

LAPORAN TUGAS BESAR
IF2211
Strategi Algoritma

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Worms”

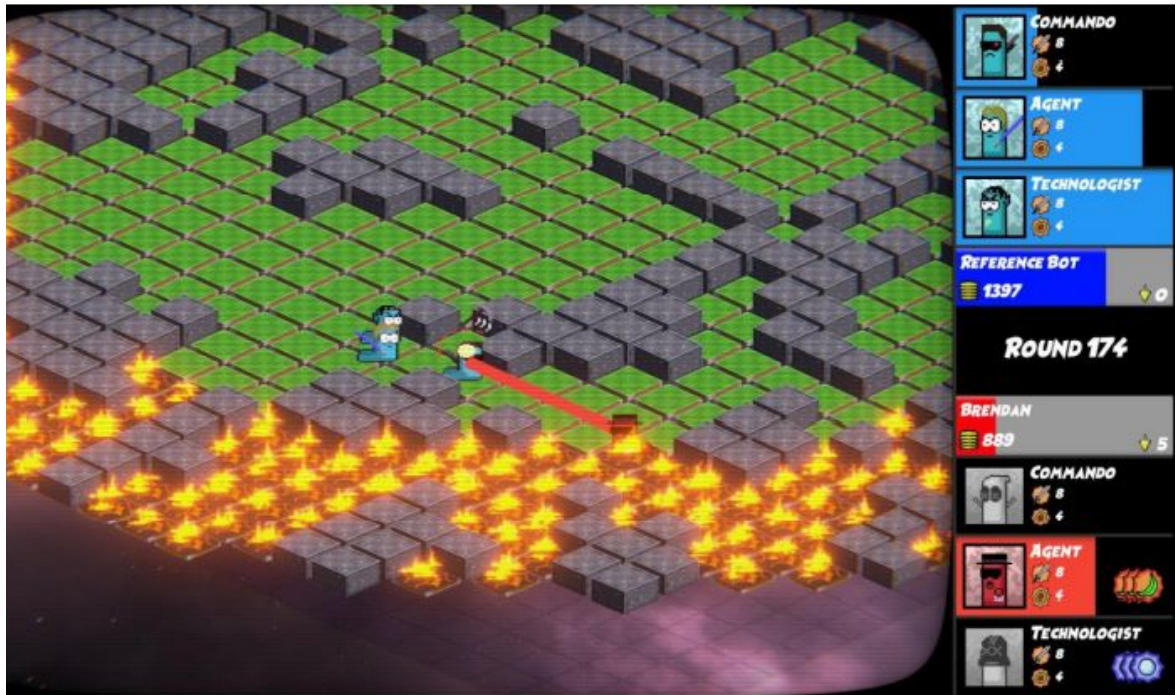


Disusun oleh:
Nathaniel Jason (13519108)
James Chandra (13519078)
Hizkia Raditya Pratama Roosadi (13519087)

TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2020/2021

I. Deskripsi Tugas

Worms adalah sebuah turn-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 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>. IF2211 Strategi Algoritma - Tugas Besar 1 2 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>).

II. Landasan Teori

Pada bab ini, dasar teori mengenai algoritma greedy serta penggunaan *game engine* untuk menjalankan worms akan dibahas.

2.1. Algoritma Greedy

Algoritma *greedy* merupakan algoritma penyelesaian masalah yang pada setiap langkah penyelesaiannya, pilihan yang optimal selalu diambil dengan harapan hasil optimal penyelesaian masalah dapat tercapai (Cormen, Leiserson, Rivest and Stein, 2009). Sesuai dengan namanya, prinsip dari algoritma ini adalah mengambil pilihan yang terbaik sekarang tanpa memikirkan konsekuensi pilihan tersebut. Pilihan tersebut dikenal juga dengan nama *local optimum*. Solusi yang diharapkan dari memilih *local optimum* adalah solusi yang paling baik secara keseluruhan atau dikenal juga sebagai *global optimum*.

Algoritma *greedy* biasanya digunakan dalam permasalahan optimasi. Persoalan optimasi adalah persoalan-persoalan yang mencari *global optimum*. Terdapat 2 jenis permasalahan optimasi yaitu maksimasi (mencari solusi terbesar) dan minimasi (mencari solusi terkecil). Contoh persoalan optimasi yang cukup terkenal dalam bidangnya adalah: Permasalahan knapsack, Permasalahan penjual keliling, kode Huffman, dsb.

Dalam algoritma *greedy*, terdapat beberapa elemen yang digunakan untuk membantu memvisualisasikan proses penyelesaian masalah. Elemen-elemen tersebut adalah

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif : memaksimumkan atau meminimumkan

Dengan menggunakan keenam elemen diatas, algoritma *greedy* melakukan proses penyelesaian masalahnya (Munir, 2021). Dalam tugas ini, algoritma *greedy* akan digunakan untuk menyelesaikan permasalahan optimasi “mempercepat kekalahan lawan dalam permainan Worms”. Permainan tersebut akan dimainkan dalam sebuah *game engine* yang akan dijelaskan juga dalam laporan ini.

2.2. Game Engine Permainan Worms

Sub-bab ini akan menjelaskan mengenai *game engine* yang digunakan untuk permainan Worms, cara penambahan pemain, bagian apa yang dikerjakan serta bagaimana implementasi algoritma *greedy*, dan terakhir pembahasan mengenai menjalankan *game engine* untuk permainan.

2.2.1. Prerequisit

Seperti yang sudah disebutkan, permainan Worms dalam tugas ini dijalankan menggunakan *game engine* yang sudah tersedia. *Game engine* untuk keperluan menjalankan permainan Worms dapat diunduh pada laman berikut: (<https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2>). Beberapa perangkat lunak yang diperlukan sebagai prerequisit untuk menjalankan *game engine*, sesuai yang tertulis pada dokumentasi & aturan permainan, adalah Java (direkomendasikan Java 8), IntelliJ IDEA (IDE untuk membuat program dalam bahasa Java), Maven (perangkat pembangun program untuk Java), dan Node JS.

2.2.2. Starter-pack

File yang diunduh dari laman adalah file zip yang didalamnya terdapat folder bernama *starter-pack*. *Game engine* dimuat dalam folder *starter-pack* tersebut. Tampilan dalam dari folder *starter-pack* adalah sebagai berikut

Name	Date modified	Type	Size
.idea	2/8/2021 11:09 AM	File folder	
match-logs	2/8/2021 9:09 PM	File folder	
reference-bot	9/2/2019 5:38 PM	File folder	
starter-bots	2/7/2021 9:42 PM	File folder	
ec-2019-game-engine-jvm-full-2019.3.2	9/2/2019 5:38 PM	Executable Jar File	1,898 KB
game-config	9/2/2019 5:38 PM	JSON Source File	2 KB
game-runner-config	2/8/2021 11:18 AM	JSON Source File	1 KB
game-runner-jar-with-dependencies	9/2/2019 5:38 PM	Executable Jar File	9,766 KB
makefile	9/2/2019 5:38 PM	File	1 KB
run	9/2/2019 5:38 PM	Windows Batch File	1 KB

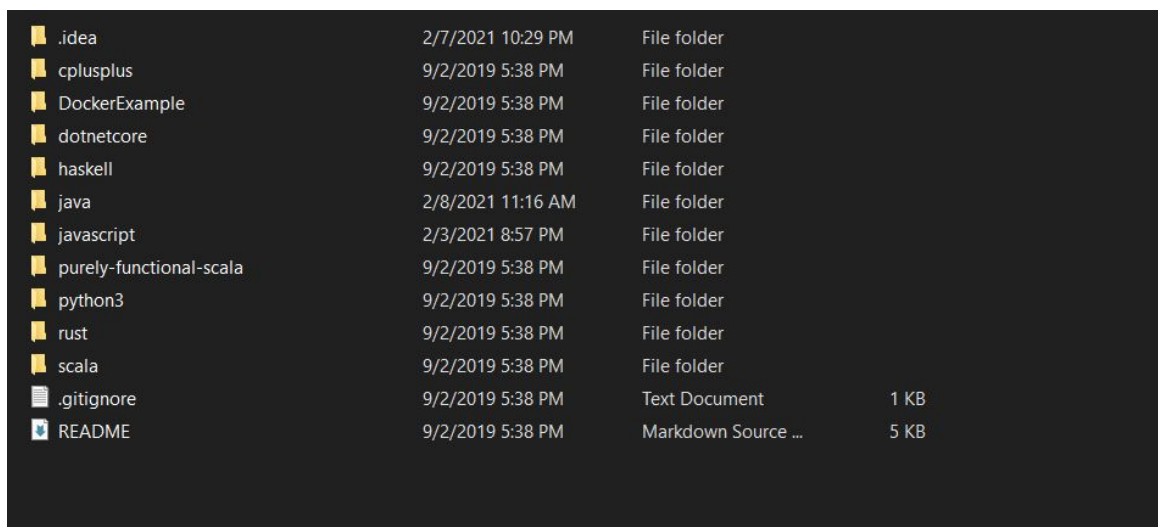
Gambar 2. Tampilan *starter-pack*

Seperti yang dapat dilihat, *game engine* sudah tersedia dalam bentuk Executable Jar File. File tersebut digunakan untuk menjalankan permainan. Aturan dan cara kerja permainan dapat

dilihat secara lebih lanjut dalam laman berikut: (<https://github.com/EntelectChallenge/2019-Worms/blob/develop/game-engine/game-rules.md>).

2.2.3. Pemain dan Penambahan Pemain

Pada permainan ini, entitas pemain yang akan diuji adalah dalam bentuk bot yang dapat menjalankan perintah sesuai dengan program. Pada awalnya, terdapat 2 bot yang dapat digunakan sebagai pemain. Bot pertama adalah reference bot, bot bawaan dari *game engine* yang menjalankan program dalam bahasa JavaScript untuk melakukan permainan. Bot kedua adalah starter bot, bot yang dapat diprogram ulang sesuai kebutuhan. Sebagai penyesuaian terhadap permainan Worms yang dijadikan kompetisi oleh organisasi Entelect pada tahun 2019, starter bot tersedia dalam berbagai bahasa, seperti python, C++, Java, dan lainnya. Bagian starter bot ini yang akan diprogram ulang sesuai kebutuhan pada tugas ini. Tampilan folder starter-bot adalah sebagai berikut.



.idea	2/7/2021 10:29 PM	File folder	
cplusplus	9/2/2019 5:38 PM	File folder	
DockerExample	9/2/2019 5:38 PM	File folder	
dotnetcore	9/2/2019 5:38 PM	File folder	
haskell	9/2/2019 5:38 PM	File folder	
java	2/8/2021 11:16 AM	File folder	
javascript	2/3/2021 8:57 PM	File folder	
purely-functional-scala	9/2/2019 5:38 PM	File folder	
python3	9/2/2019 5:38 PM	File folder	
rust	9/2/2019 5:38 PM	File folder	
scala	9/2/2019 5:38 PM	File folder	
.gitignore	9/2/2019 5:38 PM	Text Document	1 KB
README	9/2/2019 5:38 PM	Markdown Source ...	5 KB

Gambar 3. Tampilan Starter Bots dan Variasi Bahasa Pemrogramannya

Untuk menambahkan pemain/bot yang diinginkan sebagai pemain, programmer harus mengedit file `game-runner-config.json` yang berada di direktori `starter-pack`. Bagian dalam dari `game-runner-config.json` ditampilkan sebagai berikut

```

1  {
2    "round-state-output-location": "./match-logs",
3    "game-config-file-location": "game-config.json",
4    "game-engine-jar": "./ec-2019-game-engine-jvm-full-2019.3.2.jar",
5    "verbose-mode": true,
6    "max-runtime-ms": 1000,
7    "player-a": "./starter-bots/java",
8    "player-b": "./reference-bot/javascript",
9    "max-request-retries": 10,
10   "request-timeout-ms": 5000,
11   "is-tournament-mode": false,
12   "tournament": {
13     "connection-string": "",
14     "bots-container": "",
15     "match-logs-container": "",
16     "game-engine-container": "",
17     "api-endpoint": "http://localhost"
18   }
19 }
20

```

Gambar 4. Tampilan File game-runner-config.json

Pemrogram perlu mengganti bagian kode dalam file game-runner-config.json dengan alamat bot yang ingin dijadikan pemain agar dapat dimasukan ke permainan Worms. Bagian kode yang harus diubah adalah bagian berikut

```

{
  "round-state-output-location": "./match-logs",
  "game-config-file-location": "game-config.json",
  "game-engine-jar": "./ec-2019-game-engine-jvm-full-2019.3.2.jar",
  "verbose-mode": true,
  "max-runtime-ms": 1000,
  "player-a": "./starter-bots/java",
  "player-b": "./reference-bot/javascript",
  "max-request-retries": 10,
  "request-timeout-ms": 5000,
  "is-tournament-mode": false,
  "tournament": {
    "connection-string": "",
    "bots-container": "",
    "match-logs-container": "",
    "game-engine-container": "",
    "api-endpoint": "http://localhost"
  }
}

```

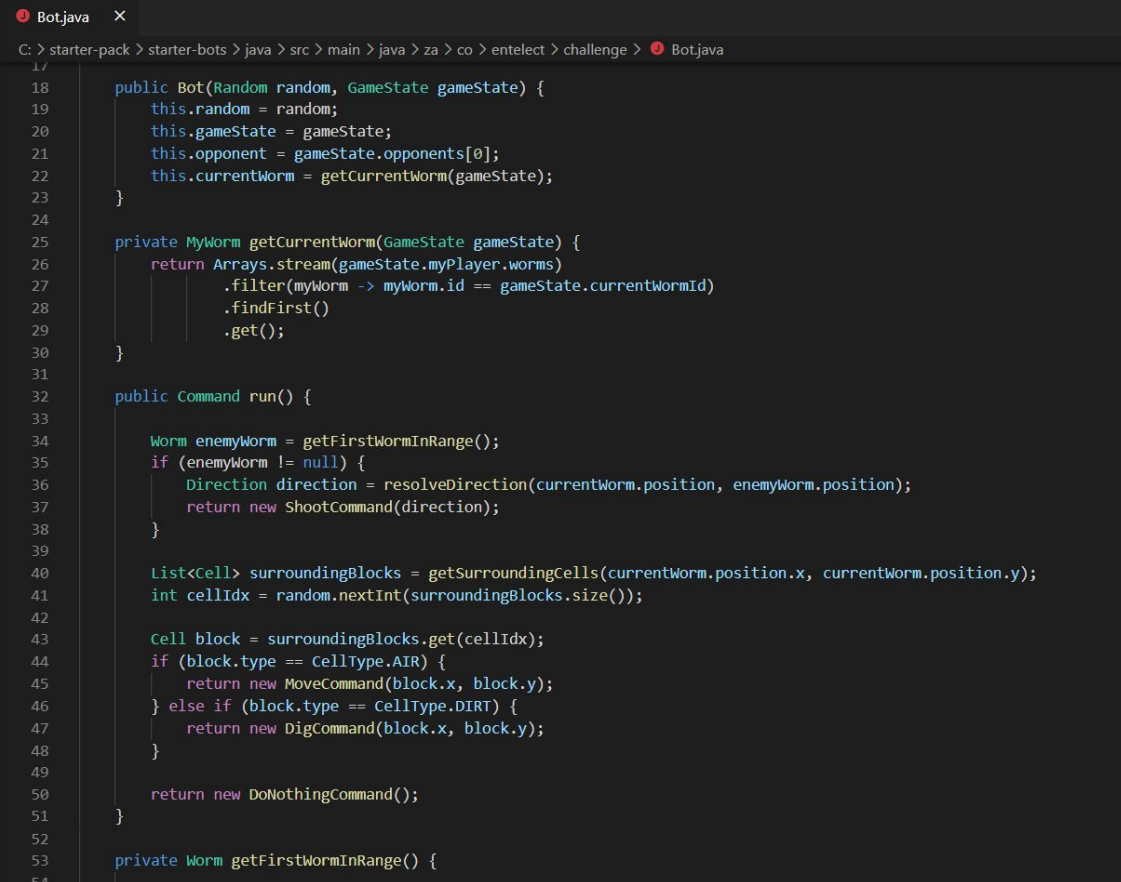
Gambar 5. Bagian yang Diedit

Untuk tugas ini, karena bahasa pemrograman yang digunakan sesuai spesifikasi tugas adalah Java, file executable .jar hasil build dari program bahasa Java yang ditulis perlu

diletakkan di direktori starter-pack. Bab selanjutnya akan membahas lebih detail mengenai hal tersebut serta bot Java yang ditulis secara keseluruhan.

2.2.4. Bot.java

Sesuai dengan spesifikasi, bahasa pemrograman yang digunakan dalam tugas ini adalah Java. File bot.java adalah file yang akan digunakan untuk tugas ini. Didalam bot.java, sudah tersedia beberapa algoritma yang dapat digunakan untuk memainkan Worms. Bersama dengan algoritma ini, penulis akan mengimplementasikan algoritma *greedy* serta mengembangkan atau menambahkan algoritma lain sesuai kebutuhan. Perlu diingat lagi, objektif utama dari tugas pemrograman ini adalah membunuh Worm lawan secepat mungkin. Berikut adalah tampilan sekilas dari file bot.java.



```
17
18 public Bot(Random random, GameState gameState) {
19     this.random = random;
20     this.gameState = gameState;
21     this.opponent = gameState.opponents[0];
22     this.currentWorm = getCurrentWorm(gameState);
23 }
24
25 private MyWorm getCurrentWorm(GameState gameState) {
26     return Arrays.stream(gameState.myPlayer.worms)
27         .filter(myWorm -> myWorm.id == gameState.currentWormId)
28         .findFirst()
29         .get();
30 }
31
32 public Command run() {
33
34     Worm enemyWorm = getFirstWormInRange();
35     if (enemyWorm != null) {
36         Direction direction = resolveDirection(currentWorm.position, enemyWorm.position);
37         return new ShootCommand(direction);
38     }
39
40     List<Cell> surroundingBlocks = getSurroundingCells(currentWorm.position.x, currentWorm.position.y);
41     int cellIdx = random.nextInt(surroundingBlocks.size());
42
43     Cell block = surroundingBlocks.get(cellIdx);
44     if (block.type == CellType.AIR) {
45         return new MoveCommand(block.x, block.y);
46     } else if (block.type == CellType.DIRT) {
47         return new DigCommand(block.x, block.y);
48     }
49
50     return new DoNothingCommand();
51 }
52
53 private Worm getFirstWormInRange() {
54
```

Gambar 6. Tampilan bot.java

Setelah program diimplementasikan dalam bot.java, bot kemudian harus di build menggunakan Maven sehingga terbentuk executable jar file dari bot tersebut. File .jar ini adalah file yang akan dibaca oleh *game engine* sebagai pemain. Maka dari itu, alamat ke bot yang dituliskan pada bagian game-runner-config.json, khusus untuk bot yang menggunakan bahasa

Java, adalah alamat direktori tempat executable jar file tersimpan. Setelah menuliskan alamat yang sesuai, permainan Worms dapat langsung dijalankan.

2.3. Menjalankan Permainan

Untuk menjalankan permainan, pengguna cukup menjalankan file run.bat yang terdapat pada direktori starter-pack. Setelah dijalankan, permainan akan berjalan secara otomatis. Pemenang akan diumumkan pada akhir permainan. Berikut adalah tampilan dari permainan Worms di command line.



Gambar 7. Tampilan Permainan

Tampilan permainan dalam command line dapat diklasifikasikan sebagai membosankan. Untuk meningkatkan kualitas visualisasi pengguna, visualizer untuk permainan dapat digunakan. Visualizer serta cara penggunaannya dapat ditemukan pada link berikut: <https://github.com/dlweatherhead/entelect-challenge-2019-visualiser/releases/tag/v1.0f1>.

III. Pemanfaatan strategi greedy

Pemanfaatan strategi greedy akan dilakukan dalam pemrograman bot worms disini, algoritme greedy seperti yang sebelumnya telah dipaparkan merupakan algoritme yang akan mengambil local optimum/pilihan langkah yang memberikan hasil paling {kata sifat}, sehingga merupakan pilihan yang memberikan *outcome* paling diinginkan.

3.1. Mapping persoalan Worms menjadi elemen - elemen algoritma *greedy*

3.1.1 Himpunan kandidat

Command move, dig, shoot, bananabomb, snowball. koordinat pada *map* permainan.

3.1.2 Himpunan solusi

Command move, dig, shoot, bananabomb, snowball, koordinat pada *map* permainan yang diatur sedemikian rupa untuk mempercepat pembunuhan *worms* musuh.

3.1.3 Fungsi solusi

Fungsi untuk menentukan apakah semua *worms* musuh sudah mati atau belum.

3.1.4 Fungsi seleksi

Jika memungkinkan untuk menyerang, maka akan memiliki prioritas *bananabomb, snowball*, dan terakhir *shoot*. *Command* yang dipilih harus memberikan *damage* ke lawan dengan lebih besar dari 0. Koordinat yang dipilih adalah koordinat yang akan memberikan total serangan terbesar ke musuh, untuk serangan yang bersifat *area*. Untuk *shoot*, akan dipilih arah mata angin yang dapat mengenai musuh tanpa terhalang *worms* tim sendiri ataupun *dirt*. Jika tidak memungkinkan untuk menyerang maka akan bergerak mendekati musuh terdekat.

3.1.5 Fungsi kelayakan

Memeriksa apakah *command* dan koordinat adalah aksi yang valid atau tidak. Hal ini dapat dicapai dengan cara mengecek status apakah suatu *worms* bisa melakukan serangan tertentu dan memiliki *ammo* lebih besar dari 0. Untuk koordinat yang valid, kita dapat memproyeksikan semua koordinat yang valid sesuai dengan aturan, sehingga dapat dipastikan bahwa kemungkinan koordinat yang dipilih adalah valid.

3.1.6 Fungsi obyektif

Memilih command serangan dengan prioritas *bananabomb*, *snowball*, lalu *shoot*. Koordinat yang dipilih adalah koordinat yang menghasilkan *damage* yang paling banyak ke tim musuh. Bergerak dengan arah menuju *worms* musuh yang terdekat untuk mempercepat menyerang musuh.

3.2. Alternatif solusi *greedy*

3.2.1 Alternatif solusi 1

Worms akan selalu berusaha untuk melakukan serangan jika memenuhi kondisi mungkin menyerang, yaitu terdapat musuh yang dapat terkena serangan pada jarak dan radius yang valid. Serangan dilakukan dengan *command shoot*, *bananabomb*, ataupun *snowball*. Target koordinat atau arah mata angin untuk *command* serangan dipilih secara acak. *Command* serangan yang dipilih pun diprioritaskan dari serangan spesial (*bananabomb*, *snowball*), jika tidak memungkinkan lagi untuk menggunakan serangan spesial. Jika *worms* tidak memungkinkan untuk menyerang, maka ia akan bergerak ke arah *worms* musuh yang paling dekat dari dia.

3.2.2 Alternatif solusi 2

Worms akan selalu berusaha untuk melakukan serangan jika memenuhi kondisi mungkin menyerang, yaitu terdapat musuh yang dapat terkena serangan pada jarak dan radius yang valid. Serangan dilakukan dengan *command shoot*, *bananabomb*, ataupun *snowball*. *Command* serangan yang dipilih pun diprioritaskan dari serangan spesial (*bananabomb*, *snowball*), jika tidak memungkinkan lagi untuk menggunakan serangan spesial. Target koordinat atau arah mata angin untuk *command* serangan dipilih berdasarkan *worms* musuh pertama kali yang terdeteksi dapat terkena serangan. Jika *worms* tidak memungkinkan untuk menyerang, maka ia akan bergerak ke arah *worms* musuh yang paling dekat dari dia.

3.2.3 Alternatif solusi 3

Worms akan *greedy* mendapatkan poin sebanyak mungkin tanpa menyerang dan berusaha untuk memberikan hasil seri, tidak menang dan tidak kalah, yaitu dengan cara bertahan sampai *round* 400. Untuk mendapatkan poin semaksimal mungkin tanpa menyerang adalah dengan cara *command move*, *dig*, dan *snowball*. Untuk *command snowball*, akan dipilih koordinat yang akan membekukan *worms* musuh paling banyak, jika terdapat beberapa koordinat yang membekukan *worms* musuh paling banyak, maka akan dipilih koordinat yang membekukan *worms* sendiri paling sedikit. Untuk *command move* dan *dig* akan diprioritaskan untuk selalu menghindari dari serangan musuh, yaitu dengan cara memproyeksikan semua kemungkinan koordinat yang bisa terkena serangan musuh.

3.3. Analisis efisiensi solusi *greedy*

3.3.1 Alternatif solusi 1

Implementasi dari alternatif solusi ini cukup mudah. Untuk mengecek apakah kita memungkinkan menyerang atau tidak, kita hanya perlu memproyeksikan seluruh kemungkinan koordinat yang valid. Cara memproyeksikannya pun terdapat 2 tahap untuk *worms* dengan *profession technologist* dan *agent*. Untuk *command shoot*. Kita hanya perlu untuk mengiterasi semua koordinat yang berjarak sejauh *range command shoot*, tentunya dengan mengikuti pola yang valid berdasarkan aturan permainan ini. Untuk *command snowball* dan *bananabomb*, proyeksi dilakukan 2 tahap, pertama kita akan memproyeksikan koordinat mana saja yang valid untuk dilempari *snowball* atau *bananabomb*. Kemudian kita akan memproyeksikan lagi koordinat yang akan terkena efek serangan *snowball* atau *bananabomb*, untuk setiap koordinat yang valid untuk dilempari. Jika terdapat *worms* musuh pada salah satu koordinat tersebut maka akan dinyatakan bahwa *worms* memungkinkan untuk menyerang. Implementasi untuk kondisi memungkinkan menyerang pun sangat sederhana, yaitu hanya dengan cara melakukan pilihan acak pada koordinat yang valid untuk *command* serangan, kompleksitasnya adalah konstan. Untuk kasus tidak memungkinkan menyerang, kompleksitasnya juga konstan, karena kita hanya mengiterasi ke semua worm musuh, yang dimana jumlahnya konstan. Untuk efisiensi, bisa dikatakan ini yang paling efisien dari antara ketiga alternatif solusi.

3.3.2 Alternatif solusi 2

Implementasi dari alternatif solusi ini lebih kompleks dibandingkan dengan alternatif solusi 1. Untuk mengecek apakah kita memungkinkan menyerang atau tidak, kita hanya perlu memproyeksikan seluruh kemungkinan koordinat yang valid. Cara memproyeksikannya pun terdapat 2 tahap untuk *worms* dengan *profession technologist* dan *agent*. Untuk *command shoot*. Kita hanya perlu untuk mengiterasi semua koordinat yang berjarak sejauh *range command shoot*, tentunya dengan mengikuti pola yang valid berdasarkan aturan permainan ini. Untuk *command snowball* dan *bananabomb*, proyeksi dilakukan 2 tahap, pertama kita akan memproyeksikan koordinat mana saja yang valid untuk dilempari *snowball* atau *bananabomb*. Kemudian kita akan memproyeksikan lagi koordinat yang akan terkena efek serangan *snowball* atau *bananabomb*, untuk setiap koordinat yang valid untuk dilempari. Jika terdapat *worms* musuh pada salah satu koordinat tersebut maka akan dinyatakan bahwa *worms* memungkinkan untuk menyerang. Implementasi untuk kondisi memungkinkan menyerang adalah sebagai berikut. Untuk setiap koordinat yang valid untuk dilakukan serangan, kita akan mencari koordinat mana yang pertama kali ditemukan *worms* musuh. Untuk kasus tidak memungkinkan menyerang, kompleksitasnya konstan, karena kita hanya mengiterasi ke semua worm musuh, yang dimana jumlahnya konstan. Untuk efisiensi, ini kurang efisien dibandingkan dengan alternatif solusi 1, karena pada kasus memungkinkan menyerang, kita melakukan iterasi lagi untuk mencari koordinat mana yang terdapat *worms* musuh.

3.3.3 Alternatif solusi 3

Implementasi dari solusi ini bisa dikatakan hampir mirip dengan alternatif solusi 2. Kita disini menggunakan prinsip yang sama pada alternatif 1 dan 2, yaitu pengecekan untuk kondisi mungkin menyerang. Tetapi bedanya pada alternatif solusi ini, kita akan memproyeksikan semua koordinat yang mungkin terkena serangan oleh musuh, yang kemudian akan kita jadikan acuan sebagai *command move* atau *dig* yang akan kita lakukan dengan tujuan menghindar. Cara memproyeksikannya pun terdapat 2 tahap untuk *worms* dengan *profession technologist* dan *agent*. Untuk *command shoot*. Kita hanya perlu untuk mengiterasi semua koordinat yang berjarak sejauh *range command shoot*, tentunya dengan mengikuti pola yang valid berdasarkan aturan permainan ini. Untuk *command snowball* dan *bananabomb*, proyeksi dilakukan 2 tahap, pertama kita akan memproyeksikan koordinat mana saja yang valid untuk dilempari *snowball* atau *bananabomb*. Kemudian kita akan memproyeksikan lagi koordinat yang akan terkena efek serangan *snowball* atau *bananabomb*, untuk setiap koordinat yang valid untuk dilempari. *Worms* tim kita sendiri akan memilih *command move* atau *dig* yang tidak terdapat pada kumpulan koordinat yang mungkin terkena efek dari serangan musuh itu. Jika tidak memungkinkan maka, *worms* akan melakukan serangan juga ke musuh. Untuk implementasi kasus tidak memungkinkan menghindar, kita bergerak dengan prioritas menjauh dari *worms* musuh.

3.3.4 Strategi yang dipilih

Implementasi dari strategi ini dimulai dengan mengecek apakah *worms* yang kita kendalikan saat ini memungkinkan untuk menyerang atau tidak. Kita akan mengecek untuk 3 kemungkinan *command* serangan yang dilakukan. Suatu *worms* dikatakan memungkinkan untuk menyerang jika damage maksimum dari 3 kemungkinan serangan yang ada adalah lebih besar dari 0. Terdapat cara tersendiri untuk setiap *command* tersebut. Untuk mencari *command bananabomb* yang terbaik, dilakukan dengan cara memproyeksikan semua koordinat yang mungkin untuk dilempari *bananabomb*. Kemudian untuk setiap koordinat tersebut kita akan memproyeksikan *area* atau koordinat yang terkena efek dari serangan tersebut. Akan dipilih koordinat yang akan mengenai musuh paling banyak, dan minimal terkena 1 dikurangi jumlah *worms* musuh yang masih hidup. Misalkan, jika musuh memiliki 3 *worms* yang hidup, maka minimal harus mengenai 2 *worms* musuh. Untuk mencari *command snowball* yang terbaik, dilakukan cara yang sama pada kasus *command bananabomb*. Untuk mencari *command shoot* yang terbaik, kita akan memproyeksikan semua koordinat yang mungkin untuk terkena serangan. Kemudian kita akan pilih koordinat yang terdapat *worms* musuh. Kita juga akan mengecek untuk mencapai koordinat tersebut, apakah terdapat halangan berupa *worms* tim sendiri ataupun *dirt*. Untuk ketiga kasus *command serangan*, jika tidak ditemukan koordinat yang valid, maka akan mengembalikan damage 0. Jika *worms* tidak memungkinkan untuk menyerang, maka ia akan bergerak ke arah *worms* musuh yang paling dekat dari dia. Efisiensi dari algoritma ini cukup baik, karena semua proses pengulangan dan iterasi pada algoritma ini dilakukan sebanyak suatu konstanta. Sebagai contoh kita mengiterasi ke semua elemen pada array *worms* musuh yang dimana jumlahnya sama

pada setiap game. Sebagai contoh yang lain, kita juga mengiterasi ke semua koordinat yang valid untuk *bananabomb*, yang dimana proses pengulangannya juga dilakukan sebanyak suatu konstanta, yaitu *range* dari *bananabomb*.

3.4. Analisis efektivitas solusi *greedy*

3.4.1 Alternatif solusi 1

Alternatif solusi ini bisa dikatakan adalah yang paling tidak efektif dibandingkan dengan alternatif solusi lainnya, walaupun tingkat efisiensinya paling baik. Hal ini dikarenakan pada kondisi memungkinkan menyerang, koordinat serangan dipilih secara acak, sehingga kemungkinan musuh untuk terkena serangan pun juga acak. Ada juga kemungkinan bahwa *worms* kita sendiri terkena serangan.

3.4.2 Alternatif solusi 2

Alternatif solusi ini cukup efektif dalam memenangkan permainan ini, dikarenakan untuk kondisi memungkinkan menyerang, kita sudah pasti akan mengenai *worms* musuh. Salah satu kekurangan dari alternatif solusi ini adalah kita mungkin saja tidak menghasilkan *damage* yang maksimal ke *worms musuh*, dikarenakan pemilihan koordinat serangan dilakukan dengan cara mencari koordinat pertama yang dapat mengenai musuh. Sehingga, untuk serangan yang memiliki *area damage* atau *effect* tidak akan memberikan hasil yang maksimal.

3.4.3 Alternatif solusi 3

Untuk efektifitas dari pergerakan cukup baik dalam hal bertahan, dikarenakan kita dapat terhindar dari serangan musuh. Efektifitas untuk alternatif solusi ini kurang baik secara umum. Hal ini dikarenakan, tujuan utama dari permainan ini adalah membunuh *worms* musuh untuk memenangkan permainan. Kurang efektifnya alternatif solusi ini juga dikarenakan sangat sulit sekali untuk menghindari jika kita sudah pada *round* yang mendekati akhir, dimana *lava* sudah mulai meluas yang mengakibatkan mengecilnya kumpulan koordinat yang mungkin untuk ditempati. Hal ini juga akan membuat kita semakin sulit menghindari.

3.4.4 Solusi yang dipilih

Solusi ini cukup efektif membawa kita mendapat ke tujuan utama dari permainan ini, ataupun kondisi memenangkan permainan ini, yaitu mengalahkan *worms* musuh dengan cara membunuh sebelum semua *worms* kita sendiri habis terbunuh. Untuk kondisi memungkinkan menyerang,

kita mencari serangan dan koordinat yang akan memberikan efek serangan yang paling besar, sehingga kita memaksimalkan penggunaan senjata kita.

3.5. Strategi *greedy* yang dipilih

Secara garis besar, terdapat 2 kondisi dalam strategi ini, kondisi memungkinkan menyerang dan tidak. Suatu kondisi dikatakan tidak memungkinkan menyerang, jika damage dari *command* serangan terbaik adalah 0. Cara mendapatkan serangan terbaiknya adalah sebagai berikut. Kita akan memprioritaskan *bananabomb* terlebih dahulu, lalu baru *snowball*, dan yang terakhir adalah *shoot*. Terdapat cara tersendiri untuk setiap *command* tersebut. Untuk mencari *command bananabomb* yang terbaik, dilakukan dengan cara memproyeksikan semua koordinat yang mungkin untuk dilempari *bananabomb*. Kemudian untuk setiap koordinat tersebut kita akan memproyeksikan *area* atau koordinat yang terkena efek dari serangan tersebut. Akan dipilih koordinat yang akan mengenai musuh paling banyak, dan minimal terkena 1 dikurangi jumlah *worms* musuh yang masih hidup. Misalkan, jika musuh memiliki 3 *worms* yang hidup, maka minimal harus mengenai 2 *worms* musuh. Untuk mencari *command snowball* yang terbaik, dilakukan cara yang sama pada kasus *command bananabomb*. Untuk mencari *command shoot* yang terbaik, kita akan memproyeksikan semua koordinat yang mungkin untuk terkena serangan. Kemudian kita akan pilih koordinat yang terdapat *worms* musuh. Kita juga akan mengecek untuk mencapai koordinat tersebut, apakah terdapat halangan berupa *worms* tim sendiri ataupun *dirt*. Untuk ketiga kasus *command serangan*, jika tidak ditemukan koordinat yang valid, maka akan mengembalikan damage 0. Jika *worms* tidak memungkinkan untuk menyerang, maka ia akan bergerak ke arah *worms* musuh yang paling dekat dari dia.

Pertimbangannya dan konsiderasi dalam memilih solusi ini adalah dari efektivitas dan efisiensi. Untuk efektivitas, solusi ini cukup efektif membawa kita mendapat ke tujuan utama dari permainan ini, ataupun kondisi memenangkan permainan ini, yaitu mengalahkan *worms* musuh dengan cara membunuh sebelum semua *worms* kita sendiri habis terbunuh. Untuk kondisi memungkinkan menyerang, kita mencari serangan dan koordinat yang akan memberikan efek serangan yang paling besar, sehingga kita memaksimalkan penggunaan senjata kita. Untuk efisiensi dari solusi ini, memang kalau dibandingkan dengan alternatif solusi 1 sampai 3, paling tidak efisien, tetapi dengan pertimbangan bahwa jumlah input tidak terlalu besar, dan kecepatan menjalankan program tidak berpengaruh terhadap usaha kita dalam memenangkan permainan, maka, efektivitas tidak terlalu menjadi konsiderasi utama untuk kelompok kami. Tetapi jika dibandingkan atau dihitung kompleksitasnya pun, algoritma ini juga tidak terlalu berat dikarenakan kebanyakan iterasi dilakukan sebanyak suatu konstanta, sebagai contoh iterasi semua *worms* musuh

IV. Implementasi dan pengujian

Bab ini akan membahas mengenai implementasi program secara langsung pada *game engine* yang akan dijelaskan secara lisan menggunakan notasi *pseudocode* pada bagian berikut disertai dengan komentar untuk memudahkan *readability*. Akan terdapat pula penjelasan mengenai struktur data yang relevan terhadap keberjalanan program, dan analisis kualitatif terhadap algoritma/program yang telah dibuat untuk menilai keoptimalan hasil yang didapat.

4.1. Implementasi program dalam *game engine* (*pseudocode* dengan komentar)

Pemaparan implementasi program dalam *game engine* akan dilakukan menggunakan notasi *pseudocode* yang disertai komentar untuk memperjelas paparan, dan akan dilakukan secara runtut dari tingkat abstraksi atas ke rendah di file `bot.java`, sehingga akan dimulai dari penjelasan command Run kemudian penjelasan lebih lanjut tentang masing-masing komponen fungsi yang ada di dalam Run.

```

function run() → Command
{ Menjalankan public method/fungsi yang mengembalikan command langkah yang
dipilih berdasarkan algoritme yang telah didesain }

KAMUS LOKAL
    bestAttackCommand : AttackCommand { command attack terbaik }
    nextMove : Coordinate { koordinat berisi petak untuk move selanjutnya }
    nextCell : Cell { jenis petak berikutnya yang akan dipijak }

ALGORITME
{ menentukan serangan terbaik, menginit bestAttackCommand dengan instansi
kelas AttackCommand }
bestAttackCommand ← getBestAttackCommand()

{ kondisional apabila bestAttackCommand 0, maka artinya tidak ada serangan
valid yang bisa dilakukan sehingga }
if (bestAttackCommand ≠ 0) then

    { kondisional terhadap atribut command berupa string, kemudian mereturn
command serangan yang sesuai }
    if (bestAttackCommand.command = "shoot") then
        return ShootCommand(resolveDirection(currentWorm.position,
bestAttackCommand.coordinate))
    else if (bestAttackCommand.command = "bananabomb") then
        return BananaBombCommand(bestAttackCommand.coordinate.x,
bestAttackCommand.coordinate.y)
    else
        return SnowballCommand(bestAttackCommand.coordinate.x,
bestAttackCommand.coordinate.y)

else { bestAttackCommand = 0 }

    { Init variabel dengan koordinat langkah berikutnya menuju lawan dan
jenis dari petak tersebut secara GREEDY/SELALU mencoba menuju ke tempat
musuh terdekat }
    nextMove ← pursuitEnemy()
    nextCell ← getCell(nextMove.x, nextMove.y)

    { Apabila jenis petak AIR maka bisa bergerak, namun bila DIRT, harus
digali terlebih dahulu }
    if (nextCell.type = CellType.AIR) then
        return MoveCommand(nextMove.x, nextMove.y)
    else if (nextCell.type = CellType.DIRT) then
        Return DigCommand(nextMove.x, nextMove.y)

{ apabila masuk ke kondisional-else namun ternyata tidak bisa bergerak
kemana-mana, contoh dikelilingi lava }
return DoNothingCommand()

```

Gambar 8. Pseudocode Command run()

Bisa dilihat bahwa pada command run terdapat banyak method/fungsi komponen lain yang langsung digunakan, berikut adalah pseudocode dari fungsi-fungsi tersebut.

```
function getBestAttackCommand() → AttackCommand
{ Mengembalikan AttackCommand yang dipilih secara GREEDY berdasarkan command
yang bisa memberikan damage paling besar terlebih dahulu }
```

KAMUS LOKAL

```
bestShootCommand : AttackCommand
bestBananaBombCommand : AttackCommand
bestSnowBallCommand : AttackCommand
```

ALGORITME

```
{ inisiasi variabel dengan masing-masing pilihan serangan yang bisa
dilakukan }
bestShootCommand ← getBestShootCommand()
bestBananaBombCommand ← getBestBananabombCommand()
bestSnowballCommand ← getBestSnowballCommand()

{ Kondisional dengan prioritas serangan yang paling besar terlebih
dahulu dengan splash area besar, kemudian menurun hingga ke shoot }
if (bestBananaBombCommand.damage > 0) then
    return bestBananaBombCommand
else if (bestSnowBallCommand.damage > 0) then
    return bestSnowBallCommand
else
    return bestShootCommand
```

Gambar 9. Pseudocode method getBestAttackCommand()

Di dalam fungsi getBestAttackCommand terdapat kondisional yang menentukan prioritas GREEDY serangan yang ingin diambil, dari damage terbesar dan splash terbesar hingga yang terkecil, kemudian terdapat pula fungsi komponen di dalam fungsi getBestAttackCommand seperti getBestShootCommand, getBestBananabombCommand yang pada intinya mencari tempat hasil tembakan/serangan yang nantinya akan menghasilkan impact (total damage) terbesar atau menghasilkan sebuah hit TANPA adanya obstruksi (misal untuk shoot).

← →

```
function pursuitEnemy() → Coordinate
{ Mengembalikan Coordinate/petak yang bisa membuat current worm lebih dekat
dengan musuh terdekat }
```

KAMUS LOKAL

```
closestEnemyWorm : Worm
myX : integer
myY : integer
allSurroundingCoordinate : Coordinate[]
closestAfterDistance : integer
closestAddedX : integer
closestAddedY : integer
addedMyX : integer
addedMyY : integer
```

ALGORITME

```
{ pengisian variabel-variabel yang akan digunakan }
{ getClosestEnemy akan mengembalikan Worm dengan jarak euclidean paling
dekat dari currentWorm.position }
closestEnemyWorm ← getClosestEnemy()

{ mencatat pula posisi x dan y dari Worm kita yang sedang diseleksi }
myX ← currentWorm.position.x
myY ← currentWorm.position.y

{ mendapatkan array dari coordinate yang berada di sekitar worm }
allSurroundingCoordinate ← Coordinate.getAllSurroundingCoordinate()

{ inisialisasi variabel penyimpan jarak euclid terdekat, perhatikan
bahwa diinisiasi dengan nilai yang SANGAT besar dan pasti tergantikan di
iterasi pertama pencarian min }
closestAfterDistance ← 99999
closestAddedX ← 0
closestAddedY ← 0

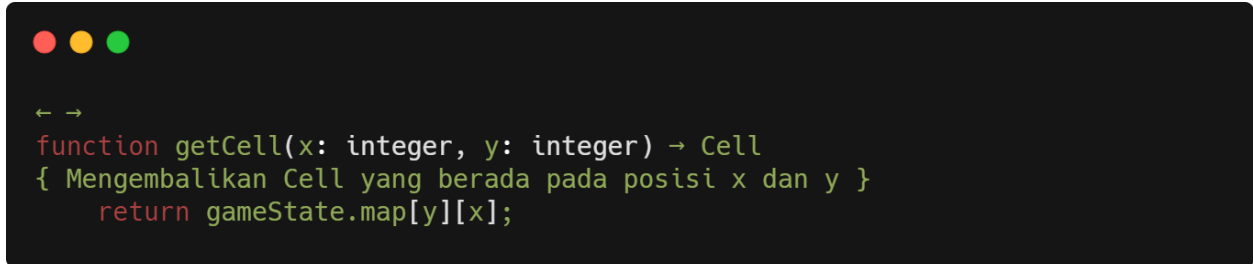
{ traversal tanpa index }
currentCoordinate traversal in (allSurroundingCoordinate)
{ addedMy_ merupakan hasil penambahan posisi worm sekarang dengan
(0,1), (1,1), (1,0) dst }
addedMyX ← myX + currentCoordinate.x
addedMyY ← myY + currentCoordinate.y

{ dilakukan pengecekan terhadap euclidean distance posisi baru
setelah ditambahkan, kemudian dicari min }
if (closestAfterDistance > euclideanDistance(addedMyX, addedMyY,
closestEnemyWorm.position.x, closestEnemyWorm.position.y)) then
    closestAfterDistance ← euclideanDistance(addedMyX, addedMyY,
closestEnemyWorm.position.x, closestEnemyWorm.position.y)
    closestAddedX ← addedMyX
    closestAddedY ← addedMyY

{ pengembalian coordinate yang membuat jarak euclid antar worm dengan
enemy terdekat semakin dekat lagi }
return Coordinate(closestAddedX, closestAddedY)
```

Gambar 10. Pseudocode method pursueEnemy()

PursuitEnemy juga memiliki fungsi-fungsi menarik lain yang terkandung di dalamnya seperti contohnya euclideanDistance yang merupakan fungsi bawaan bot.java yang melakukan kalkulasi terhadap jarak euclid menggunakan rumus yang disediakan pada aturan permainan ($\text{jarak} = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$). Terdapat pula getClosestEnemy() yang mengembalikan worm yang berjarak euclid paling kecil dari current Worm yang terseleksi.



```
← →  
function getCell(x: integer, y: integer) → Cell  
{ Mengembalikan Cell yang berada pada posisi x dan y }  
  return gameState.map[y][x];
```

Gambar 11. Pseudocode method getCell

Perhatikan bahwa getCell adalah fungsi yang mengembalikan tipe data Cell yang berisikan informasi tentang Cell tersebut pada x dan y yang dispesifikasikan pada parameter. Informasi lebih lanjut mengenai tipe data gameState serta Cell akan dipaparkan pada bagian berikut.\

4.2. Penjelasan struktur data yang digunakan dalam program

Terdapat beberapa struktur data/.java yang berisikan tentang atribut-atribut kelas yang nanti akan digunakan di run, seperti AttackCommand, Coordinate, Cell, dan Worm.

4.2.1 BananaBombWeapon

Kelas ini berperan sebagai penyimpan atribut damage, range dari weapon tersebut, jumlah banana bomb tersisa/dimiliki oleh worm serta damage radius dari banana bomb yakni radius impact dari petak-petak yang terkena bananabomb. Perhatikan semua tipe data pada masing-masing atribut BananaBombWeapon adalah integer.

4.2.2 Cell

Kelas Cell pada Cell.java akan berisikan atribut x yang menyimpan posisi pada sumbu x dari suatu cell, kemudian posisi pada sumbu y cell tersebut (kedua data x dan y bertipe integer), disertakan dengan tipe dari cell tersebut yang bertipe data CellType yakni sebuah enumeration class yang berisi jenis cell yang ada, kemudian terdapat pula atribut powerup bertipe PowerUp yang akan dijelaskan lebih lanjut pada bagian berikut.

4.2.3 Coordinate

Kelas Coordinate digunakan sebagai penyimpan penambahan/pengurangan x dan y dari petak awal. Kelas ini merupakan subclass dari object class di java yang merupakan parent class semua class, berisikan atribut integer x dan y yang menyatakan sumbu x dan y suatu koordinat, kemudian terdapat konstruktor untuk kelas Coordinate. Terdapat pula method bawaan seperti isEqualCoordinate yang memeriksa apakah koordinat masukan sama dengan koordinat instansi objek, lalu method pengecekan Is{direction} yang melakukan pengecekan apakah koordinat instansi objek mengarah ke arah mata angin yang mana, perhatikan karena petak awal diumpamakan sebagai $x, y = 0,0$ maka hanya harus mengecek apakah x dan y sama dengan, lebih besar atau lebih kecil dari 0 untuk melakukan penentuan arah mata angin.

4.2.4 GameState

Kelas GameState menyimpan berbagai data tentang game yang sedang berlangsung, seperti ronde ke berapa game sekarang, maksimum ronde yang bisa dimainkan, panjang sisi map, ID worm yang sedang bermain, kemudian atribut yang menghitung command DoNothingCommand. Perhatikan kelima atribut tersebut semua bertipe integer. Kemudian terdapat pula atribut myPlayer bertipe MyPlayer (akan dijelaskan pada bagian berikut), opponents bertipe data array of Opponent (juga akan dijelaskan pada bagian-bagian berikut) serta map yang menyimpan tampak map yang bertipe matrix of Cell.

4.2.5 MyPlayer

Kelas ini berisikan tentang informasi umum tentang player yang nanti akan digunakan untuk melakukan pengecekan mati serta display di cmd/visualizer. Kelas ini memiliki atribut id pemain, skor pemain pada saat ini, serta health dari player pada saat ini, semua bertipe integer. Kemudian terdapat pula atribut worms yang bertipe MyWorm (akan dijelaskan lebih lanjut pada bagian berikut).

4.2.6 MyWorm

Kelas ini merupakan child/sub-class dari kelas Worm (akan dijelaskan lebih lanjut pada bagian berikut), namun memiliki tambahan atribut yakni weapon yang bertipe Weapon (akan dijelaskan lebih lanjut). Kelas ini digunakan untuk menyetor data satu worm yang dimiliki oleh bot starter.

4.2.7 Opponent

Opponent merupakan kelas yang perannya sama dengan MyPlayer namun dipakai untuk penyimpanan data opponent, memiliki atribut integer yakni id pemain serta skor pemain saat itu, serta array of Worm yang dinamakan worms yang dipakai untuk menyetor data worms-worms musuh.

4.2.8 Position

Kelas ini digunakan untuk menyimpan posisi ABSOLUT yang relatif terhadap petak map keseluruhan dan bukan koordinat yang relatif dari titik acuan yang menjadi 0,0. Kelas ini berisikan atribut x dan y bertipe integer.

4.2.9 PowerUp

Kelas ini berisikan atribut type bertipe PowerUpType serta value bertipe integer. PowerUpType merupakan enumeration class yang hanya berisikan 1 kemungkinan yakni HEALTH_PACK, kemudian value digunakan untuk merepresentasikan banyaknya health points yang diheal oleh healthpack.

4.2.10 Position

Kelas ini digunakan untuk menyimpan posisi ABSOLUT yang relatif terhadap petak map keseluruhan dan bukan koordinat yang relatif dari titik acuan yang menjadi 0,0. Kelas ini berisikan atribut x dan y bertipe integer.

4.2.11 SnowballWeapon

Kelas ini berisikan data umum tentang durasi freeze bertipe integer, range dari weapon snowball bertipe integer, kemudian atribut integer yang menjadi counter berapa banyak snowball yang tersisa, kemudian terdapat pula atribut integer yang menyatakan radius freeze/berapa radius petak yang terkena efek freeze snowball.

4.2.12 Weapon

Kelas Weapon ini berisikan atribut damage bertipe integer yakni damage weapon kemudian atribut integer yang menyatakan range.

4.2.13 Worm

Kelas Worm berisikan tentang data umum satu worm, berisikan ID bertipe integer, health points yang dimiliki worm bertipe integer, kemudian position bertipe Position yang merupakan posisi worm saat ini, lalu diggingRange yang bertipe integer yang menyatakan radius/jangkauan

dari worm untuk melakukan command dig, lalu movementRange yang sama-sama menyatakan jangkauan untuk gerakan worm.

Terdapat pula profession yang bertipe string, seperti Commando, Technologist dan lainnya, yang menyatakan profesi worm, lalu bananaBombWeapon bertipe BananaBombWeapon yang berisikan data umum banana bomb untuk worm tersebut, kemudian SnowballWeapon yang memiliki fungsi sama. Terdapat pula method canBananaBomb dan canSnowball yang mengembalikan boolean dan mengecek apakah count snowball atau bananabomb cukup untuk melakukan serangan.

4.3. Analisis desain solusi algoritma *greedy*

Secara garis besar, algoritma *greedy* yang digunakan dalam tugas ini mampu memenangkan 16 dari 40 permainan yang dicoba. Algoritma *greedy* yang digunakan untuk mengatur posisi Worm saat menembak menggunakan *euclidean distance* yang mengukur jarak terpendek antara Worm musuh dan Worm pemain. Hal ini menyebabkan adanya ketidakefisienan dalam pengaturan posisi Worm saat ingin menembak musuh.

Berdasarkan pengujian yang dilakukan secara praktis, dapat dilihat pula bahwa, angle/sudut yang didekatkan oleh worm memiliki alur yang tidak biasa/linier dikarenakan perhitungan kedekatan menggunakan jarak euclidean tersebut.

Salah satu percobaan yang berakhir dalam kemenangan menunjukkan bahwa alur dari permainan tetap mirip dengan permainan yang berakhir dalam kekalahan, biasa diawali dengan 2 worm starter mendekati 1 worm enemy yang terdekat dengan kedua worm, kemudian memusnahkan worm tersebut, lalu baru mendekati 2 worm enemy lain yang poin healthnya sudah dikikis oleh 1 worm starter lain.

Namun kekalahan atau kemenangan biasa lebih bergantung dengan reposisi/penyesuaian ulang posisi worm musuh, jika dilakukan dalam cara tertentu, bisa menyebabkan worm starter untuk melakukan reposisi juga walau sudah lock-on target, dan mulai menembak, kemudian reposisi ini juga bisa mengurangi serangan yang dapat diberikan oleh worm starter karena reposisi tersebut dapat dilakukan secara tidak efisien disebabkan oleh perhitungan dengan jarak euclid.

Adapula kasus dimana, karena bot worm selalu menjadi pengejar yang melakukan pursuit terhadap enemy terdekat, biasa worm starter akan menggali dan membuka jalan yang akan ditemui oleh tembakan musuh, sehingga, walau bot starter menjadi pengejar yang konstan, bisa jadi bot starter yang akan terkena tembakan terlebih dahulu.

Sehingga kesimpulannya adalah kekalahan dari algoritme greedy bot biasa disebabkan oleh reposisi yang tidak efisien dan mengorbankan langkah/bisa ditembak lebih banyak dari menembak, kemudian bisa juga karena menjadi pengejar yang bergerak kemudian terkena tembakan pertama kali dari musuh yang sedang diam. Sedangkan kemenangan bisa disimpulkan dikarenakan oleh sifat ketiga bot yang serupa dan menargetkan musuh terdekat, dan biasa dimulai dengan 2 bot worm starter mendekati worm musuh yang biasa musnah kemudian 2 bot worm starter dapat pindah ke target-target berikutnya dan akhirnya menang.

V. Kesimpulan dan saran

Dalam bab ini, kesimpulan dan saran dari tugas secara keseluruhan akan dijelaskan.

5.1. Kesimpulan

Secara keseluruhan, bot yang diprogram dapat memenangkan 16 dari 40 permainan yang diuji. Meskipun persentase kemenangan bernilai 40%, bot telah menunjukkan kemampuan untuk menang melawan bot referensi. Selain itu, implementasi algoritma *greedy* dalam bot yang digunakan juga sudah berjalan. Hal ini juga membuktikan bahwa algoritma *greedy* terkadang tidak memiliki tingkat efektivitas yang tinggi tetapi memiliki tingkat efisiensi yang baik.

5.2. Saran

Secara umum, tugas sudah berjalan dengan baik. Walau begitu, beberapa hal untuk meningkatkan performa yang sudah ada masih dapat dilakukan. Untuk kedepannya, algoritma untuk menentukan posisi shooting Worm dapat lebih diperbaiki agar tidak selalu mengambil jarak terpendek antara Worm musuh dan Worm pemain. Selain itu, penggunaan komentar yang lebih baik mendeskripsikan setiap proses dapat ditulis oleh pemrogram. Mungkin juga, untuk kedepannya, bahasa pemrograman yang digunakan dapat bebas dipilih mengingat banyaknya variasi bahasa untuk bot pada program ini. Terakhir, untuk grammar dari kalimat “It’s not worth if you’re not have fun” dapat diperbaiki. Secara umum, tim ini sudah mengeluarkan usaha paling maksimal untuk keberjalanan program dan tugas dikerjakan dengan baik. Atas perhatian pembaca dalam membaca laporan ini, penulis mengucapkan terima kasih.

Daftar Pustaka

Cormen, T., Leiserson, C., Rivest, R. and Stein, C., 2009. *Introduction to algorithms*. 3rd ed. Massachusetts: Massachusetts Institute of Technology, p.414.

Munir, R. 2021. *Algoritma Greedy (2021) Bag 1*. Slide kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung

Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). *When the greedy algorithm fails*. Discrete optimization, 1(2), 121-127.

DeVore, R. A., & Temlyakov, V. N. (1996). *Some remarks on greedy algorithms*. Advances in computational Mathematics, 5(1), 173-187.