

**LAPORAN TUGAS BESAR 1**  
**IF2211**  
**STRATEGI ALGORITMA**

**PEMANFAATAN ALGORITMA GREEDY DALAM**  
**APLIKASI PERMAINAN “GALAXIO”**

**SEMESTER II TAHUN 2022/2023**

**oleh:**

<b>Wilson Tansil</b>	<b>13521054</b>
<b>Frankie Huang</b>	<b>13521092</b>
<b>William Nixon</b>	<b>13521123</b>

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

**2023**

**Daftar Isi**

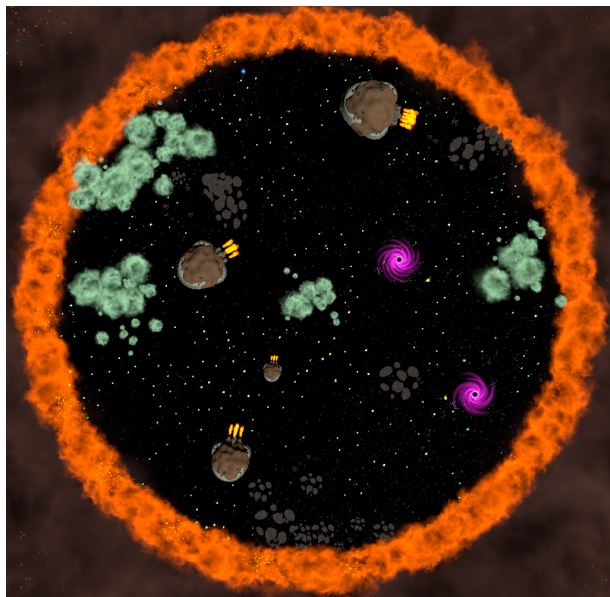
<b>Daftar Isi</b>	<b>2</b>
<b>BAB 1. DESKRIPSI MASALAH</b>	<b>3</b>
1.1. Deskripsi Tugas	3
1.2. Spesifikasi Permainan	4
<b>BAB 2. TEORI SINGKAT</b>	<b>6</b>
2.1. Algoritma Greedy	6
2.2. Teknis Permainan Galaxio	7
2.2.1. Panduan Singkat Instalasi	7
2.2.2. Teknis Menentukan Aksi Bot	7
2.2.3. Menjalankan dan Memvisualisasikan Permainan	8
<b>BAB 3 APLIKASI STRATEGI GREEDY</b>	<b>11</b>
3.1. Pemetaan Persoalan Galaxio menjadi Algoritma Greedy	11
3.1.1. Himpunan Kandidat	11
3.1.2. Himpunan Solusi	11
3.1.3. Fungsi Solusi	11
3.1.4. Fungsi Seleksi	11
3.1.5. Fungsi Kelayakan	11
3.1.6. Fungsi Objektif	11
3.2. Eksplorasi Alternatif Solusi Greedy	12
3.3. Analisis Efisiensi Secara Teoritis dari Alternatif Solusi Persoalan	13
3.4. Analisis Efektivitas Secara Teoritis dari Alternatif Solusi Persoalan	14
3.5. Analisis Strategi Greedy yang Dipilih	14
<b>BAB 4 IMPLEMENTASI DAN PENGUJIAN</b>	<b>15</b>
4.1. Implementasi Algoritma Greedy	15
4.1.1. Fungsi Driver	15
4.1.2. Fungsi Escape	16
4.1.3. Fungsi Attack	17
4.1.4. Fungsi Defend	18
4.1.5. Fungsi Eat	18
4.2. Struktur Data yang Digunakan	18
4.3. Pengujian dan Analisis Desain Algoritma Greedy	20

<b>BAB 5 KESIMPULAN DAN SARAN</b>	<b>22</b>
5.1. Kesimpulan	22
5.2. Saran	23
<b>DAFTAR PUSTAKA</b>	<b>23</b>

## **BAB 1. DESKRIPSI MASALAH**

### **1.1. 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. 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>

## 1.2. Spesifikasi Permainan

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 salvo torpedo (ukuran 10) mengurangi ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. *Food* akan disebar 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 menembakannya 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 mengonsumsi 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*. Untuk daftar *commands* yang tersedia, bisa merujuk ke tautan [panduan](#) di spesifikasi tugas.
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 2. TEORI SINGKAT**

### **2.1. Algoritma Greedy**

Algoritma greedy merupakan algoritma yang mengikuti suatu heuristik yang telah ditentukan sebelumnya, dengan harapan agar dapat memilih solusi optimum lokal di setiap langkah. Di dalam banyak permasalahan, algoritma ini tidak menghasilkan solusi yang optimum secara global, tetapi pemilihan heuristik greedy yang baik dapat memperoleh solusi optimum lokal yang cukup baik di dalam durasi waktu yang lebih singkat pula.

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. Dalam tugas ini, algoritma greedy akan digunakan untuk menyelesaikan permasalahan optimasi “Memperpanjang Masa Bertahan Hidup dalam Permainan Galaxio”.

## 2.2. Teknis Permainan Galaxio

### 2.2.1. Panduan Singkat Instalasi

Langkah pertama yang dapat dilakukan untuk memulai adalah sebagai berikut.

1. Download latest release starter pack.zip dari tautan berikut  
<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>
2. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut.
  - Java (minimal Java 11):  
<https://www.oracle.com/java/technologies/downloads/#java>
  - IntelliJ IDEA: <https://www.jetbrains.com/idea/>
  - .Net Core 3.1: Link tersedia pada panduan berikut.

Panduan mengenai cara menjalankan permainan, membuat bot, build src code, dan melihat visualizer bisa dicek melalui tautan berikut.

[https://docs.google.com/document/d/1Ym2KomFPLIG\\_KAbm3A0bnhw4\\_XQAsOKzpTa70IgnLNU/edit#](https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa70IgnLNU/edit#)

Starter-pack mengandung semua bagian kode yang diperlukan untuk menjalankan bot di mesin lokal. Beberapa modul yang telah diberikan yaitu:

1. Game Engine - Bertanggung jawab untuk memastikan aturan di dalam permainan berjalan dengan baik dengan memanggil perintah perintah dari bot dan memastikan bahwa mereka valid.
2. Game Runner - Bertanggung jawab untuk mengatur permainan yang sedang berjalan di antara pemain, akan memanggil perintah yang diberikan oleh bot dan memberikannya kepada Game Engine.
3. Logger - Bertanggung jawab untuk mencatat segala perubahan state di dalam game dan membuat sebuah file log yang dapat divisualisasikan.
4. Starter Bots - Terdiri dari bot sederhana yang dapat menjadi acuan pengembangan bot selanjutnya. Terdapat juga bot referensi yang dapat menjadi acuan uji coba kelayakan bot.

### 2.2.2. Teknis Menentukan Aksi Bot

Terdapat beberapa bot referensi dalam berbagai berbahasa di dalam starter pack. Program bot ini memiliki tanggung jawab sebagai berikut:

1. Mengkoneksikan diri ke SignalR (Game Runner yang telah berjalan)
2. Meregistrasikan token dan nickname bot ke runner.
3. Mendengarkan game runner setiap terjadi event yang telah dipublish.
4. Merespon terhadap event tersebut dengan suatu aksi baru.

```

public class BotService {
    private GameObject bot;
    private PlayerAction playerAction;
    private GameState gameState;

    private boolean firedTeleporter;
    private Integer headingTeleporter;
    private boolean startafterburner;
    public List<GameObject> foodList;
    public List<GameObject> superFoodList;
    public List<GameObject> nearPlayerList;
    public List<GameObject> torpedoList;
    public List<GameObject> teleporterList;
    public List<GameObject> asteroidList;
    public List<GameObject> gasCloudList;
    public List<GameObject> wormHoleList;
}

```

Terdapat sebuah kelas bernama BotService yang memiliki atribut sebagai berikut. Atribut List merupakan atribut yang kelompok kami tambahkan untuk mengklasifikasikan objek sesuai kedekatannya dengan bot tersebut. Kami juga menambahkan beberapa boolean seperti firedTeleporter dan startAfterBurner untuk mengingat apakah bot kami telah mengaktifkan fitur-fitur tersebut.

Pada setiap game tick, maka method computeNextPlayerAction akan dipanggil. Untuk mengubah perilaku dari bot di tick tersebut, maka atribut heading dan playerAction dari bot tersebut dapat diubah sesuai keinginan. Terdapat beberapa method-method built-in yang dapat membantu melakukan kalkulasi pada program, seperti getHeadingBetween() untuk mengetahui sudut dua objek dan juga beberapa enum untuk menentukan jenis aksi ataupun objek.

### 2.2.3. Menjalankan dan Memvisualisasikan Permainan

Program yang telah diimplementasikan harus di build terlebih dahulu menggunakan Maven (package) sehingga terbentuk sebuah file .jar. File .jar ini kemudian dapat diaktifkan agar didaftarkan oleh game engine sebagai salah satu pemain. Pada sistem operasi Windows, sistem permainan dapat dijalankan dengan mengeksekusi batch script berikut setelah dotnet berhasil terpasang pada direktori root.

```

@echo off
:: Game Runner
cd ./runner-publish/

```



```

start "" dotnet GameRunner.dll

:: Game Engine
cd ../engine-publish/
timeout /t 1
start "" dotnet Engine.dll

:: Game Logger
cd ../logger-publish/
timeout /t 1
start "" dotnet Logger.dll

:: Bots
cd ../reference-bot-publish/
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" dotnet ReferenceBot.dll
cd ../

pause

```

Untuk sistem operasi berbasis UNIX/Linux, permainan dapat dijalankan dengan mengeksekusi shell script berikut setelah dotnet berhasil terpasang pada direktori root.

```

#!/bin/bash

export DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1
cd ./runner-publish/ && dotnet GameRunner.dll &
cd ./engine-publish/ && sleep 1 && dotnet Engine.dll &
cd ./logger-publish/ && sleep 1 && dotnet Logger.dll &

# Bots
cd ./reference-bot-publish/ && sleep 3 && dotnet ReferenceBot.dll &
cd ./reference-bot-publish/ && sleep 3 && dotnet ReferenceBot.dll &
cd ./reference-bot-publish/ && sleep 3 && dotnet ReferenceBot.dll &
cd ./reference-bot-publish/ && sleep 3 && dotnet ReferenceBot.dll &

```

```
wait
```

Bagian :: Bots dapat dimodifikasi dari menjalankan Reference Bot menjadi Bot yang telah dibuat dengan menjalankan file .jar yang telah di package. Setelah ini, game akan berjalan dan ketika selesai menghasilkan sebuah file .log di dalam folder game-logger. File ini dapat divisualisasikan dengan membuka aplikasi visualizer di dalam starter-pack.

Untuk lebih lengkapnya, dapat mengacu kepada dokumen referensi getting started yang telah disediakan oleh tim Strategi Algoritma.

[[https://docs.google.com/document/d/1Ym2KomFPLIG\\_KAbm3A0bnhw4\\_XQAsOKzpTa70IgnLNU/edit?pli=1](https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa70IgnLNU/edit?pli=1)]

## BAB 3 APLIKASI STRATEGI GREEDY

### 3.1. Pemetaan Persoalan Galaxio menjadi Algoritma Greedy

Dalam permainan Galaxio, algoritma greedy diharapkan dapat memperpanjang masa hidup bot hasil implementasi agar dapat menjadi yang terakhir bertahan hidup.

#### 3.1.1. Himpunan Kandidat

Mode *eat, attack, escape, defend*.

Command & Heading Player Action (FORWARD, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRETELEPORT, TELEPORT, ACTIVATESHIELD)

#### 3.1.2. Himpunan Solusi

Pilihan yang mempertinggi kemungkinan bot bertahan hidup paling lama melalui mode yang telah dipilih serta Command dan Heading pada player action.

#### 3.1.3. Fungsi Solusi

Fungsi untuk menentukan apakah bot implementasi bertahan hidup paling lama.

#### 3.1.4. Fungsi Seleksi

Memilih mode yang memenuhi suatu kumpulan persyaratan, dengan urutan sebagai berikut: *eat, defend, escape, attack*.

Fungsi *eat* memilih arah forward terbaik untuk memperoleh makanan yang terdekat. Fungsi *defend* memilih arah forward terbaik, mengaktifkan shield untuk menghindari proyektil musuh. Fungsi *escape* memilih arah forward terbaik, mengaktifkan afterburner untuk melarikan diri dari musuh. Fungsi *attack* memilih arah forward terbaik untuk mengejar musuh, arah penembakan torpedo, dan penembakan teleport serta waktu teleport terbaik untuk membunuh musuh.

#### 3.1.5. Fungsi Kelayakan

Memeriksa apakah *command* dan koordinat adalah aksi yang valid dengan mengecek status apakah *spaceship* pemain bisa melakukan tindakan tertentu dan memiliki *salvo*, *shield*, dan *teleporter* lebih besar dari 0.

#### 3.1.6. Fungsi Objektif

Memilih tindakan yang memaksimalkan waktu bertahan hidup bot implementasi.

Memilih tindakan yang memaksimalkan ukuran dari bot implementasi.

Memilih tindakan yang mengecilkan/membunuh bot lawan.

## 3.2. Eksplorasi Alternatif Solusi Greedy

### 3.2.1. Definisi Solusi Greedy Defensive

Implementasi dari solusi ini terbagi menjadi dua jenis yakni defend dengan shield dan afterburner. Afterburner sendiri akan diaktifkan ketika ada bot lawan yang mendekat dalam radius konstanta tertentu. Arah hindar ditentukan dari angle lawan ditambah konstanta derajat tertentu sehingga bot tersebut tetap berada di dalam arena permainan. Afterburner sendiri akan mati apabila bot hasil implementasi telah berada di dalam jarak yang aman dengan mempertimbangkan jarak kedua bot dan ukuran bot. Defend dengan menggunakan shield akan diaktifkan apabila terdapat salvo yang mendekat dalam radius konstanta yang telah ditentukan.

### 3.2.2. Definisi Solusi Greedy Offensive

Pemain akan selalu berusaha untuk melakukan serangan pada musuh jika ukuran suatu pemain melewati *threshold*. Pada awal permainan, pemain akan berfokus untuk mencari makanan secepatnya agar ukuran bot mumpuni untuk melakukan aksi penyerangan. Untuk metode penyerangan, diperlukan suatu nilai konstan dimana pemain dapat memilih untuk mencari lawan atau mulai menyerang lawan. Jika jarak antara pemain dan lawan berada dalam suatu radius konstan tersebut, maka pemain akan mulai menyerang lawan dengan menembakkan *SALVO* atau *TELEPORTER*.

Pemain akan menembakkan *SALVO* ke arah musuh agar dapat memperkecil ukuran musuh tersebut dan menambah ukuran diri sendiri. Pemain juga dapat menembakkan *TELEPORTER* ke arah musuh yang lebih kecil, dan kemudian melakukan *TELEPORT* untuk memakan musuh tersebut. Pada alternatif ini, tidak terdapat suatu metode / aksi yang akan dieksekusi jika pemain berada dalam keadaan yang berbahaya.

### 3.2.3. Definisi Solusi Greedy Skala Prioritas

Pemain akan memiliki skala prioritas yang diberikan kepada setiap perintah di dalam *playerAction*, yang memiliki nilai antara 0 - 10. Akan terdapat fungsi greedy untuk setiap jenis *playerAction* yang akan mengembalikan objek *playerAction* dengan nilai prioritas tertentu. Misalnya, terdapat fungsi independen untuk menentukan gerakan, fungsi independen untuk menentukan apakah harus menembakkan *SALVO* atau *TELEPORTER*, dan lain sebagainya.

Pada setiap game tick, maka seluruh fungsi yang merepresentasikan *playerAction* akan dipanggil. Nilai return yang paling tinggi akan selalu mendapatkan prioritas untuk dijalankan di tick tersebut, misalnya command FORWARD mendapat prioritas 2 dan ACTIVATE SHIELD mendapat prioritas 8 (karena bot dihadapkan dengan misil), maka pada tick tersebut, bot akan memilih ACTIVATE SHIELD sebagai *playerAction*.

### 3.2.4. Definisi Solusi Greedy Campuran

Pemain akan menentukan aksi terbaik yang akan dilakukan yang terbagi menjadi 4 jenis “mode”, yaitu mode *eat*, *attack*, *escape*, dan *defend*. Pada awal permainan, pemain akan memasuki mode *eat*, dimana pemain akan berusaha untuk mencari makanan secepatnya agar ukuran bot membesar. Kemudian, pemain akan mengecek apakah pemain berada dalam keadaan *isDanger* atau *isStrong*.

Jika pemain berada dalam keadaan *isDanger*, maka akan dilakukan pengecekan kedua, dimana pemain akan mengecek apakah salvo terdekat dari pemain berada dalam suatu jarak tertentu. Jika iya, maka shield akan diaktifkan dan jika tidak, maka pemain akan berusaha untuk menjauhi musuh tersebut. Jika pemain berada dalam keadaan *isStrong*, maka pemain akan mulai menyerang lawan yang terdekat dari pemain. Jika pemain tidak berada dalam kedua mode tersebut, maka pemain akan memasuki mode *eat*, dimana pemain akan berusaha untuk mencari makanan terdekat.

## 3.3. Analisis Efisiensi Secara Teoritis dari Alternatif Solusi Persoalan

### 3.3.1. Efisiensi Solusi Greedy Defensive

Algoritma ini cukup efisien untuk diimplementasikan, mengingat bahwa algoritma cukup melihat apakah bot berada pada jarak tertentu dan melihat apakah bot yang saling berdekatan (yang berada pada radius *threshold*) itu sendiri memiliki size yang lebih besar atau tidak atau salvo yang mengarah ke kita. Arah pelarian dari bot tersebut tidak mengarah secara linear melainkan sesuai dengan pergerakan lawan yang paling mendekati ditambah dengan sudut tertentu. Apabila terdapat bot yang lebih besar dan berdekatan, serta kita masih memiliki ukuran untuk mengaktifkan afterburner maka akan dihidupkan afterburner atau apabila kita masih memiliki shield dan terdapat salvo yang mengarah ke kita maka akan diaktifkan shield. Algoritma ini melakukan pengecekan jarak setiap player dan misil sehingga memiliki kompleksitas waktu  $O(N)$ .

### 3.3.2. Efisiensi Solusi Greedy Offensive

Algoritma ini cukup efisien dan mudah untuk diimplementasikan seperti pada solusi sebelumnya. Pada alternatif ini, pemain akan bermain secara *offensive*, dimana pemain akan mulai mencari dan menyerang lawan jika melebihi suatu ukuran tertentu. Jika bot masih berukuran di bawah *threshold*, maka bot akan berfokus untuk mencari makanan agar bisa bertambah besar. Jika ukuran bot telah melebihi *threshold*, maka bot akan mulai mendekati dan menyerang bot pemain lawan. Untuk mendekati musuh, cara yang akan diimplementasikan cukup mudah, yaitu cari saja musuh dengan ukuran yang terkecil, lalu arahkan *heading* kita menuju lokasi musuh. Untuk mulai menembak *salvo* atau *teleporter* diperlukan suatu konstanta jarak antara pemain dan musuh agar kemungkinan *salvo* atau *teleporter* mengenai musuh lebih besar. Untuk menghitung

koordinat penembakan salvo, maka diperlukan suatu fungsi untuk menghitung pada *heading* berapa suatu salvo ditembakkan. Algoritma ini melakukan pengecekan pada jarak musuh berukuran kecil terdekat yang memiliki kompleksitas waktu  $O(N)$ .

### 3.3.3. Efisiensi Solusi Greedy Skala Prioritas

Implementasi dari relatif kurang efisien dan jauh lebih sulit diimplementasikan dibandingkan algoritma greedy *offensive* dan *defensive*. Hal ini dikarenakan perlunya dibuat sebuah fungsi untuk masing-masing playerAction seperti fungsi untuk bergerak, fungsi untuk menembakkan salvo dan teleporter, fungsi untuk mengaktifkan shield, dst. Masing-masing fungsi ini harus memberikan keputusan terbaik untuk fitur tersebut bersama dengan skala prioritasnya. Di akhir ronde, hasil keputusan dengan skala prioritas tertinggillah yang akan dipilih. Algoritma ini juga kurang efisien karena harus memanggil seluruh fungsi untuk menentukan keputusan dan prioritas masing-masing action di setiap ronde. Algoritma ini melakukan pengecekan pada setiap objek ( $N$ ) sebanyak banyaknya jenis player action yang dimiliki ( $M$ ) dan memiliki kompleksitas waktu  $O(NM)$ .

### 3.3.4. Efisiensi Solusi Greedy Campuran

Implementasi dari algoritma greedy campuran lebih rumit daripada algoritma *offensive* dan *defensive* akan tetapi lebih sederhana daripada greedy skala prioritas. Implementasi dari solusi ini adalah gabungan dari implementasi solusi pertama dan kedua, dimana algoritma greedy yang bersifat *offensive* dan *defensive* sama-sama digunakan. Pada implementasi ini, terdapat 4 jenis “mode” yang dapat dipilih pada suatu waktu, yaitu mode *eat*, *attack*, *escape*, dan *defend*.

Implementasi dari mode eat dan defend cukup mudah, pemain akan fokus untuk mencari *FOOD* atau *SUPERFOOD* yang berjarak paling dekat dari pemain. Pada mode *defend*, pemain akan mengaktifkan *SHIELD* untuk melindungi pemain dari serangan *SALVO* lawan. Implementasi dari mode attack dan escape relatif lebih sulit. Pada mode *attack*, pemain akan menembak *salvo* jika pemain berada pada suatu konstanta *firing range* dan jika terdapat pemain yang lebih kecil dari kita, maka kita akan mengejar pemain tersebut. Selain itu, mode ini juga mempertimbangkan apakah sebaiknya kita melakukan aksi *teleport* agar dapat memakan lawan dengan melihat situasi pada saat tersebut. Pada mode *escape*, pemain akan berusaha untuk kabur dari pemain lawan jika pemain lawan sedang mendekati pemain. Pada setiap mode yang dipilih, akan dilakukan pengecekan pada setiap objek di dalam map sehingga memiliki kompleksitas waktu  $O(N)$ .

### **3.4. Analisis Efektivitas Secara Teoritis dari Alternatif Solusi Persoalan**

#### **3.4.1. Efektivitas Solusi Greedy Defensive**

Algoritma ini cukup efektif ketika dihadapkan dengan situasi di mana bot lawan sudah relatif lebih besar terlebih dahulu dari bot hasil implementasi. Bot akan menyalakan shield dan melarikan diri ketika dalam bahaya. Akan tetapi kurang efektif karena algoritma ini hanya bergantung kepada *food* ataupun *superfood* untuk menambahkan ukuran bot. Seharusnya, mekanisme *salvo* dan *teleport* juga dimanfaatkan karena merupakan fitur permainan yang dapat membuat bot hasil implementasi memiliki ukuran yang lebih besar.

#### **3.4.2. Efektivitas Solusi Greedy Offensive**

Algoritma ini cukup efektif ketika bot implementasi memiliki ukuran yang lebih besar daripada bot-bot lawan pada permulaan permainan. Akan tetapi, solusi ini memiliki kelemahan karena tidak pernah menghindari serangan musuh untuk bertahan hidup. Alternatif solusi ini efektif dalam memenangkan permainan melawan ReferenceBot, dikarenakan ReferenceBot tidak pernah melakukan tindakan-tindakan agresif seperti menembakkan *SALVO* atau *TELEPORTER* kepada pemain.

Walaupun relatif lebih efektif dibandingkan solusi greedy *defensive*, tetapi alternatif solusi ini masih belum sempurna, karena bot tidak bisa melindungi dirinya sendiri. Jika dipertandingkan dengan bot implementasi lain yang juga bisa menyerang, maka bot memiliki resiko tertelan oleh musuh akibat kehilangan ukuran karena misil dan lain sebagainya.

#### **3.4.3. Efektivitas Solusi Greedy Skala Prioritas**

Jika diimplementasikan dengan benar, algoritma ini berpotensi untuk menjadi sangat efektif. Hal ini karena setiap pilihan action akan diproses dan diberikan bobot prioritas oleh modul-modul tersendiri. Nilai efektivitas paling tinggi dari solusi ini adalah setiap pilihan yang dilakukan oleh bot dapat di *fine-tune* sedemikian rupa dengan mengubah nilai prioritas dari aksi yang ingin dilakukan. Jika terdapat aksi yang ingin diprioritaskan untuk dilakukan, maka nilai prioritas *playerAction* tersebut dapat ditingkatkan. Sebaliknya, jika ada operasi yang tidak diinginkan, maka nilai prioritasnya dapat diturunkan. Kekurangan strategi ini hanyalah kesulitan implementasinya.

#### 3.4.4. Efektivitas Solusi Greedy Campuran

Secara general, solusi ini berhasil secara efektif dalam memenangkan semua permainan melawan ReferenceBot dikarenakan alasan yang serupa dengan alasan yang ada di atas dan pada permainan yang seluruhnya berisi bot dengan solusi ini hampir semuanya berjalan dalam waktu yang cukup lama. Solusi inilah yang akan kami pilih, karena kami merasa bahwa pilihan ini bersifat *offensive* dan *defensive* sekaligus; dan memudahkan kami untuk membagi tugas dikarenakan kode yang dibuat bersifat relatif modular.

### 3.5. Analisis Strategi Greedy yang Dipilih

Kelompok kami memilih untuk mengimplementasikan algoritma greedy campuran yang memiliki berbagai mode. Hal ini karena beberapa faktor, yaitu karena algoritma ini dirasa cukup untuk menangani berbagai situasi yang mungkin dihadapi bot secara umum. Agar strategi ini dapat disebut sebagai algoritma greedy, maka kami menentukan urutan mode yang selalu akan diprioritaskan di dalam implementasi kode. Urutan mode yang akan diimplementasikan ialah sebagai berikut:

1. Defend mode, yaitu mode bertahan ketika ditembak misil
2. Escape mode yaitu mode untuk berlari ketika berada dekat dengan musuh
3. Attack mode, yaitu mode untuk menyerang menggunakan misil, teleporter, dan mengejar musuh.
4. Eat mode, yaitu mode untuk mencari makan dan membesarkan ukuran.

Walaupun algoritma ini mungkin tidak sebaik algoritma greedy dengan skala prioritas dalam *tunability*, algoritma ini relatif lebih mudah diimplementasikan tetapi tidak kalah efektif dari algoritma tersebut. Hasil pengujian dan analisis dari performa strategi greedy pilihan ini dapat dilihat lebih lanjut pada Bab 4.



## BAB 4 IMPLEMENTASI DAN PENGUJIAN

### 4.1. Implementasi Algoritma Greedy

#### 4.1.1. Fungsi Driver

Berikut merupakan pseudocode dari driver yang mengandung logika untuk memilih mode program greedy yang akan dijalankan dan dipilih. Driver ini terletak di fungsi getPlayerAction(). Fungsi driver akan melakukan pengecekan terhadap berbagai kasus dan menentukan kondisi bot saat ini. Misalnya apakah sedang berada dalam keadaan bahaya ataupun jika sedang berada dekat dengan bot yang lebih besar. Mode yang paling sesuai untuk situasi di atas kemudian akan dipilih.

```
// Klasifikasi objek yang terdapat di map ke beberapa kategori
populateObjectData();

// Boolean untuk mengetahui keadaan saat ini
boolean isDanger = closestBotSize > botSize && enemyIsNear;
boolean isStrong = botSize > SIZE_THRESHOLD;

// Menggunakan informasi keadaan untuk memilih mode
// Jika berbahaya, maka dapat menggunakan mode defend ataupun escape

if (isDanger) {
    if (salvoNear) {
        defendMode()
    } else {
        escapeMode()
    }
} // Jika bot cukup besar, maka coba serang bot lain.
else if (isStrong) {
    attackMode()
}
else {
    // Jika belum cukup besar dan tidak dalam bahaya, maka coba cari
    makanan
    eatMode()
}

// Coba lihat apakah ada aksi urgent yang harus dilakukan di ronde
ini
// Misalnya melakukan teleport ketika teleporter sudah berada di
dekat pemain lain
```

```
overrideUrgentActions()
```

#### 4.1.2. Fungsi Escape

Berikut merupakan pseudocode dari fungsi escape yang mengandung logika agar mengantarkan bot menjauhi lawan yang lebih besar darinya. Adapun logika menarik yang diimplementasikan ke dalam program yaitu untuk memasuki kawasan asteroid jika musuh terus mengejar, mengingat bahwa musuh mungkin memiliki logika untuk menghindari bertabrakan dengan objek tersebut.

```
nearestBot = nearPlayerList.get(0);

// Cari jarak dari kita dengan musuh terdekat
var distance = getDistanceBetween(bot, nearestBot);

// Hitung arah dari musuh terdekat
var angle = nearestBot.getCurrentHeading();

// Kalau ada bot dalam range escape dengan ukuran yang lebih besar,
maka hindari dia dan nyalakan afterburner
if (distance <= ESCAPEDISTANCE && nearestBot.getSize() >
bot.getSize()){
    this.playerAction.heading = (angle + 45) % 360;
    toggleAfterBurner()
}

// Jika sudah aman, maka matikan afterburner
if ( (distance > ESCAPEDISTANCE || nearestBot.getSize() <=
bot.getSize()) && this.startafterburner){
    turnOffAfterburner()
}

// Jika masih dalam mode defend, maka coba menuju asteroid
if (distanceAsteroid < distance){
    this.playerAction.heading =
getHeadingBetween(asteroidList.get(0));
}
```

#### 4.1.3. Fungsi Attack

Berikut merupakan pseudocode dari fungsi attack yang mengandung logika untuk melakukan langkah-langkah agresif kepada lawan. Mode ini aktif ketika bot pemain sudah cukup besar dan jika ada bot dengan ukuran yang lebih kecil daripada bot pemain di sekitar.

```
// Jika memiliki salvo dan kita dalam range firing, fire salvo
boolean fireSalvo = this.bot.torpedoSalvoCount > 0 && distance <
FIRINGDISTANCE && nearestBot.getSize() > 15

    if (fireSalvo) {
        FIRESALVO()
    }

    // Kejar player yang lebih kecil dari ukuran kita.
    boolean chasePlayer = this.bot.size > nearestBot.getSize() +
10 && distance < PURSUEDISTANCE && deltaAngle > 5

    if (chasePlayer) {
        MOVE_TOWARDS_PLAYER()
    }

    // Cari apakah kita bisa menyerang player dengan menggunakan
teleport
    boolean teleportPlayerList = getPlayerToTeleport()

    if (teleportPlayerList.size() > 0 && !hasFiredTeleport &&
teleportAvailable) {
        FIRE_TELEPORT()
        hasFiredTeleport = true
    }
```

#### 4.1.4. Fungsi Defend

Berikut merupakan pseudocode dari fungsi defend yang bertanggung jawab untuk menjaga bot dari serangan salvo yang menarget bot pemain. Fungsi ini akan mencoba untuk menghindari salvo dan juga mengaktifkan shield.

```
// Coba menghindari terlebih dahulu
var nearestBot = nearPlayerList.get(0);
int angle = nearestBot.getCurrentHeading();
playerAction.heading = (angle + 60) % 360;
```

```
// Jika kita mempunyai shield, maka aktifkan shield
if (hasShield){
    ACTIVATE_SHIELD()
}
```

#### 4.1.5. Fungsi Eat

Berikut merupakan pseudocode dari fungsi eat yang mengandung logika untuk memilih makanan berupa food maupun superfood terdekat dengan bot. Fungsi ini merupakan fungsi default yang digunakan bot ketika tidak harus melakukan pemanggilan terhadap fungsi lain.

```
// Hitung jarak menuju makanan dan supermakanan terdekat
double foodDistance = getFoodDistance()
double superFoodDistance = getSuperFoodDistance()

if (multiplier * superFoodDistance <= foodDistance){
    headToSuperFood()
} else {
    headToFood()
}
```

## 4.2. Struktur Data yang Digunakan

### 4.2.1. Object Types

Struktur data ini bertanggung jawab untuk menentukan jenis dari setiap objek di dalam permainan. Nilai enum dari object types akan digunakan oleh kelas game object untuk menentukan jenis objek tersebut.

```
public enum ObjectTypes {
    PLAYER(1),
    FOOD(2),
    WORMHOLE(3),
    GASCLOUD(4),
    ASTEROIDFIELD(5),
    TORPEDOSALVO(6),
    SUPERFOOD(7),
    SUPERNOVAPICKUP(8),
    SUPERNOVABOMB(9),
    TELEPORTER(10),
    SHIELD(11);
}
```

```
}
```

#### 4.2.2. Effect Types

Struktur data ini bertanggung jawab untuk menentukan jenis efek yang sedang dialami oleh setiap objek di dalam permainan. Efek bersifat akumulatif, misalnya jika player sedang mengaktifkan shield dan afterburner secara bersamaan, maka nilai statusnya adalah  $16 + 1 = 17$ .

```
public enum EffectTypes {  
    AFTERBURNER(1),  
    ASTEROIDFIELD(2),  
    GASCLOUD(4),  
    SUPERFOOD(8),  
    SHIELD(16);  
}
```

#### 4.2.3. Player Actions

Struktur data ini bertanggung jawab untuk menyimpan nilai dari setiap aksi yang dapat dilakukan oleh player di dalam permainan.

```
public enum PlayerActions {  
    FORWARD(1),  
    STOP(2),  
    STARTAFTERBURNER(3),  
    STOPAFTERBURNER(4),  
    FIRETORPEDOES(5),  
    FIRESUPERNOVA(6),  
    DETONATESUPERNOVA(7),  
    FIRETELEPORT(8),  
    TELEPORT(9),  
    ACTIVATESHIELD(10);  
}
```

#### 4.2.4. Game Object

Struktur data ini menyatakan atribut yang dimiliki oleh setiap objek di dalam game. Objek selain player tidak memiliki nilai effect, torpedo, supernova, teleporter, ataupun shield.

```
public class GameObject {  
    public UUID id;  
    public Integer size;  
    public Integer speed;
```

```
public Integer currentHeading;  
public Position position;  
public ObjectTypes gameObjectType;  
public Integer effects = 0;  
public Integer torpedoSalvoCount = 0;  
public Integer superNovaAvailable = 0;  
public Integer teleporterCount = 0;  
public Integer shieldCount = 0;  
}
```

### 4.3. Pengujian dan Analisis Desain Algoritma Greedy

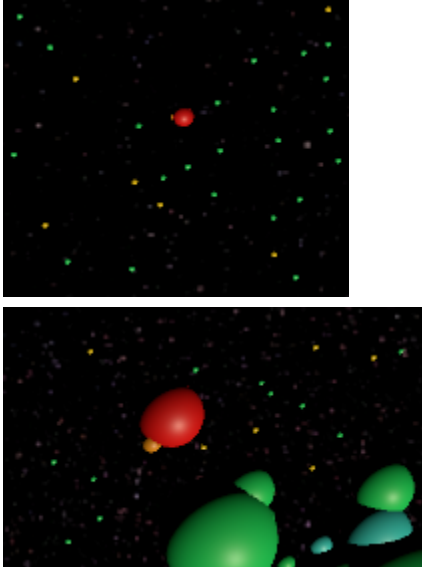
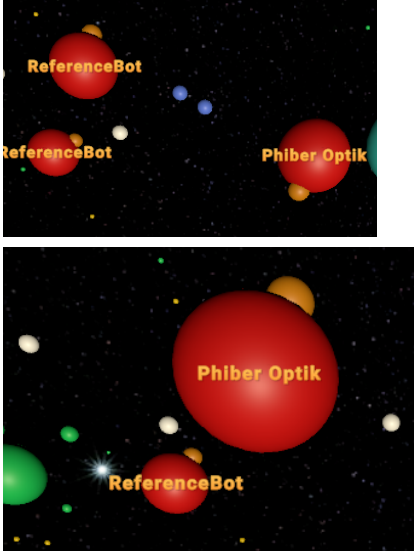
Secara garis besar, pertandingan melawan bot referensi menunjukkan bahwa bot greedy campuran hasil implementasi berhasil memenangkan ~92.5%, yaitu dengan perolehan kemenangan sebanyak 37 dari 40 ronde. Pertandingan dilakukan dengan menggunakan 3 bot referensi dan 1 bot hasil implementasi.

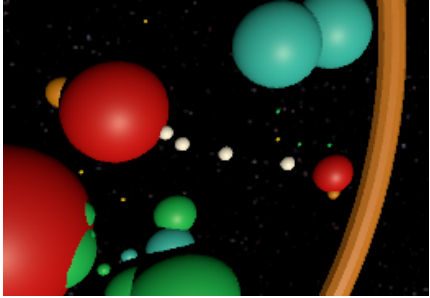
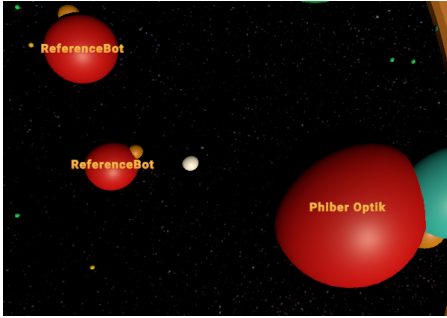
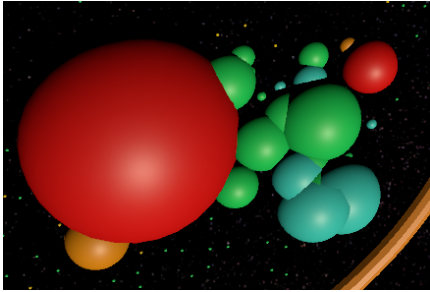
Hasil pengujian menunjukkan beberapa faktor yang mempengaruhi performa bot hasil implementasi. Faktor utama yang mempengaruhi baik/tidaknya algoritma greedy adalah titik spawn bot pada map. Performa buruk cenderung terjadi ketika bot berada dekat dengan banyak objek berbahaya pada map seperti gas cloud, black hole, dsb. Performa bot cenderung dapat berfungsi dengan baik apabila diletakkan di titik yang cenderung kosong dan tidak memiliki banyak objek di sekitarnya.

Situasi yang cenderung membuahkan hasil yang baik adalah ketika bot berada di wilayah yang tidak padat, sehingga mode *eat* dapat dengan mudah membawa ukuran bot menjadi relatif besar. Ketika ukuran bot sudah besar, maka bot akan cenderung bertransisi ke mode-mode tambahan lain seperti mode *attack* ataupun *defend* secara lebih efektif. Sebaliknya, hasil observasi menunjukkan bahwa bot memiliki performa yang buruk ketika tidak mendapat kesempatan untuk memakan banyak makanan di awal permainan. Hal ini menyebabkan bot untuk selalu cenderung berada di mode *eat* saja, karena tidak dapat memenuhi syarat minimum ukuran bot untuk memasuki berbagai mode lain.

Algoritma greedy campuran ini juga kurang baik jika dihadapkan terhadap 2 pilihan mode sekaligus. Secara greedy, algoritma akan mengutamakan mode defend dan escape dibandingkan attack jika dihadapkan dengan musuh yang tidak memiliki ukuran yang berbeda jauh daripada bot hasil implementasi. Padahal, terdapat kemungkinan bahwa bot dapat mengalahkan musuh dengan ukuran yang serupa jika menembakkan missile ataupun salvo. Hal ini karena algoritma greedy hasil implementasi menekankan bahwa akan lebih aman jika bot selalu melarikan diri apabila ukurannya lebih kecil daripada musuh.

Beberapa observasi unik lain melibatkan banyaknya proyektil yang dapat dilepaskan oleh bot. Ketika mencoba untuk menembakan salvo ataupun teleporter, kerap kali diluncurkan  $> 1$  proyektil secara sekaligus. Hal ini karena bot belum sempat melakukan pergantian player action ketika game tick dari game engine telah berganti.

Gambar Uji	Deskripsi
	<p>Mode Eat: Gambar di atas menunjukkan tempat spawn bot pada wilayah yang relatif kosong, sehingga bot dapat bertumbuh dan bertambah size dengan cepat.</p> <p>Gambar di bawah menunjukkan tempat spawn bot pada tempat yang relatif penuh, sehingga bot sulit untuk bertumbuh karena harus menghindarinya.</p>
	<p>Mode Attack: Gambar di samping menunjukkan teleporter (proyektil biru) yang digunakan bot sebagai taktik agresif. Ketika teleporter sudah berada dekat bot lawan yang ukurannya relatif kecil, bot akan mengaktifkan teleport dan memakan bot lawan.</p>

	<p>Mode Attack:</p> <p>Gambar di samping menunjukkan salvo/misil (proyektil putih) yang ditembakkan oleh bot implementasi yang relatif lebih kecil kepada bot lawan. Sebagai akibatnya, bot dapat bertambah besar.</p>
	<p>Mode Escape:</p> <p>Gambar di samping menunjukkan bot implementasi yang sedang bersembunyi di balik asteroid field (biru) ketika dikejar oleh bot lawan yang ukurannya sangat besar.</p>
	<p>Mode Defend:</p> <p>Gambar di samping menunjukkan bot implementasi yang mengaktifkan <i>shield</i> ketika terdapat <i>salvo</i> yang berada di dekat bot.</p>



## **BAB 5 KESIMPULAN DAN SARAN**

### **5.1. Kesimpulan**

Dalam pertandingan melawan bot referensi, bot greedy campuran berhasil memenangkan 92,5% pertandingan dengan perolehan kemenangan 37 dari 40 ronde. Performa bot hasil implementasi dipengaruhi oleh beberapa faktor seperti titik spawn bot pada map. Bot akan berfungsi lebih baik jika ditempatkan di daerah kosong dan tidak memiliki banyak objek berbahaya di sekitarnya. Bot juga akan bertransisi ke mode-mode tambahan seperti mode *attack* atau *defend* jika ukurannya sudah besar. Namun, performa bot akan buruk jika tidak mendapat kesempatan untuk memakan banyak makanan di awal permainan. Algoritma greedy juga kurang baik jika dihadapkan terhadap 2 pilihan mode sekaligus. Bot juga seringkali melepaskan > 1 proyektil secara sekaligus ketika mencoba menembakan salvo atau teleporter.

### **5.2. Saran**

Beberapa saran yang dapat diberikan berdasarkan hasil pengujian yang telah dilakukan yaitu:

1. Pemilihan titik spawn bot pada map dapat mempengaruhi performa bot secara signifikan. Oleh karena itu, perlu dilakukan analisis lebih lanjut untuk menentukan titik spawn bot yang optimal agar bot dapat berfungsi dengan baik
2. Situasi di mana bot berada di wilayah yang tidak padat cenderung membuahkan hasil yang baik. Oleh karena itu, perlu dilakukan analisis lebih lanjut terkait strategi penempatan bot pada map agar bot dapat dengan mudah memakan makanan dan tumbuh secara optimal.
3. Algoritma greedy campuran ini kurang baik jika dihadapkan dengan 2 pilihan mode sekaligus. Oleh karena itu, perlu dilakukan pengembangan algoritma yang lebih canggih dan adaptif agar bot dapat lebih efektif dalam mengambil keputusan terkait mode yang harus dipilih dalam situasi yang kompleks.
4. Dalam pengembangan bot game, perlu memperhatikan permasalahan pergantian player action ketika game tick dari game engine telah berganti. Sebaiknya dilakukan pengujian dan perbaikan secara terus menerus untuk memastikan bot dapat beroperasi dengan baik pada setiap game tick.

Dengan memperhatikan saran-saran tersebut, diharapkan dapat meningkatkan performa dan efektivitas bot game pada permainan agar dapat bersaing secara lebih baik dengan pemain manusia.

## DAFTAR PUSTAKA

Bednorz W, 2008. Advances in greedy algorithms. Russia.

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

Munir, R. 2021. Algoritma Greedy (2021) Bag 2 . Slide kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung

Munir, R. 2021. Algoritma Greedy (2021) Bag 3 . Slide kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung

<https://www.codingame.com/learn/greedy>

[Get Started Galaxio]

[https://docs.google.com/document/d/1Ym2KomFPLIG\\_KAbm3A0bnhw4\\_XQAsOKzpTa70IgnLNU/edit?pli=1](https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa70IgnLNU/edit?pli=1)