

**TUGAS BESAR 1 IF2211 STRATEGI ALGORITMA**  
**SEMESTER II TAHUN 2022/2023**  
**PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN**  
**“GALAXIO”**



Disusun Oleh:

**SiPalingGreedy**

13521059 Arleen Chrysantha Gunardi

13521124 Michael Jonathan Halim

13521127 Marcel Ryan Antony

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**

**2023**

## DAFTAR ISI

|                                                                              |           |
|------------------------------------------------------------------------------|-----------|
| <b>DAFTAR ISI</b>                                                            | <b>2</b>  |
| <b>DAFTAR TABEL</b>                                                          | <b>5</b>  |
| <b>DAFTAR GAMBAR</b>                                                         | <b>6</b>  |
| <b>BAB I DESKRIPSI TUGAS</b>                                                 | <b>7</b>  |
| 1.1 Deskripsi Masalah                                                        | 7         |
| <b>BAB II LANDASAN TEORI</b>                                                 | <b>10</b> |
| 2.1 Algoritma Greedy                                                         | 10        |
| 2.3 Garis Besar Cara Kerja Bot dan Game Object pada Permainan Galaxio        | 13        |
| 2.4 Garis Besar Implementasi Algoritma Greedy ke dalam Bot Permainan Galaxio | 15        |
| 2.5 Metode Kalkulasi Sudut Antar Garis Singgung Lingkaran dari Sebuah Titik  | 15        |
| <b>BAB III APLIKASI STRATEGI GREEDY</b>                                      | <b>17</b> |
| 3.1 Ikhtisar Strategi Permainan                                              | 17        |
| 3.2. Pemetaan Elemen / Komponen Algoritma Greedy pada Bot Permainan Galaxio  | 17        |
| 3.2.1 Greedy by Scanning Area                                                | 17        |
| 3.2.2 Greedy by Makanan                                                      | 18        |
| 3.2.3 Greedy by Musuh                                                        | 19        |
| 3.2.4 Greedy by Obstacle                                                     | 19        |
| 3.2.5 Greedy by Supernova                                                    | 20        |
| 3.2.6 Greedy by Teleport                                                     | 21        |
| 3.2.7 Greedy by Torpedo                                                      | 22        |
| 3.2.8 Greedy by Afterburner                                                  | 22        |
| 3.2.9 Greedy by Shield                                                       | 23        |
| 3.3 Eksplorasi Alternatif Solusi Algoritma Greedy pada Bot Permainan Galaxio | 24        |
| 3.3.1 Greedy by Wormhole                                                     | 24        |
| 3.3.2 Greedy by Tick                                                         | 24        |
| 3.3.3 Greedy by Supernova with the Most Enemies                              | 24        |
| 3.3.4 Greedy by Teleport with the Most Smaller Enemies                       | 25        |
| 3.3.5 Greedy by Edge                                                         | 25        |
| 3.4 Analisis Efisiensi dari Kumpulan Solusi Algoritma Greedy                 | 25        |
| 3.4.1 Analisis Efisiensi dari Greedy by Scanning Area                        | 26        |
| 3.4.2 Analisis Efisiensi dari Greedy by Makanan                              | 26        |
| 3.4.3 Analisis Efisiensi dari Greedy by Musuh                                | 26        |
| 3.4.4 Analisis Efisiensi dari Greedy by Obstacle                             | 26        |
| 3.4.5 Analisis Efisiensi dari Greedy by Supernova                            | 26        |

|                                                                  |           |
|------------------------------------------------------------------|-----------|
| 3.4.6 Analisis Efisiensi dari Greedy by Teleport                 | 27        |
| 3.4.7 Analisis Efisiensi dari Greedy by Torpedo                  | 27        |
| 3.4.8 Analisis Efisiensi dari Greedy by Afterburner              | 27        |
| 3.4.9 Analisis Efisiensi dari Greedy by Shield                   | 27        |
| 3.5 Strategi Greedy yang Digunakan pada Program Bot              | 27        |
| <b>BAB IV IMPLEMENTASI DAN PENGUJIAN</b>                         | <b>30</b> |
| 4.1 Implementasi Algoritma Greedy pada Bot Permainan Galaxio     | 30        |
| 4.1.1 Object Type (ObjectTypes.java)                             | 30        |
| 4.1.2 Player Action (PlayerActions.java)                         | 31        |
| 4.1.3 Main (computeNextAction)                                   | 31        |
| 4.1.4 Scan Area (scanArea)                                       | 40        |
| 4.1.5 Fire Supernova (checkFireSupernova)                        | 45        |
| 4.1.6 Detonate Supernova (checkDetonateSupernova)                | 46        |
| 4.1.7 Shield (checkShieldCondition)                              | 47        |
| 4.1.8 Enemy Chase (checkEnemyChase)                              | 49        |
| 4.1.9 Owner Teleporter (checkOwnerTeleporter)                    | 49        |
| 4.1.10 Getting Teleporter (getTeleporter)                        | 50        |
| 4.1.11 Teleport (doTeleport)                                     | 50        |
| 4.1.12 Edge Constraint (sizeUntukEdge)                           | 51        |
| 4.1.13 Distance Between Two Objects (getDistanceBetween)         | 51        |
| 4.1.14 Distance Between an Object and World (getDistanceBetween) | 52        |
| 4.1.15 Heading From Bot to Object (getHeadingBetween)            | 53        |
| 4.1.16 Heading From Object to Bot (getHeadingObject)             | 54        |
| 4.1.17 Heading From Bot to World Center (getHeadingBetween)      | 55        |
| 4.1.18 Convert Radian to Degrees (toDegrees)                     | 56        |
| 4.2 Penjelasan Struktur Data pada Bot Permainan Galaxio          | 57        |
| 4.2.1 BotService (BotService.java)                               | 57        |
| 4.2.2 GameState (GameState.java)                                 | 58        |
| 4.2.3 World (World.java)                                         | 59        |
| 4.3 Pengujian Bot Permainan Galaxio                              | 59        |
| 4.3.1 Pengujian Greedy by Scan Area                              | 60        |
| 4.3.2 Pengujian Greedy by Afterburner                            | 61        |
| 4.3.3 Pengujian Greedy by Torpedo                                | 62        |
| 4.3.4 Pengujian Greedy by Supernova                              | 63        |
| 4.3.5 Pengujian Greedy by Teleport                               | 64        |
| 4.3.6 Pengujian Greedy by Shield                                 | 66        |
| 4.3.7 Pengujian Greedy by Makanan                                | 67        |
| 4.3.8 Pengujian Greedy by Musuh                                  | 68        |

|                                    |           |
|------------------------------------|-----------|
| 4.3.9 Pengujian Greedy by Obstacle | 70        |
| <b>BAB V PENUTUP</b>               | <b>72</b> |
| 5.1 Simpulan                       | 72        |
| 5.2 Saran                          | 72        |
| 5.3 Refleksi                       | 73        |
| <b>DAFTAR PUSTAKA</b>              | <b>74</b> |
| <b>LAMPIRAN</b>                    | <b>75</b> |

## DAFTAR TABEL

|                                                               |    |
|---------------------------------------------------------------|----|
| <b>Tabel 3.2.1</b> Komponen Algoritma Greedy by Scanning Area | 17 |
| <b>Tabel 3.2.2</b> Komponen Algoritma Greedy by Makanan       | 18 |
| <b>Tabel 3.2.3</b> Komponen Algoritma Greedy by Musuh         | 19 |
| <b>Tabel 3.2.4</b> Komponen Algoritma Greedy by Obstacle      | 20 |
| <b>Tabel 3.2.5</b> Komponen Algoritma Greedy by Supernova     | 21 |
| <b>Tabel 3.2.6</b> Komponen Algoritma Greedy by Teleport      | 21 |
| <b>Tabel 3.2.7</b> Komponen Algoritma Greedy by Torpedo       | 22 |
| <b>Tabel 3.2.8</b> Komponen Algoritma Greedy by Afterburner   | 23 |
| <b>Tabel 3.2.9</b> Komponen Algoritma Greedy by Shield        | 23 |

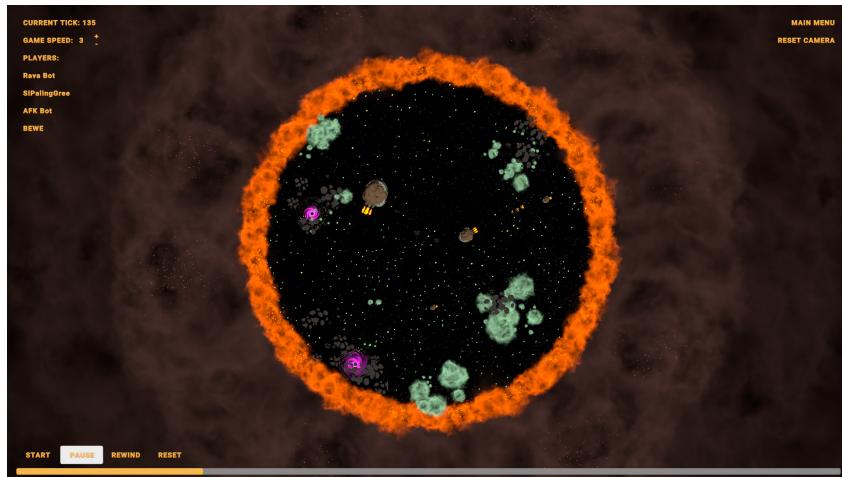
## DAFTAR GAMBAR

|                                                                                |    |
|--------------------------------------------------------------------------------|----|
| <b>Gambar 1.1</b> Tampilan Antarmuka Visualizer Galaxio                        | 6  |
| <b>Gambar 2.5.1</b> Sudut Antara Garis Singgung Lingkaran                      | 15 |
| <b>Gambar 4.1.1</b> Ilustrasi Scan Area Menghindari Musuh                      | 45 |
| <b>Gambar 4.1.2</b> Ilustrasi Cek Torpedo Untuk Shield                         | 48 |
| <b>Gambar 4.1.3</b> Ilustrasi Hitung Jarak Antara Bot dengan Game Object       | 52 |
| <b>Gambar 4.1.4</b> Ilustrasi Hitung Jarak Antara Bot dengan Titik Tengah Peta | 53 |
| <b>Gambar 4.1.5</b> Ilustrasi Hitung Sudut Antara Bot dengan Objek Lain        | 54 |
| <b>Gambar 4.1.6</b> Ilustrasi Hitung Sudut Antara Objek Lain dengan Bot        | 55 |
| <b>Gambar 4.1.7</b> Ilustrasi Hitung Sudut Antara Bot dengan Titik Pusat Peta  | 56 |
| <b>Gambar 4.3.1</b> Bot Melakukan Scan Area                                    | 60 |
| <b>Gambar 4.3.2</b> Bot Menuju Daerah Optimal                                  | 60 |
| <b>Gambar 4.3.3</b> Bot Saat Melakukan Aksi ACTIVATEAFTERBURNER                | 61 |
| <b>Gambar 4.3.4</b> Bot Saat Melakukan Aksi STOPAFTERBURNER                    | 61 |
| <b>Gambar 4.3.5</b> Bot Saat Melakukan Aksi FIRETORPEDOES                      | 62 |
| <b>Gambar 4.3.6</b> Bot Saat Melakukan Aksi FIRESUPERNOVA                      | 63 |
| <b>Gambar 4.3.7</b> Bot Saat Melakukan Aksi DETONATESUPERNOVA                  | 63 |
| <b>Gambar 4.3.8</b> Bot Saat Melakukan Aksi FIRETELEPORT                       | 64 |
| <b>Gambar 4.3.9</b> Bot Saat Melakukan Aksi TELEPORT                           | 65 |
| <b>Gambar 4.3.10</b> Bot Saat Melakukan Aksi ACTIVATESHIELD                    | 66 |
| <b>Gambar 4.3.11</b> Bot Saat Sedang Melakukan Aksi Greedy by Makanan (1)      | 67 |
| <b>Gambar 4.3.12</b> Bot Saat Sedang Melakukan Aksi Greedy by Makanan (2)      | 67 |
| <b>Gambar 4.3.13</b> Bot Sebelum Melakukan Aksi Greedy by Musuh                | 68 |
| <b>Gambar 4.3.14</b> Bot Setelah Melakukan Aksi Greedy by Musuh                | 69 |
| <b>Gambar 4.3.15</b> Bot Menghindari Obstacle Gas Cloud                        | 70 |
| <b>Gambar 4.3.16</b> Bot Menghindari Obstacle Asteroid Field                   | 70 |

# BAB I

## DESKRIPSI TUGAS

### 1.1 Deskripsi Masalah



Gambar 1.1 Tampilan Antarmuka Visualizer Galaxio

Galaxio merupakan suatu permainan pertandingan antar bot kapal pemainnya yang berlatar belakang di luar angkasa. Setiap pemain diwakilkan oleh sebuah bot kapal pada permainan. Satu pemain yang tetap bertahan hingga akhir permainan akan menjadi pemenangnya. Permainan berakhir ketika hanya tersisa satu pemain yang bertahan. Untuk memenangkan permainan, setiap bot kapal dapat memakan makanan atau bot kapal pemain lain yang lebih kecil, menembakkan torpedo, mengaktifkan shield, dan lain-lain. Kemenangan dapat diraih dengan strategi permainan yang baik. Oleh karena itu, dibuatlah suatu *game engine* yang mengimplementasikan strategi algoritma Greedy.

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 disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal player. Apabila player mengkonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatkannya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada

lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.

8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. 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 merupakan salah satu strategi algoritma untuk memecahkan masalah optimasi. Prinsip dari algoritma Greedy adalah “*take what you can get now*”. Dengan algoritma Greedy, permasalahan diselesaikan langkah per langkah sehingga pada tiap langkah, pilihan terbaik diambil tanpa memperhatikan konsekuensi. Pilihan terbaik lokal tersebut diharapkan menjadi pilihan terbaik secara global.

Algoritma Greedy merupakan algoritma yang lebih optimal dibandingkan dengan algoritma Brute Force: Exhaustive Search. Hal tersebut disebabkan karena algoritma Greedy tidak mengevaluasi seluruh kemungkinan, tidak seperti algoritma Exhaustive Search yang mengevaluasi seluruh kemungkinan yang ada. Algoritma Greedy juga melakukan seleksi sehingga didapatkan nilai yang optimal, berbeda dengan algoritma Exhaustive Search. Akan tetapi, algoritma Greedy tidak menjamin mendapatkan hasil yang optimal. Dengan algoritma Exhaustive Search, solusi optimal pasti didapatkan. Namun, dengan algoritma Greedy, kemungkinan solusi yang didapatkan adalah solusi yang mendekati solusi optimal. Jadi, algoritma Greedy bisa mendapatkan solusi mendekati optimal dengan waktu yang lebih efektif dibandingkan dengan algoritma Exhaustive Search yang mendapatkan solusi optimal eksak dengan waktu yang lebih tidak efektif.

Dalam mengimplementasikan algoritma Greedy, terdapat beberapa elemen yang harus diperhatikan, di antaranya adalah sebagai berikut.

1. Himpunan Kandidat (C) adalah himpunan yang terdiri dari kandidat yang akan dipilih pada setiap langkah.
2. Himpunan Solusi (S) adalah himpunan yang terdiri dari kandidat yang sudah dipilih dari himpunan kandidat (C).
3. Fungsi Solusi adalah fungsi untuk menentukan himpunan kandidat yang dipilih sudah menghasilkan solusi.
4. Fungsi Seleksi adalah fungsi untuk memilih kandidat yang sesuai dengan strategi algoritma Greedy yang bersifat heuristik.

5. Fungsi Kelayakan adalah fungsi untuk memeriksa kelayakan kandidat yang dipilih sebagai anggota dari Himpunan Solusi ( $S$ ).
6. Fungsi Obyektif adalah fungsi untuk menentukan solusi yang paling optimal  
Artinya, dari himpunan kandidat  $C$ , dicari sebuah himpunan bagian dari  $C$  sebagai himpunan solusi  $S$ . Himpunan solusi ini harus memenuhi fungsi-fungsi yang memenuhi kriteria untuk mencapai solusi optimal.

Algoritma Greedy dapat menyelesaikan berbagai macam persoalan yang membutuhkan solusi optimal, salah satunya adalah strategi untuk memenangkan permainan yang mengharuskan pemainnya untuk memiliki skor terbesar untuk menang. Karena Galaxio adalah permainan dengan pemain terbesar dan terakhir bertahan sebagai pemenang, maka permainan ini dapat dimenangkan dengan strategi algoritma Greedy. Adapun strategi algoritma Greedy yang diimplementasikan pada permainan Galaxio akan dijabarkan pada Bab III.

## 2.2 Garis Besar Game Engine Permainan Galaxio

*Game engine* merupakan sistem perangkat lunak yang digunakan untuk pengembangan dan penciptaan suatu *game*. *Game engine* sendiri pada dasarnya adalah sebuah *library* yang digunakan untuk membuat *game*. *Game engine* memberikan kemudahan bagi para pengembang *game* karena ia menyediakan fungsi-fungsi inti yang akan sering digunakan oleh *game* tersebut seperti hukum-hukum fisika atau kecerdasan buatan.

Dalam permainan Galaxio, terdapat beberapa komponen yang digunakan agar *game* dapat berjalan yaitu sebagai berikut.

1. *Game Engine*: Folder engine publish berlaku sebagai *game engine* yang mengandung *file-file environment* untuk menjalankan game mengikuti *setting* variabel yang terdapat dalam *file-file* tersebut. Contohnya pada *file* appsettings.json terdapat variabel-variabel yang kita dapat atur seperti jumlah bot yang akan ditandingkan, radius dari peta, kecepatan, nilai-nilai awal objek game, dan aturan-aturan lainnya.
2. Game Logger: Folder logger publish digunakan untuk menuliskan log dari suatu ronde permainan yang telah dijalankan. Log tersebut berisi *state-state* pada *game* setiap tick yang bisa digunakan oleh pemain untuk melihat aksi yang dilakukan dan kondisi yang dimiliki saat ronde permainan tersebut dilakukan. Logs ini akan dijalankan dalam program Galaxio yang terdapat pada folder visualiser.

3. Game Runner: Folder runner publish memiliki tanggung jawab untuk meng-fasilitasi pertandingan antar bot. Game runner bertindak sebagai perantara *game engine* dan bot. *Game engine* menghasilkan informasi *state* yang kemudian dikirimkan ke bot melalui game runner. Bot-bot yang bertanding dalam permainan perlu dikoneksikan ke runner terlebih dahulu sebelum mulai pertandingan agar terdaftar dan terhubung dengan *game engine*.
4. Bot (Player): Pemain pada permainan ini tidak dikontrol secara manual namun menggunakan bot yang memiliki logika berdasarkan algoritma yang telah dirancang dan diimplementasikan. Bot dapat dibuat dengan beberapa bahasa pemrograman seperti .NET, C++, Java, JavaScript, Kotlin, NETCore, dan Python. Pembuatan bot dapat diakses pada folder starter-bots yang sudah diberikan *boilerplate code* untuk masing-masing bahasa yang tersedia.

Adapun starter-pack yang merupakan folder berisi file-file untuk menjalankan permainan pada local dan membuat bot dengan algoritma kita. Starter-pack sendiri berisi beberapa komponen utama yaitu

1. Folder engine-publish : Folder ini berisi file-file *game engine* dan .json yang mengandung *environment variables* untuk *state* game yang dijalankan. Untuk menjalankan *game engine*, digunakan file Engine.dll yang terdapat dalam folder tersebut.
2. Folder logger-publish : Folder ini berisi file-file untuk mencatat log juga file Logger.dll untuk menjalankan logger pada game.
3. Folder runner-publish : Folder ini berisi file-file untuk menjalankan runner pada game seperti GameRunner.dll.
4. Folder reference-bot : Folder ini berisi bot referensi yaitu bot yang sudah dibuat oleh developer game untuk testing dan referensi kita dalam pembuatan bot.
5. Folder starter-bot : Folder ini berisi *boilerplate code* dalam bahasa-bahasa pemrograman yang telah dipilih dan disediakan untuk membuat bot.
6. Folder visualiser : Folder ini berisi *executable software* untuk memvisualisasikan file log yang telah dibuat oleh logger.
7. File run.sh : File ini digunakan untuk menjalankan permainan dengan mudah.

### 2.3 Garis Besar Cara Kerja Bot dan Game Object pada Permainan Galaxio

Adapun garis besar cara kerja bot dan *game object* pada permainan Galaxio dijabarkan sebagai berikut. Pada saat permainan dimulai, bot akan menerima data yang telah disediakan dari Permainan Galaxio. Data-data yang dapat diambil oleh bot pada permulaan *game* adalah sebagai berikut :

1. *World* : menjelaskan keadaan *map* seperti radius lingkaran dan *center point* dari *map*.  
Berikut setiap objek pada *World* :
  - a. *centerPoint*: titik tengah dari *map* ( $x = y = 0$ )
  - b. *radius*: radius *map* pada permainan (terus mengecil)
  - c. *currentTick*: Nomor ronde pada saat ini
2. *gameObjects* : *array of objects* yang berisi data-data objek yang ada pada permainan
  - a. *id*: id unik tiap objek
  - b. *size*: ukuran dari objek
  - c. *speed*: kecepatan dari objek (hanya ada pada objek-objek yang dapat bergerak)
  - d. *currentHeading*: arah dari objek(hanya ada pada objek-objek yang dapat bergerak)
  - e. *position*: posisi dari objek
  - f. *gameObjectType*: tipe dari objek
3. *playerGameObject* : *array of objects* yang berisi data-data bot pemain pada permainan
  - a. *id*: id unik bot pemain
  - b. *size*: ukuran bot pemain
  - c. *speed*: kecepatan bot pemain
  - d. *currentHeading*: arah bot pemain
  - e. *position*: posisi bot pemain
  - f. *effect*: menandakan bot pemain sedang terkena efek apa
  - g. *torpedoSalvoCount*: jumlah torpedo bot pemain
  - h. *supernovaAvailable*: jumlah *supernova* bot pemain
  - i. *teleporterCount*: jumlah *teleporter* bot pemain
  - j. *shieldCount*: menandakan *shield* bot pemain ada atau tidak

Selain itu, seluruh objek pada *game* didefinisikan pada file *ObjectTypes.java* dengan kelas *ObjectTypes*. Adapun kelas *ObjectTypes* dapat berupa sebagai berikut:

1. Player: Objek kapal pemain
2. Food: Objek makanan yang muncul pada peta dan dapat dikonsumsi oleh kapal pemain
3. Wormhole: Objek wormhole yang muncul pada peta dan dapat memindahkan kapal pemain ke wormhole pasangannya
4. Gas Cloud: Objek gas cloud yang muncul pada peta dan dapat memberikan *damage* pada kapal pemain
5. Asteroid Field: Objek kawasan asteroid yang muncul pada peta dan dapat memberikan *damage* pada kapal pemain
6. Torpedo Salvo: Objek peluru torpedo yang ditembakkan oleh kapal pemain
7. Superfood: Objek makanan yang dapat memberikan *power-up* pada kapal pemain yang mengonsumsinya
8. Supernova Pickup: Objek supernova yang muncul pada peta dan dapat diambil oleh kapal pemain
9. Supernova Bomb: Objek supernova yang ditembakkan oleh kapal pemain
10. Teleporter: Objek teleporter yang ditembakkan oleh kapal pemain untuk melakukan teleportasi ke lokasi lain pada peta
11. Shield: Objek shield yang dapat diaktifkan oleh kapal pemain untuk melindungi kapal pemain dari serangan torpedo musuh

Pemain dapat melakukan berbagai aksi yang didefinisikan pada file PlayerActions.java dengan kelas PlayerActions. Adapun kelas PlayerActions berisi sebagai berikut:

1. Forward: Kapal pemain maju
2. Stop: Kapal pemain berhenti
3. Start Afterburner: Kapal pemain memulai afterburner
4. Stop Afterburner: Kapal pemain menghentikan afterburner
5. Fire Torpedoes: Kapal pemain menembakkan torpedo
6. Fire Supernova: Kapal pemain menembakkan supernova
7. Detonate Supernova: Kapal pemain meledakkan supernova yang telah ditembak
8. Fire Teleport: Kapal pemain menembakkan teleporter
9. Teleport: Kapal pemain melakukan teleportasi ke lokasi teleporter
10. Activate Shield: Kapal pemain mengaktifkan *shield*

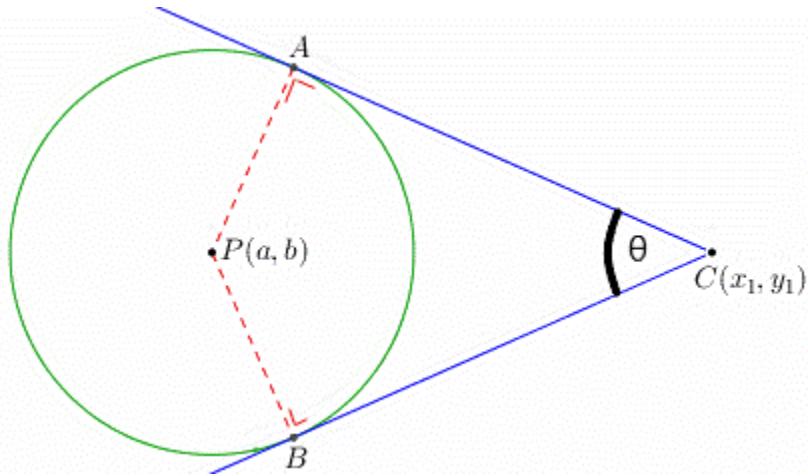
## 2.4 Garis Besar Implementasi Algoritma Greedy ke dalam Bot Permainan Galaxio

Dalam merancang *game engine*, strategi algoritma Greedy diimplementasikan. Sesuai dengan definisi tentang algoritma Greedy pada subbab 2.1, algoritma Greedy merupakan suatu algoritma untuk mendapatkan solusi optimum.

Pada bot permainan Galaxio, strategi Greedy yang digunakan adalah untuk mencari solusi optimum, yaitu cara untuk mempertahankan bot kapal hingga akhir permainan. Untuk dapat bertahan, bot kapal harus memiliki ukuran yang lebih besar daripada seluruh bot kapal lainnya. Akan tetapi, strategi Greedy yang diimplementasikan pada bot permainan hanya dapat menghasilkan solusi optimum lokal. Hal ini terjadi karena bot melakukan proses *scan area* terdekat dari bot. Dengan kata lain, bot tidak memeriksa seluruh kemungkinan aksi yang dapat dilakukan oleh bot secara *brute force*. Alhasil, solusi yang didapatkan merupakan solusi optimum lokal.

## 2.5 Metode Kalkulasi Sudut Antar Garis Singgung Lingkaran dari Sebuah Titik

Dalam algoritma bot, akan digunakan sebuah metode kalkulasi untuk menghitung suatu sudut antar garis singgung lingkaran dari suatu titik seperti gambar di bawah berikut.



**Gambar 2.5.1** Sudut Antara Garis Singgung Lingkaran

Cara menghitung sudut antar garis singgung lingkaran dari suatu titik adalah sebagai berikut.

1. Hitunglah jarak dari titik C ke titik P dengan rumus manhattan.

$$D = \sqrt{(x_1 - a)^2 + (y_1 - b)^2}$$

2. Karena sudah diketahui radius dari lingkaran, gunakan rumus trigonometri untuk mendapat setengah dari sudut  $\theta$ .

$$\frac{1}{2}\theta = \arcsin(R/D)$$

dengan R adalah radius lingkaran dan D adalah jarak titik C ke titik pusat lingkaran.

3. Telah kita dapatkan  $\theta$  dengan mengalikan hitungan pada langkah ke-2 yaitu  $\frac{1}{2}\theta$  dengan dua.

Rumus ini akan sering digunakan dalam algoritma untuk memperkirakan sudut dari titik pusat bot ke lawan.

## **BAB III**

### **APLIKASI STRATEGI GREEDY**

#### **3.1 Ikhtisar Strategi Permainan**

Untuk memenangkan permainan Galaxio, kapal pemain harus menjadi satu-satunya kapal yang bertahan hingga akhir permainan. Dengan kata lain, kapal pemain harus melakukan berbagai upaya untuk memperbesar ukurannya sambil menghindari *obstacle* dan musuh yang lebih besar. Algoritma Greedy diimplementasikan dengan tujuan untuk membuat kapal pemain berukuran paling besar di antara semua kapal pemain lainnya.

Strategi algoritma Greedy yang diimplementasikan pada permainan dapat dikategorikan menjadi 9 kategori berdasarkan elemen permainan yang diprioritaskan, yaitu berdasarkan scan area, makanan, musuh, *obstacle*, supernova, *teleport*, torpedo, afterburner, dan shield. Hal tersebut dilakukan agar bot kapal dapat menentukan aksi apa yang harus dilakukan sehingga membawa keuntungan terbesar bagi dirinya.

#### **3.2. Pemetaan Elemen / Komponen Algoritma Greedy pada Bot Permainan Galaxio**

##### **3.2.1 Greedy by Scanning Area**

Greedy by Scanning Area adalah pendekatan algoritma Greedy yang memprioritaskan agar bot selalu pergi ke daerah yang paling menguntungkan bagi bot tersebut. Untuk mencari daerah terdekat yang paling menguntungkan, digunakan sistem *scoring* yang dipengaruhi oleh jarak bot kepada makanan, *obstacle*, musuh yang lebih besar atau lebih kecil, serta *pick-up items*. Penyeleksian daerah dilakukan dengan membagi daerah menjadi 12 bagian dengan batas jarak pengecekan tiap daerah tidak lebih besar dari 250 dari bot.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.1 Komponen Algoritma Greedy by Scanning Area**

| <b>Nama Komponen</b> | <b>Definisi Komponen</b>                                                                          |
|----------------------|---------------------------------------------------------------------------------------------------|
| Himpunan kandidat    | Semua objek yang berada dalam jangkauan jarak.                                                    |
| Himpunan solusi      | Daerah yang dibagi menjadi 12 bagian dengan batas jarak pengecekan tidak lebih dari 250 dari bot. |
| Fungsi solusi        | Melakukan pengecekan untuk menemukan daerah terbaik untuk bot.                                    |

|                  |                                                                                          |
|------------------|------------------------------------------------------------------------------------------|
| Fungsi seleksi   | Memilih daerah terbaik berdasarkan seluruh objek yang ada di daerah tersebut.            |
| Fungsi kelayakan | Melakukan pengecekan apakah objek tersebut memiliki jarak tidak lebih dari 250 dari bot. |
| Fungsi objektif  | Memaksimumkan keuntungan bot dengan mencari daerah terbaik.                              |

### 3.2.2 Greedy by Makanan

Greedy by Makanan adalah pendekatan algoritma Greedy yang memprioritaskan agar bot mengonsumsi objek-objek makanan terdekat yang berada pada jangkauan area yang paling menguntungkan bagi bot. Adapun skor untuk objek makanan adalah sebagai berikut:

$$score_{food} = \frac{3}{d_{bf}} + \frac{3}{d_{fc}}$$

dengan  $d_{bf}$  adalah jarak antara bot dengan makanan dihitung dari ujung terluar bot, dan  $d_{fc}$  adalah jarak antara makanan dengan titik pusat peta. Sistem *scoring* ini dirancang sehingga makanan yang lebih dekat dengan pemain dan lebih dekat ke pusat peta memiliki skor yang lebih tinggi dan diprioritaskan dibandingkan makanan lainnya.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.2** Komponen Algoritma Greedy by Makanan

| Nama Komponen     | Definisi Komponen                                                                   |
|-------------------|-------------------------------------------------------------------------------------|
| Himpunan kandidat | Semua makanan yang berada dalam jangkauan jarak.                                    |
| Himpunan solusi   | Makanan dengan skor tertinggi pada area terpilih.                                   |
| Fungsi solusi     | Melakukan pengecekan untuk menemukan makanan terdekat dengan bot dan pusat peta.    |
| Fungsi seleksi    | Memilih makanan terdekat dengan bot dan pusat peta pada area terpilih.              |
| Fungsi kelayakan  | Melakukan pengecekan apakah objek tersebut memiliki tipe objek food atau superfood. |
| Fungsi objektif   | Memaksimumkan keuntungan bot dengan mencari makanan terdekat.                       |

### 3.2.3 Greedy by Musuh

Greedy by Musuh adalah pendekatan algoritma Greedy yang memprioritaskan agar bot mengonsumsi bot lain yang ukurannya lebih kecil. Adapun skor untuk bot lain yang ukurannya lebih kecil adalah sebagai berikut :

$$score_{enemy} = \frac{S_e}{d_{be}},$$

dengan  $d_{be}$  adalah jarak antara bot dengan bot musuh yang dihitung dari ujung terluar bot, dan  $S_e$  adalah ukuran dari bot musuh. Sistem *scoring* ini dirancang agar bot memprioritaskan mengejar musuh yang lebih kecil daripada mengejar makanan yang terdekat.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.3** Komponen Algoritma Greedy by Musuh

| Nama Komponen     | Definisi Komponen                                                                           |
|-------------------|---------------------------------------------------------------------------------------------|
| Himpunan kandidat | Semua bot yang ukurannya lebih kecil dan berada dalam jangkauan.                            |
| Himpunan solusi   | Bot dengan skor tertinggi pada area terpilih.                                               |
| Fungsi solusi     | Melakukan pengecekan untuk menemukan bot yang lebih kecil dari bot dan terdekat dengan bot. |
| Fungsi seleksi    | Memilih bot musuh terkecil yang terdekat dengan bot.                                        |
| Fungsi kelayakan  | Melakukan pengecekan apakah objek tersebut memiliki tipe objek player.                      |
| Fungsi objektif   | Memaksimumkan keuntungan bot dengan mencari musuh yang dapat dimakan.                       |

### 3.2.4 Greedy by Obstacle

Greedy by Obstacle adalah pendekatan algoritma Greedy yang memprioritaskan agar bot terkena *damage* akibat terbentur *obstacles* seminimal mungkin. Adapun jenis *obstacle* yang ada pada Galaxio adalah gas cloud dan asteroid field.

Apabila pada area yang di-*scan* terdapat gas cloud, maka bot akan mengecek apakah ada musuh pada area tersebut. Jika tidak terdapat musuh pada area yang sedang di-*scan*, maka bot akan *skip* area tersebut dan area tersebut tidak akan dipilih. Jika terdapat musuh pada area yang sedang di-*scan*, maka akan diterapkan sistem *scoring* sebagai berikut:

$$score_{gas\ cloud} = - \frac{s_g}{d_{bg}},$$

dengan  $s_g$  ukuran gas cloud dan  $d_{bg}$  jarak antara bot dengan gas cloud.

Untuk asteroid field, sistem *scoring* yang diterapkan adalah sebagai berikut:

$$score_{asteroid} = - \frac{s_a}{d_{ba}},$$

dengan  $s_a$  ukuran asteroid field dan  $d_{ba}$  jarak antara bot dengan asteroid field.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.4** Komponen Algoritma Greedy by Obstacle

| Nama Komponen     | Definisi Komponen                                                                             |
|-------------------|-----------------------------------------------------------------------------------------------|
| Himpunan kandidat | Semua obstacle (gas cloud dan asteroid field) yang berada dalam jangkauan.                    |
| Himpunan solusi   | Obstacle yang valid.                                                                          |
| Fungsi solusi     | Melakukan pengecekan untuk menemukan obstacle pada jangkauan.                                 |
| Fungsi seleksi    | Memilih obstacle yang valid.                                                                  |
| Fungsi kelayakan  | Melakukan pengecekan apakah objek tersebut memiliki tipe objek asteroid field atau gas cloud. |
| Fungsi objektif   | Meminimalkan kemungkinan untuk terkena <i>damage</i> akibat obstacle.                         |

### 3.2.5 Greedy by Supernova

Greedy by Supernova adalah pendekatan algoritma Greedy yang memprioritaskan agar bot mengambil supernova apabila supernova berada pada jangkauan daerah pengecekan bot. Adapun skor untuk supernova adalah sebagai berikut :

$$score_{supernova} = \frac{10}{d_{bs}},$$

dengan  $d_{bs}$  adalah jarak antara bot dengan supernova. Sistem *scoring* ini dirancang agar bot memprioritaskan supernova daripada hal-hal lain apabila supernova berada pada area yang sedang di-*scan* oleh bot.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.5** Komponen Algoritma Greedy by Supernova

| <b>Nama Komponen</b> | <b>Definisi Komponen</b>                                                                          |
|----------------------|---------------------------------------------------------------------------------------------------|
| Himpunan kandidat    | Supernova yang berada dalam jangkauan.                                                            |
| Himpunan solusi      | Supernova yang ada di map.                                                                        |
| Fungsi solusi        | Melakukan pengecekan untuk menemukan supernova pada jangkauan.                                    |
| Fungsi seleksi       | Memilih supernova yang ada pada jangkauan.                                                        |
| Fungsi kelayakan     | Melakukan pengecekan apakah objek tersebut memiliki tipe objek supernova.                         |
| Fungsi objektif      | Memaksimumkan keuntungan bot yaitu mengecilkan musuh dengan meledakkan supernova di daerah musuh. |

### 3.2.6 Greedy by Teleport

Greedy by Teleport adalah pendekatan algoritma Greedy yang memprioritaskan agar bot menembakkan teleporter ke arah yang paling menguntungkan bagi bot. Teleporter ditembakkan ke arah daerah yang memiliki musuh dengan ukuran yang lebih kecil daripada bot pada jarak yang relatif dekat. Teleporter juga ditembakkan jika dan hanya jika bot tidak sedang menembakkan supernova. Bot akan melakukan teleportasi ketika teleporter sudah dekat dengan musuh yang ingin dimakan.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.6** Komponen Algoritma Greedy by Teleport

| <b>Nama Komponen</b> | <b>Definisi Komponen</b>                                                   |
|----------------------|----------------------------------------------------------------------------|
| Himpunan kandidat    | Semua teleporter yang ada pada peta.                                       |
| Himpunan solusi      | Teleporter yang dimiliki oleh bot.                                         |
| Fungsi solusi        | Melakukan pengecekan apakah ada musuh yang dekat dengan teleporter bot.    |
| Fungsi seleksi       | Memilih teleporter yang dekat dengan musuh.                                |
| Fungsi kelayakan     | Melakukan pengecekan apakah objek tersebut memiliki tipe objek teleporter. |
| Fungsi objektif      | Memaksimumkan keuntungan bot dengan mencari lawan yang dapat               |

|  |          |
|--|----------|
|  | dimakan. |
|--|----------|

### 3.2.7 Greedy by Torpedo

Greedy by Torpedo adalah pendekatan algoritma Greedy yang memprioritaskan agar bot menembakkan torpedo ke arah musuh.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.7 Komponen Algoritma Greedy by Torpedo**

| Nama Komponen     | Definisi Komponen                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Himpunan kandidat | Semua musuh yang ada pada peta                                                                                                                                    |
| Himpunan solusi   | Musuh yang dapat ditembak oleh bot                                                                                                                                |
| Fungsi solusi     | Melakukan pengecekan apakah ada musuh yang dapat ditembak dengan arti bahwa tidak ada objek yang menghalangi musuh sehingga torpedo tidak hancur oleh objek lain. |
| Fungsi seleksi    | Memilih musuh yang tidak terhalang oleh objek lain                                                                                                                |
| Fungsi kelayakan  | Melakukan pengecekan apakah objek tersebut memiliki tipe objek player.                                                                                            |
| Fungsi objektif   | Memaksimumkan keuntungan bot dengan mengurangi ukuran lawan dan meningkatkan ukuran bot.                                                                          |

### 3.2.8 Greedy by Afterburner

Greedy by Afterburner adalah pendekatan algoritma Greedy yang memprioritaskan keselamatan bot. Algoritma Greedy ini akan digunakan apabila ada bot musuh yang ukurannya lebih besar dari bot, maka bot akan mengaktifkan afterburner agar kecepatan bot menjadi dua kali lipat sehingga bot dapat lebih cepat kabur dari bot musuh dan bot tetap akan menerapkan Greedy by Makanan sambil mengaktifkan afterburner. Dengan begitu, meskipun afterburner akan memakan *size* bot, bot tetap tidak akan mengalami pengurangan *size* yang signifikan dikarenakan bot juga tetap menerapkan Greedy by Makanan. Afterburner akan aktif apabila ada bot musuh yang sedang mengarah ke bot dan jarak bot musuh antara bot lebih kecil dari 200. Sedangkan afterburner akan dinonaktifkan apabila jarak bot musuh antara bot lebih dari 200.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.8** Komponen Algoritma Greedy by Afterburner

| <b>Nama Komponen</b> | <b>Definisi Komponen</b>                                                                           |
|----------------------|----------------------------------------------------------------------------------------------------|
| Himpunan kandidat    | Semua musuh yang ada pada peta.                                                                    |
| Himpunan solusi      | Musuh yang memiliki potensi untuk memakan bot.                                                     |
| Fungsi solusi        | Melakukan pengecekan apakah ada musuh memiliki potensi untuk memakan bot pada jangkauan jarak 250. |
| Fungsi seleksi       | Memilih musuh yang ada di jangkauan jarak 250 dan memiliki potensi untuk memakan bot.              |
| Fungsi kelayakan     | Melakukan pengecekan apakah objek tersebut memiliki tipe objek player.                             |
| Fungsi objektif      | Meminimalkan kemungkinan untuk dimakan oleh player lain.                                           |

### 3.2.9 Greedy by Shield

Greedy by Shield adalah pendekatan algoritma Greedy yang memprioritaskan keselamatan bot dengan mengaktifkan shield. Shield diaktifkan ketika ada torpedo musuh yang ditembakkan menuju bot.

Adapun komponen dari algoritma Greedy pada bagian ini adalah sebagai berikut.

**Tabel 3.2.9** Komponen Algoritma Greedy by Shield

| <b>Nama Komponen</b> | <b>Definisi Komponen</b>                                                                                          |
|----------------------|-------------------------------------------------------------------------------------------------------------------|
| Himpunan kandidat    | Semua torpedo yang ada pada peta.                                                                                 |
| Himpunan solusi      | Torpedo yang sedang mengarah ke bot dan dalam jangkauan jarak 200 dari bot.                                       |
| Fungsi solusi        | Melakukan pengecekan apakah ada torpedo yang sedang mengarah ke bot dan sudah dalam jangkauan jarak 200 dari bot. |
| Fungsi seleksi       | Memilih torpedo yang sedang mengarah ke bot dan dalam jangkauan jarak 200 dari bot.                               |
| Fungsi kelayakan     | Melakukan pengecekan apakah objek tersebut memiliki tipe objek salvo.                                             |
| Fungsi objektif      | Meminimalkan kemungkinan untuk terkena <i>damage</i> akibat torpedo salvo.                                        |

### 3.3 Eksplorasi Alternatif Solusi Algoritma Greedy pada Bot Permainan Galaxio

Untuk memenangkan permainan Galaxio, terdapat berbagai macam strategi yang dapat digunakan, salah satunya adalah dengan strategi algoritma Greedy. Akan tetapi, terdapat pula solusi alternatif untuk menyelesaikan permainan, yaitu dengan beberapa strategi berikut.

#### 3.3.1 Greedy by Wormhole

Pada strategi Greedy by Wormhole, bot kapal melakukan pengecekan terhadap setiap objek wormhole yang dapat dikunjungi oleh bot kapal beserta dengan pasangannya. Dengan demikian, bot kapal dapat mempertimbangkan untuk berpindah ke pasangan wormhole dengan area yang lebih menguntungkan daripada area bot kapal saat itu.

#### 3.3.2 Greedy by Tick

Pada strategi Greedy By Tick ini rumus *scoring* tidak hanya menggunakan jarak antar bot dan ke objek saja untuk menentukan *score*-nya, namun juga mementingkan berapa tick yang diperlukan bot untuk mencapai objek tersebut. Berikut penjabaran rumus-rumus *scoring* yang dapat digunakan untuk Greedy by Tick:

$$roundToObject = \frac{d_{bo}}{s_b},$$

$d_{bo}$  : distance antara bot dengan objek,  $s_b$  : kecepatan bot

Sistem *scoring* untuk makanan menggunakan Greedy by Tick:

$$score_{food} = \frac{3}{roundToObject},$$

Sistem *scoring* untuk musuh yang lebih kecil menggunakan Greedy by Tick:

$$score_{enemy} = \frac{s_e}{roundToObject},$$

Sistem *scoring* untuk obstacle menggunakan Greedy by Tick:

$$score_{asteroid} = -\frac{s_a}{roundToObject},$$

dengan strategi Greedy by Tick, bot akan mengambil keputusan terbaik tiap tick-nya bergantung pada tingkat ketepatan sistem *scoring* kita. Semakin bagus sistem *scoring* maka akan semakin bagus juga keputusan yang diambil bot tiap tick-nya.

#### 3.3.3 Greedy by Supernova with the Most Enemies

Pada strategi Greedy by Supernova with the Most Enemies, bot kapal memprioritaskan untuk menembakkan supernova ke arah daerah yang terdapat banyak musuh. Supernova

diledakkan ketika supernova bomb telah mencapai jarak terdekat dengan para musuh. Dengan begitu, supernova yang diledakkan akan lebih efektif dan memberikan *damage* ke lebih banyak musuh. Akan tetapi, sebelum meledakkan supernova, jarak antara bot kapal dengan supernova bomb yang ditembakkan perlu dipertimbangkan agar supernova tidak meledak pada lokasi yang dekat dengan bot kapal. Apabila supernova meledak dekat bot kapal, bot kapal juga berpotensi terkena *damage* akibat supernova gas cloud.

### **3.3.4 Greedy by Teleport with the Most Smaller Enemies**

Pada strategi Greedy by Teleport with the Most Smaller Enemies, bot kapal memprioritaskan untuk menembakkan teleporter ke arah daerah yang terdapat banyak musuh yang dapat dimakan. Bot kapal akan teleportasi ketika teleporter telah mencapai jarak terdekat dengan para musuh yang akan dimakan. Dengan begitu, bot kapal akan lebih mudah untuk memakan banyak musuh sekaligus. Akibatnya, bot kapal dapat mengalami pertambahan ukuran yang melesat.

### **3.3.5 Greedy by Edge**

Pada strategi Greedy by Edge, bot kapal memprioritaskan untuk membesarkan diri di ujung-ujung map dengan tujuan untuk menghindarkan diri se bisa mungkin dari musuh yang dapat mengejar dan memakannya. Perlu diperhatikan bahwa banyak sekali objek-objek yang hancur akibat *circular edge* sedangkan bot dipasang di dekat tengah-tengah map pada permulaan. Akibatnya, banyak objek-objek yang dapat memperbesar bot namun tidak digunakan karena lokasinya yang jauh dari starting point kapal-kapal. Strategi ini memungkinkan untuk bot mengambil kesempatan menggunakan objek-objek yang dapat memperbesarkan bot kapal pada permulaan permainan sehingga kapal bergerak secara Greedy dan dengan risiko termakan yang lebih rendah.

## **3.4 Analisis Efisiensi dari Kumpulan Solusi Algoritma Greedy**

Untuk menentukan efektivitas dan efisiensi dari suatu algoritma, dibutuhkan suatu tolak ukur. Dengan tolak ukur tersebut, algoritma yang berbeda dapat dibandingkan. Salah satu metode untuk mengukur efisiensi algoritma adalah dengan notasi Big-O. Pada notasi Big-O, hal yang menjadi pertimbangan adalah bagaimana hubungan antara pertumbuhan waktu dan memori

dengan pertumbuhan ukuran data. Berikut adalah analisis efisiensi dari algoritma Greedy yang diimplementasikan pada bot.

#### **3.4.1 Analisis Efisiensi dari Greedy by Scanning Area**

Pada strategi Greedy by Scanning Area, dilakukan pencarian objek-objek game dalam jangkauan area yang terdefinisi yaitu 360 derajat sejauh 250. Kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.2 Analisis Efisiensi dari Greedy by Makanan**

Pada strategi Greedy by Makanan, bot perlu mendapatkan *list* dari seluruh objek makanan yang berada dalam jangkauan area terpilih. Berdasarkan *list* makanan tersebut, bot dapat menentukan makanan yang paling menguntungkan. Bot akan mengiterasi seluruh makanan pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.3 Analisis Efisiensi dari Greedy by Musuh**

Pada strategi Greedy by Musuh, bot perlu mendapatkan *list* dari seluruh bot lain yang lebih kecil dan berada dalam jangkauan *scan area*. Berdasarkan *list player* tersebut, bot akan menentukan area dengan musuh mana yang paling menguntungkan dan pergi ke daerah tersebut. Kompleksitas algoritma yang dibutuhkan dari strategi ini adalah  $O(n)$  karena bot akan mengiterasi seluruh bot lain yang ada di *list*.

#### **3.4.4 Analisis Efisiensi dari Greedy by Obstacle**

Pada strategi Greedy by Makanan, bot perlu mendapatkan *list* dari seluruh objek *obstacle* yang berada dalam jangkauan area terpilih. Berdasarkan *list obstacle* tersebut, bot dapat menentukan ke arah mana bot harus pergi. Bot akan mengiterasi seluruh *obstacle* pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.5 Analisis Efisiensi dari Greedy by Supernova**

Pada strategi Greedy by Supernova, bot hanya perlu mendapatkan elemen pertama dari *list supernova pick up* yang berada pada peta. Selain itu, bot hanya perlu melakukan satu kali

pengecekan untuk menembakkan supernova. Dengan demikian, kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(1)$ .

#### **3.4.6 Analisis Efisiensi dari Greedy by Teleport**

Pada strategi Greedy by Teleport, bot perlu mendapatkan *list* dari seluruh pemain (musuhList) yang berada dalam peta. Berdasarkan *list* musuh tersebut, bot dapat menentukan kapan bot harus melakukan teleportasi. Bot akan mengiterasi seluruh musuh pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.7 Analisis Efisiensi dari Greedy by Torpedo**

Pada strategi Greedy by Torpedo, bot perlu mendapatkan *list* dari seluruh pemain (musuhList) yang berada dalam peta. Berdasarkan *list* musuh tersebut, bot dapat menentukan kapan bot harus melakukan penembakan torpedo. Bot akan mengiterasi seluruh musuh pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.8 Analisis Efisiensi dari Greedy by Afterburner**

Pada strategi Greedy by Afterburner, bot perlu mendapatkan *list* dari seluruh pemain (playerList) yang berada dalam area. Berdasarkan *list* musuh tersebut, bot dapat menentukan kapan bot harus mengaktifkan afterburner. Bot akan mengiterasi seluruh musuh pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

#### **3.4.9 Analisis Efisiensi dari Greedy by Shield**

Pada strategi Greedy by Shield, bot perlu mendapatkan *list* dari seluruh torpedo (torpedoList) yang berada dalam peta. Berdasarkan *list* torpedo tersebut, bot dapat menentukan kapan bot harus mengaktifkan shield. Bot akan mengiterasi seluruh torpedo pada *list*, sehingga kompleksitas waktu yang dibutuhkan dari strategi ini adalah  $O(n)$ .

### **3.5 Strategi Greedy yang Digunakan pada Program Bot**

Strategi heuristik yang digunakan sebagai algoritma Greedy utama dari program bot adalah penggabungan dari seluruh strategi Greedy by yang sudah dijelaskan pada subbab 3.2. Strategi-strategi Greedy-by tersebut akan kita gabungkan, namun tetap perlu dipertimbangkan

juga strategi apa yang harus diprioritaskan terlebih dahulu sebab kita juga tidak bisa memakai seluruh strategi sekaligus. Cara untuk menentukan prioritas strateginya adalah dengan membandingkan efisiensi dan keuntungan yang bisa didapatkan oleh setiap strategi. Untuk urutan prioritas strategi yang disusun adalah sebagai berikut.

1. Strategi Greedy By Supernova
2. Strategi Greedy By Teleport
3. Strategi Greedy By Shield
4. Strategi Greedy By Afterburner
5. Strategi Greedy By Torpedo
6. Strategi Greedy By Chase Enemy
7. Strategi Greedy By Food

Alasan kelompok kami memilih urutan prioritas di atas adalah berdasarkan efek yang diberikan untuk keseluruhan *player* pada permainan, bukan hanya bot kapal sendiri saja. Supernova menjadi strategi yang diprioritaskan pertama karena supernova menghasilkan gas cloud yang berukuran besar dan bertahan dalam peta selama *edge* belum mencapai supernova. Hal ini membuat supernova menjadi salah satu *action* yang sangat mematikan karena dapat mematikan banyak *player* sekaligus sehingga memiliki *score* yang besar dalam perhitungan *scan area*. Kemudian, *teleport* dijadikan prioritas kedua karena *teleport* merupakan aksi yang paling menguntungkan untuk mematikan lawan dan membesarakan bot kapal sendiri. *Teleport* dapat dilakukan dari lokasi yang sangat jauh dari musuh sehingga hanya memerlukan beberapa *tick* saja untuk memakan lawan. Lalu, *shield* dijadikan prioritas ketiga karena bot memerlukan pertahanan dari lawan sehingga tidak membesarakan *size* musuh dan tidak terlalu mengecilkan *size* kita. Lalu, *afterburner* dijadikan prioritas keempat karena bot memerlukan strategi khusus untuk menghindari dari musuh yang lebih besar sehingga tidak termakan. Oleh karena itu, bot akan mengaktifkan *afterburner* untuk mempercepat *movement* kita. Selama *afterburner* aktif, bot tetap menjalankan strategi Greedy seperti biasanya. Bot akan mematikan *afterburner* jika sudah tidak ada musuh yang lebih besar di dekatnya. Prioritas selanjutnya adalah menembakkan torpedo kepada musuh. Hal ini dilakukan apabila tidak ada objek lain yang menghalangi jalur

torpedo yang akan ditembaknya. Kemudian, prioritas selanjutnya adalah mengejar bot yang lebih kecil darinya apabila tidak bisa ditembak. Terakhir, bot akan mengejar makanan terdekat.

Terdapat suatu kasus khusus yaitu ketika hasil *scanning area* bot adalah area yang kosong (semua *list* yang dihasilkan *empty*). Apabila hal ini terjadi, maka bot akan mengusahakan untuk melakukan beberapa hal, yaitu menembak torpedo ke musuh, mencari makanan terdekat di luar area, dan menuju ke *world center*. Tujuan dari bot menuju ke *world center* adalah untuk menghindari *edge* yang terus mengecil. Biasanya hal ini bisa terjadi ketika sudah tidak ada makanan lagi dalam peta dan tersisa musuh saja. Maka, bot kita akan memprioritaskan untuk menjauh dari *edge*, menembak, dan mengejar musuh. Kondisi ini biasa disebut dengan *endgame*.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Algoritma Greedy pada Bot Permainan Galaxio

##### 4.1.1 Object Type (ObjectTypes.java)

```
Enum ObjectTypes
    PLAYER(1),
    FOOD(2),
    WORMHOLE(3),
    GAS_CLOUD(4),
    ASTEROID_FIELD(5),
    TORPEDO_SALVO(6),
    SUPERFOOD(7),
    SUPERNOVA_PICKUP(8),
    SUPERNOVA_BOMB(9),
    TELEPORTER(10),
    SHIELD(11);

    value = Integer

    Function ObjectTypes(Integer value)
        value ← value

    Function ObjectTypes valueOf(Integer value)
        for (ObjectTypes objectType : ObjectTypes.values())
            if (objectType.value = value) then
                → objectType

        → IllegalArgumentException("Value not found")
```

ObjectTypes.java berisi definisi dari *enum* ObjectTypes yang berisi tipe-tipe objek, yaitu player, food, wormhole, gas cloud, asteroid field, torpedo salvo, superfood, supernova pickup, supernova bomb, teleporter, dan shield. ObjectTypes juga memiliki atribut *value* bertipe Integer yang unik untuk setiap tipe objek.

#### 4.1.2 Player Action (PlayerActions.java)

```
Enum PlayerActions
    FORWARD(1),
    STOP(2),
    STARTAFTERBURNER(3),
    STOPAFTERBURNER(4),
    FIRETORPEDOES(5),
    FIRESUPERNOVA(6),
    DETONATESUPERNOVA(7),
    FIRETELEPORT(8),
    TELEPORT(9),
    ACTIVATESHIELD(10)

    value = Integer

    Function PlayerActions(Integer value)
        value ← value
```

PlayerActions.java berisi definisi dari *enums* PlayerActions yang berisi aksi-aksi pemain. Definisi ini akan sering dipakai dalam implementasi algoritma utama untuk mempermudah juga dalam pembuatan kode.

#### 4.1.3 Main (computeNextAction)

```
Function computeNextPlayerAction(PlayerAction playerAction)
    // Output Current Tick
    if(gameState.getWorld().getCurrentTick() ≠ NULL) then
        output(gameState.getWorld().getCurrentTick())
        output(": ")

    // Initialize action and heading
    playerAction.action ← PlayerActions.FORWARD
    playerAction.heading ← new Random().nextInt(360)

    // While game objects not empty
```

```

if (!gameState.getGameObjects().isEmpty()) then
    // Getting object list
    objectList ← gameState.getGameObjects()

    // Getting food list
    foodList ← gameState.getGameObjects()
        .stream().filter(item -> item.getGameObjectType() =
ObjectTypes.FOOD or item.getGameObjectType() = ObjectTypes.SUPERFOOD)
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList())

    // Getting food not near edge list
    foodNotNearEdgeList ← gameState.getGameObjects()
        .stream()
        .filter(item -> (item.getGameObjectType() = ObjectTypes.FOOD or
item.getGameObjectType() = ObjectTypes.SUPERFOOD) and
            (gameState.getWorld().radius - getDistanceBetween(item,
gameState.getWorld()) ≥ 1.2 * bot.size))
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList())

    // Getting player list
    playerList ← gameState.getPlayerGameObjects()
        .stream()
        .filter(enemy -> enemy.id ≠ bot.id)
        .sorted(Comparator
            .comparing(enemy -> getDistanceBetween(bot, enemy)))
        .collect(Collectors.toList())

    // Getting enemy list to shoot
    musuhList ← gameState.getPlayerGameObjects()

```

```

    .stream()
    .filter(enemy -> enemy.id != bot.id)
    .sorted(Comparator
        .comparing(enemy -> getDistanceBetween(bot, enemy)))
    .collect(Collectors.toList())

    // Getting teleporter list
    teleportList ← gameState.getGameObjects()
        .stream()
        .filter(item -> item.getGameObjectType() =
ObjectTypes.TELEPORTER)
        .collect(Collectors.toList())

    // Getting obstacle list to avoid
    obstacleList ← gameState.getGameObjects()
        .stream()
        .filter(item -> item.getGameObjectType() = ObjectTypes.GAS_CLOUD
or item.getGameObjectType() = ObjectTypes.ASTEROID_FIELD)
        .sorted(Comparator
            .comparing(obstacle -> getDistanceBetween(bot, obstacle)))
        .collect(Collectors.toList())

    // Getting torpedo list to activate shield
    torpedoList ← gameState.getGameObjects()
        .stream()
        .filter(item -> item.getGameObjectType() =
ObjectTypes.TORPEDO_SALVO)
        .sorted(Comparator
            .comparing(torpedo -> getDistanceBetween(bot, torpedo)))
        .collect(Collectors.toList())

    // Getting supernova list to get and shoot supernova
    supernovaList ← gameState.getGameObjects()

```

```

        .stream()
        .filter(item -> item.getGameObjectType() =
ObjectTypes.SUPERNOVA_PICKUP)
            .collect(Collectors.toList())

        // Getting supernova bomb list to detonate supernova
        supernovaBombList ← gameState.getGameObjects()
            .stream()
            .filter(item -> item.getGameObjectType() =
ObjectTypes.SUPERNOVA_BOMB)
                .collect(Collectors.toList())

        // Choosing best area
        chosenArea ← scanArea(foodList, playerList, obstacleList, supernovaList)
        foodList ← chosenArea.get(0)
        playerList ← chosenArea.get(1)
        obstacleList ← chosenArea.get(2)
        supernovaList ← chosenArea.get(3)

        // Checking available teleport
        myTeleporter ← NULL
        if(checkOwnerTeleporter(teleportList)) then
            myTeleporter ← getTeleporter(teleportList)

        // Checking shield active or not
        if(tickShield ≠ -999 and gameState.getWorld().getCurrentTick() - tickShield >= 20) then
            tickShield ← -999
            shieldActive ← False

        // If we can detonate supernova
        if(shootSupernova and checkDetonateSupernova(supernovaBombList,
musuhList)) then

```

```

playerAction.action ← PlayerActions.DETONATESUPERNOWA
output("LEDAKIN GAN")
shootSupernova ← false

else if(bot.supernovaAvailable > 0 and checkFireSupernova(musuhList) and
!shootSupernova) then
    // If we can shoot supernova
    playerAction.action ← PlayerActions.FIRESUPERNOWA
    musuhGedeList ← gameState.getPlayerGameObjects()
        .stream()
        .filter(enemy -> enemy.id ≠ bot.id)
        .sorted(Comparator
            .comparing(enemy -> enemy.size))
        .collect(Collectors.toList())
    for i in [musuhGedeList.size() - 1, 0]
        if(getDistanceBetween(bot, musuhGedeList.get(i)) ≥ 300) then
            playerAction.heading =
getHeadingBetween(musuhGedeList.get(i)) then
            break
        output("TEMBAKKK")
        shootSupernova ← True
    else if(supernovaList.size() > 0) then
        // If we can chase supernova
        playerAction.heading ← getHeadingBetween(supernovaList.get(0))
        output("Kejer Supernova")
        if(getDistanceBetween(bot, supernovaList.get(0)) < 30) then
            output("Deket Supernova Bang")

    else if(myTeleporter ≠ NULL and musuhList.size() > 0 and
doTeleport(myTeleporter, musuhList)) then
        // If we can teleport now
        playerAction.action ← PlayerActions.TELEPORT
        headingTeleporter ← -999
        shootTeleporter ← 0

```

```

        output("Nging")
    else if(torpedoList.size() > 0 and checkShieldCondition(torpedoList) and
!shieldActive) then
        // If we have to activate shield
        playerAction.action ← PlayerActions.ACTIVATESHIELD
        shieldActive ← True
        tickShield ← gameState.getWorld().getCurrentTick()
        output("Aktifin Shield Gan")
    else if (musuhList.size() > 0 and !checkEnemyChase(musuhList) and
afterBurnerActive) then
        // If we can stop afterburner
        playerAction.action ← PlayerActions.STOPAFTERBURNER
        afterBurnerActive ← False
        output("Stop afterburner gan")
    else if (musuhList.size() > 0 and checkEnemyChase(musuhList) and
!afterBurnerActive) then
        // If we can start afterburner
        playerAction.action ← PlayerActions.STARTAFTERBURNER
        afterBurnerActive ← True
        output("Aktifin afterburner gan")
    else if(!shootSupernova and shootTeleporter = 0 and musuhList.size() > 0
and bot.teleporterCount > 0 and headingTeleporter = -999 and bot.size > 60 and
bot.size - 40 ≥ musuhList.get(0).size and musuhList.get(0).size ≥ 30) then
        // If we can shoot teleporter to enemy
        head ← getHeadingBetween(musuhList.get(0))
        playerAction.heading ← head
        headingTeleporter ← head
        playerAction.action ← PlayerActions.FIRETELEPORT
        shootTeleporter ← 2
        output("Tembak Teleport Gan")
    else if(foodList.size() = 0 and playerList.size() = 0) then
        // If there is no food and enemy within area
        if(!shootTorpedo and musuhList.size() > 0 and

```

```

getDistanceBetween(musuhList.get(0), bot) - musuhList.get(0).size - bot.size <
bot.size + 150) then
    playerAction.heading ← getHeadingBetween(musuhList.get(0))
    // Checking if we can shoot enemy or not
    nabrak ← False
    for x in [0, objectList.size() - 1]
        if(Math.abs(playerAction.heading -
getHeadingBetween(objectList.get(x))) ≤ 10 and getDistanceBetween(bot,
objectList.get(x)) ≤ getDistanceBetween(bot, musuhList.get(0))) then
            nabrak ← True
            output("Duh nabrak gabisa nembak gan, ")
            break
        if(!nabrak and musuhList.get(0).size ≥ 10)
            output("Nembak gan")
            playerAction.action ← PlayerActions.FIRETORPEDOES
            shootTorpedo ← True
        else if(foodNotNearEdgeList.size() > 0) then
            // Chase food not near edge
            playerAction.heading ←
getHeadingBetween(foodNotNearEdgeList.get(0))
            output("Cari makan yang gak di ujung")
            shootTorpedo ← False
        else
            // Heading center
            output("Ke tengah bang")
            playerAction.heading ←
getHeadingBetween(gameState.getWorld())
            shootTorpedo ← False
    else
        if(foodNotNearEdgeList.size() > 0) then
            playerAction.heading ←
getHeadingBetween(foodNotNearEdgeList.get(0))
            output("Cari makan yang gak di ujung")

```

```

        shootTorpedo ← False
    else
        output("Ke tengah bang")
        playerAction.heading ←
getHeadingBetween(gameState.getWorld())
        shootTorpedo ← False
    else if(playerList.size() > 0 and playerList.get(0).size < bot.size) then
        // If there is a smaller enemy nearby
        playerAction.heading ← getHeadingBetween(playerList.get(0))
        nabrak ← False
        for x in [0, objectList.size() - 1]
            if(abs(playerAction.heading -
getHeadingBetween(objectList.get(x))) ≤ 10 and getDistanceBetween(bot,
objectList.get(x)) ≤ getDistanceBetween(bot, musuhList.get(0))) then
                nabrak ← True
                output("Duh nabrak gabisa nembak gan, ")
                break
        // If we can shoot the enemy without colliding with another game
object and enemy size >= 15
        if(!shootTorpedo and !nabrak and playerList.get(0).size ≥ 15) then
            output("Nembak gan")
            playerAction.action ← PlayerActions.FIRETORPEDOES
            shootTorpedo ← True
        else
            // Chase enemy
            output("Kejer musuh gan")
            shootTorpedo ← False
    else if(foodList.size() > 0) then
        // If there is foods nearby
        if(!shootTorpedo and musuhList.size() > 0 and
getDistanceBetween(musuhList.get(0), bot) - musuhList.get(0).size - bot.size ≤
bot.size + 150) then
            // If we can shoot enemy nearby

```

```

playerAction.heading ← getHeadingBetween(musuhList.get(0))

nabrak ← False

for x in [0, foodList.size() - 1]
    if(playerAction.heading = getHeadingBetween(foodList.get(x)))

then
    nabrak ← True
    Break

    if(!nabrak and (musuhList.get(0).size ≥ 20 or
(musuhList.get(0).size ≥ 10 and getDistanceBetween(bot, musuhList.get(0)) <
100))) then
        // If torpedo will not collide and enemy size >= 20
        output("Nembak gan")
        playerAction.action ← PlayerActions.FIRETORPEDOES
        shootTorpedo ← True
    else
        // Eat food if it collides
        output("Kejer makanan gan")
        playerAction.heading ← getHeadingBetween(foodList.get(0))
        shootTorpedo ← False
    else
        // Eat food
        output("Kejer makanan gan")
        playerAction.heading ← getHeadingBetween(foodList.get(0))
        shootTorpedo ← False

if(shootTeleporter > 0) then
    shootTeleporter ← shootTeleporter - 1

playerAction ← playerAction

```

*Function computeNextAction merupakan fungsi utama dari pergerakan bot. Fungsi ini sudah dipecah menjadi berbagai subfungsi untuk modularitas. Maka, fungsi ini digunakan untuk*

mengurutkan prioritas strategi greedy yang akan dipilih. Pertama, player action akan diinisialisasi dan dilakukan pencarian dari keseluruhan objek menjadi beberapa *list* sesuai tipe objek. Setelah objek sudah dipecah menjadi *list-list*, dilakukan *scanning* area dan bot akan memilih area yang terbaik. Lalu, dilakukan pengecekan kepemilikan teleporter dan keaktifan shield. Setelah itu, masuk ke algoritma utama yaitu pengurutan strategi greedy. Sesuai yang sudah dijelaskan pada bab 3.5, bot akan menentukan strategi yang bisa dilakukan pada saat itu. Untuk penjelasan pemeriksaan kondisi, digunakan subfungsi yang sudah dibuat dan dijelaskan pada implementasi fungsi berikutnya.

#### 4.1.4 Scan Area (scanArea)

```
Function List<List<GameObject>> scanArea(List<GameObject> foodList,
List<GameObject> playerList, List<GameObject> obstacleList, List<GameObject>
supernovaList)

    // Initialize area, max score, and list objects
    area ← 0
    maxScore ← 0

    // Iterate area per 30 degrees
    for i in [0,11]
        // Setting heading limit for each area
        headingFirst ← i * 30
        headingLast ← headingFirst + 30
        // Setting max distance for area
        maxDistance ← 250
        // Initializing temporary score and object list
        score ← 0

        // Iterate Food List
        idx ← 0
        while (idx < foodList.size() and getDistanceBetween(foodList.get(idx),
bot) ≤ maxDistance) do
            head ← getHeadingBetween(foodList.get(idx))
            if(head ≥ headingFirst and head ≤ headingLast) then
                if(gameState.getWorld().radius -
getDistanceBetween(foodList.get(idx), gameState.getWorld()) ≥
```

```

sizeUntukEdge(gameState.getWorld().radius, bot.size)) then
    if(foodList.get(idx).getGameObjectType() = ObjectTypes.FOOD)
then
    // Scoring for each valid food
    score ← score + (3.0 / (getDistanceBetween(bot,
foodList.get(idx)) - bot.size) + (3.0 / getDistanceBetween(foodList.get(idx),
gameState.getWorld())))
    for j in [0, playerList.size() - 1]
        if (playerList.get(j).size > bot.size and
getDistanceBetween(bot, playerList.get(j)) - bot.size - playerList.get(j).size ≤
50 and getDistanceBetween(foodList.get(idx), playerList.get(j)) -
playerList.get(j).size > getDistanceBetween(bot, foodList.get(idx)) - bot.size)
then
            score ← score - (3.0 /
(getDistanceBetween(foodList.get(idx), playerList.get(j)) -
playerList.get(j).size))

        else
            // Scoring for each valid superfood
            score ← score + (6.0 / (getDistanceBetween(bot,
foodList.get(idx)) - bot.size) + (6.0 / getDistanceBetween(foodList.get(idx),
gameState.getWorld())))
            for j in [0, playerList.size() - 1]
                if (playerList.get(j).size > bot.size and
getDistanceBetween(bot, playerList.get(j)) - bot.size - playerList.get(j).size ≤
50 and getDistanceBetween(foodList.get(idx), playerList.get(j)) -
playerList.get(j).size > getDistanceBetween(bot, foodList.get(idx)) - bot.size)
then
                    score ← score - (6.0 /
(getDistanceBetween(foodList.get(idx), playerList.get(j)) -
playerList.get(j).size))

                    tempFoodList.add(foodList.get(idx))
                    idx ← idx + 1
// Iterate Player List
idx ← 0
chooseToSkip ← False

```

```

while (idx < playerList.size() and
getDistanceBetween(playerList.get(idx), bot) - playerList.get(idx).size -
bot.size ≤ maxDistance) do
    // Scoring for each valid player with trigonometry
    if(gameState.getWorld().radius -
getDistanceBetween(playerList.get(idx), gameState.getWorld()) ≥ 20) then
        head ← getHeadingBetween(playerList.get(idx))
        theta ← playerList.get(idx).size /
getDistanceBetween(playerList.get(idx), bot)
        theta ← toDegrees(arcsin(theta))
        headFirstEnemy ← (head - theta) mod 360
        headLastEnemy ← (head + theta) mod 360
        if((headingFirst ≥ headFirstEnemy and headingFirst ≤
headLastEnemy) or (headingLast ≥ headFirstEnemy and headingLast ≤ headLastEnemy)
or (head ≥ headingFirst and head ≤ headingLast)) then
            if(playerList.get(idx).size * 6.0 / 5.0 ≤ bot.size) then
                score ← score + (playerList.get(idx).size /
getDistanceBetween(bot, playerList.get(idx)))
                tempPlayerList.add(playerList.get(idx))
            else
                chooseToSkip ← True
                break
        idx ← idx + 1

        if(chooseToSkip) then
            break

    // Iterate Obstacle List
    idx ← 0
    chooseToSkip ← False
    while (idx < obstacleList.size() and
getDistanceBetween(obstacleList.get(idx), bot) ≤ maxDistance) do
        head ← getHeadingBetween(obstacleList.get(idx))
        if(head ≥ headingFirst and head ≤ headingLast) then
            // Scoring for each valid obstacle
            if(obstacleList.get(idx).getGameObjectType() =
ObjectTypes.GAS_CLOUD) then

```

```

        // Skip area if it contains gas cloud and enemy is not
nearby

        if(playerList.size() > 0 and
getDistanceBetween(playerList.get(0), bot) > maxDistance and
getDistanceBetween(obstacleList.get(idx), bot) - bot.size ≤ 75) then
            chooseToSkip ← True
            break
        else
            score ← score - (obstacleList.get(idx).size /
getDistanceBetween(bot, obstacleList.get(idx)))
            tempObstacleList.add(obstacleList.get(idx))

        else
            score ← score - (obstacleList.get(idx).size /
getDistanceBetween(bot, obstacleList.get(idx)))
            tempObstacleList.add(obstacleList.get(idx))

        idx ← idx + 1

        if(chooseToSkip) then
            break

        // Iterate Supernova List
        idx ← 0
        while (idx < supernovaList.size() and
getDistanceBetween(supernovaList.get(idx), bot) ≤ maxDistance) do
            head ← getHeadingBetween(supernovaList.get(idx))
            // Scoring for supernova
            if(head ≥ headingFirst and head ≤ headingLast) then
                score ← score + 10.0 / getDistanceBetween(bot,
supernovaList.get(idx))
                if(playerList.size() > 0) then
                    minDistanceSupernova ←
getDistanceBetween(supernovaList.get(idx), playerList.get(0))
                    idxSupernova ← 0
                    for k in [1, playerList.size()-1]
                        check ← getDistanceBetween(supernovaList.get(idx),
playerList.get(k))

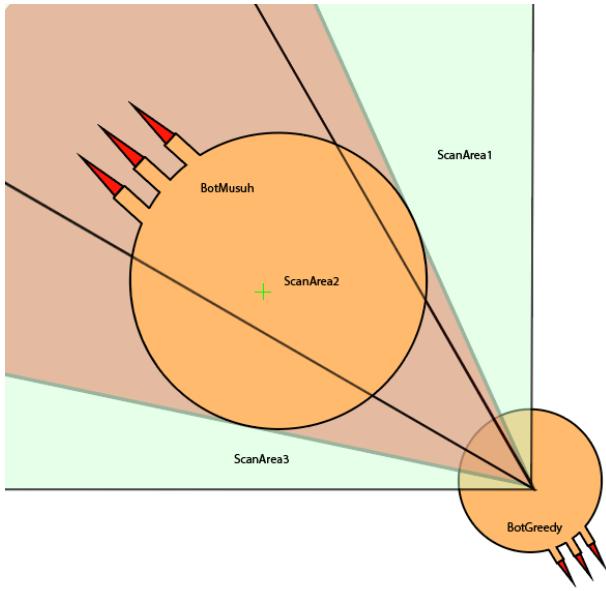
```

```

        if(check < minDistanceSupernova) then
            check ← minDistanceSupernova
            idxSupernova ← k
            score = score - 10.0 /
getDistanceBetween(playerList.get(idxSupernova), supernovaList.get(idx))
            tempSupernovaList.add(supernovaList.get(idx))
            idx ← idx + 1

        // Choosing area
        if(score > maxScore) then
            maxScore ← score
            area ← i
            fixFoodList ← tempFoodList
            fixPlayerList ← tempPlayerList
            fixObstacleList ← tempObstacleList
            fixSupernovaList ← tempSupernovaList
        → Arrays.asList(fixFoodList, fixPlayerList, fixObstacleList,
fixSupernovaList)
    
```

*Function* scanArea digunakan untuk melakukan *scanning* pada area yang berada dekat dengan bot yaitu sejauh 250 dan secara 360 derajat. Area yang terdefinisi disini adalah 30 derajat bagian dari 360 yang artinya terdapat 12 area yang akan discan. *Scanning* ini dilakukan untuk semua objek yang ada pada peta dan digunakan sistem *scoring* untuk menentukan area mana yang terbaik untuk bot pada saat itu. Dimulai dari scanning makanan terlebih dahulu. Makanan disini dibagi menjadi 2 jenis yaitu food dan superfood. Masing-masing memiliki skor yang berbeda. Lalu, untuk *scanning* player, apabila ada musuh yang lebih kecil, maka akan dimasukkan ke dalam *list*. Namun, apabila terdapat musuh yang lebih besar, maka area tersebut akan otomatis di-*skip* oleh bot. Hal itu ditangani dengan rumus sebagai berikut.



**Gambar 4.1.1 Ilustrasi Scan Area Menghindari Musuh**

$$\theta_1 = \theta + \arcsin(R/D)$$

$$\theta_2 = \theta - \arcsin(R/D)$$

dengan  $\theta$  adalah sudut dari bot ke musuh,  $\theta_1$  adalah sudut dari bot ke bagian kiri musuh,  $\theta_2$  adalah sudut dari bot ke bagian kanan musuh, R adalah radius musuh, dan D adalah jarak bot ke musuh. Hal ini ditentukan dengan kondisi apabila area yang di-scan beririsan dengan kapal musuh, maka area itu akan di-skip. Kemudian, akan dilakukan scanning untuk obstacle. Apabila ada gas cloud dan tidak dekat musuh, maka area itu akan di-skip. Sisanya, akan dilakukan scoring sesuai rumus yang sudah ada pada bab sebelumnya. Terakhir, akan dilakukan scanning untuk supernova. Apabila, terdapat supernova pada area, maka akan dilakukan scoring. Selesai melakukan iterasi kepada semua objek, maka perlu ditentukan apakah area tersebut memiliki skor yang lebih baik dari sebelumnya atau tidak. Fungsi akan mengembalikan *list-list* objek pada area yang memiliki skor terbaik.

#### 4.1.5 Fire Supernova (checkFireSupernova)

```
Function checkFireSupernova(List<GameObject> musuhList)
```

```

// Function to check if we can fire supernova or not
check ← False
for i in [0, musuhList.size()]
    if(getDistanceBetween(musuhList.get(i), bot) ≥ 300) then
        check ← True
        break
→ check

```

*Function* ini digunakan untuk mengecek apakah bot dapat menembakkan supernova dengan mencari bot musuh yang terletak paling jauh dari bot dan jarak antara bot dan bot musuh lebih dari 300. Apabila kondisi terpenuhi maka *function* akan meng-*return value* true.

#### 4.1.6 Detonate Supernova (checkDetonateSupernova)

```

Function checkDetonateSupernova(List<GameObject> supernovaList, List<GameObject>
musuhList)
    // Function to check if we can detonate supernova now or not
    if(supernovaList.size() = 0) then
        → False
    else
        supernova ← supernovaList.get(0)
        if(getDistanceBetween(supernova, bot) - bot.size ≤ 200) then
            → False
        else
            check ← False
            for i in [0, musuhList.size()]
                if(getDistanceBetween(musuhList.get(i), supernova) ≤ 100) then
                    check ← True
                    break
            → check

```

*Function* ini digunakan untuk mengecek apakah bot dapat meledakkan supernova. Beberapa kondisi untuk meledakkan supernova adalah mengecek terlebih dahulu apakah ada supernova yang telah ditembakkan pada map, apabila ada maka hitung jarak antara bot dan supernova

apabila jarak lebih besar dari 200 dan ada bot musuh yang jaraknya  $\leq$  100 dari supernova maka function akan mengembalikan *value* true untuk meledakkan supernova.

#### 4.1.7 Shield (checkShieldCondition)

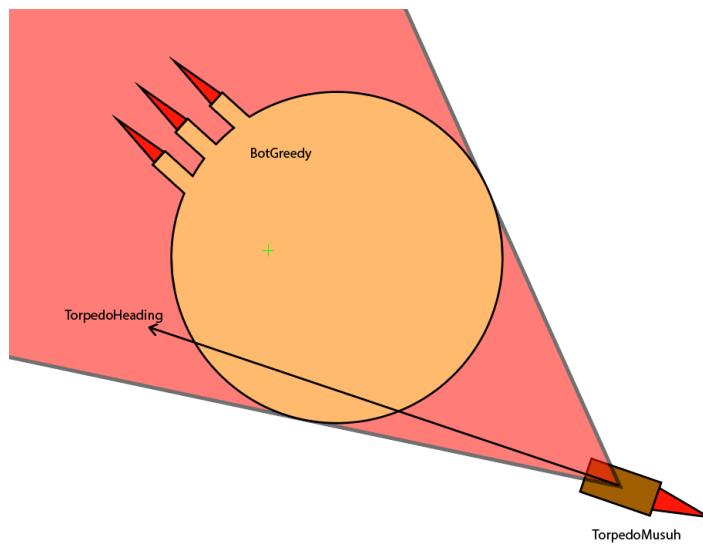
```
Function checkShieldCondition(List<GameObject> torpedoList)
    // Function to check if we need to use shield or not
    count ← 0
    for i in [0...torpedoList.size()-1]
        head ← getHeadingObject(torpedoList.get(i))
        theta ← bot.size / getDistanceBetween(torpedoList.get(i),bot)
        theta ← toDegrees(asin(theta))
        headFirstBot ← (head - theta) mod 360
        headLastBot ← (head + theta) mod 360
        if(torpedoList.get(i).currentHeading ≥ headFirstBot and
        torpedoList.get(i).currentHeading ≤ headLastBot and
        getDistanceBetween(torpedoList.get(i),bot) < 200) then
            count++
            if(i + 1 < torpedoList.size()) then
                for j in [i+1...torpedoList.size()-1]
                    head ← getHeadingObject(torpedoList.get(j))
                    theta ← bot.size /
        getDistanceBetween(torpedoList.get(j),bot)
                    theta ← toDegrees(asin(theta))
                    headFirstBot ← (head - theta) mod 360
                    headLastBot ← (head + theta) mod 360
                    if(torpedoList.get(j).currentHeading ≥ headFirstBot and
        torpedoList.get(j).currentHeading ≤ headLastBot and
        getDistanceBetween(torpedoList.get(j), torpedoList.get(i)) < 200) then
                        count++
                        break
            if(count > 1 and bot.size > 40) then
                → True
            else
                → False
```

*Function* checkShieldCondition digunakan untuk memeriksa apakah bot perlu mengaktifkan shield atau tidak. Untuk memeriksa kondisi tersebut, diperlukan sebuah perhitungan secara geometri karena torpedo bisa saja mengenai bagian bot manapun. Oleh karena itu, hal yang dilakukan untuk pemeriksaan adalah dengan melakukan iterasi dari keseluruhan torpedo yang ada pada peta, lalu memeriksa apakah torpedo tersebut mengarah ke kita atau tidak dengan rumus sebagai berikut.

$$\theta_1 = \theta + \arcsin(R/D)$$

$$\theta_2 = \theta - \arcsin(R/D)$$

dengan  $\theta$  adalah sudut dari torpedo ke bot,  $\theta_1$  adalah sudut dari torpedo ke bagian kiri bot,  $\theta_2$  adalah sudut dari torpedo ke bagian kanan bot, R adalah radius bot, dan D adalah jarak torpedo ke bot. Dari perhitungan di atas, kita dapat membandingkan atribut currentHeading dari torpedo dengan  $\theta_1$  dan  $\theta_2$ , apabila currentHeading berada di antara  $\theta_1$  dan  $\theta_2$ , maka terdapat torpedo yang mengarah ke bot kita. Bot akan mengaktifkan shield apabila terdapat setidaknya dua torpedo yang mengarah ke dirinya dan jaraknya lebih kecil dari 200.



**Gambar 4.1.2** Ilustrasi Cek Torpedo Untuk Shield

#### 4.1.8 Enemy Chase (checkEnemyChase)

```
Function checkEnemyChase(List <GameObject> musuhList)
    // Function to check if we are being chased by an enemy or not
    count ← 0
    for i in [0...musuhList.size()-1]
        if(abs(getHeadingObject(musuhList.get(i)) -
musuhList.get(i).currentHeading) ≤ 10 and getDistanceBetween(musuhList.get(i),
bot) < 200 and musuhList.get(i).size > bot.size) then
            count++
        if (count ≥ 1 and bot.size ≥ 20) then
            output("Ada yang ngejar min, ")
            → True
        else
            → False
```

*Function* ini digunakan untuk mengecek apakah ada musuh yang sedang mengarah ke bot dan *distance* bot musuh tersebut lebih kecil dari 200 atau tidak. Apabila ditemukan bahwa ada bot musuh dengan syarat kondisi di atas, function akan meng-*return* nilai true.

#### 4.1.9 Owner Teleporter (checkOwnerTeleporter)

```
Function checkOwnerTeleporter(List<GameObject> teleportList)
    // Function to check owner teleporter
    for i in [0...teleportList.size()-1]
        if(abs(teleportList.get(i).currentHeading - headingTeleporter) ≤ 3) then
            → True
        → False
```

*Function* ini digunakan untuk mengecek teleporter mana yang merupakan milik bot. Untuk mengecek hal tersebut, akan dilakukan pengecekan heading untuk semua teleporter yang ada di *teleportList*. Apabila ada teleporter yang headingnya  $\pm 3$  dengan heading bot saat menembakkan teleporter maka *function* akan meng-*return* value true.

#### 4.1.10 Getting Teleporter (getTeleporter)

```
Function getTeleporter(List<GameObject> teleportList)
    // Function to get our teleporter
    for i in [0,teleportList.size()]
        if(abs(teleportList.get(i).currentHeading - headingTeleporter) ≤ 3) then
            → teleportList.get(i)
        → NULL
```

*Function* ini digunakan untuk mengambil teleporter milik bot. *Function* ini hampir sama dengan *function* checkOwnerTeleporter yang berbeda hanyalah pada *function* ini value yang di-*return* adalah sebuah *gameObject* yang merupakan teleporter.

#### 4.1.11 Teleport (doTeleport)

```
Function doTeleport(GameObject myTeleport, List<GameObject> musuhList)
    // Function to check if we can teleport now or not
    minDistance ← getDistanceBetween(myTeleport, musuhList.get(0))
    musuhIncar ← musuhList.get(0)
    for i in [1...musuhList.size()-1]
        jarak ← getDistanceBetween(myTeleport, musuhList.get(i))
        if(jarak < minDistance) then
            minDistance ← jarak;
            musuhIncar ← musuhList.get(i)

        if(minDistance - bot.size - musuhIncar.size ≤ 0 and musuhIncar.size + 20 < bot.size) then
            → True
        else
            → False
```

*Function* ini digunakan untuk menentukan apakah bot akan melakukan *teleport* atau tidak. Untuk melakukan *teleport* ada beberapa kondisi yang harus terpenuhi terlebih dahulu. Pertama akan dilakukan pengecekan untuk bot-bot musuh yang masih terdapat pada *map*. Kemudian, akan dipilih bot musuh yang terdekat dengan bot kita. Kemudian, akan dilakukan pengecekan

apakah *size* bot musuh lebih kecil daripada bot, apabila iya maka *function* akan meng-*return value* true.

#### 4.1.12 Edge Constraint (sizeUntukEdge)

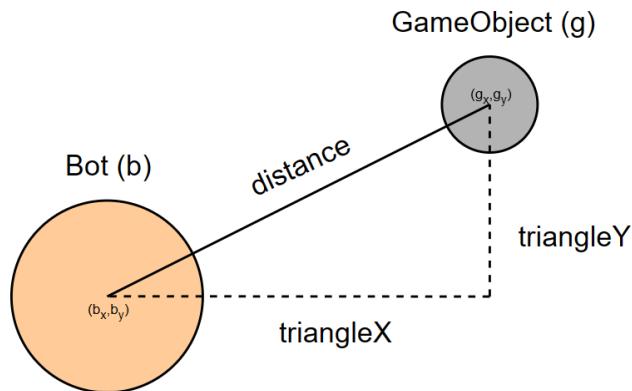
```
Function sizeUntukEdge (int radius, int size)
    // Function to constraint bot movement from edge
    if (size ≤ 0.25 * radius) then
        → 1.7 * size
    else if (size > 0.25 * radius and size ≤ 0.5 * radius) then
        → 1.4 * size
    else
        → 1.2 * size
```

*Function* sizeUntukEdge digunakan untuk menghitung batasan dari suatu objek terhadap edge. Apabila objek dekat dengan edge menurut perhitungan fungsi ini, maka objek tidak dianggap valid oleh strategi. Fungsi ini juga digunakan agar bot tidak berada di posisi yang dekat dengan *edge*. Rumus yang digunakan untuk menghitung adalah jika *size* dari bot lebih kecil dari  $\frac{1}{4}$  radius world, maka fungsi akan mengembalikan batasan yaitu  $1.7 * size$  bot. Jika *size* lebih besar dari  $\frac{1}{4}$  radius world dan lebih kecil dari  $\frac{1}{2}$  radius world, maka fungsi akan mengembalikan  $1.4 * size$  bot. Jika *size* lebih besar dari  $\frac{1}{2}$  radius world, maka fungsi akan mengembalikan  $1.2 * size$  bot. Hasil perhitungan ini adalah jarak minimal yang harus dimiliki oleh objek antara dirinya dengan *edge*.

#### 4.1.13 Distance Between Two Objects (getDistanceBetween)

```
Function getDistanceBetween(GameObject object1, GameObject object2)
    // Function to get distance between two object
    triangleX ← abs(object1.getPosition().x - object2.getPosition().x)
    triangleY ← abs(object1.getPosition().y - object2.getPosition().y)
    → sqrt(triangleX * triangleX + triangleY * triangleY)
```

*Function* ini digunakan untuk mencari jarak antara dua objek dengan menggunakan posisi kedua objek tersebut yang merupakan titik. Berikut adalah ilustrasi dari perhitungan jarak antara dua objek.



**Gambar 4.1.3** Ilustrasi Hitung Jarak Antara Bot dengan Game Object

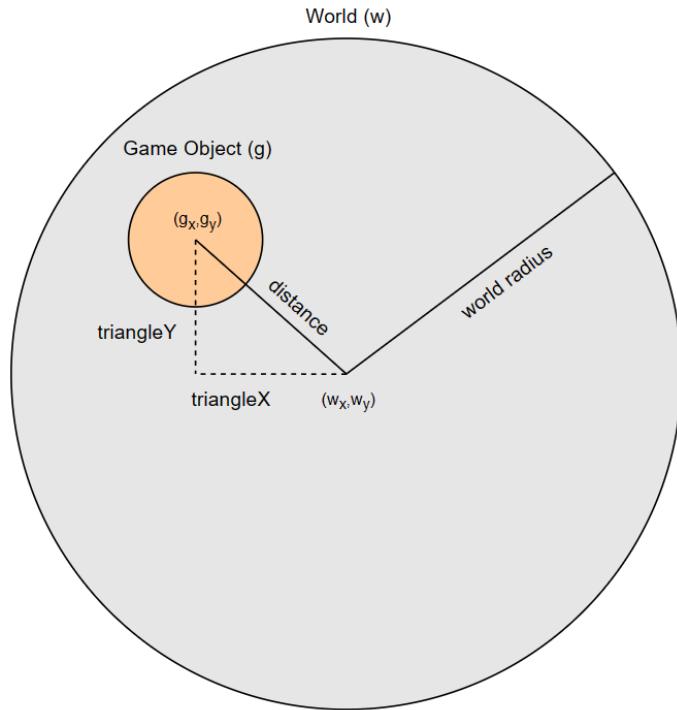
Secara matematis, jarak antara dua objek dapat dituliskan sebagai berikut.

$$\begin{aligned} \textit{distance} &= \sqrt{\textit{triangleX}^2 + \textit{triangleY}^2}, \\ \textit{triangleX} &= |b_x - g_x|, \\ \textit{triangleY} &= |b_y - g_y| \end{aligned}$$

#### 4.1.14 Distance Between an Object and World (getDistanceBetween)

```
Function getDistanceBetween(GameObject object1, World world)
    // Function to get distance between an object and world center
    triangleX ← abs(object1.getPosition().x - world.getCenterPoint().x)
    triangleY ← abs(object1.getPosition().y - world.getCenterPoint().y)
    → sqrt(triangleX * triangleX + triangleY * triangleY)
```

*Function* ini digunakan untuk mencari jarak antara objek dengan titik tengah world menggunakan posisi objek tersebut dengan posisi titik tengah world yang merupakan titik. Berikut adalah ilustrasi dari perhitungan jarak antara objek dengan titik tengah world.



**Gambar 4.1.4** Ilustrasi Hitung Jarak Antara Bot dengan Titik Tengah Peta

Secara matematis, jarak antara objek dengan titik pusat world dapat dituliskan sebagai berikut.

$$\begin{aligned}
 distance &= \sqrt{triangleX^2 + triangleY^2}, \\
 triangleX &= |g_x - w_x|, \\
 triangleY &= |g_y - w_y|
 \end{aligned}$$

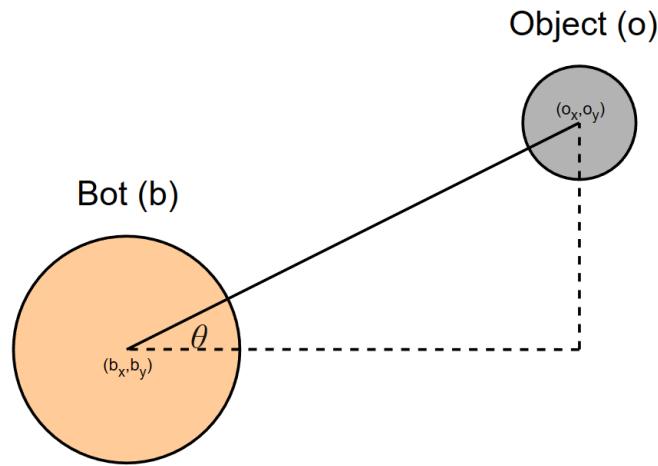
#### 4.1.15 Heading From Bot to Object (getHeadingBetween)

```

Function getHeadingBetween(GameObject otherObject)
    // Function to get heading from bot to object
    direction ← toDegrees(arctan(otherObject.getPosition().y -
bot.getPosition().y,
                                otherObject.getPosition().x - bot.getPosition().x))
    → (direction + 360) mod 360

```

*Function* `getHeadingBetween` ini memiliki parameter `world` sehingga fungsi ini menghitung jarak dari bot ke titik tengah dari `world` dengan rumus sebagai berikut.



**Gambar 4.1.5** Ilustrasi Hitung Sudut Antara Bot dengan Objek Lain

Secara matematis, sudut antara bot dengan objek lain dapat dituliskan sebagai berikut.

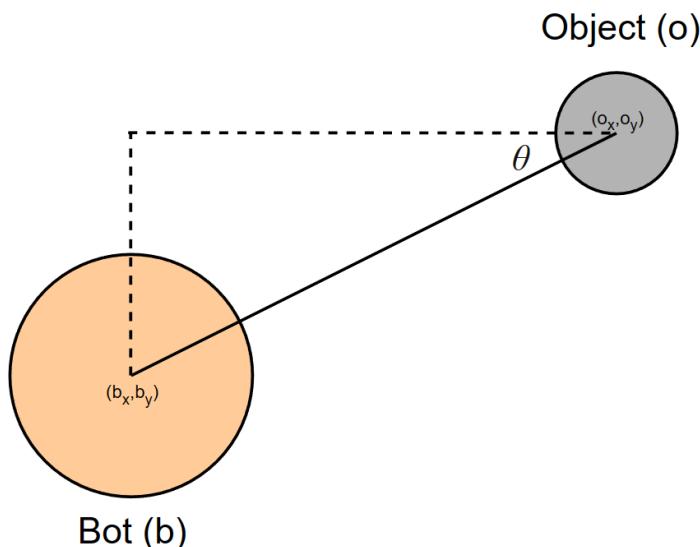
$$\theta = \arctan\left(\frac{o_y - b_y}{o_x - b_x}\right)$$

dengan  $(b_x, b_y)$  adalah posisi x dan y dari bot dan  $(o_x, o_y)$  adalah posisi x dan y dari objek.

#### 4.1.16 Heading From Object to Bot (`getHeadingObject`)

```
Function getHeadingObject(GameObject otherObject)
    // Function to get heading from object to bot
    direction ← toDegrees(arctan(bot.getPosition().y -
otherObject.getPosition().y,
                                bot.getPosition().x - otherObject.getPosition().x))
    → (direction + 360) mod 360
```

*Function* `getHeadingObject` ini memiliki parameter `GameObject` sehingga fungsi ini menghitung sudut dari objek tersebut ke bot dengan rumus sebagai berikut.



**Gambar 4.1.6 Ilustrasi Hitung Sudut Antara Objek Lain dengan Bot**

Secara matematis, sudut antara objek lain dengan bot dapat dituliskan sebagai berikut.

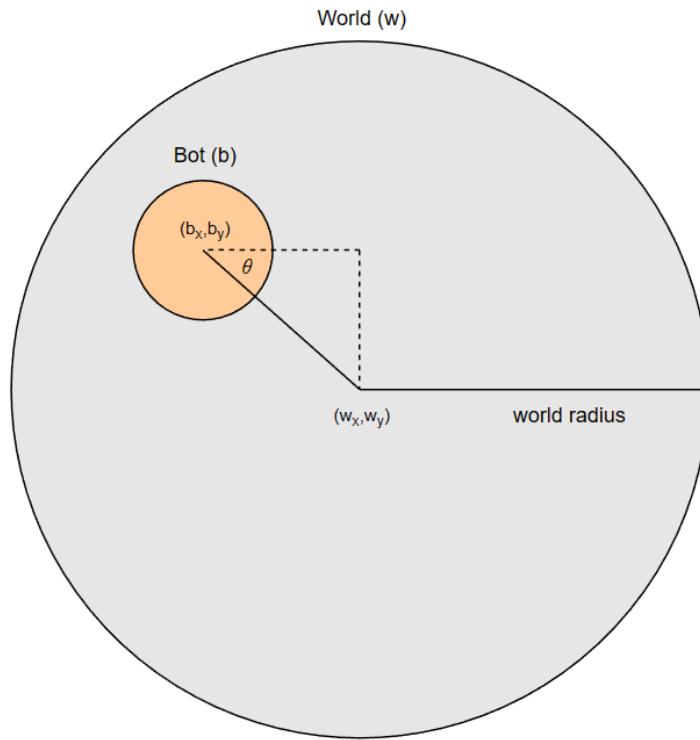
$$\theta = \arctan\left(\frac{b_y - o_y}{b_x - o_x}\right)$$

dengan  $(b_x, b_y)$  adalah posisi x dan y dari bot dan  $(o_x, o_y)$  adalah posisi x dan y dari objek.

#### 4.1.17 Heading From Bot to World Center (getHeadingBetween)

```
Function getHeadingBetween(World world)
    // Function to get heading from bot to world
    direction ← toDegrees(arctan(world.getCenterPoint().y - bot.getPosition().y,
        world.getCenterPoint().x - bot.getPosition().x))
    → (direction + 360) mod 360
```

*Function getHeadingBetween* ini memiliki parameter *world* sehingga fungsi ini menghitung sudut dari bot ke titik tengah dari *world* dengan rumus sebagai berikut.



**Gambar 4.1.7** Ilustrasi Hitung Sudut Antara Bot dengan Titik Pusat Peta

Secara matematis, sudut antara objek lain dengan bot dapat dituliskan sebagai berikut.

$$\theta = \arctan\left(\frac{w_y - b_y}{w_x - b_x}\right)$$

dengan  $(b_x, b_y)$  adalah posisi x dan y dari bot dan  $(w_x, w_y)$  adalah posisi x dan y dari titik tengah world.

#### 4.1.18 Convert Radian to Degrees (toDegrees)

```
Function toDegrees(double v)
    // Function to calculate degrees from radian
    → (int) (v * (180 / π))
```

*Function toDegrees* mengkonversikan nilai radian ke derajat. Rumus yang digunakan adalah sebagai berikut.

$$\theta = v * \frac{180}{\pi}$$

dengan  $\theta$  adalah sudut dalam derajat dan  $v$  adalah sudut dalam radian.

## 4.2 Penjelasan Struktur Data pada Bot Permainan Galaxio

Bot permainan Galaxio dirancang dengan pemrograman berorientasi objek (*object-oriented programming*) yang terdiri dari beberapa kelas beserta struktur datanya. Berikut adalah penjelasan mengenai kelas beserta struktur datanya.

### 4.2.1 BotService (BotService.java)

Struktur data pada bot permainan Galaxio berupa class yang memiliki atribut-atribut yang bersifat *private*. Atribut tersebut antara lain adalah sebagai berikut.

1. bot

```
private GameObject bot;
```

Atribut bot adalah atribut bertipe GameObject. Atribut ini mendefinisikan objek bot kapal pada permainan.

2. playerAction

```
private PlayerAction playerAction;
```

Atribut playerAction adalah atribut bertipe PlayerAction. Atribut ini mendefinisikan aksi yang dilakukan oleh pemain pada permainan.

3. gameState;

```
private GameState gameState;
```

Atribut gameState adalah atribut bertipe GameState. Atribut ini mendefinisikan game state, yaitu state pada permainan yang berisikan World, list dari GameObject (gameObjects dan playerGameObjects).

4. shootTorpedo;

```
private boolean shootTorpedo; // Indicates bot shoted torpedo
```

Atribut shootTorpedo adalah atribut bertipe boolean. Atribut ini bernilai true jika bot kapal menembakkan torpedo atau bernilai false jika bot kapal tidak menembakkan torpedo.

5. headingTeleporter;

```
private int headingTeleporter; // Saving our heading teleporter
```

Atribut headingTeleporter adalah atribut bertipe int. Atribut ini mendefinisikan arah dari pergerakan teleporter.

6. shootTeleporter;

```
private int shootTeleporter; // Indicates bot shoted teleporter
```

Atribut shootTeleporter adalah atribut bertipe int. Atribut ini mendefinisikan jumlah teleporter yang sedang ditembakkan oleh bot kapal.

7. shieldActive;

```
private boolean shieldActive; // Indicates shield is active or not
```

Atribut shieldActive adalah atribut bertipe boolean. Atribut ini bernilai true jika bot kapal sedang mengaktifkan shield atau bernilai false jika bot kapal tidak sedang mengaktifkan shield.

8. tickShield;

```
private int tickShield; // Counting tick for duration of shield
```

Atribut tickShield adalah atribut bertipe int. Atribut ini mendefinisikan seberapa lama shield diaktifkan.

9. afterBurnerActive;

```
private boolean afterBurnerActive; //Indicates afterburner is active or not
```

Atribut afterBurnerActive adalah atribut bertipe boolean. Atribut ini bernilai true jika bot kapal sedang mengaktifkan afterburner atau bernilai false jika bot kapal tidak sedang mengaktifkan afterburner.

10. shootSupernova;

```
private boolean shootSupernova; // Indicates bot shoted supernova
```

Atribut shootSupernova adalah atribut bertipe boolean. Atribut ini bernilai true jika bot kapal sedang menembakkan supernova atau bernilai false jika bot kapal sedang tidak menembakkan supernova.

#### 4.2.2 GameState (GameState.java)

Struktur data pada game state berupa class yang memiliki atribut-atribut yang bersifat *private*. Atribut tersebut antara lain adalah sebagai berikut.

1. world

```
public World world;
```

Atribut world adalah atribut bertipe World. Atribut ini mendefinisikan world permainan, yaitu posisi titik tengah peta, radius peta, dan tick sekarang.

2. gameObjects

```
public List<GameObject> gameObjects;
```

Atribut gameObjects adalah atribut bertipe List of GameObject. Atribut ini mendefinisikan semua game object nonpemain yang ada pada permainan, misalnya gas cloud, asteroid fields, food, dan lain-lain. Adapun game object bertipe pemain memiliki 6 atribut, yaitu id, size, speed, currentHeading, position, dan gameObjectType.

3. playerGameObjects

```
public List<GameObject> playerGameObjects;
```

Atribut playerGameObjects adalah atribut bertipe List of GameObject. Atribut ini mendefinisikan semua game object pemain yang ada pada permainan. Adapun game object

bertipe pemain memiliki 11 atribut, yaitu id, size, speed, currentHeading, position, gameObjectType, effect, torpedoSalvoCount, supernovaAvailable, teleporterCount, dan shieldCount.

#### 4.2.3 World (World.java)

Struktur data pada world berupa class yang memiliki atribut-atribut yang bersifat *private*. Atribut tersebut antara lain adalah sebagai berikut.

1. centerPoint

```
public Position centerPoint;
```

Atribut centerPoint adalah atribut bertipe Position. Atribut ini mendefinisikan posisi titik tengah peta.

2. radius

```
public Integer radius;
```

Atribut radius adalah atribut bertipe Integer. Atribut ini mendefinisikan radius (ukuran) dari peta.

3. currentTick

```
public Integer currentTick;
```

Atribut currentTick adalah atribut bertipe Integer. Atribut ini mendefinisikan jumlah tick yang sudah berlalu sejak permainan dimulai.

### 4.3 Pengujian Bot Permainan Galaxio

Berikut adalah analisis yang dilakukan terhadap beberapa pertandingan melawan ReferenceBot menggunakan *visualizer*.

#### 4.3.1 Pengujian Greedy by Scan Area



Gambar 4.3.1 Bot Melakukan Scan Area

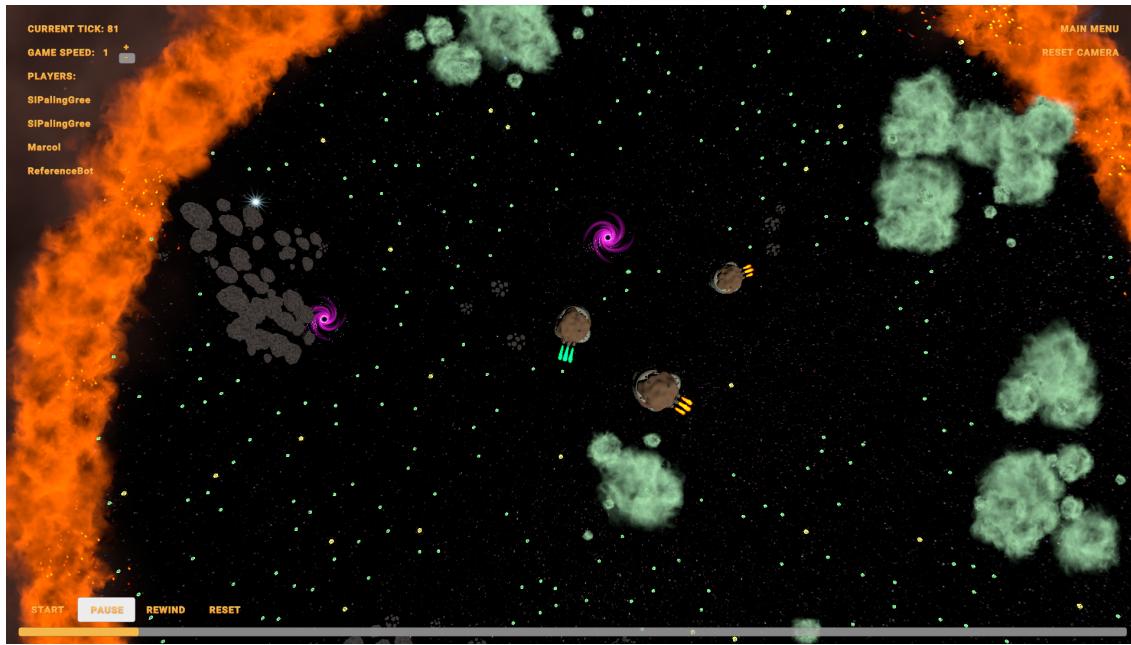


Gambar 4.3.2 Bot Menuju Daerah Optimal

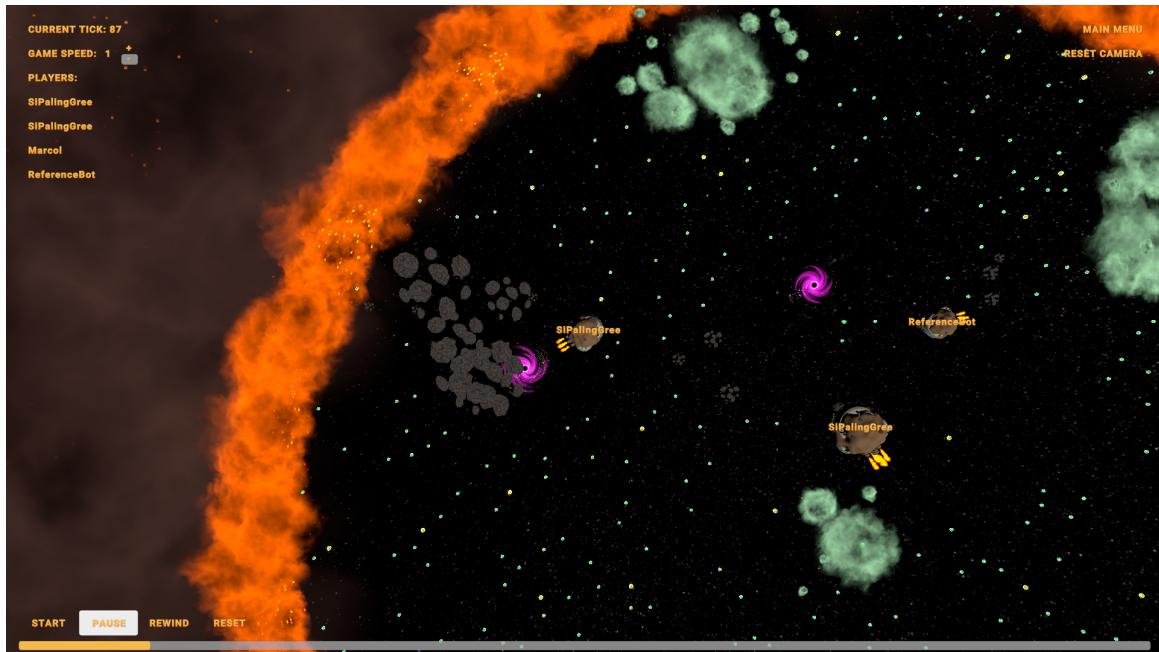
Bot SiPalingGreedy melakukan *scan area* pada setiap tick untuk menentukan daerah yang paling menguntungkan (optimal). Pada Gambar 4.3.2, dapat dilihat bahwa bot telah bergerak

menuju daerah yang banyak makanan. Hal ini terjadi karena daerah tersebut mendapatkan skor tertinggi pada sistem *scoring* yang ada pada program.

#### 4.3.2 Pengujian Greedy by Afterburner



Gambar 4.3.3 Bot Saat Melakukan Aksi ACTIVATEAFTERBURNER



Gambar 4.3.4 Bot Saat Melakukan Aksi STOPAFTERTURNER

Pada Gambar 4.3.3, bot SiPalingGreedy berada di posisi yang dekat dengan musuh, sehingga bot mengaktifkan afterburner dan melakukan aksi berdasarkan strategi Greedy terhadap aspek lainnya. Pada kasus ini, bot mengaktifkan afterburner dan mengincar makanan di sekitarnya (Gambar 4.3.4).

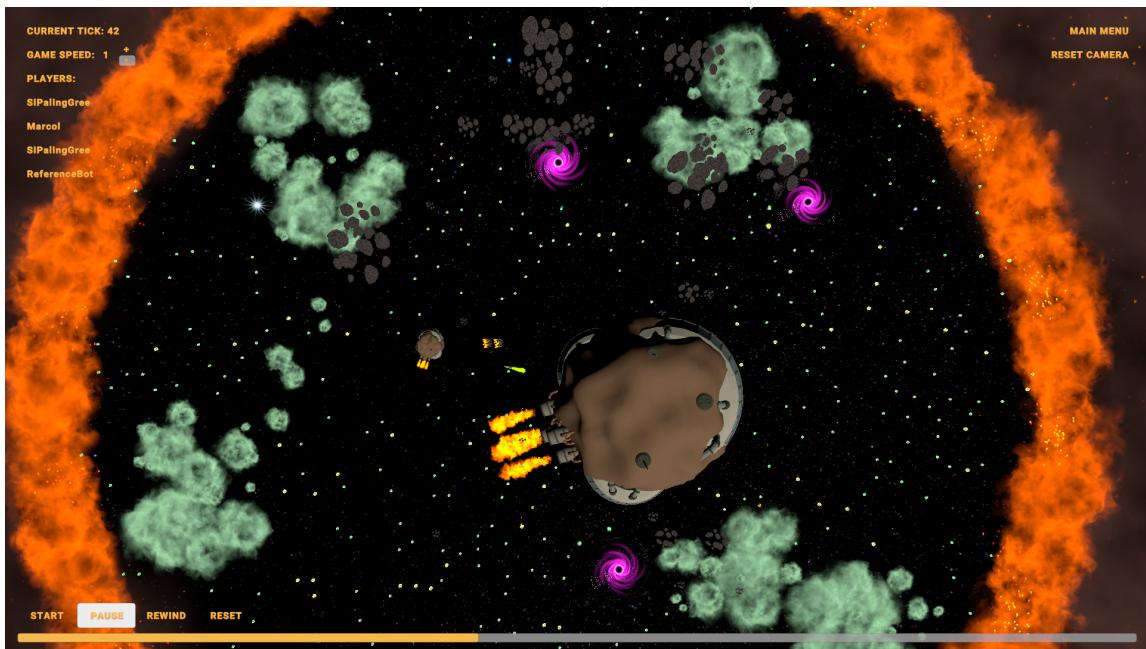
### 4.3.3 Pengujian Greedy by Torpedo



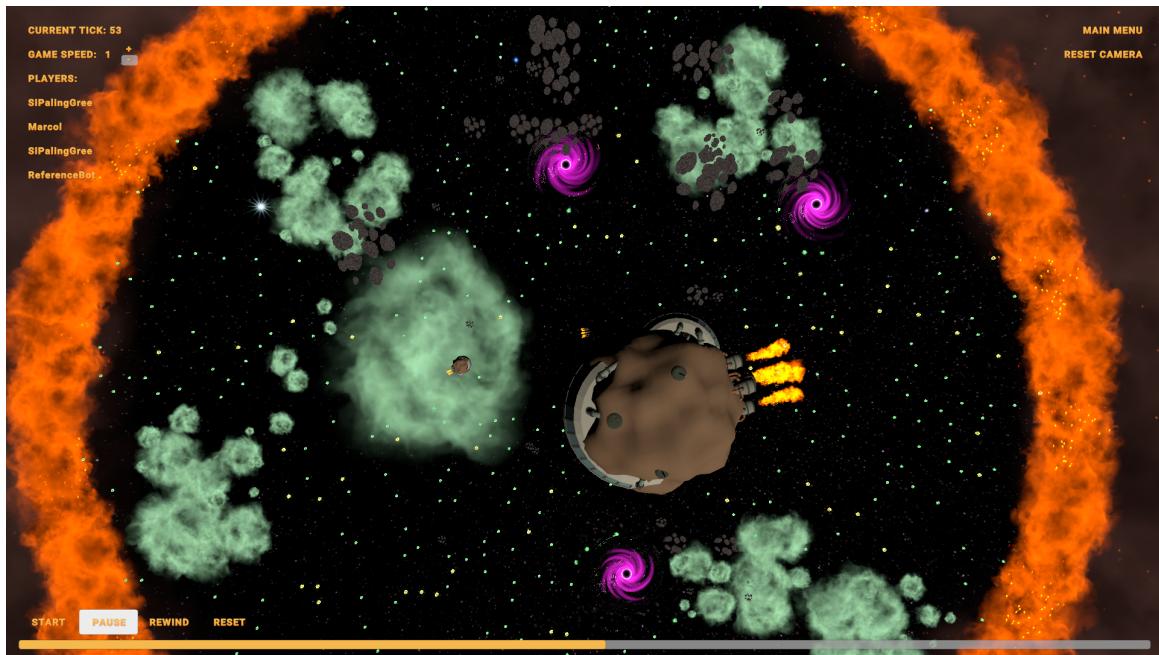
Gambar 4.3.5 Bot Saat Melakukan Aksi FIRETORPEDOES

Pada Gambar 4.3.5, terdapat bot musuh dekat bot SiPalingGreedy, sehingga bot menembakkan torpedo untuk mengecilkan ukuran bot musuh. Bot hanya akan menembakkan torpedo jika tidak ada objek lain di antara bot SiPalingGreedy dengan bot target. Hal ini dilakukan agar torpedo yang ditembakkan efektif mengenai musuh, tidak menabrak objek lain, misalnya makanan.

#### 4.3.4 Pengujian Greedy by Supernova



Gambar 4.3.6 Bot Saat Melakukan Aksi FIRESUPERNOVA



Gambar 4.3.7 Bot Saat Melakukan Aksi DETONATESUPERNOVA

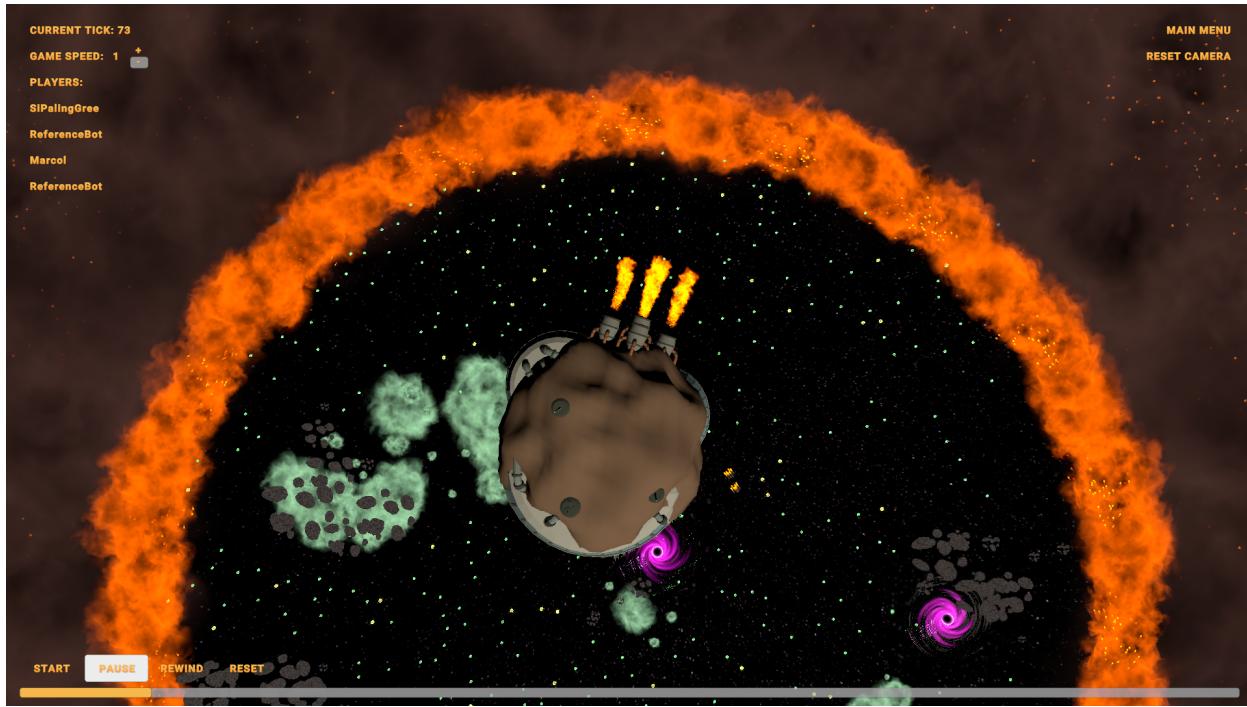
Pada Gambar 4.3.6, bot SiPalingGreedy menembakkan supernova bomb ke arah bot musuh. Supernova bomb ditembakkan kepada daerah musuh dengan jarak yang cukup jauh dari bot SiPalingGreedy. Jarak tersebut adalah jarak yang cukup jauh sehingga bot tidak terkena efek

apa pun dari ledakan supernova. Pada Gambar 4.3.7, supernova bomb diledakkan menjadi gas cloud pada jarak yang dekat dengan bot musuh yang ditargetkan dan juga cukup jauh sehingga bot tidak terkena *damage*. Bot musuh yang terkena ledakan supernova bertambah kecil karena efek gas cloud.

#### 4.3.5 Pengujian Greedy by Teleport



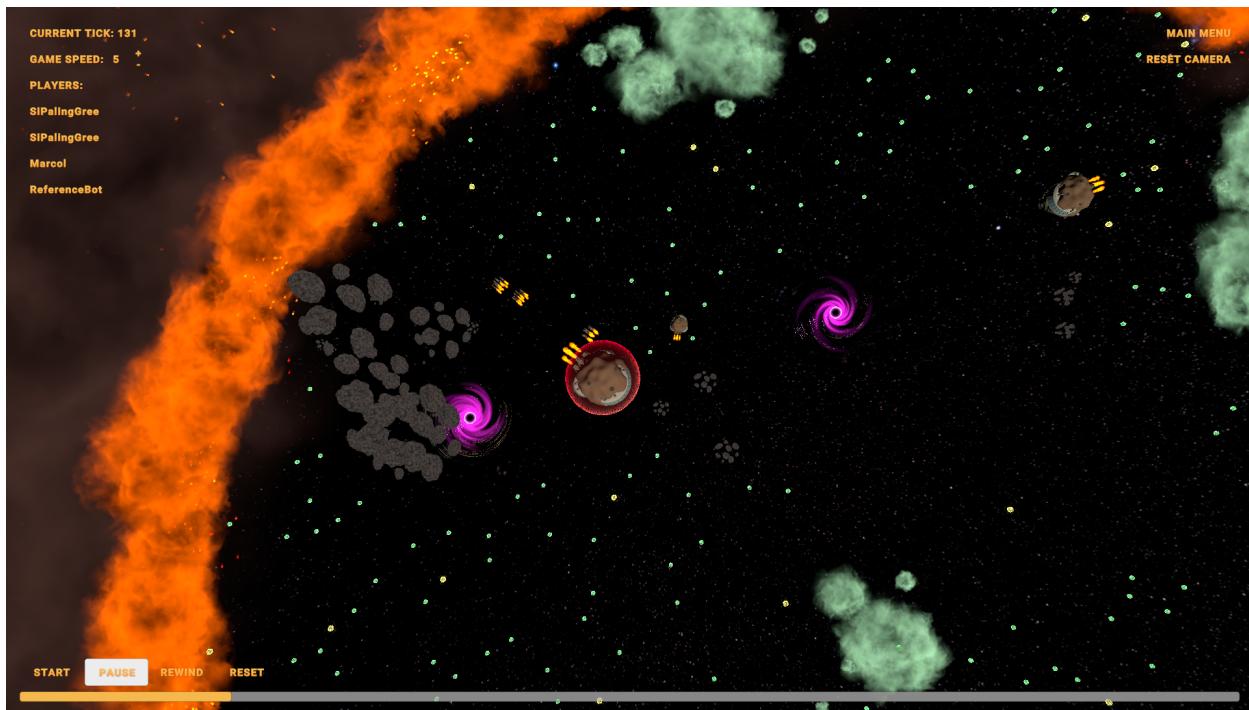
**Gambar 4.3.8** Bot Saat Melakukan Aksi FIRETELEPORT



**Gambar 4.3.9** Bot Saat Melakukan Aksi TELEPORT

Pada kedua gambar di atas, bisa dilihat bahwa bot menggunakan strategi greedy by teleport sehingga bot menembakkan teleport ke musuh yang lebih kecil darinya dan melakukan aksi teleport ketika teleporter sudah mencapai musuh. Untuk kasus ini, bot tidak memiliki supernova sehingga bot memilih untuk menggunakan teleporter untuk memakan musuh. Kasus ini sudah menghasilkan nilai yang optimal karena bot bisa melakukan berbagai cara untuk membesarkan dirinya seperti memakan banyak makanan namun berdasarkan scoring lebih menguntungkan jika bot melakukan aksi teleport untuk memakan musuh dan membesarkan diri.

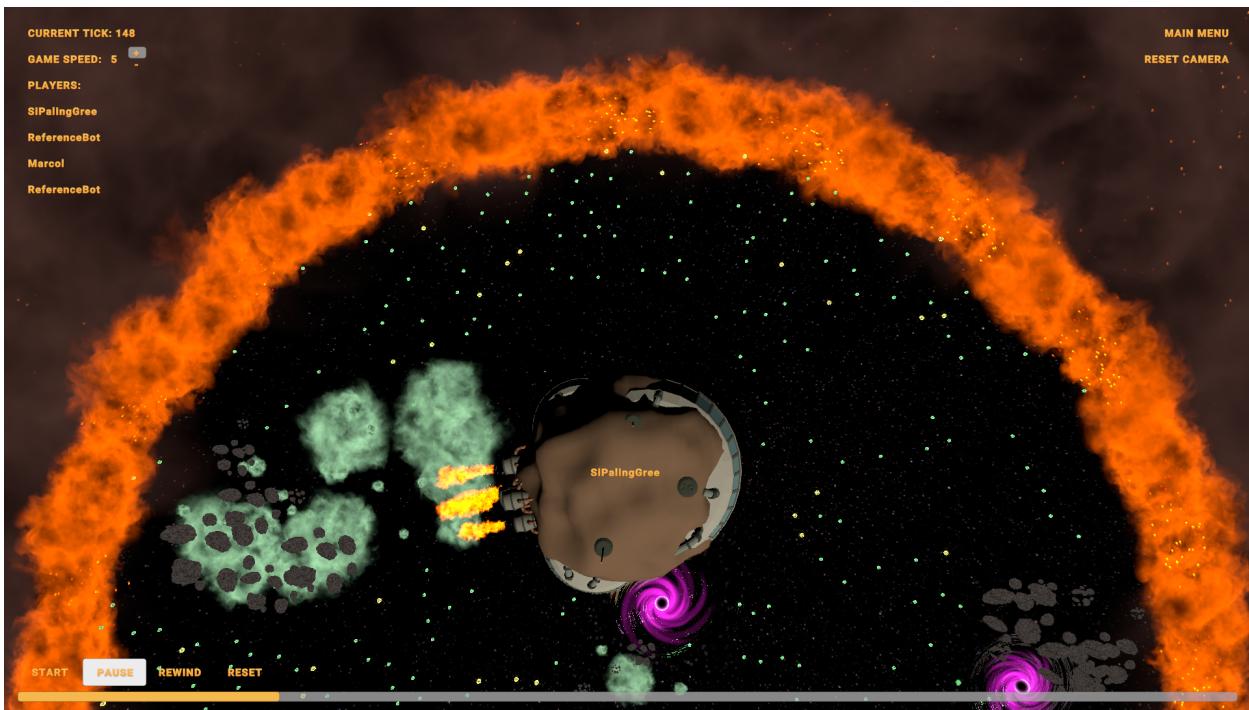
#### 4.3.6 Pengujian Greedy by Shield



Gambar 4.3.10 Bot Saat Melakukan Aksi ACTIVATESHIELD

Pada gambar di atas, bisa dilihat bahwa bot telah mengaktifkan shield dengan menggunakan strategi greedy by shield. Hasil ini sudah menghasilkan nilai yang optimal karena bisa dilihat bahwa bot musuh berusaha keras untuk mengecilkan bot kami dan membesarkan dirinya. Namun, hal itu bisa ditiadakan dengan mengaktifkan shield sehingga bot kami tidak terkena torpedo dan torpedo juga terpental keluar peta.

#### 4.3.7 Pengujian Greedy by Makanan



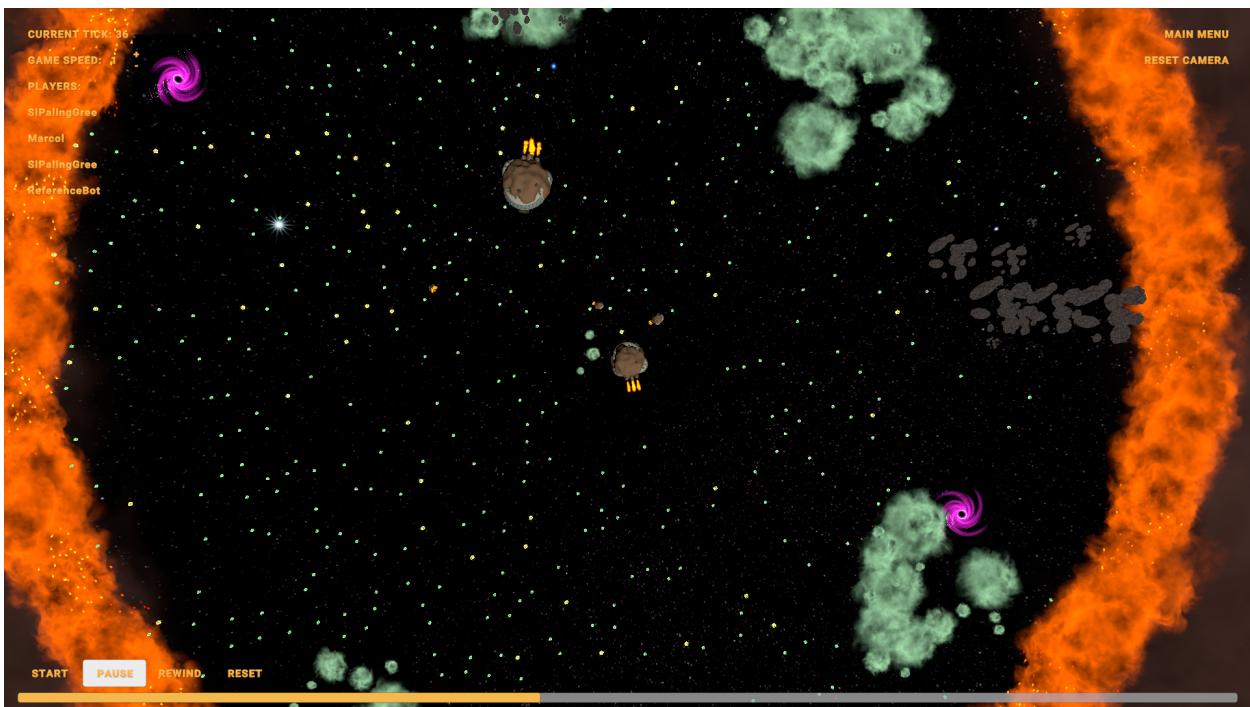
Gambar 4.3.11 Bot Saat Sedang Melakukan Aksi Greedy by Makanan (1)



Gambar 4.3.12 Bot Saat Sedang Melakukan Aksi Greedy by Makanan (2)

Pada gambar di atas, bisa dilihat bahwa bot sedang mengejar makanan yang terdekat dengan dirinya. Kasus ini menghasilkan nilai yang optimal karena bot saat itu berada di posisi yang tidak menguntungkan yaitu dekat gas cloud sehingga bot mengejar makanan yang menjauhi gas cloud dan juga ke area yang lebih menguntungkan dirinya. Bot juga memilih daerah yang memiliki super food lebih banyak karena berdasarkan scoring pun super food menghasilkan skor yang lebih banyak.

#### 4.3.8 Pengujian Greedy by Musuh



**Gambar 4.3.13** Bot Sebelum Melakukan Aksi Greedy by Musuh



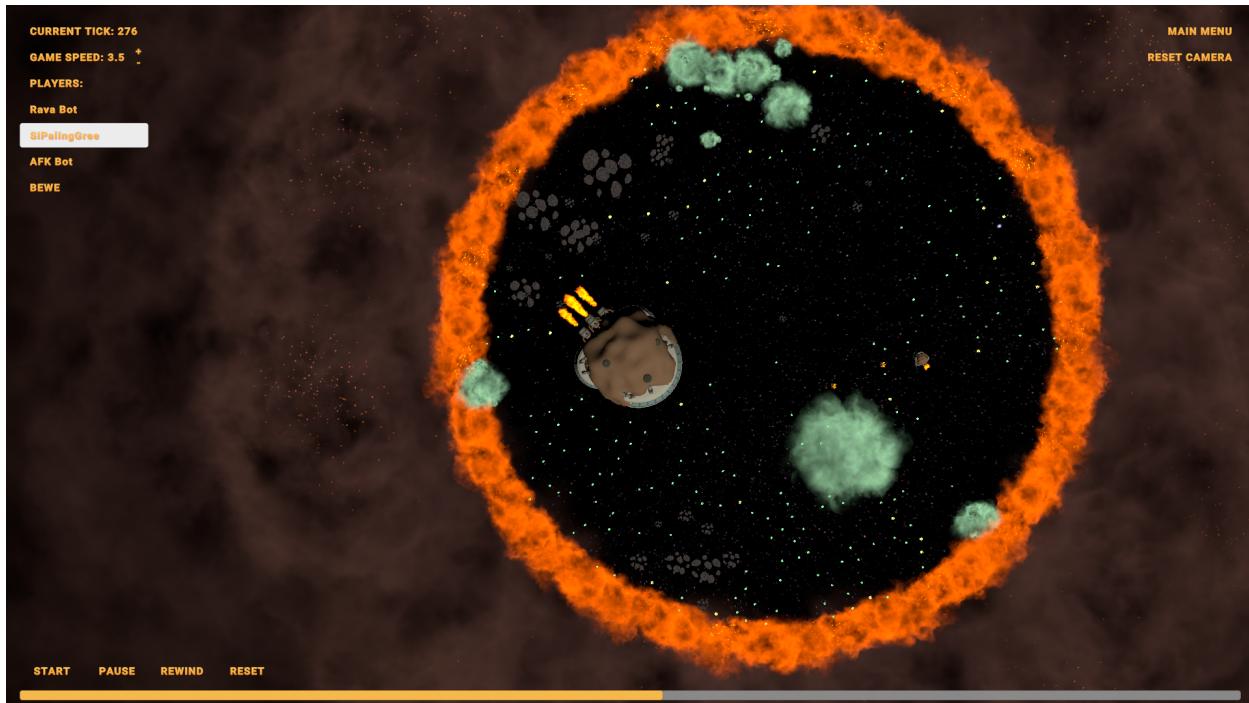
**Gambar 4.3.14** Bot Setelah Melakukan Aksi Greedy by Musuh

Pada gambar di atas, dapat dilihat bahwa bot sedang mengejar dua musuh yang lebih kecil dari dirinya. Kasus ini sudah menggunakan strategi Greedy by musuh dengan mengejar kedua musuh yang lebih kecil dan memakan mereka. Kasus ini juga sudah menghasilkan nilai yang optimal karena berdasarkan *scoring*, skor yang dihasilkan dari dua bot sangatlah besar untuk area tersebut sehingga bot memilih area tersebut dan mengejar musuh untuk dimakan.

#### 4.3.9 Pengujian Greedy by Obstacle



Gambar 4.3.15 Bot Menghindari Obstacle Gas Cloud



Gambar 4.3.16 Bot Menghindari Obstacle Asteroid Field

Pada Gambar 4.3.15, bot sedang berada di dekat gas cloud. Untuk mencegah terkena *damage* yang ditimbulkan gas cloud, bot menghindari area gas cloud jika tidak ada musuh dengan ukuran lebih besar yang sedang mengejar bot. Bot tetap menghindari area gas cloud meskipun terdapat banyak makanan di sekitar gas cloud.

Pada Gambar 4.3.16, bot sedang berada di dekat asteroid field. Untuk mencegah terkena *damage* yang ditimbulkan asteroid field, bot menghindari area asteroid field. Hal ini terjadi karena asteroid field mengurangi skor area pada sistem *scoring scan area*, sedangkan pada arah yang sedang dituju bot terdapat banyak makanan, sehingga skor area asteroid field lebih rendah daripada area dengan banyak makanan.

## **BAB V**

### **PENUTUP**

#### **5.1 Simpulan**

Galaxio merupakan suatu permainan pertandingan antar bot kapal pemainnya yang berlatar belakang di luar angkasa. Pemain memenangkan permainan jika dirinya menjadi pemain terakhir yang bertahan hingga akhir permainan. Untuk memenangkan permainan, pemain harus memiliki ukuran bot kapal terbesar di antara pemain lainnya. Karena permasalahan permainan Galaxio merupakan permasalahan optimasi, maka algoritma Greedy dapat diimplementasikan pada permainan ini.

Dengan algoritma Greedy, bot kapal dapat menentukan aksi yang harus dilakukan agar memperoleh keuntungan sebesar-besarnya serta terkena kerugian sekecil-kecilnya. Implementasi algoritma Greedy adalah dengan menentukan prioritas aksi yang harus diambil pada setiap *tick*. Algoritma Greedy yang diimplementasikan didasarkan pada komponen permainan, mulai dari scan area, makanan, musuh, obstacle, supernova, teleport, torpedo, afterburner, dan shield. Dengan metode sistem *scoring*, bot kapal dapat menentukan arah pergerakan serta aksi yang akan dilakukan.

#### **5.2 Saran**

Dari proses penggeraan tugas besar ini, kami memiliki banyak kendala selama mengerjakan karena waktu yang kurang cukup dan tugas lainnya yang juga harus kami selesaikan. Oleh karena itu, kami memiliki beberapa saran agar tugas ini bisa menjadi lebih baik, yaitu

- a. Menyediakan tutorial *coding* untuk melengkapi hal-hal yang kurang dari starter-pack (karena jika tidak, *engine* yang dijalankan setiap bot bisa berbeda-beda).
- b. Melengkapi strategi-strategi Greedy yang masih bisa diimplementasikan seperti Greedy by wormhole.
- c. Memperbanyak penggunaan perhitungan matematis agar pergerakan bot dan objek yang dimilikinya lebih akurat.
- d. Membetulkan *engine-engine* yang masih rusak, seperti apabila supernova tidak diledakkan dan keluar dari peta, maka *engine* akan crash.
- e. Menggunakan komponen *effects* agar strategi Greedy semakin optimal.

### **5.3 Refleksi**

- a. Arleen Chrysantha Gunardi | 13521059

Tugas besar ini menjadi media bagi saya untuk melakukan eksplorasi lebih lanjut mengenai penerapan algoritma Greedy untuk merancang strategi permainan. Melalui tugas ini, saya juga diberi kesempatan untuk menulis program berorientasi objek dalam bahasa Java serta memahami cara kerja serta struktur program yang cukup kompleks. Selain itu, saya juga dapat meningkatkan kemampuan untuk bekerja sama dalam tim, mulai dari *brainstorming* ide algoritma, melakukan *testing* program, dan lain-lain.

- b. Michael Jonathan Halim | 13521124

Dari tugas besar ini, saya belajar banyak hal baru seperti bagaimana cara implementasi Object-Oriented Programming dengan baik, bagaimana cara berkolaborasi dengan teman sekelompok lebih baik karena uniknya untuk tugas besar ini memang perlu banyak sekali strategi yang harus dirancang secara bersama-sama, dan bagaimana menganalisis cara pikir lawan untuk menemukan strategi baru yang lebih optimum.

- c. Marcel Ryan Antony | 13521127

Dari tugas besar ini, saya belajar banyak hal baru seperti pengaplikasian algoritma Greedy untuk mencari strategi terbaik untuk sebuah permainan. Selain itu, saya juga menjadi lebih paham dengan konsep Object-Oriented Programming dan bahasa Java karena program-program yang dibuat pada bot ini hampir semua menggunakan konsep Object-Oriented Programming dan menggunakan bahasa pemrograman Java. Selain hal-hal teknikal, pada tugas besar kali ini saya juga belajar untuk meningkatkan beberapa *soft-skill* saya seperti, kemampuan berkomunikasi dan bekerja sama dengan tim, manajemen waktu, berpikir secara kritis, dan lain-lain.

## **DAFTAR PUSTAKA**

Shiba, Najmaa. 2022. IDS Digital College: STMIK Indo Daya Suvana.

<https://ids.ac.id/pengertian-game-engine-jenis-dan-fungsinya/>

Munir, Rinaldi. 2023. Algoritma Greedy (2023) Bag1. Homepage Rinaldi Munir Sekolah Teknik Elektro dan Informatika (STEI) ITB.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

Munir, Rinaldi. 2023. Algoritma Greedy (2023) Bag2. Homepage Rinaldi Munir Sekolah Teknik Elektro dan Informatika (STEI) ITB.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf)

Munir, Rinaldi. 2023. Algoritma Greedy (2023) Bag3. Homepage Rinaldi Munir Sekolah Teknik Elektro dan Informatika (STEI) ITB.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf)

Entelect. 2021. Entelect Challenge 2021 - Galaxio.

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

## **LAMPIRAN**

Tautan repository tugas besar: [https://github.com/maikeljh/Tubes1\\_SiPalingGreedy](https://github.com/maikeljh/Tubes1_SiPalingGreedy)

Tautan video: <https://youtu.be/p0QRm8lh5IA>