

# **IF2211 - Strategi Algoritma**

## **Laporan Tugas Besar 1**



Nama	NIM
Tabitha Permalla	13521111
Nicholas Liem	13521135
Brigita Tri Carolina	13521156

**Institut Teknologi Bandung**  
**Sekolah Teknik Elektro dan Informatika**  
**Tahun Ajaran 2022/2023**

---

# Daftar Isi

<b>1</b>	<b>Deskripsi Masalah</b>	<b>1</b>
<b>2</b>	<b>Landasan Teori</b>	<b>4</b>
2.1	Algoritma Greedy Secara Umum . . . . .	4
2.1.1	Elemen-elemen Algrotima Greedy . . . . .	4
2.2	Cara Kerja Program Secara Umum . . . . .	5
2.2.1	Game Engine . . . . .	5
2.2.2	Game Engine Entelect Galaxio . . . . .	5
2.2.3	Struktur Game Engine . . . . .	6
2.2.4	Pengembangan Bot . . . . .	7
2.2.5	Implementasi Algoritma Greedy ke dalam Bot . . . . .	8
2.2.6	Menjalankan Permainan . . . . .	8
<b>3</b>	<b>Aplikasi Strategi Greedy</b>	<b>9</b>
3.1	Eksplorasi Alternatif Solusi Greedy . . . . .	9
3.1.1	Greedy by Points . . . . .	9
3.1.2	Greedy by Defense . . . . .	10
3.1.3	Greedy by Attack . . . . .	12
3.2	Strategi Algoritma Greedy yang Dipilih dan Alasannya . . . . .	13
<b>4</b>	<b>Implementasi dan Pengujian</b>	<b>15</b>
4.1	Implementasi Algoritma Greedy dalam Pseudocode . . . . .	15
4.2	Struktur Data Program . . . . .	23
4.3	Analisis dan Pengujian Desain Solusi . . . . .	24
<b>5</b>	<b>Kesimpulan dan Saran</b>	<b>29</b>
5.1	Kesimpulan . . . . .	29
5.2	Saran . . . . .	29
5.3	Komentar . . . . .	29
5.4	Refleksi . . . . .	29
<b>6</b>	<b>Daftar Pustaka</b>	<b>30</b>
<b>7</b>	<b>Lampiran</b>	<b>31</b>
7.1	Link Repository GitHub . . . . .	31
7.2	Link Video Demo (YouTube) . . . . .	31

---

## 1 Deskripsi Masalah

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 Galaxio

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) 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 melewati 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. 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 tak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakkannya 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\_AFTERBURNER
  - (d) STOP\_AFTERBURNER
  - (e) FIRE\_TORPEDOES

- 
- (f) FIRE\_SUPERNOVA
  - (g) DETONATE\_SUPERNOVA
  - (h) FIRE\_TELEPORTER
  - (i) 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.

---

## 2 Landasan Teori

### 2.1 Algoritma Greedy Secara Umum

Algoritma greedy adalah salah satu metode pemecahan masalah dengan mengambil keputusan pada setiap langkah berdasarkan solusi terbaik saat ini, tanpa mempertimbangkan dampak jangka panjang dari keputusan tersebut. Ini berarti bahwa algoritma ini akan selalu memilih pilihan terbaik saat ini, tanpa memikirkan apakah pilihan tersebut akan menghasilkan solusi optimal dalam jangka panjang.

Contoh umum dari penggunaan algoritma greedy adalah dalam pemrograman mesin paket (Knapsack problem). Dalam masalah ini, seseorang harus memutuskan apa yang harus dibawa dalam tas mereka berdasarkan nilai dan berat barang-barang yang tersedia. Algoritma greedy akan memilih barang dengan nilai tertinggi per satuan berat, dan memasukkannya ke dalam tas sampai tas tersebut penuh.

Algoritma greedy juga digunakan dalam pemrograman jaringan (Network flow problem), di mana seseorang harus memutuskan jalur mana yang harus diamalkan melalui jaringan agar mencapai tujuannya dengan biaya minimum. Algoritma greedy akan memilih jalur dengan biaya per satuan jarak minimum, dan terus memilih jalur tersebut sampai tujuan tercapai.

Kode Huffman adalah contoh lain dari algoritma greedy. Dalam kompresi data, algoritma ini digunakan untuk menentukan representasi simbol yang paling efisien dalam bentuk bit. Algoritma akan memilih simbol dengan frekuensi tertinggi untuk memperoleh representasi terpendek, dan terus membuat pilihan terbaik sampai semua simbol dalam data terkompresi.

Secara umum, algoritma greedy dapat digunakan dalam berbagai masalah yang membutuhkan pembuatan keputusan step-by-step dengan memilih solusi terbaik pada setiap langkah. Namun, algoritma ini tidak selalu menghasilkan solusi optimal dalam jangka panjang, dan harus digunakan dengan hati-hati untuk memastikan bahwa solusi yang dihasilkan sesuai dengan tujuan yang diinginkan.

#### 2.1.1 Elemen-elemen Algrotima Greedy

Algoritma Greedy umumnya memiliki elemen-elemen sebagai berikut:

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 him-punan solusi (layak atau tidak)
  6. Fungsi objektif :  
memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa:

Algoritma greedy melibatkan pencarian sebuah himpunan bagian,  $S$ , dari him-punan kandidat,  $C$ ; yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu  $S$  menyatakan suatu solusi dan  $S$  dioptimisasi oleh fungsi objektif.

## 2.2 Cara Kerja Program Secara Umum

### 2.2.1 Game Engine

*Game engine* dapat didefinisikan sebagai perangkat lunak dasar dari permainan komputer ataupun *video game*. *Game engine* juga dapat dikatakan sebagai *frame-work* dari pengembangan *video game*. *Game engine* umumnya mencakup berba-gai *library* serta program/komponen pendukung lainnya. Fungsi utama dari *game engine* adalah mempermudah proses pengembangan *video game* serta mengurangi biaya pengembangan. Hal ini dapat terjadi sebab *game engine* membantu pengem-bang agar tidak harus bekerja dari nol. Salah satu *game engine* yang cukup populer dan umum digunakan adalah *Unity*.

### 2.2.2 Game Engine Entelect Galaxio

Dalam permainan *Entelect 2021 - Galaxio*, terdapat beberapa komponen yang diper-lukan agar permainan dapat berjalan:

1. *Engine* - *Engine* bertanggung jawab untuk melaksanakan peraturan-peraturan dalam permainan, dengan mengaplikasikan *command* bot ke *game state* apa-bila *command* tersebut valid
2. *Runner* - *Runnner* bertanggung jawab untuk menjalankan pertandingan antar pemain, memanggil *command* yang sesuai dengan yang diberikan tiap-tiap bot, dan menyerahkannya kepada *engine* untuk dieksekusi
3. *Logger* - *Logger* bertanggung jawab untuk menangkap semua perubahan ter-hadap *game state*, serta pengecualian apapun, dan menuliskannya ke dalam *log file* di akhir permainan
4. *Starter-bots* - *Starter bots* dengan logika yang terbatas dapat digunakan se-bagi titik awal bot yang dibangun. Folder ini juga berisi bot dengan nama *Ref-erenceBot*, yang dapat digunakan untuk menguji bot yang dibangun. Dalam penggerjaan tugas besar ini, bot dibangun dari *Starter bot* bahasa *Java*.
5. *ec-compose* - Folder yang berisi semua berkas (*file*) yang dibutuhkan untuk menjalankan *game* di *docker*.

---

### 2.2.3 Struktur Game Engine

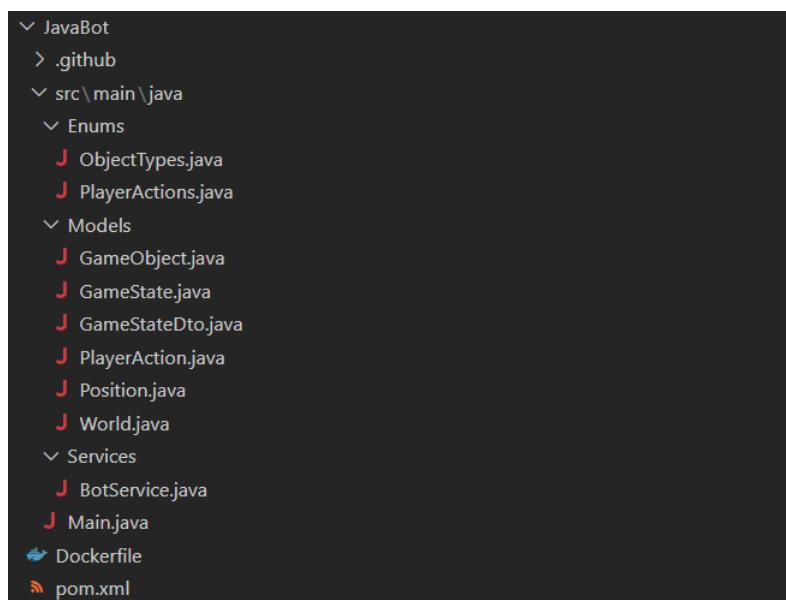
Untuk memulai penggerjaan tugas besar ini, atau secara umum untuk membangun bot Galaxio, diperlukan untuk mengunduh *starter-pack* yang telah disediakan oleh *EntelectChallenge*. *Starter pack* ini dapat diunduh pada tautan <https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>. Dalam tugas besar ini secara khusus, bot akan dibangun dalam bahasa pemrograman *Java*.

*Starter pack* yang disediakan berisi berkas-berkas sebagai berikut:

📁 engine-publish	File folder
📁 logger-publish	File folder
📁 reference-bot-publish	File folder
📁 runner-publish	File folder
📁 starter-bots	File folder
📁 visualiser	File folder
📄 building-a-bot.md	MD File
📄 README.md	MD File
🌈 run	Shell Script

Gambar 2.1 Struktur Folder Starter Pack

Bot yang kami bangun dalam tugas besar ini dibangun dalam *path starter-pack/starter-bots/JavaBot*. Folder tersebut memiliki struktur sebagai berikut:



Gambar 2.2 Struktur Folder JavaBot

Berkaitan dengan *file* yang perlu difokuskan dalam pengembangan bot akan dijelaskan dalam subbab selanjutnya.

---

#### 2.2.4 Pengembangan Bot

Sebelum menerapkan algoritma yang sudah dirancang ke dalam bot, terdapat beberapa hal yang perlu diperlengkapi terlebih dahulu. Hal ini dikarenakan ada beberapa hal yang belum diperbarui di dalam *starter pack* yang disediakan. Sebelum pengembangan bot lebih lanjut, *command* bot perlu dilengkapi dengan menyunting file *PlayerActions.java* di dalam folder *Enums*. Pembaharuan yang perlu dilakukan adalah sebagai berikut:

```
package Enums;

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

    public final Integer value;

    private PlayerActions(Integer value) {
        this.value = value;
    }
}
```

Gambar 2.3 Updated Enums/PlayerActions.java

Setelah memperbarui file *PlayerActions.java* seperti di atas, pengembangan bot dapat mulai dilaksanakan. Apabila kita memperhatikan kode program *Main.java*, terdapat suatu fungsi *computeNextPlayerAction* yang dipanggil.



```
//This is a blocking call
hubConnection.start().subscribe(() -> {
    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());
        }
    });
    hubConnection.stop();
})
```

Gambar 2.4 Function computeNextPlayerAction in Main.java

---

Implementasi fungsi *computeNextPlayerAction* ini dapat ditemukan di dalam *Services/BotService.java*. Oleh karena itu, dalam pengembangan bot ini, pemrogram dapat melakukan pengeditan dalam file *BotService.java* ini. Potongan dari fungsi *computeNextPlayerAction* adalah sebagai berikut:

```
public void computeNextPlayerAction(PlayerAction playerAction) {
    playerAction.action = PlayerActions.FORWARD;
    playerAction.heading = new Random().nextInt(bound: 360);

    if (target == null || target == worldCenter) {
        playerAction.heading = findingNewTarget();
    } else {
        // decide if new target is an object or a player
        if (!gameState.getGameObjects().stream().filter(item -> item
            .isEmpty()) {
    
```

Gambar 2.5 computeNextPlayerAction

### 2.2.5 Implementasi Algoritma Greedy ke dalam Bot

Seperti yang telah disebutkan dalam subbab sebelumnya, pengembangan bot dapat dilakukan dengan mengedit file *BotService.java*. Algoritma yang telah dirancangkan, (dalam kasus tugas besar ini secara khusus, algoritma greedy), dapat diimplementasikan ke dalam fungsi-fungsi di dalam *BotService.java*. Hal ini dapat dilakukan dengan langsung mengedit fungsi *computeNextPlayerAction* ataupun dengan menambahkan fungsi-fungsi ataupun prosedur lainnya.

### 2.2.6 Menjalankan Permainan

Untuk menjalankan permainan, kita perlu terlebih dahulu menjalankan *Game Runner*, kemudian *Game Engine* dan juga *Game Logger*. Terakhir, kita perlu menjalankan bot yang ada sejumlah *BotCount* yang telah ditetapkan, baik bot yang kita kembangkan ataupun *ReferenceBot*. Untuk mempermudah menjalankan permainan, semua perintah untuk menjalankan program-program yang ada dapat dituliskan dalam file *run.bat* (khususnya apabila dijalankan dalam sistem operasi *Windows*).

Setelah program menjalankan permainan sampai selesai, *log* permainan akan tersimpan di folder *logger-publish*. *Log* ini kemudian dapat dijalankan di *visualizer* sehingga mengamati perilaku bot menjadi lebih mudah.

---

### 3 Aplikasi Strategi Greedy

#### 3.1 Eksplorasi Alternatif Solusi Greedy

##### 3.1.1 Greedy by Points

Dalam permainan *Galaxio*, terdapat 2 cara untuk memenangkan permainan. Cara pertama adalah dengan menjadi satu-satunya pemain bertahan di akhir permainan, atau menjadi pemain dengan skor tertinggi pada kasus *tie breaking*. Sesuai dengan peraturan permainan, pemain akan mendapatkan 10 poin saat mengonsumsi kapal pemain lain, 1 poin saat mengonsumsi makanan, dan 1 poin saat melewati *wormhole*. Pada setiap langkah, diprioritaskan untuk memperoleh skor maksimal.

1. Mapping elemen *greedy*
  - (a) Himpunan kandidat: Semua *command* yang tersedia (himpunan *Player-Actions*)
    - i. FORWARD
    - ii. STOP
    - iii. STARTAFTERBURNER
    - iv. STOPAFTERBURNER
    - v. FIRETORPEDOES
    - vi. FIRESUPERNOVA
    - vii. DETONATESUPERNOVA
    - viii. FIRETELEPORT
    - ix. TELEPORT
    - x. ACTIVATESHIELD
  - (b) Himpunan solusi: *Command* yang terpilih
    - i. FORWARD
    - ii. STOP
  - (c) Fungsi solusi: Memeriksa apakah *command* yang dipilih memberikan poin terbanyak dan dapat dilaksanakan
  - (d) Fungsi seleksi: Memilih *command* yang memberikan poin terbanyak
  - (e) Fungsi kelayakan: Memeriksa apakah *command* valid dan dapat dijalankan
  - (f) Fungsi objektif: Memaksimalkan poin yang didapat dalam permainan
2. Analisis Efisiensi Solusi

*Greedy by points* didasarkan pada jumlah poin yang mungkin didapat dalam suatu langkah, dan apakah langkah tersebut mungkin untuk dilaksanakan. Hal ini dilakukan dengan pengecekan jarak bot ke musuh, ke makanan, dan ke *wormhole*. Selain itu, juga dilakukan perbandingan antara ukuran bot dengan ukuran musuh, untuk memastikan apakah bot musuh mungkin untuk dikonsumsi.

---

Dalam pengecekan jarak bot ke objek-objek dalam permainan, secara khusus bot musuh, makanan, dan *wormhole*, dilakukan pengurutan per jenis objek untuk mendapatkan letak objek terdekat. Setelah didapatkan letak objek-objek terdekat, diperiksa apakah memungkinkan untuk mengonsumsi/menggunakan objek tersebut.

Misal  $n$  adalah jumlah objek yang diukur jaraknya dari bot. Maka dibutuhkan  $n$  kali pengukuran. Karena objek-objek ini juga perlu diurutkan, diketahui bahwa kompleksitas algoritma *sort* di *Java* adalah  $O(n \log n)$ . Selain itu, juga dilakukan pengecekan terhadap ukuran musuh, yang dapat diasumsikan sebagai  $n$  langkah. Kemudian, barulah dipilih target yang paling menguntungkan. Sehingga didapat total kompleksitas algoritma adalah  $T(n) = 2n + n \log n = O(n \log n)$ .

### 3. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Permainan berakhir dengan *tie breaking*
- Bot musuh lebih kecil dari ukuran bot sendiri
- Terdapat banyak makanan dan *wormhole* pada *map*

Strategi ini tidak efektif apabila:

- Permainan tidak berakhir dengan *tie breaking*
- Bot musuh lebih besar dari ukuran bot sendiri, dan bot musuh mengejar bot sendiri
- Terdapat lebih banyak *obstacle*, dibandingkan makanan dan *wormhole*

#### 3.1.2 Greedy by Defense

Dalam permainan *Galaxio*, bot dapat mengalami *damage* karena beberapa alasan. Di antaranya adalah bot keluar dari border, bot menghantam asteroid, bot memasuki *gas cloud*, bot dihantam oleh *torpedo* musuh, bot terkena dampak ledakan *supernova*, atau bot bertabrakan/dimakan oleh musuh. Oleh karena itu diperlukan mekanisme *defense* untuk meminimalisir *damage* yang dialami bot.

Terdapat beberapa mekanisme *defense* yang dapat diterapkan dalam permainan ini. Pertama-tama, bot bergerak menghindari objek-objek permainan yang mungkin memberikan *damage*. Menghindar dapat dilakukan dengan memanfaatkan *teleport*, *wormhole*, ataupun sekadar berbelok jalur. Selain menghindar, bot dapat mengaktifkan *shield* untuk melindungi diri.

##### 1. Mapping elemen *greedy*

(a) Himpunan kandidat: Semua *command* yang tersedia (himpunan *Player-Actions*)

- i. FORWARD
- ii. STOP

- 
- iii. STARTAFTERBURNER
  - iv. STOPAFTERBURNER
  - v. FIRETORPEDOES
  - vi. FIRESUPERNOVA
  - vii. DETONATESUPERNOVA
  - viii. FIRETELEPORT
  - ix. TELEPORT
  - x. ACTIVATESHIELD
- (b) Himpunan solusi: *Command* yang terpilih
- i. FORWARD
  - ii. STOP
  - iii. STARTAFTERBURNER
  - iv. STOPAFTERBURNER
  - v. FIRETELEPORT
  - vi. TELEPORT
  - vii. ACTIVATESHIELD
- (c) Fungsi solusi: Memeriksa apakah *command* yang dipilih mengurangi/memimalkan *damage* dan dapat dilaksanakan
- (d) Fungsi seleksi: Memilih *command* yang meminimalkan *damage*
- (e) Fungsi kelayakan: Memeriksa apakah *command* valid dan dapat dijalankan
- (f) Fungsi objektif: Meminimalkan *damage* yang diterima bot

## 2. Analisis Efisiensi Solusi

*Greedy by defense* didasarkan pada *damage* yang mungkin disebabkan oleh berbagai objek dalam permainan. Oleh karena itu, pertama-tama perlu dilakukan pemeriksaan lokasi objek-objek yang mungkin membahayakan bot. Proses ini memakan sebanyak n langkah. Selanjutnya, perlu dilakukan kalkulasi dan seleksi terhadap langkah yang dapat meminimalkan *damage*. Hal ini juga dapat dibulatkan menjadi n langkah. Sehingga kompleksitas total algoritma adalah  $T(n) = n + n = O(n)$ .

## 3. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Terdapat banyak ancaman, baik dari situasi *world* ataupun dari pihak musuh
- Lokasi *wormhole* keluaran dan/atau *teleporter* terdapat di kawasan yang 'aman'

Strategi ini tidak efektif apabila:

- *Defense* tidak sepenuhnya diperlukan karena minimnya ancaman
- Lokasi *wormhole* keluaran dan/atau *teleporter* terdapat di kawasan yang dekat dengan berbagai ancaman lainnya

---

### 3.1.3 Greedy by Attack

Dalam permainan *Galaxio*, cara paling umum untuk menang adalah dengan mengalahkan musuh dan menjadi pemain terakhir yang bertahan. Hal ini dapat dicapai dengan membunuh musuh ataupun mengonsumsi musuh. Hal ini dapat dilakukan dengan menembakkan *torpedo* pada musuh, meledakkan *supernova* di dekat musuh, dan langsung mengonsumsi musuh.

1. Mapping elemen *greedy*
  - (a) Himpunan kandidat: Semua *command* yang tersedia (himpunan *Player-Actions*)
    - i. FORWARD
    - ii. STOP
    - iii. STARTAFTERBURNER
    - iv. STOPAFTERBURNER
    - v. FIRETORPEDOES
    - vi. FIRESUPERNOVA
    - vii. DETONATESUPERNOVA
    - viii. FIRETELEPORT
    - ix. TELEPORT
    - x. ACTIVATESHIELD
  - (b) Himpunan solusi: *Command* yang terpilih
    - i. FORWARD
    - ii. STOP
    - iii. STARTAFTERBURNER
    - iv. STOPAFTERBURNER
    - v. FIRETORPEDOES
    - vi. FIRESUPERNOVA
    - vii. DETONATESUPERNOVA
  - (c) Fungsi solusi: Memeriksa apakah *command* yang dipilih membantu bot mengalahkan musuh dan dapat dilaksanakan
  - (d) Fungsi seleksi: Memilih *command* yang memberi *damage* terbesar pada musuh dan memperbolehkan bot untuk mengonsumsi musuh
  - (e) Fungsi kelayakan: Memeriksa apakah *command* valid dan dapat dijalankan
  - (f) Fungsi objektif: Memaksimalkan *damage* pada musuh untuk kemudian mengonsumsi musuh
2. Analisis Efisiensi Solusi

*Greedy by attack* didasarkan pada *damage* yang mungkin kita sebabkan kepada musuh dalam permainan. Oleh karena itu, pertama-tama perlu dilakukan pemeriksaan lokasi serta ukuran musuh dalam permainan. Apabila musuh lebih kecil daripada bot sendiri, bot dapat langsung mengejar musuh dan

---

mengonsumsi musuh. Namun, apabila musuh lebih besar, maka bot perlu menembakkan *torpedo* ataupun meledakkan *supernova* guna memberi *damage* pada musuh. Seluruh langkah-langkah ini memiliki kompleksitas algoritma total  $T(n) = n = O(n)$ .

### 3. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Sebagian besar musuh berukuran lebih kecil
- Supernova muncul di *world* dan dapat diperoleh dengan cepat
- Tidak terdapat banyak ancaman, baik dari musuh ataupun dari objek-objek lainnya

Strategi ini tidak efektif apabila:

- Sebagian besar musuh berukuran lebih besar
- *Torpedo* tidak ditembakkan dengan akurat, atau *Supernova* tidak diledakkan di waktu/tempat yang tepat
- Terdapat banyak ancaman, baik dari musuh ataupun dari objek-objek lainnya

## 3.2 Strategi Algoritma Greedy yang Dipilih dan Alasannya

Tiap-tiap algoritma *greedy* yang disebutkan di atas memiliki kelebihan dan kekurangannya masing-masing. Oleh karena itu, untuk mengoptimasi bot yang kami bangun, kami menggabungkan dan memilah bagian-bagian dari tiga strategi di atas. Kami menyebut algoritma yang kami terapkan ini sebagai algoritma *greedy by survival*. Tujuan akhir yang ingin dicapai dari algoritma ini adalah menjadi bot yang bertahan sampai akhir permainan.

Dari sisi *greedy by points* dan juga *greedy by attack*, bot akan memprioritaskan untuk mengonsumsi musuh. Ketika bot sudah menetapkan musuh sebagai target, bot akan mengejar dan menembaki musuh, dan kemudian mengonsumsi musuh. Prioritas kedua, khususnya apabila tidak terdapat musuh yang lebih kecil daripada bot, adalah mengonsumsi makanan sebanyak-banyaknya.

Dari sisi *greedy by points* dan juga *greedy by defense*, bot akan memperhitungkan *defense* yang paling menguntungkan. Apabila memungkinkan untuk memasuki *wormhole*, bot akan bergerak menuju *wormhole* terdekat. Selain menjadi sarana untuk menghindar, melewati *wormhole* juga menambah poin. Apabila sulit untuk menghindar, maka bot akan mengaktifkan *shield* untuk meminimalkan *damage*.

Secara umum, algoritma yang kami terapkan bertujuan untuk mempertahankan bot selama mungkin dalam permainan. Tujuan ini diharapkan dapat tercapai dengan jalan mengalahkan musuh sembari melindungi diri.

Strategi ini dipilih karena dinilai bahwa strategi ini telah cukup menyeimbangkan ketiga strategi yang sudah disebutkan sebelumnya. Kami menyadari perlunya memperhatikan keberjalanannya permainan dari berbagai sisi. Oleh karena itu,

---

perlu dilakukan analisis dari berbagai perspektif. Kami menilai algoritma kami sudah cukup mencakup berbagai perspektif.

Dalam perancangan bot kami, alur berpikir kami adalah sebagai berikut:

1. Prioritaskan menyerang dan mengonsumsi musuh
2. Apabila tidak ada musuh yang lebih kecil, prioritaskan mengonsumsi makanan
3. Apabila musuh yang lebih besar mengejar bot, prioritaskan kabur sembari mencari makanan; apabila tidak ada makanan dalam jarak dekat, dan posisi bot lebih dekat ke *wormhole*, maka bergerak menuju *wormhole*
4. Apabila terdapat *torpedo* yang mendekat, prioritaskan menggunakan *shield*
5. Dalam semua situasi, sebisa mungkin menghindari *border*, *gas cloud*, *asteroid* serta objek-objek lain yang mengancam
6. Selain dalam situasi kabur dari musuh, bot akan menghindari *wormhole* (untuk menghindari perpindahan yang tidak perlu)

---

## 4 Implementasi dan Pengujian

### 4.1 Implementasi Algoritma Greedy dalam Pseudocode

Algoritma Greedy yang kami pilih kami implementasikan di dalam kelas BotService. Berikut adalah pseudocode implementasi fungsi-fungsi yang terkandung dalam kelas BotService.

```
/* Setter dan Getter */

BotService()
    // Menginisialisasi BotService
    playerAction <- new PlayerAction()
    gameState <- new GameState()
    attacking <- false
    Position position <- new Position()
    position.setX(0)
    position.setY(0)
    worldCenter <- new GameObject(null, null, null, null, position,
        null, null, null, null, null, null)
    abON <- false
    targeted <- false
    target <- worldCenter
    eating <- false

    setBot(newBot)
    bot <- newBot

    getBot()
        -> bot

    setPlayerAction(newPlayerAction)
        playerAction <- newPlayerAction

    getPlayerAction()
        -> playerAction

    setGameState(newGameState)
        gameState <- newGameState
        updateSelfState()

    getGameState()
        -> gameState

    updateSelfState()
        optionalBot <- gameState.getPlayerGameObjects().stream().filter
            (gameObject -> gameObject.id.equals(bot.id)).findAny()
        optionalBot.ifPresent(bot -> this.bot = bot)

    getPlayerList()
        -> gameState.getPlayerGameObjects().stream().filter(item ->
            item.getObjectType() = ObjectTypes.PLAYER && item.getId() !=
            bot.id).sorted(Comparator.comparing(item -> getDistanceBetween(
                bot, item))).collect(Collectors.toList())
```

---

```

getObjectList()
    -> gameState.getGameObjects().stream().filter(item -> item.
getObjectType() = ot).sorted(Comparator.comparing(item ->
getDistanceBetween(bot, item))).collect(Collectors.toList())

/* Fungsi Utama */
computeNextPlayerAction()
    /* Prosedur akhir untuk menentukan apakah heading ke target
layak. Fungsi akan menentukan juga action akhir dari bot */

    // Inisialisasi awal player action dan heading
    playerAction.action <- PlayerActions.FORWARD
    playerAction.heading <- new Random().nextInt(360)

    // Log current tick
    DISPLAY("Current Tick = " + gameState.world.getCurrentTick())

    // Penanganan utama ketika player near border
    distanceFromCenter <- getDistanceBetween(bot, worldCenter)
    if ((distanceFromCenter + (1.5 * bot.size)) > gameState.world.
    getRadius() * 7 / 9) then
        DISPLAY("NEAR BORDER")
        this.attacking <- false
        this.target <- worldCenter
    end if

    // Mencari new target ketika boolean telah ter-update
    playerAction.heading <- findingNewTarget()

    // Mekanisme Defense Using Shield
    torpedoList <- getObjectList(ObjectTypes.TORPEDOSALVO)
    if (!torpedoList.isEmpty()) then
        nearestTorpedoSalvo <- torpedoList[0]
        if (targeted && getDistanceBetween(bot, nearestTorpedoSalvo
) < 70 && bot.size > 30) then
            playerAction.action <- PlayerActions.ACTIVATESHIELD
            System.out.println("ACTIVATING SHIELD")
        end if
    end if

    playerList <- getPlayerList()
    averagePlayerSize <- 0
    if (playerList != null) then
        averagePlayerSize <- findAverageSize(playerList)
    end if

    if (eating && bot.size > 20 && (bot.size > 40 || bot.size >=
averagePlayerSize) && bot.torpedoSalvoCInteger > 0
        or (eating && getDistanceBetween(bot, playerList.get(0)
) < 50)) then
        playerAction.setHeading(getHeadingBetween(playerList[0]))
        playerAction.action = PlayerActions.FIRETORPEDOES
        DISPLAY("Firing Torpedo")
    end if

```

```

if (attacking && !abON && bot.size > target.getSize() + 10
    && target.getGameObjectType() = ObjectTypes.PLAYER)
then
    playerAction.action <- PlayerActions.STARTAFTERBURNER
    this.abON <- true
    DISPLAY("AfterBurner On")
else if ((!attacking or bot.size < 45
        or (bot.size < target.getSize() + 10 && target.
getGameObjectType() = ObjectTypes.PLAYER)) && abON) {
    playerAction.action <- PlayerActions.STOPAFTERBURNER
    this.abON <- false
    DISPLAY("AfterBurner Off")
else if (attacking && bot.size > 20 && (bot.size > 40 || bot.
size >= averagePlayerSize) &&
        bot.torpedoSalvoCInteger > 0) then
    playerAction.action <- PlayerActions.FIRETORPEDOES
    DISPLAY("Firing Torpedo")
end if

this.playerAction = playerAction

/* Fungsi Pendukung */
findingNewTarget()
    // Mengembalikan heading untuk target baru pada setiap langkah
    // Penanganan null value
    if (not (gameState.getGameObjects().isEmpty()) or not (
gameState.getPlayerGameObjects().isEmpty())) then

        // Menginisialisasi beberapa list yang memiliki tipe objek sama
        playerList = getPlayerList()
        foodList = getObjectList(ObjectTypes.FOOD)
        gasCloudsList = getObjectList(ObjectTypes.GASCLOUD)
        wormHoleList = getObjectList(ObjectTypes.WORMHOLE)
        asteroidList = getObjectList(ObjectTypes.ASTEROIDFIELD)

        // Penanganan null value
        if (!wormHoleList.isEmpty())
            nearestWormhole <- wormHoleList[0]
        else
            nearestWormhole <- null
    end if

    // Penanganan null value
    if (!asteroidList.isEmpty()) then
        nearestAsteroid <- asteroidList[0]
    else
        nearestAsteroid <- null

    // Penanganan null value
    if (!gasCloudsList.isEmpty()) then
        nearestGasCloud <- gasCloudsList[0]
    else

```

```
nearestGasCloud <- null
```

```
// Penanganan null value
if (!foodList.isEmpty()) then
    nearestFood <- foodList[0]
else
    nearestFood <- null
end if

// Log ketika list player empty artinya bot memenangkan pertandingan
if (playerList.isEmpty())
    DISPLAY("YEY MENANG")

// Inisialisasi player terdekat dan beberapa heading yang akan digunakan
nearestPlayer = playerList[0]
headsToNearestPlayer = getHeadingBetween(nearestPlayer)
headsToNearestFood = getHeadingBetween(nearestFood)

// Prioritas dan mencari heading ketika target adalah worldCenter
if (target = worldCenter) then
    this.attacking <- false
    if (nearestFood != null) then
        this.target <- nearestFood
        heading <- getHeadingBetween(nearestFood)
    else
        this.target <- nearestPlayer
        heading <- getHeadingBetween(nearestPlayer)
    end if
else
    if (nearestPlayer.getSize() > bot.size) then
        this.attacking <- false
        this.target <- nearestFood
        this.targeted <- true
        heading <- headsFarFromAttacker(nearestPlayer,
            foodList, wormHoleList)
    else if (nearestPlayer.getSize() + 10 < bot.size) then
        this.attacking <- true
        this.target <- nearestPlayer
        this.targeted <- false
        heading <- headsToNearestPlayer
    else if (nearestFood != null) then
        this.attacking <- false
        this.target <- nearestFood
        this.targeted <- false
        heading <- headsToNearestFood
    else
        this.target <- worldCenter
        heading <- getHeadingBetween(worldCenter)
        this.attacking <- false
        this.targeted <- false
    end if
end if
```



---

```

        distanceTargetFromGasClouds <- getDistanceBetween(
nearestGasCloud, target)
        if (nearestGasCloud != null) then
            if (distanceTargetFromGasClouds < nearestGasCloud.
getSize() + 25 &&
                absolut(getHeadingBetween(nearestGasCloud) -
getHeadingBetween(target)) < 10) then
                    this.attacking <- false
                    heading <- headsFarFromAttacker(nearestGasCloud,
foodList, wormHoleList)
                    -> heading
                end if
            end if

        distanceTargetFromWormhole <- getDistanceBetween(
nearestWormhole, target)
        if (nearestWormhole != null) then
            if (distanceTargetFromWormhole < nearestWormhole.
getSize() + 25 &&
                absolut(getHeadingBetween(nearestWormhole) -
getHeadingBetween(target)) < 10)
                    then
                        this.attacking <- false
                        heading <- headsFarFromAttacker(nearestWormhole,
foodList, wormHoleList)
                        -> heading
                    end if
            end if

        distanceTargetFromAsteroid <- getDistanceBetween(
nearestAsteroid, target)
        if (nearestAsteroid != null) then
            if (distanceTargetFromAsteroid < nearestAsteroid.
getSize() + 25 &&
                absolut(getHeadingBetween(nearestAsteroid) -
getHeadingBetween(target)) < 10) then
                    this.attacking <- false
                    heading <- headsFarFromAttacker(nearestAsteroid,
foodList, wormHoleList)
                    -> heading
                end if
            end if
            -> heading
        end if
    -> getHeadingBetween(worldCenter)

headsFarFromAttacker()
// Menginisiasi langkah untuk menghindar dari attacker
GameObject nearestFood <- foodList[0];
if (nearestFood = null) then
    heading <- headsInverse(enemy)
if (!wormHoleList.isEmpty()) then
    nearestWormHole <- wormHoleList.get(0);

```

---

```

    else
        nearestWormHole <- null

    // Mencari jarak antara satu item dengan item lainnya
    distanceFromEnemy <- getDistanceBetween(bot, enemy)
    foodAndEnemyDistance <- getDistanceBetween(nearestFood, enemy)
    wormHoleAndEnemyDistance <- getDistanceBetween(nearestWormHole,
                                                    enemy)

    // Mencari object makanan yang sesuai dengan parameter untuk
    menjauhi attacker
    for (GameObject food : foodList)
        if (Math.abs(getHeadingBetween(food) - getHeadingBetween(
            enemy)) >= 180
            && getDistanceBetween(food, enemy) > enemy.speed) then
            selectedFood <- food
            break
    end for

    // Pengecekan ulang dan mencari target / heading baru untuk
    menjauhi attacker
i     if (distanceFromEnemy > enemy.speed && foodAndEnemyDistance >
        distanceFromEnemy &&
        selectedFood != null && nearestFood != null) then
        if (distanceFromEnemy > 200) then
            DISPLAY("Ambil makanan terdekat")
            heading <- getHeadingBetween(nearestFood)
            this.eating <- true
        else
            DISPLAY("Ambil makanan yang ada di belakang bot dan
            enemy")
            heading <- getHeadingBetween(selectedFood)
            this.eating <- true
            DISPLAY("EATING")
        end if
    else if (wormHoleAndEnemyDistance > distanceFromEnemy &&
            nearestWormHole != null &&
            enemy.size > bot.size &&
            distanceFromEnemy < 10 &&
            bot.size < nearestWormHole.getSize())
        heading <- getHeadingBetween(nearestWormHole)
        DISPLAY("GOING TO WORMHOLE")
        this.eating <- false
    else if (foodList != null && selectedFood != null)
        this.attacking <- false
        heading <- getHeadingBetween(selectedFood)
        DISPLAY("GOING AWAY FROM ATTACKER")
        this.eating <- false
    else
        heading <- getHeadingBetween(nearestFood)
        this.eating <- true
end if
-> heading

```

---

```

headsInverse(target)
    // Mengembalikan heading yang berlawanan dengan target
    -> toDegrees(Math.atan2(enemy.position.y - bot.position.y,
        enemy.position.x
        bot.position.x));

getDistanceBetween(object1, object2)
    // Penanganan null value
    if (object1 = null or object2 == null) {
        -> 0
    }
    // Mengembalikan jarak antara target1 dan target2
    triangleX <- Math.abs(object1.getPosition().x - object2.
    getPosition().x)
    triangleY <- Math.abs(object1.getPosition().y - object2.
    getPosition().y)
    -> Math.sqrt(triangleX * triangleX + triangleY * triangleY)

getHeadingBetween(otherObject)
    direction <- toDegrees(arctan(otherObject.getPosition().y - bot
    .getPosition().y,
        otherObject.getPosition().x - bot.getPosition().x))
    -> (direction + 360) % 360;

toDegrees(v)
    -> (v * (180 / PI))

findAverageSize(gameObjects)
    if (gameObjects.isEmpty()) then
        -> 0
    totalSize <- 0
    for (gameObject in gameObjects) {
        totalSize <- totalSize + gameObject.size
    }
    -> totalSize / gameObjects.size()

```

---

## 4.2 Struktur Data Program

Pada bot *Galaxio*, secara khusus pada *JavaBot*, pendekatan yang digunakan adalah pendekatan berorientasi objek. Sehingga struktur data dienkapsulasi ke dalam *class*/kelas. Struktur data program bot *Galaxio* secara umum terbagi ke dalam folder *Enums*, *Models*, *Services*, dan file *Main.java*.

### 1. Enums

Bagian ini berisi kelas-kelas yang mengandung konstanta-konstanta dalam permainan, seperti objek-objek dalam permainan dan aksi / *command* bot.

- ObjectTypes

Kelas untuk menyimpan enumerasi tipe-tipe objek dalam permainan, yaitu

- (a) PLAYER,
- (b) FOOD,
- (c) WORMHOLE,
- (d) GASCLOUD,
- (e) ASTEROIDFIELD,
- (f) TORPEDOSALVO,
- (g) SUPERFOOD,
- (h) SUPERNOVAPICKUP,
- (i) SUPERNOVABOMB,
- (j) TELEPORTER,
- (k) SHIELD

- PlayerActions

Kelas untuk menyimpan enumerasi aksi-aksi yang dapat dilakukan oleh bot dalam permainan, yaitu

- (a) FORWARD,
- (b) STOP,
- (c) STARTAFTERBURNER,
- (d) STOPAFTERBURNER,
- (e) FIRETORPEDOES,
- (f) FIRESUPERNOVA,
- (g) DETONATESUPERNOVA,
- (h) FIRETELEPORT,
- (i) TELEPORT,
- (j) ACTIVATESHIELD

### 2. Models

Bagian ini berisi kelas-kelas yang berisi implementasi fungsi-fungsi primitif dari objek-objek dalam game serta aksi-aksi bot. Bagian ini juga berisi kelas-kelas yang mendefinisikan lingkungan permainan.

---

- **GameObject**

Kelas ini berisi *id*, *size*, *speed*, *currentHeading*, *position*, *gameObjectType*, *efInteger*, *torpedoSalvoCInteger*, *supInteger*, *telInteger*, dan *shieInteger* dari objek-objek dalam permainan

- **GameState**

Kelas ini berisi *world*, *gameObjects*, dan *playerGameObjects* dari kondisi permainan dalam satu waktu (*GameState*)

- **GameStateDto**

Kelas ini berisi *world*, *gameObjects*, dan *playerObjects* dari kondisi permainan dalam satu waktu (*GameStateDto*)

- **PlayerAction**

Kelas ini berisi *playerId*, *action*, dan *heading* dari aksi bot (*PlayerAction*)

- **Position**

Kelas ini berisi koordinat (x,y), yang menggambarkan posisi pada peta permainan.

- **World**

Kelas ini berisi *centerPoint*, *radius*, dan *currentTick* yang menggambarkan status permainan.

### 3. Services

- **BotService**

Sesuai pengembangan dan modifikasi yang kami lakukan, kelas ini berisi *bot*, *target*, *worldCenter*, (*boolean*) *attackin*, (*boolean*) *targeted*, *playerAction*, *gameState*, dan (*boolean*) *abON*. Kelas ini berisi fungsi-fungsi yang mengkalkulasikan langkah-langkah yang akan diambil oleh bot. Strategi algoritma greedy yang kami rancangkan dituangkan di dalam kelas ini.

### 4. Main

Kelas ini berisi program utama bot yang memanggil semua fungsi yang diperlukan agar bot dapat berjalan dengan baik.

## 4.3 Analisis dan Pengujian Desain Solusi

Kami melakukan analisis dan pengujian terhadap bot yang kami bangun dengan cara mempertarungkan bot yang kami kembangkan dengan ReferenceBot serta bot kami sendiri. Untuk memudahkan analisis perilaku, kami melakukan pengamatan dengan dua cara.

Cara pertama adalah dengan menganalisis log bot kami selama permainan dijalankan. Dengan memanfaatkan log tersebut, kami dapat mengidentifikasi *error* di dalam program bot kami. Selain itu, kami dapat lebih mudah menganalisis status bot kami serta langkah-langkah yang diambil oleh bot kami dalam setiap *tick*.

Cara kedua adalah dengan menganalisis keberjalanannya permainan melalui *visualizer Galaxio*. Hal ini membantu kami untuk menganalisis kondisi di sekitar bot

---

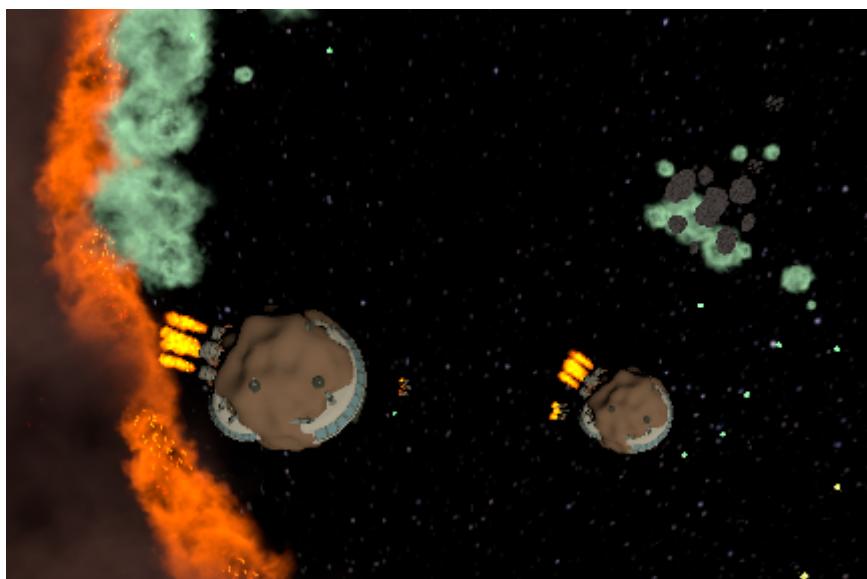
dengan lebih mudah. Selain itu, melakukan pengamatan melalui *visualizer* juga membantu kami memastikan bahwa *command* berhasil dilaksanakan.

Pada tahap awal pengembangan bot ini, kami sudah mulai mengimplementasikan kombinasi dari algoritma *greedy by points*, *greedy by attack*, dan *greedy by defense*. Bot memiliki prioritas untuk mengonsumsi musuh dan juga mencari makan. Selain itu, kami merancang agar bot dapat menyalakan *Afterburner* serta menembakkan *torpedo* ke arah musuh saat sedang menyerang musuh. Berkaitan dengan *defense*, bot dirancang agar bergerak mencari makan yang jauh dari musuh yang mengejar. Apabila tidak ada makanan, bot dirancang untuk bergerak ke arah yang berlawanan dengan bot musuh yang mengejar.

Ketika melakukan analisis dan pengujian, kami menemukan beberapa permasalahan. Pertama-tama, terdapat kesalahan sintaks pada *command FIRETORPEDOES* dan *STARTAFTERBURNER*. Kesalahan ini dapat kami ketahui setelah mengamati log bot serta *visualizer*. Dari log, seharusnya bot sudah memasuki *if block* yang memanggil *command*. Akan tetapi, ketika memeriksa di *visualizer*, kami menemukan bahwa *command* tersebut belum berhasil dipanggil.

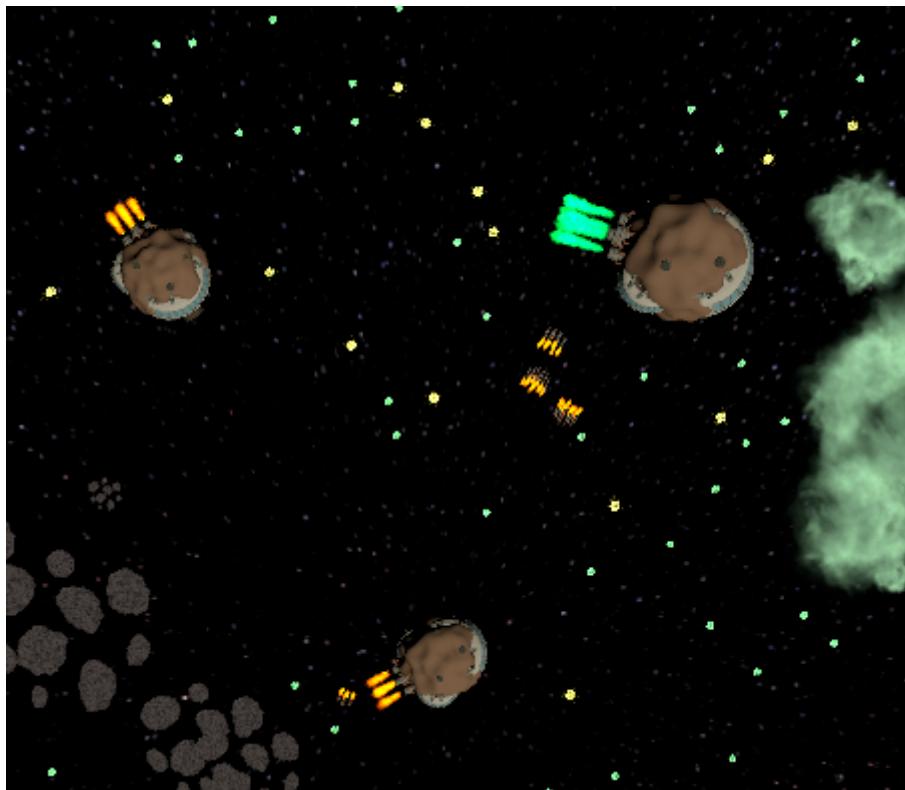
```
AfterBurner On  
Current Tick = 147  
ATTACKING  
EATING  
DIKEJAR MUSUH  
AfterBurner Off
```

Gambar 4.1 Contoh Log Afterburner On



Gambar 4.2 Bot menyerang namun afterburner tidak menyala pada visualizer

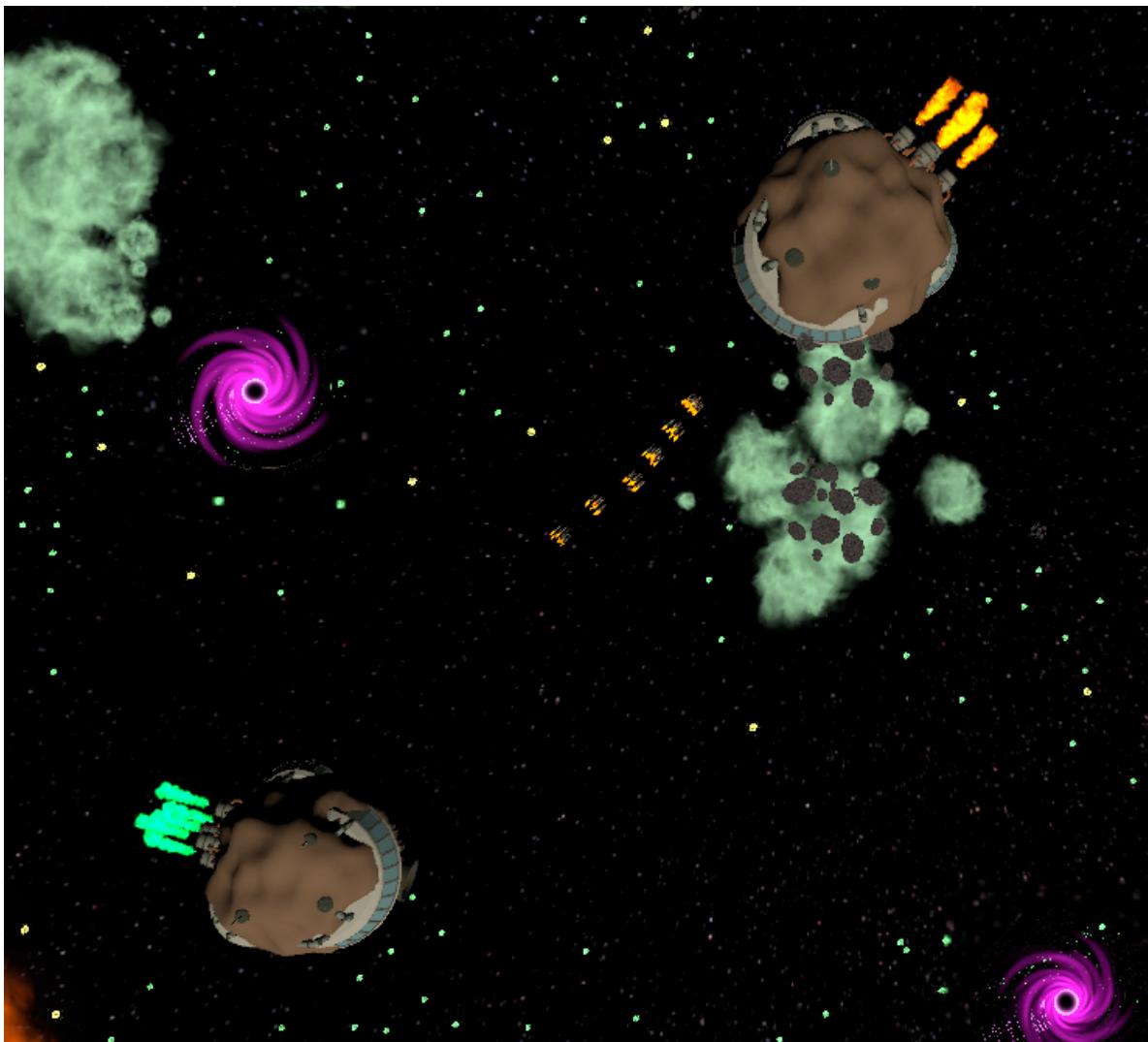
Pada tahap selanjutnya, kami memperbaiki *command FIRETORPEDOES* dan *STARTAFTERBURNER*. Selain itu, kami melakukan modifikasi pada *conditional* pemanggilan kedua *command* ini dengan tujuan untuk mengoptimasi penyerangan. Di saat yang sama, kami juga membatasi pemanggilan kedua *command* ini agar tidak menyebabkan kerugian yang berlebihan pada diri sendiri.



Gambar 4.3 Bot Menggunakan Afterburner

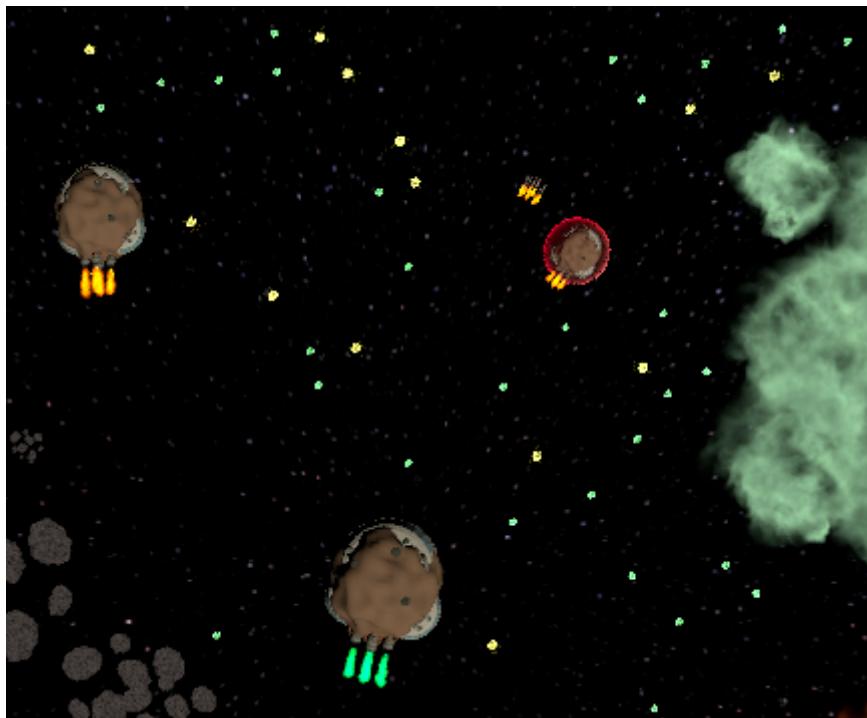
Selain itu, kami juga berusaha mengoptimasi *defense* bot dengan cara menarahkan bot ke *wormhole* saat sedang diserang. Bot diarahkan ke *wormhole* ketika tidak ada makanan yang cukup dekat dan memenuhi parameter.

Dalam pengujian pada tahap ini, kami menemukan bahwa fungsi-fungsi *attacking* sudah cukup optimal. Akan tetapi, terkait *defense*, kami menemukan masih terdapat sejumlah permasalahan. Salah satunya adalah ketika bot sudah berhasil memasuki *wormhole*, bot malah bergerak kembali ke *wormhole* keluaran. Sehingga bot malah berpindah secara bolak-balik, dan kami menemukan bahwa hal ini justru akan merugikan bot.



Gambar 4.4 Bot firing torpedoes tidak hanya pada kondisi menyerang musuh

Oleh karena itu, kami melanjutkan pengembangan bot kami dengan berfokus pada elemen-elemen *defense* dari bot. Kami berfokus pada mekanisme menghindar dari objek-objek yang merugikan bot. Pertama-tama kami berfokus agar bot menghindari *gas cloud* dan *border*. Selanjutnya kami juga melakukan pemanggilan *command ACTIVATESHIELD* guna melindungi bot dari torpedo yang ditembakkan oleh musuh. Terakhir, kami juga menyadari bahwa kami harus menghindari asteroid serta *wormhole*, khususnya pada saat bot tidak sedang dikejar musuh.



Gambar 4.5 Bot Memakai Shield

Setelah melakukan berbagai tahap pengujian, kami menemukan bahwa kami perlu melakukan optimasi pada algortima *greedy by attack*. Kami memutuskan untuk mengubah parameter pemanggilan *command FIRETORPEDOES*. Sehingga *command* dipanggil bukan hanya saat menyerang/mengejar musuh, tetapi juga saat bot dikejar musuh.

Terakhir, kami melakukan optimasi secara menyeluruh. Kami melakukan modifikasi terhadap *heading* agar lebih akurat. Selain itu, kami juga menambahkan sejumlah *boolean* untuk membantu menentukan langkah yang akan diambil.

---

## 5 Kesimpulan dan Saran

### 5.1 Kesimpulan

Dari tugas besar ini, kami berhasil mengembangkan sebuah bot untuk permainan *Galaxio* dengan memanfaatkan pendekatan algoritma greedy. Dalam implementasinya, kami tidak berpatok pada satu algoritma greedy saja. Kami belajar untuk mengoptimasi program kami dengan mengolaborasikan berbagai strategi algoritma greedy yang ada. Tautan yang berisi *source code* tugas ini serta video penjelasan kami dapat dilihat pada bagian Lampiran.

### 5.2 Saran

Saran-saran untuk kelompok kami dalam penggerjaan Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2022/2023 adalah sebagai berikut:

1. Kode program yang digunakan di dalam tugas besar ini dapat dikembangkan lebih lagi, baik dari sisi strategi algoritma, modularitas, dan lain sebagainya
2. Strategi algoritma greedy yang kami implementasikan mungkin belum menjadi strategi yang terbaik, oleh karena itu perlu dilakukan eksplorasi dan pengembangan lebih lanjut
3. Penentuan alur berpikir seharusnya ditetapkan di awal, supaya alur kerja lebih baik dan tidak 'bolak-balik'
4. Perlunya semangat dan motivasi yang konsisten dalam penggerjaan tugas, sehingga beban tugas tidak tertumpuk di akhir

### 5.3 Komentar

Dalam penggerjaan tugas besar ini, kami belajar untuk bekerja dalam *framework* yang telah disediakan. Kami menemukan bahwa kami masih kesulitan dan merasa kurang nyaman bekerja dengan program yang telah ditulis oleh orang lain. Namun, kami menyadari bahwa hal ini perlu terus dilatih, sebab akan berguna bagi kami di masa yang akan datang.

### 5.4 Refleksi

Dari tugas besar ini, kami belajar, pertama-tama, untuk mengabstraksikan permasalahan serta merancang suatu strategi algoritma greedy untuk menyelesaikan permasalahan tersebut. Kami juga belajar untuk bekerja mengikuti *framework* yang disediakan. Selain itu, berkaitan dengan bahasa pemrograman yang kami gunakan, bahasa *Java*, kami bukan hanya belajar lebih banyak terkait bahasa *Java*, tetapi kami juga belajar untuk berpikir dalam paradigma berorientasi objek, walaupun masih sangat terbatas. Di sisi lain, kami juga belajar untuk mengelola waktu serta tanggung jawab yang kami miliki. Kami juga belajar lebih lagi untuk bekerja sama dan menjaga komunikasi yang baik. Kami belajar untuk saling melengkapi sehingga dapat menyelesaikan tugas dengan baik.

---

## 6 Daftar Pustaka

- EntelectChallenge. *EntelectChallenge/2021-Galaxio*. 2021. URL: <https://github.com/EntelectChallenge/2021-Galaxio> (visited on 02/16/2023).
- Munir, Rinaldi. *Algoritma Greedy (Bagian 1)*. 2023. URL: [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) (visited on 02/16/2023).
- Wikipedia. *Game Engine*. URL: [https://en.wikipedia.org/wiki/Game\\_engine](https://en.wikipedia.org/wiki/Game_engine) (visited on 02/16/2023).

---

## **7 Lampiran**

### **7.1 Link Repository GitHub**

Repo GitHub [\[Click Me!\]](#) or [https://github.com/NicholasLiem/Tubes1\\_Algonauts](https://github.com/NicholasLiem/Tubes1_Algonauts)

### **7.2 Link Video Demo (YouTube)**

Video Demo (YouTube) [\[Click Me!\]](#) or <https://youtu.be/YXGTKDhJs6g>