lokasi lawan. Untuk setiap ronde, jika *worm* kita dapat melakukan *move command* ke *cell* di sekitar maka pilih *cell* dengan jarak euclidean terjauh dari semua *worm* lawan. Jika *worm* tidak bisa melakukan *move command*, maka lakukan *dig command* ke *cell* di sekitar dengan jarak euclidean terjauh dari semua *worm* lawan.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *cell* disekitar *worm* kita.
- Himpunan Solusi: *Cell* terpilih.
- Fungsi Solusi: Memeriksa apakah *cell* merupakan *cell* dengan jarak euclidean terjauh dari semua *worm* lawan serta *worm* kita dapat melakukan *move command* atau *dig command* ke *cell* tersebut.
- Fungsi Seleksi: Pilihlah *cell* dengan jarak euclidean terjauh dari semua lawan.
- Fungsi Kelayakan: *Cell* bukan bertipe *deep space* atau lava dan *cell* tidak berada pada jangkauan serangan lawan.
- Fungsi Objektif: Memaksimalkan jarak antara *worm* kita dengan *worm* lawan.

b. Analisis Efisiensi Solusi

Terdapat delapan *cell* yang harus dicek jenis tanahnya (8). Jika masih bisa *move* (1) maka lakukan *move* dulu jika tidak bisa maka lakukan *dig* (1). *Cell* tempat dia akan melakukan *move/dig* disortir berdasarkan jarak totalnya terhadap *worm* musuh yang masih hidup (*looping* 3 kali). Sehingga

$$T(n) = 8 * 1 * 3 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Bila jumlah *worm* kita yang hidup lebih sedikit dari jumlah *worm* lawan yang hidup. Sehingga lebih efektif untuk melakukan pelarian diri daripada melakukan pertarungan untuk mencoba menang dengan poin.

Strategi ini tidak efektif apabila:

- Jumlah *worm* kita sama atau lebih banyak dengan jumlah *worm* lawan, sehingga lebih baik untuk melakukan pertarungan jarak dekat supaya menang dengan mengalahkan semua *worm* lawan.

3.1.4. Greedy by Lava Location

Greedy by Lava Location adalah strategi untuk menghindari lava yang muncul di peta. Lava muncul pada ronde 100 dan semakin lama semakin ketengah hingga menyisakan kotak kecil dengan radius 4 pada ronde 350. Untuk menghindari lava maka *worm* juga harus menghindari daerah di sekitar lava karena lava mengisi daerah di sekitarnya secara tiba-tiba, namun apabila pertandingan sudah berada diatas ronde 320 kita tidak perlu menghindari daerah sekitar lava karena itu hanya akan mengurangi daerah untuk lari apabila sedang bertempur dengan *worm* lawan. Apabila sedang tidak disekitar lava maka greedy ini tidak perlu dilakukan.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *cell* disekitar *worm* yang aktif atau tidak ada *cell* yang dipilih.
- Himpunan Solusi: Salah satu *cell* disekitar *worm* yang sedang aktif atau tidak ada *cell* yang dipilih.
- Fungsi Solusi: Memeriksa apakah *cell* yang dipilih tidak berada di sekitar lava.
- Fungsi Seleksi: Memilih *cell* disekitar *worm* aktif yang tidak berada di lava maupun di sekitar lava jika ronde masih dibawah 320. Apabila tidak ada *cell* yang bisa dipilih untuk bergerak maka pilih *cell* yang memenuhi kriteria dan bisa di-*dig*.
- Fungsi Kelayakan: Mengevaluasi apakah *worm* yang sedang aktif perlu untuk menjauhi lava dan daerah disekitarnya. Apabila tidak abaikan *cell* yang dipilih.
- Fungsi Objektif: Memaksimalkan jarak *worm* kita dengan lava.

b. Analisis Efisiensi Solusi

Seleksi yang dilakukan adalah mengecek setiap *cell* di peta ($n*n$), apabila *cell* tersebut lava maka cek sekitarnya (9), lalu dicek apakah *worm* perlu menjauhi lava (1).

$$T(n) = 9 n^2 = O(n^2)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Hampir efektif untuk semua saat karena lava bisa dihindari dengan mudah dan lava tidak menghalangi jalur pelarian diri.

Strategi ini tidak efektif apabila:

- Bot lawan sangat pintar sehingga dapat memojokkan bot kita ke lava.

3.1.5. *Greedy by Worm Type*

Greedy by worm type adalah strategi untuk menyerang lawan berdasarkan jenis *worm* mereka. Terdapat tiga jenis *worm* yaitu *Agent*, *Commando*, dan *Technologist*. Prioritas jenis lawan yang diincar terlebih dahulu adalah *Agent* > *Technologist* > *Commando*. *Agent* diprioritaskan karena memiliki *banana bomb* yang jika dilempar memiliki *damage* yang besar dan dapat membersihkan *dirt* dalam area yang cukup luas. *Technologist* menjadi prioritas selanjutnya karena dapat membekukan *worm*. *Commando* menjadi prioritas terakhir karena tidak mempunyai serangan area dan relatif hanya memiliki serangan dasar yang dimiliki oleh semua jenis *worms*. Walaupun begitu, *commando* memiliki lebih banyak *health point* terbesar tapi hal itu tidak berdampak signifikan dibandingkan serangan area yang dimiliki oleh *Agent* atau *Technologist*.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat. *Agent*, *Commando*, dan *Technologist*.
- Himpunan Solusi: Salah satu *worm* dengan jenis tertentu
- Fungsi Solusi: Memeriksa apakah *worm lawan* yang dipilih masih hidup dan dipilih sesuai prioritas
- Fungsi Seleksi: Memilih *worm* lawan dengan prioritas tertinggi yang masih hidup
- Fungsi Kelayakan: Memeriksa apakah *worm* lawan masih hidup (memiliki *health poin* lebih dari 0).
- Fungsi Objektif: Membunuh *worm* lawan dengan prioritas tertinggi yang masih hidup dengan meminimalkan *damage* yang mungkin diterima oleh *worm* kita.

b. Analisis Efisiensi Solusi

Terdapat tiga *worm* yang harus diseleksi(3) lalu mengecek cell yang mengarah ke *worm* musuh(1) supaya dapat melakukan (*move/dig*) atau *shoot* (3) sehingga kompleksitasnya adalah

$$T(n) = 3 \cdot 3 \cdot 1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Strategi bot lawan adalah menyerang dengan berpencar, sehingga *worm* kita bisa mengepung *worm* lawan mula dari *worm* yang paling berbahaya.

Strategi ini tidak efektif apabila:

- Bot lawan sudah memprediksi strategi ini sehingga *worm* kita yang mengincar *worm* lawan sesuai prioritas akan dikepung oleh semua *worm* lawan yang masih hidup dan dilawan secara bersamaan.

3.1.6. Greedy by Bomb Area

Greedy by bomb area adalah strategi untuk memaksimalkan *damage bomb* ke *worm* lawan tanpa perlu memberi *damage* ke *worm* kita. Pada setiap ronde dengan *worm* yang sedang aktif bertipe Agent dan masih memiliki *banana bomb*, pilih *cell* yang bukan bertipe *deep space* dan masih termasuk dalam jangkauan *banana bomb* dengan jumlah *damage* terbesar untuk semua *worm* lawan tapi tidak memberikan *damage* ke *worm* kita.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *cell* yang ada pada map.
- Himpunan Solusi: *Cell* terpilih.
- Fungsi Solusi: Memeriksa apakah *cell* terpilih berada dalam jangkauan *banana bomb*, bukan bertipe *deep space*, dan memiliki jumlah *damage* terbesar untuk semua *worm* lawan tapi memiliki jumlah *damage* terkecil untuk semua *worm* kita.
- Fungsi Seleksi: Pilih *cell* yang berada pada jangkauan *banana bomb*.
- Fungsi Kelayakan: Memeriksa apakah *cell* yang dipilih bukan bertipe *deep space* dan jika *banana bomb* dilempar ke *cell* tersebut tidak memberikan *damage* ke *worm* kita.
- Fungsi Objektif: Maksimumkan jumlah *damage* yang diterima *worm* lawan.

b. Analisis Efisiensi Solusi:

Seleksi yang dilakukan adalah mengecek setiap *cell* ($n \times n$), apabila *cell* tersebut dapat dilempar *bomb* maka hitung *damage* yang dihasilkan oleh *bomb* tersebut ke *worm* musuh dan *worm* kita. Seleksi dalam menghitung *damage* adalah cek titik tengah dalam melempar *bomb* (1), lakukan *loop* untuk semua arah ads (n), lalu lakukan *loop* sebanyak *radius bomb* (2). Cek kelayakan dalam melempar *bomb* lalu urutkan berdasarkan *damage* terbesar. Algoritma pengurutan menggunakan fungsi bawaan sehingga tidak dihitung efisiensi waktunya. Efisiensi waktu algoritma adalah

$$T(n) = n \times n \times (1+2) = O(n^2)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Bot lawan menggunakan strategi yang memerlukan *worm*-nya untuk berkumpul

Strategi ini tidak efektif apabila:

- Bot lawan menggunakan strategi dengan *worm* yang berpencair
- Bot lawan menggunakan strategi dengan *worm*-nya yang selalu berada di dekat *worm* kita

3.1.7. Greedy by Freezed Area

Greedy by freezed area adalah strategi untuk membekukan musuh sebanyak mungkin dengan kondisi tertentu. Pada setiap ronde dengan *current worm* yang memiliki profesi *technologist* dan masih memiliki *snowball*, pilih *cell* yang bukan bertipe *deep space* dan masih termasuk dalam jangkauan *snowball* dengan jumlah *worm lawan* yang berada pada radius *snowball* terbanyak, *worm* kita tidak berada pada radius *snowball*, dan ada *worm lawan* beku yang bisa di *shoot* pada ronde selanjutnya.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: Semua *cell* yang ada pada map.
- Himpunan Solusi: *Cell* terpilih.
- Fungsi Solusi: Memeriksa apakah *cells* terpilih berada dalam dalam jangkauan *snowball* dengan jumlah *worm* lawan yang berada pada radius *snowball* terbanyak, *worm* kita tidak berada pada radius *snowball*, dan ada *worm lawan* beku yang bisa di *shoot* pada ronde selanjutnya.
- Fungsi Seleksi: Pilih *cell* yang berada pada jangkauan *snowball*.
- Fungsi Kelayakan: Memeriksa apakah *cell* yang dipilih bukan bertipe *deep space* dan ada *worm* lawan beku yang bisa di *shoot* pada ronde selanjutnya.
- Fungsi Objektif: Maksimumkan jumlah *worm lawan* yang berada pada radius *snowball* (jumlah *worm lawan* yang akan beku)

b. Analisis Efisiensi Solusi

Seleksi yang dilakukan adalah mengecek setiap *cell* ($n \times n$), apabila *cell* tersebut dapat dilempar *snowball* maka hitung *worm* yang dapat dibekukan. Seleksi dalam menghitung *worm* yang dapat dibekukan adalah cek titik tengah dalam melempar *snowball* (1), lakukan *loop* untuk semua arah (n), lalu lakukan *loop* sebanyak *radius*

snowball (1). Cek kelayakan dalam melempar *snowball* lalu urutkan berdasarkan *jumlah musuh* terbanyak. Algoritma pengurutan menggunakan fungsi bawaan sehingga tidak dihitung efisiensi waktunya. Efisiensi waktu algoritma adalah

$$T(n) = n * n * (1+2) = O(n^2)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Bot lawan menggunakan strategi yang memerlukan *worm*-nya untuk berkumpul

Strategi ini tidak efektif apabila:

- Bot lawan menggunakan strategi dengan *worm* yang berpencar
- Bot lawan menggunakan strategi dengan *worm*-nya yang selalu berada di dekat *worm* kita

3.2 Solusi Algoritma Greedy yang Dipilih dan Pertimbangannya

Pada *game* Worms ini terdapat beberapa solusi algoritma greedy yang bisa diimplementasikan dan digunakan tergantung pada kondisi worm yang sedang aktif untuk setiap rondonya. Sehingga, diperlukan suatu pertimbangan kapan suatu solusi algoritma greedy harus dipilih. Jika dilihat dari perspektif yang lebih luas, terdapat beberapa skema inti yang harus dimiliki sebuah *bot*, antara lain skema untuk menangani gerakan *worm* saat lava mulai menyebar, skema untuk menangani pertarungan dengan *worm* lawan, dan skema untuk melakukan pelarian diri.

Pada implementasi program yang kami lakukan, kami hanya memilih beberapa alternatif solusi algoritma greedy untuk menangani ketiga skema diatas. Untuk menangani gerakan worm saat lava mulai menyebar, kami menggunakan alternatif solusi *greedy by lava location*. Kondisi ini harus ditangani karena lava memiliki *damage* yang kontinu dan memiliki kemungkinan yang sedikit untuk ditempati *worm* lawan sehingga jika tidak ditangani maka akan sangat merugikan. Untuk menangani pertarungan dengan *worm* lawan, dapat dibagi menjadi beberapa skema yaitu skema saat masih memiliki *banana bomb*, skema saat masih memiliki *snowball*, dan skema saat hanya bisa melakukan *command shoot*. Untuk menangani skema saat masih memiliki *banana bomb*, kami menggunakan alternatif solusi *greedy by bomb area*. Untuk menangani skema saat masih memiliki *snowball*, kami menggunakan alternatif solusi *greedy by frozen area*. Untuk menangani skema saat hanya bisa melakukan *command shoot*, kami mempertimbangkan dua

alternatif solusi, jika musuh berada dalam jangkauan *worm* kita maka lakukan *greedy by health point*. Jika musuh tidak berada dalam jangkauan serangan *worm* kita maka lakukan *greedy by enemy location*. Kemudian, untuk menangani skema pelarian diri, kami menggunakan alternatif solusi *greedy by enemy location v2*.

Kami tidak menggunakan algoritma *greedy by worm type* dengan beberapa pertimbangan yaitu untuk pertarungan jarak dekat solusi ini tidak efektif karena lebih baik untuk menyerang *worm* lawan dengan *health point* terendah untuk mengurangi jumlah *worm* lawan yang hidup (menggunakan algoritma *greedy by health point*). Untuk pertarungan jarak jauh, jika semua *worm* kita menargetkan suatu *worm* yang sama dengan kondisi *health point* lawan masih penuh itu hanya akan memancing serangan balik yang mungkin dilakukan oleh lawan. Karena jika kita menargetkan suatu *worm* dengan jenis tertentu maka secara tidak langsung semua *worm* kita berkumpul pada suatu daerah. Akibatnya, *worm* lawan dapat dengan mudah memberikan serangan area seperti *snowball* dan *banana bomb*. Hal ini membuat lawan dapat memberikan *damage* yang besar ke *bot* kita dan mendapatkan banyak point karena banyaknya *worm* yang terkena serangan area tersebut.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Pseudocode

Note: Apabila seluruh bot.java dijadikan pseudocode akan membutuhkan 22 halaman sehingga kami memutuskan untuk menjadikan pseudocode hanya program strateginya. Kode lengkap bisa dilihat langsung di source-code nya dan sudah diberi komentar.

```
// Main Strategy
function run() → Command
Deklarasi
  center : Position
  lavaAndAdjacent : set of Cell
  bombedLocation : list of CellAndBombDamage
  freezedLocation : list of CellandFreezeCount
  allMove : list of MoveCommand
  allDig : list of DigCommand
  shootedEnemyWorm : list of Worm
  dangerousCell : set of Cell
  myWormCell : Cell
  shot : list of Worm
  predictedCurrentEnemyShot : set of Cell
  nonLavaMoves : list of MoveCommand
  nonLavaDigs : list of DigCommand
  nonLavaAndSafeMoves : list of MoveCommand
  surroundingBlocks : list of Cell
  cellIdx : int
  block : Cell
  closestEnemy : Worm
  direction : Direction
  toOther : Command
  safeDig : list of DigCommand
  closestFriend : MyWorm

Body
  // Persiapkan semua data yang dibutuhkan
  center ← Position(16, 16)
  lavaAndAdjacent ← getLavaAndAdjacent()
  bombedLocation ← getAllBombedLocation()
  freezedLocation ← getFreezedLocation()
  allMove ← getAllMoveCommand()
  allDig ← getAllDigCommand()
  shootedEnemyWorm ← getAllShootedWorm(currentWorm)
  dangerousCell ← getDangerousCells()
  myWormCell ← gameState.map[currWorm.position.y][currWorm.position.x]

  // Jika masih bisa select dan ada worm yang bisa di shoot
  // Maka select worm kita yang bisa melakukan shoot ke worm lawan tersebut
  if (myPlayer.remainSelect > 0) then
    for(Σ(Semua worm kita) sebagai myWorm) do
      shot ← getAllShootedWorm(myWorm)
      for(Σ(Semua worm di shot) sebagai shotWorm) do
        if (myWorm.health > 8 and myWorm.frozenTime = 0 and shotWorm.frozenTime > 1) then
          → ShootCommand(resolveDirection(myWorm.position, shotWorm.position), myWorm.id)
        endif
      endfor
    endfor
  endif

  // Jika worm kita yang hidup tersisa 1 dan darah ≤ 8, hindari pertarungan
  if (livingMyOwnWorm() = 1 and currentWorm.health ≤ 8) then
    output ("Escape from enemy")
```

```

    predictedCurrentEnemyShot ← getPredictedDangerousCells(true)
    → chooseMoveCommandToPosition(filterMoveByCells(allMove, predictedCurrentEnemyShot),
center)
endif

// Jika worm kita berada di cell lava atau disekitar lava
if (lavaAndAdjacent.contains(myWormCell) and (gameState.currentRound < 320 or
not(dangerousCell.contains(myWormCell))) then
    output("Our worm is on or next to lava")
    onLavaMoves ← filterMoveByCells(allMove, lavaAndAdjacent)
    nonLavaDigs ← allDig.filter(dig →
not(lavaAndAdjacent.contains(gameState.map[dig.getY()][dig.getX()])))
    nonLavaAndSafeMoves ← getAllSafeMoveCommand(nonLavaMoves)

// Move ke Non Lava cell yang aman dari jangkauan serangan worm lawan dan menuju ke tengah
map
if (not (nonLavaAndSafeMoves.isEmpty())) then
    output("Moving to center while avoiding lava and danger")
    → chooseMoveCommandToPosition(nonLavaAndSafeMoves, center)
endif

// Move ke Non Lava Cell dan menuju ke tengah map
if (not(nonLavaMoves.isEmpty())) then
    output("Moving to center while avoiding lava")
    → chooseMoveCommandToPosition(nonLavaMoves, center)
endif

// Move ke cell yang menuju tengah map tanpa memperdulikan lava atau serangan worm
if (not(allMove.isEmpty())) then
    output("Moving to center")
    → chooseMoveCommandToPosition(allMove, center)
endif

// Tidak bisa melakukan move
// Lakukan dig ke cell yang mengarah ke tengah dan menghindari lava
if (not(nonLavaDigs.isEmpty())) then
    output("Digging to center while avoiding lava")
    → chooseDigCommandToCenter(nonLavaDigs)
endif
endif

// Jika lawan sedang menargetkan agent atau technologist kita
// Worm lawan sedang berkumpul dan bergerak ke arah agent, sehingga bisa di bomb
// selama agent kita masih memiliki banana bomb
if (not(bombedLocation.isEmpty())) then
    // urutkan bombedLocation berdasarkan jumlah damage yang diterima musuh
    output("Throwing banana bomb to enemy worm")
    → BananaCommand(bombedLocation.get(0).cell.x,bombedLocation.get(0).cell.y)
endif
// Worm lawan sedang berkumpul dan bergerak ke arah technologist kita
// Jika mereka sudah berada pada range maka enemy worm tersebut bisa di freeze
// selama technologist kita masih memiliki snowball
if (not(freezedLocation.isEmpty()))
    // urutkan freezedLocation berdasarkan jumlah musuh yang terfreeze
    output("Throwing snowball to enemy worm")
    → SnowballCommand(freezedLocation.get(0).cell.x,freezedLocation.get(0).cell.y)
endif

// Jika Worm kita yang aktif sedang dalam keadaan bahaya
if (dangerousCell.contains(myWormCell)) then
    output("Our worm maybe in danger, choose what to do wisely")
    // Jika worm yang membuat worm kita dalam bahaya sedang beku, serang worm tersebut
    if (shootedEnemyWorm.size()==1 and shootedEnemyWorm.get(0).frozenTime>0) then
        output("Attack that freezed enemy")
        direction ← resolveDirection(currentWorm.position, shootedEnemyWorm.get(0).position)
        → ShootCommand(direction)
    else if (shootedEnemyWorm.size()==1 and mustBattle(shootedEnemyWorm.get(0))) then

```

```

        // Jika worm kita harus melakukan pertarungan
        output("Attack with opportunity")
        Direction direction ← resolveDirection(currentWorm.position,
shootedEnemyWorm.get(0).position)
        → ShootCommand(direction)
    else if(escapeFromDanger()≠null) then
        // Jika worm kita harus melarikan diri dari bahaya
        output("Escaping from danger")
        → escapeFromDanger()
    endif
endif

// Jika worm kita yang aktif bisa melakukan serangan ke worm lawan tanpa bahaya,
// maka serang worm lawan yang berada pada jangkauan dengan health poin terkecil
if (not(shootedEnemyWorm.isEmpty())) then
    // shootedEnemyWorm diurutkan berdasarkan darah musuh
    output("Shooting worm")
    direction ← resolveDirection(currentWorm.position, shootedEnemyWorm.get(0).position)
    → ShootCommand(direction)
endif

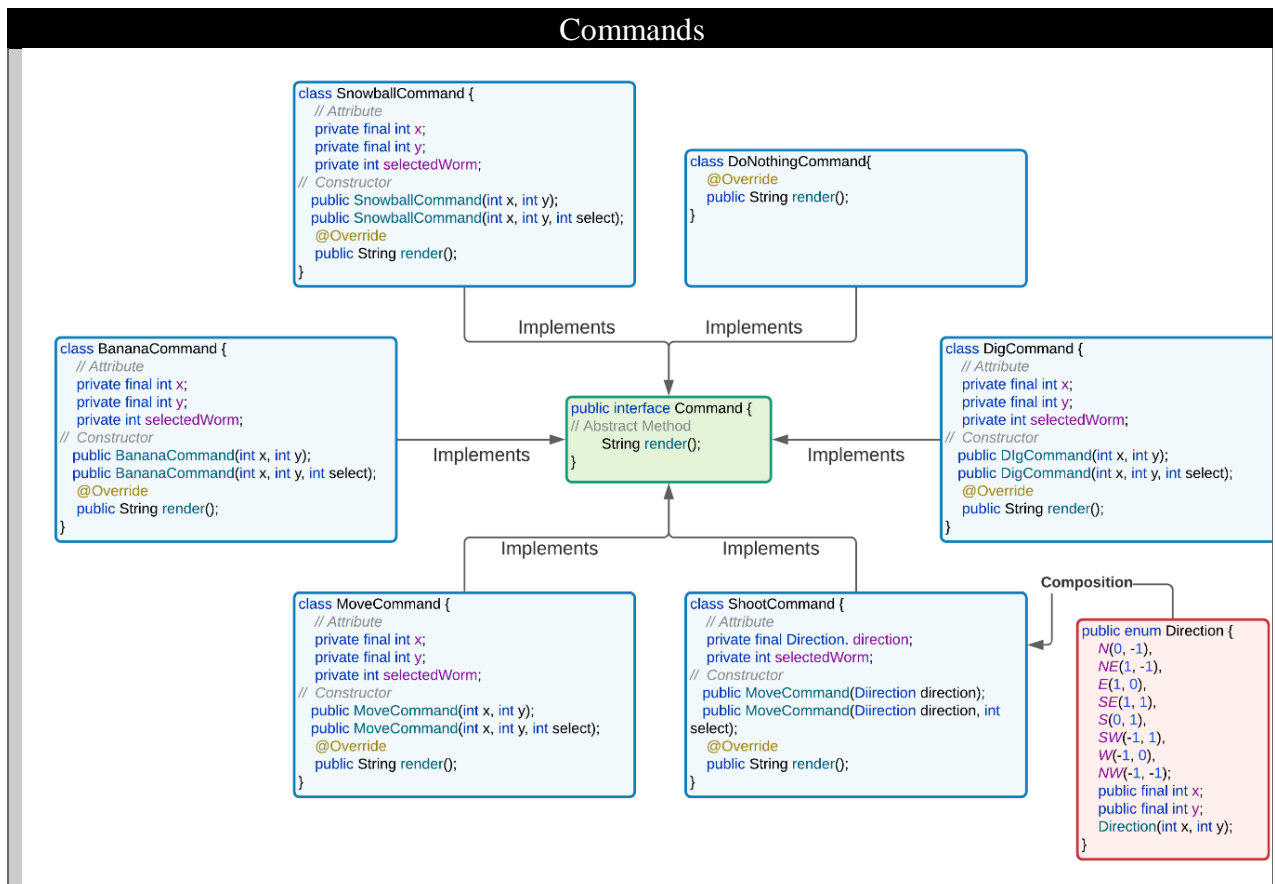
// Jika worm kita yang aktif tidak sedang ditarget lawan, maka gerak ke worm kita yang lain
if (currentWorm ≠ PredictTargetedWorm() and wormAlone()) then
    toOther ← toOtherWorm(PredictTargetedWorm())
    if(toOther ≠ null) then
        → toOther
    endif
endif

// Tidak ada bahaya yang berarti, baik lava maupun worm lawan.
// Tidak ada juga worm lawan yang bisa dilawan.
safeDig ← getAllSafeDigCommand(getAllDigCommand())
if(not(safeDig.isEmpty())) then
    // Jika ada cell yang bisa di dig dan tidak berada pada jangkauan lawan
    output("Digging surrounding cell")
    → safestDigCommand(safeDig)
else then
    // Periksa apakah lawan sedang berkumpul atau tidak
    if(isEnemyGather() and livingMyOwnWorm() > 1) then
        // Jika worm lawan sedang berkumpul dan worm kita masih lebih dari 1,
        // maka lakukan pelarian diri menuju worm kita yang lain supaya bisa menambah
        // kekuatan jika memang diperlukan adanya pertarungan nantinya
        closestFriend ← findClosestFriendWorm()
        if(closestFriend ≠ null and moveOrDigTo(closestFriend.position) ≠ null) then
            output("Escaping blockade to closest friend")
            → moveOrDigTo(closestFriend.position)
        endif
        // Jika worm lawan tidak sedang berkumpul, maka gerak ke lawan terdekat
    else if (not(isEnemyGather())) then
        closestEnemy ← getClosestOpponent()
        if (closestEnemy ≠ null and moveOrDigTo(closestEnemy.position) ≠ null) then
            output("Moving to closest enemy")
            → moveOrDigTo(closestEnemy.position)
        endif
    endif
endif

// Tidak ada strategi yang bisa dilakukan
// Melakukan randomisasi gerakan
surroundingBlocks ← getSurroundingCells(currentWorm.position.x, currentWorm.position.y)
cellIdx ← random.nextInt(surroundingBlocks.size())
output("Random Command")
block ← surroundingBlocks.get(cellIdx)
if (block.type = CellType.AIR and block.occupier = null) then
    → new MoveCommand(block.x, block.y)
else if (block.type = CellType.DIRT) then
    → DigCommand(block.x, block.y)
endif
→ DoNothingCommand()

```


4.2. Struktur Data

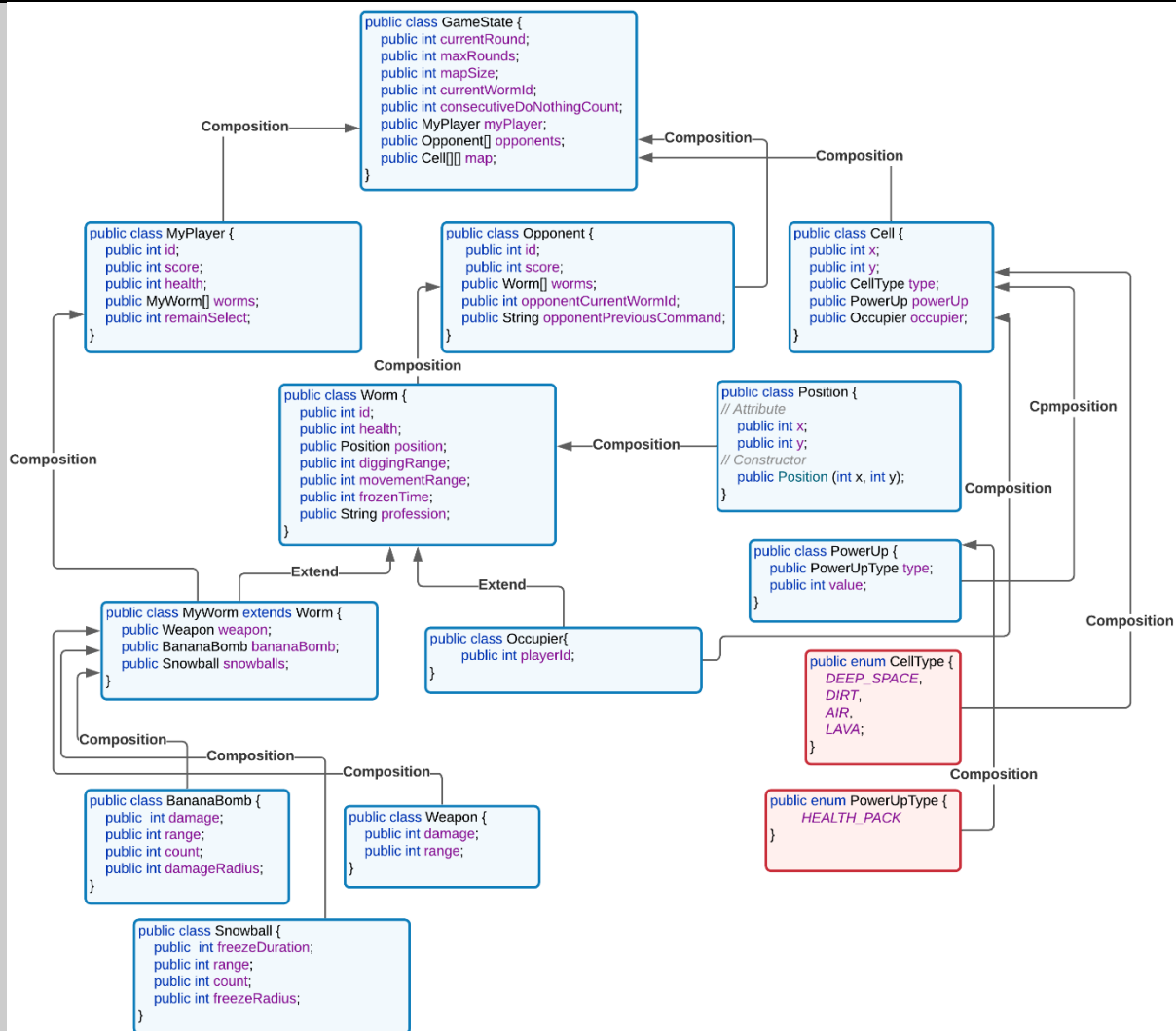


Data Tambahan:

1. *Class BananaCommand dan SnowballCommand* adalah kelas tambahan yang dibuat untuk mengirimkan *command* melempar *banana bomb* dan *snowball*.
2. *Attribute selectedWorm* pada setiap *class* kecuali *do nothing* adalah *attribute* tambahan agar dapat melakukan *command select* pada *worm* untuk melakukan semua aksi yang terdefinisi.
3. *Method render* yang *dioverride* sudah dimodifikasi sehingga bisa menerima perintah *select*.

Terdapat enam kelas terkait *command* yang digunakan untuk mengirim perintah kepada *worm* untuk melakukan suatu aksi. Perintah disampaikan menggunakan *method render* yang harus *dioverride* oleh semua *class* karena semua *class* mengimplementasikan *interface command* yang mempunyai *abstract method render*. *Method render* akan dipanggil di program utama untuk mengirimkan pesan *command* kepada *game engine*. Setiap *class* (kecuali *do nothing*) minimal mempunyai *attribute* yaitu *worm* yang diseleksi dan target melakukan aksi. Jika *shoot* maka targetnya dalam bentuk *direction*, sisanya dalam bentuk (x,y).

Entities



Data Tambahan:

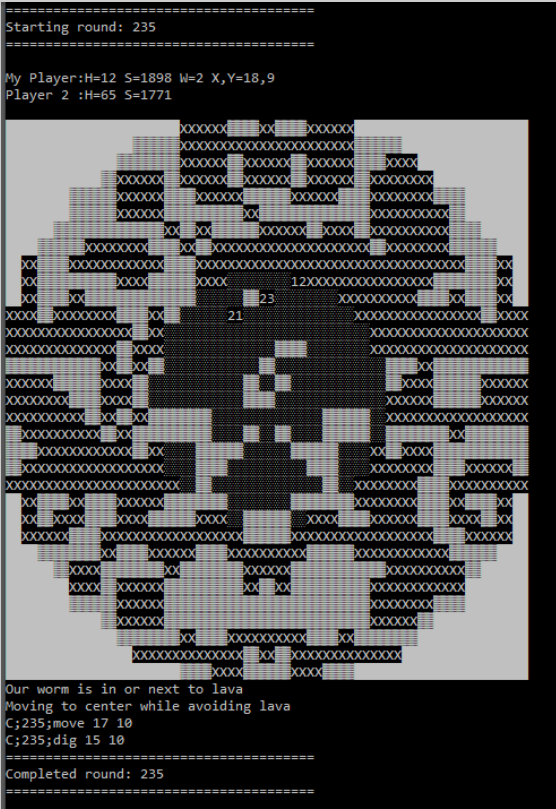
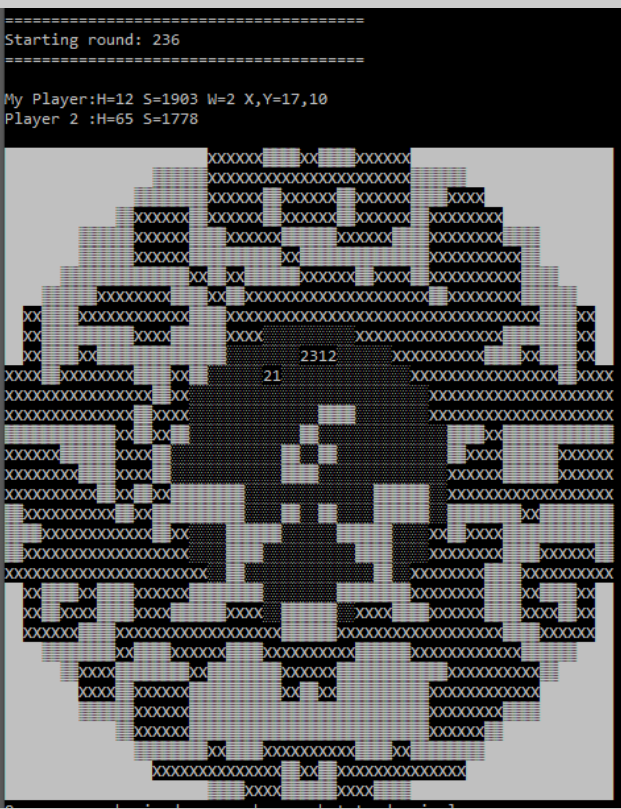
1. *Class BananaBomb* dan *Snowball* adalah *class* tambahan yang dibuat untuk mendefinisikan objek *banana bomb* yang dapat dilempar oleh *agent* dan *snowball* yang dapat dilempar oleh *technologist*. *Class BananaBomb* dan *Snowball* juga akan menjadi elemen dari *class MyWorm*.
2. *Class Occupier* adalah *class* tambahan yang dibuat untuk mendefinisikan *worm* yang menempati suatu *cell*. *Class Occupier* akan diturunkan dari *class Worm* dengan tambahan attribute *playerId* yang merepresentasikan Id dari player yang *worm*nya menempati suatu *cell* (x,y).
3. Attribute *remainSelect* pada *class MyPlayer* adalah nilai integer yang merupakan jumlah *command select* tersisa untuk *player* kita.
4. Attribute *opponentCurrentWormId* dan *opponentPreviousCommand* pada *class Opponent* adalah *attribute* tambahan yang berguna untuk mengecek id dari *worm* musuh yang sedang digunakan pada giliran ini dan mengecek *command* yang dikirimkan musuh pada giliran sebelum ini.
5. Attribute *frozenTime* dan *profession* pada *class Worm* adalah *attribute* tambahan yang berguna untuk mengetahui *type* dari *worm* musuh dan mengetahui apakah musuh sedang dalam kondisi dibekukan atau tidak.

Entities merupakan *class* yang berisikan objek-objek yang dapat di instansiasi pada *game worms*. *Class* paling utama dalam game adalah *GameState*. Dari *GameState* kita bisa mengetahui data peta pada game, data *player* kita, dan *player* musuh. Dari data *player* tersebut kita juga bisa mengetahui data *worm* kita

maupun *worm* musuh. Data *worm* yang dimiliki oleh musuh tidak bisa semuanya kita ketahui. Misalnya kita dapat menghitung jumlah *BananaBomb* dan *SnowBall* yang tersisa pada *worm agent* dan *technologist* yang kita kendalikan namun kita tidak bisa mengetahui data tersebut pada *worm* musuh. Kita juga tidak bisa mengetahui data senjata yang dimiliki oleh *worm* musuh. *Class* ini menjadi dasar-dasar perkembangan dari *strategy greedy* yang akan dibuat.

4.3. Analisis dan Pengujian

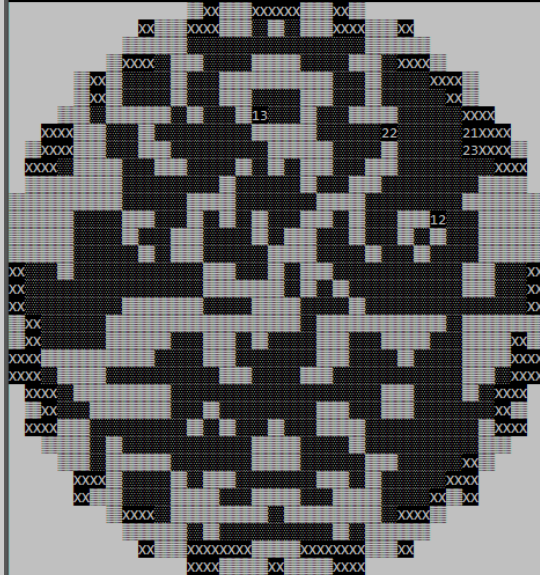
Berikut hasil pengujian strategi yang diimplementasikan beserta analisisnya. Pengujian dilakukan dengan melawan bot referensi yang sudah ada.

Pengujian 1	
Hasil	
	
Analisis	
<p>Pada round ke-235, <i>worm</i> kita sedang berada pada <i>cell</i> di sebelah lava. Karena itu, bot akan memilih strategi untuk kabur dari lava. Pada kasus ini, strategi <i>greedy</i> berhasil kabur dari lava dengan menjalankan perintah <i>move</i> mendekati titik tengah area permainan. Strategi menghindari lava akan gagal ketika <i>worm</i> yang sedang berada di dekat lava dikelilingi oleh <i>worm</i> lawan sehingga tidak bisa bergerak ataupun menggali.</p>	

Pengujian 2

Hasil

```
=====
Starting round: 116
=====
My Player:H=200 S=827 W=2 X,Y=26,12
Player 2 :H=242 S=892
```

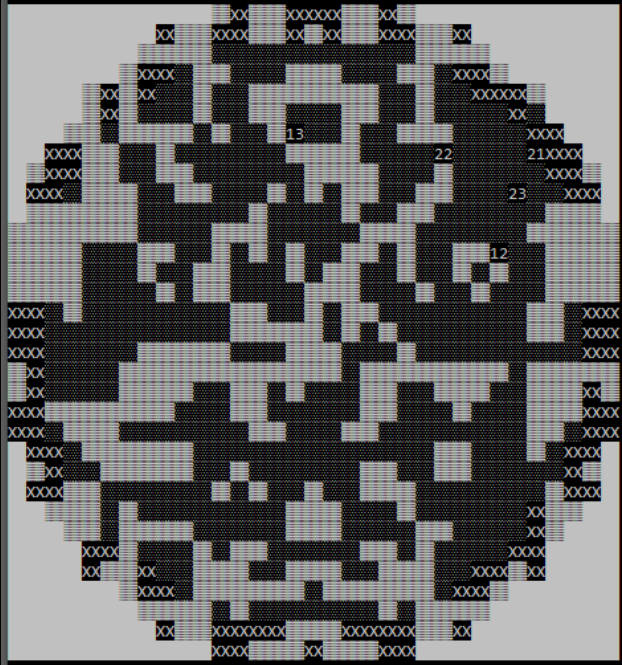


```
Bombing enemy worm
C;116;banana 28 7
C;116;move 27 9
```

```
Completed round: 116
=====
```

```
=====
Starting round: 117
=====
```

```
My Player:H=200 S=874 W=3 X,Y=15,6
Player 2 :H=219 S=890
```



Agent

Analisis

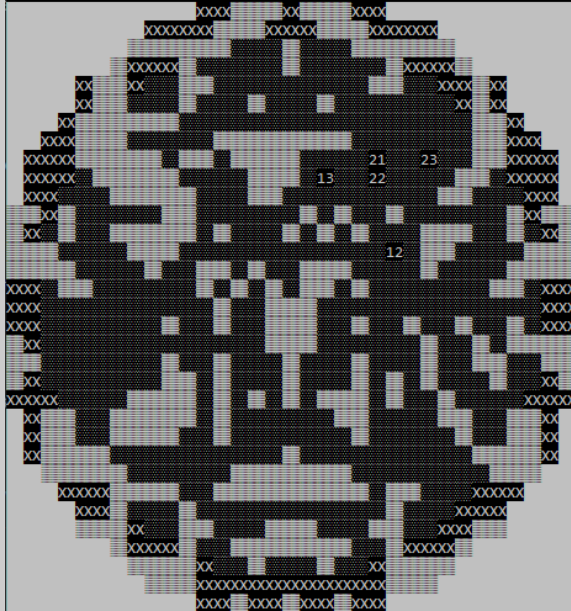
Pada pengujian ini, dapat dilihat dua *worm* lawan yaitu 21 dan 23 sedang berdekatan dan tidak ada *worm* kita yang berjarak sangat dekat dengan mereka. Worm 12 milik kita yang berjenis *Agent* berada cukup dekat dengan kedua *worm* musuh sehingga dapat dijangkau oleh pelemparan *banana bomb*. Worm 12 juga masih memiliki *banana bomb* sehingga *bot* akan memberi perintah untuk melempar *banana bomb* ke *cell* yang akan menghasilkan *damage* terbesar kepada lawan yaitu *cell* (28, 7). Dapat dilihat *bot* berhasil melempar *banana bomb* dan *health point* lawan berkurang dari 242 menjadi 219. Strategi pelemparan bom ini kurang optimal jika semua *worm* musuh bergerak tersebar atau selalu menempel dengan *worm* kita.

Pengujian 3

Hasil

```
=====
Starting round: 122
=====
```

```
My Player:H=200 S=1001 W=3 X,Y=18,9
Player 2 :H=234 S=1036
```



```
Freezing enemy worm
```

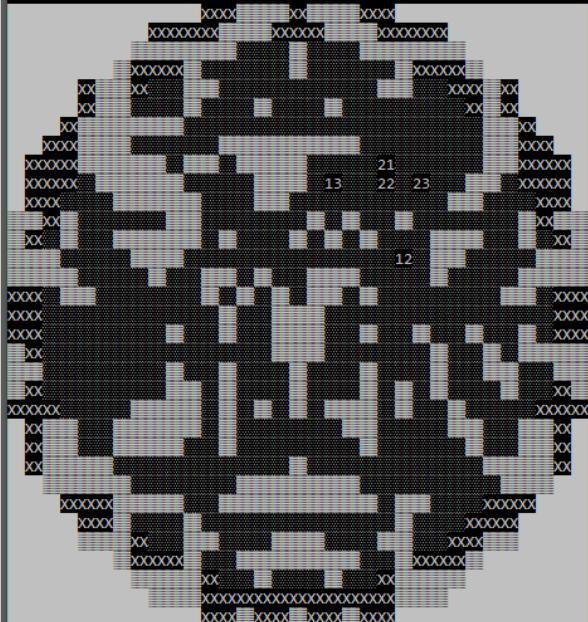
```
C;122;snowball 20 8
```

```
C;122;move 23 9
```

```
Completed round: 122
```

```
=====
Starting round: 123
=====
```

```
My Player:H=200 S=1035 W=2 X,Y=22,13
Player 2 :H=234 S=1041
```



```
C;123;select 3;shoot E
```

```
C;123;move 20 9
```

```
Completed round: 123
```

Analisis

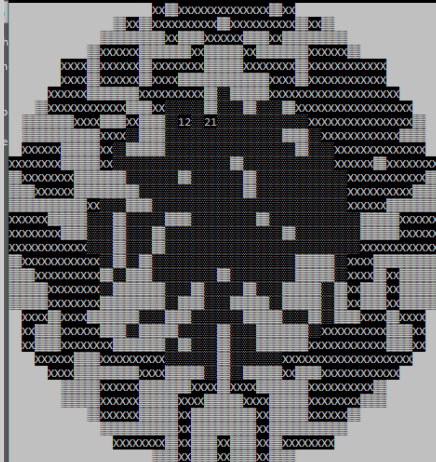
Pada pengujian ini, dapat dilihat dua *worm* lawan yaitu 21 dan 22 sedang berdekatan dan tidak ada *worm* kita yang berjarak sangat dekat dengan mereka. Worm 13 milik kita yang berjenis *Technologist* berada cukup dekat dengan kedua *worm* musuh sehingga dapat dijangkau oleh pelemparan *snowball*. Karena *worm* 13 juga masih memiliki *snowball* dan *worm* 13 juga dapat menembak *worm* 22, *bot* akan memberi perintah untuk melempar *snowball* ke *cell* yang akan membekukan *worm* lawan yang paling banyak yaitu *cell* (20, 8) yang akan menyebabkan *worm* 21 dan 22 beku. Dapat dilihat *bot* berhasil melempar *snowball* dan pada *round* selanjutnya, *worm* 13 akan dipilih untuk menyerang *worm* 22. Strategi pelemparan *snowball* ini kurang optimal jika semua *worm* musuh bergerak tersebar.

Pengujian 4

Hasil

Starting round: 192

My Player:H=44 S=1671 W=2 X,Y=13,8
Player 2 :H=55 S=1489



Our worm maybe in danger, choose what to do wisely

Move to Non Lava Cell

Escaping from danger

Move to Non Lava Cell

C:192;move 12 9

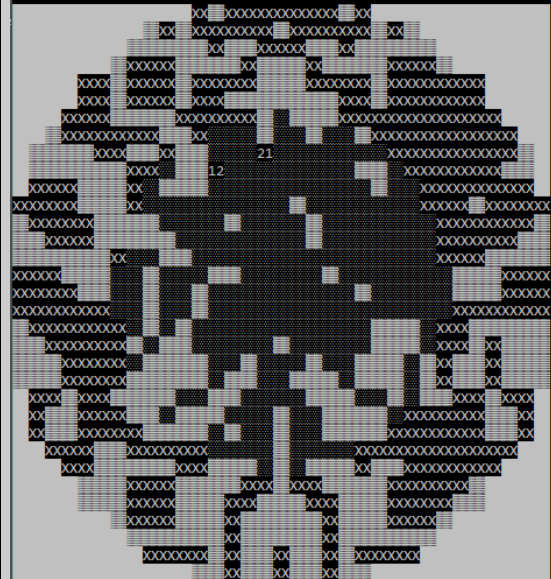
C:192;shoot W

GameError - Player 2, worm 1, round 192: Worm(player=2, id=1)'s shot BLOCKED at 11 8 from 15 8

Completed round: 192

Starting round: 193

My Player:H=44 S=1676 W=2 X,Y=12,9
Player 2 :H=55 S=1411



Analisis

Pada hasil pengujian di atas, *worm* 12 milik kita memiliki *health point* yang lebih sedikit dari *worm* lawan 21 dan berada dalam posisi yang bisa diserang oleh lawan. Oleh karena itu, *bot* akan menjalankan strategi kabur. Strategi kabur akan memberikan perintah *move* pada *worm* menuju tempat yang paling jauh dari semua musuh sambil mencoba menghindari tembakan. Pertama-tama akan dicoba kabur ke *cell* yang bukan lava atau sekitarnya. Pada hasil pengujian di atas, *worm* sudah berhasil kabur menghindari tembakan lawan dan *health point* tidak berkurang. Strategi ini kurang optimal saat *worm* kita terpojok dan dikelilingi *worm* sehingga tidak dapat kabur. Strategi ini juga kurang optimal jika *worm* lawan dapat memprediksi gerakan kita.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Dari tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 berjudul “Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan 'Worms', kami berhasil membuat sebuah *bot* dalam permainan "Worms" yang memanfaatkan algoritma greedy. Program yang dibuat tidak hanya menggunakan satu algoritma greedy melainkan beberapa algoritma greedy yang dikombinasikan untuk menghasilkan strategi yang terbaik.

2. Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2020/2021 adalah:

- a. Algoritma yang *digunakan* pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kinerja.
- b. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.
- c. Penulisan *pseudocode* tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca *pseudocode* dengan asumsi program sudah *well commented*.

DAFTAR PUSTAKA

Efendi, I. (2016, December 18). Pengertian ALGORITMA Greedy. Retrieved February 14, 2021, from <https://www.it-jurnal.com/pengertian-algoritma-greedy/>

(Materi tentang Greedy).

EntelectChallenge, B. (2019, September 02). Entelectchallenge/2019-worms. Retrieved February 11, 2021, from <https://github.com/EntelectChallenge/2019-Worms>

(Entellect Github)

Apa itu game engine. (2012, March 08). Retrieved February 14, 2021, from <https://rickykurn.wordpress.com/2012/03/08/apa-itu-game-engine/>

(Game Engine)

.