

LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI
PERMAINAN “GALAXIO”



KELOMPOK MILKY WAY

ANGGOTA :

- 1. 13521044 RACHEL GABRIELA CHEN**
- 2. 13521046 JEFFREY CHOW**
- 3. 13521074 EUGENE YAP JIN QUAN**

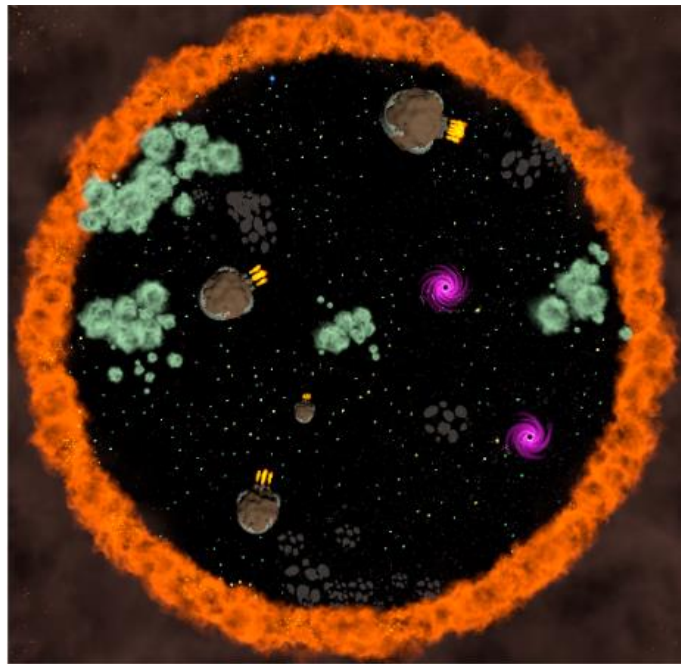
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2023

BAB I

DESKRIPSI MASALAH

Galaxio adalah sebuah game *battle royale* yang mempertandingkan beberapa bot kapal sekaligus. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal menjadi kapal terakhir yang bertahan. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.



Gambar 1. Ilustasi Game Galaxio

Beberapa aturan umum dari *game* ini adalah sebagai berikut.

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

3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal player. Apabila player mengonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command.

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.

Command yang dapat dilakukan oleh setiap pemain adalah sebagai berikut:

1. FORWARD (Bergerak maju sesuai dengan heading yang ditentukan)
2. STOP (Menghentikan gerakan bot)
3. STARTAFTERBURNER (Menyalakan afterburner bot)
4. STOPAFTERBURNER (Mematikan afterburner bot)
5. FIRETORPEDOES (Menembakkan torpedo sesuai dengan heading)
6. FIRESUPERNOVA (Menembakkan supernova sesuai dengan heading)
7. DETONATESUPERNOVA (Menyalakan efek supernova yang telah ditembakkan)
8. FIRETELEPORT (Menembakkan teleporter sesuai dengan heading)
9. TELEPORT (Memindahkan bot ke lokasi teleporter yang telah ditembakkan)
10. ACTIVATESHIELD (Menyalakan efek shield pada bot)

BAB II

LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) dengan prinsip *take what you can get now*, yaitu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma ini berusaha untuk mencapai solusi yang optimal dengan melakukan pemilihan yang paling rasional dan efektif pada setiap tahapnya.

2.2 Elemen Algoritma Greedy

Elemen-elemen yang terdapat pada algoritma greedy, antara lain:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi : menentukan apakah himpunan yang dipilih sudah memberikan solusi
4. Fungsi seleksi (*selection function*): memilih kandidat berdasarkan strategi greedy tertentu . Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: fungsi untuk memaksimumkan atau meminimumkan

2.3 Game Engine Galaxio

Untuk memulai menjalankan *game* dan mengimplementasikan bot, diperlukan starter pack game Galaxio. Starter pack dapat diunduh melalui link ini: <https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>. Selain itu dibutuhkan prasyarat .NET Core 3.1.

Ada beberapa komponen penting yang perlu diketahui untuk memahami cara kerja game, yakni:

1. Engine: komponen yang berperan dalam mengimplementasikan logika dan peraturan game.
2. Runner: komponen yang berperan dalam menggelar sebuah *match* antar-bot dan menghubungkan bot dengan engine.

3. Logger: komponen yang berperan untuk mencatat keberjalanan permainan dalam bentuk sebuah *log*. Hasil log dapat divisualisasikan menggunakan visualiser.

Struktur file starter pack secara umum, yaitu:

- Folder engine-publish: berisi game engine
- Folder runner-publish: berisi game runner
- Folder logger-publish: berisi game logger
- Folder starter-bots: berisi starter bot dalam berbagai bahasa pemrograman
- Folder visualiser: berisi visualiser untuk berbagai OS.

Berikut adalah garis besar cara kerja program game Galaxio:

1. Saat runner dijalankan, runner akan meng-*host* sebuah match pada *hostname* yang telah di-setting. Di lokal, runner akan meng-*host* pada localhost:5000.
2. Kemudian, engine dijalankan untuk melakukan koneksi dengan runner. Setelah itu, engine akan menunggu semua bot pemain terkoneksi ke runner. Logger melakukan hal yang sama. Jumlah bot yang dimainkan dalam sebuah match diatur dengan atribut BotCount pada file “appsettings.json” yang terdapat dalam folder “runner-publish” dan “engine-publish”.
3. Saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi BotCount, permainan akan dimulai.
4. Bot-bot yang terkoneksi kemudian akan mendengarkan event-event dari runner dan juga mengirim event kepada runner yang berupa aksi bot.
5. *Match* akan berlangsung sampai selesai. Setelah selesai, *log* akan dicatat dalam file *json*.

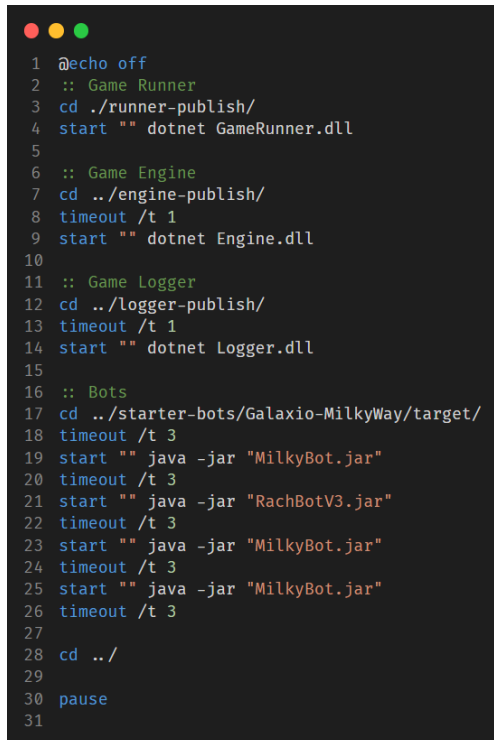
2.4 Alur Menjalankan Program

Berikut adalah langkah-langkah untuk menjalankan program:

1. Ubah konfigurasi jumlah bot yang ingin dimainkan dengan mengatur atribut BotCount pada file “appsettings.json” dalam folder “runner-publish” dan “engine-publish”
2. Buka terminal baru pada folder “runner-publish”.
3. Pada terminal, jalankan runner dengan perintah “dotnet GameRunner.dll”
4. Buka terminal baru pada folder “engine-publish”.
5. Pada terminal, jalankan runner dengan perintah “dotnet Engine.dll”
6. Buka terminal baru pada folder “logger-publish”

7. Pada terminal, jalankan runner dengan perintah “dotnet Engine.dll”
8. Jalankan seluruh bot yang ingin dimainkan.
9. Setelah permainan selesai, *log* permainan akan tercatat dalam 2 *file* JSON dengan format nama “*GameStateLog_{Timestamp}*” dalam folder “logger-publish”

Untuk UNIX-base OS, permainan juga dapat dijalankan dengan menjalankan file *run.sh* yang tersedia dalam starter-pack. Untuk Windows OS, dapat dibuat sebuah file *run.bat* seperti berikut:



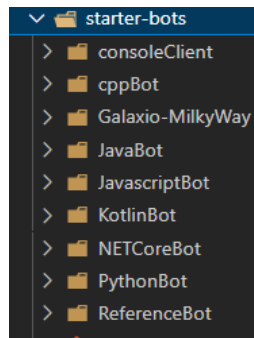
```
1 @echo off
2 :: Game Runner
3 cd ../runner-publish/
4 start "" dotnet GameRunner.dll
5
6 :: Game Engine
7 cd ../engine-publish/
8 timeout /t 1
9 start "" dotnet Engine.dll
10
11 :: Game Logger
12 cd ../logger-publish/
13 timeout /t 1
14 start "" dotnet Logger.dll
15
16 :: Bots
17 cd ../starter-bots/Galaxio-MilkyWay/target/
18 timeout /t 3
19 start "" java -jar "MilkyBot.jar"
20 timeout /t 3
21 start "" java -jar "RachBotV3.jar"
22 timeout /t 3
23 start "" java -jar "MilkyBot.jar"
24 timeout /t 3
25 start "" java -jar "MilkyBot.jar"
26 timeout /t 3
27
28 cd ../
29
30 pause
31
```

Gambar 2. File *run.bat*

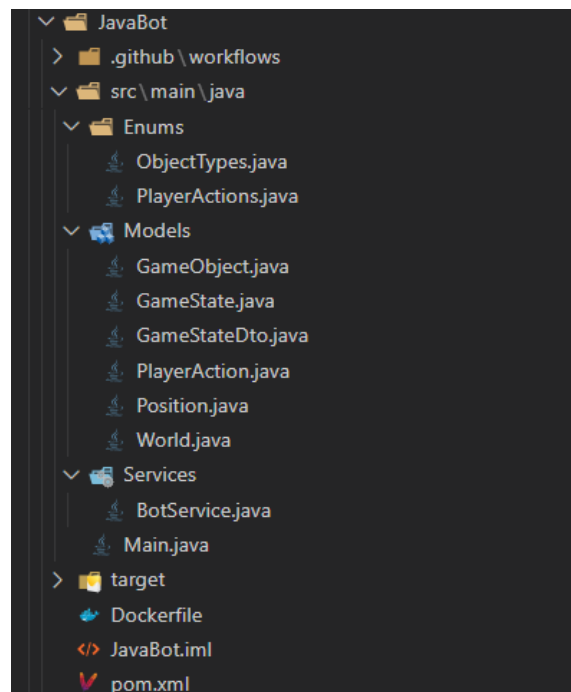
Perintah `java -jar "MilikyBot.jar"` disesuaikan dengan starter-bot yang akan digunakan.

2.5 Alur Pengembangan Bot

Dalam folder “starter-bot”, terdapat *boilerplate* untuk pengembangan bot dalam berbagai bahasa pemrograman.

*Gambar 3. Starter-bot*

Pada laporan ini, akan dijelaskan pengembangan bot dengan bahasa pemrograman java. Berikut adalah struktur file dari JavaBot:

*Gambar 4. Struktur file JavaBot*

Pengembangan dapat dilakukan pada folder src. Untuk mengakses semua fitur dari Galaxio, diperlukan beberapa modifikasi:


```

1 package Enums;
2
3 public enum PlayerActions {
4     FORWARD(1),
5     STOP(2),
6     STARTAFTERBURNER(3),
7     STOPAFTERBURNER(4),
8     FIRETORPEDOES(5),
9     FIRESUPERNOVA(6),
10    DETONATESUPERNOVA(7),
11    FIRETELEPORT(8),
12    TELEPORT(9),
13    ACTIVATESHIELD(10);
14
15    public final Integer value;
16
17    private PlayerActions(Integer value) {
18        this.value = value;
19    }
20 }

```

Gambar 5. Modifikasi Enums PlayerActions.java

```

1 package Enums;
2
3 public enum ObjectTypes {
4     PLAYER(1),
5     FOOD(2),
6     WORMHOLE(3),
7     GASCLLOUD(4),
8     ASTEROIDFIELD(5),
9     TORPEDOSALVO(6),
10    SUPERFOOD(7),
11    SUPERNOVAPICKUP(8),
12    SUPERNOVABOMB(9),
13    TELEPORTER(10),
14    SHIELD(11);
15
16    public final Integer value;
17
18    ObjectTypes(Integer value) {
19        this.value = value;
20    }
21
22    public static ObjectTypes valueOf(Integer value) {
23        for (ObjectTypes objectType : ObjectTypes.values()) {
24            if (objectType.value == value)
25                return objectType;
26        }
27        throw new IllegalArgumentException("Value not found");
28    }
29 }
30 }

```

Gambar 6. Modifikasi Enums ObjectTypes.java

Dapat juga ditambahkan enums Effects.java untuk memudahkan mengecek effects yang aktif pada sebuah bot. Berikut adalah Effects.java:

```

1 package Enums;
2
3 public enum Effects {
4     IsAfterburner(1),
5     InAsteroidField(2),
6     InGasCloud(4),
7     HasSuperfood(8),
8     HasShield(16);
9
10    public final Integer value;
11
12    Effects(Integer value) {
13        this.value = value;
14    }
15
16    public Integer getValue(){
17        return value;
18    }
19
20    public static Effects valueOf(Integer value) {
21        for (Effects effect : Effects.values()) {
22            if (effect.value == value)
23                return effect;
24        }
25
26        throw new IllegalArgumentException("Value not found");
27    }
28 }

```

Gambar 7. Enums Effects.java

Pada GameObjects.java, perlu dilakukan modifikasi pada atribut class, fungsi FromStateList dan juga pada kontruktor class.

```

1 public UUID id;
2 public Integer size;
3 public Integer speed;
4 public Integer currentHeading;
5 public Position position;
6 public ObjectTypes gameObjectType;
7 public Integer effects;
8 public Integer torpedoSalvoCount;
9 public Integer supernovaAvailable;
10 public Integer teleporterCount;
11 public Integer shieldCount;

```

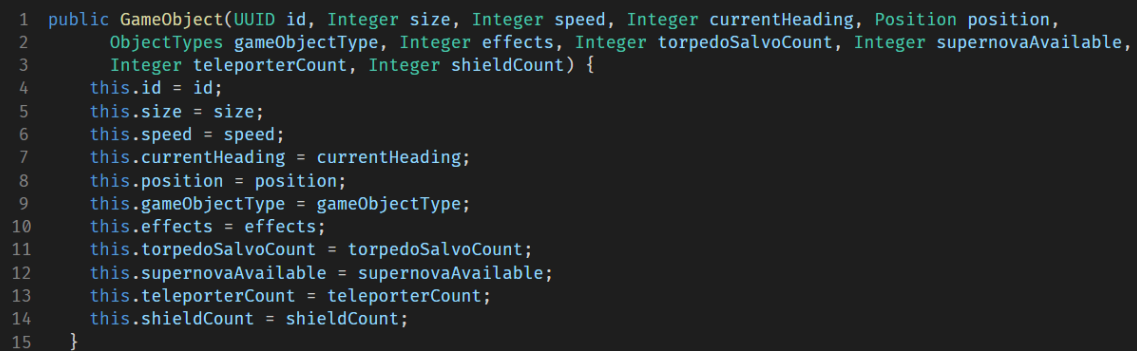
Gambar 8. Modifikasi Atribut Class GameObjects

```

1 public static GameObject FromStateList(UUID id, List<Integer> stateList) {
2     Position position = new Position(stateList.get(4), stateList.get(5));
3     if (stateList.size() != 11) {
4         return new GameObject(id, stateList.get(0), stateList.get(1), stateList.get(2), position,
5             ObjectTypes.valueOf(stateList.get(3)), 0, 0, 0, 0, 0);
6     }
7     return new GameObject(id, stateList.get(0), stateList.get(1), stateList.get(2), position,
8         ObjectTypes.valueOf(stateList.get(3)), stateList.get(6), stateList.get(7),
9         stateList.get(8), stateList.get(9), stateList.get(10));
10 }
11

```

Gambar 9. Modifikasi Fungsi FromStateList



```
1 public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position,  
2     ObjectTypes gameObjectType, Integer effects, Integer torpedoSalvoCount, Integer supernovaAvailable,  
3     Integer teleporterCount, Integer shieldCount) {  
4     this.id = id;  
5     this.size = size;  
6     this.speed = speed;  
7     this.currentHeading = currentHeading;  
8     this.position = position;  
9     this.gameObjectType = gameObjectType;  
10    this.effects = effects;  
11    this.torpedoSalvoCount = torpedoSalvoCount;  
12    this.supernovaAvailable = supernovaAvailable;  
13    this.teleporterCount = teleporterCount;  
14    this.shieldCount = shieldCount;  
15 }
```

Gambar 10. Modifikasi Konstruktor GameObjects

Pengembangan bot dapat dimulai dengan mengubah string “Coffee Bot” pada Main.java menjadi nama bot yang dikehendaki. Pada Main.java, aksi bot ditentukan oleh method `computeNextPlayerAction` yang terdapat di class `BotService`. Implementasi strategi dan logika bot dilakukan pada fungsi `computeNextPlayerAction` di `BotService.java`.

Setelah bot selesai dikembangkan, dapat dilakukan build menggunakan maven atau IntelliJ IDEA.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Alternatif Greedy

3.1.1 Greedy by Damage to Opponent

Greedy by damage to opponent adalah strategi *greedy* yang mengutamakan melakukan *damage* sebesar-besarnya terhadap lawan-lawan yang ada. *Damage* terhadap opponent dapat terjadi saat bot bertabrakan dengan lawan dan memiliki ukuran lebih besar dari lawan atau saat bot menembakkan torpedo ke arah kapal lawan. Pada setiap tick, prioritaskan mengurangi ukuran lawan dengan memberi *damage* ke kapal lawan.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: semua pasangan aksi (FIRETELEPORTER, TELEPORT, FIRETORPEDOES, FORWARD, STARTAFTERBURNER) dan *heading* (PlayerAction) yang mungkin.
- b. Himpunan solusi: pasangan aksi dan *heading* (PlayerAction) yang menyebabkan *damage* optimum terhadap lawan.
- c. Fungsi solusi: memeriksa apakah PlayerAction yang dipilih dapat berhasil menyebabkan *damage* pada lawan.
- d. Fungsi seleksi: memilih PlayerAction yang paling mungkin berhasil untuk menyebabkan *damage* pada lawan.
- e. Fungsi kelayakan: memeriksa apakah PlayerAction memenuhi restriksi-restriksi dan kondisi untuk dilakukan saat itu.
- f. Fungsi obyektif: ukuran yang dikorbankan bot saat melakukan PlayerAction minimum.

- **Analisis Efisiensi Solusi**

Pada himpunan kandidat, terdapat 5 aksi yang perlu dibandingkan. Dilakukan pengecekan pemilihan setiap aksi untuk menyerang lawan. Apabila aksi yang dilakukan memungkinkan untuk menyebabkan *damage* pada lawan, pilih aksi tersebut. Pemilihan aksi diprioritaskan berdasarkan *damage*. Sebelum dilakukan pengecekan, list opponent di-*sort* berdasarkan jarak dengan bot dan juga berdasarkan ukurannya. Kompleksitas untuk tahap ini adalah $O(n \log n)$.

Pertama, dilakukan pengecekan untuk aksi FIRETELEPORTER. Lawan diurutkan berdasarkan ukuran terbesar karena tujuan dari penembakan teleporter adalah untuk memudahkan bot menabrak lawan sehingga *damage* dapat dimaksimalkan. Kompleksitas dari tahap ini adalah $O(n)$

Kedua, dilakukan pengecekan untuk aksi TELEPORT. Pengecekan ini dilakukan dengan mencari teleporter yang berada dekat dengan lawan yang memungkinkan bot menabrak lawan. Kompleksitas dari tahap ini adalah $O(n^2)$.

Ketiga, dilakukan pengecekan untuk aksi FIRETORPEDOES. Pada tahap ini, dilakukan pencarian lawan yang memungkinkan untuk ditembak. Lawan diurutkan berdasarkan jarak dengan bot karena semakin dekat jarak tembak, semakin akurat tembakan dan semakin besar *damage* yang bisa didapatkan. Kompleksitas dari tahap ini adalah $O(n)$

Selanjutnya, dilakukan pengecekan untuk aksi STARTAFTERBURNER dan FORWARD. Dilakukan pencarian pencarian lawan yang dapat dikejar oleh bot sehingga dapat terjadi *collision* dengan lawan. Lawan diurutkan berdasarkan jarak dengan bot. Kompleksitas dari tahap ini adalah $O(n)$.

Kompleksitas waktu algoritma ini secara umum adalah $O(n^2)$,

- **Analisis Efektivitas Solusi**

Strategi ini mengutamakan implikasi jangka panjang dari aksi yang dipilih dibanding *immediate benefit*.

Strategi ini efektif apabila:

- Ukuran bot cukup besar sehingga dapat pengorbanan ukuran bot sebanding dengan keuntungan dari *damage* yang diberikan terhadap lawan.
- Jarak antar bot dengan lawan cukup kecil.

Strategi ini kurang efektif apabila:

- Ukuran bot kecil.
- Bot lawan cukup cerdas untuk mendeteksi ancaman yang datang ke arahnya sehingga implikasi yang diharapkan dari aksi yang dipilih tidak terjadi.

3.1.2 Greedy by Bot Size Gain

Greedy by Bot Size Gain adalah strategi yang mengutamakan mempertahankan atau memperbesar ukuran bot dengan mengaktifkan shield, mencari makanan, atau menghentikan afterburner. Pada setiap tick, prioritaskan aksi yang dapat memaksimalkan ukuran bot secara *immediate*.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: semua pasangan aksi (FORWARD, STOPAFTERBURNER, ACTIVATESHIELD) dan *heading* (PlayerAction) yang mungkin.
- b. Himpunan solusi: pasangan aksi dan *heading* (PlayerAction) yang memaksimalkan ukuran bot.
- c. Fungsi solusi: memeriksa apakah PlayerAction yang dipilih dapat memaksimalkan ukuran bot.
- d. Fungsi seleksi: memilih PlayerAction yang paling memaksimalkan ukuran bot.
- e. Fungsi kelayakan: memeriksa apakah PlayerAction memenuhi restriksi-riksi dan kondisi untuk dilakukan saat itu.
- f. Fungsi obyektif: ukuran yang dikorbankan bot saat melakukan PlayerAction minimum.

- **Analisis Efisiensi Solusi**

Pada algoritma ini dilakukan pengecekan untuk aksi FORWARD, STOPAFTERBURNER, dan ACTIVATESHIELD. Pertama dilakukan *sorting* list FOOD, SUPERFOOD, dan TORPEDO berdasarkan jarak dengan bot. Untuk seleksi aksi FORWARD, dicari makanan dengan jarak terdekat dan juga dengan perbedaan heading minimum dengan kompleksitas $O(n \log n)$. Makanan yang diprioritaskan adalah makanan bertipe SUPERFOOD. Kompleksitas dari tahap ini adalah $O(n)$.

Selanjutnya, dilakukan pengecekan untuk aksi STOPAFTERBURNER. Jika efek afterburner mengurangi ukuran bot dengan signifikan tanpa memberikan keuntungan bagi bot, maka aksi ini dipilih. Kompleksitas dari tahap ini adalah $O(1)$ karena keputusan diambil hanya mempertimbangkan lawan terdekat.

Kemudian, dilakukan pengecekan untuk aksi ACTIVATESHIELD. Jika terdapat torpedo yang mendekati bot, maka aksi ini dipilih. Kompleksitas dari tahap ini adalah $O(1)$ karena keputusan diambil hanya mempertimbangkan lawan terdekat.

- **Analisis Efektivitas Solusi**

Strategi ini efektif apabila:

- Berada pada tahap *early game* dan terdapat banyak FOOD yang dapat diambil oleh bot.
- Tidak terdapat banyak *obstacle* dalam map.
- Bot sedang mendapatkan *damage* dari lawan.

Strategi ini kurang efektif apabila:

- Game berada pada tahap *late game* sehingga FOOD yang ditarget berada pada jarak yang jauh.
- FOOD berada dekat dengan *obstacle*.
- Bot berhadapan dengan musuh dengan ukuran lebih besar.

3.1.3 Greedy by Least Damaging Action

Greedy by Least Damaging Action adalah strategi yang memilih aksi yang meminimumkan *immediate lost* ukuran bot. Algoritma ini berguna karena beberapa aksi yang dilakukan oleh bot membutuhkan pengorbanan ukuran bot. Pada setiap tick, prioritaskan aksi dengan *cost size* paling rendah.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: semua pasangan aksi dan *heading* (PlayerAction) yang mungkin.
- b. Himpunan solusi: pasangan aksi dan *heading* (PlayerAction) dengan *cost* minimum.
- c. Fungsi solusi: memeriksa apakah PlayerAction dapat dilakukan dan memiliki *cost* yang tidak merugikan bot.
- d. Fungsi seleksi: memilih PlayerAction dengan *cost* minimum.
- e. Fungsi kelayakan: memeriksa apakah PlayerAction memenuhi restriksi-restriksi dan kondisi untuk dilakukan saat itu.
- f. Fungsi obyektif: target dari aksi memiliki jarak minimum.

- **Analisis Efisiensi Solusi**

Pada *greedy by least damaging action*, setiap object dalam world diurutkan berdasarkan jarak dari bot dengan kompleksitas $O(n \log n)$. Aksi FORWARD akan diprioritaskan karena memiliki *cost* 0. Target dari aksi forward adalah objek-objek *non-obstacle* (FOOD, SUPERFOOD, dan PLAYER) dengan ukuran lebih kecil dari bot dan memiliki jarak minimum. Kompleksitas dari tahap ini adalah $O(n)$.

Jika tidak ada aksi FORWARD yang memenuhi fungsi kelayakan, maka selanjutnya akan diperiksa aksi TELEPORT (*cost* 0) yang memeriksa jarak antara setiap teleporter dan bot lawan dengan kompleksitas waktu $O(n^2)$. Jika belum ada solusi, diperiksa aksi STOPAFTERBURNER dengan kompleksitas waktu $O(1)$.

Jika masih belum ada solusi dari aksi-aksi dengan *cost* 0, aksi-aksi lainnya diperiksa dengan urutan:

1. FIRETORPEDO (*cost* 10) : memeriksa kelayakan *heading* terhadap setiap lawan dengan kompleksitas $O(n)$.
2. ACTIVATESHIELD (*cost* 20) : memeriksa apakah *shield* diperlukan terhadap setiap torpedo dengan kompleksitas $O(n)$. Aksi ini lebih diprioritaskan dibanding FIRETELEPORTER karena implikasinya meminimumkan damage yang diterima oleh bot.
3. FIRETELEPORTER (*cost* 20) : memeriksa apakah ada lawan yang berada dalam jangkauan teleporter dengan kompleksitas $O(n)$.
4. STARTAFTERBURNER (*cost* -1/tick) : memeriksa apakah ada lawan yang dapat dikejar dengan kompleksitas $O(n)$.

- **Analisis Efektivitas Solusi**

Strategi ini efektif apabila:

- Ukuran bot kecil.
- Game berada pada tahap *early game*.
- Jumlah lawan kecil.

Strategi ini kurang efektif apabila:

- Bot terus mendapatkan *damage* dari lawan karena tidak adanya mekanisme “perlindungan”.
- Ukuran bot sudah cukup besar.

3.2 Strategi Greedy yang Diimplementasikan

Berdasarkan alternatif greedy pada 3.1, dapat dilihat bahwa terdapat beberapa alternatif greedy dengan efektivitasnya masing-masing. Bot yang diimplementasikan menggunakan pendekatan greedy by damage to opponent dan greedy by bot size sesuai dengan kondisi gamestate pada setiap tick. Poin penting dari implementasi bot antara lain:

1. Utamakan *greedy by damage to opponent* karena dapat memberikan keuntungan yang lebih besar daripada *greedy by bot size*. Selain itu, *greedy by damage to opponent* juga memudahkan bot untuk menjadi bot terakhir yang bertahan dalam permainan.
2. Jika tidak ada solusi yang memenuhi fungsi kelayakan pada *greedy by damage*, pilih *PlayerAction* dengan pendekatan *greedy by bot size*.
3. Jika tidak ada solusi yang memenuhi fungsi kelayakan pada *greedy by bot size*, pilih aksi FORWARD dengan *heading* yang memenuhi fungsi kelayakan.
4. Pada setiap tick, lakukan pengecekan apabila bot mendekati batas *world*, jika ya, arahkan bot ke tengah *world*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Repositori Github

https://github.com/chaerla/Tubes1_MilkyWay

4.2 Implementasi dalam *Pseudocode*

sort GameObjects

```
opponentsByDist <- sort ascending distance between bot and opponents
opponentsBySize <- sort ascending size between bot and opponents
foodList <- sort ascending distance between bot and foods
wormholeList <- sort ascending distance between bot and wormholes
gasCloudList <- sort ascending distance between bot and gasclouds
asteroidList <- sort ascending distance between bot and asteroid fields
torpedoList <- sort ascending distance between bot and torpedoes
superFoodList <- sort ascending distance between bot and superfoods
teleporterList <- sort ascending distance between bot and teleporters
foodListByHeadingGap <- sort ascending gap between bot and foods
{ algoritma sorting terdefinisi }
```

Restriksi heading menuju obstacles

```
headingRestriction <- new DegreeRestriction() { DegreeRestriction adalah kelas yang
untuk heading yang dilarang }
{ restriksi heading ke asteroid terdekat }
if (length(asteroidList) > 0) then
    if (inAsteroidField || (distanceBetween(bot and asteroidList[0] - bot.size()
- asteroidList[0].size()) <= 20) then
        headingToAsteroid <- getHeadingBetween(asteroidList[0])
        headingRestriction.restrictRange(headingToAsteroid, headingGap
(asteroidList[0], bot))

{ restriksi heading ke wormhole terdekat }
if (length(wormholeList) > 0) then
    if (distanceBetween(bot, wormholeList[0]) <= 20) then
        headingToWormhole <- getHeadingBetween(wormholeList[0])
        restrictRange(headingToWormhole, headingGap (wormholeList[0], bot))
```

```

{ restriksi heading ke gasCloud terdekat }
if (length(gasCloudList) > 0) then
    if (distanceBetween(bot, gasCloudList[0]) <= 20) then
        headingToGasCloud <- getHeadingBetween(gasCloudList[0])
        restrictRange(headingToGasCloud, headingGap (gasCloudList[0], bot))

```

Greedy by Damage to Opponent

```

distToNearestOpp <- distanceBetween(bot, opponentsByDist[0])
headToNearestOpp <- getHeadingBetween(opponentsByDist[0])
strategied <- false
chase <- false
{ Mengecek untuk aksi FIRETELEPORT }
if (not strategied and bot.hasTeleporter()) then
    oppIndex <- -1
    { Syarat FIRETELEPORT: ukuran opponent < ukuran bot - cost FIRETELEPORT -
    10, ukuran bot > 60, dan waktu sejak teleporter terakhir ditembakkan > 50
    (menghindari teleporter ditembakkan berulang kali )
    i traversal [0..length((opponentBySize)) -1]
        if (opponentBySize[i].size() < bot.size() - 30 and bot.size() > 60
and world.tick() - lastTeleporterTick > 50) then
            oppIndex <- i
            break
    { Jika ukuran opponent yang sedang ditraverse > ukuran bot, break karena yang lain
    pasti lebih besar juga }
        else if (opponentsBySize[i].size() > bot.size()) then
            break
    if (oppIndex != -1) then
        playerAction.heading<-getHeadingBetween(bot,
opponentsBySize[oppIndex])
        playerAction.action <- FIRETELEPORT
        strategied <- true

```

```

{ Mengecek untuk aksi TELEPORT }
if (not strategied) then
    foundValidTarget <- false
    for (tele in teleporterList) do
        for (opponent in opponentsBySize) do
            { Syarat TELEPORT : jarak antara lawan dengan teleporter <
            (ukuran bot + ukuran lawan) * 1.1 dan ukuran bot > ukuran lawan }
            if (distanceBetween(opponent,tele)< bot.size() +
            opponent.size() * 1.1 and bot.size() > opponent.size()) then
                playerAction.heading <- getHeadingBetween(bot,
                opponent)
                playerAction.action <- TELEPORT
                strategied <- true
                foundValidTarget <- true
                break
            if (foundValidTarget) then
                break

{ Mengecek aksi FIRETORPEDOES }
{ Hanya mengecek dengan lawan terdekat }
if (not strategied and
    bot.hasTorpedo() and
    bot.size() > 50 and
    (((length(foodList) + bot.size() < opponentsByDist[0].size()) and
    distToNearestOpp < world.radius() * 0.4) or
    (opponentsByDist[0].size() < bot.size() - bot.torpedoSalvoCount * 10 and
    distToNearestOpp < world.getRadius() * 1.2 )
    or (distToNearestOpp < 75)
then
    playerAction.heading <- headToNearestOpp
    playerAction.action <- FIRETORPEDOES
    strategied <- true
    if (isUsingAfterBurner and
        (bot.size()-(distToNearestOpp/bot.speed()))>
    opponentsByDist[0].size()) and
        distToNearestOpp < world.radius() * 0.8
    )
    then

```

```

        playerAction.heading <- headToNearestOpp
        playerAction.action <- FIRETORPEDOES
        strategied <- true
        if ( isUsingAfterBurner and
            ( bot.size() - (distToNearestOpp/bot.speed()) >
opponentsByDist[0].size()) and
            distToNearestOpp < world.radius() * 0.8
        ) then
            if (bot.torpedoSalvoCount <= 3) then
                playerAction.heading <- headToNearestOpp
                playerAction.action <- FORWARD
                chase <- true
{ Mengecek aksi    STARTAFTERBURNER }
{ Syarat STARTAFTERBURNER: ukuran bot dengan pengurangan setiap tick akibat
afterburner harus lebih besar dari lawan terdekat }
if ( not strategied and
    (bot.size() - (distToNearestOpp/bot.speed()) > opponentsByDist[0].size() *
1.2) and
    distToNearestOpp < world.radius() * 0.8
) then
    if (not isUsingAfterBurner) then
        playerAction.heading <- headToNearestOpp
        playerAction.action <- STARTAFTERBURNER
    else
        { Jika afterburner sudah aktif, jalan ke arah lawan terdekat }
        playerAction.heading <- headToNearestOpp
        playerAction.action <- FORWARD
    strategied <- true
    chase <- true

```

Greedy by Bot Size Gain

```

{proses pemilihan target food}
if (not strategied) then
  {memastikan food tidak berada di antara bot dan opponent yang lebih besar
  dan berada pada jarak dekat}
  for (opponent in opponentsByDist) do
    if (distanceBetween(bot, opponent) < 100 and bot.size() <
    opponent.size()) then
      headingToThisOpp <- headingBetween(bot, opponent)
      restrictRange(headingToThisOpp)

  {mencari food yang valid}
  indexFood <- -1
  i traversal [0..length(foodList)-1]
    if (isDegValid(headingBetween(bot, foodList[i]))) then
      indexFood <- i
      break

  {mencari superfood yang valid}
  indexSuperFood <- -1
  if (not hasSuperFood) then
    i traversal [0..length(superFoodList)-1]
      if (isDegValid(headingBetween(bot, superFoodList[i])))
      then
        indexSuperFood <- i
        break

  {mencari makanan dengan selisih heading terkecil terhadap heading bot}
  i traversal [0..length(foodListByHeadingGap)-1]
    currFood <- foodListByHeadingGap[i]
    if (isDegValid(headingBetween(bot, currFood)) and
        abs(distanceBetween(bot, foodList[indexFood])) -
        distanceBetween(currFood, bot) <= 2)
    then
      indexFood <- indexOf(foodList.filter(item in foodList ->
        item.currentHeading()==currFood.currentHeading()))
      break

```

```

{pemilihan food atau superfood}
if (indexFood != -1 and indexSuperFood != -1) then
    if (distanceBetween(bot, foodList[indexFood]) * 1.5 >=
        distanceBetween(bot, superFoodList[indexSuperFood])) then
        {target adalah superfood}
        heading <- distanceBetween(bot, superFoodList[indexSuperFood])
    else
        {target adalah food}
        heading <- headingBetween(bot, foodList[indexFood])
else if (indexFood != -1) then
    heading <- headingBetween(bot, foodList[indexFood])
else if (indexSuperFood != -1) then
    heading <- headingBetween(bot, superFoodList[indexSuperFood])
else
    heading <- getNearestValidHeading(bot.currentHeading, 1)

playerAction.heading <- heading
playerAction.action <- PlayerActions.FORWARD

```

Command Finalization

```

{melakukan finalisasi terhadap strategi/command terpilih; ganti strategi apabila ada hal
yang mendesak}
{cek 1: apakah shield perlu diaktifkan (cek torpedo di sekitar bot)}
if (length(torpedoList) > 0) then
    torpedoHeading <- torpedoList[0].heading
    if (bot.hasShield() and
        not isUsingShield and
        distanceBetween(bot,torpedoList[0]) < bot.size() + 50 and
        torpedoList[0].size() >= 2 and
        bot.size() > 50 and
        bot.size() < 350
    ) then
        playerAction.action <- ACTIVATESHIELD
{cek 2: apakah bot berada di dekat world border; x dan y adalah posisi bot dalam kartesian}
if (x2 + y2 < (0.9 * (worldRadius - bot.size()))2 and
    playerAction != FIRETORPEDOES)
then
    playerAction.action <- FORWARD
    playerAction.heading <- heading to world center

```

```

{cek 3: apakah afterburner dapat mematikan bot (cek ukuran bot dan status afterburner)}
if (usingAfterBurner and
    (not chase || usingShield || (
        distanceBetween(bot,opponentsByDist[0]) < bot.size() * 2 and
        opponentsByDist[0].size() > bot.size()
    ))
then
    playerAction.action <- STOPAFTERBURNER

{menyimpan informasi teleporter jika command final adalah teleport}
if (playerAction.action = FIRETELEPORT) then
    lastTeleporterTick <- world.getCurrentTick()

{mengirim strategi/command}
this.playerAction <- playerAction

```

4.3 Struktur Data Program

Bahasa pemrograman yang dipilih untuk implemementasi bot adalah Java. Struktur data pada program didefinisikan dengan menggunakan *class*. *Class* yang digunakan dalam pengembangan bot terdapat pada folder Enums dan Models.

Class pada “Models” antara lain:

- **GameObjects.java**
GameObjects merupakan *class* untuk objek-objek pada game. Setiap objek pada *class* ini memiliki atribut *id*, *size*, *speed*, *currentHeading*, *position*, *gameObjectType*, *effects*, *toperdoSalvoCount*, *supernovaAvaliable*, *teleporterCount*, dan *shieldCount*.
- **PlayerActions.java**
PlayerActions merupakan *class* untuk aksi-aksi bot yang menyimpan ID player, aksi player, dan juga heading untuk aksi tersebut. PlayerActions juga merupakan kelas yang mengirimkan aksi *player*.
- **Position.java**
Position merupakan *class* untuk posisi objek dalam game, yang terdiri atas koordinat *x* dan *y*.
- **World.java**
World merupakan *class* untuk world/area permainan dan memiliki atribut *centerPoint*, *radius*, dan *currentTick*.

- DegreeRestriction.java

DegreeRestriction merupakan *class* yang terdiri atas sebuah list of boolean. Indeks setiap elemen dalam list menyatakan *heading* (0-359) dan akan bernilai true jika *heading* tersebut tidak layak.

Pada “Enums” terdapat:

- PlayerActions.java

PlayerActions berisi enumerasi dari aksi-aksi yang dapat dilakukan oleh bot dalam game. PlayerActions terdiri atas FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRESUPERNOVA, DETONATESUPERNOVA, FIRETELEPORT, TELEPORT, dan ACTIVATESHIELD.

- Effects.java

Effects berisi enumerasi dari efek-efek yang dapat dipunyai oleh bot. Effects terdiri dari IsAfterburner, InAsteroidField, InGasCloud, HasSuperfood, HasShield. Efek ini disimpan dalam dioperasikan secara *bitwise*.

- ObjectTypes.java

ObjectTypes berisi enumerasi dari tipe-tipe objek dalam game. ObjectTypes terdiri atas PLAYER, FOOD, WORMHOLE, GASECLOUD, ASTEROIDFIELD, TORPEDOSALVO, SUPERFOOD, SUPERNOVAPICKUP, SUPERNOVABOMB, TELEPORTER, dan SHIELD.

Pada Services terdapat:

- BotService.java

BotService menyimpan state dari game pada setiap tick, informasi tentang bot, dan juga aksi yang dikirimkan bot pada setiap tick. Pada kelas ini juga didefinisikan method `computeNextPlayerAction` dimana algoritma greedy diimplementasikan. Kami juga menambahkan member `lastTeleporterTick` untuk menyimpan kapan teleporter terakhir kali ditembakkan.

4.4 Analisis dan Pengujian



Kami melakukan analisis dan pengujian terhadap MilkyBot (atau disebut dengan Bot pada analisis) dengan cara mempertarungkan MilkyBot dengan bot lawan dari kelompok lain. Pada contoh pengujian yang dilampirkan pada bawah ini, MilkyBot melawan 3 buah bot yaitu dari bot NaN, bot BotOne, dan bot Phiber Optik. Kami melakukan pengujian pada beberapa aksi yang dapat dilakukan oleh bot.

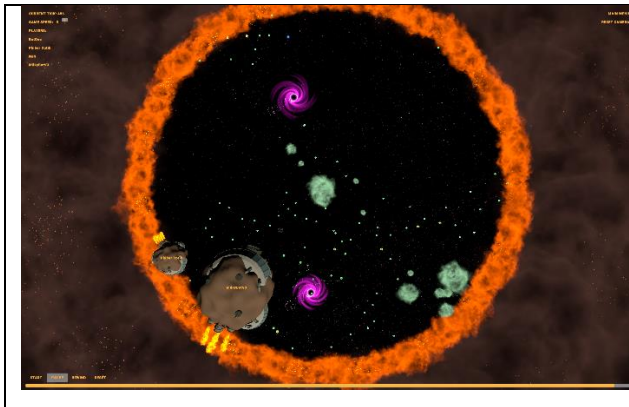
4.4.1 Teleporter

Pada gambar pertama tampak bahwa Bot berhasil menembakkan sebuah *teleporter* ke arah lawan. Berdasarkan gambar, perilaku Bot sesuai dengan implementasi, yaitu menembak *teleporter* ke arah lawan yang memiliki ukuran lebih kecil dari Bot.

Pada gambar kedua, tampak bahwa *teleporter* sudah berada pada jarak relatif dekat terhadap lawan. Berdasarkan implementasi kami, apabila lawan tidak berhasil menghindari *teleporter* dalam waktu dekat, Bot akan berpindah menuju *teleporter*.

Karena syarat ukuran dan jarak antara parameter Bot, lawan, dan *teleporter* masih terpenuhi, Bot melakukan TELEPORT menuju lawan. Pada gambar di samping tampak bahwa Bot telah mengonsumsi lawan setelah melakukan TELEPORT.

Gambar	Keterangan
	<p>MilkyBot (kanan atas) menembakkan sebuah <i>teleporter</i> ke arah lawan yang memiliki ukuran lebih kecil (kiri bawah)</p>
	<p>Teleporter yang telah ditembak oleh MilkyBot semakin mendekati posisi lawan (kiri bawah)</p>



MilkyBot melakukan TELEPORT menuju teleporter dan mengonsumsi lawan dengan ukuran lebih kecil tersebut sehingga aksi FIRETELEPORT dan TELEPORT yang telah dilakukan **optimum**.

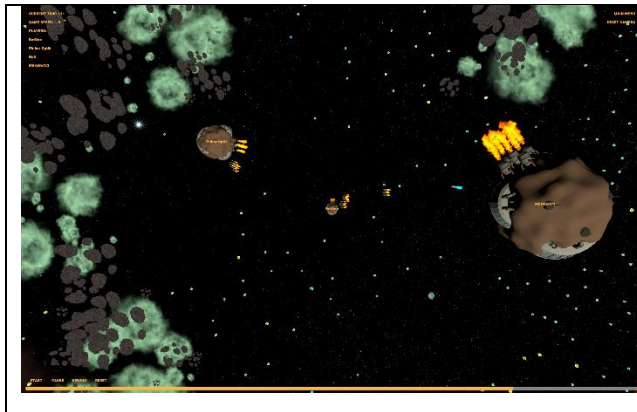
4.4.2 Torpedo

Pada implementasi Bot, kami menetapkan aturan-aturan untuk menembakkan torpedo. Dalam implementasinya, kami menetapkan musuh terdekat sebagai target penembakkan torpedo. Selain itu, kami juga mengimplementasikan batasan jarak terjauh antara Bot dan lawan (dengan tujuan mengurangi pemborosan), serta batasan ukuran Bot.

Pada ketiga gambar di bawah, tampak bahwa Bot hanya menembakkan torpedo terhadap satu target, yaitu lawan terdekat (tengah). Pada gambar ketiga, tampak bahwa torpedo berhasil mengenai lawan.

Kasus 1:

Gambar	Keterangan
	<p>MilkyBot (kanan) menembakkan sejumlah torpedo menuju lawan (tengah gambar)</p>
	<p>Torpedo yang telah ditembakkan mendekati lawan yang dituju</p>



Sebagian dari torpedo yang ditembakkan oleh MilkyBot berhasil mengenai lawan dan mengurangi ukuran lawan sehingga FIRETORPEDOES yang dilakukan **optimum**.


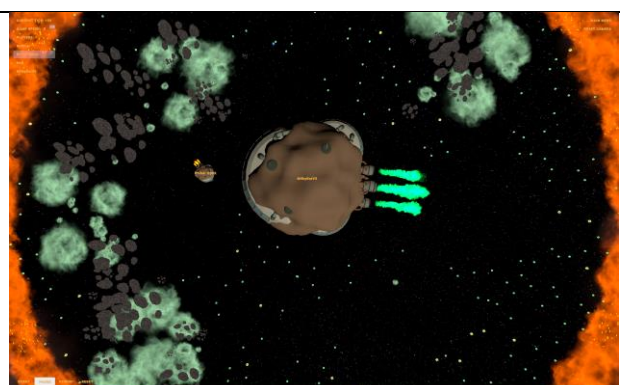
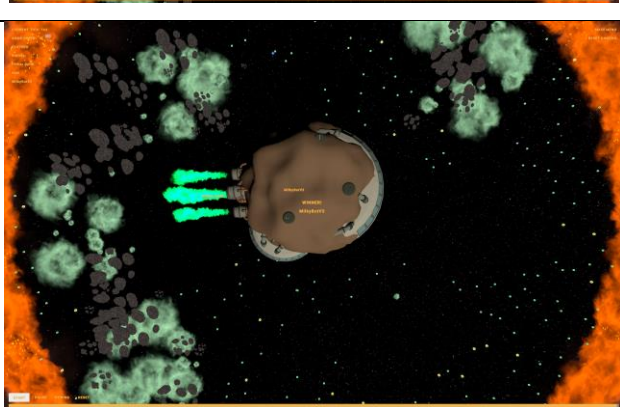
Kasus 2:

Gambar	Keterangan
	MilkyBot menembakkan teleporter.
	Teleporter berada jauh dari sasaran sehingga tidak pernah dilakukan aksi TELEPORT dan pengorbanan size bot menjadi sia-sia. FIRETELEPORT tidak optimum .

4.4.3 Afterburner

Pada ketiga gambar di bawah, Bot (kanan) menetapkan lawan (kiri) sebagai target utama. Pada gambar kedua, Bot menyalakan afterburner (ditandai dengan warna api yang berbeda) untuk mengejar lawan. Pada gambar ketiga, Bot terlihat berhasil mengonsumsi lawan.

Gambar	Keterangan
--------	------------

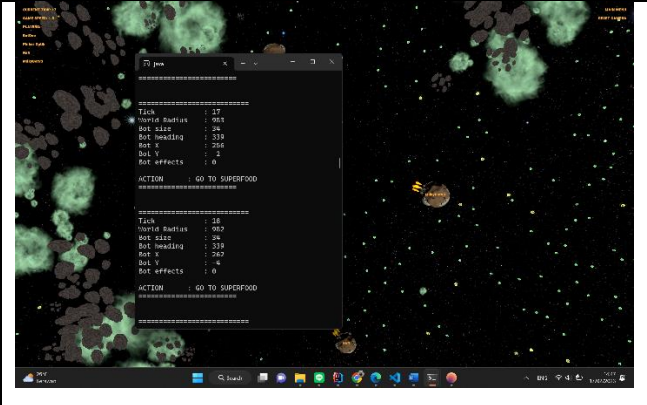
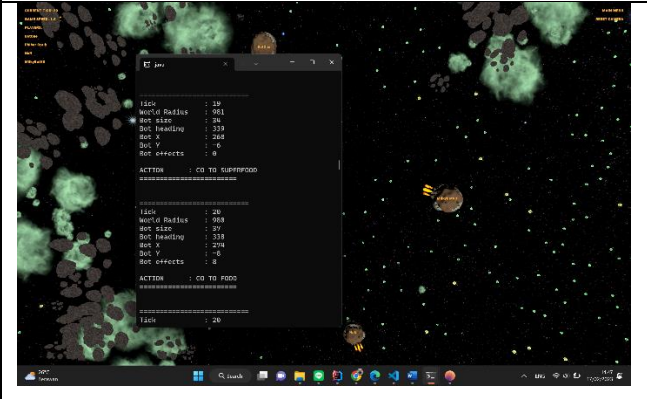
	<p>MilkyBot (kanan) mendeteksi bahwa lawan terdekat (kiri) berada dalam jangkauan penyerangan</p>
	<p>MilkyBot mengaktifkan afterburner untuk mengejar lawan tersebut</p>
	<p>MilkyBot berhasil menangkap lawan sehingga STARTAFTERBURNER dan FORWARD yang dipilih optimum.</p>

4.4.4 Food dan Superfood

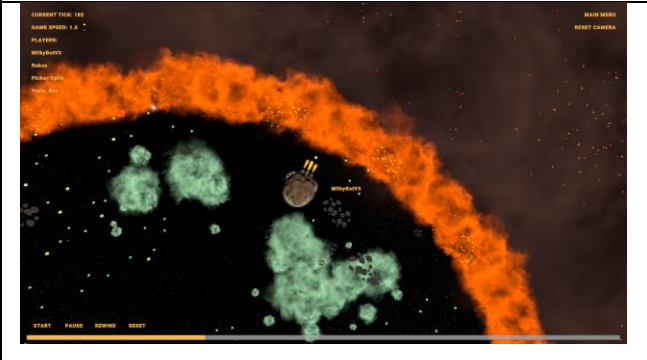
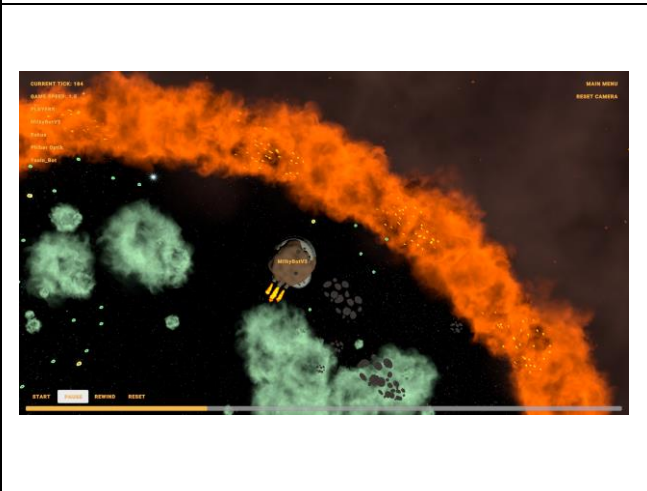
Berdasarkan implementasi, Bot akan mencari food atau super food yang tidak berada pada heading terlarang. Untuk pemilihan makanan, Bot akan memeriksa ketersediaan makanan, efek super food pada Bot, dan jarak. Secara umum, prioritas utama adalah super food.

Kasus 1:

Gambar	Keterangan
--------	------------



	<p>MilkyBot memprioritaskan super food terdekat (tertulis pada terminal)</p>
	<p>MilkyBot memprioritaskan food terdekat (tertulis pada terminal) dan berhasil memakannya sehingga FORWARD dengan heading tersebut optimal.</p>

Kasus 2:

Gambar	Keterangan
	<p>Bot berada dekat dengan world border.</p>
	<p>MilkyBot berusaha memakan makanan yang paling dekat. Namun, karena makanan tersebut dekat dengan world border, pada tick selanjutnya MilkyBot berusaha menjauhi makanan tersebut. Akibatnya, MilkyBot berputar-putar pada lokasi tersebut tanpa berhasil memakan makanan. Aksi dan heading yang dipilih tidak optimum.</p>

4.4.5 Activate Shield

Aktivasi shield pada Bot bergantung pada objek torpedo di sekitar Bot. Apabila torpedo lawan memasuki area yang didefinisikan di sekitar Bot dan Bot memiliki ukuran yang tepat, Bot akan mengaktifkan shield untuk memantulkan torpedo.

Gambar	Keterangan
	Sejumlah torpedo tampak sedang menuju MilkyBot
	MilkyBot mengaktifkan shield untuk mengatasi torpedo sehingga ACTIVATESHIELD optimum.

BAB V

KESIMPULAN DAN SARAN

Dari tugas besar IF2211 Strategi Algoritma ini, kami telah berhasil membuat bot permainan “Galaxio” menggunakan strategi Greedy. Strategi yang digunakan tidak hanya satu, melainkan kombinasi dari beberapa strategi yang memiliki prioritasnya masing-masing sehingga terciptalah sebuah bot dengan strategi terbaik versi Milky Way.

Algoritma greedy terbukti digunakan untuk membuat bot yang cukup *decent* untuk permainan Galaxio karena solusi yang dipilih merupakan optimum lokal. Implementasi algoritma greedy pada MilkyWayBot telah berhasil membuat bot ini memenangkan *match* melawan bot-bot lain secara konsisten. Pada beberapa *match*, bot berhasil mengutamakan *damage to opponent* dan dengan cepat mengalahkan bot-bot lain secara efektif.

Namun, terkadang algoritma greedy gagal menemukan solusi optimum global yang mengakibatkan bot gagal mendapatkan keuntungan terbesar. Beberapa *match* menunjukkan bot gagal berlaku optimum akibat lokasi *spawn* yang kurang menguntungkan. Alternatif lain yang dapat digunakan selain algoritma greedy adalah *exhaustive search* yang mengecek setiap heading dan action serta keuntungan dari aksi tersebut.

Saran pengembangan untuk tugas besar ini adalah:

1. Lebih komunikatif agar ide-ide yang dimiliki dapat dipahami oleh seluruh anggota tim dan terealisasi.
2. Mengerjakan bot tidak mendekati deadline agar bot dapat lebih sempurna.
3. Strategi yang digunakan dapat lebih di optimasi baik dari segi format kode sampai efektivitas kode.

DAFTAR PUSTAKA

[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), terakhir diakses 17 Februari 2023, 12.30

[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), terakhir diakses 17 Februari 2023, 12.31

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf), terakhir diakses 17 Februari 2023, 12.32

<https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md>, terakhir diakses 17 Februari 2023, 12.33

REPOSITORY

https://github.com/chaerla/Tubes1_MilkyWay

YOUTUBE VIDEO

<https://youtu.be/spuyQ7w2kmk>