

Tugas Besar 1 IF2211 Strategi Algoritma  
Semester II tahun 2022/2023



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

Disusun oleh:

Farizki Kurniawan	13521082
I Putu Bakta Hari Sudewa	13521150
Dewana Gustavus Haraka Otang	13521173

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2023**

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB I</b>	
<b>DESKRIPSI TUGAS</b>	<b>2</b>
<b>BAB II</b>	
<b>LANDASAN TEORI</b>	<b>3</b>
2.1. Algoritma Greedy	3
2.2. Galaxio	4
2.3. Cara Menjalankan Game dan Memodifikasi Bot Permainan Galaxio	6
<b>BAB III</b>	
<b>APLIKASI STRATEGI GREEDY</b>	<b>9</b>
3.1 Mapping Persoalan Permainan Galaxio	9
3.2 Eksplorasi Alternatif Solusi Greedy pada Permainan Galaxio	10
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy	12
3.4 Strategi Greedy yang Dipilih	14
<b>BAB IV</b>	
<b>IMPLEMENTASI DAN PENGUJIAN</b>	<b>15</b>
4.1 Implementasi Algoritma pada Program Bot	15
4.2 Struktur Data	24
4.3 Analisis Desain Solusi pada Setiap Pengujian	27
<b>BAB V</b>	
<b>PENUTUP</b>	<b>30</b>
5.1 Kesimpulan	30
5.2 Saran	30
5.3 Komentar dan Refleksi	30
<b>DAFTAR REFERENSI</b>	<b>31</b>
<b>LAMPIRAN</b>	<b>32</b>

## **BAB I**

### **DESKRIPSI TUGAS**

Galaxio adalah sebuah *game battle royale* yang mempertandingkan bot kapal pemain dengan beberapa bot kapal yang lain. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal tetap hidup hingga akhir permainan. Pemain harus mengimplementasikan strategi tertentu pada bot untuk menentukan keputusan yang diambil bot pada saat jalannya permainan agar bot tersebut dapat memenangkan pertandingan.

Tugas yang perlu dilakukan adalah untuk mengimplementasikan bot kapal pada permainan Galaxio dengan tujuan memenangkan permainan menggunakan strategi *greedy* pada implementasi algoritmanyanya. Strategi *greedy* yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menghindari objek yang dapat mengurangi ukuran kapal.

Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi *greedy* untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

## BAB II

### LANDASAN TEORI

#### 2.1. Algoritma *Greedy*

Algoritma *greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi hanya terdiri dari dua jenis, yaitu maksimasi (*maximation*) dan minimasi (*minimization*). Algoritma *greedy* memiliki prinsip untuk mendapatkan langkah “terbaik” yang bisa didapat untuk saat ini. Ibaratnya seperti orang amnesia yang mendaki gunung berkabut, ia tidak dapat mengingat apa yang telah dilaluinya dan jarak pandangnya terbatas (tidak dapat melihat jauh kedepan).

Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*). Pada setiap langkah, terdapat banyak pilihan yang perlu dievaluasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang “terbaik” dalam menentukan pilihan. Tidak bisa mundur lagi (kembali) ke langkah sebelumnya. Pada setiap langkah, dipilih suatu optimum lokal dengan harapan langkah-langkah selanjutnya akan mengarah ke solusi optimum global.

Algoritma *greedy* memiliki beberapa elemen, yaitu

- **Himpunan kandidat,  $C$**

Berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb).

- **Himpunan solusi,  $S$**

Berisi kandidat yang sudah dipilih.

- **Fungsi solusi**

Menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.

- **Fungsi seleksi (*selection function*)**

Memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.

- **Fungsi kelayakan (*feasible*)**

Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)

- **Fungsi objektif**

Memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen tersebut, maka dapat disimpulkan algoritma *greedy* melibatkan pencarian sebuah himpunan bagian,  $S$ , dari himpunan kandidat,  $C$ ; yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu  $S$  menyatakan suatu solusi dan  $S$  dioptimasi oleh fungsi objektif.

## 2.2. Galaxio

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

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer  $x,y$  yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu Players, Food, Wormholes, Gas Clouds, Asteroid Fields. Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan  $x$ . Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. Heading dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek afterburner akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat  $x,y$  dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal player. Apabila player mengkonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan

mengcil 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 mengcil 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 IF2211 Strategi Algoritma - Tugas Besar 1 3 Salvo akan mengcil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Berikut jenis-jenis dari command yang ada dalam permainan:
  - a. FORWARD
  - b. STOP
  - c. START\_AFTERSHOCK
  - d. STOP\_AFTERSHOCK

- e. FIRE\_TORPEDOES
  - f. FIRE\_SUPERNOVA
  - g. DETONATE\_SUPERNOVA
  - h. FIRE\_TELEPORTER
  - i. TELEPORTUSE\_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.

### 2.3. Cara Menjalankan Game dan Memodifikasi Bot Permainan Galaxio

Dalam Tugas Besar Strategi Algoritma ini, digunakan bahasa pemrograman Java, jadi *bot* yang akan dimodifikasi adalah *JavaBot* yang berbasis bahasa pemrograman Java. Pada *starter-pack* yang diberikan disediakan *starter-bots* sebagai titik awal dari pengembangan *bot*. *Starter-bots* sudah dibekali kode yang dibutuhkan oleh *bot* untuk melakukan koneksi dengan *runner* sehingga modifikasi *bot* yang akan dilakukan dapat difokuskan kepada strategi apa yang ingin digunakan.

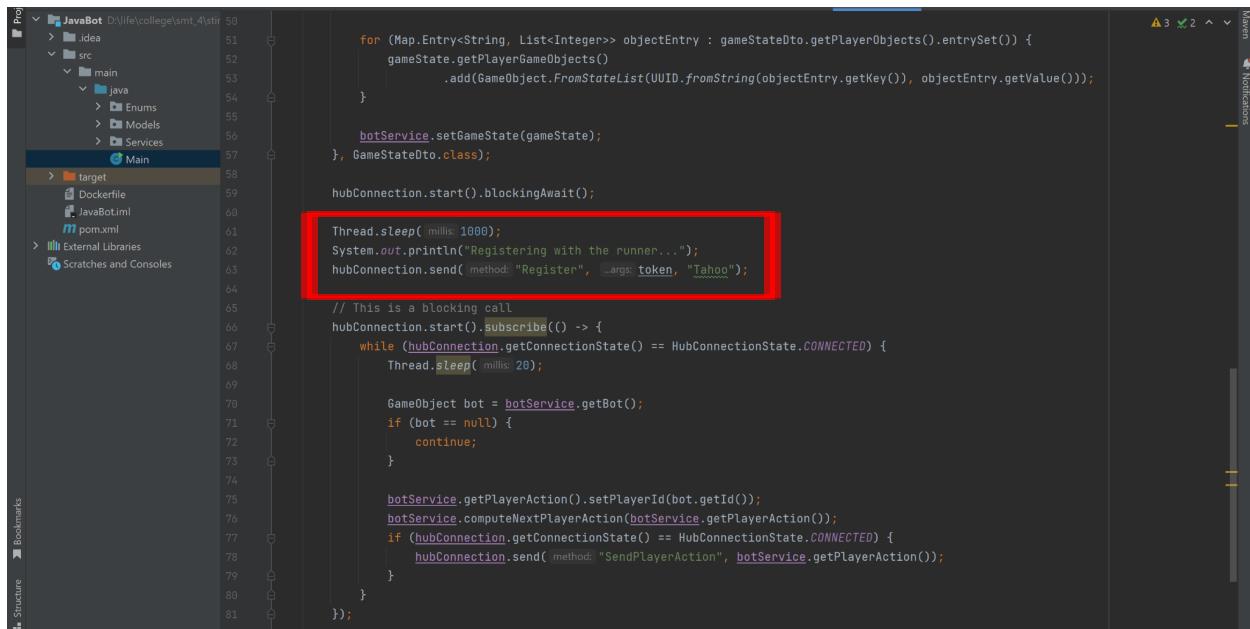
Sebelum melakukan modifikasi *bot*, akan dijelaskan secara garis besar terkait cara kerja program Galaxio, yaitu

- Runner –saat dijalankan– akan meng-host sebuah *match* pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
- Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
- Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
- Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar *match* dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut *BotCount* yang dimiliki file JSON *"appsettings.json"*. File tersebut terdapat di dalam folder *"runner-publish"* dan *"engine-publish"*.
- Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
- Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah *RecieveGameState* karena memberikan status game.
- Bot juga mengirim event kepada runner yang berisi aksi bot.
- Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi *match*.

Berikutnya, game Galaxio dapat dijalankan dengan cara sebagai berikut pada *Windows*,

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON "appsettings.json" dalam folder "runner-publish" dan "engine-publish"
2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah "dotnet GameRunner.dll"
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah "dotnet Engine.dll"
6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah "dotnet Logger.dll"
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON "GameStateLog\_{Timestamp}" dalam folder "logger-publish". Kedua file tersebut diantaranya *GameComplete* (hasil akhir dari permainan) dan proses dalam permainan tersebut.

Untuk melakukan modifikasi *bot*, langkah pertama yang "disarankan" untuk dilakukan adalah menuju ke file *Main.java*, yang terletak pada direktori `/starter-pack/starter-bots/JavaBot/src/main/java /Main.java`, kemudian pada baris 63 terdapat string "Coffee Bot" yang dapat diganti dengan *string* lain, *string* ini merepresentasikan nama *bot* yang akan dimodifikasi.



```
for (Map.Entry<String, List<Integer>> objectEntry : gameStateDto.getPlayerObjects().entrySet()) {
    gameState.getPlayerGameObjects()
        .add(GameObject.FromStateList(UUID.fromString(objectEntry.getKey()), objectEntry.getValue()));
}

botService.setGameState(gameState);
}, GameStateDto.class);

hubConnection.start().blockUntilConnected();

Thread.sleep( 1000 );
System.out.println("Registering with the runner...");
hubConnection.send( method: "Register", args: token, "Tahoo" );

// This is a blocking call
hubConnection.start().subscribe() -> {
    while (hubConnection.getConnectionState() == HubConnectionState.CONNECTED) {
        Thread.sleep( millis: 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( method: "SendPlayerAction", botService.getPlayerAction());
}
}
```

**Gambar 2.3.1.** Modifikasi nama *bot* pada file *Main.java*. Pada gambar ini, *bot* diubah namanya menjadi "Tahoo"

Langkah berikutnya adalah melakukan implementasi strategi yang diinginkan. Strategi diimplementasikan pada file *BotService.java*, di dalam fungsi *computeNextPlayerAction* yang terletak pada direktori `../JavaBot/src/main/java/Services/BotService.java`. Implementasi strategi dapat dilakukan di dalam fungsi tersebut, atau dapat juga dengan membuat suatu kelas baru.

Jika ingin melakukan pengujian atau menjalankan *bot*, langkah selanjutnya adalah melakukan *build* pada *source code bot* menjadi suatu file *.jar*. Proses *build* ini dapat dilakukan dengan menggunakan *maven* atau melalui *build tool* pada *intelliJ*. Kemudian setelah build berhasil, *bot* dapat dijalankan dengan menggunakan perintah `java -jar path_menuju_file_jar_bot` yang dapat dijalankan langsung melalui terminal atau diletakkan pada file *run script (.bat atau .sh)*.

## **BAB III**

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

#### **3.1 Mapping Persoalan Permainan Galaxio**

Strategi dari penyelesaian persoalan permainan Galaxio dapat dimulai dari melakukan *mapping* pada komponen-komponen yang ada. Komponen pada permainan ini memiliki *mapping* sebagai berikut.

1. Himpunan Kandidat, C, adalah himpunan yang berisi kandidat yang akan dipilih pada setiap langkah. Dalam permainan Galaxio, himpunan ini berisi jenis *command* yang dapat dilakukan oleh bot dalam suatu *tick*, dimana beberapa *command* juga memerlukan informasi *heading*, yaitu informasi yang menunjukkan arah. Seluruh anggota himpunan kandidat meliputi:
  - FORWARD, beserta *heading*
  - STOP
  - START\_AFTERCLOUD
  - STOP\_AFTERCLOUD
  - FIRE\_TORPEDOES, beserta *heading*
  - FIRE\_SUPERNOVA, beserta *heading*
  - DETONATE\_SUPERNOVA
  - FIRE\_TELEPORTER, beserta *heading*
  - TELEPORT
  - USE\_SHIELD
2. Himpunan Solusi, S, adalah himpunan yang berisi kandidat yang sudah dipilih. Dalam permainan Galaxio, himpunan ini ialah suatu *command* beserta informasi *heading* bila diperlukan yang dipilih dari himpunan kandidat.
3. Fungsi Solusi adalah fungsi yang menentukan apakah himpunan kandidat yang dimiliki sudah memberikan solusi. Dalam kasus ini, fungsi solusi adalah fungsi yang mengecek apakah *command* telah diberikan beserta informasinya.
4. Fungsi Seleksi adalah fungsi yang memilih kandidat berdasarkan strategi *greedy* yang ditentukan. Dalam kasus ini, fungsi seleksi adalah sebuah fungsi yang menentukan *command* mana yang sebaiknya dipakai berdasarkan informasi yang dimiliki, dimana

*command* tersebut diasumsikan dapat memaksimalkan peluang bot untuk memenangkan pertandingan.

5. Fungsi Kelayakan adalah fungsi yang memeriksa apakah kandidat yang dipilih layak dimasukkan sebagai solusi. Dalam kasus ini, fungsi kelayakan adalah fungsi yang mengecek apakah *command* yang telah diberikan merupakan *command* yang valid dan apakah *command* tersebut dapat dijalankan dalam kondisi bot pada *tick* tersebut. Misalnya, jika *command* merupakan FIRE\_TORPEDOES, maka fungsi harus mengecek apakah bot memiliki torpedo untuk ditembakkan.
6. Fungsi Objektif adalah fungsi yang berjalan dengan cara memaksimumkan ataupun meminimumkan suatu hal. Dalam kasus ini, fungsi objektif secara keseluruhan adalah untuk memaksimumkan waktu bertahan bot selama berada di map agar dapat memenangkan pertandingan.

## 3.2 Eksplorasi Alternatif Solusi *Greedy* pada Permainan Galaxio

### 3.2.1 *Greedy by Distance*

Pada alternatif ini, bot akan fokus kepada objek-objek yang berjarak paling dekat dengan bot dan menentukan aksinya berdasarkan jenis benda tersebut. Bot akan memiliki fokus pada command FORWARD, dan USE\_SHIELD untuk menanggapi objek-objek yang berada di dekatnya. Apabila objek yang paling dekat dengan bot menguntungkan bagi bot, seperti makanan dan recharge supernova, maka bot akan bergerak menuju objek tersebut dengan FORWARD. Apabila objek yang paling dekat dengan bot merugikan bagi bot, seperti *gas cloud* dan torpedo, maka bot akan menghindar dari objek tersebut dengan FORWARD. Apabila objek terdekat merupakan torpedo dan hendak mengenai bot, maka bot dapat memanggil *command* USE\_SHIELD.

### 3.2.2 *Greedy by Virtual Point*

Pada alternatif ini, bot akan menyediakan beberapa *virtual point* untuk setiap arah yang bisa ia datangi (360 derajat). Bot akan memiliki fokus pada command FORWARD untuk bergerak menuju titik yang ingin dituju. Bot akan bergerak menuju titik yang memberikan paling banyak ‘keuntungan’ jika bot bergerak ke arah tersebut. Keuntungan ini dihitung secara heuristik dengan prinsip membandingkan jarak antara titik tersebut dan objek yang menguntungkan serta objek

yang merugikan. Semakin dekat dengan objek yang menguntungkan, maka keuntungan semakin besar dan semakin dekat dengan objek yang merugikan, maka keuntungan semakin kecil.

### **3.2.3 Greedy by Attacking Enemies**

Pada alternatif ini, bot akan fokus dalam menyerang objek yang bertipe *player*. Bot ini memiliki fokus pada FIRE\_TORPEDOES, FIRE\_SUPERNOVA, DETONATE\_SUPERNOVA, FIRE\_TELEPORTER, dan TELEPORT. Bot akan mencari lawan-lawan terdekat yang bisa ditembak menggunakan FIRE\_TORPEDOES dan mengurangi ukuran atau bahkan mengeliminasi mereka. Selain itu, bot akan mencari lawan-lawan jauh yang dapat diserang menggunakan SUPERNOVA ataupun TELEPORTER. Dalam menembakkan supernova, bot harus memerhatikan agar ledakan dari supernova tidak akan mengenai bot secara langsung. Lalu, dalam menembakkan teleporter, bot harus memastikan bahwa saat ia melakukan teleport dan berkurang ukuran, bot dapat langsung memangsa bot lain di saat itu juga.

### **3.2.4 Greedy by Avoiding**

Pada alternatif ini, bot akan berfokus pada keputusan untuk menghindari objek-objek yang dapat membahayakan bot. Idealnya, bot tidak mementingkan ukuran bot untuk memaksimalkan kecepatan yang dimiliki agar dapat menghindar dengan lebih baik. Bot ini memiliki fokus pada FORWARD, STOP, dan SHIELD. Bot akan terus menghindari objek-objek yang membahayakan di sekitarnya dengan FORWARD. Saat bot sudah berada di tempat yang dirasa ‘aman’, dimana tidak terdapat objek berbahaya di sekitarnya pada jarak konstan tertentu, maka ia akan STOP. Apabila bot ditarget dengan torpedo, maka bot akan menggunakan SHIELD jika memiliki ukuran yang cukup dan melarikan diri jika tidak.

### **3.2.5 Greedy by Eating**

Pada alternatif ini, bot akan berfokus pada objek-objek yang dapat dimakan oleh bot, yaitu food, superfood, dan players. Bot ini memiliki fokus pada *command* FORWARD, START\_AFTERSHOCK, dan STOP\_AFTERSHOCK. Bot akan berusaha untuk mengosongkan makanan pada map secepat mungkin dengan menargetkan makanan-makanan yang berada di dekatnya. Idealnya, bot akan memiliki ukuran yang cukup untuk menyalakan

afterburner dan bergerak dengan kecepatan yang tinggi. Hal ini akan membantu bot untuk mengosongkan map dan memangsa bot-bot lain.

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

#### **3.3.1 Alternatif Solusi 1**

Alternatif metode *greedy* ini memiliki tujuan utama untuk memaksimalkan keuntungan yang bisa didapatkan bot serta meminimalisir kerugian yang dialami. Kelebihan dari solusi ini ialah aksi bot yang *versatile* dan dapat melakukan aksi sesuai dengan kondisi yang ada, dimana bot ini dapat menanggapi secara positif apabila terdapat objek yang menguntungkan di dekatnya, dan dapat melarikan diri apabila ia ditarget ataupun berada di dekat objek-objek yang dapat membahayakan dirinya. Adapun kelemahan dari solusi ini ialah kurang agresifnya bot dalam menghadapi musuh dan juga ketidakmampuan bot dalam memilih ‘segmen’ yang menguntungkan. Misalnya, suatu segmen A memiliki lebih banyak makanan dibanding segmen B, tetapi terdapat makanan di B yang merupakan objek terdekat dengan bot. Maka bot akan bergerak menuju B meskipun segmen A dapat menawarkan keuntungan yang lebih baik pada bot.

#### **3.3.2 Alternatif Solusi 2**

Sama dengan alternatif solusi 1, alternatif metode *greedy* ini memiliki tujuan utama untuk memaksimalkan keuntungan yang bisa didapatkan bot serta meminimalisir kerugian yang dialami. Kelebihan dari solusi ini ialah bot memiliki kemampuan untuk memilih segmen yang menguntungkan, karena arah yang dipilih oleh bot akan dinilai berdasarkan objek-objek yang berada di sekitarnya dan bukan hanya satu objek saja. Adapun kelemahan dari solusi ini adalah dapatnya terjadi kerancuan dalam menentukan arah untuk bergerak. Misalnya, pada *tick* pertama bot diarahkan menuju ke arah segmen A tetapi pada *tick* berikutnya muncul/hilang beberapa objek di map yang dapat mempengaruhi penilaian bot terhadap lokasi yang ia tuju sehingga ia diarahkan menuju ke arah segmen yang lain. Terdapat kelemahan lain dari solusi ini, yaitu ketidakpastian bot dalam mendapatkan makanan. Hal ini terjadi karena beda halnya dengan solusi 1, solusi ini menargetkan ‘segmen’. Namun, tidak terdapat kepastian apakah makanan terdapat dalam segmen tersebut atau hanya mendekati saja. Sehingga, dapat terjadi kasus dimana bot bergerak mendekati makanan, tetapi tidak menyentuhnya sama sekali.

### **3.3.3 Alternatif Solusi 3**

Alternatif metode *greedy* ini memiliki tujuan utama untuk mengeliminasi sebanyak mungkin bot lawan yang dapat dieliminasi. Kelebihan dari solusi ini adalah bot dapat secara aktif menyerang lawan sehingga lawan tidak berkembang terlalu besar dan mengancam bot pemain, karena bot terus berusaha mengurangi ukuran lawan dan memakan lawan dengan teleporter jika dimungkinkan. Adapun kelemahan dari solusi ini adalah terdapatnya pengurangan terus menerus dari ukuran bot yang terus menembaki objek-objek yang dapat mengancam bot pada akhirnya. Selain itu, apabila tembakan bot meleset dan tidak dapat mengenai target, bot justru akan semakin mengecil tanpa menyerang lawan dengan benar.

### **3.3.4 Alternatif Solusi 4**

Alternatif metode *greedy* ini memiliki tujuan utama untuk bertahan pada map selama mungkin. Kelebihan dari solusi ini adalah bot dapat memiliki manuverabilitas yang tinggi dan melarikan diri apabila terdapat ancaman terhadap bot, sehingga bot dapat bertahan dengan waktu yang lama di map. Kelemahan dari solusi ini adalah bot tidak memiliki fokus dalam meningkatkan ukurannya, sehingga kesempatan untuk memenangkan pertandingan pada saat sudah memasuki *late-game* cukup kecil, dimana umumnya bot-bot lawan memiliki ukuran yang cukup besar dan dapat memangsa bot pemain.

### **3.3.5 Alternatif Solusi 5**

Alternatif metode *greedy* ini memiliki tujuan utama untuk memperbesar ukuran bot dengan memanfaatkan objek-objek yang dapat dimakan. Kelebihan dari solusi ini adalah bot dapat memiliki ukuran yang cukup besar dan memiliki banyak opsi untuk memangsa bot-bot lain ataupun menggunakan berbagai macam *command* yang mengurangi ukuran, seperti torpedo, teleporter, dan shield. Ukuran yang besar juga dapat mencegah bot dari dimangsa oleh bot lain dan tereliminasi oleh bot lain. Adapun kelemahan dari solusi ini adalah ukuran yang terlalu besar pada permulaan permainan justru dapat menyebabkan bot lebih mudah ditarget dengan menggunakan torpedo dan juga lebih mudah mengenai *gas cloud* yang dapat mengurangi ukuran bot dengan cepat.

### 3.4 Strategi *Greedy* yang Dipilih

Dari alternatif solusi yang ada, kelompok kami pada akhirnya memilih untuk mengimplementasikan strategi *greedy by distance* yang dikombinasikan dengan strategi *greedy by attacking enemies*.

Pada strategi ini, bot akan pertama-tama fokus pada objek-objek yang berada di sekitarnya. Bot kemudian akan menentukan gerakan berdasarkan objek terdekat dari bot tersebut. Apabila objek terdekat merupakan objek yang menguntungkan, berupa food, superfood, supernova charge, dan bot lawan yang lebih kecil, maka bot akan bergerak menuju objek tersebut. Sedangkan, apabila objek terdekat merupakan objek yang membahayakan, berupa torpedo, teleporter, gas cloud, asteroid, dan bot lawan yang lebih besar, maka bot akan bergerak menuju ke arah yang berlawanan dengan objek tersebut. Serta, apabila terdapat objek berupa torpedo yang mengarah pada bot pemain, bot akan menggunakan SHIELD.

Setelah menentukan arah gerak, bot akan menentukan apakah terdapat bot lawan yang dapat diserang dengan menggunakan to afterburner, torpedo, teleporter, ataupun supernova. Penentuan ‘dapat diserang’ ini ditentukan menggunakan algoritma sesuai dengan *commandnya*, dengan berdasarkan variabel-variabel yang diuji melalui *trial and error*. Sebagai contoh, torpedo baru dapat ditembakkan saat bot sudah memiliki ukuran tertentu dan bot lawan terdapat pada *range* yang telah ditentukan. Bot juga dapat menargetkan bot yang dekat dan kecil dengan menggunakan *afterburner* dan mengejar lawan tersebut.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Algoritma pada Program Bot

Berikut ialah implementasi algoritma yang digunakan pada program bot dalam bentuk pseudocode. Seluruh pseudocode yang ditunjukkan merupakan suatu bagian dari sebuah class bernama *Strategy*, sehingga terdapat beberapa variabel yang dapat dibagi antar variabel. Perlu diketahui bahwa bentuk pseudocode telah dimodifikasi sedikit dari *source code* asli, namun dengan esensi yang sama, untuk membentuk pseudocode yang lebih mudah terbaca. Pseudocode ditampilkan dari prosedur-prosedur bagian pelengkap, kemudian di akhir akan ditampilkan pseudocode yang menjadi penggerak utama dari program.

```
procedure extractGameObject(input service: BotService, output
strongEnemy, weakEnemy: List of gameObject, output objectList: List
of List of GameObject)
{ Prosedur untuk mengelompokkan gameObject berdasarkan jarak dan
tipe}
KAMUS
gameObjectList : List of gameObject
enumIdx : integer

Elemen class yang dipakai:
gameState : gameState
objectEnumSize : integer
bot : gameObject

ALGORITMA
bot = service.getBot();
gameState = service.getGameState();
gameObjectList = gameState.getGameObjects();
playerList = gameState.getPlayerGameObjects();

{jika tidak ada gameObject atau tidak ada player, keluar dari
prosedur}
if(isEmpty(gameObjectList) && isEmpty(playerList))return;

{mengelompokkan object berdasarkan tipenya}
for(gameObject in gameObjectList) do
    enumIdx = gameObject.getGameObjectType().getValue()
    {enumIdx adalah indikator ‘tipe’ dari objek}
    objectList[enumIdx].add(gameObject)
```

```

for(gameObject in playerList) do
    enumIdx = gameObject.getGameObjectType().getValue();
    objectList[enumIdx].add(gameObject);
{sorting object berdasarkan jarak}
traversal i 1..objectEnumSize
    sort(objectList[i], comp=getDistanceBetween(bot,object))

{mengelompokkan strong dan weak enemy}
for(player in playerList) do
    if(bot.size >= player.size + weakEnemySizeOffset) then
        weakEnemy.add(player)
    else
        strongEnemy.add(player)

```

**procedure moveLogic(input/output PlayerAction : PlayerAction)**  
{ Prosedur untuk menentukan arah gerak bot berdasarkan gameObject terdekat }

#### KAMUS

towardObject : List of List of GameObject  
avoidObject : List of List of GameObject  
towardDist : List of double  
avoidDist : List of double  
dist : double

Elemen class yang dipakai:

objectList : List of List of GameObject

#### ALGORITMA

```

towardObject.add(objectList[2]); // food
towardObject.add(objectList[7]); // super food
towardObject.add(objectList[8]); // supernova pick
towardObject.add(weakEnemy);

avoidObject.add(objectList[4]); // gas
avoidObject.add(objectList[6]); // torpedo
avoidObject.add(objectList[10]); // teleporter
avoidObject.add(strongEnemy);

{calculate distance for toward & avoid objects}
for(objects in towardObject)
    dist = getShortestObjectListDistance(objects);
    if(dist != infinity) then
        dist -= objects.get(0).size + bot.size;
    towardDist.add(dist);

```

```

for(objects in avoidObject)
    dist = getShortestObjectListDistance(objects);
    if(dist != infinity)then
        dist -= objects.get(0).size + bot.size;
        avoidDist.add(dist);

targetDist = infinity;

{get nearest object, determine heading}
traversal i 1..towardDist.size()
    if(towardDist.get(i) < targetDist) then
        targetDist = towardDist.get(i)
        playerAction.heading= getHeadingBetween(towardObject.get(i)
            .get(0))

traversal i 1..avoidDist.size()
    if(avoidDist.get(i) < targetDist) then
        targetDist = avoidDist.get(i);
        playerAction.heading = (getHeadingBetween(avoidObject.get(i)
            .get(0)) + 180) % 360;

```

**procedure** activateAfterBurnerLogic(input/output PlayerAction : PlayerAction)  
{Prosedur untuk menentukan aktivasi afterburner}

**KAMUS**

```

maximumDist      : int
minimumSize     : double
bigSize         : double
fireChance      : int
enemyMinimumSize : double
usingBoosterCostOffset : double
weakSizeMultiplier : double
strongSizeMultiplier : double
useBooster       : boolean

```

Elemen class yang dipakai:  
weakEnemy, strongEnemy : List of GameObject

**ALGORITMA**  
{Jika afterburner telah dihidupkan, keluar dari prosedur}  
if(bot.effect.haveAfterburner())return;

```

maximumDist = 60;
minimumSize = 80;
{Jika tidak ada player lain atau size terlalu kecil, keluar prosedur}
if(isEmpty(objectList[1]) || bot.size < minimumSize) return;

bigSize = 150;

if(bot.size > bigSize) then
    fireChance = randomFrom(40,100)
else
    fireChance = random.nextInt(0,100);
{Penggunaan random number pada fireChance pada kode diperlukan
sebagai preventive measure dari bug program berupa pemrosesan tick
berkali-kali}

if(fireChance < 75) return
enemyMinimumSize = 20
usingBoosterCostOffset = 15
weakSizeMultiplier = 0.74
strongSizeMultiplier = 1.25
useBooster = false;
for(enemy in weakEnemy)
    dist = getDistanceBetween(bot, enemy) - bot.size - enemy.size;
    if(enemy.size < enemyMinimumSize) continue
    if(enemy.size > (bot.size - usingBoosterCostOffset) *
weakSizeMultiplier) continue
    if(dist <= maximumDist) then
        useBooster = true
        break

for(enemy in strongEnemy){
    dist = getDistanceBetween(bot, enemy) - bot.size - enemy.size;
    if(dist <= maximumDist) then
        useBooster = true
        break

    if(useBooster) then
        playerAction.action = PlayerActions.STARTAFTERBURNER;
    
```

```

procedure deactivateAfterBurnerLogic(input/output PlayerAction :
PlayerAction)
{Prosedur untuk menentukan deaktivasi afterburner}
KAMUS
minimumSize : double
    
```

```

turnoffBooster : boolean
maximumDist    : int
usingBoosterCostOffset : double
weakSizeMultiplier      : double
strongSizeMultiplier    : double

Elemen class yang dipakai:
objectList : List of List of GameObject
bot        : GameObject

ALGORITMA
if(!bot.effect.haveAfterburner())return;

minimumSize = 50
turnoffBooster= true
usingBoosterCostOffset = 7
weakSizeMultiplier=0.9
strongSizeMultiplier=1.1

{Jika player tidak kosong dan bot cukup besar}
if(notEmpty(objectList[1]) || bot.size >= minimumSize) then
    maximumDist = 70;
    for(enemy in weakEnemy)
        dist = getDistanceBetween(bot, enemy) - bot.size - enemy.size
        if(enemy.size > (bot.size - usingBoosterCostOffset) *
        weakSizeMultiplier) then
            continue
        if(dist < maximumDist) then
            turnoffBooster = false
            break
    for(enemy in strongEnemy)
        dist = getDistanceBetween(bot, enemy) - bot.size - enemy.size;
        if(enemy.size < (bot.size - usingBoosterCostOffset) *
        strongSizeMultiplier) then
            continue
        if(dist < maximumDist) then
            turnoffBooster = false
            break

    if(turnoffBooster) then
        playerAction.action = PlayerActions.STOPAFTERBURNER;

```

```

procedure fireTorpedoLogic(input/output PlayerAction : PlayerAction)
{Prosedur untuk menentukan penembakan torpedo}

```

**KAMUS**

```
fireChance : int  
dist       : double
```

Elemen class yang dipakai:

```
objectList : List of List of GameObject  
playerList : List of GameObject  
bot        : GameObject
```

**ALGORITMA**

```
{jika tidak ada pemain, keluar prosedur}  
if(isEmpty(objectList[1]))return  
if (bot.TorpedoSalvoCount>=3) then  
    fireChance = randomFrom(40,100)  
else if (bot.TorpedoSalvoCount>=2 AND nearestPlayer<100) then  
    fireChance = randomFrom(75,100)  
else  
    fireChance = randomFrom(0,100)  
{Penggunaan random number pada fireChance pada kode diperlukan  
sebagai preventive measure dari bug program berupa pemrosesan tick  
berkali-kali yang mengakibatkan tembakan terjadi berturut-turut}  
if (bot.size>minimumSize AND fireChance>=97 AND notEmpty(playerList))  
then  
    for(player in PlayerList) do  
        dist = getDistanceBetween(bot, player)  
        if(dist >= distLowerBound) then  
            Break  
        else  
            playerAction.action = FIRETORPEDOES  
            playerAction.heading = getHeadingBetween(bot,player)
```

**procedure** fireTeleporterLogic(input/output PlayerAction :

PlayerAction)

{Prosedur untuk menentukan penembakan teleporter}

**KAMUS**

```
target      : GameObject  
minimumSize : double  
fireChance  : int  
dist        : double  
distLowerBound : double  
weakSizeMultiplier : double
```

Elemen class yang dipakai:

```
objectList : List of List of GameObject
```

```

bot      : GameObject
weakEnemy : List of GameObject

ALGORITMA
minimumSize = 80;
target = null;

if (bot.size>250) then
    fireChance=randomFrom(40,100);
else
    fireChance=randomFrom(0,100);
{Penggunaan random number pada fireChance pada kode diperlukan sebagai preventive measure dari bug program berupa pemrosesan tick berkali-kali yang mengakibatkan tembakan terjadi berturut-turut}

if(notEmpty(weakEnemy) AND bot.TeleporterCount > 0 AND bot.size >= minimumSize AND isEmpty(teleporterList) AND fireChance>=97) then
    distLowerBound = 1200;
    weakSizeMultiplier = 0.9;
    for(enemy in weakEnemy) do
        if(enemy.size <= (bot.size - 20) * weakSizeMultiplier) then
            if(getDistanceBetween(bot, enemy) <= distLowerBound) then
                target = enemy
                break

if (target != null) then
    playerAction.action = FIRETELEPORT;
    playerAction.heading = getHeadingBetween(target);

```

```

procedure teleportLogic(input/output PlayerAction : PlayerAction)
{Prosedur untuk menentukan aksi teleport}

KAMUS
distLowerBound : double
dist          : double

Elemen class yang dipakai:
objectList : List of List of GameObject
bot       : GameObject
weakEnemy : List of GameObject

ALGORITMA
if(notEmpty(objectList [10])) then
    distLowerBound = 20;

```

```

for(teleporter in teleporterList) do
    for(enemy in weakEnemy)do
        dist = getDistanceBetween(enemy, teleporter)
        dist -= enemy.size + bot.size;
        if(dist <= distLowerBound) then
            playerAction.action = PlayerActions.TELEPORT;

```

```

procedure shieldLogic(input/output PlayerAction : PlayerAction)
{Prosedur untuk menentukan aktivasi shield}

KAMUS
minimumSize      : int
distLowBound     : int
toleratedAngle   : int
useShield        : boolean
angle1           : int
dist             : double

Elemen class yang dipakai:
objectList : List of List of GameObject
bot       : GameObject

ALGORITMA
minimumSize = 50;
if(bot.ShieldCount == 0 || bot.size < minimumSize) return
distLowerBound = 250;
toleratedAngle = 10;
useShield = false;

for(torpedo in objectList[6])
    if(torpedo.size <= minimumTorpedoSize) continue;
    dist = getDistanceBetween(bot, torpedo) - bot.size - torpedo.size;
    angle1 = Math.abs(torpedo.currentHeading -
        ((getHeadingBetween(torpedo) + 180) % 360))
    if(dist > distLowerBound || angle1 > toleratedAngle)
        continue
    useShield = true
    break

if(useShield) then
    playerAction.action = PlayerActions.ACTIVATESHIELD;

```

```

procedure fireSupernovaLogic(input/output PlayerAction :
PlayerAction)
{Prosedur untuk menentukan penembakan supernova}

```

**KAMUS**

Elemen class yang dipakai:  
objectList : List of List of GameObject  
bot : GameObject

**ALGORITMA**

```
{Tembakkan ke player terjauh}  
if(bot.SupernovaAvailable != 0) then  
    playerAction.heading =getHeadingBetween(objectList[1].get(ob  
jectList[1].size()-1))  
    playerAction.action = PlayerActions.FIRESUPERNOVA
```

```
procedure detonateSupernovaLogic(input/output PlayerAction :  
PlayerAction)
```

{Prosedur untuk menentukan aktivasi supernova}

**KAMUS**

distLowerBound : double  
Elemen class yang dipakai:  
objectList : List of List of GameObject  
bot : GameObject

**ALGORITMA**

```
if(isEmpty(objectList[9]))return;  
distLowerBound = 300;  
if(getShortestObjectListDistance(objectList[9]) > distLowerBound)then  
    playerAction.action = PlayerActions.DETONATESUPERNOVA
```

```
procedure compute(input BotService, input/output PlayerAction :  
PlayerAction)
```

{Prosedur yang menjadi penggerak utama dalam penentuan PlayerAction}

**ALGORITMA**

```
extractGameObject()  
defaultAction(playerAction)  
moveLogic(playerAction)  
activateAfterBurnerLogic(playerAction)  
deactivateAfterBurnerLogic(playerAction)  
fireTorpedoLogic(playerAction)  
fireTeleporterLogic(playerAction)  
teleportLogic(playerAction)  
shieldLogic(playerAction)  
fireSupernovaLogic(playerAction)  
detonateSupernovaLogic(playerAction)  
service.setPlayerAction(playerAction)
```

## 4.2 Struktur Data

Terdapat beberapa *class* yang digunakan pada implementasi solusi ini. Berikut adalah gambaran dari struktur data pada tiap *class*.

1. PlayerAction, sebagai aksi dari bot pemain pada setiap *tick*, dengan jenis aksi pada action dan arah pada heading

```
public class PlayerAction {  
    public UUID playerId;  
    public PlayerActions action;  
    public int heading;
```

2. Enumerasi PlayerActions, melakukan *mapping command* terhadap suatu nilai integer

```
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);
```

3. GameObject, yang dapat mendeskripsikan berbagai objek mulai dari pemain, torpedo, rintangan, hingga makanan. Suatu data dapat tersedia ataupun tidak tersedia tergantung pada tipe objek.

```
public class GameObject {  
    public UUID id;  
    public Integer size;  
    public Integer speed;  
    public Integer currentHeading;  
    public Position position;  
    public ObjectTypes gameObjectType;  
    public Integer effect;  
    public Integer TorpedoSalvoCount;  
    public Integer SupernovaAvailable;  
    public Integer TeleporterCount;  
    public Integer ShieldCount;
```

4. Enumerasi gameObjectType, melakukan *mapping* tipe objek pada suatu nilai integer

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

5. GameState, memberikan informasi *realtime* pada setiap *tick* mengenai map beserta objek-objek di dalamnya.

```
public class GameState {  
    public World world;  
    public List<GameObject> gameObjects;  
    public List<GameObject> playerGameObjects;
```

6. World, memberi informasi mengenai ukuran dan titik tengah map, serta tick yang berlangsung.

```
public class World {  
    public Position centerPoint;  
    public Integer radius;  
    public Integer currentTick;
```

7. Position, menunjuk pada suatu titik pada World.

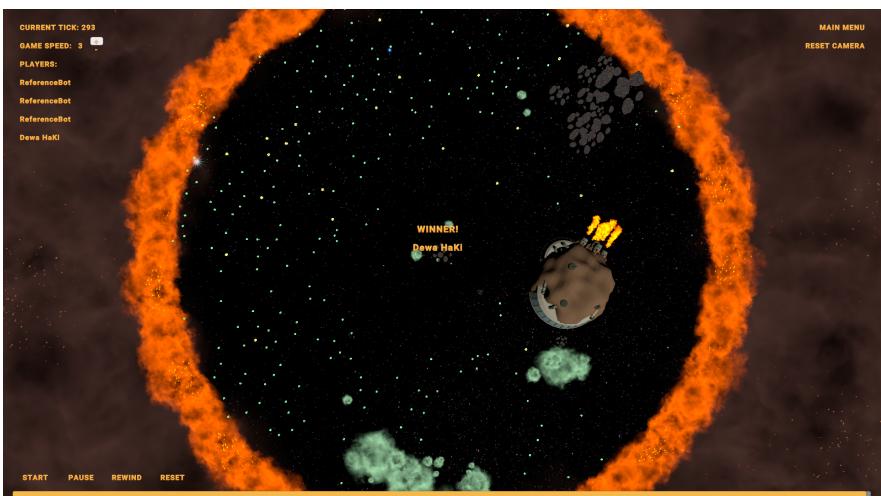
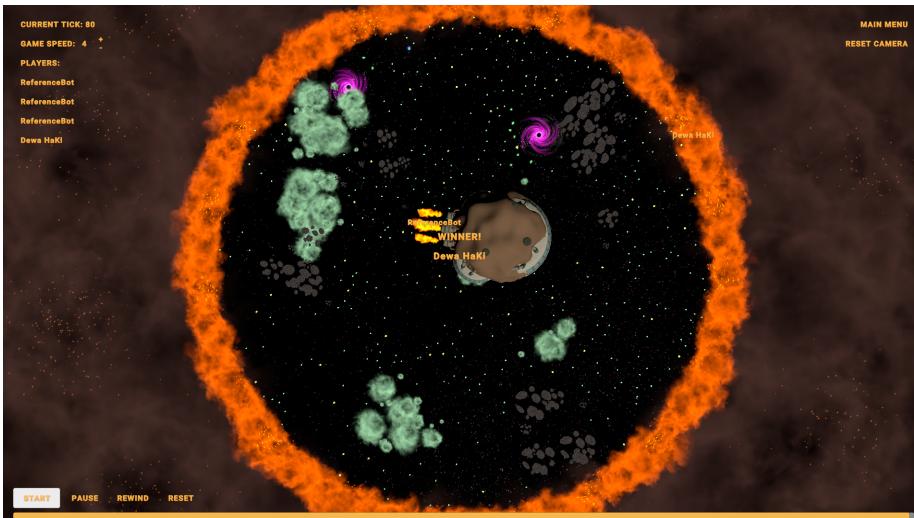
```
public class Position {  
    public int x;  
    public int y;
```

8. Strategy, merupakan class buatan untuk mempermudah mengimplementasikan berbagai jenis fungsi di dalamnya. Mengandung informasi mengenai permainan yang telah diproses oleh prosedur-prosedur di dalamnya.

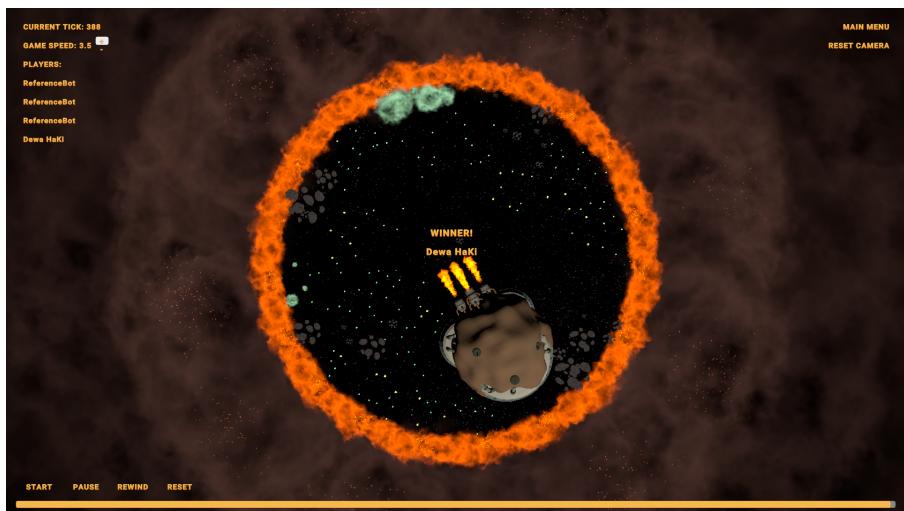
```
public class Strategy {  
    public static final Integer objectEnumSize = ObjectTypes.values().length;  
    public static final Integer infinity = Integer.MAX_VALUE;  
  
    public static GameObject bot;  
    public static GameState gameState;  
    public static World world;  
    public static Random random = new Random();  
    public static List<GameObject>[] objectList = new List[objectEnumSize+1];  
    public static List<GameObject> weakEnemy;  
    public static List<GameObject> strongEnemy;  
    public static long startTime;
```

#### 4.3 Analisis Desain Solusi pada Setiap Pengujian

Dilakukan percobaan bot DewaHaki melawan 3 ReferenceBot sebagai berikut.

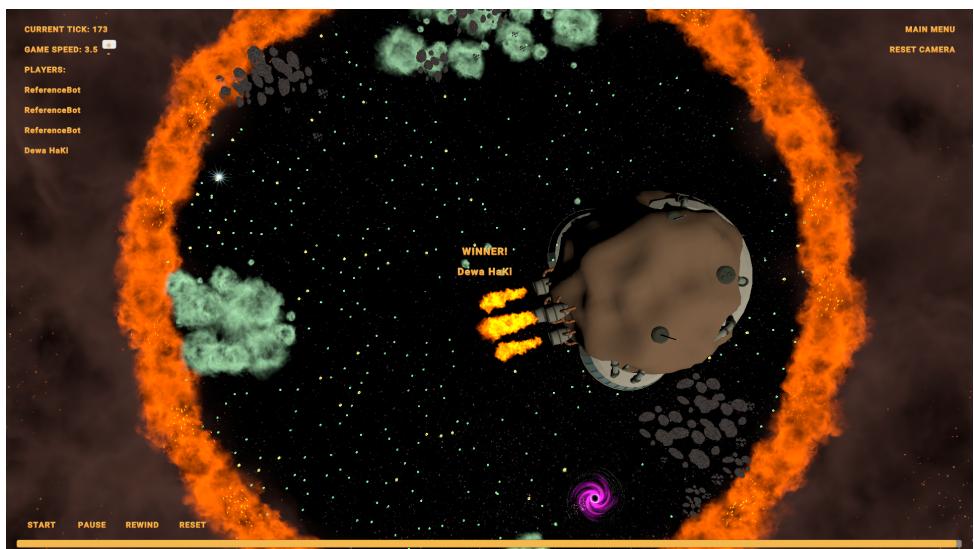
Nomor Percobaan	Hasil
Percobaan 1	 <p>DewaHaki win Tick passed : 293</p>
Percobaan 2	 <p>DewaHaki win Tick passed: 80</p>

Percobaan 3



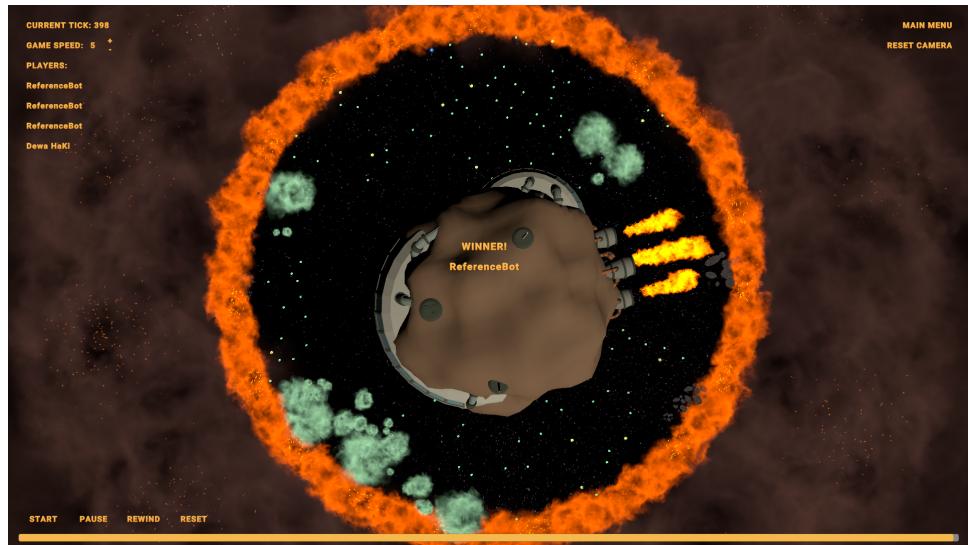
DowaHaKi win  
Tick passed: 388

Percobaan 4



DowaHaKi win  
Tick passed: 173

## Percobaan 5



ReferenceBot win  
Tick passed: 398

Berdasarkan hasil pengujian dari bot Dewa HaKi melawan bot reference yang telah disediakan oleh Entellect Challenge, bot ini memenangkan sebanyak 4 dari 5 pertandingan yang dicobakan. Perlu diketahui bahwa selain dari strategi, kemenangan bot juga sangat bergantung kepada *spawn point*, seperti apakah terdapat banyak makanan pada tempat *spawn* bot atau terdapat lebih banyak *gas cloud*.

Salah satu masalah utama dari Bot DewaHaki adalah terkadang bot cukup ‘bingung’ untuk memilih arah gerak jika berada di antara 2 objek yang jaraknya sama persis. Hal ini menyebabkan bot terdiam di tempat dan menghabiskan *tick* yang cukup berharga untuk berkembang.

Implementasi torpedo dan teleporter dapat dikatakan cukup optimal melihat kemenangan bot umumnya disebabkan pengeliminasian lawan yang cukup cepat dari penggunaan kedua atribut tersebut. Namun, implementasi torpedo dapat dikembangkan dalam segi frekuensi torpedo ditembakkan. Seringkali, bot hanya menembakkan torpedo sebanyak satu atau dua buah sehingga torpedo tidak digunakan secara maksimal.

## **BAB V**

### **PENUTUP**

#### **5.1 Kesimpulan**

Dalam melakukan modifikasi *bot* permainan Galaxio ini, kelompok kami telah berhasil mengimplementasikan algoritma *greedy* untuk memenangkan *game* melawan *reference bot* maupun *bot* lain. Algoritma *greedy* yang kami implementasikan dalam *bot* ini tak terlepas dari berbagai *trials* dan *error* selama proses modifikasinya. Didampingi dengan “heuristik” yang sangat membantu dalam menyempurnakan *bot* ini, strategi *greedy* merupakan solusi yang cukup baik untuk membuat *bot* yang cukup tangguh dan mampu memenangkan *game*.

#### **5.2 Saran**

Beberapa saran yang dapat kami berikan, yaitu

- Masih terdapat beberapa *error* tak terduga yang terjadi saat *game* berlangsung yang kami tidak tahu penyebabnya apa serta terdapat satu prasyarat yang tidak disebutkan dalam spesifikasi yang diberikan sehingga cukup mengalami kendala saat pertama kali melakukan *setup*. Mungkin, untuk kedepannya dapat diberikan prasyarat yang lebih lengkap dalam spesifikasi sehingga *setup* dapat dilakukan dengan lebih mudah dan lancar.
- Bot dapat dikembangkan pada bagian penembakan torpedo menggunakan pengaplikasian prinsip geometri dengan memastikan bahwa pada lintasan torpedo, tidak terdapat halangan yang dapat menghambat torpedo.

#### **5.3 Komentar dan Refleksi**

Tugas besar mengimplementasi bot kali ini dapat dibilang cukup menantang dalam segi pengimplementasiannya. Kode yang ditulis seringkali tidak berjalan sesuai yang diharapkan dan terdapat beberapa *bug* yang menyebabkan program harus didesain dengan mengakomodasi *bug* tersebut. Namun, pengalaman membuat program dalam lingkup kode yang telah ‘matang’ seperti ini juga dapat menjadi pelajaran yang baik untuk kelompok kami.

## **DAFTAR REFERENSI**

- [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)
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf)
- [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](#)
- <https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md#game-tick-payload>
- <https://www.youtube.com/watch?v=ICoPyQA2Fhc>

## **LAMPIRAN**

Tautan repository tugas besar: [https://github.com/DewanaGustavus/Tubes1\\_DewaHaKi](https://github.com/DewanaGustavus/Tubes1_DewaHaKi)

Tautan video demo tugas: <https://youtu.be/uBFEmvEkP4>