IF2211 Strategi Algoritma

IMPLEMENTASI ALGORITMA GREEDY
PADA APLIKASI PERMAINAN "GALAXIO"

Laporan Tugas Besar I

Disusun untuk memenuhi tugas mata kuliah Aljabar Linear dan Geometri

Pada Semester 2 (dua) Tahun Akademik 2022/2023

Disusun Oleh:

Enrique Alifio Ditya 13521142
Rava Maulana 13521149
Vanessa Rebecca Wiyono 13521151

Kelompok ClosedAI Tech.

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

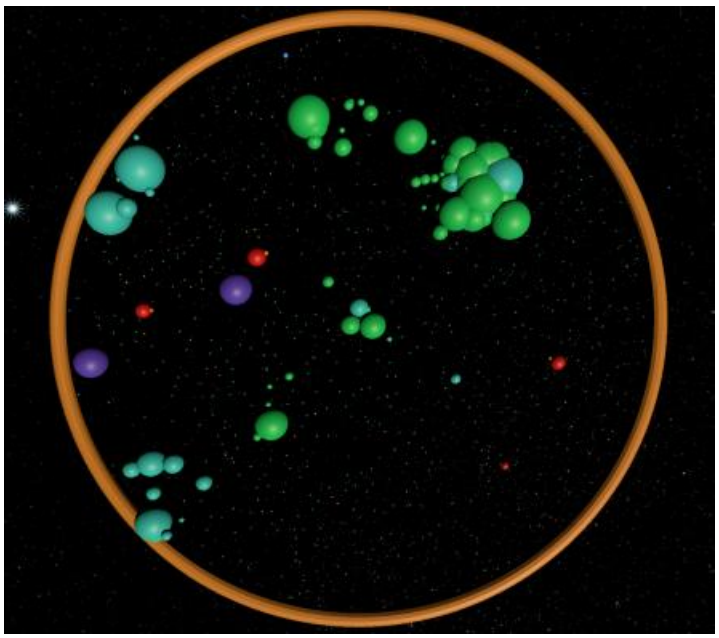
BANDUNG
2023

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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



Gambar 1. Ilustrasi permainan Galaxio

IF2211 Strategi Algoritma - Tugas Besar 1 1

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Galaxio. Game engine dapat diperoleh pada laman berikut:

https://github.com/EntelectChallenge/2021-Galaxio

Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan Galaxio dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada starter-bots di dalam starter-pack pada laman berikut ini:

https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2

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 denga 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 ukurandan 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 IF2211 Strategi Algoritma - Tugas Besar 1 2 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 dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Berikut jenis-jenis dari command yang ada dalam permainan:

a. FORWARD
b. STOP
c. START_AFTERBURNER
d. STOP_AFTERBURNER
e. FIRE_TORPEDOES
f. FIRE_SUPERNOVA
g. DETONATE_SUPERNOVA
h. FIRE_TELEPORTER
i. TELEPORTUSE_SHIELD

11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal player lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi. Adapun peraturan yang lebih lengkap dari permainan Galaxio, dapat dilihat pada laman :

    https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md

1.2. Spesifikasi Tugas

Pada tugas besar kali ini, anda diminta untuk membuat sebuah bot untuk bermain permainan Galaxio yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut:

1. Download latest release starter pack.zip dari tautan berikut https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2
2. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut:
    a. Java (minimal Java 11)
       https://www.oracle.com/java/technologies/downloads/#java
    b. IntelIiJ IDEA: https://www.jetbrains.com/idea/
    c. NodeJS: https://nodejs.org/en/download/

d. .Net Core 3.1: https://dotnet.microsoft.com/en-us/download/dotnet/3.1

3. Panduan mengenai cara menjalankan permainan, membuat bot, build src code, dan melihat visualizer bisa dicek melalui tautan berikut: https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa7 0IgnLNU/edit#
4. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di starter-bot. Ingat bahwa bot kalian harus menggunakan bahasa Java. Dilarang menggunakan kode program yang sudah ada, mahasiswa wajib membuat program/strategi sendiri. Tetapi, belajar dari program yang sudah ada tidak dilarang.

Strategi greedy yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menghindari objek yang dapat mengurangi ukuran kapal. Buatlah strategi greedy terbaik, karena setiap bot dari masing-masing kelompok akan diadu dalam suatu kompetisi Tubes 1 (TBD).

Strategi greedy harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB II
LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma Greedy merupakan suatu algoritma yang penggunaannya sangat populer dalam mengatasi persoalan yang membutuhkan optimasi. Algoritma Greedy memecahkan masalah secara bertahap langkah demi langkah sehingga akan diambil opsi optimal dalam setiap langkah sebagaimana telah disesuaikan dengan tujuann penggunaan algoritma. Karena setiap langkah yang diambil dianggap bersifat optimum, maka diharapkan akan menghasilkan hasil akhir yang optimum juga. Umumnya, algoritma Greedy digunakan dalam permasalahan

optimasi seperti permasalahan optimasi (*maximization*) dan permasalahan meminimasi (*minimization*).

Walaupun solusi setiap langkah yang dipilih telah merupakan solusi optimum, namun solusi tersebut masih bersifat optimum lokal sehingga belum tentu menghasilkan hasil yang optimum juga pada hasil keseluruhan. Selain itu, fungsi seleksi yang dapat digunakan dengan menerapkan algoritma *greedy* untuk menyelesaikan suatu persoalan juga dapat bervariasi karena cara pendekatan yang dilakukan oleh perancang algoritma berbeda-beda. Algoritma *greedy* tidak bekerja secara menyeluruh pada semua kemungkinan yang ada, sehingga tidak dapat dipastikan bahwa penggunaan algoritma *greedy* dapat memecahkan segala jenis persoalan dan selalu menghasilkan solusi paling optimum.

Berikut merupakan elemen-elemen yang menggambarkan Algoritma Greedy:

1. Terdapat himpunan kandidat (C) yang berisi kandidat-kandidat yang akan dipilih pada setiap langkah.
2. Terdapat himpunan solusi (S) yang berisi kandidat-kandidat yang telah dipilih.
3. Terdapaat Fungsi seleksi (*selection function*) yang digunakan untuk memilih kandidat berdasarkan strategi *greedy* tertentu.
4. Terdapat fungsi kelayakan (*feasible*) yang digunakan untuk memeriksa apakah kandidat yang dipilih layak untuk dimasukkan ke dalam himpunan solusi.
5. Terdapat fungsi solusi yang digunakan untuk menentukan apakah himpunan kandidat yang dipilih telah memberikan solusi.
6. Terdapat fungsi objektif yang digunakan untuk memaksimalkan atau meminimumkan aksi yang akan dilakukan.

2.2 Pendekatan *Game Engine*

Game Engine merupakan sebuah mesin yang diracang secara khusus untuk menjalankan suatu program atau aplikasi permainan. Pada permainan "Galaxio", *game engine* dibuat dengan Unity dan telah tersedia di *starter pack* yang berasal dari Entelect Challenge 2021. *Game engine* ini telah dilengkapi dengan *visualizer* yang digunakan untuk memvisualisasikan program dengan grafis layaknya permainan komputer pada umumnya.

Setelah mengunduh starter bot yang telah disediakan, kami mengubah susunan folder menjadi tiga folder utama yaitu bin, doc, dan src. Folder bin berisikan executable program, folder doc berisi laporan tugas besar, dan folder src berisi seluruh source code yang digunakan

dallam program dan didalamnya terdapat folder berrnamakan “RavaBot”, yaitu source code bot kami. Penambahan kode untuk memperjelas effect serta game object terdapat dalam file-file pada folder “Enums” dan “Models” pada folder RavaBot. Selain itu, hasil implementasi algoritma *greedy* yang akan digunakan oleh bot dalam permainan terdapat pada file BotService.Java dalam folder “Services” pada folder “RavaBot”. Selama permainan berlangsung, pemain tidak dapat memilih opsi algoritma *greedy* dan bot akan berjalan secara otomatis dengan algoritma *greedy* yang telah ditentukan oleh kelompok kami.

Terdapat berbagai aksi yang dapat dilakukan oleh bot, seperti aksi menyerang yang antara lain adalah menembak torpedo, menembakkan supernova, meledakkan supernova, dan mengejar bot musuh, dan aksi bertahan seperti aktivasi perisai. Selain aksi menyerang dan bertahan, terdapat beberapa aksi yang bersifat netral karena penggunaannya dapat digunakan dalam menyerang maupun dalam bertahan. Aksi-aksi netral tersebut antara lain adalah bergerak maju kedepan sesuai *heading* yang telah ditentukan untuk berbagai kasus sebagaimana perhitungannya menggunakan algoritma *greedy*, aksi teleportasi, penggunaan afterburner yang berguna untuk meningkatkan kecepatan namun berdampak pada ukuran bot, dan pemberhentian penggunaan afterburner, serta aksi berhenti.

Untuk menjalankan *game engine*, terdapat beberapa hal yang diperlukan seperti Java (minimal Java 11), Net Core, dan Maven. Setelah mengunduh *starter pack* dari github Entellect Challenge 2021, pengguna dapat mengunduh RavaBot yang telah kami buat melalui repo github kelompok kami. Sebelum memulai permainan, pengguna dapat melakukan konfigurasi terkait berapa banyak pemain/bot yang diinginkan dalam suatu game melalui file “appsettings..json” yang terletak di folder “engine-publish”. Agar bot dapat dijalankan pada *game engine*, kode perlu diubah menjadi file jar terlebih dahulu dengan menggunakan maven “mvn clear package” yang kemudian akan terbentuk dalam folder “target” berupa *executable file jar* (dalam kasus ini file .jar telah kami sediakan sehingga tidak perlu mengubah file menjadi .jar terlebih dahulu). Kemudian, buka terminal lalu cd ke directory folder “starter-pack” dan jalankan *game engine* dengan command sebagai berikut (asumsi 1 *game* dengan 4 bot. 1 RavaBot dan 3 ReferenceBot):

```
@echo off

cd ./starter-bots/RavaBot/
start "" mvn clean package
cd ../../
```

```
:: Game Runner
cd ./runner-publish/
start "" dotnet GameRunner.dll

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

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

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

timeout /t 5
cd ../starter-bots/RavaBot/target
start "" java -jar RavaBot.jar
::timeout /t 5
::start "" java -jar RavaBot.jar
cd ../../../../

pause
```

Setelah *game engine* dan seluruh bot telah berhasil dijalankan, *game state log* akan tersimpan dalam folder "logger-publish" dan dapat divisualisasikan dengan menggunakan visualier bertipe aplikasi yang telah tersedia pada starterpack dalam folder "visualiser". Berikut cara menjalankan visualizer:

1. Jalankan aplikasi Galaxio
2. Buka menu "Options"
3. Salin path folder "logger-publish" pada "Log Files Lovation", lalu "Save"
4. Pilih menu "Load"
5. Pilih *file* JSON yang ingin diload pada "Game Log", lalu "Start"
6. Setelah masuk ke visualisasinya, terdapat opsi *start, pause, rewind,* dan *reset*

7. Selamat, *game* sudah siap untuk dijalankan !

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Pemetaan Algorita *Greedy* pada Permasalahan Permainan "Galaxio"

a. Pemetaan Umum Permasalahan Permainan

Berikut merupakan pemetaan permasalahan permainan "Galaxio" secara umum menjadi elemen-elemen penyusun Algoritma *Greedy*.

Elemen Algoritma *Greedy*	Pemetaan pada Permainan *Galaxio*
Himpunan Kandidat (C)	Fungsi-fungsi yang mungkin dijalankan oleh bot dalam permainan, dapat merupakan fungsi untuk melakukan aksi seperti startafterburner, menembakkan torpedo, menembakkan supernova, meledakkan supernova, teleportasi, mengaktifkan perisai, serta arah heading yang dituju.
Himpunan Solusi (S)	Fungsi terbaik yang dipilih sesuai dengan kondisi setiap tik selama permainan berlangsung.
Fungsi Solusi	Memeriksa apakah fungsi yang dipilih merupakan fungsi yang terdifinisi dan valid dalam permainan. Jika tidak valid, maka fungsi tidak akan dijalankan oleh *game engine*.
Fungsi Seleksi (*Selection Function*)	Memilih perintah yang sesuai untuk setiap state dengan prioritas sebagaimana algoritma greedy telah diterapkan guna mengoptimalisasi setiap aksi serta

	pergerakan bot. Fungsi seleksi dilakukan dengan membuat urutan kode if else.
Fungsi Kelayakan (*Feasibility*)	Memeriksa apakah semua kondisi yang diperlukan untuk mengeksekusi suatu fungsi terpilih sudah terpennuhi atau belum. Validasi dilakukan terhadapp komponen-komponen permainan, misalnya jarak antara bot pengguna dengan bot musuh, ukuran bot pengguna, kecepatan pengguna, dan atribut-atribut yang dimiliki seperti torpedo salvo.
Fungsi Objektif	Fungsi yang bertujuan untuk memenangkan permainan, baik dengan cara pertahanan maupun penyerangan. Fungsi ini berguna untuk memaksimasi ukuran bot dengan harapan menjadi bot terbesar dalam permainan untuk memenangkan lawan.

b. Pemetaan Pergerakan dalam Algoritma Greedy

Pergerakan merupakan salah satu aspek terpenting dalam permainan "Galaxio" karena setiap pergerakan memiliki dampak besar pada menang kalahnnya bot.

Elemen Algoritma *Greedy*	Pemetaan pada Permainan "Galaxio"
Himpunan Kandidat (C)	Himpunan dalam bentuk list yang berisikan fungsi-fungsi yang berkaitan dengan pergerakan dalam permainan "Galaxio". Fungsi bergerak tidak dinyatakan secara eksplisit, namun diimplementasikan kedalam fungsi-fungsi lainnya. Contoh dari fungsi yang berkait dengan pergerakan adalah fungsi penentu arah heading yang ditentukan oleh berbagai faktor, seperti makanan

	terdekat, musuh terdekat beserta ukurannya, *incoming* torpedo terdekat, gas cloud, dan asteroid field.
Himpunan Solusi (S)	Himpunan dalam bentuk list yang berisikan perintah terbaik yang dipilih sesuai dengan kondisi tiap saat selama permainan berlangsung sehingga heading yang dipilih merupakan heading paling optimal.
Fungsi Solusi	Fungsi yang digunakan untuk memeriksa apakah fungsi yang digunakan valid dalam permainan. Jika tidak valid, maka fungsi tidak akan dijalankan oleh *game engine.*
Fungsi Seleksi (*Selection Function*)	Penggunaannya bertujuan untuk memilih fungsi mana yang akan dijalankan sesuai dengan prioritas strategi algoritma *greedy* yang telah diimplementasikan sesuai dengan urutan if else pada kode.
Fungsi Kelayakan (*Feasibility*)	Memeriksa apakah semua kondisi yang diperlukan untuk mengeksekusi suatu fungsi yang telah diseleksi. Validasi dilakukan terhadap validitas koordinat, atribut yang dimiliki, serta objek pada area permainan.
Fungsi Objektif	Fungsi yang digunakan untuk memaksimalkan ukuran bot dalam setiap pergerakan. Terdapat algoritma perhitungan rasio yang merupakan hasil bagi antara size dengan jarak suatu objek terhadap bot untuk membantu kalkulasi optimasi. Sehingga ketimbang diam dan tidak melakukan apa-apa, bot akan selalu bergerak ke arah yang paling menguntungkan.

c. Pemetaan Penyerangan dalam Algoritma Greedy

Strategi penyerangan merupakan salah satu aspek yang paling penting dalam permainan "Galaxio" karena menjadi dasar bagi bot ketika bertemu dengan bot lain. Apabila bot tidak dilengkapi dengan kemampuan menyerang, maka akan lebih sulit bagi suatu bot untuk menambah ukuran secara signifikan karena satu-satunya faktor yang dapat menyebabkan perbesaran ukuran hanya berasal dari makanan.

Elemen Algoritma *Greedy*	Pemetaan pada Permainan "Galaxio"
Himpunan Kandidat (C)	Himpunan yang berisikan fungsi-fungsi yang berkaitann dengan penyerangan dalam permainan "Galaxio". Fungsi-fungsi yang tergolong dalam kategori penyerangan antara lain adalah fungsi untuk menembakkan torpedo, fungsi untuk mengejar musuh, fungsi untuk menembakkan supernova, dan fungsi untuk meledakkan supernova. Selain itu, terdapat juga fungsi yang tergolong netral karena dapat digunakan dalam penyerangan maupun pertahanan, seperti fungsi untuk teleport dan fungsi untuk menyalakan afterburner. Afterburner dapat digunakan dalam penyerangan ketika ingin mengejar musuh.
Himpunan Solusi (S)	Himpunan yang berisikan fungsi-fungsi pilihan sesuai dengan kondisi permainan, yaitu posisi, ukuran, serta jarak antara RavaBot dengan bot lawan.
Fungsi Solusi	Fungsi yang digunakan untuk melakukan validasi terhadap fungsi yang dipilih. Apabila fungsi yang dipilih tidak valid (contoh: ingin melakukan penyerangan

	hanya dalam area permainan yang valid), maka aksi tidak akan dilakukan.
Fungsi Seleksi (*Selection Function*)	Fungsi yang dilakukan untuk menyeleksi fungsi mana yang sesuai untuk dilakukan pada setiap *game state* sesuai dengan prioritas sebagaimana telah diimplementasikan menggunakan algoritma *greedy*. Sebagai contoh, terdapat fungsi untuk menyerang yang mengharuskan serangan ditujukan pada bot musuh terdekat dengan size yang lebih kecil dari RavaBot. Hal ini dilakukan dengan mendeteksi seluruh bot lawan pada arena permainan, lalu mengurutkannya kedalam list berdasarkan jarak terdekat dan meninjau item pertama pada list.
Fungsi Kelayakan (*Feasibility*)	Fungsi untuk memeriksa apakah semua kondisi yang diperlukan untuk mengeksekusi suatu fungsi terpilih sudah terpenuhi atau belum. Sebagai contoh adalah melakukan pengecheckan terhadap jumlah supernova yang dimiliki sebelum melakukan penyerangan.
Fungsi Objektif	Memaksimalkan serangan terhadap musuh dengan melakukan kalkulasi terhadap ukuran bot dengan bot musuh. Apabila perbedaan ukuran tidak terlalu signifikan, maka bot tidak akan secara agresif mengejar musuh karena terdapat kemungkinan dimana ukuran musuh kemudian menjadi lebih besar ketika jarak bot sudah sangat dekat dengan bot musuh.

d. Pemetaan Pertahanan dalam Algoritma *Greedy*

Selain pernyerangan, pertahanan juga merupakan aspek yang sangat penting untuk mencegah bot terbunuh oleh bot musuh. Oleh karena itu, implementasi algoritma *Greedy* untuk pertahanan bertujuan untuk memaksimalkan pertahanan bot dari serangan bot lain.

Elemen Algoritma *Greedy*	Pemetaan pada Permainan "Galaxio"
Himpunan Kandidat (C)	Himpunan yang berisikan fungsi-fungsi yang berkaitann dengan pertahanan dalam permainan "Galaxio". Fungsi-fungsi yang tergolong dalam kategori pertahanan antara lain adalah fungsi untuk menggunakan shield, serta fungsi untuk mengubah arah heading agar pergerakan tetap optimal namun menjauhi musuh. Selain itu, terdapat juga fungsi yang tergolong netral karena dapat digunakan baik dalam penyerangan maupun pertahanan, seperti fungsi untuk teleport dan fungsi untuk menyalakan afterburner. Afterburner dapat digunakan sebagai pertahanan ketika dinyalakan untuk kabur dari musuh.
Himpunan Solusi (S)	Himpunan yang berisikan fungsi-fungsi pilihan sesuai dengan kondisi permainan, yaitu posisi, ukuran, serta jarak antara RavaBot dengan bot lawan, dan jarak bot dengan *incoming torpedo* terdekat.
Fungsi Solusi	Fungsi yang digunakan untuk melakukan validasi terhadap fungsi yang dipilih. Apabila fungsi yang dipilih tidak valid, maka aksi tidak akan dilakukan.

Fungsi Seleksi (*Selection Function*)	Fungsi yang dilakukan untuk menyeleksi fungsi mana yang sesuai untuk dilakukan pada setiap *game state* sesuai dengan prioritas sebagaimana telah diimplementasikan menggunakan algoritma *greedy*. Sebagai contoh, terdapat fungsi untuk bertahan dengan afterburner. Apabila hasil kalkulasi menunjukkan bahwa bot dapat "lolos" dari musuh tanpa perlu menggunakan afterburmer, maka program akan memilih untuk tidak menggunakan afterburner.
Fungsi Kelayakan (*Feasibility*)	Fungsi untuk memeriksa apakah semua kondisi yang diperlukan untuk mengeksekusi suatu fungsi terpilih sudah terpenuhi atau belum. Sebagai contoh adalah melakukan pengecheckan terhadap ukuran bot sebelum menyalakan afterburner. Apabila ukuran bot terlalu kecil, maka afterburner tidak akan dinyalakan karena dapat berpotensi meledakkan bot.
Fungsi Objektif	Memaksimalkan pertahanan terhadap musuh dengan melakukan perhitungan terhadap ukuran kita dengan ukuran musuh. Apabila perbedaan ukuran sangat sedikit, maka bot tidak akan menggunakan afterburner untuk kabur dari musuh karena masih ada kemungkinan dimana ukuran keduanya kemudian menjadi sama.

3.2 Eksplorasi Alternatif Strategi *Greedy* dalam Permainan "Galaxio"

Current heading merupakan suatu hal yang sangat penting karena menjadi penentu arah gerak bot, dan untuk menentukannya terdapat beberapa strategi yang dapat digunakan. Guna mempermudah pemilihan strategi, alternatif strategi dapat dibagi menjadi tiga kategori yaitu optimasi makanan, penyerangan, dan pertahanan.

a. Strategi Optimasi Makanan

Strategi optimasi pertama adalah pergerakan bot yang melakukan optimalisasi dengan berfokus pada aksi "makan" sehingga ukurannya terus bertambah. Optimalisasi dengan cara "makan" ini akan mengakibatkan bot selalu bergerak menuju makanan terdekat dengan asumsi bahwa semakin banyak makanan yang dimakan, maka semakin besar pula ukuran bot nantinya dan semakin besar pula kemungkinan untuk menang.

Selain optimasi yang dilakukan dengan makan, opsi optimasi pergerakan lainnya adalah dengan menentukan arah gerak bot berdasarkan posisi musuh. Terdapat dua hal yang memungkinkan dalam hal ini, dan hal tersebut adalah memperhitungkan jarak musuh untuk bertindak secara ofensif, atau memperhitungkan jarak musuh untuk bergerak secara defensif. Apabila dilakukan secara ofensif, maka *heading* akan selalu ditujukan pada musuh lebih kecil terdekat, dan apabila dilakukan secara defensive, maka *heading* akan selalu dibuat untuk menjauhi musuh lebih besar terdekat.

Cara optimasi lain yang dapat dilakukan adalah dengan memilih arah berdasarkan perhitungan densitas. Setiap objek akan diberi skor tertentu untuk menunjukkan seberapa menguntungkannya objek tersebut dan kemudian dilakukan perhitungan dengan membagi skor dan jarak. Pergerakan bot akan mengikuti arah yang memiliki densitas tertinggi.

b. Strategi Penyerangan

Sama halnya dengan melakukan pergerakan, terdapat beberapa pendekatan strategi dalam melakukan penyerangan, dalam hal ini kelompok kami terpikirkan untuk merancang dua algoritma yang antara lain adalah dengan cara memfokuskan bot pada aksi "mengejar" musuh dan memfokuskan bot pada aksi "menembak" musuh menggunakan torpedo ataupun supernova.

Aksi mengejar musuh merupakan suatu hal ofensif yang penting dan perlu dilakukan. Apabila bot berhasil mengejar dan "memakan" musuh maka ukuran bot akan bertambah secara signifikan dan total musuh akan langsung berkurang satu, sehingga

kemungkinan untuk memenangkan permainan juga bertambah. Apabila terfokus dalam mengejar musuh secara agresif, maka prioritas utama bot adalah mengejar musuh dengan ukuran yang lebih kecil dari dirinya sendiri ketimbang mencari makan atau melakukan hal lainnya.

Cara lain yang dapat dilakukan untuk melakukan penyerangan adalah dengan melakukan optimasi melalui "penembakan" torpedo maupun supernova. Torpedo memiliki keunggulan disbanding "mengejar" musuh karena dapat dilakukan dari jarak yang lebih jauh sehingga tidak memakan waktu bagi bot untuk mendekatkan diri ke musuh. Torpedo memiliki efek samping untuk mengurangi ukuran bot peluncur torpedo, sehingga apabila tidak dilakukan dengan benar maka penggunaan torpedo justru dapat menimbulkan kerugian dan bukannya optimasi. Untuk memaksimalkan penyerangan, terdapat beberapa objek yang dapat digunakan, seperti *afterburner* untuk mempercepat gerak bot ketika bergerak mengejar musuh, dan *teleport* untuk teleportasi ke posisi yang dekat dengan musuh yang dikejar, Dengan algoritma ini, bot diharapkan untuk memenangkan permainan dengan cara membunuh semua musuh lainnya.

c. Strategi Pertahanan

Berbeda dengan strategi optimasi penyerangan, optimasi pertahanan akan menghasilkan bot yang "pasif" karena tidak melakukan penyerangan dan justru menghindar dari musuh. Pertahanan dapat dilakukan dengan cara terus memperhitungkan jarak antara bot dengan musuh, sehingga bot akan selalu bergerak kearah yang jauh dari musuh, terlebih dengan musuh yang memiliki ukuran lebih besar.Karena disetting untuk menjauh dari musuh, maka harapannya bot akan dapat memenangkan permainan dengan cara "last man standing" atau membiarkan bot/musuh lain saling membunuh dan menjadi pasif hingga tersisa dirinya sendiri.

Strategi pertahhanan untuk selalu menjauhi musuh merupakan salah satu implementasi algoritma *greedy* karena mencari arah pergerakan yang menghasilkan jarak maksimum antara bot dengan musuh. Untuk memaksimalkan pertahanan, algoritma bot dapat disertai dengan penggunaan object tambahan seperti *shield* untuk men-*defect* torpedo, afterburner untuk mempercepat pergerakan ketika kabur dari musuh, dan teleport untuk teleportasi menjauhi musuh.

3.3 Analisis Efisiensi dan Efektivitas Strategi *Greedy*

Dalam permainan "Galaxio", kompleksiitas waktu algoritma bukan merupakan hal utama yang perlu diperhatikan karena setiap aksi akan diterminasi sesuai dengan tik yang telah diatur sebagaimanna oleh penyelenggara *Game,* yaitu Entellect Challenge 2021. Walau demikian, efektivitas dan efisiensi dari algoritma merupakan hal sangat penting yang perlu dianalisis karena keefektivan algoritma akan mempengaruhi kemampuan bot. Berikut merupakan analisis terhadap efisiensi dan efektivitas dari setiap strategi yang telah disebutkan pada bagian 3.2.

3.3.1 Analisis Efisiensi

Untuk strategi pergerakan yang berfokus pada aksi makan, memilih *heading* menuju makanan terdekat merupakan suatu hal yang tepat karena akan mengoptimalisasi makanan yang dimakan. Namun, perlu dilakukan perhitungan lain seperti berapa banyak makanan lain yang terdapat disekitar makanan tersebut. Sebagai contoh, apabila terdapat 1 makanan berjarak 1 yang disekitarnya tidak ada makanan lain dan makanan yang berjarak 2 namun dikelilingi oleh makanan lain, maka bot akan memilih untuk bergerak menuju makanan terdekat, yaitu yang berjarak 1 tetapi tidak dikelilingi oleh makanan lain. Padahal, realitanya akan lebih optimal jika bot memilih makanan yang berjarak 2 namun dekelilingi oleh banyak makanan lainnya.

Selain itu, perlu dilakukan pertimbangan terhadap musuh yang berada disekitar makanan. Akan menjadi suatu hal yang merugikan apabila bot bergerak menuju makanan terdekat padahal makanan tersebut terletak sangat dekat dengan musuh yang memiliki ukuran lebih besar dari diri bot itu sendiri. Jika hal ini terjadi, bot justru akan kalah karena berujung termakan oleh bot musuh. Selain mempertimbangkan banyak makanan pada suatu area dan mempertimbangkan musuh yang terletak dengan makanan tersebut, bot juga perlu mempertimbangkan objek-objek disekitar makanan, seperti gas cloud, wormhole, dan lain-lain.

Dalam strategi optimasi penyerangan, akan menjadi suatu hal baik apabila bot bertindak agresif dan berhasil memakan atau membunuh seluruh bot musuh. Namun, menjadi bot yang sangat agresif bukanlah merupakan suatu hal baik. Apabila robot terlelu berfokus pada mengejar musuh lain, maka bot akan mengesampingkan makanan sehingga ukurannya bertambah besar dalam kurun waktu yang lama, padahal ukuran musuh telah bertambah besar. Terdapat kemungkinan dimana dimana ukuran bot musuh yang tadinya lebih kecil dari bot

pengguna sudah menjadi lebih besar saat bot pengguna berada didekatnya. Jika hal ini terjadi, maka bot justru akan terbunuh karena termakan oleh bot musuh.

Selain penyerangan dengan cara mengejar musuh, penyerangan yang terlalu agresif melalui penembakan torpedo juga merupakan suatu hal yang kurang efektif karena walaupun aksi penembakan torpedo dapat menambah ukuran bot penembaknya apabila musuh terkena torpedo, namun aksi ini juga dapat menyebabkan ukuran bot penembak menyusut apabila tembakan tidak tepat sasaran. Jika aksi penembakan terus dilakukan secara agresif, maka terdapat kemungkinan dimana ukuran bot penembak justru kerap mengecil dan menghasilkan hasil yang tidak menguntungkan karena bot menjadi lebih rentan terbunuh oleh bot lain dan bot tidak mencari makanan untuk memperbesar dirinya, melainkan tetap fokus dalam menembak bot musuh.

Opsi strategi lainnya adalah strategi pertahanan yang menyebabkan bot menjadi pasif karena tidak dirancang untuk mengejar dan menyerang bot lain, melainkan terus menghindar dari bot lain. Aksi yang terlalu pasif ini kurang efisien karena tidak menutup kemungkinan dimana ukuran bot menjadi jauh llebih kecil dari bot musuh yang bersifat agresif dan akhirnya menyebabkan "termakannya" bot oleh bot musuh yang berukuran lebih besar. Apabila algoritma menghindar dilakukan dengan sangat baik hingga musuh tidak berhasil "memakan/membunuh" bot, maka tetap ada kemungkinan bagi musuh untuk memenangkan permainan aoabila waktu sudah habis dan ukuran musuh lebih besar ketimbang bot pengguna yang hanya berfokus pada pertahanan dari musuh karena tidak terlalu memperhatikan makanan serta memakan musuh lainnya untuk menambah ukuran.

Untuk mempermudah peninjauan efisiensi dan efektivitas setiap strategi, berikut tabel yang menampilkan efisiensi dan efektivitas dari setiap strategi:

Nama Strategi	Efisiensi	Efektivitas
Greedy Makanan	menggunakan linear search untuk mengecheck makanan terdekat. *Best case:* O(n) Ketika semua item dalam array telah terurut sehingga	+: Efektif apabila diterapkan pada kondisi dimana tidak terdapat objek merugikan (bot musuh, gas cloud, asteroid field) disekitar makanan

	hanya perlu dilakukan searching tanpa sorting *Worst case:* O(n log n) ketika jarak makanan dalam array belum terurut sehingga diperlukan algoritma sorting	-: Tidak efektif apabila terdapat musuh maupun objek merugikan seperti gas cloud dan asteroid field disekitar
Greedy Penyerangan	menggunakan linear search untuk mengecheck musuh terdekat. *Best case:*O(n) Ketika semua item dalam array telah terurut sehingga hanya perlu dilakukan searching tanpa sorting *Worst case:*O(n log n) Ketika jarak musuh dalam list belum terurut sehingga diperlukan algoritma sorting	+: Efektif ketika ukuran musuh lebih kecil dari ukuran bot dan jarak keduanya tidak terlalu jauh -: Tidak efektif ketika seluruh musuh berukuran sama atau bahkan lebih besar dari bot, dengan jarak antar bot dan musuh yang cukup jauh
Greedy Pertahanan	menggunakan linear search untuk mengecheck ancaman terdekat *Best case:*O(n) Ketiak semua item dalam array telah terurut sehingga hanya perlu dilakukan searching tanpa sorting *Worst case:*O(n)	+: Efektif ketika jumlah musuh sedikit dan area permainan masih luas karena memungkinkan pergerakan bot yang lebih leluasa. -: Tidak efektif apabila jumlah musuh cukup banyak dengan area yang sempit, sehingga menyebabkan kebingungan pemilihan langkah pada bot dan berakibat pada arah gerak bot

	Ketika jarak ancaman dalam list belum terurut sehingga diperlukan algoritma sorting	yang tidak optimal (bergerak kanan kiri secara bolak-balik karena bingung menentukan langkah selanjutnya, atau justru hanya bergerak lurus kedepan).

3.4 Strategi *Greedy* yang Diimplementasikan Dalam Program

Dalam implementasinya, kelompok kami mengkombinasikan ketiga strategi *greedy* yang telah dijelaskan dalam bab 3.2 dan 3.3, yaitu *greedy* makanan, *greedy* menyerang, dan *greedy* pertahanan guna menhasilkan hasil seoptimal mungkin untuk memenangkan permainan. Ketiga strategi dikombinasikan dengan cara melakukan pengecekan kodisi untuk pemanggilan setiap fungsi. Pemanggilan fungsi bergerak menuju makanan akan diatur sebagai mode default sehingga bot akan terus bergerak menuju makanan terdekat apabila tidak ada ancaman atau kasus khusus.

Metode penyerangan akan digunakan ketika terdapat bot lain yang berukuran lebih kecil dari RavaBot dan berjarak tidak terlalu jauh. Pada saat itu, RavaBot akan menembakkan torpedo pada bot yang berukuran lebih kecil tersebut sembari bergerak menuju makanan terdekat yang letaknya juga dekat dengan bot sasaran. Selain itu, penggunaan aksi lain seperti teleport, dan fire supernova juga akan digunakan guna melakukan optimasi penyerangan. Bot akan mencari enemi terbesar yang berukuran lebih kecil dari dirinya dan melakukan teleportasi sehingga penyerangan dilakukan ketika bot sasaran telah berada dalam hit radius RavaBot. Selain itu RavaBot juga dapat menembakkan supernova pada musuh.

Berbeda dengan metode penyerangan, metode pertahanan akan dilakukan ketika bot terdekat yang terdeteksi merupakan bot yang berukuran sama atau lebih besar sehingga berpotensi untuk memakan/menghancurkan RavaBot. RavaBot akan bergerak ke arah yang berlawanan dari bot ancaman tersebut. Walau begitu, arah yang dituju tidak benar-benar berlawanan seratus delapan puluh derajat, melainkan menuju area berlawanan paling optimal yang memungkinkan RavaBot untuk menghindar dari bot musuh tetapi juga tetap dapat melakukan aksi makan. Hal ini dikalkulasikan dengan menggunakan fungsi rasio, yaitu pembagian antara ukuran dengan jarak, sehingga rasio yang besar menandakan musuh yang paling tepat untuk dijadikan target karena merupakan bot yang lebih kecil dari RavaBot, namun

berukuran paling besar ketimbang bot kecil lainnya dan memiliki jarak paling dekat dengan RavaBot. Selain menghindar dengan cara mengubah heading, bot juga dapat melakukan aksi seperti menggunakan shield ketika berada pada jarak yang dekat dengan incoming torpedo. Selain objek yang bertipe "enemy", bentuk strategi pertahanan lainnya terdapat pada algoritma bot yang akan menghindari gas cloud serta asteroid field.

Urutan prioritas aksi bot disusun sedemikian rupa sehingga teleportasi dianggap sebagai prioritas utama karena selain menambah poin, kemungkinan untuk menambah ukuran akibat melakukan teleportasi juga sangat besar. Setelah itu, prioritas kedua berada pada aksi gerak, dimana bot akan melakukan pengecekan apakah dirinya berada pada posisi yang berdekatan dengan border area permainan. Apabila jaraknya dekat, maka bot akan bergerak ke area permainan yang lebih tengah. Hal ini bertujuan untuk mencegah terjadinya aksi "bunuh diri" pada bot. (fungsi "computeNextPlayerAction" pada file BotService.java mengandung urutan prioritas aksi bot melalui if else.)

Prioritas ketiga merupakan aksi yang mempertimbangkan musuh. Aksi ini dapat mencakup aksi penyerangan maupun aksi pertahanan. Apabila terdapat musuh dengan rasio besar, maka bot akan melakukan aksi penyerangan. Aksi penyeranganpun terbagi lagi, karena dapat dilakukan melalui berbagai cara. Jika jarak cukup jauh, maka bot akan menembak musuh dan apabila jarak tidak jauh, maka bot akan mengejar dan memakan musuh. Selain penyerangan, bot juga akan melakukan pertahanan ketika ada bot berukuran lebih besar yang berada dalam jarak yang cukup dekat dengan dirinya. Aksi pertahanan ini dilakukan dengan cara mengubah arah heading menjauhi musuh dan tetap mengutamakan heading paling optimal, dimana bot tetap melakukan aksi makan saat menghindar dari musuh. Selain itu, bot juga dapat mengaktifkan shield untuk melindungi dirinya dari serangan torpedo.

Setelah aksi yang menyangkut musuh, prioritas selanjutnya berada pada aksi yang menghindari obstacle seperti gas cloud dan asteroid fields yang dapat merugikan bot. Aksi terakhir yang belum disebut adalah aksi makan. Aksi ini debuat sedemikian rupa sehingga menjadi aksi default yang akan selalu dilakukan oleh bot. Apabila tidak ada kasus dengan prioritas yang lebih tinggi, maka bot akan bergerak kearah makanan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy*

Implementasi algoritma *greedy* dalam program ini terdapat pada file BotAttack.java, BotMovement.java, dan BotService.java yang terletak dalam folder "Services". Berikut pseudocode dari algoritma *greedy* yang digunakan:

4.1.1. Fungsi implementasi algoritma Greedy pada file "BotAttack,java"

public static PlayerAction attackEnemy(Bot player, GameObject target) { PlayerAction playerAction = new PlayerAction();

function attackEnemy(player.getBot(), target) → playerAction
{menerima input berupa bot pemain dan objek yang menjadi target. Menghasilkan output berupa tipe data class playerAction, yaitu aksi yang akan
dilakukan oleh bot.}
KAMUS LOKAL
{ playerAction memiliki tipe PlayerAction }
playerAction: PlayerAction

ALGORITMA
{ keterangan: getToll() menghasilkan batas jarak yang ditoleransi }

{ Kondisi ketika musuh berada pada jarak dekat (kurang atau sama dengan batas toleransi jarak yang telah dikalkulasi) }
if (BotUtil.getActualDistance(player.getBot(), target) <= player.getToll() / 2) then
    if (player.isAfterburnerActive()) then
        { bot akan menghentikan afterburner }
        playerAction.action ← PlayerActions.STOPAFTERBURNER
    else
        { bot akan menembakkan torpedo kearah musuh }
        playerAction.action ← PlayerActions.FIRETORPEDOES
        playerAction.heading ← player.getHeadingBetween(target)
else if (player.getBot().getSize() >= 50 && !player.isAfterburnerActive()) then

```
                    { bot akan menyalakan afterburner }
                    playerAction.action ← PlayerActions.STARTAFTERBURNER
    else if (player.isAfterburnerActive() && player.getBot().getSize() <= 50) then
            { bot akan menghentikan afterburner }
            playerAction.action ← PlayerActions.STOPAFTERBURNER
    else
            { bot akan bergerak maju untuk mengejar musuh }
            playerAction.action ← PlayerActions.FORWARD
            playerAction.heading ← player.getHeadingBetween(target);

    { mengembalikan return berupa playerAction }
    → playerAction
```

4.1.2. Fungsi implementasi algoritma Greedy pada file "BotMovement,java"

public static int getOptimalHeading(Bot player, int toll)

```
    function getOptimalHeading(Bot player, int toll)
    {menerima masukan berupa bot pemain dan toleransi jarak.

    Menghasilkan heading yang paling optimal}

    KAMUS LOKAL
    allObject: list of GameObject
    nearestConsumables: list of GameObject
    nearestEnemies: list of GameObject

    ALGORITMA
    { urutan kondisi (if) disesuaikan berdasarkan skala prioritas strategi algoritma }

    { kondisi ketika musuh terdekat berukuran lebih besar}
    if (nearestEnemy ≠ null) then
        if (nearestEnemy.getSize() >= player.getBot().getSize()) then
            →BotMovement.avoidThreatHeading(player, allObjects, nearestEnemy, 45, 45)
```

```
{ kondisi ketika tidak ada musuh yang menjadi persoalan, sehingga
mempertimbangkan objek consumables }
if (nearestConsumable ≠ null) then
    if (Math.abs(player.getHeadingBetween(nearestConsumable) -
    player.getBot().currentHeading) > 165) then
        →player.getBot().currentHeading
    →player.getHeadingBetween(nearestConsumable)
→player.getHeadingBetween(player.getCenterPoint());
```

4.1.3 Fungsi implementasi algoritma Greedy pada file "BotMovement,java"

private GameObject getHighestRatioEnemy()

```
function getHighestRatioEnemy()
{fungsi yang mengkalkulasi densitas musuh untuk mengoptimalkan keuntungan.
Perhitungan dilakukan dengan membagi ukuran dan jarak, dan menghasilkan rasio
musuh maksimum.}

KAMUS LOKAL
maxRatio: double
maxRatioPlayer: GameObject

ALGORITMA
if (!gameState.getPlayerGameObjects().isEmpty()) then
        for (GameObject player : gameState.getPlayerGameObjects())
          if (player.getId() != this.bot.getId() && player.getSize() /
          BotUtil.getActualDistance(this.bot, player) > maxRatio) then
            maxRatio ← player.getSize() / BotUtil.getActualDistance(this.bot, player)
            maxRatioPlayer ← player
→maxRatioPlayer
```

public void computeNextPlayerAction(PlayerAction playerAction)

```
procedure computeNextPlayerAction(PlayerAction playerAction)
{I.S. playerAction yang bertipe PlayerAction}
{f.s. playerAction}

KAMUS LOKAL
largestSmallerEnemy: gameObject
nearestEnemies: list of gameObject
nearObstacles: list of gameObject
nearestObjectOutside: list of gameObject
isOk: boolean

ALGORITMA
if (!gameState.getPlayerGameObjects().isEmpty()) then

    {teleporting}
    if (this.isFiringTeleporter&&this.getNearestObjects(gameState.getGameObjects(),
    "TRAVERSAL").isEmpty()  && gameState.getWorld().getCurrentTick() –
    this.fireTeleporterTick > 40) then
          this.isFiringTeleporter ← false
    for (GameObject player : gameState.getPlayerGameObjects())
        if (player.getId() != this.bot.getId() && this.bot.getSize() - 15 >
        player.getSize() then
                largestSmallerEnemy ← player
    if (this.isFiringTeleporter && !gameState.getGameObjects().isEmpty() &&
    largestSmallerEnemy != null) then
        for (GameObject teleporter : gameState.getGameObjects())
            if if (player.getId() != this.bot.getId() && this.bot.getSize() - 15 >
            player.getSize() && player.getSize() > largestSmallerEnemySize) then
                largestSmallerEnemy ←player
   if (this.isFiringTeleporter && !gameState.getGameObjects().isEmpty() &&
   largestSmallerEnemy != null) then
        if (teleporter.getGameObjectType() == ObjectTypes.TELEPORTER)  then
            if (BotUtil.getActualDistance(largestSmallerEnemy, teleporter) <
```

```
            this.bot.getSize() && largestSmallerEnemy.getSize() < this.bot.getSize())
            then
                playerAction.action ← PlayerActions.TELEPORT
                playerAction.heading ← BotMovement.getOptimalHeading(this,
                this.toll)
                playerAction ← BotAttack.checkAfterburner(this, playerAction)
                playerAction ←playerAction
                isFiringTeleporter ← false

    if (!this.isFiringTeleporter && largestSmallerEnemy != null && this.bot.getSize()
    > 80 && this.bot.getSize() > largestSmallerEnemy.getSize() &&
    this.bot.getTeleportCount() > 0) then
        playerAction.action ← PlayerActions.FIRETELEPORT
        playerAction.heading ← getHeadingBetween(largestSmallerEnemy)
        playerAction ← BotAttack.checkAfterburner(this, playerAction)

        playerAction ← playerAction
        fireTeleporterTick ← gameState.getWorld().getCurrentTick()
        isFiringTeleporter ← true;

    {nearest enemy}
    nearestEnemies ← getNearestObjects(gameState.getPlayerGameObjects(),
    "ENEMY")
    if (!nearestEnemies.isEmpty()) then
        nearestEnemy ← nearestEnemies.get(0)
    else
        nearestEnemy ←null

    {nearest consumables}
    nearestConsumables ← getNearestObjects(gameState.getPlayerGameObjects(),
    "CONSUMABLES")

if (!nearestConsumables.isEmpty()) then
        nearestConsumable ← nearestConsumables.get(0)
```

```
else
        nearestConsumable = null

{Check if out of bounds}
if (BotUtil.getDistanceBetween(centerPoint, bot) + (1.75 * this.bot.getSize()) + 50 >
this.gameState.getWorld().getRadius()) then
        playerAction.action ← PlayerActions.FORWARD
        if (nearestConsumable != null) then
            playerAction.heading ← this.getHeadingBetween(nearestConsumable)
        else
            playerAction.heading ← this.getHeadingBetween(centerPoint)
        playerAction ← BotAttack.checkAfterburner(this, playerAction)
        playerAction ← playerAction

if (this.getBot().getTorpedoCount() >= 5 && this.getBot().getSize() >= 50) then
        isOk ← BotAttack.fireRandomTorpedo(this)

if (nearestEnemy != null && this.getBot().getSize() - nearestEnemy.getSize() > 0
&& BotUtil.getActualDistance(this.getBot(), nearestEnemy) < this.toll) then
        {menyerang musuh}
        playerAction ← BotAttack.attackEnemy(this, nearestEnemy)
        playerAction←playerAction

{check if near or in an obstacle}
if (!nearestObstacles.isEmpty()) then
        nearestObstacle ← nearestObstacles.get(0)
else
        nearestObstacle ← null

if (nearestObstacle != null && BotUtil.getDistanceBetween(nearestObstacle,
this.bot) < (1.75 * this.bot.getSize() + nearestObstacle.getSize())) then
        nearestObjectOutside ← this.getNearestObjects(gameState.getGameObjects(),
        "CONSUMABLES").stream().filter(item ->BotUtil.getDistanceBetween(item,
        nearestObstacle) > (1.75 * this.bot.getSize() + nearestObstacle.getSize()))
```

```
            .sorted(Comparator.comparing(item -> BotUtil.getDistanceBetween(this.bot,
            item))).collect(Collectors.toList()) then
                if (!nearestObjectOutside.isEmpty()) then
                    playerAction.heading ← getHeadingBetween
                    (nearestObjectOutside.get(0))
                    playerAction.action ← PlayerActions.FORWARD
                    playerAction ← BotAttack.checkAfterburner(this, playerAction)
                    playerAction ← playerAction

        {consider other targets}
        playerAction.heading ←BotMovement.getOptimalHeading(this, this.toll)
         playerAction.action ← PlayerActions.FORWARD

        playerAction ← BotAttack.checkAfterburner(this, playerAction)
        playerAction ← playerAction
```

4.2. Struktur Data Program Galaxio

Struktur data yang digunakan pada permainan Galaxio berbasiskan kelas/*class.* Terdapat beberapa kelas yang diberikan oleh disediakan oleh starter pack dari permainan Galaxio yang telah disediakan oleh Entelect, namun kami menambahkan beberapa atribut serta mengubah method dari kelas tersebut sebagaimana algoritma greedy diterapkan. Kelas-kelas tersebut antara lain adalah GameObject, GameState, PlayerAction, Position, World, Bot, BotAttack, BotMovement, BotService, dan BotUtil.

a. Class GameObject

   Kelas GameObject merupakan kelas yang menyimpan objek-objek yang valid dalam permainan, dan setiap objek memiliki atribut seperti id, size, speed, currentHeading, position, gameObjectType, position, gameObjectType, effects, TorpedoSalvoCount, SupernovaAvailable, TeleportCount, dan ShieldCount. Dalam hal ini, atribut "gameObjectType" menunjukkan jenis objek. Sebagai contoh, gameObjectType berupa "ENEMY" yang menandakan bahwa objek tersebut adalah bot musuh.

b. Class GameState

Kelas ini berfungsi untuk menenjukan keadaan permainan dan terdiri dari tiga atriibut, yaitu world, gameObjects, dan playerGameObjects.

c. Class PlayerAction

Kelas PlayerAction mendeskripsikan aksi dari pemain dan memiliki tiga atribut yaitu playerId yang menunjukkan pemain, action yang merupakan class PlayerActions dan menggambarkan aksi dari pemain, serta heading yang menunjukkan arah gerak pemain.

d. Class Position

Kelas position menunjukkan posisi pemain berdasarkan koordinat, sehingga memiliki dua atribut yaitu x sebagai absis dan y sebagai ordinat.

e. Class World

Kelas World menunjukkan area yang valid dalam permainan dan terdiri dari tiga atribut, yaitu CenterPoint yang menunjukkan koordinat titik tengah area permainan, radius yang menunjukkan radius area permainan, dan currentTick yang menunjukkan waktu/tik permainan.

f. Class Bot

Kelas bot menyimpan atribut-atribut yang dimiliki oleh suatu bot. Atribut-atribut tersebut antara lain adalah bot yang merupakan kelas dari GameObject, playerAction yang merupakan kelas dari PlayerAction, gameState yang merupakan kelas dari GameState, CenterPoint yang merupakan kelas dari GameObject, isFiringTeleporter yang bertipe boolean, dan fireTeleporterTick yang bertipe integer.

g. Class BotAttack

Kelas BotAttack tidak menyimpan atribut khusus seperti pada kelas-kelas lainnya, namun kelas ini berisikan method-method yang akan digunakan oleh bot ketika akan melakukan penyerangan. Beberapa method yang terdapat dalam kelas ini antara lain adalah boolean isAligned untuk mengecheck apakah alignment dari bot dan target telah berada pada rentang threshold tertentu,

method attackEnemy untuk mengontrol logika dari suatu bot ketika melakukan penyerangan, dan method checkAfterburner untuk menghentikan afterburner ketika bot tidak sedang berada dalam kondisi menyerang

h. Class BotMovement

Sama seeprti kelas BotAttack, kelas BotMovement juga tidak memiliki atribut khusus, namun terdiri dari method-method yang digunakan dalam pergerakan robot. Method-method tersebut antara lain adalah method avoidThreatHeading yang menghasilkan integer berupa heading ancaman, serta getOptimalHeading yang menghasilkan int dan berguna untuk mencari heading yang optimal.

i. Class BotService

Kelas BotService juga tidak memiliki atribut khusus, namun berisikan method-method untuk algoritma bot. Tidak spesifik untuk bergerak, menyerang, atau bertahan, method-method pada BotService bersifat lebih umum karena implementasinya dapat digunakan dalam berbagai aksi. Salah satu contoh dari method utama adalah method computeNextPlayerAction yang berguna untuk menentukan aksi apa yang akan dilakukan oleh bot. Entah aksi tersebut mmerupakan aksi penyerangan, pertahanan, atau bergerak mencari makan.

j. Class BotUtil

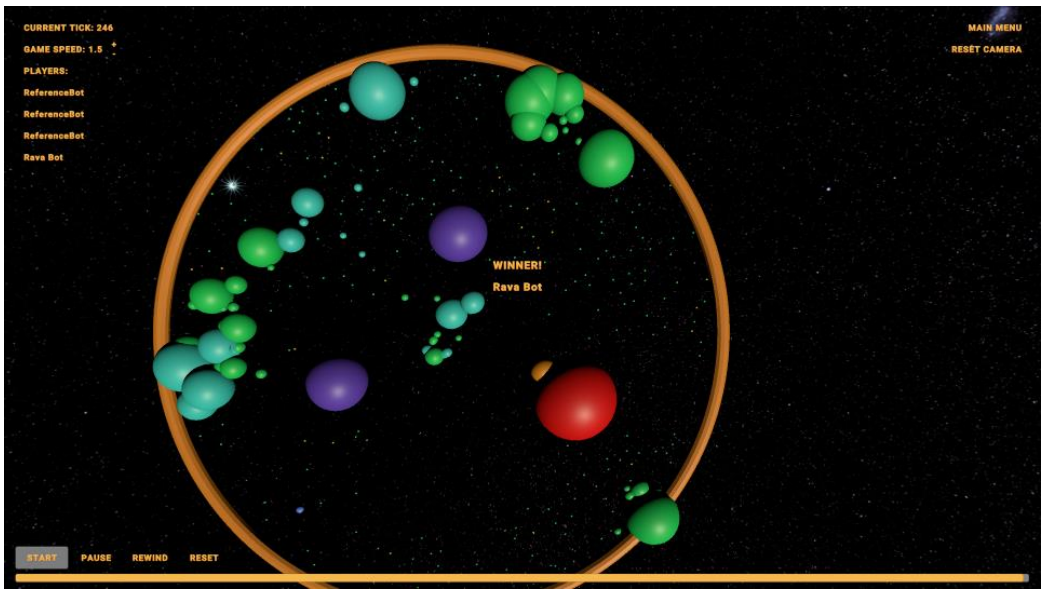
Kelas BotUtil memiliki berbagai method yang implementasinya bersifat umum dan tidak digunakan secara eksplisit, melainkan dipanggil dalam method-method lainnya. Method-method yang terdapat dalam kelas ini antara lain adalah getDistanceBetween untuk mendapatkan jarak antara dua objek, getActualDistance untuk mendapatkan jarak aktual dari dua objek, getDistanceToNearestBorder untuk menentukan jarak antara suatu objek dengan border area permainan, writeToFile untuk pencatatan gameState, dan getHeadingBetween untuk menentukan heading antara dua objek

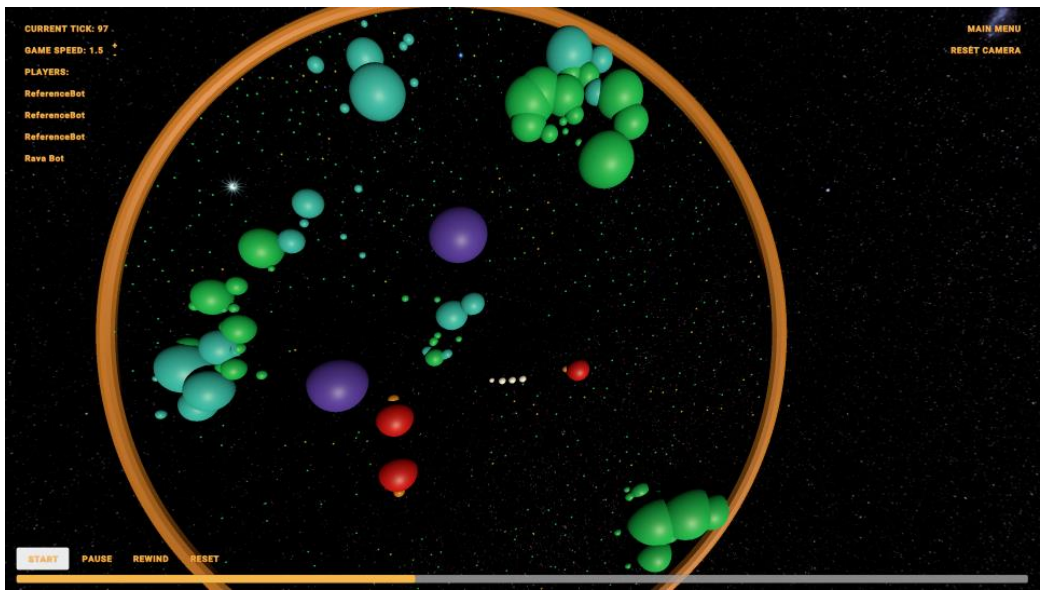
4.3. Analisis Pengujian Algoritma Greedy

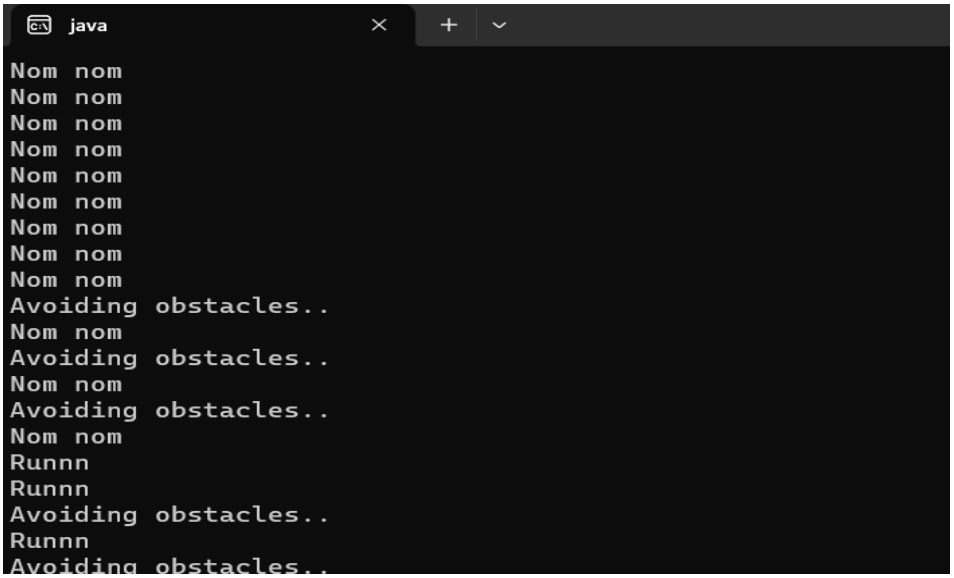
Percobaan untuk mengambil data uji dilakukan dengan melakukan clean maven package dengan *command* ‘mvn clean package’ hingga terbentuk *executable jar file*. Dalam

kasus ini, file jar telah tersedia dalam folder target. Setelah itu, jalankan `run.bat`yang telah tersedia pada folder starter-pack dan kini gameplay telah siap untuk divisualisasikan. Pada kesempatan kali ini, kami melakukan tiga kali pengujian terhadap bot yang telah kami rancang.

1. Pengujian Pertama



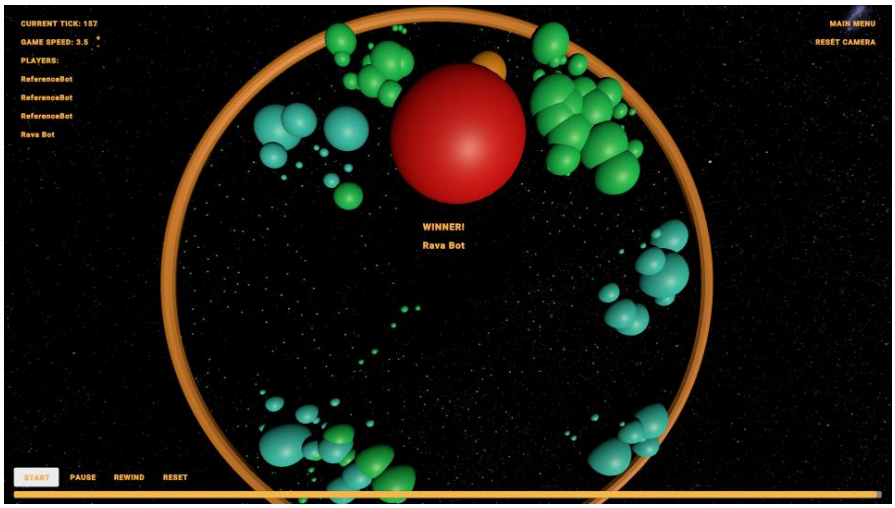


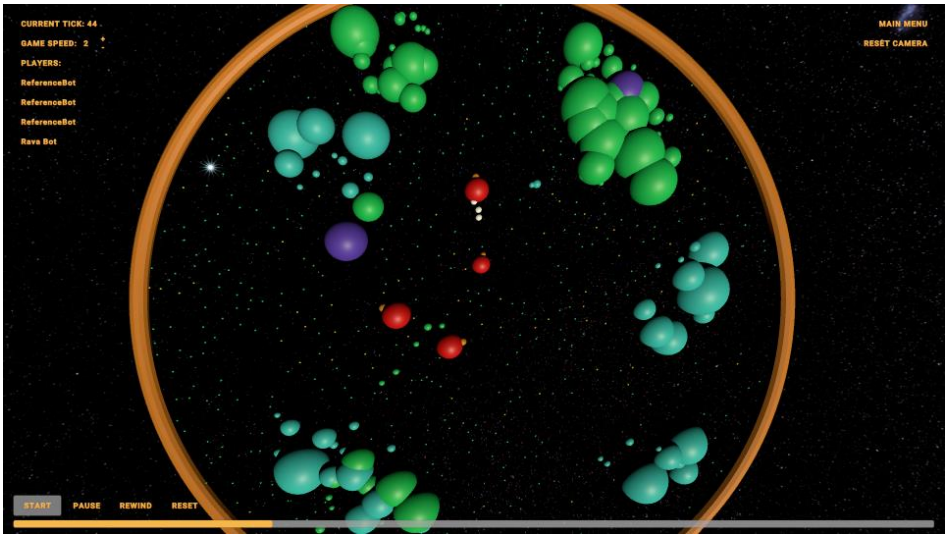


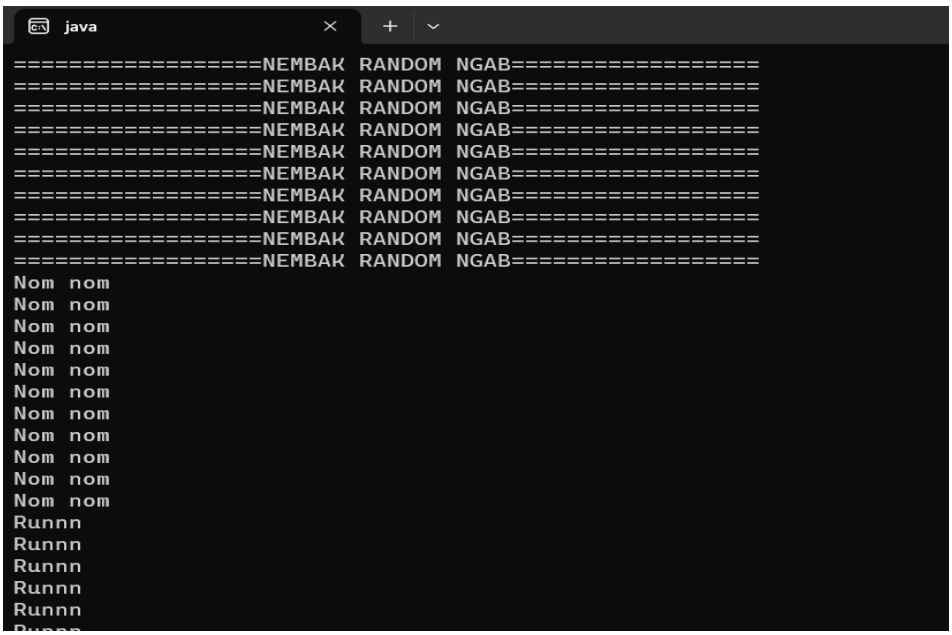
Pada percobaan pertama, bot kami berhasil mengalahkan *reference bot* yang telah disediakan oleh Entelect dengan cara menyerang bot musuh lalu kabur dan makan, lalu menyerang lagi kemudian kabur untuk makan lagi. Hal tersebut diulangi hingga bot sudah berukuran lebih besar dari bot musuh hingga akhirnya bot akan mengejar bot musuh. Berikut visualisasi pada akhir permainan ketika bot sudah menang dan ketika bot sedang menembak bot lain yang ukurannya lebih besar. Foto terakhir menampilkan terminal java, menunjukkan aksi yang sedang dilakukan oleh robot setiap saat. "Nom nom" menandakan bot sedang melakukan aksi makan, "Avoiding obstacles" menandakan bot yang sedang menghindari obstacle seperti gas cloud, dan "Run" menandakan bot yang sedang menghindar dari bot musuh.

Dari percobaan ini terbukti bahwa algoritma gabungan dari algoritma *greedy* aksi makan, aksi menyerang, dan aksi pertahanan membuahkan hasil yang optimal untuk memenuhi objektif dari permainan, yaitu memenangkan permainan. *Greedy* dalam hal makan terbukti dari bot yang selalu mencari arah dimana terdapat makanan, bahkan bot akan terus melakukan aksi makan sembari melakukan pertahanan maupun penyerangan. Selain itu, implementasi *greedy* lainnya adalah dalam pemilihan aksi yang berdasarkan pada rasio dari hasil pembagian ukuran dengan jarak. Bot akan memilih musuh dengan rasio yang lebih tinggi karena menandakan bahwa musuh berukuran besar namun berada pada jarak dekat. Agar bot tidak mendekatkan diri pada musuh yang berukuran lebih besar darinya, maka terdapat fungsi pengecekan yang memvalidasi apakah ukuran musuh lebih besar atau lebih kecil dari bot. Pada pengujian ini, permainan berhasil dimenangkan pada tik ke 246.

2. Pengujian Kedua



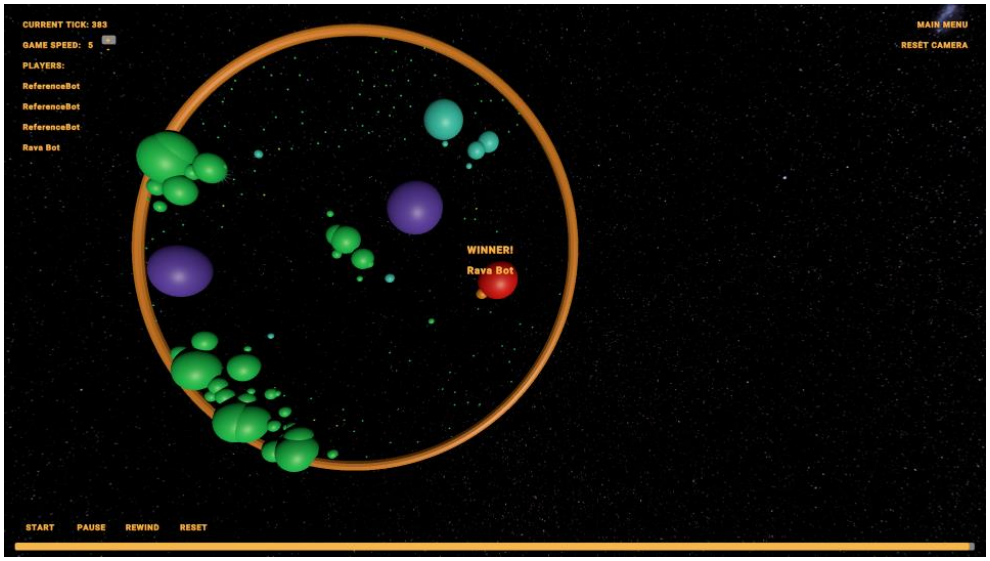


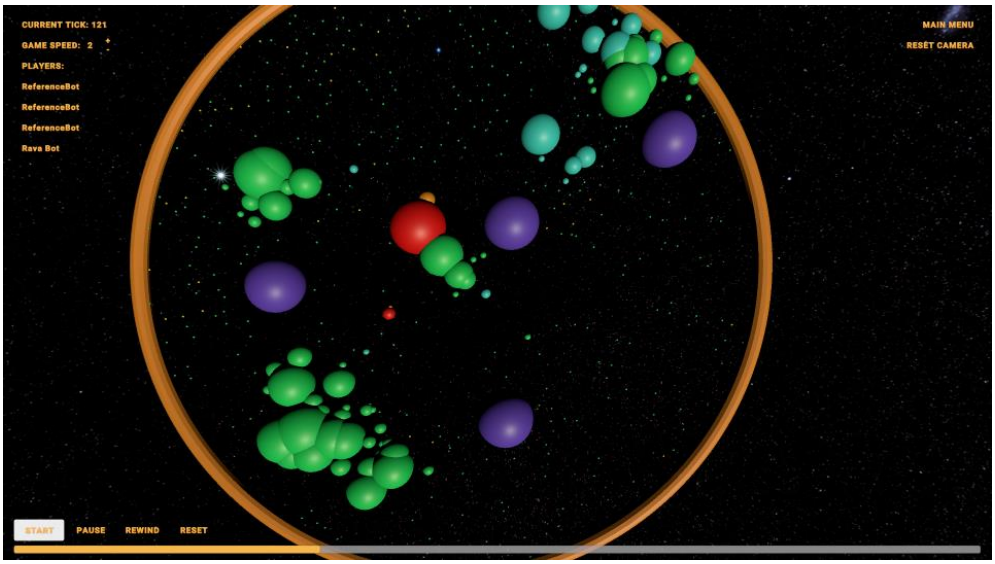


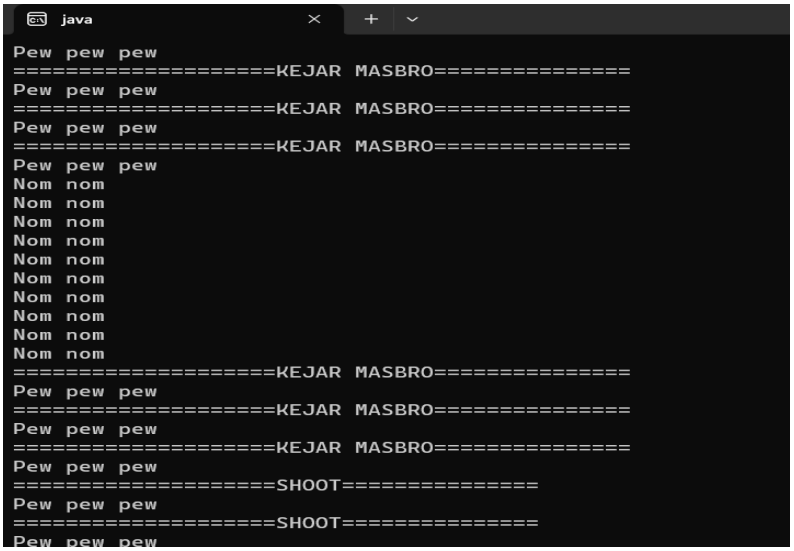
Pada pengujian kedua, bot berhasil mengalahkan refereneBot dengan strategi yang mirip dengan percobaan pertama, yaitu mencari musuh dengan rasio tertinggi dan

menyerangnya sembari terus melakukan aksi makan. Pada terminal, “NEMBAK RANDOM NGAB” menunjukkan aksi bot yang sedang menembak musuh paling optimal berdasarkan rasio. Kata “random” dipilih untuk menandakan aksi bot yang offensive karena telah memiliki jumlah torpedo_count yang cukup tinggi. Walau demikian, arah tembakan tetap berdasarkan kalkulasi paling optimal dengan algoritma *greedy*. Dalam percobaan ini, permainan berhasil dimenangkat pada tick ke-157.

3. Pengujian Ketiga







Pengujian ketiga lebih menarik ketimbang pengujian pertama dan kedua, karena pada pengujian ketiga terdapat kasus dimana bot menjadi berukuran sangat kecil dan musuh memiliki ukuran yang jauh lebih besar, sebagaimana ditampilkan pada gambar kedua. Dalam hal ini, terbukti bahwa algoritma *greedy* yang diimplementasikan telah berhasil memenuhi objektif untuk memenangkan permainan. Ketika ukuran bot sangat kecil, maka ia akan terus kabur dari musuh sembari mencari makan dan menggunakan teleport untuk menyerang musuh ketika perbedaan ukuran keduanya sudah tidak terlalu signifikan. Foto ketiga merupakan terminal yang menunjukkan aksi makan dan penyerangan. “pew pew” menandakan aksi penyerangan yang kemudian terbagi lagi menjadi dua, yaitu aksi menyerang dengan mengejar dan aksi menyerang dengan menembak. “KEJAR MASBRO” menunjukkan aksi penyerangan dengan mengejar bot musuh dan “SHOOT” menunjukkan aksi penyerangan dengan menembakkan torpedo. Permainan ketiga ini berhasil dimenangkan oleh RavaBot pada tick ke-383.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Algoritmma *greedy* yang ditujukan untuk memecahkan permasalah optimasi dapat diimplementasikan dalam permainan Galaxio untuk mencapai tujuan objektif dari permainan ini, yaitu memenangkan permainan baik dengan cara mengeliminasi seluruh bot lawan dengan cara mengejar dan memakannya serta menembakkan torpedo pada bot lawan, memenangkan

permainan dengan cara terus melakukan pertahanan sehingga menjadi “last man standing”, atau memenangkan permainan dengan cara memaksimalkan aksi makan sehingga menjadi bot dengan ukuran terbesar dan berakhir dengan memenangkan permainan. Selain itu, algoritma *greedy* juga dapat digunakan dalam memilih method mana yang akan dijalankan oleh bot, sehingga arah gerak dan aksi yang dilakukan oleh robot selalu merupakan opsi terbaik yang paling optimal dari setiap langkah.

Meski demikian, pada tugas besar 1 Strategi Algoritma ini kami menyadari bahwa algoritma *greedy* yang dibuat tidak selalu membuahkan hasil yang ppaling optimal dalam implementasi permainan Galaxio. Hal ini dapat terlihat ketika algoritma dibuat sedemikian rupa sehingga hanya berfokus pada optimasi salah satu aspek, baik makanan, pertahanan, penyerangan, dan densitas tidak akan membuahkan hasil optimal karena terbukti menghasilkan *win rate* terhadap reference bot yang lebih rendah ketimbang menggunakan algoritma yang merupakan gabungan dari aspek-aspek tersebut. Agar menghasilkan hasil yang optimal dan memenuhi objektif dari game yatu memenangkan permainan, kami menggabungkan strategi algoritma *greedy* untuk penyerangan, pertahanan, aksi memakan, dan densitas yang disesuaikan dengan kondisi selama permainan berlangsung.

5.2. Saran

Pengimplementasian algoritma *greedy* pada permainan Galaxio masih dapat dioptimasi lagi dengan memperhatikan detail kegunaan dari masing-masing objek pada game. Sebelum melakukan implementasi algoritma, penting untuk membaca dan memahami peraturan serta objek pada permainan yang telah disediakan oleh Entellect karena jumlahnya yang cukup banyak. Pemahaman terkait peraturan dan detail dari setiap objek akan sangat membantu dalam mengimplementasikan algoritma. Selain itu, hal ini ditujukan agar programmer mengerti atribut dan method dasar apa saja yang sudah menjadi bawaan dasar dari permainan.

DAFTAR PUSTAKA

Entellect. 2021. “Entellect Challenge 2021 - Galaxio” dari 2021-Galaxio/game-rules.md at develop · EntelectChallenge/2021-Galaxio · GitHub

Munir, R. (2021). Algoritma Greedy Bagian 1[PDF]. Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

Munir, R. (2021). Algoritma Greedy Bagian 2[PDF]. Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

Munir, R. (2021). Algoritma Greedy Bagian 3[PDF]. Institut Teknologi Bandung. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf