

**LAPORAN TUGAS BESAR 1  
IF2211 STRATEGI ALGORITMA  
PEMANFAATAN ALGORITMA GREEDY DALAM  
APLIKASI PERMAINAN “GALAXIO”**



Kelompok 2B1 Reuni:

Margaretha Olivia Haryono (13521071)

Austin Gabriel Pardosi (13521084)

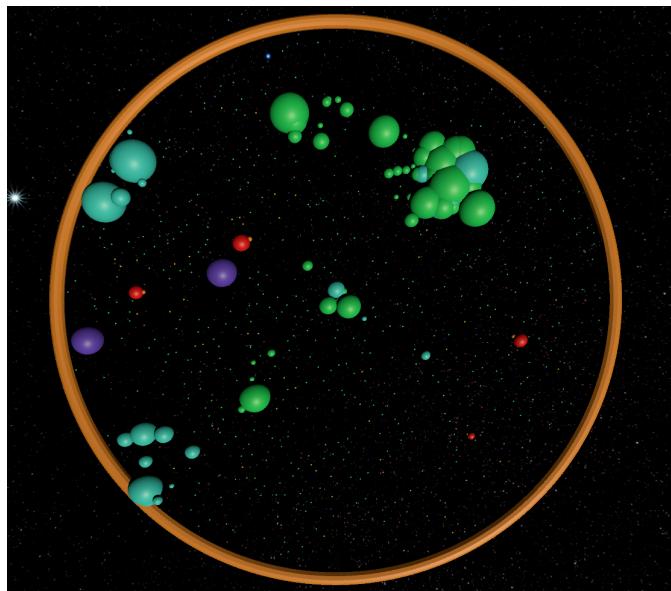
Michael Leon Putra Widhi (13521108)

**Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2023**

# BAB I

## DESKRIPSI TUGAS

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



**Gambar 1.** Ilustrasi permainan *Galaxio*

Bahasa pemrograman yang digunakan pada tugas besar ini adalah *Java*. Bahasa *Java* tersebut digunakan untuk membuat algoritma pada bot. *IDE* yang digunakan untuk membantu membuat projek ini adalah *IntelliJ IDEA*. *IntelliJ IDEA* merupakan *IDE* yang kompatibel dengan bahasa *Java*, dikarenakan beberapa tools-nya seperti *Maven* sudah built in tanpa perlu menambahkan *extension*. Untuk menjalankan permainan, digunakan sebuah *game engine* yang diciptakan oleh *Entelect Challenge* yang terdapat pada *repository githubnya*.

Spesifikasi permainan yang digunakan di dalam tugas besar ini disesuaikan dengan spesifikasi permainan “*Galaxio*” yang terdapat pada *repository Entelect Challenge*. Beberapa peraturan umum yang harus diikuti oleh bot adalah diantaranya :

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 maksimum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu *Players*, *Food*, *Wormholes*, *Gas Clouds*, *Asteroid Fields*. Berikut adalah penjelasan detail mengenai setiap jenis objek tersebut :

- a. *Players* : Objek ini merupakan pemain yang berada di dalam peta, antara lain pemain kami dan pemain lawan.
- b. *Food* : *Food* adalah objek kecil yang perlu dikumpulkan untuk “menumbuhkan” ukuran dari pemain.
- c. *Wormholes* : Objek ini berwujud dua buah titik pada peta yang saling terkoneksi di kedua belah sisi. *Wormholes* memungkinkan *players* untuk melakukan perpindahan tempat melalui kedua titik tersebut.
- d. *Gas Clouds* : Objek ini merupakan zona berbahaya yang akan melukai pemain saat dilintasi.
- e. *Asteroids Fields* : Objek ini merupakan zona berbahaya yang akan memperlambat pemain saat dilintasi.

Selain berbagai objek diatas, 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 *torpedo salvo* (ukuran 10) mengurangkan ukuran kapal sebanyak 5. Adapun sebuah kapal memiliki komponen sebagai berikut :

- a. Kecepatan : kapal akan bergerak  $x$  posisi ke depan setiap detik permainan, di mana  $x$  adalah kecepatan kapal. Kecepatan akan mulai dari 20 dan berkurang seiring pertumbuhan kapal.
- b. Ukuran - kapal akan mulai dengan radius 10.
- c. Tujuan - kapal akan bergerak ke arah ini, nilai berada diantara 0 dan 359 derajat.

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 melewatinya dengan syarat *wormhole* lebih besar dari kapal *player*.
5. Gas *Clouds* akan tersebar pada peta. Kapal dapat melewati gas *cloud*. Setiap kapal bertabrakan dengan gas *cloud*, ukuran dari kapal akan mengecil 1 setiap *tick* game. Saat kapal tidak lagi bertabrakan dengan gas *cloud*, maka efek pengurangan akan hilang.
6. *Torpedo Salvo* akan muncul pada peta yang berasal dari kapal lain. *Torpedo Salvo* berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. *Torpedo Salvo* dapat mengurangi ukuran kapal yang ditabraknya. *Torpedo Salvo* akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. *Supernova* merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. *Player* yang menembakkannya dapat meledakannya dan memberi damage ke *player* yang berada dalam zona. Area ledakan akan berubah menjadi gas *cloud*.
8. *Player* dapat meluncurkan *teleporter* pada suatu arah di peta. *Teleporter* tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. *Player* tersebut dapat berpindah ke tempat *teleporter* tersebut. Harga setiap peluncuran *teleporter* adalah 20. Setiap 100 *tick* *player* akan mendapatkan 1 *teleporter* dengan jumlah maximum adalah 10.
9. Ketika kapal *player* bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar

hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

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

- a. FORWARD : *Command* ini akan mulai menggerakkan bot pemain ke arah yang ditentukan dalam derajat.
- b. STOP : *Command* ini akan berhenti menggerakkan bot hingga perintah gerakan lain dikeluarkan. Tidak ada inersia, oleh karena itu kapal langsung berhenti.
- c. STARTAFTERBURNER : *Command* ini mengaktifkan *afterburner* bot. Perhatikan bahwa kecepatan hanya digunakan saat bot sedang bergerak ke suatu arah, oleh karena itu *afterburner* hanya akan berpengaruh saat pemain sedang bergerak. Biaya *afterburner* akan selalu berlaku jika efek ini aktif, yaitu mengurangi ukuran bot sebanyak 1 satuan setiap *tick*.
- d. STOPAFTERBURNER : *Command* ini menonaktifkan *afterburner* kapal pemain.
- e. FIRETORPEDOES : *Command* ini menggunakan 1 *salvo charge* dan ukuran 5 untuk mengirimkan *torpedo salvo* ke arah yang diberikan.
- f. FIRESUPERNOVA : *Command* ini menggunakan pengambil supernova untuk mengirimkan supernova ke pos yang diberikan.
- g. DETONATESUPERNOVA : *Command* ini meledakkan supernova yang ada
- h. FIRETELEPORTER : *Command* ini menghabiskan 1 biaya teleportasi dan 20 ukuran untuk mengirim *teleporter* ke arah yang diberikan
- i. TELEPORT : *Command* ini memindahkan pemain ke *teleporter* yang ada jika ada.
- j. ACTIVATESHIELD : *Command* ini mengaktifkan perisai (*shield*) pemain untuk membelokkan torpedo yang masuk jika memilikinya.

11. Pada permainan ini, terdapat pula mekanisme scoring dari setiap pemain. Mekanisme lengkap dari scoring adalah sebagai berikut:

- a. Skor akan disimpan untuk setiap pemain yang akan terlihat setelah pertandingan selesai.
- b. Mengkonsumsi kapal pemain lain = 10 poin
- c. Mengkonsumsi makanan = 1 poin

d. Melintasi lubang cacing = 1 poin

Perhatikan bahwa nilai ini hanya mewakili dan dapat berubah selama penyeimbangan.

12. 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

### A. Algoritma Greedy

*Greedy* merupakan sebuah teknik dalam strategi penyelesaian masalah, bukan suatu algoritma khusus. Teknik *greedy* biasanya memiliki waktu eksekusi yang cepat dan biasanya mudah untuk diimplementasikan, namun terkadang sulit dibuktikan kebenarannya.

Suatu persoalan dapat diselesaikan dengan teknik *greedy* jika persoalan tersebut memiliki sifat berikut : solusi optimal dari persoalan dapat ditentukan dari solusi optimal sub persoalan tersebut, dan pada setiap sub persoalan, ada suatu langkah yang bisa dilakukan yang mana langkah tersebut menghasilkan solusi optimal pada sub persoalan tersebut.

Elemen - elemen algoritma *greedy* :

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

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa Algoritma *greedy* melibatkan pencarian sebuah himpunan bagian (S), dari himpunan kandidat (C), yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimasi oleh fungsi objektif. Skema umum algoritma greedy menggunakan pseudocode dengan pendefinisian elemen/komponennya adalah sebagai berikut:

```

function greedy( $C : \text{himpunan\_kandidat}$ )  $\rightarrow \text{himpunan\_solusi}$ 
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
 $x : \text{kandidat}$ 
 $S : \text{himpunan\_solusi}$ 

Algoritma:
 $S \leftarrow \{\}$  {inisialisasi  $S$  dengan kosong}
while (not SOLUSI( $S$ )) and ( $C \neq \{\}$ ) do
     $x \leftarrow \text{SELEKSI}(C)$  {pilih sebuah kandidat dari  $C$ }
     $C \leftarrow C - \{x\}$  {buang  $x$  dari  $C$  karena sudah dipilih}
    if LAYAK( $S \cup \{x\}$ ) then { $x$  memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi}
         $S \leftarrow S \cup \{x\}$  {masukkan  $x$  ke dalam himpunan solusi}
    endif
endwhile
{/SOLUSI( $S$ ) or  $C = \{\}$ }

if SOLUSI( $S$ ) then {solusi sudah lengkap}
    return  $S$ 
else
    write('tidak ada solusi')
endif

```

**Gambar 2.1.** Skema Umum Algoritma *Greedy*

Pada akhir tiap iterasi, solusi yang terbentuk adalah optimum lokal, dan pada akhir kalang *while-do* akan ditemukan optimum global, namun optimum global yang ditemukan ini belum tentu merupakan solusi yang terbaik karena algoritma *greedy* tidak melakukan operasi secara menyeluruh kepada semua kemungkinan yang ada.

Kesimpulannya, algoritma *greedy* dapat digunakan untuk masalah yang hanya membutuhkan solusi hampiran dan tidak memerlukan solusi terbaik mutlak. Solusi ini terkadang lebih baik daripada algoritma yang menghasilkan solusi eksak dengan kebutuhan waktu yang eksponensial. Untuk contoh dari hal ini kita dapat melihat *Traveling Salesman Problem*, dimana penggunaan algoritma *greedy* akan jauh lebih cepat dibandingkan dengan penggunaan *brute force*, walaupun solusi yang ditemukan biasanya hanya hampiran dari solusi optimal.

## B. Garis Besar Cara Kerja Bot pada Permainan Galaxio

Cara kerja bot pada permainan Galaxio ini secara garis besar yaitu bot akan menerima data-data dari keadaan *object* yang terdapat di permainan pada *state* saat ini. Data-data tersebut yaitu:

- *id*: ID dari *object*.
- *size*: ukuran dari *object*.
- *speed*: kecepatan dari *object*.

- *currentHeading*: arah gerak dari *object*.
- *position*: posisi dari *object*. Posisi ini ditunjukkan dalam bentuk koordinat.
  - *x*: posisi x dalam koordinat kartesius.
  - *y*: posisi y dalam koordinat kartesius.
- *gameObjectType*: tipe dari *object* tersebut dalam bentuk integer. Terdapat 11 tipe dari *object* pada permainan ini, yaitu sebagai berikut:
  - 1 : *PLAYER*
  - 2 : *FOOD*
  - 3 : *WORMHOLE*
  - 4 : *GAS CLOUD*
  - 5 : *ASTEROID FIELD*
  - 6 : *TORPEDO SALVO*
  - 7 : *SUPERFOOD*
  - 8 : *SUPERNOVA PICKUP*
  - 9 : *SUPERNOVA BOMB*
  - 10: *TELEPORTER*
  - 11: *SHIELD*

*Object* yang bertipe *player* memiliki *command-command* yang dapat dilakukan selama keberjalanannya permainan, seperti yang sudah dijelaskan pada bab sebelumnya.

- *effects*: *flag* yang menentukan *effect* yang mengenai bot. Bit *flag* ini direpresentasikan sebagai berikut:
  - 0 = Tidak ada *effect*
  - 1 = *Afterburner* aktif
  - 2 = *Asteroid field*
  - 4 = *Gas cloud*

Setelah mengambil data-data tersebut, bot dapat mengetahui keadaan yang terjadi pada *state* permainan atau *tick* saat itu, sehingga bot akan mengeksekusi perintah yang sesuai dengan keadaan *tick* tersebut. Algoritma yang digunakan dalam penentuan perintah pada setiap keadaan atau *tick* adalah algoritma *greedy*.

## C. Implementasi Algoritma Greedy pada Permainan Galaxio

Dalam permainan ini, penulis mengimplementasikan beberapa algoritma *greedy* dalam pembuatan bot. Terdapat enam upa-strategi *greedy* serta satu algoritma *greedy* utama yang diimplementasikan. Keenam strategi *greedy* yang kami buat yaitu sebagai berikut.

1. Strategi *Greedy* untuk Memperoleh *Food* dan *SuperFood*

Strategi ini digunakan untuk melakukan implementasi skema pencarian makanan.

2. Strategi *Greedy* untuk Menggunakan *Torpedo Salvo*

Strategi ini digunakan untuk menembakkan *torpedo salvo* ketika ukuran bot cukup dan terdapat bot yang dekat sehingga dimungkinkan untuk menembakkan *torpedo salvo* terhadap bot lawan tersebut.

3. Strategi *Greedy* untuk Menggunakan *Teleporter*

Strategi ini digunakan untuk menembakkan *teleporter* ketika ukuran bot cukup dan terdapat bot yang dekat sehingga dimungkinkan untuk menembakkan *teleporter* terhadap bot lawan tersebut.

4. Strategi *Greedy* untuk Menghindari *Gas Clouds*

Strategi ini dijalankan untuk menghindari *gas clouds* yang terdapat dalam peta, sehingga ketika bot berjalan menuju *gas clouds*, bot akan membelokkan arah geraknya.

5. Strategi *Greedy* untuk Menghindari *Edge Map*

Ketika bot berada dekat dengan ujung (lingkaran peta), bot akan mengarahkan kapalnya ke arah tengah peta sehingga tidak keluar dari jalur peta.

6. Strategi *Greedy* untuk Menggunakan *Shield*

Strategi dalam penggunaan *shield* ini menangani dua buah kasus, yaitu *teleporter* serta *torpedo salvo*. Pada intinya kedua kasus ini menerapkan teknik yang sama, yaitu menggunakan hukum kinematika 1 dimensi untuk menyalakan *shield* perlindungan pada waktu yang tepat.

Seluruh strategi *greedy* yang dijelaskan di atas tersebut disatukan dalam algoritma *greedy* utama sesuai dengan tingkat prioritas masing-masing upa-strategi. Pada kondisi umum, bot akan melakukan *greedy by food* dan *greedy by enemy*. Ketika mendapat kasus tertentu pada suatu *state*, maka bot akan mengeksekusi algoritma yang bersesuaian.

## D. Garis Besar Game Engine pada Permainan Galaxio

Pada permainan Galaxio ini, *game engine* yang digunakan adalah Unity. *Game engine* ini digunakan untuk memvisualisasikan pergerakan bot dan keberjalanannya dalam permainan pada suatu pertandingan antar bot. Aplikasi ini sudah tersedia pada *repository starter-pack* yang diberikan oleh *Entelect* (penyelenggara permainan). *Repository* tersebut terdiri dari beberapa folder, yaitu:

- *engine-publish*
- *logger-publish*
- *reference-bot-publish*
- *runner-publish*
- *starter-bots*
- *visualizer*

Di dalam folder *engine-publish* dan *runner-publish*, terdapat file *appsettings.json* yang berisi semua pengaturan permainannya, salah satunya yaitu penentuan jumlah bot yang dapat ditandingkan dalam suatu permainan. Jumlah bot ini disesuaikan dengan jumlah bot yang ditandingkan (atau pada file *run.bat*).

Permainan dapat dilakukan dalam dapat dilakukan melalui *command line* ataupun menjalankan file *run.bat* (pada Windows) atau *run.sh* (pada Linux). File ini berisi *scripting language* untuk menjalankan permainan pada OS. File ini akan menjalankan *game engine* pada permainan Galaxio serta menghubungkan bot yang telah dibuat dengan sistem. Pada tugas ini, kami menggunakan bahasa Java dalam pembuatan bot, sehingga sebelum dapat dijalankan, bot harus *di-build* terlebih dahulu menggunakan *maven* sehingga dapat menghasilkan file *.jar* yang dapat dieksekusi.

Setelah permainan selesai, *log* dari permainan akan tersimpan secara otomatis pada folder *logger-publish*. Kemudian, pengguna dapat melihat visualisasinya pada aplikasi *Galaxio.exe* di dalam folder *visualizer*. Ketika menjalankan aplikasi untuk pertama kali, pengguna harus melakukan *load* terhadap *path* dari penyimpanan *log* permainan. Setelah itu, pengguna dapat memilih *log* permainan yang mana yang ingin dijalankan.

## BAB III

# APLIKASI STRATEGI GREEDY

### A. Pemetaan Elemen/Komponen Algoritma *Greedy* Pada Permainan Galaxio

Program ini terdiri dari beberapa menu yang dapat melakukan berbagai instruksi, diantaranya

#### 1. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *General Bot*

Dalam permainan Galaxio, tujuan setiap bot pemain berusaha untuk bertahan sampai akhir. Terdapat banyak cara untuk meraih hal tersebut, seperti berusaha membuat ukuran bot semakin besar, melakukan penyerangan terhadap bot lawan, menggunakan berbagai macam *playerAction* untuk tetap bertahan hingga akhir, dan lain-lain.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Seluruh permutasi <i>command</i> yang telah dipaparkan sebelumnya untuk setiap <i>tick</i> nya.
Himpunan solusi	Kemungkinan permutasi <i>command</i> yang memungkinkan bot bertahan sampai akhir.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command</i> tersebut dapat membuat bot bertahan sampai akhir.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas <i>command</i> yang harus diikuti. Bentuk fungsi heuristik tersebut mengikuti berbagai fungsi seleksi yang lebih kecil untuk setiap kasus pada <i>game state</i> . Tingkat prioritas tersebut bersifat statik selama sebuah <i>tick</i> permainan berlangsung.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Daftar <i>command</i> yang valid telah dipaparkan pada bab sebelumnya. Selain itu

	<i>command</i> harus dipastikan tidak membuat bot mati pada saat menjalankan <i>command</i> .
Fungsi objektif	Mencari permutasi dari <i>command</i> yang membuat bot pemain memenangkan permainan dengan jumlah ronde paling sedikit.

2. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Food and SuperFood Eating Mechanism*

Beberapa commands yang dijalankan pada permainan ini memerlukan ukuran dari bot. Oleh sebab itu, dalam permainan Galaxio, setiap bot harus memperbesar ukurannya agar dapat melakukan berbagai macam *actions* yang ada. Jenis makanan yang ada adalah *Food* dan *SuperFood* yang memiliki jenis kemampuan berbeda. Adapun cara pemain bot mendapatkan makanan adalah dengan “menabrakkan” dirinya pada makanan tersebut. Akan tetapi, jika ukuran bot terlalu besar, maka pergerakannya akan lebih lambat, sehingga ada suatu saat dimana bot tidak perlu mengambil makanan kembali.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Seluruh permutasi <i>command</i> memakan <i>Food</i> , memakan <i>SuperFood</i> , dan tidak memakan keduanya pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command</i> yang membuat bot memilih memakan <i>Food</i> , memakan <i>SuperFood</i> , dan tidak memakan keduanya sehingga memperbesar ukurannya secara optimal.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command</i> tersebut dapat dapat membuat bot memperbesar ukurannya secara optimal.
Fungsi seleksi	Memilih command berdasarkan data keadaan <i>game state</i>

	saat tersebut serta tingkat prioritas pemilihan jenis makanan dan aksi terhadap makanan yang harus dilakukan untuk membuat bot mencapai ukuran optimal.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat memperbesar bot secara optimal.
Fungsi objektif	Mencari permutasi dari <i>command</i> yang membuat bot pemain memperoleh ukuran yang paling optimal sehingga dapat menggunakan <i>command</i> lain.

### 3. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Torpedo Salvo Firing Mechanism*

Setiap pemain berkesempatan untuk meluncurkan *Torpedo Salvo* kepada lawannya untuk memberikan damage kepada setiap objek yang dilewatinya sebesar 10 satuan dan akan menambah ukuran bot penembak sebesar *damage* yang diterima oleh pemain lain. Penembakan torpedo salvo ini juga membawa konsekuensi memperkecil ukuran dari bot penembak sebesar 5 satuan dan hanya bisa dilakukan setiap 10 *ticks*. Penembakan *Torpedo Salvo* yang benar akan membuat strategi penyerangan yang efektif bagi lawan.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Seluruh permutasi <i>command FIRETORPEDO</i> dan <i>command</i> lainnya pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command FIRETORPEDO</i> dan <i>command</i> lainnya yang membuat bot dapat melakukan penyerangan menggunakan <i>Torpedo Salvo</i> secara optimal.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command</i>

	<i>FIRETORPEDO</i> dan <i>command</i> lainnya dapat membuat bot melakukan penyerangan menggunakan <i>Torpedo Salvo</i> secara optimal.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas pemilihan <i>command</i> <i>FIRETORPEDO</i> dan <i>command</i> lainnya yang harus dilakukan untuk membuat bot melakukan penyerangan menggunakan <i>Torpedo Salvo</i> secara optimal.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat membuat bot melakukan penyerangan menggunakan <i>Torpedo Salvo</i> secara optimal.
Fungsi objektif	Mencari permutasi dari <i>command</i> <i>FIRETORPEDO</i> dan <i>command</i> lainnya yang membuat bot melakukan penyerangan menggunakan <i>Torpedo Salvo</i> secara optimal.

#### 4. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Teleporter Firing Mechanism*

Setiap pemain berkesempatan untuk meluncurkan *teleporter* untuk melakukan teleportasi ke titik yang ditentukan. *Command* ini akan menjadi sangat berguna untuk melakukan perpindahan dari suatu titik ke titik lainnya secara cepat. *Command* yang digunakan untuk menembakkan *teleporter* adalah *FIRETELEPORTER* dan *command* yang digunakan untuk melakukan perpindahan tempat adalah *TELEPORT*. Meskipun demikian, *teleporter* akan menghabiskan cost sebesar 20 ukuran bot dan hanya bisa dilakukan 1 kali setiap 100 *ticks*.

<b>Nama Elemen/Komponen</b>	<b>Definisi Elemen/Komponen</b>
Himpunan kandidat	Seluruh permutasi <i>command FIRETELEPORTER</i> , <i>TELEPORT</i> , dan <i>command</i> lainnya pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command FIRETELEPORTER</i> , <i>TELEPORT</i> , dan <i>command</i> lainnya yang membuat bot dapat melakukan teleportasi secara optimal.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command FIRETELEPORTER</i> , <i>TELEPORT</i> , dan <i>command</i> lainnya dapat membuat bot melakukan teleportasi secara optimal.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas pemilihan <i>command FIRETELEPORTER</i> , <i>TELEPORT</i> , dan <i>command</i> lainnya yang harus dilakukan untuk membuat bot melakukan teleportasi secara optimal.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat membuat bot melakukan teleportasi secara optimal.
Fungsi objektif	Mencari permutasi dari <i>command FIRETELEPORTER</i> , <i>TELEPORT</i> , dan <i>command</i> lainnya yang membuat bot melakukan teleportasi secara optimal.

5. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Gas Clouds Avoiding Mechanism*

Sebuah peta yang telah tergenerasi memiliki sebuah objek bernama *gas clouds*. Objek ini merupakan objek bahaya yang membuat sebuah bot yang melewatinya

mengalami pengurangan ukuran sepanjang berada pada area tersebut. Inti strategi dari mekanisme ini adalah menghindari *Gas Clouds* se bisa mungkin sehingga pemain tidak mengalami pengurangan ukuran secara terus-menerus.

<b>Nama Elemen/Komponen</b>	<b>Definisi Elemen/Komponen</b>
Himpunan kandidat	Seluruh permutasi <i>command</i> yang akan menghindari <i>Gas Clouds</i> dan <i>command</i> lainnya pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command</i> yang akan menghindari <i>Gas Clouds</i> dan <i>command</i> lainnya yang membuat bot dapat menghindari <i>Gas Clouds</i> sejauh mungkin.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command</i> yang akan menghindari <i>Gas Clouds</i> dan <i>command</i> lainnya dapat membuat bot dapat menghindari <i>Gas Clouds</i> sejauh mungkin.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas pemilihan <i>command</i> yang akan menghindari <i>Gas Clouds</i> dan <i>command</i> lainnya yang harus dilakukan untuk membuat bot dapat menghindari <i>Gas Clouds</i> sejauh mungkin.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat membuat bot dapat menghindari <i>Gas Clouds</i> sejauh mungkin.
Fungsi objektif	Mencari permutasi dari <i>command</i> yang akan menghindari <i>Gas Clouds</i> dan <i>command</i> lainnya yang membuat bot dapat menghindari <i>Gas Clouds</i> sejauh mungkin.

6. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Edge Map Avoiding Mechanism*

Sebuah peta pasti memiliki batas dengan radius tertentu. Nilai dari radius ini akan terus berkurang sebanyak 1 satuan setiap *tick*. Pemain yang berada di luar batas peta akan terus mengalami pengurangan ukuran sebesar 1 satuan setiap *tick*. Mekanisme objektif ini digunakan untuk menghindari ujung dari peta sejauh mungkin sehingga ukuran dari pemain tidak terus mengalami penurunan.

<b>Nama Elemen/Komponen</b>	<b>Definisi Elemen/Komponen</b>
Himpunan kandidat	Seluruh permutasi <i>command</i> yang membuat pemain menghindari ujung dari peta pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command</i> yang membuat pemain menghindari ujung dari peta dan <i>command</i> lainnya yang membuat bot dapat menghindari ujung dari peta sejauh mungkin.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command</i> yang membuat pemain menghindari ujung dari peta dan <i>command</i> lainnya dapat membuat bot dapat menghindari ujung dari peta sejauh mungkin.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas pemilihan <i>command</i> yang membuat pemain menghindari ujung dari peta dan <i>command</i> lainnya yang harus dilakukan untuk membuat bot dapat menghindari ujung dari peta sejauh mungkin.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat membuat bot dapat menghindari ujung dari peta sejauh mungkin.

Fungsi objektif	Mencari permutasi dari <i>command</i> yang membuat pemain menghindari ujung dari peta dan <i>command</i> lainnya yang membuat bot dapat menghindari ujung dari peta sejauh mungkin.
-----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

7. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Self-protection by Shield Mechanism*

Salah satu cara yang dapat dilakukan untuk menghindari serangan dari musuh adalah dengan menggunakan *Shield*. Adapun *shield* ini akan membuat kita resisten dengan efek *Torpedo Salvo* dan *Teleporter* yang dilayangkan oleh lawan. Akan tetapi, *Shield* ini membuat setiap pemain yang menggunakannya harus membayar ukuran bot sebesar 20 satuan, sehingga Command *USESHIELD* ini harus dapat digunakan dengan baik sesuai dengan kebutuhannya.

<b>Nama Elemen/Komponen</b>	<b>Definisi Elemen/Komponen</b>
Himpunan kandidat	Seluruh permutasi <i>command USESHIELD</i> dan <i>command</i> lainnya pada suatu <i>tick</i> .
Himpunan solusi	Kemungkinan permutasi <i>command USESHIELD</i> dan <i>command</i> lainnya yang membuat bot dapat melakukan perlindungan diri dari serangan <i>Torpedo Salvo</i> dan <i>Teleporter</i> dari lawan secara optimal.
Fungsi solusi	Melakukan pengecekan apakah permutasi <i>command USESHIELD</i> dan <i>command</i> lainnya dapat membuat bot dapat melakukan perlindungan diri dari serangan <i>Torpedo Salvo</i> dan <i>Teleporter</i> dari lawan secara optimal.
Fungsi seleksi	Memilih <i>command</i> berdasarkan data keadaan <i>game state</i> saat tersebut serta tingkat prioritas pemilihan <i>command</i>

	<i>USESHIELD</i> dan <i>command</i> lainnya yang harus dilakukan untuk membuat bot dapat melakukan perlindungan diri dari serangan <i>Torpedo Salvo</i> dan <i>Teleporter</i> dari lawan secara optimal.
Fungsi kelayakan	Memeriksa apakah <i>command</i> yang dituliskan oleh bot merupakan <i>command</i> yang valid. Selain itu <i>command</i> harus dipastikan dapat membuat bot dapat melakukan perlindungan diri dari serangan <i>Torpedo Salvo</i> dan <i>Teleporter</i> dari lawan secara optimal.
Fungsi objektif	Mencari permutasi dari <i>command USESHIELD</i> dan <i>command</i> lainnya yang membuat bot dapat melakukan perlindungan diri dari serangan <i>Torpedo Salvo</i> dan <i>Teleporter</i> dari lawan secara optimal.

## B. Eksplorasi Alternatif Solusi *Greedy* pada Bot Permainan *Galaxio*

Pada dasarnya terdapat banyak alternatif solusi algoritma greedy yang dapat dilakukan pada permainan Galaxio ini. Hal ini dikarenakan terdapat banyak elemen dari *game engine* yang dapat diakses oleh bot dan kemudian diubah *state* pada elemen tersebut. Selain itu, elemen-elemen tersebut saling berinteraksi dengan elemen lainnya sehingga jika terdapat perubahan pada suatu *state* elemen, terdapat peluang bahwa *state* lainnya berubah pula. Banyaknya aksi yang mungkin dilakukan oleh bot membuat implementasi algoritma *greedy* pada permainan ini dapat dibuat dengan melihat *state* yang sedang dihadapi oleh game dan melakukan pemrosesan pemberian respons atau aksi yang harus dilakukan untuk menyikapi *state* yang sedang terjadi tersebut. Berikut adalah detail dari implementasi algoritma *greedy* yang sudah penulis buat.

### 1. Strategi *Greedy* untuk Memperoleh *Food* dan *SuperFood*

Seperti yang sudah disebutkan sebelumnya, setiap pemain memerlukan makanan untuk menambah ukuran dan menggunakan beberapa *command* yang menggunakan ukuran bot sebagai *charge* nya. Oleh sebab itu, strategi ini ditujukan untuk menentukan

langkah efektif dalam pemilihan *Food*, *SuperFood*, atau bukan keduanya sehingga bot dapat mengalami perbesaran ukuran secara optimal.

Sebelum mengimplementasikan algoritma, penulis membuat 2 buah senarai yaitu *FoodList* dan *SuperFoodList* dengan isi dari senarai ini adalah masing-masing objek tersebut diurutkan berdasarkan jarak terdekat dengan bot pemain, sehingga objek yang lebih dekat akan memiliki indeks yang paling kecil dari senarai tersebut. Berikut adalah implementasinya :

- a. Jika jarak antara bot ke *SuperFood* lebih besar 1.25 kali jarak antara bot dengan *Food*, maka bot akan memilih untuk memakan *SuperFood*. Tujuan dari dibuatnya pembobotan jarak ini adalah memaksimalkan perbesaran ukuran yang dihasilkan oleh bot setelah makan. *SuperFood* memiliki *multiplier* yang membuat ukuran bot yang memakannya akan lebih besar daripada *Food* biasa. Berikut adalah pembagian kasus kemungkinannya :
  - Jika jarak antara 2 buah *SuperFood* terdekat adalah sama, maka bot akan diarahkan untuk memilih *SuperFood* yang pertama muncul pada *SuperFoodList*.
  - Jika hanya ada 1 buah *SuperFood* terdekat, maka ambil *SuperFood* tersebut.
  - Jika ukuran dari bot dikali 5 sudah lebih besar dari radius peta, maka bot diarahkan untuk tidak mengambil makanan tersebut.
- b. Jika jarak antara bot ke *SuperFood* lebih kecil dari jarak antara bot dengan *Food* yang telah didefinisikan pada poin (a), maka bot akan memilih untuk memakan *Food*. Berikut adalah pembagian kasus kemungkinannya :
  - Jika jarak antara 2 buah *Food* terdekat adalah sama, maka bot akan diarahkan untuk memilih *Food* yang pertama muncul pada *FoodList*.
  - Jika hanya ada 1 buah *Food* terdekat, maka ambil *Food* tersebut.
  - Jika ukuran dari bot dikali 5 sudah lebih besar dari radius peta, maka bot diarahkan untuk tidak mengambil makanan tersebut.

## 2. Strategi *Greedy* untuk Menggunakan *Torpedo Salvo*

*Torpedo Salvo* merupakan salah satu komponen yang dapat dilayangkan oleh sebuah bot kepada bot lain untuk melakukan penyerangan. Objek ini akan melakukan pengurangan ukuran sebesar 10 satuan pada setiap objek yang dilaluinya, termasuk pemain lain, dan menambah ukuran penembak jika berhasil mengenai lawan. Strategi ini mengarahkan bot untuk melakukan penyerangan kepada lawan jika sudah mencapai standar ukuran yang cukup.

Sebelum mengimplementasikan algoritma, penulis membuat senarai yaitu *TorpedoSalvoList* dengan isi dari senarai ini adalah jarak dari setiap *Torpedo Salvo* ke pemain diurutkan berdasarkan jarak terdekat dengan bot pemain. Berikut adalah implementasinya :

- a. Jika bot lain terdekat memiliki ukuran yang lebih besar dari pemain, maka akan dilakukan skema penyerangan pertama, berikut adalah pembagian kasus kemungkinannya :
  - Jika ukuran dari bot minimal 50 satuan, maka bot akan menembak *Torpedo Salvo* melalui command *FIRETORPEDO*. Arah hadap dari bot akan diarahkan ke pemain tersebut sehingga memperbesar peluang hasil tembakan tepat sasaran.
  - Jika ukuran bot lebih kecil dari itu, maka bot akan diarahkan untuk menjauhi pemain terdekat tersebut dengan mengalikan arah hadapnya dengan -1 dan *actions forward* menjauhi pemain.
- b. Jika bot lain terdekat memiliki ukuran yang lebih kecil dari pemain, maka akan dilakukan skema penyerangan kedua, berikut adalah pembagian kasus kemungkinannya :
  - Jika ukuran dari bot minimal 50 satuan, maka bot akan menembak *Torpedo Salvo* melalui command *FIRETORPEDO*. Arah hadap dari bot akan diarahkan ke pemain tersebut sehingga memperbesar peluang hasil tembakan tepat sasaran.
  - Jika ukuran bot lebih kecil dari itu, maka bot akan diarahkan untuk mendekati pemain tersebut dan menjalankan mekanisme makan. Untuk menjelaskan, jika pemain yang lebih besar menabrakkan diri pada pemain

yang lebih kecil, maka pemain yang lebih besar akan “memakan” pemain yang lebih kecil maksimal sebesar ukurannya.

### 3. Strategi *Greedy* untuk Menggunakan *Teleporter*

Objek Teleporter memungkinkan setiap bot untuk melakukan teleportasi dari satu titik ke titik lainnya. Meskipun demikian, power up ini memerlukan charge sebesar 20 satuan ukuran dari bot, sehingga diperlukan sebuah strategi yang efektif dalam penggunaannya.

Penulis membuat senarai yaitu *TeleporterList* dengan isi dari senarai ini adalah jarak dari setiap *Teleporter* ke pemain diurutkan berdasarkan jarak terdekat dengan bot pemain. Berikut adalah implementasinya :

- a. Jika ukuran bot pemain dikurangi 20 bernilai lebih besar daripada bot pemain lain yang terdekat dan ukuran bot pemain lebih besar dari 50 satuan, maka pemain siap untuk menembak teleport dengan menjalankan command *FIRETELEPORTER*.
- b. Setelah teleport ditembakkan, akan dilakukan kalkulasi waktu tiba dari teleport menggunakan persamaan kinematika 1 dimensi. yaitu :

$$t = \frac{s}{v}$$

Dengan  $s$  adalah jarak tempuh teleporter, yaitu jarak bot dengan pemain terdekat,  $v$  adalah kecepatan dari teleporter, yaitu 20 satuan per tick, dan  $t$  adalah waktu tempuh yang diperlukan teleporter untuk tiba di tujuan. Proses pencatatan waktu dilakukan melalui perubahan radius pada saat teleport tiba di tujuan.

- c. Jika kondisi di atas tercapai, dilakukan pencarian teleporter yang mengarah pada target, karena pada dasarnya setiap pemain tidak dapat mengetahui pasti siapakah pemilik teleporter tersebut. Jika setelah pencarian kepemilikan teleporter selesai dilaksanakan dan masih memenuhi kondisi kinetika yang nilainya ditoleransi 4 detik lebih telat dan 4 detik lebih awal, maka pemain akan langsung melakukan teleport dengan command *TELEPORT*.

### 4. Strategi *Greedy* untuk Menghindari *Gas Clouds*

Pada sebuah peta yang telah tergenerasi memiliki sebuah objek bernama *Gas Clouds*. Objek ini merupakan objek bahaya yang membuat sebuah bot yang melewatinya mengalami pengurangan ukuran sepanjang berada pada area tersebut. Inti strategi dari mekanisme ini adalah menghindari *Gas Clouds* sebisa mungkin sehingga pemain tidak mengalami pengurangan ukuran secara terus-menerus.

Penulis membuat senarai yaitu *GasCloudList* dengan isi dari senarai ini adalah jarak dari setiap *Gas Clouds* ke pemain diurutkan berdasarkan jarak terdekat dengan bot pemain. Berikut adalah implementasinya :

- a. Langkah pertama pastikan *Gas Clouds* dalam peta, jika tidak ada, maka strategi ini kurang relevan dijalankan.
- b. Jika ada *Gas Clouds* dan jarak bot pemain dengan *Gas Clouds* terdekat kurang dari 0.8 ukurannya, maka pemain harus menghindar. Adapun cara melakukan penghindaran adalah membelokkan arah hadapnya  $90^\circ$  dari arah hadap ke *Gas Clouds*.

#### 5. Strategi *Greedy* untuk Menghindari *Edge Map*

Seperti yang sudah dijelaskan sebelumnya bahwa ukuran atau radius dari peta terus mengalami pengurangan sebesar 1 satuan setiap tick nya. Selain itu pemain mungkin saja mengambil makanan atau mengejar musuh dengan posisi yang mendekati ujung. Melalui berbagai kemungkinan kasus tersebut, diperlukan pengendali kasus yang akan diimplementasikan pada bagian ini. Berikut adalah implementasinya :

- a. Cek posisi dari bot pemain. Pemain dikatakan dekat dengan ujung jika jarak ke tengah/pusat peta lebih besar dari 0.75 radius peta.
- b. Jika kondisi (a) terpenuhi, maka pemain akan menjauhi ujung peta dengan mengubah *heading* ke pusat peta dengan *actions forward*. Akan tetapi, jika kondisi tersebut tidak terpenuhi, maka bot akan menjalankan strategi *Food and SuperFood eating mechanism* atau mekanisme *Enemy Chasing*.

#### 6. Strategi *Greedy* untuk Menggunakan *Shield*

Setiap pemain diberi kesempatan untuk menggunakan Shield dari serangan lawan. Adapun shield dapat digunakan untuk “memantulkan” serangan *Torpedo Salvo* atau

*Teleporter* milik lawan. Karena penggunaan Shield ini memerlukan cost yaitu bot size dalam jumlah yang besar, maka diperlukan sebuah strategi yang optimal untuk menangani kasus ini.

Mekanisme penggunaan Shield ini dibagi menjadi dua buah kasus. Yaitu berdasarkan *Teleporter* lawan maupun *Torpedo Salvo* lawan. Berikut adalah implementasinya :

a. *Shield by Teleporter*

Pada bagian ini shield akan dinyalakan jika ada *Teleporter* yang dekat dengan pemain. Posisi dekat dinyatakan dengan arah heading dari *Teleporter* ke pemain yang ditoleransi  $20^\circ$  ke atas dan  $20^\circ$  ke bawah. Jika kondisi tersebut terpenuhi, maka akan dilakukan proses kalkulasi waktu *Teleporter* tiba dengan persamaan Kinematika 1 Dimensi yang sudah dijelaskan sebelumnya dengan toleransi sebesar 3 detik sebelum dan 3 detik setelah prediksi. Jika memenuhi kondisi tersebut, maka *Shield* akan dinyalakan untuk melakukan perlindungan.

b. *Shield by Torpedo Salvo*

Sama seperti metode sebelumnya, pada bagian ini shield akan dinyalakan jika ada *Torpedo Salvo* yang dekat dengan pemain. Posisi dekat dinyatakan dengan arah heading dari *Torpedo Salvo* ke pemain yang ditoleransi  $20^\circ$  ke atas dan  $20^\circ$  ke bawah. Jika kondisi tersebut terpenuhi, maka akan dilakukan proses kalkulasi waktu *Torpedo Salvo* tiba dengan persamaan Kinematika 1 Dimensi yang sudah dijelaskan sebelumnya dengan toleransi sebesar 3 detik sebelum dan 3 detik setelah prediksi. Jika memenuhi kondisi tersebut, maka *Shield* akan dinyalakan untuk melakukan perlindungan.

## C. Analisis Efisiensi dari Kumpulan Solusi Algoritma *Greedy*

Pada permainan Galaxio, banyak *gamestate* yang bisa kita ketahui dengan mudah seperti *state* pemain, *state* lawan, senarai objek lain dalam peta, dan lain-lain. Berbagai kondisi tersebut membuat penulis mampu untuk mengimplementasikan algoritma *greedy* yang tepat berdasarkan kumpulan kondisi tersebut. Adapun keterbatasan pemrosesan aksi yang hanya dapat melakukan 1 aksi setiap *tick* dan banyaknya state yang mungkin terjadi

pada *tick* yang sama seringkali membuat bot tidak dapat menangani sebuah kasus kompleks secara bersamaan.

Pada *Food and SuperFood Eating Mechanism*, kita cukup memperhatikan kedekatan posisi antara Food dan SuperFood terdekat lalu membandingkannya. Dengan tidak memperhitungkan proses penyaringan senarai *FoodList* dan *SuperFoodList*, maka kompleksitas dari strategi ini adalah  $O(1)$ .

Pada bagian *Enemy Chasing* yang terdiri dari *Torpedo Salvo Firing Mechanism* dan *Teleporter Firing Mechanism* terdapat pengaksesan senarai yang digunakan untuk melakukan pengecekan kondisi, posisi, dan *state* dari lawan. Perkiraan kompleksitas waktu untuk strategi ini adalah  $O(n)$  dengan  $n$  adalah jumlah pemain lain yang ada di dalam peta.

Pada bagian *Avoidation Scheme* yang terdiri dari *Gas Clouds Avoiding Mechanism* dan *Edge Map Avoiding Mechanism*, strategi dapat dilakukan secara lempang (*straight-forward*) dengan mengetahui *state* dari *game*. Perkiraan kompleksitas waktu dari strategi ini adalah  $O(1)$ .

Terakhir, ada *Self-protection by Shield Mechanism* yang memerlukan dua buah senarai, yaitu *TeleporterList* dan *TorpedoSalvoList* yang membuat harus melakukan penyelidikan terhadap isi senarai untuk mendapatkan apakah terdapat *Teleporter* atau *Torpedo Salvo* yang dekat dan mengarah ke pemain. Oleh sebab itu, kompleksitas waktu dari strategi ini adalah  $O(n)$  dengan  $n$  adalah jumlah *Teleporter* maupun *Torpedo Salvo* bergantung pada senarai yang bersangkutan.

## D. Analisis Efektivitas dari Kumpulan Solusi Algoritma *Greedy*

Strategi *Food and SuperFood Eating Mechanism* secara umum cukup efektif untuk diimplementasikan. Proses pengimplementasian strategi ini sempat mengalami sedikit kendala seperti bot yang mengalami kebingungan dalam memilih 2 buah makanan yang memiliki jarak yang sama, sehingga diperlukan penanganan yang lebih efektif terhadap kasus ujung serupa.

Strategi *Enemy Chasing* yang terdiri dari *Torpedo Salvo Firing Mechanism* dan *Teleporter Firing Mechanism* merupakan salah satu strategi utama yang ada pada seluruh strategi yang disebutkan. Strategi ini menurut penulis sudah cukup efektif meskipun keadaan

penembakan teleporter relatif sulit untuk dijumpai karena proses penelusuran senarai menggunakan kalang yang relatif lama.

Strategi *Avoidation Scheme* yang terdiri dari *Gas Clouds Avoiding Mechanism* dan *Edge Map Avoiding Mechanism* direncanakan dengan cukup baik, walaupun di beberapa kasus pada saat jumlah makanan pada radius tertentu sudah habis, bot memilih untuk keluar, tetapi kasus tersebut relatif jarang untuk ditemui. Hal tersebut bisa terjadi karena bot tidak menerima *state* aksi yang perlu dieksekusi jika perancangan kemungkinan kasus kurang matang.

Terakhir, ada *Self-protection by Shield Mechanism* yang berjalan dengan sangat baik. Strategi ini berhasil membuat bot tidak mengalami pengurangan ukuran yang signifikan walaupun dikenai *Torpedo Salvo* dan didekati dengan *Teleporter* lawan.

## E. Strategi *Greedy* yang Digunakan pada Bot

Strategi yang penulis ambil sebagai algoritma *greedy* utama dari program bot permainan Galaxio ini adalah penggabungan dari seluruh strategi yang telah dijelaskan secara detail sebelumnya. Penulis mengambil seluruh strategi untuk setiap mekanisme pada permainan *Galaxio* agar program bot dapat menangani seluruh kemungkinan kasus dari game state secara efektif dan tepat sasaran.

Permasalahan yang berikutnya timbul adalah cara menyatukan keseluruhan strategi yang telah didefinisikan tersebut. Agar seluruh strategi dapat digabungkan menjadi suatu algoritma *greedy* utama, penulis harus mendefinisikan secara jelas urutan prioritas eksekusi strategi. Adapun cara mendefinisikan prioritas adalah dengan menggunakan strategi heuristik dan prediksi *state*, sehingga bot memiliki kebebasan untuk menentukan state yang akan dieksekusi terlebih dahulu. Untuk menghindari saling-timpal antar aksi, maka aksi dengan prioritas paling tinggi akan berada pada posisi paling bawah.

Setelah melakukan banyak percobaan, pemikiran, dan perencanaan yang matang mengenai aksi yang perlu dieksekusi, penulis berhasil melakukan formulasi penentuan prioritas strategi yang akan dieksekusi dan permutasi *command* yang dapat membuat bot pemain memenangkan permainan.

Berikut adalah urutan formulasi prioritas eksekusi strategi :

1. *Edge Map Avoiding Mechanism*

2. *Food and SuperFood Eating Mechanism* dan *Torpedo Salvo Firing Mechanism* yang dijalankan secara bergantian saat terpicu oleh kondisi zona serang dan ukuran bot.
3. *Teleporter Firing Mechanism*
4. *Self-protection by Shield Mechanism*
5. *Gas Clouds Avoiding Mechanism*

## BAB IV

# IMPLEMENTASI DAN PENGUJIAN

Implementasi program yang kami buat dapat dilihat pada halaman berikut ini

[https://github.com/margarethaolivia/Tubes1\\_2B1Reuni](https://github.com/margarethaolivia/Tubes1_2B1Reuni)

### A. Pseudocode

{File BotService}

{ Prosedur aksi utama yang dipanggil di Main }

**procedure** computeNextPlayer( PlayerAction: playerAction)

#### KAMUS LOKAL

count, add1, add2, add3, add4 = integer

radiusPeta, jumlahPlayerTotal, vtelp, init, finpos = int

jarak, finals= real

#### ALGORITMA

{Inisiasi data yang diperlukan}

count <- 0

add1 <- 0

add2 <- 0

add3 <- 0

add4 <- 0

**if** (not gameState.getGameObject().isEmpty()) **then**

{Mendefinisikan data-data penting}

foodList <- gameState.getGameObject().stream().filter(item->item.getGameObjectType() ==  
ObjectTypes.FOOD).sorted(Comparator.comparing(item->getDistanceBetween(bot,item))).collect(Collectors.toLi  
st())

superFoodList <- gameState.getGameObject().stream().filter(item->item.getGameObjectType() ==  
ObjectTypes.SUPERFOOD).sorted(Comparator.comparing(item -> getDistanceBetween(bot,  
item))).collect(Collectors.toList())

```

otherPlayerList <- gameState.getPlayerGameObject().stream().filter(item -> item.getId() != bot.id).sorted(Comparator.comparing(item -> getDistanceBetween(bot, item))).collect(Collectors.toList())

gasCloudList <-
gameState.getPlayerGameObject().stream().filter(item->item.getGameObjectType()==ObjectTypes.GAS_CLOUD).sorted(Comparator.comparing(item->getDistanceBetween(bot, item))).collect(Collectors.toList());=

otherPlayerListSize <- gameState.getPlayerGameObject().stream().filter(item->item.getId() != bot.id).sorted(Comparator.comparing(item->getSizeBetween(bot,item))).collect(Collectors.toList())

teleporterList <- gameState.getPlayerGameObject().stream().filter(item->item.getGameObjectType() == ObjectTypes.TELEPORTER).sorted(Comparator.comparing(item -> getDistanceBetween(bot, item))).collect(Collectors.toList())

torpedoSalvoList <-
gameState.getPlayerGameObject().stream().filter(item->item.getGameObjectType()==ObjectTypes.TORPEDO_SALVO).sorted(Comparator.comparing(item -> getDistanceBetween(bot, item))).collect(Collectors.toList())

```

*{Implementasi Greedy}*

```

radiusPeta <- getGameState().getWorld().radius
jumlahPlayerTotal <- otherPlayerList.size() + 1
pemainLainTerdekat <- otherPlayerList.get(0)
pemainKecilLainTerdekat <- otherPlayerListSize.get(0)
superFoodTerdekat <- superFoodList.get(0)
superFoodTerdekat2 <- superFoodList.get(1)
foodTerdekat <- foodList.get(0)
foodTerdekat2 <- foodList.get(1)

if (not otherPlayerList.isEmpty()) then
{Menghandle Posisi Ketika di Ujung Peta}
if (getDistanceToPoint(centralMap) > 0.75*(getGameState().getWorld().radius)) then
    playerAction.heading <- getHeadingToPoint(centralMap);
    playerAction.action <- PlayerActions.FORWARD;
else
{Greedy I}
    if (zonaTembakMusuh(pemainLainTerdekat)&&masihKecil()) then

```

```

makan.cariMakan(bot, playerAction, radiusPeta, superFoodTerdekat, superFoodTerdekat2,
foodTerdekat, foodTerdekat2)

    count <- count + 1

else if (not zonaTembakMusuh(pemainLainTerdekat)&& masihKecil()) then

    makan.cariMakan(bot, playerAction, radiusPeta, superFoodTerdekat, superFoodTerdekat2,
foodTerdekat, foodTerdekat2)

        count <- count + 2;

else if (zonaTembakMusuh(pemainLainTerdekat) && not masihKecil()) then

    kejarmusuh.ketemuMusuh(bot, playerAction, pemainLainTerdekat)

        count <- count + 3

else

    playerAction.heading = getHeadingBetween(bot, pemainLainTerdekat)

    playerAction.action = PlayerActions.FORWARD;

    count <- count + 4

{Greedy 2}

if (not teleporterList.isEmpty()) then

    if ((teleporterList.get(0).currentHeading - getHeadingBetween(teleporterList.get(0),
bot)) >= -20 && (teleporterList.get(0).currentHeading - getHeadingBetween(teleporterList.get(0), bot)) <= 20)
then

        add1 <- add1 + 1

        jarak <- getDistanceBetween(bot, teleporterList.get(0))

        vtelp <- 20

        init <- radiusPeta

        finals <- init - (jarak / vtelp)

        finpos <- (int) Math.round(finals)

        if ((radiusPeta - finpos >= -3) && (radiusPeta - finpos <= 3)) then

            add1 <- add1 + 2;

            playerAction.action <- PlayerActions.ACTIVATESHIELD

{Greedy 3}

if (bot.getSize() - 20 > pemainKecilLainTerdekat.getSize() && bot.getSize() >= 80) then

    add2 <- add2 + 1;

    playerAction.heading <- getHeadingBetween(bot, pemainKecilLainTerdekat)

    playerAction.action <- PlayerActions.FIRETELEPORT

    jarak <- getDistanceBetween(bot, pemainKecilLainTerdekat)

```

```

vtelp < 20;
init <- radiusPeta;
finals <- init - (jarak / vtelp);
finpos <- (int) Math.round(finals);

if ((radiusPeta - finpos >= -4) && (radiusPeta - finpos <= 4)) then
    for (i traversal [0..teleporterList.size()])
        if ((teleporterList.get(i) .currentHeading-getHeadingBetween(teleporterList.get(i),
pemainKecilLainTerdekat)) >= -30 && (teleporterList.get(i)
.currentHeading-getHeadingBetween(teleporterList.get(i), pemainKecilLainTerdekat)) <= 30) then
            add2 <- add2 + 2;
            playerAction.action <- PlayerActions.TELEPORT
            break

```

{Greedy 4}

```

if (not torpedoSalvoList.isEmpty()) then
    if ((torpedoSalvoList.get(0).currentHeading - getHeadingBetween(torpedoSalvoList.get(0), bot)) >= -20
&& (torpedoSalvoList.get(0).currentHeading - getHeadingBetween(torpedoSalvoList.get(0), bot)) <= 20) then
        add3 <- add3 + 1
        jarak <- getDistanceBetween(bot, torpedoSalvoList.get(0))
        vtor <- 60
        init <- radiusPeta
        finals <- init - (jarak / vtor)
        finpos <- (int) Math.round(finals)
        if ((radiusPeta - finpos >= -3) && (radiusPeta - finpos <= 3)) then
            add3 <- add3 + 2
            playerAction.action <- PlayerActions.ACTIVATESHIELD

```

{Greedy 5}

```

if (not gasCloudList.isEmpty()) then
    add4 <- add4 + 1
    if (getDistanceBetween(bot, gasCloudList.get(0)) <= 0.8 * gasCloudList.get(0).getSize()) then
        add4 <- add4 + 2
        playerAction.heading <- (getHeadingBetween(bot, gasCloudList.get(0)) + 90) % 360
        playerAction.action <- PlayerActions.FORWARD
else

```

```
    output("MENANG!!")  
  
    printState(jumlahplayerTotal, count, bot, playerAction, add1, add2, add3, add4)  
  
    this.playerAction <- playerAction;  
  
{ Fungsi yang mendefinisikan zona tembak musuh }  
function zonaTembakMusuh(pemainLainTerdekat : GameObject) -> boolean
```

#### KAMUS LOKAL

-

### ALGORITMA

```
-> getDistanceBetween(bot, pemainLainTerdekat) <= (getGameState().getWorld().radius) / 1.25
```

```
{ Fungsi yang mendefinisikan ukuran bot }
```

```
function masihKecil() -> boolean
```

#### KAMUS LOKAL

min : integer

### ALGORITMA

```
if (40 < (getGameState().getWorld().radius) / 14) then
```

    min <- 40

else

    min <- (getGameState().getWorld().radius) / 14

```
-> (bot.getSize() < min)
```

```
{ Fungsi yang mengambil besarnya jarak antara dua objek }
```

```
function getDistanceBetween(object1 : GameObject , object2 : GameObject ) -> double
```

#### KAMUS LOKAL

triangleX, triangleY : double

### ALGORITMA

```
triangleX <- Math.abs(object1.getPosition().x - object2.getPosition().x)
```

```
triangleY = <-Math.abs(object1.getPosition().y - object2.getPosition().y)
```

```
->Math.sqrt(triangleX * triangleX + triangleY * triangleY)
```

{ Fungsi yang mengambil besarnya jarak antara objek(bot) dengan titik }

```
function getDistanceToPoint(target : Position) -> double
```

#### KAMUS LOKAL

triangleX, triangleY : double

#### ALGORITMA

```
triangleX <- Math.abs(bot.getPosition().x - target.x)
```

```
triangleY <- Math.abs(bot.getPosition().y - target.y)
```

```
-> Math.sqrt(triangleX * triangleX + triangleY * triangleY)
```

{ Fungsi yang mengambil arah yang diberikan objek bot ke otherObject }

```
function getHeadingBetween(otherObject1 : GameObject, otherObject2 : GameObject) -> integer
```

#### KAMUS LOKAL

direction : integer

#### ALGORITMA

```
direction <- toDegrees(Math.atan2(otherObject2.getPosition().y - otherObject1.getPosition().y,  
otherObject2.getPosition().x - otherObject1.getPosition().x))
```

```
-> (direction + 360) % 360
```

{ Fungsi yang mencari perbedaan size bot 1 dengan lainnya }

```
function getSizeBetween(object1 : GameObject, object2 : GameObject) -> integer {
```

#### KAMUS LOKAL

size1, size2, result : integer

#### ALGORITMA

```
size1 <- object1.getSize()
```

```
size2 <- object2.getSize()
```

```
result <- 0
```

```
if (size1 > size2) then
```

```
    result <- size1 - size2
```

```

else
    result <- size2 - size1

-> result

```

{ Fungsi yang mengambil arah yang diberikan objek bot ke sebuah titik }

```
function getHeadingToPoint(target : Position) -> integer
```

### KAMUS LOKAL

```
direction : integer
```

### ALGORITMA

```
direction <- toDegrees(Math.atan2(target.y - bot.getPosition().y, target.x - bot.getPosition().x))
-> (direction + 360) % 360
```

{ Fungsi yang melakukan konversi dari radian }

```
function toDegrees(v : real) -> integer
-> (int) (v * (180 / Math.PI))
```

{ Prosedur yang melakukan pencetakan state dari game yang diperlukan }

```
procedure printState(jumlahplayerTotal : integer, flag : integer, bot : GameObject, playerAction :
PlayerAction, add1 : integer, add2 : integer, add3 : integer, add4 : integer)
```

### KAMUS LOKAL

```
-
```

### ALGORITMA

```
if (getGameState().getWorld().radius != null && jumlahplayerTotal != 0) then
    output ("====")
    output ("World Radius : %d\n",getGameState().getWorld().radius)
    output ("Bot Size : %d\n",bot.size)
    output ("Positions : [%d, %d]\n",bot.getPosition().x, bot.getPosition().y)
    output ("Enemy Count : %d\n",jumlahplayerTotal - 1)
```

```

if (flag == 0) then
    output ("Strategy : (0) Menghindari Ujung")
else if (flag == 1) then
    output ("Strategy : (1) Strategi Jelajah - Masih Kecil")
else if (flag == 2) then
    output ("Strategy : (2) Strategi Jelajah - Bukan Zona Tembak")
else if (flag == 3) then
    output ("Strategy : (3) Strategi Serang - Sudah Matang")
else if (flag == 4) then
    output ("Strategy : (4) Strategi Serang - Menuju Zona Tembak")

output ("Action : %s\n", playerAction.action);
output ("Heading : %d\n", playerAction.heading);
output ("AddCons Flag : %d %d %d %d\n", add1, add2, add3, add4);
output ("");

```

{ File Makan.java }

{ Fungsi untuk mengambil arah yang diberikan objek bot ke otherObject }

**function** getHeadingBetween(bot: GameObject, otherObject: GameObject) -> integer

#### KAMUS LOKAL

direction : integer

#### ALGORITMA

direction <- toDegrees(Math.atan2(otherObject.getPosition().y - bot.getPosition().y,  
otherObject.getPosition().x - bot.getPosition().x))  
-> (direction + 360) mod 360

{ Fungsi yang mengambil besarnya jarak antara dua objek }

**function** getDistanceBetween(object1: GameObject, object2: GameObject) -> real

#### KAMUS LOKAL

triangleX, triangleY : real

## ALGORITMA

```
triangleX <- Math.abs(object1.getPosition().x - object2.getPosition().x)
triangleY <- Math.abs(object1.getPosition().y - object2.getPosition().y)
-> Math.sqrt(triangleX * triangleX + triangleY * triangleY)
```

{ Fungsi yang melakukan konversi dari radian menjadi derajat }

```
function toDegrees(v: real) -> integer
```

## KAMUS LOKAL

-

## ALGORITMA

```
-> (int) (v * (180 / Math.PI))
```

{ Prosedur aksi yang dilakukan pemain bot mencari makan }

```
procedure cariMakan(bot: GameObject, PlayerAction: playerAction, radiusPeta: integer, superFoodTerdekat: GameObject, superFoodTerdekat2: GameObject, foodTerdekat: GameObject, foodTerdekat2: GameObject)
```

## KAMUS LOKAL

-

## ALGORITMA

{ Jika jarak ke superfood > 1.25 \* food dengan pembobotan, atur makan foodlist }

```
if (getDistanceBetween(bot, superFoodTerdekat) > 1.25 * getDistanceBetween(bot, foodTerdekat)) then
```

{ Jika ukuran kapal udah gede (1/5 radius peta), jangan makan lagi }

```
if (5 * bot.size >= radiusPeta) then { do nothing }
```

{ Jika jarak ke 2 makanan berbeda itu sama, pilih indeks terkecil }

```
else if (getDistanceBetween(bot, foodTerdekat) == getDistanceBetween(bot, foodTerdekat2)) then
```

```
    playerAction.heading <- getHeadingBetween(bot, foodTerdekat)
```

```
    playerAction.action <- PlayerActions.FORWARD
```

{ Kasus normal, cukup ambil makanan terdekat }

```
else
```

```
    playerAction.heading <- getHeadingBetween(bot, foodTerdekat)
```

```
    playerAction.action <- PlayerActions.FORWARD
```

{ Jika jarak ke superfood <= 1.25 \* food dengan pembobotan, atur makan superFoodlist }

```
else
```

```

{ Jika ukuran kapal udah gede (1/5 radius peta), jangan makan lagi }

if (5 * bot.size >= radiusPeta) then { do nothing }

{ Jika jarak ke 2 makanan berbeda itu sama, pilih indeks terkecil }

else if (getDistanceBetween(bot, superFoodTerdekat) == getDistanceBetween(bot,
superFoodTerdekat2)) then

    playerAction.heading <- getHeadingBetween(bot, superFoodTerdekat)

    playerAction.action <- PlayerActions.FORWARD

{ Kasus normal, cukup ambil makanan terdekat }

else

    playerAction.heading <- getHeadingBetween(bot, superFoodTerdekat)

    playerAction.action <- PlayerActions.FORWARD

```

```

{ File KejarMusuh.java }

function getHeadingBetween(bot: GameObject, otherObject: GameObject) -> integer

KAMUS LOKAL

direction : integer

ALGORITMA

direction <- toDegrees(Math.atan2(otherObject.getPosition().y - bot.getPosition().y,
otherObject.getPosition().x - bot.getPosition().x))

-> (direction + 360) % 360

{ Fungsi yang melakukan konversi dari radian menjadi derajat }

function toDegrees(v: real) -> integer

-> (int)(v * (180 / Math.PI))

{ Prosedur aksi yang dilakukan pemain saat bertemu pemain lawan }

procedure ketemuMusuh(bot: GameObject, PlayerAction: playerAction, pemainLainTerdekat: GameObject)

KAMUS LOKAL

-

ALGORITMA

{ Kalo musuh lebih besar }

```

```

if ((pemainLainTerdekat.getSize() >= bot.getSize())) then
    { Kalo ukuran bot diatas 50, artinya bot bisa nembak }
    if (bot.getSize() >= 50) then
        playerAction.action <- PlayerActions.FIRETORPEDOES
        { Kalo ga sampe 50, mending kabur ke arah belawan si bot lawan }
    else
        playerAction.action <- PlayerActions.FORWARD
    { Kalo musuh lebih kecil }
    else
        { Kalo ukuran bot diatas 50, artinya bot bisa nembak }
        if (bot.getSize() >= 50) then
            playerAction.action <- PlayerActions.FIRETORPEDOES
            { Kalo ga sampe 50, setidaknya kita lebih besar buat makan dia, arahkan ke dekatnya }
        else
            playerAction.action <- PlayerActions.FORWARD

```

## B. Struktur Data

Struktur data pada permainan Galaxio ini terdiri dari *class-class* yang terbagi dalam beberapa file. Terdapat 3 folder besar dalam pembuatan bot ini serta satu file *Main.java*. Ketiga folder tersebut yaitu:

### 1. *Enums*

Folder *Enums* berisi daftar objek-objek yang menjadi bagian dalam permainan serta daftar aksi yang dapat dilakukan oleh pemain. Berikut *file* yang terdapat dalam folder ini.

#### 1. *ObjectTypes.java*

File ini berisi kelas bernama *ObjectTypes* yang memiliki atribut *value* untuk merepresentasikan tipe dari *object* tersebut. Kelas ini memiliki *method getter* untuk mengambil nilai dari *value* tersebut.

#### 2. *PlayerActions.java*

File ini berisi kelas *PlayerActions* yang memiliki atribut *value* untuk merepresentasikan aksi dari *player* atau pemain. Kelas ini memiliki *method getter* untuk mengambil nilai dari *value* tersebut.

## 2. Models

Folder *Models* berisi pendefinisian model-model objek, posisi, peta, serta aksi yang valid dalam permainan. Berikut *file* yang terdapat dalam folder ini.

### 1. GameObject.java

File ini berisi kelas bernama *GameObject* untuk memberikan deskripsi akan sebuah *object*.

#### a. Atribut

- public UUID id
- public Integer size
- public Integer speed
- public Integer currentHeading
- public Position position
- public ObjectTypes gameObjectType
- public Integer effects
- public Integer torpedoSalvoCount
- public Integer superNovaAvailable
- public Integer teleporterCount
- public Integer shieldCount

#### b. Method

- public UUID getId()
- public void setId(UUID id)
- public int getSize()
- public void setSize(int size)
- public int getSpeed()
- public void setSpeed(int speed)
- public Position getPosition()
- public void setPosition(Position position)
- public ObjectTypes getGameObjectType()
- public void setGameObjectType(ObjectTypes gameObjectType)
- public static GameObject FromStateList(UUID id, List<Integer> stateList)

## 2. GameState.java

File ini berisi kelas bernama *GameState* untuk memberikan deskripsi akan sebuah keadaan *state* saat ini.

### a. Atribut

- public World world
- public List<GameObject> gameObjects
- public List<GameObject> playerGameObjects

### b. Method

- public GameState()
- public GameState(World world , List<GameObject> gameObjects, List<GameObject> playerGameObjects)
- public World getWorld()
- public void setWorld(World world)
- public List<GameObject> getGameObjects()
- public void setGameObjects(List<GameObject> gameObjects)
- public List<GameObject> getPlayerGameObjects()
- public void setPlayerGameObjects(List<GameObject> playerGameObjects)

## 3. GameStateDto.java

File ini berisi kelas bernama *GameStateDto* untuk men-generate kondisi dari permainan.

### a. Atribut

- private World world
- private Map<String, List<Integer>> gameObjects
- private Map<String, List<Integer>> playerObjects

### b. Method

- public Models.World getWorld()
- public void setWorld(Models.World world)
- public Map<String, List<Integer>> getGameObjects()
- public void setGameObjects(Map<String, List<Integer>> gameObjects)
- public Map<String, List<Integer>> getPlayerObjects()

- public void setPlayerObjects(Map<String, List<Integer>> playerObjects)

#### 4. PlayerAction.java

File ini berisi kelas bernama *PlayerAction* untuk menentukan aksi *player* atau pemain.

##### a. Atribut

- public UUID playerId
- public PlayerActions action
- public int heading

##### b. Method

- public UUID getPlayerId()
- public void setPlayerId(UUID playerId)
- public PlayerActions getAction()
- public void setAction(PlayerActions action)
- public int getHeading()
- public void setHeading(int heading)

#### 5. Position.java

File ini berisi kelas bernama *Position* untuk merepresentasikan suatu posisi tertentu dalam permainan.

##### a. Atribut

- public int x
- public int y

##### b. Method

- public Position()
- public Position(int x, int y)
- public int getX()
- public void setX(int x)
- public int getY()
- public void setY(int y)

#### 6. World.java

File ini berisi kelas bernama *World* untuk mendeskripsikan peta dan dunia dalam permainan.

- a. Atribut
  - public Position centerPoint
  - public Integer radius
  - public Integer currentTick
- b. Method
  - public Position getCenterPoint()
  - public void setCenterPoint(Position centerPoint)
  - public Integer getRadius()
  - public void setRadius(Integer radius)
  - public Integer getCurrentTick()
  - public void setCurrentTick(Integer currentTick)

### 3. Services

Folder *Services* berisi kelas-kelas yang dibuat untuk mengimplementasikan algoritma *greedy* pada bot. Berikut *file* yang terdapat dalam folder ini.

#### 1. BotService.java

File ini berisi kelas bernama *BotService* untuk yang berisi atribut serta *method-method* untuk melakukan algoritma *greedy* yang utama pada bot.

##### a. Atribut

- static KejarMusuh kejarmusuh { *import file KejarMusuh.java* }
- static Makan makan { *import file Makan.java* }
- private GameObject bot { *inisialisasi bot* }
- private PlayerAction playerAction { *inisialisasi playerAction* }
- private GameState gameState { *inisialisasi gameState* }
- private Position centralMap ( *inisialisasi titik tengah peta 0,0* )

##### b. Method

- public BotService()
  - { *konstruktor dari kelas BotService* }
- public GameObject getBot()
  - { *method getter untuk bot* }
- public void setBot(GameObject bot)
  - { *method setter untuk bot* }

- public PlayerAction getPlayerAction()  
*{ method getter untuk PlayerAction }*
- public void setPlayerAction(PlayerAction playerAction)  
*{ method setter untuk PlayerAction }*
- public GameState getGameState()  
*{ method getter dari GameState }*
- public void setGameState(GameState gameState)  
*{ method setter dari GameState }*
- private void updateSelfState()  
*{ method untuk melakukan perbaharuan state }*
- public void computeNextPlayerAction(PlayerAction playerAction)  
*{ method utama yang dipanggil pada Main.java dan berisi implementasi algoritma greedy yang dibuat }*
- public boolean zonaTembakMusuh(GameObject pemainLainTerdekat)  
*{ method yang mengembalikan boolean untuk mendefinisikan zona tembak musuh }*
- public boolean masihKecil()  
*{ method yang mengembalikan boolean untuk mendefinisikan ukuran bot }*
- private double getDistanceBetween(GameObject object1, GameObject object2)  
*{ method yang mengembalikan bilangan real yaitu jarak antara object1 dan object2 }*
- private double getDistanceToPoint(Position target)  
*{ method yang mengembalikan bilangan real yaitu jarak antara objek (bot) dengan titik target }*
- private int getHeadingBetween(GameObject otherObject1, GameObject otherObject2)  
*{ method yang mengembalikan bilangan integer yaitu arah yang diberikan otherObject1 ke otherObject2 }*
- private int getSizeBetween(GameObject object1, GameObject object2)

- private int getHeadingToPoint(Position target)
 

{ method yang mengembalikan bilangan integer yaitu arah yang diberikan objek (bot) ke sebuah titik target }
- private int toDegrees(double v)
 

{ method yang mengembalikan bilangan integer yaitu konversi v dari radian ke derajat }
- private void printState(int jumlahplayerTotal, int flag, GameObject bot, PlayerAction playerAction, int add1, int add2, int add3, int add4)
 

{ method untuk mencetak keadaan state game }

## 2. KejarMusuh.java

### a. Atribut

-

### b. Method

- private int getHeadingBetween(GameObject bot, GameObject otherObject)
 

{ method yang mengembalikan bilangan integer yaitu arah yang diberikan objek bot ke otherObject }
- private int toDegrees(double v)
 

{ method yang mengembalikan bilangan integer yaitu konversi v dari radian ke derajat }
- public void ketemuMusuh(GameObject bot, PlayerAction playerAction, GameObject pemainLainTerdekat)
 

{ method implementasi algoritma greedy yang dilakukan ketika pemain bertemu dengan lawan/musuh }

## 3. Makan.java

### a. Atribut

-

### b. Method

- private int getHeadingBetween(GameObject bot, GameObject otherObject)
 

{ method yang mengembalikan bilangan integer yaitu arah yang diberikan bot ke otherObject }
- private double getDistanceBetween(GameObject object1, GameObject object2)
 

{ method yang mengembalikan bilangan real yaitu jarak antara object1 dengan object2 }
- private int toDegrees(double v)
 

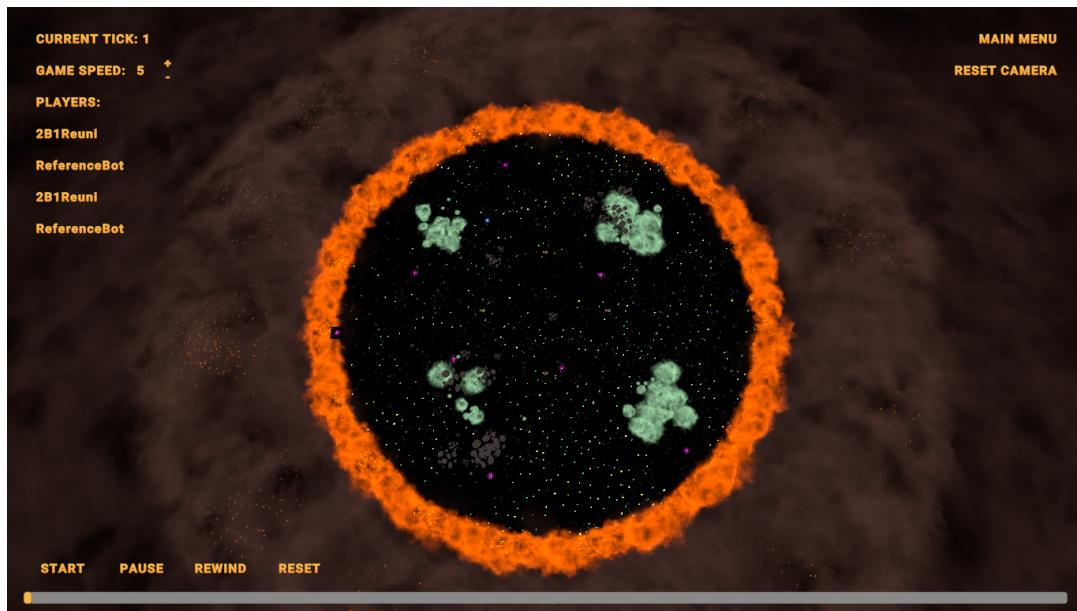
{ method yang mengembalikan bilangan integer yaitu konversi v dari radian ke derajat }
- public void cariMakan(GameObject bot, PlayerAction playerAction, int radiusPeta, GameObject superFoodTerdekat, GameObject superFoodTerdekat2, GameObject foodTerdekat, GameObject foodTerdekat2)
 

{ method implementasi algoritma greedy yang dilakukan ketika pemain mencari makan }

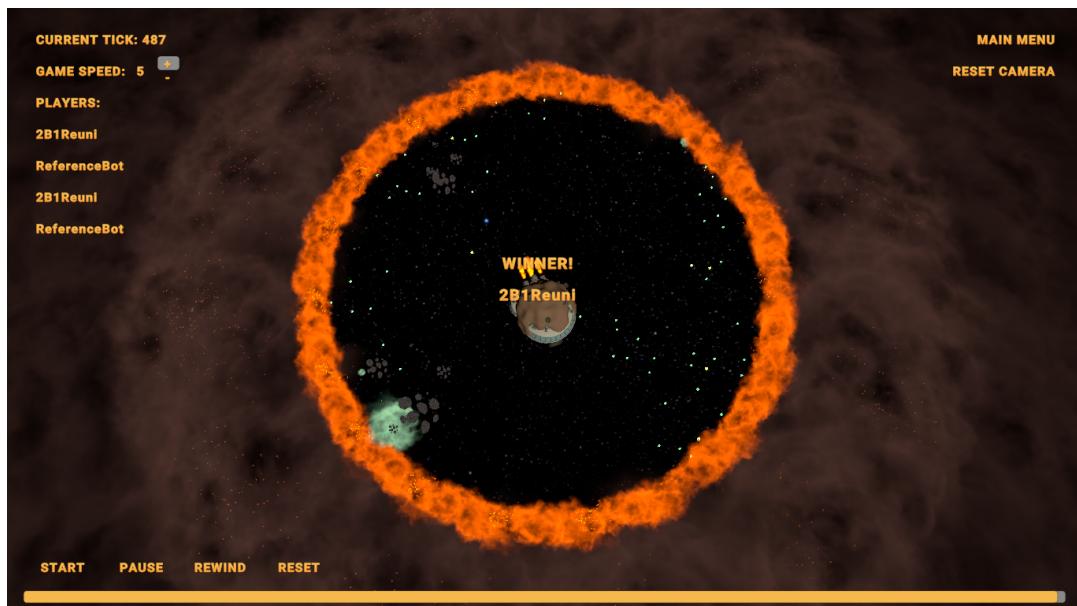
## D. Pengujian

Pengujian bot dilakukan dengan menandingkan bot buatan penulis dengan bot referensi yang telah disediakan oleh *Entelect*. Hasil pertandingan dapat berubah dengan sangat dinamis karena map dibangun secara acak dan otomatis saat permainan dimulai. Berikut adalah beberapa tangkapan layar hasil percobaannya.

## 1. Uji Kasus 1

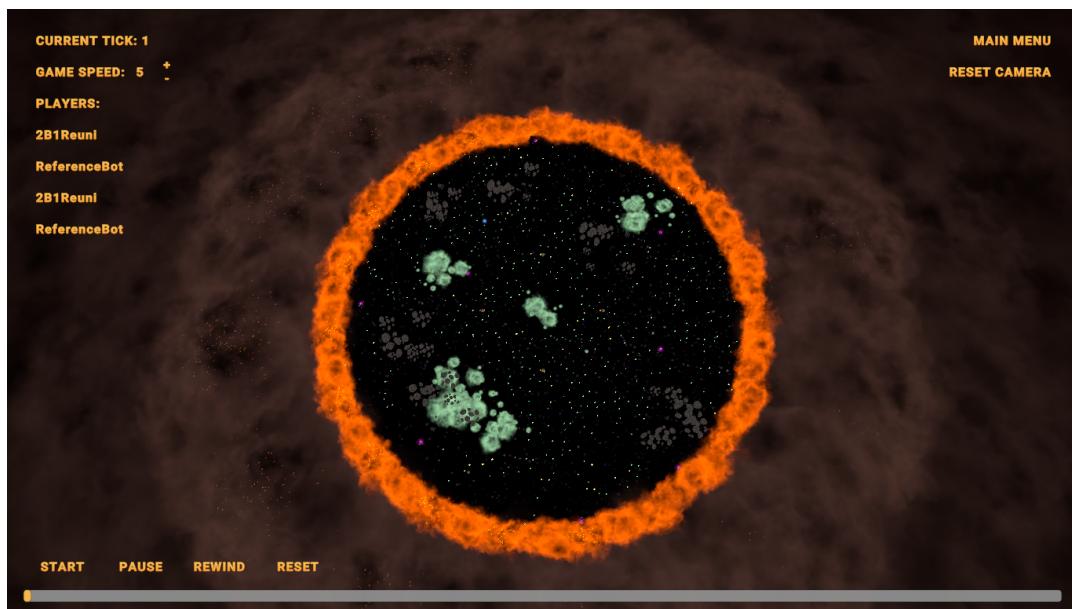


Gambar 5.1. Kondisi Awal Permainan Uji Kasus 1



Gambar 5.2. Kondisi Akhir Permainan Uji Kasus 1

## 2. Uji Kasus 2



Gambar 5.3. Kondisi Awal Permainan Uji Kasus 2



Gambar 5.4. Kondisi Akhir Permainan Uji Kasus 2

## **BAB V**

### **KESIMPULAN, DAN SARAN**

#### **A. Kesimpulan**

Permainan Galaxio dapat diselesaikan dengan strategi algoritma Greedy. Kelompok kami membuat strategi greedy sehingga pemain dapat bertahan hingga akhir. Bot yang kami buat dapat menyelesaikan permainan Galaxio dengan cukup optimal dan mampu mengeliminasi semua bot lawan. Kami juga memanfaatkan PlayerAction yang ada untuk menghindari sekaligus mengeliminasi lawan yang ada. Implementasi program yang kami buat dapat dilihat pada [laman ini](#).

#### **B. Saran**

Terkait dengan topik terkait, kami menyarankan beberapa hal untuk diperhatikan sebagai berikut:

- Dalam algoritma greedy yang dipakai, penulis menyarankan untuk memprioritaskan penggunaan teleport
- Penggunaan Supernova dianggap kurang efisien karena ketersediaannya yang cukup random di sepanjang permainan.

## **BAB VI**

## **DAFTAR PUSTAKA**

1. <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>
2. [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)
3. [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)
4. [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)

**Link Repository :** [https://github.com/margarethaolivia/Tubes1\\_2B1Reuni](https://github.com/margarethaolivia/Tubes1_2B1Reuni)

**Link Video :** <https://youtu.be/319gj98S3Hw>