Tugas Besar I IF2211 Strategi Algoritma

Semester II Tahun 2022/2023

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Galaxio"

Kelompok JAB:

Bill Clinton	13521064
Angela Livia Arumsari	13521094
Jimly Firdaus	13521102

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

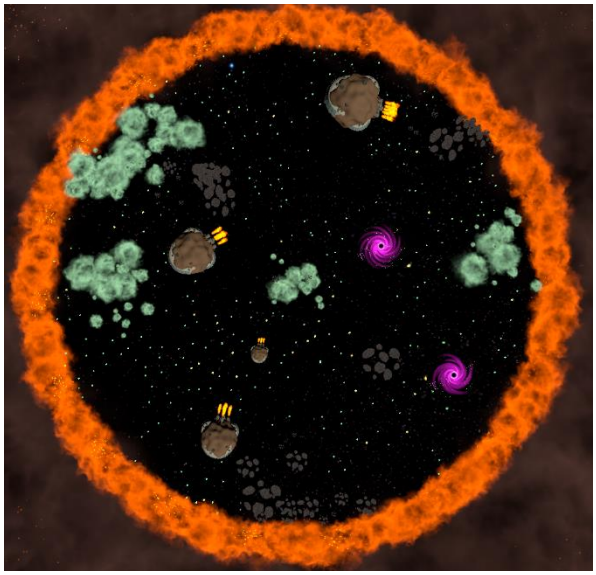
INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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



Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Galaxio. Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan Galaxio dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada starter-bots di dalam starter-pack pada laman github.

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

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada

game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu Players, Food, Wormholes, Gas Clouds, Asteroid Fields. Ukuran peta akan mengecil seiring batasan peta mengecil.

2. Kecepatan kapal dilambangkan dengan x. Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. Heading dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek afterburner akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan oleh kapal player. Apabila player mengkonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakannya dapat meledakannya dan memberi

damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.

8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Untuk daftar commands yang tersedia, bisa merujuk ke tautan panduan di spesifikasi tugas
11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal player lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi.

BAB II
LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma *Greedy* adalah suatu algoritma yang menerapkan konsep “greedy” dalam pemecahan suatu permasalahan. Algoritma ini biasanya digunakan untuk menyelesaikan permasalahan mengenai optimasi. Permasalahan optimasi ini bisa dibagi menjadi dua macam, yaitu maksimasi (*maximization*) dan minimasi (*minimization*). Dalam algoritma *greedy*, pada setiap langkah, diambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensinya dan berharap bahwa pemilihan optimum lokal pada setiap langkah ini akan berakhir dengan optimum global.

Ada beberapa elemen dalam algoritma *greedy*, yaitu himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, serta fungsi objektif. Himpunan kandidat adalah himpunan yang berisi kandidat yang akan dipilih pada setiap langkah. Himpunan solusi adalah himpunan yang berisi kandidat yang telah dipilih. Fungsi solusi adalah fungsi yang menentukan apakah himpunan kandidat yang dipilih telah memberikan solusi. Fungsi seleksi adalah fungsi untuk memilih kandidat berdasarkan strategi *greedy* tertentu. Fungsi kelayakan adalah fungsi untuk memeriksa kelayakan kandidat untuk masuk ke dalam himpunan solusi. Terakhir, fungsi objektif adalah fungsi untuk memaksimalkan atau meminimalkan nilai dalam masalah optimasi.

2.2 Cara Kerja Program *Game*

Untuk menjalankan game dibutuhkan starter pack yang dapat diunduh dari laman github Entelect Challeng 2021. Isi dari arsip *starter pack* adalah beberapa *folder* dan *script* sebagai berikut:

```
.
├── engine-publish
├── logger-publish
├── reference-bot-publish
├── runner-publish
├── starter-bots
├── visualiser
├── building-a-bot.md
```

```
├── README.md
└── run.sh
```

Ada beberapa komponen yang perlu diketahui untuk memahami cara kerja game ini, mereka adalah:

- Engine

  Engine merupakan komponen yang berperan dalam mengimplementasikan logic dan rules game.
- Runner

  Runner merupakan komponen yang berperan dalam menggelar sebuah *match* serta menghubungkan bot dengan engine.
- Logger

  Logger merupakan komponen yang berperan untuk mencatat *log* permainan sehingga kita dapat mengetahui hasil permainan. Log juga akan digunakan sebagai input dari visualizer

Garis besar cara kerja program game Galaxio adalah sebagai berikut:

1. Runner –saat dijalankan– akan meng-*host* sebuah *match* pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
4. 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”.
5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah ReceiveGameState karena memberikan status game.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi *match*.

2.3 Cara Menjalankan *Game*

Berdasarkan gambaran cara kerja program game yang telah disebutkan sebelumnya, berikut merupakan cara menjalankan game secara lokal di 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

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.

2.4 Implementasi Algoritma *Greedy* ke Dalam Bot

Starter pack menyediakan starter bot sebagai titik awal dari pengembangan bot. Starter bot sudah dibekali kode yang dibutuhkan oleh bot untuk melakukan koneksi dengan runner sehingga dapat fokus kepada strategi permainan. Starter-pack mememberikan beberapa pilihan bahasa untuk starter bot. Namun pada tugas besar ini, akan digunakan bot yang berbahasa Java.

Untuk membuat konfigurasi bot sendiri, dapat dimulai dengan menyalin folder JavaBot. Nama bot dapat diatur dengan mengubah string “Coffee Bot” pada file Main.java baris ke-61. Baris 73 sampai baris 77 kode Main.java menunjukkan bahwa bot mengirim aksi ke runner menggunakan metode “send” dengan argumen aksi bot. Aksi ini dihitung dan diperbarui oleh bot menggunakan metode computeNextPlayerAction milik kelas BotService. Algoritma greedy dapat diimplemantisakn pada metode computeNextPlayerAction tersebut.

BAB III
APLIKASI STRATEGI *GREEDY*

Pada penyusunan algoritma *greedy* untuk bot Galaxio, diperlukan mapping dan analisis lebih lanjut mengenai persoalan-persoalan yang ada di dalam permainan Galaxio. Galaxio sendiri adalah game yang memiliki banyak aspek sehingga dapat dipecah menjadi beberapa persoalan. Pada tiap persoalan akan dilakukan eksplorasi alternatif solusi greedy yang mungkin dipilih diikuti analisis efisiensi dan efektivitas dari kumpulan alternatif tersebut. Berdasarkan analisis yang dilakukan, akan dipaparkan strategi greedy yang pada akhirnya dipilih untuk diimplementasikan pada program.

3.1 Persoalan *Boundary Map*

Boundary Map atau batas peta adalah batas berbentuk lingkaran berwarna oranye dalam permainan Galaxio ini. Jika pemain keluar dari batas ini, ukuran kapal pemain akan berkurang drastis dengan cepat. Oleh karena itu, dalam strategi *greedy* kami, kami mendeklarasikan variabel distanceToBoundary untuk mencegah pemain keluar dari batas. Jika pemain berada dalam jarak aman dari *boundary*, pemain dapat melakukan aksi, misalnya farming() atau attack(), sedangkan jika pemain berada terlalu dekat dengan *boundary*, pemain melakukan aksi moveToCenter sehingga pemain menuju ke tengah peta.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari *command attack, farming, FIRETORPEDOES,* dan *moveToCenter, STARTAFTERBURNER, STOPAFTERBURNER*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah aksi yang dipilih sudah sesuai dengan keadaan jarak dengan boundary
Fungsi seleksi	Di awal, jarak antara kapal masih jauh sehingga diprioritaskan memilih aksi farming jika berada pada jarak aman dari

	boundary atau aksi moveToCenter jika berada dekat dengan *boundary*.
Fungsi kelayakan	Memeriksa apakah aksi yang dipilih tepat dan tidak membuat ukuran pemain berkurang secara drastis
Fungsi objektif	Ukuran pemain tidak berkurang secara drastis

Alternatif solusi untuk persoalan ini yaitu bot dapat pergi ke tengah dengan diiringi aksi mencari *FOOD* dan *SUPERFOOD*. Dengan demikian, bot bisa mendapatkan memperoleh keuntungan tambahin selain berusaha untuk tetap dalam peta (tidak keluar *boundary*).

3.2 Persoalan Menghindari *Gas Clouds* dan *Asteroids*

Gas Clouds dan *Asteroids* adalah objek permainan yang ada pada permainan Galaxio. *Gas Clouds* dapat menyebabkan ukuran pemain mengecil setiap *tick*, sedangkan *Asteroids* dapat menyebabkan kecepatan kapal pemain menjadi lambat. Dalam strategi *greedy* kami, kami menganggap *Gas Clouds* dan *Asteroids* sebagai *harmful objects* yang harus dihindari.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari *command attack, farming, FIRETORPEDOES*, dan *moveToCenter, STARTAFTERBURNER, STOPAFTERBURNER*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah aksi yang dilakukan sudah tepat untuk bisa menghindari *harmful objects*
Fungsi seleksi	Pilih aksi berdasarkan banyak *harmful objects* di sekitar (menggunakan *countSurroundHarmfulObjects)* Jika banyak *harmful objects* adalah 0, bot diprioritaskan untuk menjalankan *command farming*, sedangkan jika lebih

	dari 1, bot akan diprioritaskan untuk mengganti *heading* sehingga bisa menghindari *harmful objects*
Fungsi kelayakan	Memeriksa apakah aksi yang terpilih dapat membuat bot menghindari *harmful objects*
Fungsi objektif	Bot berhasil menghindari *harmful objects*

Alternatif solusi untuk persoalan ini adalah bot dapat menembakkan *torpedo salvo* ke *harmful objects* untuk memperkecil *harmful objects*. *Torpedo salvo* ini dapat memperkecil objek yang ditabraknya. Alternatif ini tidak kami pilih karena *harmful objects* ini sebenarnya berguna ketika ada lawan yang memiliki ukuran lebih besar dari kapal bot ingin mengejar kapal bot. *Harmful objects* dapat memperlambat atau mengurangi ukuran kapal lawan.

3.3 Persoalan Menghindari *Torpedo Salvo*

Torpedo Salvo adalah salah satu objek dalam permainan Galaxio. *Torpedo Salvo* ini dapat berasal dari kapal sendiri maupun kapal lawan. *Torpedo Salvo* berjalan dalam lintasan yang lurus serta dapat menghancurkan objek yang berada di lintasannya. *Torpedo Salvo* ini juga akan mengurangi ukuran kapal lawan ketika bertabrakan dengan kapal lawan. Oleh karena itu, dalam strategi *Greedy* kami, bot kami akan menghindari *Torpedo Salvo* atau menggunakan *shield* dengan menjalankan *command ACTIVATESHIELD*.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Himpunan aksi yang terdiri dari aksi penggantian *heading* atau aksi untuk menjalankan *command ACTIVATESHIELD*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah aksi yang dipilih telah sesuai dengan jarak yang terdeteksi antara kapal dengan *torpedo salvo*.
Fungsi seleksi	Pilih aksi berdasarkan jarak kapal dengan t*orpedo salvo* yang terdeteksi

	Jika jaraknya masih aman, bot diprioritaskan untuk melaksanakan penggantian *heading* (dikombinasikan dengan command *FORWARD),* sedangkan jika jaraknya dekat, bot diprioritaskan untuk melaksanakan *command ACTIVATESHIELD*
Fungsi kelayakan	Memeriksa apakah aksi yang dipilih dapat membuat bot terhindar dari tabrakan dengan t*orpedo salvo*
Fungsi objektif	Bot berhasil menghindari tabrakan dengan *torpedo salvo*

Alternatif solusi untuk persoalan ini adalah bot dapat menembakkan *torpedo salvo* untuk menghancurkan *torpedo salvo* yang ditembakkan lawan. Alternatif ini tidak kami pilih karena ketika menembakkan *torpedo salvo* untuk menghadapi banyak *torpedo salvo* yang mungkin berasal dari arah yang berbeda-beda, penembakan *torpedo salvo* bisa tidak akurat sehingga kapal bot masih terkena *torpedo salvo*. Selain itu, penembakan yang sia-sia jelas merugikan karena penembakan *torpedo salvo* juga mengurangi ukuran kapal bot sendiri.

3.4 Persoalan Mengantisipasi *Ships* Lain

Pemetaan Elemen/Komponen Algoritma *Greedy* pada persoalan *bot* berada di dekat *ships* lain

Dalam permainan *Galaxio*, persoalan yang juga cukup penting adalah bagaimana *bot* respons dari *bot* jika ada *ships* lain di dekatnya, baik berukuran lebih kecil maupun berukuran lebih besar. Hal ini akan berguna karena dapat mencegah *collision* dengan *ship* berukuran lebih besar dan memampukan *bot* untuk dapat bereaksi jika terdapat *ship* berukuran lebih kecil darinya (reaksi bukan hanya untuk *ship* lain yang berukuran lebih besar).

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari *command FIRETORPEDOES, STARTAFTERBURNER,* atau tidak melakukan kedua *command* tersebut

	melainkan melakukan *farming* untuk setiap *tick* gamenya
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah *command* yang dipilih membuat *bot* berjalan / berpindah ke daerah yang lebih aman atau melakukan *attack* jika syarat *attack* dipenuhi
Fungsi seleksi	Memilih *command* yang paling menguntungkan posisi *bot* berdasarkan keadaan pada *tick* saat itu
Fungsi kelayakan	Memeriksa apakah *command* yang dipilih oleh *bot* merupakan *command* yang *valid. Command-command* yang *valid* pada strategi ini adalah *command FIRETORPEDOES, STARTAFTERBURNER,* atau *bot* tidak melakukan kedua *command* tersebut (*farming*)
Fungsi objektif	Mencari *command* yang membuat *bot* dapat mengeliminasi *ships* lain atau berhasil menghindar dari *ships* lain

Terdapat beberapa alternatif solusi *greedy* pada persoalan ini. Alternatif *greedy* pertama yang dapat diimplementasikan adalah dengan mempertimbangkan setiap *size ships* lain yang sedang mendekati kita. Jika *size ships* yang mendekati *bot* memiliki *size* yang lebih kecil, *bot* dapat lansung mengejar *ships* tersebut tanpa harus mengecek jika *ships* tersebut masih di dalam batas aman *boundary*, mengingat setiap *ships* seharusnya sudah diprogram untuk tetap berada di dalam batas aman *boundary.*

Alternatif *greedy* yang kedua adalah yang dapat digunakan adalah dengan menembak *torpedo salvo* untuk setiap *ships* yang mendekati *bot.* Dengan hal ini, kita tidak perlu terlalu mempertimbangan perbedaan ukuran antara *ships* lain dengan *bot.* Kita hanya perlu

mempertimbangkan *threshold* aman untuk tetap dapat menembak *torpedo salvo*. Hal ini dapat dilakukan karena untuk setiap *torpedo salvo* yang berhasil mengenai *ships* lain, maka kita akan mendapatkan pertambahan *size*, sehingga ini bisa dianggap sebagai *exchange size* jika *ships* yang sedang mengejar *bot* berukuran lebih besar.

3.5 Persoalan Menembakkan *Teleporter*

Pemetaan Elemen/Komponen Algoritma *Greedy* pada persoalan *bot* menembakkan *teleporter*

Salah satu *skill* penting dalam game *Galaxio* adalah *teleporter*. Setiap *ship* memiliki *teleporter* dengan jumlah tertentu. *Teleporter* memungkinkan *ship* dapat berpindah tempat secara instan. Oleh karena itu dengan memanfaatkan *teleporter* dengan tepat akan sangat membantu *bot* dalam memenangkan permainan.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari *command* *FIRETELEPORT* dan *TELEPORT* atau tidak melakukan kedua *command* tersebut per *tick* gamenya
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah *command* yang dipilih oleh *bot* berhasil menembakkan proyektil *teleporter* dan melakukan *teleport* jika memenuhi syarat untuk melakukan *command fireTeleport* dan *teleport*
Fungsi seleksi	Memilih *command* yang paling menguntungkan posisi *bot* berdasarkan keadaan pada *tick* saat itu
Fungsi kelayakan	Memeriksa apakah *command* yang dipilih oleh *bot* merupakan *command* yang *valid. Command-command* yang *valid* pada strategi ini adalah *command* *FIRETELEPORT, TELEPORT,* atau *bot*

	tidak melakukan kedua *command* tersebut
Fungsi objektif	Mendapatkan *heading* yang tepat apabila proyektil teleport sudah ditembak dan melakukan *teleport* jika proyektil sudah berjarak tertentu dari *ship* lawan

Terdapat beberapa alternatif solusi untuk persoalan ini. Alternatif *greedy* yang pertama adalah dengan menembak *teleporter* (jika masih memiliki *teleport*) apabila ada *ships* yang memiliki ukuran yang jauh lebih rendah dibandingkan dengan *bot. Threshold* untuk menembak *teleporter* dapat dihitung melalui perbedaan *size* yang aman.

Alternatif *greedy* yang kedua adalah dengan menggunakan *teleporter* sebagai *escape* jika *bot* sudah dalam keadaan “terapit” oleh objek-objek berbahaya maupun oleh *ships* lain, dengan mempertimbangkan *size* dari *bot*.

3.6 Persoalan Mengantisipasi Objek-Objek Berbahaya

Pada persoalan ini, *bot* diperhadapkan dengan beberapa situasi:

1. *Bot* bertemu dengan kombinasi *gas clouds, asteroid* dan *ship*

   Salah satu permasalahan pada arah gerak dari *bot* adalah ketika *bot* berada dekat dengan *gas clouds, asteroid* dan *ship* secara bersamaan pada jarak yang ditentukan. Oleh karena itu dibutuhkan algoritma untuk mengetahui dimana *heading* dari *bot* yang aman agar terhindar dari hal tersebut.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari arah *heading* dari *bot* untuk setiap *tick* gamenya
Himpunan solusi	*Heading bot* yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah permutasi dari *heading bot* mengarah pada sudut yang seharusnya (menghindari objek-objek berbahaya)

Fungsi seleksi	Memilih arah *heading* yang aman bagi *bot* pada kondisi *tick* saat itu
Fungsi kelayakan	Memeriksa apakah arah *heading* yang dipilih oleh *bot* merupakan arah *heading* yang *valid*. Arah *heading* yang *valid* pada strategi ini adalah berputar ke arah lain yang tidak terdapat objek-objek berbahaya
Fungsi objektif	Mendapatkan *heading* untuk *bot* yang aman dari objek-objek berbahaya

Terdapat beberapa alternatif solusi untuk persoalan ini. Alternatif *greedy* yang pertama adalah dengan mengabaikan *asteroid fields* sebagai objek yang berbahaya karena hanya memberikan *effect slow* sehingga jika dalam kondisi dimana *bot* dalam keadaan “terapit” oleh banyak objek, maka *asteroid fields* dapat dijadikan sebagai alternatif untuk *escape* dari keadaan tersebut.

Alternatif *greedy* yang kedua adalah dengan menggunakan *torpedo salvo* untuk menghilangkan *gas clouds* jika *bot* dalam keadaan “terapit” dengan beberapa *ships* dan juga beberapa *gas clouds* sehingga dengan menembak *gas clouds*, maka *bot* dapat menggunakan *space* akibat *size gas clouds* yang mengecil sebagai alternatif untuk *escape* dari kejaran *ships* lain.

2. *Bot* bertemu dengan kombinasi *gas clouds, asteroid* dan *torpedo salvo*

   Subpermasalahan berikutnya adalah ketika *bot* bertemu dengan kombinasi posisi dari *gas clouds, asteroid,* dan *torpedo salvo.* Algoritma *greedy* kami memprioritaskan keberadaan dari *torpedo salvo* karena *torpedo salvo* dapat menguntungkan *ships* lain jika berhasil mengenai *bot.*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Permutasi dari *command FORWARD, ACTIVATESHIELD,* dan arah *heading bot* untuk setiap *tick* gamenya

Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah permutasi dari *command FORWARD, ACTIVATESHIELD,* dan arah *heading* membuat *bot* aman dari *torpedo salvo* lawan
Fungsi seleksi	Memilih *command* yang paling menguntungkan bagi *bot* pada *tick* tersebut dengan mempertimbangan *size bot*
Fungsi kelayakan	Memeriksa apakah *command* atau *heading* dari *bot valid. Command* yang *valid* pada strategi ini adalah *FORWARD, ACTIVATESHIELD,* atau *bot* hanya berputar ke *heading* yang lain, menghindari *torpedo salvo*
Fungsi objektif	*Bot* mendapatkan *damage* sekecil mungkin dari perumtasi *command-command* yang *valid*

Terdapat alternatif solusi untuk persoalan ini. Alternatif *greedy* yang dapat diimplementasikan adalah karena prioritas dari *bot* adalah berusaha untuk tidak menerima *damage* dari *torpedo salvo,* maka selain menghindar ke arah lain atau menggunakan *shield,* bot juga dapat meng-*counter* setiap *torpedo salvo* yang datang sehingga *shield* hanya dipakai jika *bot* ditembak dari berbagai arah, mengingat adanya *shield* memiliki *cooldown*.

3. *Bot* bertemu dengan kombinasi *ships* dan *boundary map*

   Subpermasalahan berikutnya adalah ketika *bot* didekati oleh *ships* dan sedang berada di dekat *boundary map.* Algoritma *greedy* kami akan menentukan dimana seharusnya *heading bot* berada untuk menghindari *collision* dengan *ships* yang lebih besar sekaligus tetap menjaga posisi agar tidak keluar dari *boundary* atau mengejar *ships*

yang lebih kecil (*method attack()*) dengan tetap berada di area aman (tidak keluar dari *boundary map*).

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari *command FIRETORPEDOES, STARTAFTERBURNER, FORWARD* dan *arah heading* dari *bot*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah permutasi dari *command FIRETORPEDOES, STARTAFTERBURNER, FORWARD* dan *heading* dari *bot* dapat membuat *bot* terhindar dari kejaran *boundary* dan *ships* yang berukuran lebih besar atau dapat mengeliminasi *ships* lain yang berukuran lebih kecil serta tetap berada di dalam *boundary map*
Fungsi seleksi	Memilih *command* yang paling menguntungkan *bot* pada *tick* saat itu dengan mempertimbangan posisi relatif terhadap *boundary* dan *size ships* yang berdekatan
Fungsi kelayakan	Memeriksa apakah *command* atau *heading* dari *bot valid. Command* yang *valid* pada strategi ini adalah *FIRETORPEDOES, STARTAFTERBURNER, FORWARD,* bergantung pada kondisi dari *bot* pada saat *tick* tersebut
Fungsi objektif	*Bot* mampu menghindari *ships* yang lebih besar serta tetap berada di dalam *boundary* atau berhasil mengeliminasi

	ships yang lebih kecil dan tetap berada di dalam *boundary*

4. *Bot* bertemu dengan kombinasi *gas clouds, asteroid* dan *boundary map*

   Pada subpermasalahan ini, *bot* diperhadapkan dengan *gas clouds, asteroids,* dan sedang posisi *bot* sedang berada di dekaat *boundary map.* Algoritma *greedy* kami akan menentukan dimana seharusnya arah *heading* dari *bot* untuk menghindari *collision* dengan objek-objek berbahaya serta tetap berada di dalam *boundary.*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Permutasi dari *command STARTAFTERBURNER, STOPAFTERBURNER, FORWARD,* dan aksi penggantian *heading*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah *command* yang dipilih dapat membuat bot terhindar dari *gas clouds* dan *asteroids*, serta menjauhi *boundary map* atau batas peta
Fungsi seleksi	Memilih *command* yang membuat *heading* dari *bot* menjauhi arah objek-objek berbahaya dan tetap berada di dalam *boundary*
Fungsi kelayakan	Memeriksa apakah *command* yang dijalankan paling menguntungkan dengan mempertimbangkan posisi relatif terhadap *boundary* serta jarak ke *harmful objects*
Fungsi objektif	Pemain terhindari dari *gas clouds* dan *asteroids*, serta menjauhi *boundary map* atau batas peta

Terdapat alternatif solusi untuk persoalan ini. Algoritma *greedy* yang dapat digunakan adalah dengan mengabaikan *asteroid fields* sebagai objek yang berbahaya ketika *bot* sudah dekat dengan *boundary*. Hal ini akan membuat *bot* akan lebih mudah mencari jalan keluar dari kondisi tersebut karena mengurangi jumlah objek yang harus dihindari, mengingat bahwa *asteroid fields* hanya memberikan efek *slow* kepada *bot* sehingga *asteroid fields* dapat dijadikan sebagai *escape route* untuk *bot.*

5. *Bot* bertemu dengan kombinasi *gas clouds, asteroids, ship* dan *torpedo salvo*
   Pada subpermasalahan ini, *bot* diperhadapkan dengan *gas clouds, asteroids, torpedo salvo,* dan *ship* lain. Karena prioritas yang kami buat berdasarkan objek-objek yang harus dihindari, maka algoritma *greedy* kami akan membuat *bot* untuk berusaha menghidari *torpedo salvo* terlebih dahulu. Karena *threshold* jarak dengan *torpedo* yang relatif lebih kecil dibandingkan dengan jarak dengan lawan, maka *bot* akan menghindari *ships* lain yang berukuran lebih besar setelah berhasil mengatasi *torpedo salvo* yang menuju ke posisinya. Kemudian *bot* akan bergerak sesuai dengan kondisi pada *tick* game saat itu.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Permutasi dari *command STARTAFTERBURNER, STOPAFTERBURNER, FORWARD, ACTIVATE SHIELD, farming, attack,* serta aksi penggantian *heading*
Himpunan solusi	Aksi yang terpilih berdasarkan kondisi *tick* saat itu
Fungsi solusi	Memeriksa apakah *command* yang dipilih dapat membuat bot terhindar dari *gas clouds, asteroids, ship* yang ukurannya lebih besar, menjauhi *boundary map* atau batas peta, atau menabrak kapal lain yang memiliki ukuran lebih kecil.
Fungsi seleksi	Pilih aksi yang paling tepat dengan mempertimbangkan jarak dengan

	harmful objects, ukuran kapal lain, serta posisi relatif terhadap *boundary*.
Fungsi kelayakan	Memeriksa apakah aksi yang dipilih sudah paling menguntungkan dengan mempertimbangkan
Fungsi objektif	Bot bisa tetap dalam map (tidak keluar *boundary*), bisa menabrak kapal lain yang memiliki ukuran lebih kecil, serta terhindar dari *harmful objects* dan kapal lain yang memiliki ukuran lebih besar

Terdapat alternatif solusi untuk persoalan ini. Alternatif *greedy* yang dapat digunakan adalah dengan menggunakan *teleport.* Karena dalam kondisi ini *bot* harus menghindari banyak objek termasuk *ships,* maka *teleport* dapat digunakan untuk berpindah ke daerah yang lebih aman, mengigat teleport tidak dapat bertabrakan dengan objek apapun sehingga meningkatkan peluang *bot* untuk bertahan jika *bot* sedang dikejar oleh *ships* yang berukuran jauh lebih besar.

3.7 Strategi *Greedy* yang Digunakan pada *Bot*

Strategi *greedy* yang digunakan oleh kami adalah kombinasi dari seluruh strategi *greedy* yang telah dipaparkan pada subbab sebelumnya (penyelesaian setiap persoalan yang mungkin terjadi pada *bot*). Setelah melawan *reference bot*, kami mempertimbangkan bagaimana urutan dan prioritas dari objek-objek yang terdapat pada game *Galaxio* ini untuk menentukan alur dari algoritma *greedy* yang telah kami buat. Urutan pengeksekusiannya adalah sebagai berikut:

- Strategi *torpedo defense.* Setiap *tick* game akan dilakukan pengecekan apakah ada *torpedo salvo* yang mengarah ke *bot.* Hal ini diprioritaskan karena *torpedo salvo* dapat ditembakkan dalam jumlah banyak (lebih dari 1) sehingga berpotensi untuk lansung mengeliminasi *bot* dari permainan.
- Strategi *attack.* Apabila terdapat *ships* yang berdekatan dengan *bot*, dengan ukuran yang lebih kecil maka *bot* akan menembakkan *torpedo salvo,* dan jika jaraknya semakin dekat, maka *bot* akan mengganti *action* menjadi *STARTAFTERBURNER* dan mengejar *ship* tersebut. Jika tersisa 1 *ship* lawan dan *bot* memiliki *size* yang lebih besar, maka *bot* akan mengaktifkan *aggressive attacking* pada *attack*, yang

membuat *bot* menembakkan proyektil *teleporter* ke arah *ship* lawan dan akan melakukan *teleport* jika proyektil berhasil mendekati *ship* lawan.

- Strategi menghindari *ship* yang berukuran yang lebih besar. Jika masih dalam jarak yang tidak terlalu dekat, *bot* akan mengubah *heading*-nya untuk menjauh dari *ships* tersebut. Jika *ship* semakin dekat, maka *bot* akan menembakkan *torpedo salvo* ke arah *ship* tersebut (*exchange size*).
- Strategi menghindari objek - objek berbahaya (*gas clouds* dan *asteroid fields*). *Bot* akan lansung mengubah *heading*-nya menuju arah yang tidak terdapat objek-objek berbahaya.
- Strategi *farming*. *Bot* akan melakukan *farming* yaitu mencari makan dengan *superfood* sebagai prioritas.
- Strategi *move to center*. *Bot* akan lansung mengubah *heading*-nya menuju pusat *map* dan bergerak ke arah pusat *map* agar tetap dalam batasan *boundary*.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy*

Pada program bot, diimplementasikan beberapa strategi greedy untuk tiap kondisi dari game. Dalam pengimplementasiannya, dibuat juga beberapa method tambahan untuk reusability dari kode. Implementasi detail yang disertakan hanya prosedur utama yang digunakan, yaitu computeNextPlayerAction.

```
procedure computeNextPlayerAction(input playerAction: PlayerAction)
{   I.S. playerAction sembarang
    F.S. set PlayerAction bot dengan playerAction yang terdefinisi }

KAMUS LOKAL
Random random <- new Random()
int INT_MAX <- Integer.MAX_VALUE

{ Inisialisasi data dari Game State }
List<GameObject> gameObjects <- this.gameState.getGameObjects()
List<GameObject> playerGameObjects <- this.gameState.getPlayerGameObjects()
World worldObjects <- this.gameState.getWorld()

{ Inisialisasi list yang berisi himpunan kandidat greedy dengan urutan berdasarkan jarak
ke bot }
List<GameObject> playerList <-playerGameObjects { Filter and sort player }
List<GameObject> foodList <- gameObjects { Filter and sort object food }
List<GameObject> gasCloudsList <- gameObjects{ Filter object gas clouds }
List<GameObject> asteroidsList <- gameObjects{ Filter object asteroid }
List<GameObject> torpedosList <- gameObjects { Filter object torpedo }

int currentSize <- this.bot.getSize()
double distanceToBoundary <- getDistanceBetween(this.bot, worldObjects) - currentSize
double  nearestShips  <-  (playerList.size()  >  1)  ?  (getDistanceBetween(this.bot,
playerList.get(1)) - currentSize - ((GameObject)playerList.get(1)).getSize()) : INT_MAX
double nearestGasCloud <- (gasCloudsList.size() > 0) ? (getDistanceBetween(this.bot,
gasCloudsList.get(0)) - currentSize - ((GameObject)gasCloudsList.get(0)).getSize()) :
INT_MAX
double nearestAsteroid <- (asteroidsList.size() > 0) ? (getDistanceBetween(this.bot,
asteroidsList.get(0)) - currentSize - ((GameObject)asteroidsList.get(0)).getSize()) :
INT_MAX
int totalHarmfulObjects <- countSurroundHarmfulObjects(nearestGasCloud, nearestAsteroid)
boolean nearBoundary <- (distanceToBoundary <= 40.0)
GameObject nearestObstacle

ALGORITMA
{ Inisialisasi aksi default }
forward()
if (not this.gameState.getGameObjects().isEmpty()) then
    { Prioritas: Melakukan cek torpedo yang perlu dihindari }
    torpedoDefense(torpedosList)

    { Melanjutkan pengecekkan hanya jika tidak ada torpedo yang perlu      dihindari }
    if (torpedoHit == 0) then
        if (nearestShips <= 350.0) then
              { Jika ada kapal musuh dalam jarak tertentu dan tidak dekat boundary map }
            if (not nearBoundary) then
                depend on (totalHarmfulObjects):
```

```
                    { Tidak ada objek berbahaya di sekitar bot }
                    totalHarmfulObjects = 0:
                              { Attack jika kapal terdekat size lebih kecil }
                        if (currentSize-((GameObject)playerList.get(1)).getSize()>5) then
                            playerAction <- attack()
                            break

                    { Ketika kapal terdekat lebih besar tapi jauh, lanjut mencari makan }
                        if (nearestShips > 200.0) then
                            farming()
                            stopAfterburner()
                            break
                        stopAfterburner()

                    { Ketika kapal terdekat lebih besar tapi dekat, serang kapal/kabur }
                        if (nearestShips > 0.5) then
                            playerAction.heading <- getHeadingBetween
                                  (playerList.get(1)) + random.nextInt(4) + 90
                            if (nearestShips < 120.0) then
                                if (nearestShips < 80.0) then
                                       playerAction.heading <-
                                            getHeadingBetween(playerList.get(1))
                                    if (this.bot.getSize() >= 30)
                                        playerAction.action <-
                                            PlayerActions.FIRETORPEDOES
                                    break
                                startAfterburner()
                                break
                            stopAfterburner()
                        break
                    { Ada 1 objek berbahaya di sekitar bot }
                    totalHarmfulObjects = 1:
                        nearestObstacle <- (nearestGasCloud < nearestAsteroid) ?
                                        gasCloudsList.get(0) : asteroidsList.get(0)

                        { Melakukan attack jika kapal terdekat size lebih kecil }
                        if (currentSize - ((GameObject)playerList.
                              get(1)).getSize() > 3) then
                            playerAction <- attack()
                            break
                        stopAfterburner()

                        { Jika kapal terdekat masih jauh, maka hindari objek berbahaya }
                        if (nearestShips > 150.0) then
                            playerAction.heading <- getHeadingBetween(nearestObstacle) *
                              -1 + random.nextInt(4) + 6
                            break

                        { Jika kapal terdekat size lebih besar, maka attack/kabur }
                        if (nearestShips > 0.5) then
                            playerAction.heading <- (getHeadingBetween(playerList.get(1))
                              + getHeadingBetween(nearestObstacle)) / -2 mod 360 +
                              random.nextInt(4) + 3
                            if (nearestShips < 120.0) then
                                if (nearestShips < 80.0) then
                                    playerAction.heading <-
                                       getHeadingBetween(playerList.get(1))
                                    if (this.bot.getSize() >= 30)
                                        playerAction.action <-
                                            PlayerActions.FIRETORPEDOES
                                    break
                                startAfterburner()
                                break
                            stopAfterburner()
                        break
                    { Ada 2 objek berbahaya di sekitar bot }
```

```
                    totalHarmfulObjects = 2:
                        { Melakukan attack jika kapal terdekat size lebih kecil }
                        if (currentSize-((GameObject)playerList.get(1)).getSize()>3) then
                            playerAction <- attack()
                            break

                        { Jika ada kapal sangat dekat dan size lebih besar, maka attack }
                        if (nearestShips < 80.0) then
                            playerAction.heading <- getHeadingBetween(playerList.get(1))
                            if (this.bot.getSize() >= 30)
                                playerAction.action <- PlayerActions.FIRETORPEDOES
                            break
                        stopAfterburner()

                        { Hindari objek berbahaya }
                        playerAction.heading <- ((getHeadingBetween(asteroidsList.get(0))
                        + getHeadingBetween(gasCloudsList.get(0)) +
                        getHeadingBetween(playerList.get(1))) / -3 + random.nextInt(4) +
                        3) mod 360
                        break
                { Jika ada kapal musuh dalam jarak tertentu dan dekat boundary map }
                  else
                  depend on (totalHarmfulObjects)
                      { Tidak ada objek berbahaya di sekitar bot }
                      totalHarmfulObjects = 0:
                          { Melakukan attack jika kapal terdekat size lebih kecil dan jarak
                          ke boundary masih memungkinkan attack }
                          if (currentSize-((GameObject)playerList.get(1)).getSize()>3) then
                              if (distanceToBoundary > 10.0) then
                                  playerAction <- attack()
                                  break
                              moveToCenter()
                              stopAfterburner()
                              break

                          { Jika ada kapal musuh lebih besar, mengamankan diri dari
                          boundary/attack }
                          if (nearestShips > 150.0) then
                              moveToCenter()
                              else if (nearestShips > 0.5) then
                              if (nearestShips < 80.0) then
                                  playerAction.heading <-
                                    getHeadingBetween(playerList.get(1))
                                  if (this.bot.getSize() >= 30)
                                      playerAction.action <- PlayerActions.FIRETORPEDOES
                                  else if
                                  playerAction.heading <-
                                    (getHeadingBetween((GameObject)playerList.get(1)) <
                                    15) ? ((getHeadingBetween(playerList.get(1)) < 0)
                                    ? ((getHeadingBetween(playerList.get(1)) +
                                    getHeadingBetween()) / -2 + 15) :
                                    ((getHeadingBetween(playerList.get(1)) +
                                    getHeadingBetween()) / -2 - 15)) :
                                    (((getHeadingBetween(playerList.get(1)) +
                                    getHeadingBetween()) / -2 + random.nextInt(4) + 3) mod
                                    360)
                          stopAfterburner()
                          break
                      { Ada 1 objek berbahaya + dekat dengan boundary map }
                      totalHarmfulObjects = 1:
                          nearestObstacle <- (nearestGasCloud < nearestAsteroid) ?
                          gasCloudsList.get(0) : asteroidsList.get(0)
                          { Melakukan attack jika kapal terdekat size lebih kecil dan jarak
                          ke boundary + objek berbahaya masih memungkinkan attack }
                          if (currentSize-((GameObject)playerList.get(1)).getSize()>3) then
                              if (distanceToBoundary > 30.0) then
```

```
                            playerAction <- attack()
                            break
                        stopAfterburner()
                        if (Math.abs(getHeadingBetween() –
                          getHeadingBetween(nearestObstacle)) > 30) then
                            playerAction.heading <- (getHeadingBetween() +
                              getHeadingBetween(nearestObstacle)) / 2 +
                              random.nextInt(4) + 90
                            break
                        playerAction.heading += 15
                        break

                    { Jika ada kapal musuh lebih besar, mengamankan diri dari boundary
                    + obstacle/attack }
                    if (nearestShips > 150.0) then
                        if (Math.abs(getHeadingBetween() –
                          getHeadingBetween(nearestObstacle)) > 30) then
                            playerAction.heading <- (getHeadingBetween() +
                              getHeadingBetween(nearestObstacle)) / 2 +
                              random.nextInt(4) + 3
                            else if
                            playerAction.heading += 15
                        else if (nearestShips > 0.5) then
                        if (nearestShips < 80.0) then
                            playerAction.heading <-
                              getHeadingBetween(playerList.get(1))
                            if (this.bot.getSize() >= 30)
                                playerAction.action <- PlayerActions.FIRETORPEDOES
                        else if
                            playerAction.heading <-
                              (getHeadingBetween(playerList.get(1)) * -1 +
                              getHeadingBetween() +
                              getHeadingBetween(nearestObstacle) + random.nextInt(4)
                              + 3) mod 360
                    stopAfterburner()
                    break
                { Ada 2 objek berbahaya + dekat boundary map }
                totalHarmfulObjects = 2:
                    { Melakukan attack jika kapal sangat dekat, selain itu menghindar
                    dari 2 objek + boundary }
                    if (nearestShips < 80.0) then
                        playerAction.heading <- getHeadingBetween(playerList.get(1))
                        if (this.bot.getSize() >= 30)
                            playerAction.action <- PlayerActions.FIRETORPEDOES
                        else if
                        playerAction.heading <-
                          ((getHeadingBetween(asteroidsList.get(0)) +
                          getHeadingBetween(gasCloudsList.get(0)) +
                          getHeadingBetween() + getHeadingBetween(playerList.get(0)))
                          / -4 + random.nextInt(4) + 3) mod 360
                    stopAfterburner()
                    break

            else if
            { Jika kapal musuh masih jauh dan tidak dekat boundary }
            if (not nearBoundary) then
                depend on (totalHarmfulObjects):
                    { Tidak ada objek berbahaya }
                    totalHarmfulObjects = 0:
                        { Greedy by distance untuk farming dengan syarat makanan tidak
                        dekat dengan objek berbahaya }
                        if (nearestGasCloud - getDistanceBetween(this.bot,
                        foodList.get(0)) - currentSize > 15.0 || nearestAsteroid –
                        getDistanceBetween(this.bot, foodList.get(0)) - currentSize >
                        15.0) then
                            farming()
```

```
                            break
                        moveToCenter()
                        break
                    { Menghindari 1 objek berbahaya }
                    totalHarmfulObjects = 1:
                        playerAction.heading <- (nearestGasCloud > nearestAsteroid) ?
                        (getHeadingBetween(asteroidsList.get(0)) + random.nextInt(4) + 3
                        + 180) : (getHeadingBetween(gasCloudsList.get(0)) +
                        random.nextInt(4) + 3 + 180)
                        break
                    { Menghindari 2 objek berbahaya }
                    totalHarmfulObjects = 2:
                        if (getDistanceBetween(asteroidsList.get(0),
                        gasCloudsList.get(0)) –
                        ((GameObject)asteroidsList.get(0)).getSize() –
                        ((GameObject)gasCloudsList.get(0)).getSize() > (currentSize +
                        15)) then
                            playerAction.heading <-
                              (getHeadingBetween(asteroidsList.get(0)) +
                              getHeadingBetween(gasCloudsList.get(0))) / 2 +
                              random.nextInt(4) + 3
                            break
                        playerAction.heading <- ((getHeadingBetween(asteroidsList.get(0))
                        + getHeadingBetween(gasCloudsList.get(0))) / -2 +
                        random.nextInt(4) + 3) mod 360
                        break
            { Jika kapal musuh masih jauh dan dekat boundary }
            else if
                depend on (totalHarmfulObjects):
                    { Tidak ada objek berbahaya, kabur dari boundary }
                    totalHarmfulObjects = 0:
                        moveToCenter()
                        break
                    { Menghinddari 1 objek berbahaya + boundary }
                    totalHarmfulObjects = 1:
                        playerAction.heading <- (nearestGasCloud > nearestAsteroid) ?
                        ((getHeadingBetween(asteroidsList.get(0)) + getHeadingBetween() +
                        360) / 3) : ((getHeadingBetween(gasCloudsList.get(0)) +
                        getHeadingBetween() + 360) / 3)
                        break
                    { Menghindari 2 objek berbahaya + boundary }
                    totalHarmfulObjects = 2:
                        playerAction.heading <- ((getHeadingBetween(asteroidsList.get(0))
                        + getHeadingBetween(gasCloudsList.get(0)) + getHeadingBetween())
                        / -3 + random.nextInt(4) + 3) mod 360
                        break
            stopAfterburner()
    { Menghentikan afterburner jika size terlalu kecil }
    if (this.bot.getSize() <= 10)
        stopAfterburner()

{ Kondisi ketika seluruh objek di map telah habis, tetapi masih ada player yang hidup }
else if (not this.gameState.getPlayerGameObjects().isEmpty()) then
    { Melakukan penghindaran dari torpedo sebagai prioritas utama dan attack jika tidak
    ada torpedo yang dihindari }
    var torpedosList <- gameState.getGameObjects()
          .stream().filter(item -> item.getGameObjectType() == ObjectTypes.TORPEDOSALVO)
          .sorted(Comparator.comparing(item -> getDistanceBetween(bot, item)))
          .collect(Collectors.toList())
    torpedoDefense(torpedosList)
    if (torpedoHit == 0)then
        playerAction <- attack()
    if (this.bot.getSize() <= 10) then
        if (this.bot.effects mod 2 not = 0) then
            this.playerAction.action <- PlayerActions.STOPAFTERBURNER
```

```
{ Set player action dari bot sesuai dengna player action yang telah terdefinisi berdasarkan
kondisi }
this.playerAction <- playerAction
```

Implementasi algoritma yang lengkap dapat dilihat pada github yang telah disertakan pada lampiran.

4.2 Struktur Data Dalam Program *Bot* Galaxio

Permainan Galaxio menggunakan struktur data berupa class. Class yang digunakan terbagi dalam empat kategori, yaitu *Enums, Models, Services,* dan *Main*. Enums berisi konstan yang diterima maupun dikirim ke *game engine*. *Models* berisi kelas-kelas yang dapat dimanfaatkan dalam instantiasi *game* Galaxio. *Services* akan berisi class dari bot yang merupakan tempat dari seluruh algoritma bot dilakukan. Aplikasi dan implementasi algoritma *greedy* dilakukan pada kelas tersebut. Kelas terakhir yaitu main, kelas yang melakukan koneksi dan instantiasi bot. Berikut merupakan penjelasan struktur data yang lebih mendalam pada kategori *Enums*.

a. ObjectTypes.java

   Class yang berisi konstan tipe object pada game Galaxio. Memiliki satu atribut yaitu value yang sesuai dengan tipe object tersebut.

b. PlayerActions.java

   Class yang berisi konstan tipe command yang dapat dilakukan player pada game Galaxio. Memiliki satu atribut yaitu value yang sesuai dengan tipe command tersebut.

Berikut merupakan penjelasan struktur data yang lebih mendalam pada kategori Models.

a. GameObject.java

   - Atribut

Atribut	Deskripsi
public UUID id	ID dari GameObject
public Integer size	Size dari GameObject
public Integer speed	Speed dari GameObject
public Integer currentHeading	Heading dari GameObject
public Position position	Posisi GameObject pada map
public ObjectTypes gameObjectType	Jenis object dari GameObject
public Integer effects	Efek yang aktif

- Metode

Metode	Deskripsi
public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, Integer effects)	Konstruktor GameObject
public UUID getId()	Getter dari atribut id
public void setId(UUID id)	Setter dari atribut id
public int getSize()	Getter dari atribut size
public void setSize(int size)	Setter dari atribut size
public int getEffects()	Getter dari atribut effects
public int getSpeed()	Getter dari atribut speed
public void setSpeed(int speed)	Setter dari atribut speed
public Position getPosition()	Getter dari atribut position
public void setPosition(Position position)	Setter dari atribut position
public ObjectTypes getGameObjectType()	Getter dari atribut gameObjectType
public void setGameObjectType(ObjectTypes gameObjectType)	Setter dari atribut gameObjectType
public static GameObject FromStateList(UUID id, List<Integer> stateList)	Megembalikan GameObject baru berdasarkan stateList

b. GameState.java

- Atribut

Atribut	Deskripsi
public World world	Map dari GameState
public List<GameObject> gameObjects	List GameObject pada GameState
public List<GameObject> playerGameObjects	List player pada GameState

- Metode

Metode	Deskripsi
public GameState()	Konstruktor GameState
public GameState(World world , List<GameObject> gameObjects, List<GameObject> playerGameObjects)	Konstruktor GameState
public World getWorld()	Getter atribut world
public void setWorld(World world)	Setter atribut world
public List<GameObject> getGameObjects()	Getter atribut gameObjects
public void setGameObjects(List<GameObject> gameObjects)	Setter atribut gameObjects
public List<GameObject> getPlayerGameObjects()	Getter atribut playerGameObjects
public void setPlayerGameObjects(List<GameObject> playerGameObjects)	Setter atribut playerGameObjects

c. GameStateDto.java

- Atribut

Atribut	Deskripsi
private World world	Map dari GameState
private Map<String, List<Integer>> gameObjects	List konstan beserta value dari GameObjects pada GameState
private Map<String, List<Integer>> playerObjects	List konstan beserta value dari playerGameObjects pada GameState

- Metode

Metode	Deskripsi
public Models.World getWorld()	Getter atribut world
public void setWorld(Models.World world)	Setter atribut world

public Map<String, List<Integer>> getGameObjects()	Getter atribut gameObjects
public void setGameObjects(Map<String, List<Integer>> gameObjects)	Setter atribut gameObjects
public Map<String, List<Integer>> getPlayerObjects()	Getter atribut playerObjects
public void setPlayerObjects(Map<String, List<Integer>> playerObjects)	Setter atribut playerObjects

d. PlayerAction.java

- Atribut

Atribut	**Deskripsi**
public UUID playerId	ID dari player
public PlayerActions action	Command yang dilakukan player berupa salahs atu PlayerActions
public int heading	Heading dari player

- Metode

Metode	**Deskripsi**
public UUID getPlayerId()	Getter atribut playerId
public void setPlayerId(UUID playerId)	Setter atribut playerId
public PlayerActions getAction()	Getter atribut action
public void setAction(PlayerActions action)	Setter atribut action
public int getHeading()	Getter atribut heading
public void setHeading(int heading)	Setter atribut heading

e. Position.java

- Atribut

Atribut	**Deskripsi**
public int x	Titik x pada map
public int y	Titik y pada map

- Metode

Metode	Deskripsi
public Position()	Konstruktor position
public Position(int x, int y)	Konstruktor position
public int getX()	Getter atribut x
public void setX(int x)	Setter atribut x
public int getY()	Getter atribut y
public void setY(int y)	Setter atribut y

f. World.java

- Atribut

Atribut	Deskripsi
public Position centerPoint	Titik tengah map
public Integer radius	Besar radius map
public Integer currentTick	Tick pada game

- Metode

Metode	Deskripsi
public Position getCenterPoint()	Getter atribut centerPoint
public void setCenterPoint(Position centerPoint)	Setter atribut centerPoint
public Integer getRadius()	Getter atribut radius
public void setRadius(Integer radius)	Setter atribut radius
public Integer getCurrentTick()	Getter atribut currentTick
public void setCurrentTick(Integer currentTick)	Setter atribut currentTick

Pada kategori Services, hanya terdapat satu class, yaitu kelas BotService. Berikut adalah penjelasan struktur data dari BotService.

- Atribut

Atribut	Deskripsi
private GameObject bot	GameObject dari bot

private PlayerAction playerAction	PlayerAction dari bot
private GameState gameState	Game state yang diperoleh
final int MAX_GAP_WITH_BOUNDARY	Jarak maksimal ke boundary map
final int MAX_GAP_WITH_OTHER_SHIPS	Jarak maksimal ke kapal lain
final int MAX_GAP_WITH_HARMFUL_OBJECTS	Jarak maksimal ke benda berbahaya
private static boolean supernovaAvail	Boolean ketersediaan supernova
private static int teleporterCount	Banyak teleporter yang dimiliki
private static boolean firedTeleport	Boolean kondisi penembakkan teleporter
private boolean aggresiveMode	Boolean mode attack aggressive
private static boolean hasActiveTeleporter	Boolean kondisi memiliki teleporter
private static int torpedoHit	Jumlah torpedo yang berbahaya untuk bot

- Metode

Metode	**Deskripsi**
public BotService()	Konstruktor kelas BotService
public GameObject getBot()	Getter atribut bot
public void setBot(GameObject bot)	Setter atribut bot
public PlayerAction getPlayerAction()	Getter atribut playerAction
public void setPlayerAction(PlayerAction playerAction)	Setter atribut playerAction
public void computeNextPlayerAction(PlayerAction playerAction)	Set playerAction sesuai dengan kondisi dan strategi greedy yang digunakan
private int countSurroundHarmfulObjects(double nearestGasCloud, double nearestAsteroid)	Menghitung banyaknya objek berbahaya di sekitar bot
private void farming()	Set playerAction untuk mencari makan di heading makanan terdekat

private void moveToCenter()	Set playerAction untuk menuju ke tengah map
private PlayerAction attack()	Set playerAction untuk melakukan penyerangan ke kapal lain sesuai dengan strategi greedy yang diterapkan
private void stopAfterburner()	Set playerAction menjadi stop afterburner
private void startAfterburner()	Set playerAction menjadi start afterburner
private void forward()	Set playerAction menjadi forward
public GameState getGameState()	Getter dari atribut gameState
public void setGameState (GameState gameState)	Setter dari atribut gameState
private void updateSelfState()	Update GameState bot
private double getDistanceBetween(GameObject object1, GameObject object2)	Menghitung jarak antara dua GameObject
private boolean isTorpedoInRange(GameObject torpedo)	Mengembalikan boolean yang menandakan apakah object torpedo berada pada range 1,5 kali size bot dari posisi bot
private boolean isTorpedoHit(GameObject torpedo)	Mengembalikan boolean yang menandakan apakah heading object torpedo dapat mengenai bot
private void torpedoDefense(List<GameObject> torpList)	Set atribut torpedoHit berdasarkan seluruh torpedo yang ada di range dan dapat mengenai bot
private double getDistanceBetween(GameObject bot, World world)	Mengembalikan jarak dari bot ke boundary map

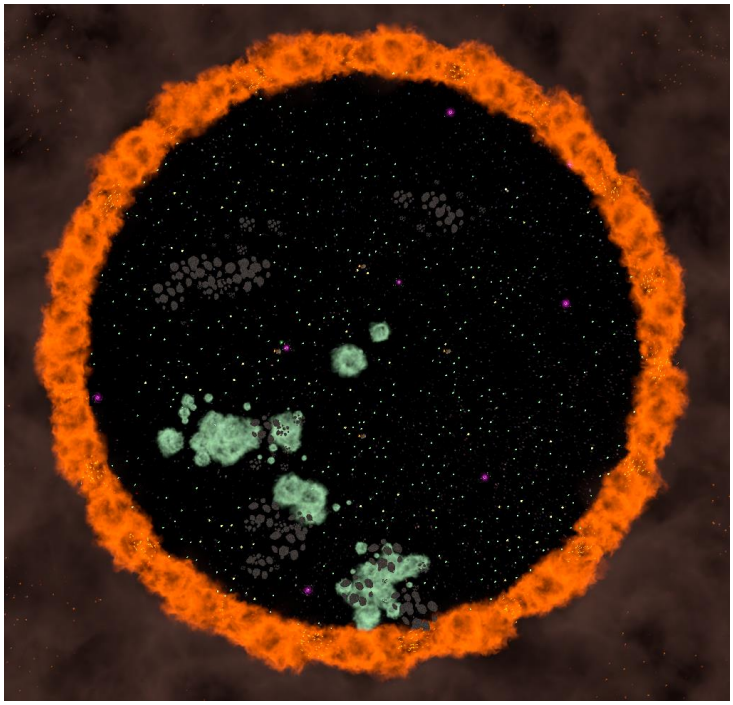
private int getHeadingBetween(GameObject object1, GameObject object2)	Mengembalikan heading dari satu object1 ke object2
private int getHeadingBetween(GameObject otherObject)	Mengembalikan heading dari bot ke GameObject lain
private int getHeadingBetween()	Mengembalikan heading dari bot ke titik pusat map
private int toDegrees(double v)	Mengembalikan sudut v radian dalam derajat

4.3 Pengujian

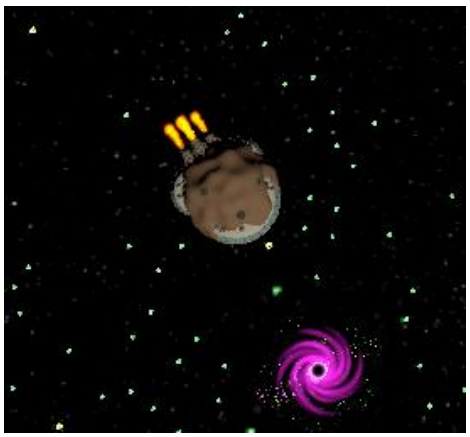
Dari implementasi strategi greedy yang telah dibuat pada program, dilakukan pengujian bot pada beberapa pertandingan. Dari proses pertandingan ini, akan dilakukan analisis solusi algoritma greedy yang diterapkan.

a. Pengujian I

Pada pengujian pertama, bot yang telah dibuat dengan nama JAB bertanding dengan tiga Reference Bot yang telah disediakan oleh permainan Galaxio. Keadaan awal permainan Galaxio adalah sebagai berikut. Bot JAB berada di paling kanan.

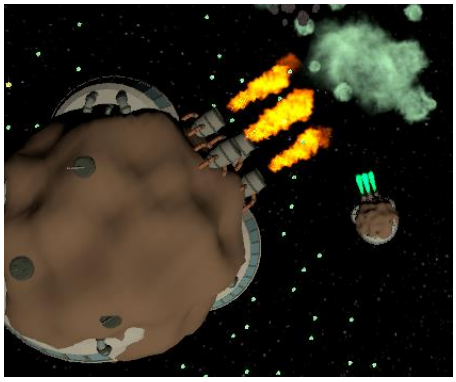


Saat awal permainan, bot JAB akan mencari *food* dan *superfood* seperti gambar berikut.

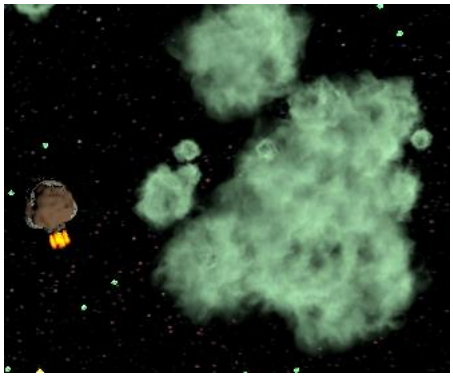


Ketika ada kapal musuh dalam range tertentu, bot akan menembakkan *torpedo salvo* seperti gambar berikut.

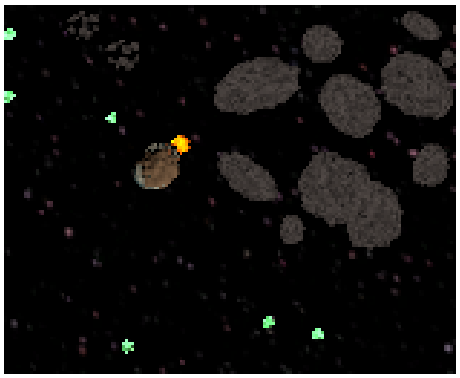
Ketika bertemu lawan dengan size yang lebih besar, bot JAB akan menggunakan *afterburner* untuk kabur.



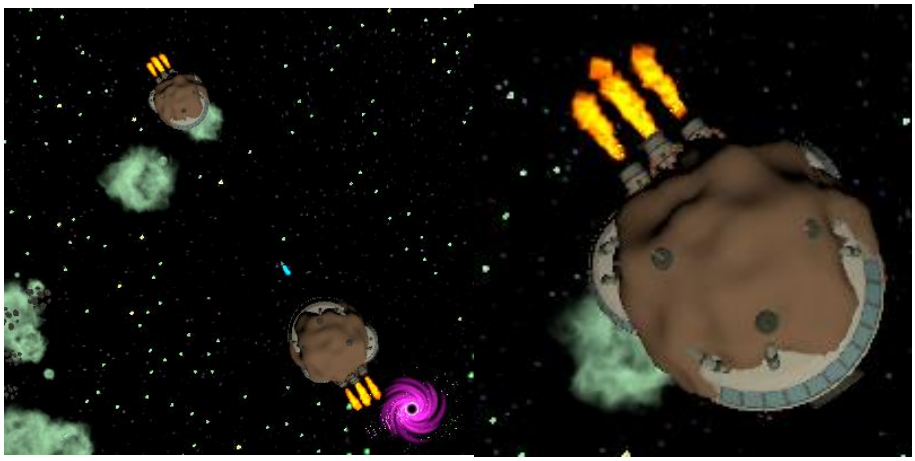
Gas Clouds dianggap sebagai salah satu bentuk *harmful objects*. Oleh karena itu, ketika terdeteksi ada *Gas Clouds* pada jarak yang dekat, bot akan mengubah *heading* kapalnya untuk menghindarinya.



Asteroids juga dianggap sebagai salah satu bentuk *harmful* objects. Oleh karena itu, ketika terdeteksi ada *Asteroids* pada jarak yang dekat, bot akan mengubah *heading* kapalnya untuk menghindarinya.



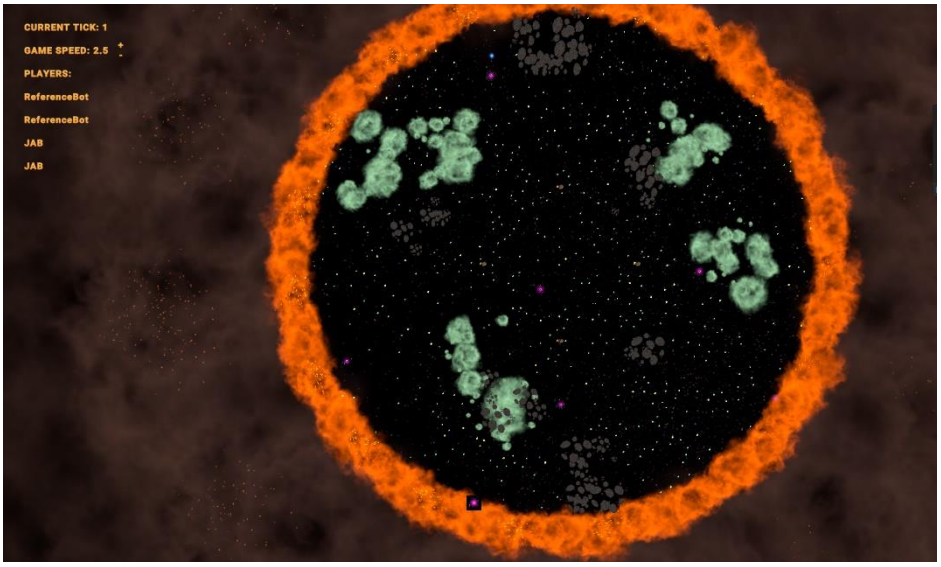
Ketika tersisa dua pemain dalam permainan, bot JAB akan mengaktifkan mode attack aggressiveMode. Dengan mode ini, bot JAB akan menembakkan teleporter jika kapal musuh memiliki size yang lebih kecil. Seperti yang dapat dilihat pada gambar di bawah ini. Pada gambar di kiri, bot JAB menembakkan teleporter ke arah lawan yang lebih kecil. Selanjutnya pada gambar di kanan, bot JAB melakukan teleport ketika teleporter yang telah ditembakkannya dekat dengan lawan.



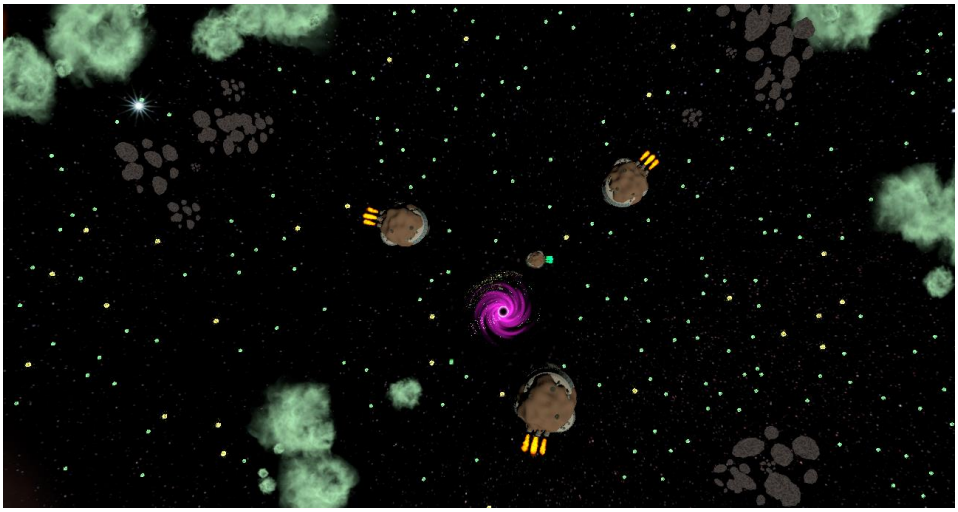
Hasil permainan dari pengujian pertama dimenangkan oleh bot JAB. Hal ini dikarenakan, bot JAB dapat memperoleh solusi yang cukup optimal pada kondisi *game* di pengujian pertama. Pada awal permainan, bot mampu mencari *food* sebanyak mungkin. Selain itu, bot mampu menghindari objek-objek berbahaya maupun kapal lawan yang lebih besar. Ketika tersisa dua kapal dan bot JAB memiliki ukuran lebih besar, dapat dipastikan bot JAB menang karena menggunakan *teleporter*.

b. Pengujian II

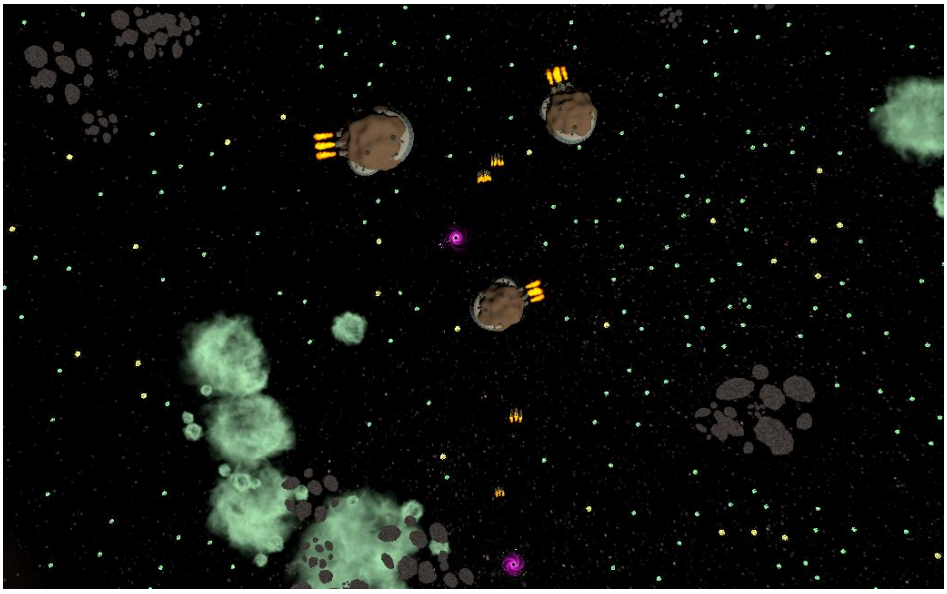
Pada pengujian kedua ini, dua bot JAB bertanding dengan dua Reference Bot. Berikut merupakan kondisi awal dari permainan. Bot JAB berada pada posisi bawah dan kanan.



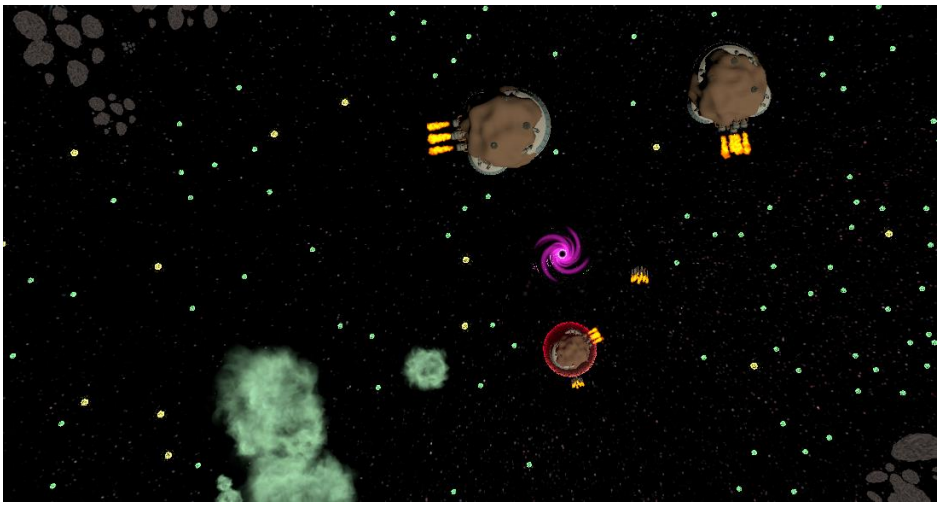
Salah satu bot JAB menggunakan afterburner untuk kabur dari kapal yang lebih besar. Namun, arah menghindar bot justru membuat bot bertemu dengan dua kapal lain yang lebih besar. Hal ini merupakan kasus di mana solusi *greedy* yang digunakan tidak menghasilkan hasil optimal. Strategi greedy yang digunakan hanya peduli pada kapal terdekat tanpa memperhitungkan kapal lain yang mungkin lebih besar. Karena hal tersebut, salah satu bot JAB kalah.



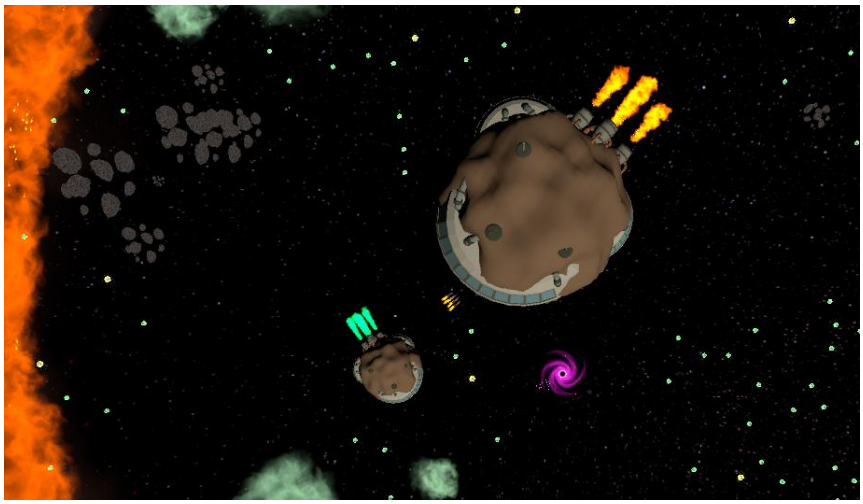
Selanjutnya bot JAB melakukan penembakkan torpedo ke kapal lain. Namun, karena ada wormhole, penembakkan torpedo ini justru mengarah ke bot sendiri. Hal ini dikarenakan wormhole tidak dipertimbangkan dalam solusi greedy yang diimplementasikan. Tanpa pertimbangan ini, dapat dihasilkan kondisi tidak optimal di mana bot menembakkan torpedo yang dapat mengarah ke dirinya sendiri.



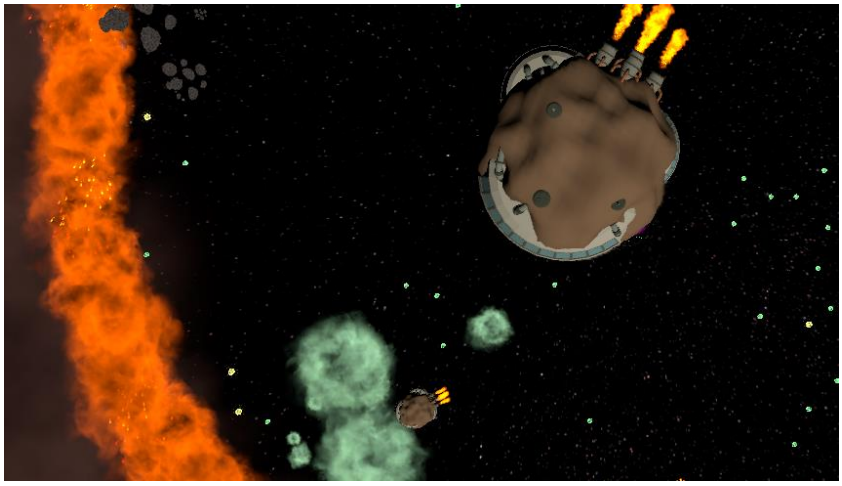
Ketika ada torpedo dengan jarak yang sangat dekat, bot akan mengaktifkan shield yang dimiliki.



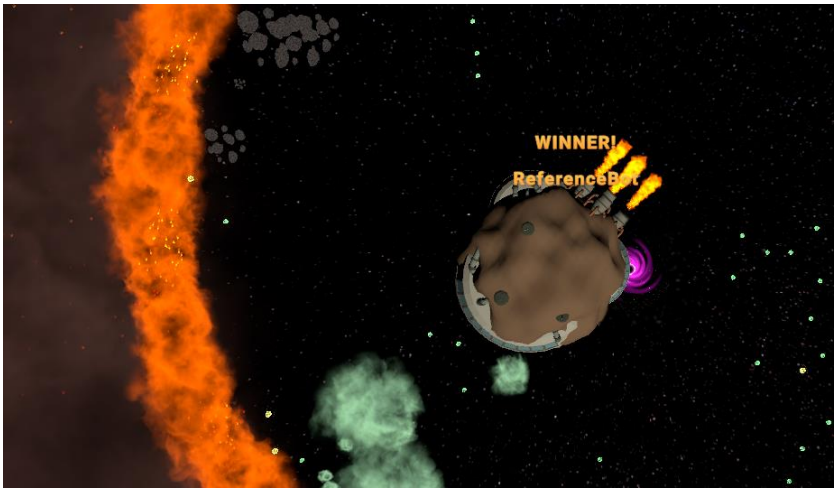
Ketika ada kapal lawan dengan jarak tertentu, bot JAB dapat kabur dan menembakkan torpedo ke kapal lawan.



Selanjutnya, terdapat kondisi tidak optimal kembali yang membuat bot JAB menghindari dari kapal besar menuju objek berbahaya. Hal ini dikarenakan ketika menentukan arah menghindar dari kapal besar, bot JAB belum mempertimbangkan seluruh objek berbahaya yang masih jauh.



Hasil pengujian kedua dimenangkan oleh Reference Bot. Hal ini dikarenakan banyak kondisi tidak optimal yang dihasilkan pada solusi greedy yang diimplementasikan. Kondisi-kondisi tersebut telah dijelaskan sebelumnya.



BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil pengerjaan tugas besar I IF2211 Strategi Algoritma ini, kami berhasil memperoleh kesimpulan sebagai berikut.

- Algoritma *Greedy* adalah pendekatan dalam bahasa pemrograman untuk menyelesaikan suatu permasalahan dengan mengambil tindakan terbaik pada setiap tahapan dengan harapan bahwa pengambilan tindakan tersebut dapat menghasilkan suatu solusi yang optimal secara global.
- Algoritma *Greedy* berhasil kami gunakan dalam penyusunan strategi bot kapal untuk memenangkan permainan Galaxio.
- Algoritma *Greedy* berhasil kami implementasikan dalam bahasa pemrograman Java.
- Algoritma *Greedy* ini mempertimbangkan objek-objek permainan serta karakteristiknya dalam permainan Galaxio.
- Ada berbagai alternatif algoritma *Greedy* yang kami temukan dalam pengerjaan bot untuk permainan Galaxio ini.

5.2 Saran

Dalam pengerjaan tugas besar I IF2211 Strategi Algoritma ini, kami menyarankan agar kode program yang diberikan di awal pada bagian Models bisa dibuat lebih lengkap sehingga pengambilan data, misalnya banyak *teleporter* yang dimiliki, bisa dilakukan dengan lebih mudah.

DAFTAR PUSTAKA

E. (n.d.). GitHub - EntelectChallenge/2021-Galaxio. GitHub. Retrieved February 10, 2023, from https://github.com/EntelectChallenge/2021-Galaxio

Algoritma Greedy. Retrieved February 15, 2023, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/stima22-23.htm

LAMPIRAN

Repository GitHub dan tautan video dapat diakses pada link sebagai berikut.

Repository : https://github.com/liviaarumsari/Tubes1_JAB.git

Video : Video Tugas Besar 1 IF2211 Strategi Algoritma - Kelompok JAB