

**LAPORAN  
TUGAS BESAR 2 IF2211  
STRATEGI ALGORITMA**

**Pemanfaatan Algoritma *Greedy* pada Aplikasi Permainan  
“Galaxio”**



**oleh**

|                                       |                 |
|---------------------------------------|-----------------|
| <b>Go Dillon Audris</b>               | <b>13521062</b> |
| <b>Salomo Reinhart Gregory Manalu</b> | <b>13521063</b> |
| <b>Nathan Tenka</b>                   | <b>13521172</b> |

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2022/2023**

## DAFTAR ISI

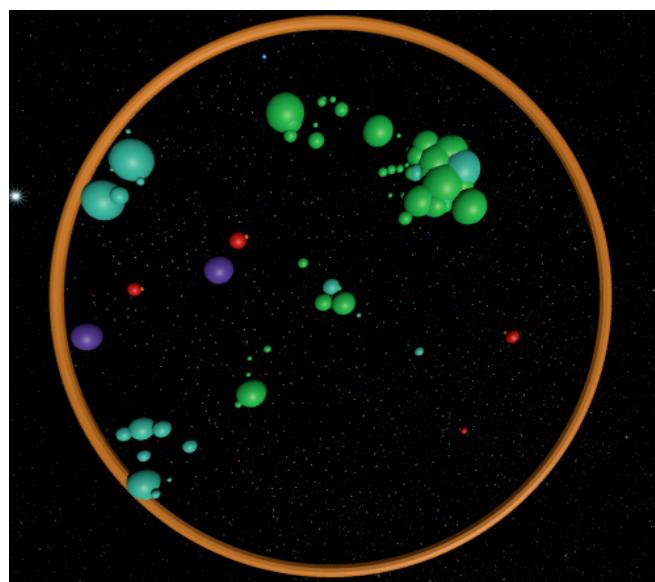
|   |           |
|---|-----------|
| <b>DAFTAR ISI</b>   | <b>2</b>  |
| <b>BAB I</b>  | <b>1</b>  |
| <b>DESKRIPSI TUGAS</b>  | <b>1</b>  |
| <b>BAB II</b>   | <b>6</b>  |
| <b>LANDASAN TEORI</b>   | <b>6</b>  |
| <b>2.1 Algoritma Greedy secara umum</b>                                 | <b>6</b>  |
| <b>2.2 Program Permainan Galaxio</b>                                    | <b>8</b>  |
| 2.2.1 Struktur Program Permainan Galaxio                                | 8         |
| 2.2.2 Cara Kerja Engine atau Mesin Permainan Galaxio                    | 9         |
| 2.2.3 Objek dan Terminologi Pada Permainan Galaxio                      | 11        |
| <b>BAB III</b>  | <b>17</b> |
| <b>APLIKASI STRATEGI GREEDY</b>   | <b>17</b> |
| <b>3.1 Pemetaan Persoalan Galaxio ke Elemen-Elemen Algoritma Greedy</b> | <b>17</b> |
| 3.1.1. Pemetaan Persoalan Galaxio Secara Umum                           | 17        |
| 3.1.2. Pemetaan Persoalan Food dan Super Food                           | 18        |
| 3.1.3. Pemetaan Persoalan Menyerang Pemain Lain:                        | 19        |
| 3.1.4 Pemetaan Persoalan Menghindari Bahaya                             | 20        |
| <b>3.2 Eksplorasi Alternatif Solusi Algoritma Greedy</b>                | <b>22</b> |
| 3.2.1 Greedy By Size  | 22        |
| 3.2.2 Greedy By Score   | 22        |
| 3.2.3 Greedy By Distance  | 23        |
| 3.2.4 Greedy By Damage  | 23        |
| <b>3.3 Analisis Efisiensi dan Efektivitas Alternatif</b>                | <b>24</b> |
| <b>Solusi Algoritma Greedy</b>  | <b>24</b> |
| 3.3.1. Analisis Efisiensi Alternatif Solusi Algoritma Greedy            | 24        |
| 3.3.2. Analisis Efektivitas Alternatif Solusi Algoritma Greedy          | 24        |
| 3.3.2.1. Greedy By Size   | 24        |
| 3.3.2.2. Greedy By Score  | 24        |
| 3.3.2.3. Greedy By Distance   | 25        |
| 3.3.2.4. Greedy By Damage   | 25        |
| <b>3.4 Strategi Algoritma Greedy Terpilih</b>                           | <b>26</b> |
| <b>BAB IV</b>   | <b>27</b> |

|   |           |
|---|-----------|
| <b>IMPLEMENTASI DAN PENGUJIAN</b>   | <b>27</b> |
| <b>4.1 Implementasi Solusi Algoritma Greedy pada Bot Galaxio</b>  | <b>27</b> |
| 4.1.1 Implementasi Algoritma Greedy Utama   | 27        |
| 4.1.2 Implementasi Method moveToCenter  | 28        |
| 4.1.3 Implementasi Method determineFood dan Method Turunannya   | 29        |
| 4.1.4 Implementasi Method determineTorpedo dan Method Turunannya  | 31        |
| 4.1.5 Implementasi Method determineAvoid dan Method Turunannya  | 32        |
| 4.1.6 Implementasi Method determineTeleport dan Method Turunannya   | 35        |
| <b>4.2 Penjelasan Struktur Data</b>   | <b>37</b> |
| <b>4.3 Analisis Desain Solusi Algoritma Greedy</b>  | <b>49</b> |
| 4.3.1 Masalah pengambilan food atau superfood dengan jarak sama   | 50        |
| 4.3.2 Hasil pengujian penembakan torpedo  | 51        |
| 4.3.3 Hasil pengujian penembakan teleporter ke lawan yang lebih kecil   | 52        |
| 4.3.4 Hasil pengujian menghindari teleporter lawan  | 54        |
| 4.3.5 Hasil pengujian aktivasi shield saat torpedo lawan dekat<br>dan menuju bot  | 56        |
| 4.3.6 Masalah navigasi bot di dalam gas clouds  | 57        |
| 4.3.7 Hasil pengujian ketika bot di dekat gas clouds  | 58        |
| 4.3.8 Hasil pengujian ketika ada makanan yang dekat dengan<br>ujung world   | 60        |
| 4.3.9 Hasil pengujian bot di dekat lawan yang lebih besar   | 62        |
| 4.3.10 Hasil pengujian bot sudah menembakkan teleporter<br>namun lawan yang dituju menjadi terlalu besar untuk dimakan. | 63        |
| <b>BAB V</b>  | <b>66</b> |
| <b>KESIMPULAN DAN SARAN</b>   | <b>66</b> |
| <b>5.1 Kesimpulan</b>   | <b>66</b> |
| <b>5.2 Saran</b>  | <b>66</b> |
| <b>DAFTAR PUSTAKA</b>   | <b>67</b> |
| <b>LAMPIRAN</b>   | <b>68</b> |
| <b>Link Repository Program</b>  | <b>69</b> |
| <b>Link Video Demo Youtube</b>  | <b>69</b> |

## BAB I

### DESKRIPSI TUGAS

*Galaxio* adalah sebuah *game battle royale* yang mempertandingkan bot kapal anda dengan beberapa bot kapal yang lain. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal anda yang tetap hidup hingga akhir permainan. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.



Gambar 1.1 Ilustrasi permainan *Galaxio*

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Galaxio*. *Game engine* dapat diperoleh pada laman berikut:

<https://github.com/EntelectChallenge/2021-Galaxio>

Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini:

<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

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

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu *Players*, *Food*, *Wormholes*, *Gas Clouds*, *Asteroid Fields*. Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan x. Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. *Heading* dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek *afterburner* akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.

3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal *player*. Apabila *player* mengkonsumsi *Food*, maka *Player* akan bertambah ukuran yang sama dengan *Food*. *Food* memiliki peluang untuk berubah menjadi *Super Food*. Apabila *Super Food* dikonsumsi maka setiap makan *Food*, efeknya akan 2 kali dari *Food* yang dikonsumsi. Efek dari *Super Food* bertahan selama 5 tick.
4. *Wormhole* ada secara berpasangan dan memperbolehkan kapal dari *player* untuk memasukinya dan keluar di pasangan satu lagi. *Wormhole* akan bertambah besar setiap tick game hingga ukuran maximum. Ketika *Wormhole* dilewati, maka *wormhole* akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat *wormhole* lebih besar dari kapal *player*.
5. *Gas Clouds* akan tersebar pada peta. Kapal dapat melewati *gas cloud*. Setiap kapal bertabrakan dengan *gas cloud*, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan *gas cloud*, maka efek pengurangan akan hilang.
6. *Torpedo Salvo* akan muncul pada peta yang berasal dari kapal lain. *Torpedo Salvo* berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. *Torpedo Salvo* dapat mengurangi ukuran kapal yang ditabraknya. *Torpedo Salvo* akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. *Supernova* merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan

dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi *gas cloud*.

8. Player dapat meluncurkan *teleporter* pada suatu arah di peta. *Teleporter* tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat *teleporter* tersebut. Harga setiap peluncuran *teleporter* adalah 20. Setiap 100 tick *player* akan mendapatkan 1 *teleporter* dengan jumlah maximum adalah 10.
9. Ketika kapal *player* bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh *player*. Setiap tick, *player* hanya dapat memberikan satu *command*. Berikut jenis-jenis dari command yang ada dalam permainan:
  - a. FORWARD
  - b. STOP
  - c. START\_AFTERRBURNER
  - d. STOP\_AFTERRBURNER
  - e. FIRE\_TORPEDOES
  - f. FIRE\_SUPERNOVA
  - g. DETONATE\_SUPERNOVA

- h. FIRE\_TELEPORTER
  - i. TELEPORT
  - j. USE\_SHIELD
11. Setiap *player* akan memiliki *score* yang hanya dapat dilihat jika permainan berakhir. *Score* ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal player lain, maka *score* bertambah 10, jika mengonsumsi *food* atau melewati *wormhole*, maka *score* bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan score tertinggi.

Adapun peraturan yang lebih lengkap dari permainan *Galaxio*, dapat dilihat pada laman :

<https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md>

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 Algoritma *Greedy* secara umum**

Algoritma *Greedy* merupakan salah satu strategi atau metode yang banyak digunakan untuk menyelesaikan persoalan optimasi (memaksimumkan atau meminimumkan sesuatu). Prinsip dari strategi *Greedy* adalah berusaha untuk mendapatkan atau mengambil pilihan yang terbaik di setiap langkah dengan harapan bahwa runtutan pilihan yang diambil memang memberikan hasil yang terbaik untuk menyelesaikan persoalan.

Walaupun strategi *Greedy* berusaha untuk mengambil pilihan yang terbaik di setiap langkah, bukan berarti runtutan pilihan yang diambil akan memberikan hasil yang paling optimal. Masalah ini dapat terjadi karena algoritma *Greedy* tidak melihat permasalahan secara menyeluruh atau melakukan operasi terhadap semua kemungkinan runtutan pilihan. Selain itu, algoritma *Greedy* juga tidak dapat melihat ke keadaan sebelumnya dan melakukan *backtracking*.

Algoritma *Greedy* memiliki beberapa komponen atau elemen yang menyusunnya, yaitu:

1. Himpunan kandidat, C:

Berisi kumpulan kandidat pilihan yang akan dipilih pada setiap langkah strategi *Greedy*.

2. Himpunan solusi, S:

Berisi kumpulan pilihan yang telah dipilih oleh strategi *Greedy*. Himpunan solusi akan membentuk runtutan pilihan yang harus dilakukan untuk menyelesaikan permasalahan optimasi.

3. Fungsi solusi:

Fungsi ini akan menentukan apakah himpunan solusi S telah memberikan solusi kepada permasalahan optimasi yang ingin diselesaikan.

4. Fungsi seleksi:

Fungsi ini akan menentukan pilihan apa yang akan diambil oleh strategi *Greedy* berdasarkan keadaan tertentu. Strategi *Greedy* yang diterapkan bersifat heuristik.

5. Fungsi kelayakan:

Fungsi ini akan memeriksa apakah kandidat yang dipilih oleh fungsi seleksi layak untuk dimasukkan ke dalam himpunan solusi atau tidak, sesuai dengan batasan permasalahan yang ingin diselesaikan.

6. Fungsi obyektif:

Fungsi obyektif bertugas untuk memaksimumkan atau meminimumkan hal-hal tertentu yang diminta oleh permasalahan.

Meskipun algoritma *Greedy* belum tentu memberikan hasil yang paling optimal, algoritma ini dapat digunakan untuk mendapatkan solusi hampiran yang mendekati optimal. Hal ini tentu akan lebih menguntungkan komputasi daripada mencari solusi optimal atau eksak yang membutuhkan kompleksitas waktu yang besar.

## 2.2 Program Permainan *Galaxio*

Permainan *Galaxio* merupakan suatu permainan berjenis *battle-royale* antara beberapa *bot* atau robot kapal pemain. Objektif dari permainan ini adalah untuk menjadi satu-satunya bot yang hidup di akhir permainan dengan poin yang tertinggi dengan melakukan berbagai aksi yang disediakan.

### 2.2.1 Struktur Program Permainan *Galaxio*

Dari *starter-pack* yang diberikan oleh pengembang permainan *Galaxio* pada tautan [berikut](#), terdapat beberapa folder penting :

1. *runner-publish*:

Berisi file dan hal-hal yang diperlukan untuk menjalankan permainan.

2. *engine-publish*:

Berisi file *engine* atau mesin permainan yang bertugas untuk mengeksekusi aksi setiap *bot* dan memberikan *state* atau status permainan kepada setiap pemain.

3. *logger-publish*:

Berisi file dan hal-hal yang diperlukan untuk mencatat status permainan di setiap waktu.

4. *reference-bot-publish*:

Berisi file referensi *bot* yang dapat digunakan oleh para pemain untuk menciptakan *bot*-nya sendiri.

5. *starter-bots*:

Berisi file-file *bot* dalam berbagai bahasa pemrograman yang dapat digunakan pemain untuk mulai menciptakan *bot*.

## 6. visualizer:

Berisi file-file untuk mengubah *log* atau catatan permainan menjadi permainan yang dapat dilihat secara visual oleh penonton ataupun pemain.

### 2.2.2 Cara Kerja *Engine* atau Mesin Permainan *Galaxio*

Sebelum permainan dapat dimulai, perlu dilakukan inisialisasi *runner*, *engine*, dan *logger* terlebih dahulu. Hal ini dapat dilakukan dengan menjalankan *command* yang terdapat pada *snippet* di bawah ini. Pastikan bahwa perangkat lunak .NET Framework versi 3.1 ke atas telah terpasang pada komputer.

```
::: Game Runner
cd ./runner-publish/
start "" dotnet GameRunner.dll

::: Game Engine
cd ../engine-publish/
timeout /t 3
start "" dotnet Engine.dll

::: Game Logger
cd ../logger-publish/
timeout /t 3
start "" dotnet Logger.dll
```

Dengan berjalannya ketiga hal tersebut, permainan *Galaxio* memiliki lingkungan atau sumber daya yang cukup untuk dapat dijalankan. Setelahnya, *engine* akan menunggu hingga semua *bot* telah bergabung ke dalam permainan. Bot dapat bergabung dengan program melalui contoh *command* berikut.

```
::: Bots
cd ../reference-bot-publish/
start "" java -jar
../starter-bots/Tubes1_agario-supremacy/target/JavaBot.jar
```

Waktu dalam permainan *Galaxio* dihitung dengan menggunakan satuan *tick*, dan pada setiap *tick*-nya, *engine* akan mengeksekusi aksi yang telah dikirim oleh setiap *bot*. Meskipun aksi dilakukan dan dieksekusi dalam setiap *tick*, namun setiap *bot* tidak diharuskan untuk menentukan aksinya tepat ketika *tick* berubah. Hal ini terlihat pada *snippet* kode program berikut ini, di mana *bot* menentukan aksinya setiap 20 milidetik.

```
while (hubConnection.getConnectionState() ==
    HubConnectionState.CONNECTED) {
    Thread.sleep(20);

    GameObject bot = botService.getBot();
    if (bot == null) {
        continue;
    }

    botService.getPlayerAction().setPlayerId(bot.getId());
    botService.computeNextPlayerAction(
        botService.getPlayerAction());
    if (hubConnection.getConnectionState() ==
        HubConnectionState.CONNECTED) {
        hubConnection.send("SendPlayerAction",
            botService.getPlayerAction());
    }
}
```

Hal ini menunjukkan bahwa setelah *bot* menentukan aksi yang akan dilakukannya, aksi tersebut tidak langsung dieksekusi. Aksi akan memasuki suatu *queue* sebelum dieksekusi oleh *engine* permainan ketika *tick berubah*.

Dapat dilihat juga pada *snippet*, bahwa terdapat suatu metode bernama *computeNextPlayerAction* yang dimiliki oleh objek *botService*. Pada metode inilah *bot* menentukan aksinya berdasarkan keadaan permainan saat ini yang

dikirimkan oleh *engine* (objek GameState). Setiap pemain akan berusaha untuk menciptakan *bot* terbaik yang dapat memenangkan permainan berdasarkan algoritma yang mereka pakai. Pada Tugas Besar 1 IF2211 Strategi Algoritma, penulis menggunakan algoritma *Greedy*.

### 2.2.3 Objek dan Terminologi Pada Permainan *Galaxio*

Terdapat berbagai objek, terminologi dan juga aksi yang dapat dilakukan oleh *bot* pada permainan *Galaxio*. Hal-hal ini dapat membantu perancang *bot* untuk lebih memahami permainan dan menciptakan *bot* yang lebih baik. Dokumentasi mengenai objek dan terminologi juga dapat dibaca pada tautan [berikut](#).

#### 1. Dunia atau *World*:

Merupakan tempat atau *arena* permainan *Galaxio*. Di dalam *world* terdapat seluruh objek permainan seperti *bot*, makanan, dan lain-lain. *World* berpusat pada titik 0,0 pada bidang kartesius dan memiliki radius atau ukuran tertentu pada awal permainan. Radius *world* akan terus mengecil seiring permainan berjalan dan *bot* yang keluar dari *world* akan terus mengecil sebesar 1 setiap *tick*.

#### 2. Objek Game:

##### a. Kapal:

Merupakan sebutan untuk pemain atau *bot* dalam permainan. Pada awal permainan, kapal akan diletakkan secara acak oleh *engine*, memiliki ukuran 10 dan kecepatan sebesar 20. Selain memiliki ukuran dan kecepatan, kapal juga memiliki *heading* atau arah, yang dihitung mulai dari sumbu-X positif pada bidang kartesius dengan nilai antara 0 sampai

359 derajat. Kapal dapat bertambah besar dengan memakan objek *food* atau *bot* lain. Seiring bertambahnya ukuran kapal, maka kecepatan kapal akan terus berkurang. Kapal dengan ukuran lebih kecil dari 5 atau yang dimakan kapal lain akan dikeluarkan dari permainan.

b. Makanan atau *food*:

Merupakan objek yang dimunculkan secara acak pada awal permainan dan dapat dimakan oleh *bot* untuk memperbesar ukurannya. *Food* tidak dapat bergerak dan akan meningkatkan ukuran *bot* sebesar 3 ketika dikonsumsi.

c. Makanan super atau *super food*:

Setiap *food* yang muncul pada permainan memiliki kesempatan untuk muncul sebagai *super food*. Ketika *super food* dikonsumsi, maka *bot* akan mendapat efek tambahan. Setiap *food* yang dikonsumsi oleh *bot* akan dilipatgandakan efeknya sebesar faktor 2 untuk 5 *tick* ke depan. Artinya, *food* yang biasanya meningkatkan ukuran *bot* sebesar 3 akan memberikan ukuran 6 kepada *bot* yang telah memakan *super food* sebelumnya. Efek *super food* dapat ditimbun selama 5 *tick* belum berakhir.

d. *Wormholes*:

Objek *wormholes* muncul secara berpasangan dan memungkinkan *bot* untuk pindah ke sisi lain *world* selama ukuran *wormholes* lebih besar dari ukuran *bot*. Setelah *bot* berpindah, maka *wormholes* akan mengecil sebesar setengah ukuran *bot*.

e. *Gas clouds*:

Merupakan objek yang bisa muncul secara berkelompok atau sendiri. *Bot* dapat melewati *gas clouds*, namun ukuran *bot* akan terus berkurang sebesar 1 di setiap *tick*.

f. *Asteroid fields*:

Merupakan objek yang bisa muncul secara berkelompok atau sendiri. *Bot* dapat melewati *asteroid fields* seperti *gas clouds*, namun kecepatan bot akan dikurangi dengan faktor 2.

g. *Torpedo*:

Setiap kapal akan diberikan 1 *torpedo salvo* setiap 10 *tick*, dan dapat menampung hingga 5 *torpedo salvo* pada setiap *tick*-nya. Menembakkan *torpedo salvo* pada arah yang ditentukan akan mengurangi ukuran kapal sebesar 5 dan menyebabkan kapal untuk mengeluarkan *torpedo*. Setiap *torpedo* terbang dengan arah lurus, pada kecepatan 60 dan ukuran awal 10. *Torpedo* dapat bertabrakan dengan objek berupa *food*, *bot*, ataupun *torpedo* lainnya. Tabrakan menyebabkan ukuran *torpedo* berkurang. Jika *torpedo* bertabrakan dengan *bot* lain, maka ukuran *torpedo* dan *bot* yang ditabrak akan berkurang dan ukuran *bot* yang menembakkan *torpedo* akan bertambah.

h. *Supernova*:

Merupakan senjata sekali pakai yang hanya muncul satu kali di dalam permainan di tengah-tengah *world*. Supernova kemudian dapat diambil untuk kemudian ditembakkan oleh *bot* ketika diperlukan. Ketika ditembakkan, supernova akan terbang dalam garis lurus dan tidak dapat

menabrak objek apapun. Supernova dapat diledakkan oleh *bot* untuk menghasilkan serangan dalam radius yang besar dan menciptakan *gas clouds* pada radius tersebut.

i. *Teleporter*:

Setiap kapal akan memiliki 1 *teleporter* di awal permainan, dan akan diberikan 1 lagi setiap 100 *tick*. Setiap kapal maksimum dapat menyimpan 10 *teleporter*. Meluncurkan *teleporter* akan mengurangi ukuran *bot* sebesar 20. *Teleporter* kemudian akan terbang pada garis lurus dengan kecepatan 20 tanpa bertabrakan dengan objek apapun. *Bot* kemudian dapat melakukan *teleport* untuk pindah ke posisi *teleporter*.

j. *Shield*:

*Bot* dapat mengaktifkan *shield* ketika diperlukan. Pengaktifan *shield* akan mengurangi ukuran sebesar 20 dan memerlukan waktu sebesar 20 *tick*. *Shield* akan aktif selama 20 detik dan memantulkan semua *torpedo* yang bertabrakan dengan lawan.

3. Aksi *bot* atau pemain:

Setiap aksi yang dikirimkan oleh *bot* juga akan memiliki atribut *heading* atau arah

a. *FORWARD*:

Perintah ini akan menyebabkan *bot* untuk bergerak pada arah yang diberikan.

b. *STOP*:

Perintah ini akan menghentikan pergerakan *bot* hingga perintah *FORWARD* diberikan lagi,

c. *START\_AFTERBURNER*:

Perintah ini akan menyebabkan *bot* mendapat efek *afterburner*. Efek ini menyebabkan kecepatan kapal dikali 2 dan ukuran kapal berkurang 1 setiap *tick*. Efek *afterburner* hanya akan muncul ketika kapal bergerak.

d. *STOP\_AFTERBURNER*:

Perintah ini akan menghilangkan efek *afterburner* dari kapal.

e. *FIRE\_TORPEDOES*:

Perintah ini akan mengurangi 1 *torpedo salvo* dan 5 ukuran *bot* untuk menembakkan *torpedo* pada arah yang diberikan.

f. *FIRE\_SUPERNOVA*:

Perintah ini akan menembakkan 1 *supernova* ke arah yang diberikan.

g. *DETONATE\_SUPERNOVA*:

Perintah ini akan meledakkan *supernova* yang ditembak oleh pemain.

h. *FIRE\_TELEPORTER*:

Perintah ini akan mengurangi 1 *teleporter* dan ukuran *bot* sebesar 20 untuk menembakkan *teleporter* ke arah yang diberikan.

i. *TELEPORT*:

Perintah ini akan menyebabkan kapal melakukan *teleport* ke *teleporter* yang ditembakkan.

j. *ACTIVATE\_SHIELD*:

Perintah ini akan mengurangi ukuran *bot* sebesar 20 dan mengaktifkan *shield* yang akan memantulkan semua torpedo yang datang selama 20 *tick* ke depan.

## **BAB III**

### **APLIKASI STRATEGI *GREEDY***

#### **3.1 Pemetaan Persoalan *Galaxio* ke Elemen-Elemen Algoritma *Greedy***

##### **3.1.1. Pemetaan Persoalan *Galaxio* Secara Umum**

Permainan *Galaxio* bisa dibagi menjadi elemen-elemen algoritma *Greedy* berikut :

| <b>Nama Elemen/Komponen</b> | <b>Definisi Elemen/Komponen</b>  |
|-----------------------------|--|
| Himpunan Kandidat           | Aksi-aksi dengan <i>heading</i> -nya yang bisa dilakukan bot (sudah dituliskan pada bab sebelumnya). |
| Himpunan Solusi             | Permutasi aksi yang membuat <i>bot</i> bertahan hidup lebih lama dibanding <i>bot</i> lain.          |
| Fungsi Solusi               | Memeriksa apakah permutasi aksi memaksimalkan masa hidup <i>bot</i> .                                |
| Fungsi Seleksi              | Pilih aksi berdasarkan <i>game state</i> dan prioritas aksi dari sub permasalahan.                   |
| Fungsi Kelayakan            | Aksi adalah aksi yang valid untuk dilakukan  |
| Fungsi Obyektif             | Mencari permutasi aksi yang membuat <i>bot</i> bertahan hidup paling lama dibanding <i>bot</i> lain. |

### 3.1.2. Pemetaan Persoalan *Food* dan *Super Food*

*Food* dan *super food* merupakan salah satu komponen utama dari permainan *Galaxio* karena *food* merupakan cara utama dan termudah untuk menambah *size* dan *score*, terutama di awal permainan. Oleh karena itu, penanganan cara mendapat *food* atau *super food* adalah salah satu komponen terpenting yang perlu ditangani.

| Nama Elemen/Komponen | Definisi Elemen/Komponen  |
|----------------------|---|
| Himpunan Kandidat    | Permutasi aksi dan <i>heading</i> yang membuat <i>bot</i> bisa berpindah menuju <i>food</i> atau <i>super food</i> . Dalam hal ini aksi yang masuk akal hanyalah FORWARD. |
| Himpunan Solusi      | Permutasi aksi dengan <i>heading</i> yang membuat <i>bot</i> mendapat <i>food</i> atau <i>super food</i> secara efisien.  |
| Fungsi Solusi        | Pilih aksi berdasarkan <i>gamestate</i> dan prioritas aksi dengan strategi pengambilan <i>food</i> atau <i>super food</i> optimum.  |
| Fungsi Seleksi       | Pilih aksi berdasarkan <i>gamestate</i> dan prioritas aksi dengan strategi pengambilan <i>food</i> atau <i>super food</i> optimum.  |
| Fungsi Kelayakan     | Aksi adalah aksi yang valid (FORWARD).  |

|                 |   |
|-----------------|---|
| Fungsi Obyektif | Mencari permutasi aksi yang membuat <i>bot</i> mendapat <i>food</i> dan <i>super food</i> sebanyak-banyaknya. |
|-----------------|---|

### 3.1.3. Pemetaan Persoalan Menyerang Pemain Lain:

Dalam permainan *Galaxio*, *bot* bisa menyerang *bot* lain dengan beberapa cara. Aksi yang bisa dilakukan untuk menyerang adalah menembak *torpedo*, *teleport*, *supernova*, maupun sekedar berjalan menuju *bot* lain yang lebih kecil.

| Nama Elemen/Komponen | Definisi Elemen/Komponen   |
|----------------------|--|
| Himpunan Kandidat    | Permutasi aksi dengan <i>heading</i> -nya yang membuat <i>bot</i> menyerang pemain lain dengan hasil optimum berdasarkan parameter yang dipilih. Dalam hal ini aksi yang mungkin adalah FORWARD, START_AFTERCLOUD, STOP_AFTERCLOUD, FIRE_TELEPORT, TELEPORT, FIRE_TORPEDO, FIRE_SUPERNOVA, dan DETONATE_SUPERNOVA. |
| Himpunan Solusi      | Permutasi aksi dengan <i>heading</i> -nya yang membuat <i>bot</i> menyerang pemain lain dengan hasil optimum berdasarkan parameter yang dipilih.   |
| Fungsi Solusi        | Memeriksa apakah permutasi aksi membuat <i>bot</i> menyerang pemain lain dengan tepat (tidak meleset atau bahkan berbalik menyebabkan kerugian).   |

|                  |  |
|------------------|--|
|                  | Fungsi seleksi : pilih aksi berdasarkan <i>gamestate</i> dan prioritas aksi dengan strategi penyerangan optimum.   |
| Fungsi Seleksi   | Pilih aksi berdasarkan <i>gamestate</i> dan prioritas aksi dengan strategi penyerangan optimum.  |
| Fungsi Kelayakan | Aksi adalah aksi yang valid (FORWARD, START_AFTERBURNER, STOP_AFTERBURNER, FIRE_TELEPORT, TELEPORT, FIRE_TORPEDO, FIRE_SUPERNOVA, dan DETONATE_SUPERNOVA). |
| Fungsi Obyektif  | Mencari permutasi aksi yang membuat <i>bot</i> mengeliminasi sebanyak mungkin pemain lain.   |

### 3.1.4 Pemetaan Persoalan Menghindari Bahaya

Dalam permainan *Galaxio* juga terdapat objek-objek yang berbahaya bagi *bot*. Objek-objek tersebut adalah *gas cloud*, *asteroid field*, *torpedo* dan *supernova* pemain lain, *bot* yang lebih besar, dan batas *world*. Objek-objek tersebut bisa mengurangi *size*, menghambat pergerakan *bot*, dan bahkan mengeliminasi *bot*. Bahaya-bahaya tersebut bisa dihindari dengan mengubah arah gerak, *teleport*, *shield*, maupun menembakkan *torpedo*.

| Nama Elemen/Komponen | Definisi Elemen/Komponen |
|----------------------|--------------------------|
|                      |                          |

|                   |   |
|-------------------|---|
| Himpunan Kandidat | Permutasi aksi beserta <i>heading</i> -nya yang membuat <i>bot</i> menghindari bahaya. Dalam hal ini aksi yang mungkin adalah FORWARD, START_AFTERTURNER, STOP_AFTERTURNER, FIRE_TELEPORT, TELEPORT, FIRE_TORPEDO, dan ACTIVATE_SHIELD. |
| Himpunan Solusi   | Permutasi aksi dengan <i>heading</i> -nya yang membuat <i>bot</i> menghindari bahaya dengan kerugian minimum.   |
| Fungsi Solusi     | Memeriksa apakah permutasi aksi membuat <i>bot</i> menjauhi bahaya dengan tepat (tidak menuju objek berbahaya).   |
| Fungsi Seleksi    | Pilih aksi berdasarkan <i>gamestate</i> dan prioritas aksi dengan strategi penghindaran optimum.  |
| Fungsi Kelayakan  | Aksi adalah aksi yang valid (FORWARD, START_AFTERTURNER, STOP_AFTERTURNER, FIRE_TELEPORT, TELEPORT, FIRE_TORPEDO, dan ACTIVATE_SHIELD).   |
| Fungsi Obyektif   | Mencari permutasi aksi yang membuat <i>bot</i> menghindari sebanyak mungkin bahaya.   |

### **3.2 Eksplorasi Alternatif Solusi Algoritma *Greedy***

Secara umum, terdapat berbagai alternatif algoritma *Greedy* yang bisa diterapkan pada persoalan *Galaxio*.

#### *3.2.1 Greedy By Size*

Strategi ini memilih permutasi aksi yang memaksimalkan ukuran *bot*. *Size* merupakan sumber daya utama pada permainan *Galaxio* yang digunakan sebagai penentu *bot* hidup atau tidak, untuk menembakkan *torpedo* dan *teleporter*, untuk menyalakan *afterburner*, dan untuk menentukan apakah *bot* memakan atau dimakan saat terjadi tabrakan dengan *bot* lain. Jadi, dengan memaksimalkan *size*, diharapkan *bot* memiliki lebih banyak pilihan aksi untuk bertahan hidup. Hal-hal yang menjadi prioritas dalam strategi ini adalah :

- Mendapat *food* dan *super food* sebanyak mungkin.
- Mengambil *size* dari pemain lain (misal melalui *torpedo* atau memakan pemain yang lebih kecil).
- Menghindari objek-objek yang mengurangi *size* (*gas cloud*, batas *world*, *torpedo* lawan, lawan yang lebih besar, *supernova*).

#### *3.2.2 Greedy By Score*

Strategi ini memilih permutasi aksi yang memaksimalkan *score* yang didapat. *Score* merupakan penentu pemenang saat terjadi *draw*. Diharapkan dengan memaksimalkan *score*, *bot* bisa meraih kemenangan. Hal-hal yang menjadi prioritas dalam strategi ini adalah :

- Mendapat *food* dan *super food* sebanyak mungkin.

- Memakan sebanyak mungkin pemain lain.
- Masuk ke *wormhole* sebanyak mungkin.

### 3.2.3 Greedy By Distance

Strategi ini memilih permutasi aksi berdasarkan target terdekat. Dengan strategi ini diharapkan permutasi aksi yang didapat akan membuat *bot* bertahan paling lama. Aksi yang mungkin di antaranya :

- Jika paling dekat dengan *food*, maka makan *food* tersebut.
- Jika paling dekat dengan pemain yang lebih besar, menjauh atau tembak *torpedo*.
- Jika paling dekat dengan pemain yang lebih kecil, mendekat untuk memakan pemain tersebut.
- Jika paling dekat dengan *torpedo* lawan, nyalakan *shield* untuk menghentikan *torpedo*.

### 3.2.4 Greedy By Damage

Strategi ini memaksimalkan *damage* yang dilakukan pada *bot* lain. Dengan strategi agresif ini diharapkan semua *bot* lawan dieliminasi oleh *bot* milik pemain. Hal yang menjadi prioritas dalam strategi ini adalah :

- Tembakkan *torpedo* sebanyak mungkin ke pemain lain.
- Makan pemain lain sebanyak mungkin.

### **3.3 Analisis Efisiensi dan Efektivitas Alternatif Solusi Algoritma *Greedy***

#### **3.3.1. Analisis Efisiensi Alternatif Solusi Algoritma *Greedy***

Semua informasi objek dalam permainan *Galaxio* diberikan dalam *gamestate*.

Seluruh alternatif strategi di atas perlu memeriksa keadaan seluruh objek yang menjadi perhatian. Oleh karena itu, semua algoritma di atas memiliki kompleksitas  $O(n)$  dengan  $n$  adalah jumlah objek.

#### **3.3.2. Analisis Efektivitas Alternatif Solusi Algoritma *Greedy***

##### **3.3.2.1. *Greedy By Size***

Seperti yang telah disebutkan sebelumnya, *size* merupakan sumber daya utama dalam permainan *Galaxio*. Dengan *size* yang besar, *bot* akan memiliki fleksibilitas lebih dalam memilih aksi dan bisa mendominasi area yang lebih luas. Namun, *bot* mungkin kalah saat tidak mampu mengakumulasi *size* sebanyak-banyaknya dan terjadi *draw* karena kekurangan *score*. Walau dengan kekurangan tersebut, menurut penulis strategi ini adalah yang paling efektif karena fleksibilitas yang didapat dengan mengakumulasi *size*.

##### **3.3.2.2. *Greedy By Score***

Strategi ini sebenarnya memiliki kemiripan dengan *Greedy by size* karena aksi yang menambah *size* (seperti memakan *food* dan memakan lawan) juga menambah *score*. Akan tetapi, karena *size* bukan prioritas utama, objek yang mengurangi *size* bot tidak terlalu diperhatikan. *Bot* juga tidak bisa mengakumulasi *size* sebanyak mungkin karena perlu melewati *wormhole* yang hanya dapat

dilewati *bot* yang lebih kecil dari ukurannya. *Wormhole* sendiri bukan mekanisme yang bisa diandalkan karena tidak diketahui *wormhole* mana yang saling terhubung dan lokasinya acak pada setiap permainan. Bisa saja *wormhole* yang dilewati malah mendekatkan *bot* dengan objek yang berbahaya. Oleh karena itu, menurut penulis strategi ini kurang efektif.

#### 3.3.2.3. *Greedy By Distance*

Strategi ini mungkin merupakan strategi yang paling sesuai dengan definisi algoritma *Greedy* karena benar-benar melihat selangkah di depan. Namun, karena hal tersebut, strategi ini menjadi kurang fleksibel. Kelemahan utama dalam strategi ini menurut penulis disebabkan banyaknya *food* yang ada, sehingga bisa saja *bot* hanya akan mendeteksi *food* sebagai objek terdekat sepanjang permainan. sedangkan objek berbahaya (seperti *torpedo* dan pemain yang lebih besar) sudah terlalu dekat untuk dihindari saat menjadi objek terdekat. Oleh karena itu, menurut penulis strategi ini juga kurang optimal.

#### 3.3.2.4. *Greedy By Damage*

Strategi ini mengutamakan menyerang lawan sebanyak mungkin. Strategi ini memang memiliki kemungkinan untuk mengeliminasi lawan dengan paling cepat, namun *bot* kurang memperhatikan pertahanannya. *Bot* juga menjadi kurang fleksibel karena hanya memprioritaskan menyerang lawan. Selain itu, *size bot* juga relatif kurang besar karena terus dipakai untuk menembakkan *torpedo*. Penulis menilai strategi ini kurang efektif dikarenakan risiko-risiko tersebut.

### **3.4 Strategi Algoritma *Greedy* Terpilih**

Dengan berbagai alternatif solusi yang ada, penulis memilih untuk mengimplementasikan algoritma *greedy by size* karena dengan *size* yang besar, *bot* memiliki lebih banyak opsi yang bisa dipakai dan bisa dengan lebih mudah memenangkan permainan. Aksi dipilih dengan prioritas sebagai berikut :

1. Tembak *teleport* ke lawan yang lebih kecil jika ukuran paling besar atau lakukan *teleport* jika *teleporter* dekat dengan musuh lebih kecil atau *teleporter* sudah cukup jauh dari ancaman.
2. Hindari objek berbahaya dengan prioritas berikut :
  - a. *Torpedo* lawan (dengan mengaktifkan *shield*)
  - b. Musuh yang lebih besar (dengan mengubah *heading* atau menembakkan *teleporter*)
  - c. *Teleporter* lawan (dengan mengubah *heading* atau menembakkan *teleporter*)
  - d. Gas cloud (dengan mengubah *heading*)
3. Serang lawan memakai *torpedo* jika *size* dari bot cukup.
4. Makan food terdekat yang tidak terlalu dekat dengan batas *world*.
5. Bergerak menuju ke tengah *world*.

Aksi dikirim tiap *tick* sesuai dengan *game state* pada *tick* tersebut supaya *bot* selalu melakukan aksi terbaik berdasarkan algoritma.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Solusi Algoritma *Greedy* pada *Bot Galaxio*

Implementasi dari solusi dari algoritma *Greedy* yang dipilih ditulis dalam bentuk *pseudocode*. Implementasi hanya ditulis pada metode-metode yang merupakan bagian penting dalam algoritma *Greedy* dan tidak mencakup metode-metode yang berisi perhitungan teknis. Hal ini untuk menjaga *readability* dari *pseudocode* agar pembaca dapat memahami algoritma dengan baik.

##### 4.1.1 Implementasi Algoritma *Greedy* Utama

Pada subbab ini, terdapat implementasi *method*:

- *computeNextPlayerAction*, sebagai method yang diinvokasi untuk mendapatkan aksi *bot*.
- *bestAction*, sebagai method utama untuk menentukan aksi *bot*.
- *setFoodSector*, sebagai *setup* sebelum menjalankan atau memanggil algoritma *bestAction*.

```
procedure computeNextPlayerAction(input/output : PlayerAction d)
{Method diinvokasi untuk mendapatkan aksi yang akan dilakukan bot}

ALGORITMA PROSEDUR

if (not init and terdapatPlayerGameObjects) then
    setFoodSector(gameState, bot, localState)
    init <- true

if (terdapatGameObjects and terdapatPlayerGameObjects) then
    d <- bestAction(gameState, bot, localState)
```

```

function bestAction(GameState a, GameObject b, LocalState c) -> PlayerAction
{Memilih aksi yang terbaik untuk dilakukan berdasarkan gameState dan localState
saat ini}

KAMUS
playerAction <- new PlayerAction()

ALGORITMA FUNGSI
{Menentukan aksi yang diambil bot, diurut berdasarkan prioritas terendah hingga
tertinggi}
playerAction <- moveToCenter(playerAction, b)
playerAction <- determineFood(a, playerAction, b, c)
playerAction <- determineTorpedo(a, playerAction, b)
playerAction <- determineAvoid(a, playerAction, b, c)
playerAction <- determineTeleport(a, playerAction, b, c)

-> playerAction

```

```

procedure setFoodSector(input: GameState a, GameObject b, output: LocalState c)
{Melakukan setup sektor makanan dari bot, sektor didefinisikan sebagai area
yang berada di dekat bot ketika permainan dimulai. Sektor ditandai dengan
heading tertinggi dan terendah dari sektor. Informasi disimpan di LocalState}

KAMUS
headingBotKePusat <- hitungHeadingKePusat(b)
setengahBesarSektor <- (360 / 2*banyakPlayer)

ALGORITMA PROSEDUR
c.lowerHeadingBase <- headingBotKePusat - setengahBesarSektor
c.higherHeadingBase <- headingBotKePusat + setengahBesarSektor

```

#### 4.1.2 Implementasi Method *moveToCenter*

Pada subbab ini, terdapat implementasi *method moveToCenter*.

```

function moveToCenter(PlayerAction d, GameObject b) -> PlayerAction
{Bot memilih aksi FORWARD dengan random heading antara b dan pusat World. Aksi ini dieksekusi jika sudah tidak ada food atau superfood pada permainan. Bot akan berusaha untuk tetap berada di dalam permainan dan menghindari World Border}

ALGORITMA FUNGSI
d.action <- FORWARD
d.heading <- pilihRandomHeadingPadaRentang(headingBotKePusat ± 45°)

-> d

```

#### 4.1.3 Implementasi *Method determineFood* dan *Method* Turunannya

Pada subbab ini, terdapat implementasi *method*:

- *determineFood*, sebagai *method* untuk menentukan apakah *bot* akan memakan makanan di dalam sektor atau makanan di World.
- *getNearestFoodInSector*, sebagai *method* untuk menentukan makanan di dalam sektor yang akan dimakan oleh *bot*.
- *getNearestFood*, sebagai *method* untuk menentukan makanan di World yang akan dimakan oleh *bot*.

```

function determineFood(GameState a, PlayerAction d, GameObject b,
                      LocalState c) -> PlayerAction
{Menentukan apakah bot akan memakan food di dalam sektor atau di World.
 Dengan pertimbangan berupa posisi bot dan keberadaan food di dalam sektor}

ALGORITMA FUNGSI
if (terdapatFoodAtauSuperFoodDiDalamGame) then
    if (botBeradaDiDalamSektor and terdapatFoodAtauSuperFoodDiDalamSektor) then
        d <- getNearestFoodInSector(a, b, c)
    else
        d <- getNearestFood(a, b, c)

```

```
-> d
```

```
function getNearestFoodInSector(GameState a, GameObject b, LocalState c)
    -> PlayerAction
{Bot memilih aksi FORWARD dengan heading ke food atau superfood terdekat di
sektor}
```

**KAMUS**

```
foodList <- ambilListMakananDiSektor
d <- new PlayerAction()
```

**ALGORITMA FUNGSI**

```
d.action <- FORWARD
d.heading <- hitungHeading(foodList[0], b)
```

```
-> d
```

```
function ambilMakananTerdekat(GameState a, GameObject b, LocalState c)
    -> PlayerAction
{Bot memilih aksi FORWARD dengan heading ke food atau superfood terdekat di
World}
```

**KAMUS**

```
foodList <- ambilListMakanan
d <- new PlayerAction()
```

**ALGORITMA FUNGSI**

```
d.action <- FORWARD
d.heading <- hitungHeading(foodList[0], b)
```

```
-> d
```

#### 4.1.4 Implementasi Method *determineTorpedo* dan Method Turunannya

Pada subbab ini, terdapat implementasi *method*:

- *determineTorpedo*, sebagai *method* untuk menentukan kelayakan penembakan torpedo.
- *stealSizeWithTorpedo*, sebagai *method* yang membentuk aksi penembakan torpedo.

```
function determineTorpedo(GameState a, PlayerAction d, GameObject b)
    -> PlayerAction

{Method untuk menentukan apakah ada kapal dalam radius tertentu yang dapat
ditembak dengan torpedo untuk mengambil atau mencuri ukuran kapal tersebut}

ALGORITMA FUNGSI

if (adaKapalDalamRadiusSekitarBot and jumlahTorpedo > 0 and
    ukuranBot > 18) then
    d <- stealSizeWithTorpedo(a, b)

-> d
```

```
function stealSizeWithTorpedo(GameState a, GameObject b) -> PlayerAction

{Bot memilih aksi FIRETORPEDOES dengan heading ke kapal terdekat yang berada di
radius sekitar bot}

KAMUS

enemiesList <- ambilListEnemiesDiRadiusSekitarBot
d <- new PlayerAction()

ALGORITMA FUNGSI

d.action <- FIRETORPEDOES
d.heading <- hitungHeading(enemiesList[0], b)

-> d
```

#### 4.1.5 Implementasi Method *determineAvoid* dan Method Turunannya

Pada subbab ini, terdapat implementasi *method*:

- *determineAvoid*, sebagai *method* untuk menentukan aksi yang akan diambil jika terdapat ancaman yang datang kepada *bot*.
- *avoidNearestGasCloud*, sebagai *method* untuk menentukan aksi ketika *bot* berdekatan dengan *gas clouds*.
- *avoidTeleport*, sebagai *method* untuk menentukan aksi ketika terdapat teleporter yang datang ke arah *bot*.
- *avoidLargerEnemy*, sebagai *method* untuk menentukan aksi ketika terdapat musuh yang besar di sekitar *bot*.
- *avoidTorpedo*, sebagai *method* untuk menentukan aksi ketika terdapat *torpedo* yang datang ke arah *bot*.

```
function determineAvoid(GameState a, PlayerAction d, GameObject b,
                      LocalState c) -> PlayerAction
{Method menentukan aksi yang diambil jika terdapat ancaman di sekitar bot}

ALGORITMA FUNGSI
d <- avoidNearestGasCloud(a, d, b, c)
d <- avoidTeleport(a, d, b, c)
d <- avoidLargerEnemy(a, d, b, c)
d <- avoidTorpedo(a, d, b, c)

-> d
```

```

function avoidNearestGasCloud(GameState a, PlayerAction d, GameObject b,
                                LocalState c) -> PlayerAction

{Bot menentukan aksi untuk menghindari gas cloud}

KAMUS

gasList <- ambilListGasClouds

ALGORITMA FUNGSI

if (terdapatGasCloudsDalamGame) then
    if (jarakGasCloudsTerdekatDenganBotCukupDekat) then
        if (BotMengarahKeGasClouds and tidakAdaTeleporterDitembakkan) then
            d.action <- FORWARD
            d.heading <- hitungHeading(gasList[0], b) + 45

-> d

```

```

function avoidTeleport(GameState a, PlayerAction d, GameObject b, LocalState c)
                        -> PlayerAction

{Bot menentukan aksi untuk menghindari teleporter yang mengarah ke bot}

KAMUS

tpList <- ambilListTeleporterYangMengarahKeBot

ALGORITMA FUNGSI

if (terdapatTeleporterYangMengarahKeBot) then
    if (tidakAdaTeleporterYangDitembakkanOlehBot and botBukanBotTerbesar) then
        d.heading <- hitungHeading(tpList[0], b) + 90
        if (ukuranBot > 35 and jumlahTeleporter > 0) then
            d.action <- FIRETELEPORT
            updateLocalStateBerkaitanDenganTeleport
        else
            d.action <- FORWARD

-> d

```

```

function avoidLargerEnemy(GameState a, PlayerAction d, GameObject b,
                         LocalState c) -> PlayerAction

{Bot menentukan aksi untuk menghindari bot yang lebih besar yang mengarah ke
bot}

KAMUS

enemyList <- ambilListKapalYangLebihBesar

ALGORITMA FUNGSI

if (terdapatKapalYangLebihBesarDiSekitarBot) then
    if (jumlahTorpedo = 0 and tidakAdaTeleporterYangDitembakkanOlehBot) then
        d.heading = hitungHeading(enemyList[0], b) + 180
        if (jumlahTeleporter > 0 and ukuranBot > 35) then
            d.action <- FIRETELEPORT
            updateLocalStateBerkaitanDenganTeleport
        else
            d.action <- FORWARD
    else
        -> d

```

```

function avoidTorpedo(GameState a, PlayerAction d, GameObject b)
                      -> PlayerAction

{Bot menentukan aksi untuk menghindari torpedo yang datang}

KAMUS

torpedoList <- ambilListTorpedo

ALGORITMA FUNGSI

if (terdapatTorpedoDalamPermainan) then
    if (jarakTorpedoDenganBotCukupDekat and torpedoMengarahKeBot) then
        if (jumlahShield > 0 and ukuranBot > 35 and shieldBelumDiaktifkan) then
            d.action <- ACTIVATESHIELD
    else
        -> d

```

#### 4.1.6 Implementasi Method *determineTeleport* dan Method Turunannya

Pada subbab ini, terdapat implementasi *method*:

- *determineTeleport*, sebagai *method* yang menentukan apakah bot akan menembak *teleporter* atau melakukan *teleport*.
- *fireTeleporterToEnemies*, sebagai *method* yang menentukan *heading* penembakan *teleporter*.
- *teleportToTeleporter*, sebagai *method* untuk melakukan aksi *teleport* ke *teleporter*.

```
function determineTeleport(GameState a, PlayerAction d, GameObject b,
                           LocalState c) -> PlayerAction
{Method menentukan aksi yang diambil jika bot bisa menembak teleporter atau
 bisa melakukan teleport}

ALGORITMA FUNGSI

if (localState.teleporterFired) then
    if (teleporterAdaDiWorld) then
        localState.teleporterStillNotAppear <- false
    else if (not localState.teleporterStillNotAppear) then
        localState.teleporterFired <- false
        localState.teleporterStillNotAppear <- true

    if (not localState.teleporterStillNotAppear and
        ((terdapatMusuhKecilDiSekitarTeleporter and localState.tpReason = 1) or
         (tidakAdaMusuhBesarDiSekitarTeleporter and teleporterCukupJauh and
          localState.tpReason = 2))) then
        d <- teleportToTeleporter(a, b, c)

    else if (botYangPalingBesar and adaMusuhYangCukupKecil and
              not localState.teleporterFired and jumlahTeleporter > 0 and
              ukuranBot > 45 and not d.action = ACTIVATESHIELD) then
        d <- fireTeleporterToEnemies(a, b, c)

-> d
```

```

function fireTeleporterToEnemies(GameState a, GameObject b, LocalState c)
    -> PlayerAction

{Bot melakukan aksi FIRETELEPORT dengan heading ke musuh target}

KAMUS

d <- new PlayerAction()

target <- ambilBotLainYangCukupKecilUntukDimakan
theta <- hitungSelisihHeadingKeTargetDenganTrigonometri

ALGORITMA FUNGSI

d.action <- FIRETELEPORT

d.heading <- hitungHeading(target, bot) ± theta
updateLocalStateBerkaitanDenganTeleport

-> d

```

```

function teleportToTeleporter(GameState a, GameObject b, LocalState c)
    -> PlayerAction

{Bot melakukan aksi TELEPORT ke teleporter}

KAMUS

d <- new PlayerAction()

teleporter <- temukanTeleporterYangDitembakkan
enemiesList <- ambilListEnemiesYangJauhDariTeleporter

ALGORITMA FUNGSI

d.action <- TELEPORT

if (terdapatMusuhYangJauhDariTeleporter) then
    d.heading <- hitungHeading(enemiesList[0], b)
else
    d.heading <- pilihRandomHeadingPadaRentang(0 sampai 360)
updateLocalStateBerkaitanDenganTeleport

-> d

```

## 4.2 Penjelasan Struktur Data

Struktur data pada permainan *Galaxio* ini berbentuk *class*. *Class-class* tersebut terdapat di dalam 4 package. 4 package yang ada dalam permainan *Galaxio* yaitu Enums, Greedy, Models, dan Services. Enums berisi enumerasi dari *objectTypes* dan *playerActions* yang bersifat konstan. Greedy berisi perintah-perintah yang dapat dilakukan oleh *bot*. Models berisi hal-hal (objek) yang ada dalam game, seperti *world* dan *torpedo*. Services berisi logika *bot* dan merupakan tempat pembuatan *bot*.

Berikut ini adalah penjelasan mengenai *class* yang ada pada *bot* game *Galaxio* ini :

### A. Enums

#### a. ObjectTypes.java

| Methods  | Description  |
|--|--|
| final Integer value                              | Nilai dari <i>ObjectTypes</i>                                      |
| ObjectTypes(Integer value)                       | Method yang akan mengembalikan nilai dari suatu <i>ObjectTypes</i> |
| public static ObjectTypes valueOf(Integer value) | Method yang akan mengembalikan <i>ObjectTypes</i> dari suatu value |

#### b. PlayerActions.java

| Methods | Description |
|---------|-------------|
|         |             |

|                                      |  |
|--------------------------------------|--|
| public final Integer value           | Nilai dari <i>PlayerActions</i>                                      |
| private PlayerActions(Integer value) | Method yang akan mengembalikan nilai dari suatu <i>PlayerActions</i> |

## B. Greedy

### a. Avoid.java

| Methods  | Description  |
|--|--|
| <i>static public PlayerAction determineAvoid(GameState gameState, PlayerAction playerAction)</i>   | Method yang akan menentukan aksi yang dipilih untuk menghindari bahaya |
| <i>static private PlayerAction avoidNearestGasCloud(GameState gameState, PlayerAction playerAction, GameObject bot, LocalState localState)</i> | Method untuk menghindari <i>Gas Cloud</i>                              |
| <i>static private PlayerAction avoidLargerEnemy(GameState gameState, PlayerAction playerAction, GameObject bot, LocalState localState)</i>     | Method untuk menghindari musuh yang memiliki ukuran lebih besar        |
| <i>static private PlayerAction avoidTeleport(GameState gameState, PlayerAction playerAction, GameObject bot, LocalState localState)</i>        | Method untuk menghindari <i>teleport</i>                               |
| <i>static private PlayerAction avoidTorpedo(GameState gameState, PlayerAction playerAction, GameObject bot)</i>                                | Method untuk menghindari <i>torpedo</i>                                |

### b. Food.java

| Methods  | Description   |
|--|---|
| <i>static public PlayerAction determineFood(GameState gameState, PlayerAction playerAction, GameObject</i> | Method yang akan menentukan makanan yang akan dimakan |

|  |  |
|--|--|
| <i>bot, LocalState localState)</i>   |  |
| <i>static public void setFoodSector(GameState gameState, GameObject bot, LocalState localState)</i>                      | Method yang akan menentukan batas-batas sektor untuk mencari dan memakan <i>food</i> |
| <i>static private List&lt;GameObject&gt; getFoodInSector(GameState gameState, GameObject bot, LocalState localState)</i> | Method untuk mencari dan memakan <i>food</i> dalam sektor                            |
| <i>static private List&lt;GameObject&gt; getFoodInGame(GameState gameState, GameObject bot)</i>                          | Method yang akan mengembalikan semua <i>food</i> yang ada dalam game                 |
| <i>static private PlayerAction getNearestFoodinSector(GameState gameState, GameObject bot, LocalState localState)</i>    | Method untuk mendapatkan makanan terdekat dalam sektor                               |
| <i>static private PlayerAction getNearestFood(GameState gameState, GameObject bot, LocalState localState)</i>            | Method untuk mendapatkan makanan terdekat  |
| <i>static private boolean botInSector(GameState gameState, GameObject bot, LocalState localState)</i>                    | Method untuk mendeteksi apakah ada bot lain atau tidak dalam sektor                  |

c. Greedy.java

| Methods  | Description                                   |
|--|---|
| <i>static public PlayerAction bestAction(GameState gameState, GameObject bot, LocalState localState)</i> | Method untuk menentukan pilihan aksi terbaik  |
| <i>static private PlayerAction moveToCenter(PlayerAction playerAction, GameObject bot)</i>               | Method untuk berpindah ke tengah <i>world</i> |

d. Helper.java

| Methods | Description |
|---------|-------------|
|         |             |

|   |   |
|---|---|
| <code>static public double<br/>getDistanceBetween(GameObject<br/>object1, GameObject object2)</code>                      | Method untuk mendapatkan jarak antara suatu <i>GameObject</i> dengan dengan <i>GameObject</i> lainnya                   |
| <code>static public double<br/>getDistanceFromCenter (GameObject<br/>item)</code>   | Method untuk mendapatkan jarak <i>GameObject</i> ke tengah <i>world</i>   |
| <code>static public int<br/>getHeadingBetween(GameObject other,<br/>GameObject bot)</code>                                | Method untuk menentukan arah antara <i>bot</i> dengan <i>GameObject</i> lainnya   |
| <code>static public int toDegrees(double v)</code>  | Method untuk mengubah satuan radian menjadi derajat   |
| <code>static public int<br/>getHeadingFromCenter(GameObject<br/>other)</code>   | Method untuk menentukan arah antara <i>GameObject</i> dengan pusat atau tengah <i>world</i>                             |
| <code>static public List&lt;GameObject&gt;<br/>gameStateToBigShipsNear (GameState<br/>gameState, GameObject bot)</code>   | Method untuk mendapatkan <i>player</i> dengan <i>size</i> yang lebih besar dan berada dekat dengan <i>bot</i>           |
| <code>static public boolean<br/>thereIsBiggerShipsNear(GameState<br/>gameState, GameObject bot)</code>                    | Method untuk mendeteksi apakah ada <i>player</i> dengan <i>size</i> yang lebih besar dan berada dekat dengan <i>bot</i> |
| <code>static public List&lt;GameObject&gt;<br/>getListOfSmallEnemies(GameState<br/>gameState, GameObject bot)</code>      | Method untuk mendapatkan <i>player</i> dengan <i>size</i> yang lebih kecil  |
| <code>static public boolean<br/>isBotTheBiggest(GameState gameState,<br/>GameObject bot)</code>                           | Method untuk mengecek apakah <i>bot</i> merupakan <i>player</i> terbesar dalam game                                     |
| <code>static public void<br/>printBotState(GameState gameState,<br/>GameObject bot, PlayerAction<br/>playerAction)</code> | Method untuk mencetak informasi dari setiap <i>tick</i> di terminal   |

e. Teleport.java

| Methods                                 | Description                 |
|---|-----------------------------|
| <code>static public PlayerAction</code> | Method yang akan menentukan |

|  |  |
|--|--|
| <i>determineTeleport(GameState gameState, PlayerAction playerAction, GameObject bot, LocalState localState)</i>                  | apakah <i>bot</i> sebaiknya melakukan <i>teleport</i> atau tidak   |
| <i>static private GameObject getTargetedEnemies(GameState gameState, GameObject bot)</i>   | Method yang akan mengembalikan <i>player</i> dengan ukuran terkecil                                      |
| <i>static private boolean thereIsSmallerEnemies(GameState gameState, GameObject bot)</i>   | Method untuk mengecek apakah ada musuh yang lebih kecil daripada <i>bot</i>                              |
| <i>static private PlayerAction fireTeleporterToEnemies(GameState gameState, GameObject bot, LocalState localState)</i>           | Method untuk menembak <i>teleporter</i> ke musuh   |
| <i>static private List&lt;GameObject&gt; findTeleporter(GameState gameState, LocalState localState)</i>                          | Method untuk mendapatkan <i>teleporter</i> yang ada dalam <i>world</i>                                   |
| <i>static private boolean teleporterStillInWorld(GameState gameState, LocalState localState)</i>                                 | Method untuk mengecek apakah <i>teleporter</i> masih ada dalam <i>world</i>                              |
| <i>static private boolean thereIsSmallerEnemiesAroundTeleporter (GameState gameState, GameObject bot, LocalState localState)</i> | Method untuk mengecek apakah ada musuh yang lebih kecil di sekitar <i>teleporter</i>                     |
| <i>static private boolean thereIsNoLargerEnemiesAroundTeleporter(GameState gameState, GameObject bot, LocalState localState)</i> | Method untuk mengecek apakah ada musuh yang lebih besar daripada <i>bot</i> di sekitar <i>teleporter</i> |
| <i>static private boolean tpFarEnough(GameState gameState, GameObject bot, LocalState localState)</i>                            | Method untuk mengecek apakah <i>teleporter</i> sudah cukup jauh  |
| <i>static private PlayerAction teleportToTeleporter(GameState gameState, GameObject bot, LocalState localState)</i>              | Method untuk melakukan <i>teleport</i> ke <i>teleporter</i>  |

f. Torpedo.java

| <b>Methods</b>   | <b>Description</b>   |
|--|--|
| <i>static public PlayerAction determineTorpedo(GameState gameState, PlayerAction playerAction, GameObject bot)</i> | Method untuk menentukan apakah <i>bot</i> menembakkan <i>torpedo</i> atau tidak                                |
| <i>static private List&lt;GameObject&gt; getListOfNearShips(GameState gameState, GameObject bot)</i>               | Method untuk mendapatkan <i>player</i> yang ada di sekitar <i>bot</i>  |
| <i>static public boolean bigShipInRadius(GameState gameState, GameObject bot)</i>                                  | Method untuk mengecek apakah ada <i>player</i> lain yang lebih besar daripada <i>bot</i> di sekitar <i>bot</i> |
| <i>static public PlayerAction stealSizeWithTorpedo(GameState gameState, GameObject bot)</i>                        | Method untuk menembakkan <i>torpedo</i>  |

## C. Models

### a. GameObject.java

| <b>Attributes</b>                        | <b>Description</b>                                       |
|--|--|
| <i>public UUID id</i>                    | Nilai ID <i>GameObject</i>                               |
| <i>public Integer size</i>               | Ukuran dari <i>GameObject</i>                            |
| <i>public Integer speed</i>              | Kecepatan dari <i>GameObject</i>                         |
| <i>public Integer currentHeading</i>     | Arah dari <i>GameObject</i>                              |
| <i>public Position position</i>          | Posisi dari <i>GameObject</i>                            |
| <i>public ObjectTypes gameObjectType</i> | Tipe Objek dari <i>GameObject</i>                        |
| <i>public Integer effects</i>            | Menandai efek yang sedang bekerja pada <i>GameObject</i> |
| <i>public Integer torpedoSalvoCount</i>  | Jumlah dari <i>torpedo</i> yang ada                      |
| <i>public Integer superNovaAvailable</i> | Jumlah dari <i>supernova</i> yang ada                    |

|   |  |
|---|--|
| <code>public Integer teleporterCount</code> | Jumlah dari <i>teleporter</i> yang ada |
| <code>public Integer shieldCount</code>     | Jumlah dari <i>shield</i> yang ada     |

| Methods   | Description  |
|---|--|
| <code>public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, Integer effects, Integer torpedoSalvoCount, Integer superNovaAvailable, Integer teleporterCount, Integer shieldCount)</code> | Constructor untuk <i>GameObject</i>                                    |
| <code>public UUID getId()</code>  | Method yang akan mengembalikan ID dari <i>GameObject</i>               |
| <code>public void setId(UUID id)</code>   | Method untuk menetapkan ID suatu <i>GameObject</i>                     |
| <code>public int getSize()</code>   | Method yang akan mengembalikan ukuran dari <i>GameObject</i>           |
| <code>public void setSize(int size)</code>  | Method untuk menetapkan ukuran suatu <i>GameObject</i>                 |
| <code>public int getSpeed()</code>  | Method yang akan mengembalikan kecepatan dari <i>GameObject</i>        |
| <code>public void setSpeed(int speed)</code>  | Method untuk menetapkan kecepatan suatu <i>GameObject</i>              |
| <code>public Position getPosition()</code>  | Method yang akan mengembalikan posisi dari <i>GameObject</i>           |
| <code>public void setPosition(Position position)</code>   | Method untuk menetapkan posisi suatu <i>GameObject</i>                 |
| <code>public ObjectTypes getGameObjectType()</code>   | Method yang akan mengembalikan tipe objek dari suatu <i>GameObject</i> |
| <code>public void setGameObjectType(ObjectTypes gameObjectType)</code>  | Method untuk menetapkan tipe objek suatu <i>GameObject</i>             |
| <code>public static GameObject</code>   | Mengembalikan <i>GameObject</i> dari                                   |

|  |      |
|--|------|
| <i>FromStateList(UUID id, List&lt;Integer&gt; stateList)</i> | list |
|--|------|

b. GameState.java

| Attributes   | Description                              |
|--|--|
| <i>public World</i>                                    | Dunia dalam game                         |
| <i>public List&lt;GameObject&gt; gameObjects</i>       | List yang berisi <i>GameObject</i>       |
| <i>public List&lt;GameObject&gt; playerGameObjects</i> | List yang berisi <i>playerGameObject</i> |

| Methods  | Description   |
|--|---|
| <i>public GameState()</i>  | Constructor untuk <i>GameState</i>                          |
| <i>public GameState(World world, List&lt;GameObject&gt; gameObjects, List&lt;GameObject&gt; playerGameObjects)</i> | Constructor dengan parameter untuk <i>GameState</i>         |
| <i>public World getWorld()</i>   | Method yang akan mengembalikan dunia / <i>world</i>         |
| <i>public void setWorld(World world)</i>   | Method untuk menetapkan dunia / <i>world</i>                |
| <i>public List&lt;GameObject&gt; getGameObjects()</i>  | Method yang akan mengembalikan list <i>GameObject</i>       |
| <i>public void setGameObjects(List&lt;GameObject&gt; gameObjects)</i>  | Method untuk menetapkan list <i>GameObject</i>              |
| <i>public List&lt;GameObject&gt; getPlayerGameObjects()</i>  | Method yang akan mengembalikan list <i>playerGameObject</i> |
| <i>public void setPlayerGameObjects(List&lt;GameObject&gt; playerGameObjects)</i>                                  | Method untuk menetapkan list <i>playerGameObject</i>        |

c. GameStateDto.java

| Attributes  | Description                              |
|---|--|
| <i>private World world</i>  | Dunia dalam game                         |
| <i>private Map&lt;String, List&lt;Integer&gt;&gt; gameObjects</i>   | List yang berisi <i>GameObject</i>       |
| <i>private Map&lt;String, List&lt;Integer&gt;&gt; playerObjects</i> | List yang berisi <i>playerGameObject</i> |

| Methods   | Description   |
|---|---|
| <i>public Models.World getWorld()</i>   | Constructor untuk <i>GameState</i>                          |
| <i>public void setWorld(Models.World world)</i>   | Constructor dengan parameter untuk <i>GameState</i>         |
| <i>public Map&lt;String, List&lt;Integer&gt;&gt; getGameObjects()</i>                     | Method yang akan mengembalikan list <i>GameObject</i>       |
| <i>public void setGameObjects(Map&lt;String, List&lt;Integer&gt;&gt; gameObjects)</i>     | Method untuk menetapkan dunia / <i>world</i>                |
| <i>public Map&lt;String, List&lt;Integer&gt;&gt; getPlayerObjects()</i>                   | Method yang akan mengembalikan list <i>playerGameObject</i> |
| <i>public void setPlayerObjects(Map&lt;String, List&lt;Integer&gt;&gt; playerObjects)</i> | Method untuk menetapkan list <i>playerGameObject</i>        |

d. LocalState.java

| Attributes                          | Description   |
|-------------------------------------|---|
| <i>public int lowerHeadingBase;</i> | Batas bawah sektor  |
| <i>public int higherHeadingBase</i> | Batas atas sektor   |
| <i>public boolean specialSector</i> | Sektor dengan batas bawah yang lebih tinggi dari batas atas |

|  |   |
|--|---|
| <i>public boolean teleporterFired</i>          | Menandai apakah <i>teleporter</i> sudah tertembak         |
| <i>public boolean teleporterStillNotAppear</i> | Menandai apakah <i>teleporter</i> sudah muncul atau belum |
| <i>public int teleporterHeading</i>            | Arah dari <i>teleporter</i>                               |
| <i>public int tpReason=0</i>                   | Tujuan dari teleport (untuk makan atau menghindar)        |
| <i>public GameObject target</i>                | Target dari <i>bot</i>                                    |
| <i>public int intention</i>                    | Tujuan dari teleport (untuk makan atau menghindar)        |

e. PlayerAction.java

| Attributes                         | Description        |
|------------------------------------|--------------------|
| <i>public UUID playerId</i>        | ID <i>player</i>   |
| <i>public PlayerActions action</i> | Aksi <i>player</i> |
| <i>public int heading</i>          | Arah <i>player</i> |

| Methods  | Description                                       |
|--|---|
| <i>public UUID getPlayerId()</i>                   | Method yang akan mengembalikan ID <i>player</i>   |
| <i>public void setId(UUID playerId)</i>            | Method untuk menetapkan ID <i>player</i>          |
| <i>public PlayerActions getAction()</i>            | Method yang akan mengembalikan aksi <i>player</i> |
| <i>public void setAction(PlayerActions action)</i> | Method untuk menetapkan aksi <i>player</i>        |
| <i>public int getHeading()</i>                     | Method yang akan mengembalikan arah <i>player</i> |
| <i>public void setHeading(int heading)</i>         | Method untuk menetapkan arah                      |

|  |               |
|--|---------------|
|  | <i>player</i> |
|--|---------------|

f. Position.java

| Attributes          | Description |
|---------------------|-------------|
| <i>public int x</i> | Koordinat x |
| <i>public int y</i> | Koordinat y |

| Methods                              | Description   |
|--------------------------------------|---|
| <i>public Position()</i>             | Constructor untuk <i>Position</i>                       |
| <i>public Position(int x, int y)</i> | Constructor dengan parameter untuk <i>Position</i>      |
| <i>public int getX()</i>             | Method yang akan mengembalikan koordinat X suatu posisi |
| <i>public void setX(int x)</i>       | Method untuk menetapkan koordinat X suatu posisi        |
| <i>public int getY()</i>             | Method yang akan mengembalikan koordinat Y suatu posisi |
| <i>public void setY(int y)</i>       | Method untuk menetapkan koordinat Y suatu posisi        |

g. World.java

| Attributes                         | Description                    |
|------------------------------------|--------------------------------|
| <i>public Position centerPoint</i> | Titik tengah dunia             |
| <i>public Integer radius</i>       | Radius dunia / <i>world</i>    |
| <i>public Integer currentTick</i>  | <i>Tick</i> permainan saat ini |

| Methods | Description |
|---------|-------------|
|---------|-------------|

|   |  |
|---|--|
| <code>public Position getCenterPoint()</code>                 | Method untuk mengembalikan posisi dari titik tengah dunia    |
| <code>public void setCenterPoint(Position centerPoint)</code> | Method untuk menetapkan posisi titik tengah dunia            |
| <code>public Integer getRadius()</code>                       | Method yang akan mengembalikan radius dari dunia             |
| <code>public void setRadius(Integer radius)</code>            | Method untuk menetapkan radius dari dunia                    |
| <code>public Integer getCurrentTick()</code>                  | Method yang akan megembalikan <i>tick</i> permainan saat ini |
| <code>public void setCurrentTick(Integer currentTick)</code>  | Method untuk menentukan <i>tick</i> permainan saat ini       |

#### D. Services

##### a. BotService.java

| Attributes  | Description                   |
|---|-------------------------------|
| <code>private GameObject bot</code>                           | <i>Bot</i>                    |
| <code>private PlayerAction playerAction</code>                | Aksi dari <i>bot</i>          |
| <code>private GameState gameState</code>                      | Keadaan permainan saat ini    |
| <code>private LocalState localState = new LocalState()</code> | Keadaan <i>bot</i> saat ini   |
| <code>private boolean init</code>                             | Menandai apakah permainan ada |

| Methods   | Description                               |
|---|---|
| <code>public BotService()</code>                | Constructor untuk <i>BotService</i>       |
| <code>public GameObject getBot()</code>         | Method yang akan mengembalikan <i>bot</i> |
| <code>public void setBot(GameObject bot)</code> | Method untuk menetapkan <i>bot</i>        |

|   |   |
|---|---|
| <code>public PlayerAction getPlayerAction()</code>                          | Method yang akan mengembalikan aksi <i>player</i> |
| <code>public void setPlayerAction(PlayerAction playerAction)</code>         | Method untuk menetapkan aksi <i>player</i>        |
| <code>public void computeNextPlayerAction(PlayerAction playerAction)</code> | Method untuk aksi yang akan diambil               |
| <code>public GameState getGameState()</code>                                | Method yang akan mengembalikan <i>gameState</i>   |
| <code>public void setGameState(GameState gameState)</code>                  | Method untuk menetapkan <i>gameState</i>          |
| <code>private void updateSelfState()</code>                                 | Method untuk meng-update state                    |

#### E. Main.java

| Methods  | Description           |
|--|-----------------------|
| <code>public static void main(String[] args) throws Exception</code> | Menjalankan permainan |

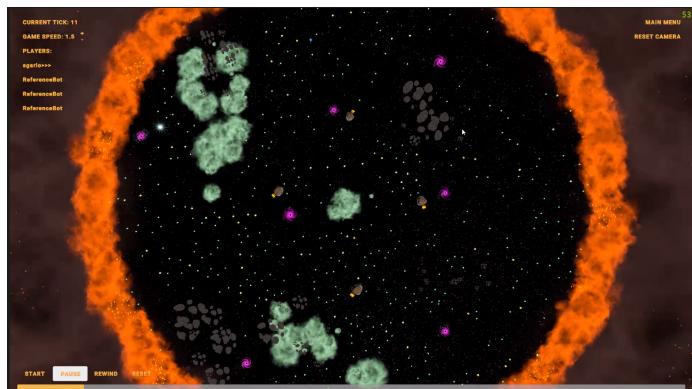
### 4.3 Analisis Desain Solusi Algoritma *Greedy*

Dalam pengujian *bot* dan desain solusi algoritma, dilakukan dua jenis pengujian. Pengujian pertama berupa mempertandingkan *bot* kami, *agario supremacy*, dengan *reference bot* yang disediakan oleh *Entelect*. Hal ini dimaksudkan agar pengujian dapat dilakukan dengan mudah dan strategi *greedy* dari *bot* dapat lebih terlihat. Pengujian jenis kedua berupa mempertandingkan sesama *bot* kami. Hal ini diperlukan untuk melihat kemampuan *bot* untuk bereaksi dalam situasi-situasi yang lebih kompleks dan membutuhkan aksi yang lebih komprehensif. Pengujian yang dilakukan akan memperlihatkan berbagai situasi

dan *edge-case* yang dialami oleh *bot*. Berikut ditampilkan berbagai hasil pengujian dan kasus-kasus yang dialami oleh bot:

#### 4.3.1 Masalah pengambilan *food* atau *superfood* dengan jarak sama

```
=====
Current Tick : 7
Size : 19
Current heading : 61
Position : -1, 352
Action : FORWARD
Action heading : 59
=====
Current Tick : 8
Size : 19
Current heading : 237
Position : -7, 343
Action : FORWARD
Action heading : 238
=====
Current Tick : 9
Size : 19
Current heading : 59
Position : -1, 352
Action : FORWARD
Action heading : 59
=====
Current Tick : 10
Size : 19
Current heading : 238
Position : -7, 343
Action : FORWARD
Action heading : 238
=====
Current Tick : 11
Size : 19
Current heading : 59
Position : -1, 352
Action : FORWARD
Action heading : 59
=====
Current Tick : 12
Size : 19
Current heading : 238
Position : -7, 343
Action : FORWARD
Action heading : 238
=====
Current Tick : 13
Size : 19
Current heading : 59
Position : -1, 352
Action : FORWARD
Action heading : 59
=====
```



Analisis :

Pada kasus ini, terjadi masalah pada *bot* yang berada di atas *map* permainan. Bot tersebut melakukan gerakan bolak-balik di antara dua *food*. Hal ini karena kedua *food* memiliki jarak yang sama, sehingga akhirnya *bot* terus menerus mengganti arah di antara kedua *food* tersebut. Hal ini diverifikasi lewat informasi bot yang ditunjukkan. Terlihat bahwa *bot* berkali-kali mengirimkan perintah FORWARD kepada *engine* dengan *heading* yang berlawanan 180 derajat silih berganti. Hasil ini bukanlah hasil optimal yang diharapkan karena seharusnya *bot* menuju salah satu *food* saja.

#### 4.3.2 Hasil pengujian penembakan torpedo



```
Current Tick      : 14
Size              : 19
Current heading   : 238
Position          : -7, 343
Action            : FIRE_TORPEDOES
Action heading    : 227
=====
Current Tick      : 15
Size              : 19
Current heading   : 59
Position          : -1, 352
Action            : FIRE_TORPEDOES
Action heading    : 227
```

Analisis :

Pada kasus diatas, terlihat bahwa *bot* yang berukuran lebih besar berusaha untuk menembak *bot* lainnya dengan menggunakan *torpedo*. Hal tersebut juga dapat dilihat pada informasi *bot* yang ditampilkan di atasnya. Terlihat bahwa *bot* mengirimkan *command* FIRE\_TORPEDOES kepada *engine*. Hal ini dilakukan

agar bot bisa mengambil atau mencuri ukuran lawan dan memperbesar ukurannya sendiri (*greedy by size*). Hal ini sesuai dan optimal dengan hasil yang diinginkan.

#### 4.3.3 Hasil pengujian penembakan *teleporter* ke lawan yang lebih kecil



```
Current Tick      : 100
Size              : 105
Current heading  : 202
Position          : 90, 210
Action            : FIRE_TELEPORT
Action heading   : 260
```



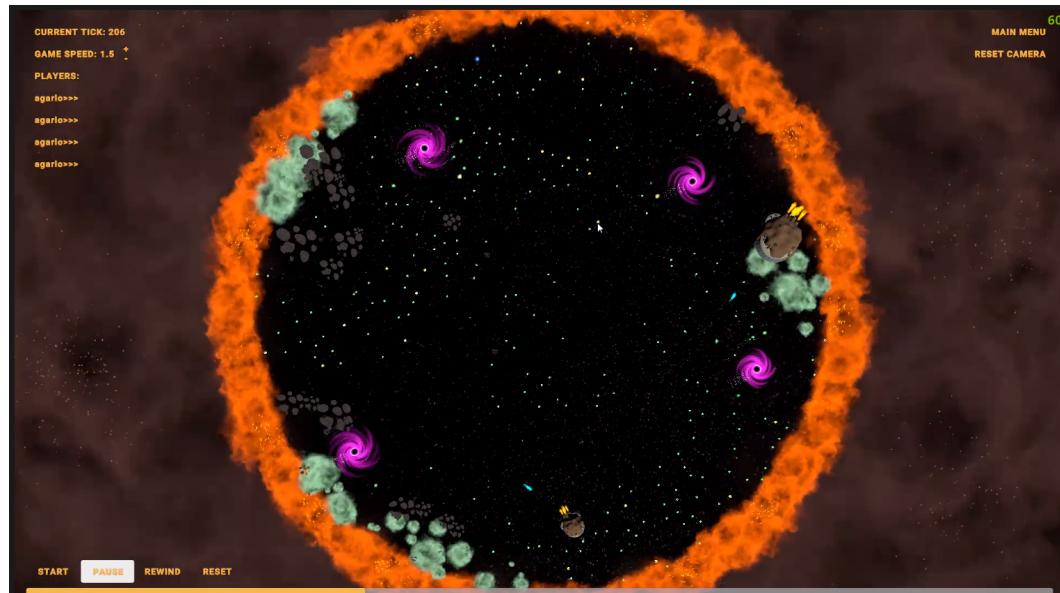
```
Current Tick      : 111
Size              : 80
Current heading   : 202
Position          : 59, 199
Action            : TELEPORT
Action heading    : 143
=====
Current Tick      : 112
Size              : 75
Current heading   : 202
Position          : 56, 198
Action            : FORWARD
Action heading    : 202
=====
Current Tick      : 113
Size              : 133
Current heading   : 22
Position          : 62, -114
Action            : FORWARD
Action heading    : 225
```

Analisis :

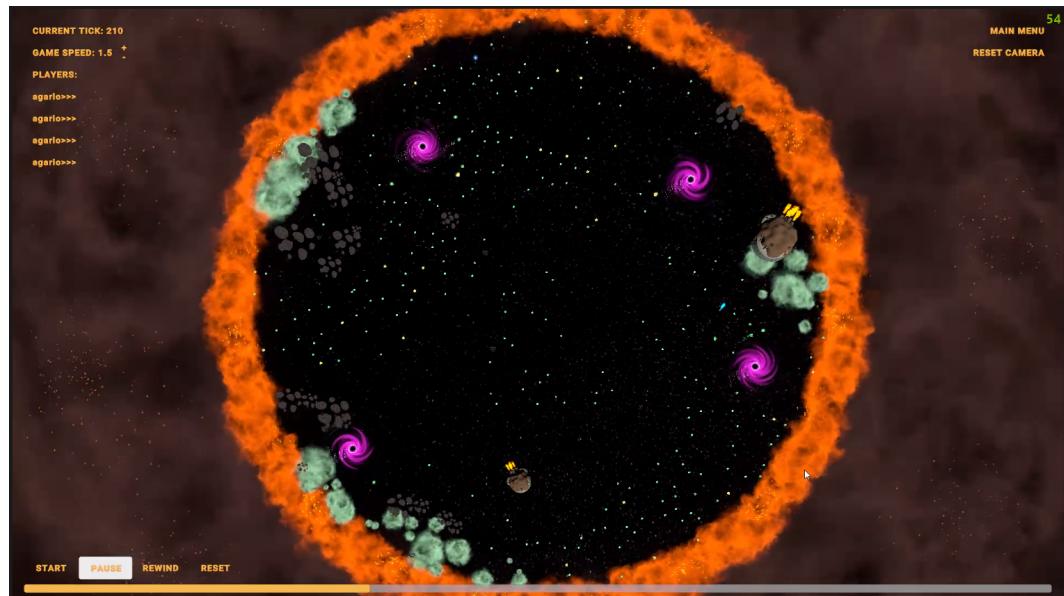
Pada 2 gambar pertama, terlihat bahwa *bot* yang lebih besar berhasil mengirim *command* untuk menembakkan *teleporter* ke arah lawan yang lebih kecil saat *bot* memiliki ukuran terbesar. Pada 2 gambar terakhir, terlihat bahwa *bot* melakukan

*teleport* saat *teleporter* sudah dekat dengan lawan tersebut. Hal ini dilakukan untuk memakan pemain tersebut dan menambah *size* sesuai strategi. Hasil sesuai dan optimal.

#### 4.3.4 Hasil pengujian menghindari *teleporter* lawan



```
Current Tick      : 202
Size              : 56
Current heading   : 302
Position          : 78, -576
Action            : FIRE_TELEPORT
Action heading    : 143
```



```
=====
Current Tick      : 207
Size              : 36
Current heading   : 302
Position          : 91, -597
Action            : TELEPORT
Action heading    : 117
=====
```

#### Analisis :

Pada 2 gambar pertama, *bot* yang lebih kecil berhasil menembakkan *teleporter* ke arah tegak lurus dari *teleporter* lawan. Pada 2 gambar terakhir, *bot* tersebut kemudian melakukan *teleport* saat jarak *teleporter* sudah cukup jauh. Hal ini dilakukan untuk menghindari dimakan oleh *bot* yang menembakkan *teleporter*. Hasil ini sesuai dengan hasil yang diinginkan dan optimal.

#### 4.3.5 Hasil pengujian aktivasi *shield* saat *torpedo* lawan dekat dan menuju *bot*



```
=====
Current Tick      : 35
Size              : 46
Current heading   : 192
Position          : 98, -1
Action            : ACTIVATE_SHIELD
Action heading    : 192
=====
```

Analisis :

Terlihat bahwa *bot* berhasil menyalakan *shield* sebelum terkena *torpedo*. Hal ini juga dapat diverifikasi lewat informasi *bot* yang mengirim *command* ACTIVATE\_SHIELD kepada *engine*. Hal ini dilakukan untuk mencegah berkurangnya *size* *bot*. Hasil sesuai dengan keinginan dan optimal.

#### 4.3.6 Masalah navigasi bot di dalam gas clouds

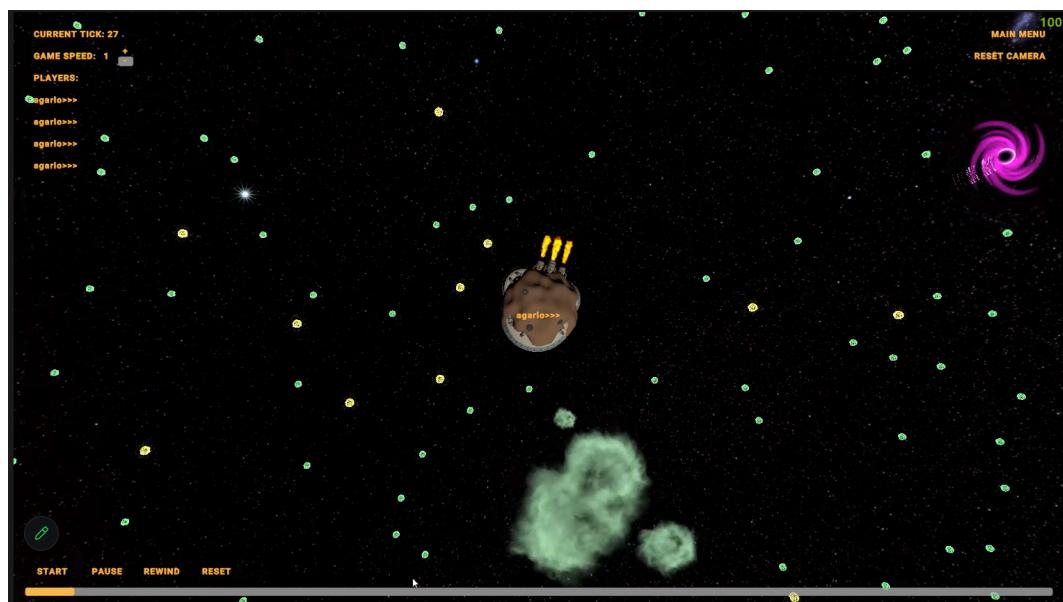


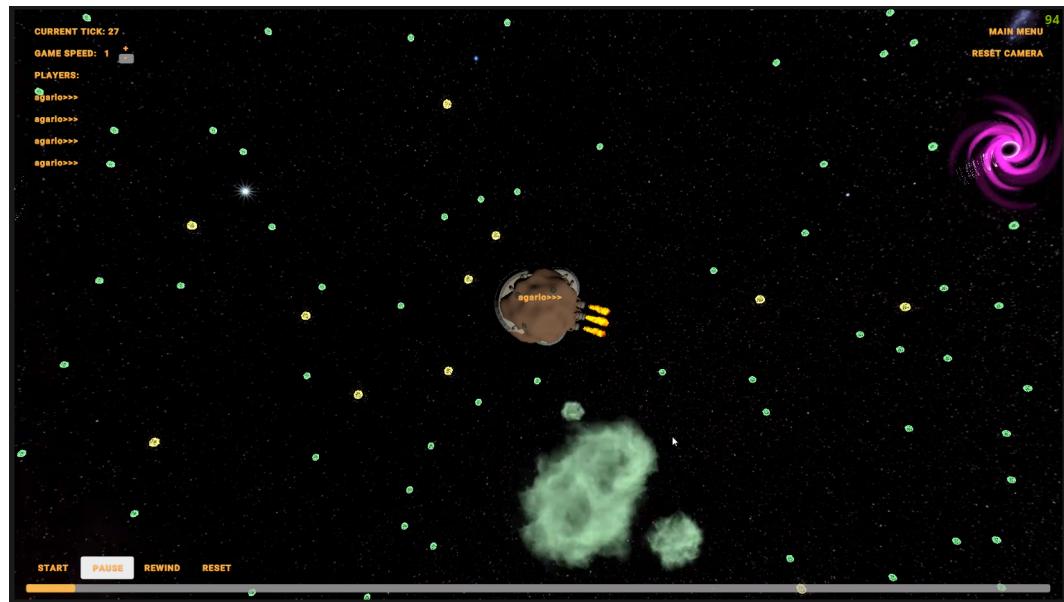
```
=====
Current Tick      : 158
Size              : 177
Current heading   : 339
Position          : 613, 270
Action            : FORWARD
Action heading    : 337
=====
Current Tick      : 159
Size              : 175
Current heading   : 338
Position          : 615, 270
Action            : FORWARD
Action heading    : 336
=====
Current Tick      : 160
Size              : 173
Current heading   : 338
Position          : 617, 270
Action            : FORWARD
Action heading    : 335
=====
Current Tick      : 161
Size              : 171
Current heading   : 336
Position          : 619, 270
Action            : FORWARD
Action heading    : 334
```

Analisis :

Terlihat dari informasi *bot* yang dikirim, bahwa *bot* masih kesulitan untuk keluar dari dalam *gas cloud*. Informasi *bot* menunjukkan bahwa *heading* dari *bot* hanya berubah sedikit demi sedikit dan menyebabkan *bot* berputar-putar di dalam *gas clouds*. Hasil ini tidak sesuai dengan keinginan dan tidak optimal.

#### 4.3.7 Hasil pengujian ketika bot di dekat gas clouds





```
=====
Current Tick      : 27
Size              : 37
Current heading   : 264
Position          : -391, -117
Action            : FORWARD
Action heading    : 264
=====
Current Tick      : 28
Size              : 37
Current heading   : 165
Position          : -397, -115
Action            : FORWARD
Action heading    : 160
=====
Current Tick      : 29
Size              : 37
Current heading   : 264
Position          : -398, -121
Action            : FORWARD
Action heading    : 270
=====
```

Analisis :

Pada gambar pertama, terlihat bahwa *bot* mengarah ke arah *gas clouds*. Di *tick* berikutnya, *bot* berhasil menjauhi *gas cloud* dengan berbelok arah menjauhi *gas clouds*. Hasil ini sesuai dengan keinginan dan optimal.

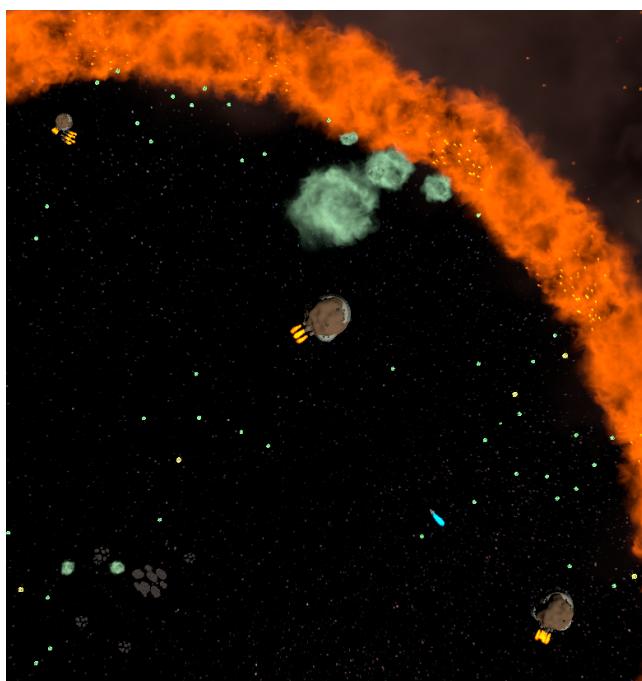
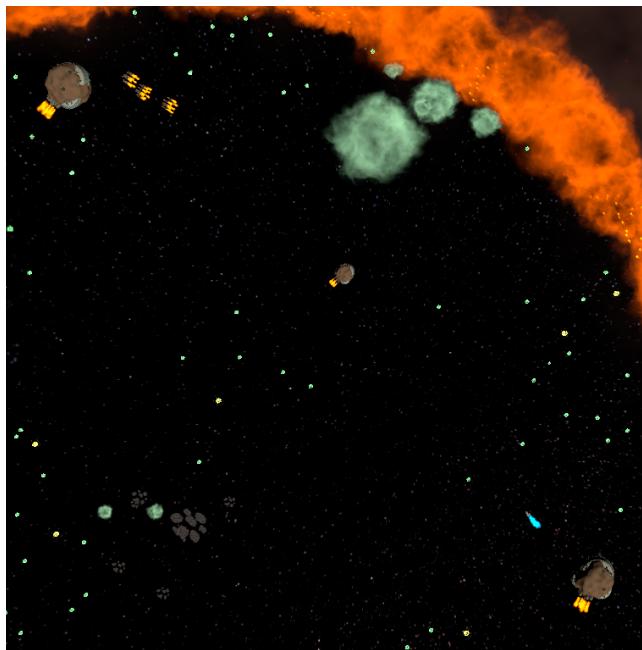
#### 4.3.8 Hasil pengujian ketika ada makanan yang dekat dengan ujung *world*



Analisis :

Terlihat bahwa *bot* tidak mengambil *food* yang terlalu dekat dengan ujung *world*. Hal ini dilakukan agar ukuran *bot* tidak berkurang akibat bersentuhan dengan ujung *world*. Hal ini sesuai dengan keinginan dan optimal.

#### 4.3.9 Masalah *bot* mencoba menghindari *teleporter* namun ada *torpedo* yang menuju ke arah *bot*

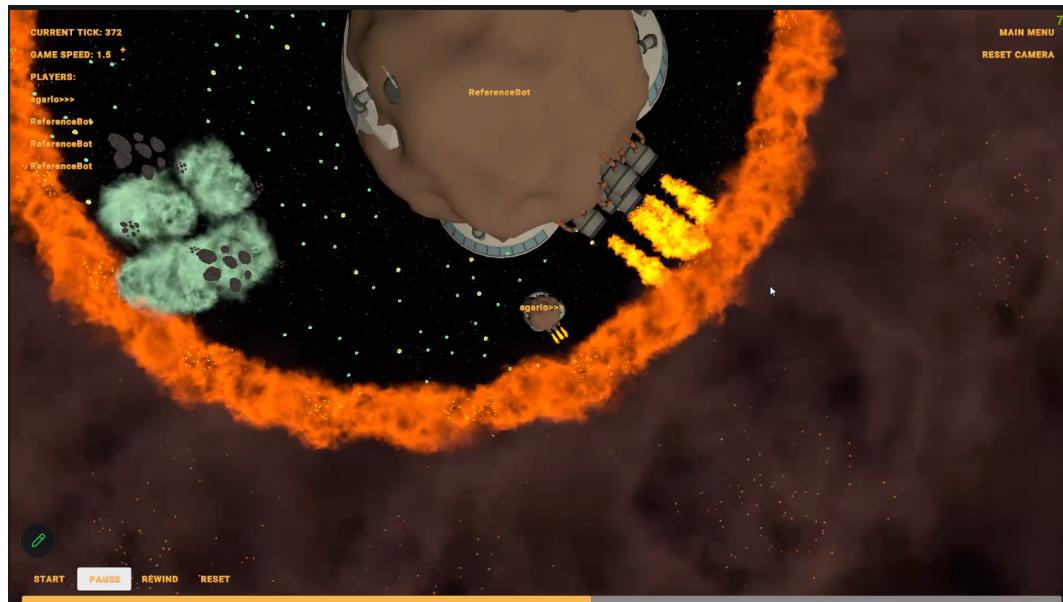


Analisis :

Pada gambar pertama, terlihat bahwa *bot* di paling atas kiri tidak memiliki *teleporter* maupun *shield* yang bisa dipakai. Jadi, *bot* mencoba menghindari *teleporter* yang menuju ke arahnya dengan mengubah arah menjadi tegak lurus

terhadap *teleporter* lawan. Namun, ada *bot* lain yang menembakkan *torpedo* ke arah tersebut. *Bot* pun tidak berhasil menghindari *torpedo* dan akhirnya tereliminasi. Hasil ini tidak optimal.

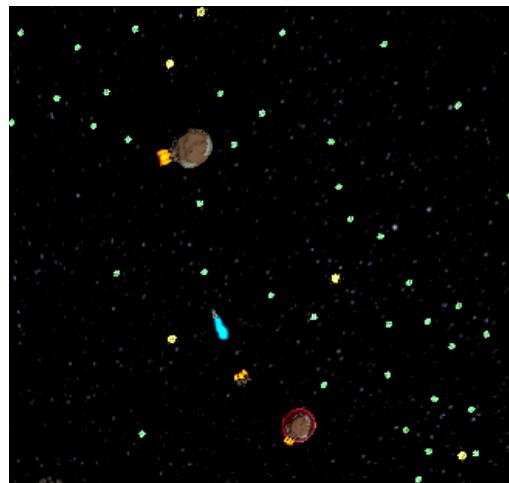
#### 4.3.9 Hasil pengujian *bot* di dekat lawan yang lebih besar



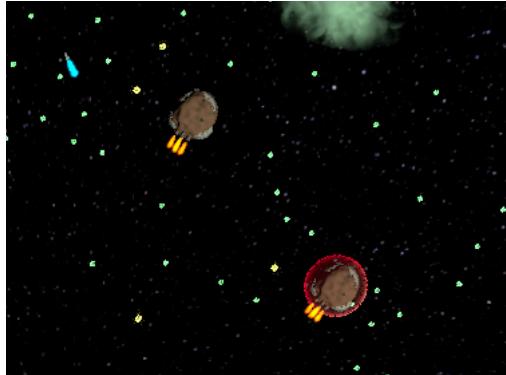
Analisis :

Terlihat bahwa *bot* terus-menerus berbalik arah saat berdekatan dengan *bot* yang lebih besar. Hal ini terjadi karena *bot* berusaha untuk menghindari musuh yang lebih besar dan juga menghindari *world border*. Meskipun *bot* berhasil menjaga ukurannya, hasil ini kurang optimal karena *bot* terus terganggu.

4.3.10 Hasil pengujian *bot* sudah menembakkan *teleporter* namun lawan yang dituju menjadi terlalu besar untuk dimakan



```
=====
Current Tick    : 64
Size           : 49
Current heading : 114
Position        : 176, 88
Action          : FIRE_TELEPORT
Action heading  : 114
=====
Current Tick    : 65
Size           : 52
Current heading : 115
Position        : 176, 92
Action          : ACTIVATE_SHIELD
Action heading  : 64
=====
Current Tick    : 66
```



```
=====
Current Tick    : 77
Size           : 18
Current heading : 74
Position        : 270, 186
Action          : FORWARD
Action heading  : 95
=====
Current Tick    : 78
Size           : 21
Current heading : 90
Position        : 270, 198
Action          : FIRE_TORPEDOES
Action heading  : 120
=====
Current Tick    : 79
Size           : 21
Current heading : 95
Position        : 269, 208
Action          : FIRE_TORPEDOES
Action heading  : 121
=====
Current Tick    : 80
Size           : 16
Current heading : 95
Position        : 268, 221
Action          : FORWARD
```

#### Analisis :

Pada 2 gambar pertama, terlihat bahwa *bot bawah* berusaha untuk menembak *teleporter* untuk memakan *bot atas*. Namun, *bot atas* kemudian menembakkan *torpedo* yang menyebabkan *bot bawah* harus mengaktifkan *shield*. Hal ini menyebabkan ukuran *bot atas* yang menjadi sasaran terlalu besar untuk dimakan saat *teleporter* mencapai lawan tersebut. Oleh karena itu, *bot* tidak melakukan *teleport* saat lawan yang dituju terlalu besar. Hasil ini sesuai dengan keinginan dan optimal.

Dari berbagai pengujian yang dilakukan, terlihat bahwa desain solusi algoritma *Greedy* yang dipilih sudah berhasil untuk mengimplementasikan sebagian besar strategi yang ditulis pada subbab 3.4. Meskipun pada beberapa kasus, *bot* belum memilih aksi yang optimal untuk memenangkan permainan.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Kami berhasil membuat *bot* yang dapat memenangkan permainan game *Galaxio*. *Bot* ini dibuat dengan mengimplementasikan algoritma *Greedy*. *Bot* ini berhasil mencapai tujuan dari game ini yaitu menang dengan menjadi pemain yang terakhir hidup. Dari *match-match* yang telah dilakukan, dapat dilihat bahwa implementasi algoritma *Greedy* sangat efektif dalam game ini. Dengan adanya strategi *Greedy*, setiap aksi yang dilakukan oleh *bot* menjadi optimal. Misalnya, jika *bot* sudah menjadi yang terbesar, maka aksi yang akan dilakukan oleh *bot* adalah menembak *teleporter* ke arah *player* lain yang lebih kecil dan *teleport* ke *player* tersebut dengan tujuan memakan *player* tersebut. Dengan melakukan ini, ukuran *bot* akan semakin besar dan kemungkinan menang semakin tinggi. Dari contoh kasus ini dapat dilihat bahwa *Greedy by size* menjadi pilihan yang optimal.

#### **5.2 Saran**

1. Objek supernova bisa digunakan untuk membantu *bot* memenangkan permainan.
2. Untuk selanjutnya, sebaiknya dilakukan percobaan untuk implementasi algoritma *Greedy* lainnya agar dapat mengetahui strategi mana yang cocok untuk kasus-kasus tertentu.

## **DAFTAR PUSTAKA**

- Munir, R. (2023). Algoritma Greedy (Bagian 1). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023. Diakses pada 13 Februari 2023, pukul 21.20 dari  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)
- Entelect. (2021). 2021-Galaxio. Entelect Challenge. Diakses pada 2 Februari 2023, pukul 19.57 dari [EntelectChallenge/2021-Galaxio \(github.com\)](https://github.com/EntelectChallenge/2021-Galaxio)

# **LAMPIRAN**

## **Link Repository Program**

[GoDillonAudris512/Tubes1\\_agario-supremacy \(github.com\)](https://github.com/GoDillonAudris512/Tubes1_agario-supremacy)

## **Link Video Demo Youtube**

[Tugas Besar Strategi Algoritma 1 Kelompok Agario Supremacy - YouTube](https://www.youtube.com/watch?v=KJLjyfzXWUo)