

**PEMANFAATAN ALGORITMA *GREEDY* DALAM APLIKASI
PERMAINAN “GALAXIO”**

Laporan Tugas Besar 1

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma
pada Semester 2 (satu) Tahun Akademik 2022/2023



Oleh

Muhamad Aji Wibisono 13521095

Chiquita Ahsanunnisa 13521129

Alisha Listya Wardhani 13521171

Kelompok T~T_k3nsl3rBr30w0ghh_T~T

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023**

DAFTAR ISI

DAFTAR ISI	1
BAB I	
DESKRIPSI TUGAS	3
1.1 Deskripsi Permasalahan	3
1.2 Spesifikasi Program	3
BAB II	
LANDASAN TEORI	6
2.1 Algoritma Greedy	6
2.2 Cara Kerja Program	8
BAB III	
APLIKASI STRATEGI GREEDY	10
3.1 Pemetaan Elemen Algoritma Greedy Bot Permainan Galaxio	10
3.2 Alternatif Solusi Strategi Greedy	14
3.2.1 Strategi Heuristik General Bot	14
3.3 Analisis Efisiensi dari Kumpulan Solusi Alternatif Strategi Greedy	19
3.4. Efektivitas Alternatif Solusi Strategi Greedy	20
3.3.1 Efektivitas Strategi Default State Bot	20
3.3.2 Efektivitas Strategi Escape State Bot	20
3.3.2 Efektivitas Strategi Dodge State Bot	21
3.3.2 Efektivitas Strategi Attack State Bot	21
3.4 Strategi Greedy yang Dipilih	22
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	24
4.1 Implementasi Algoritma Greedy	24
4.2 Struktur Data yang Digunakan	30
4.3. Eksperimen Hasil Implementasi	42
BAB V	
KESIMPULAN DAN SARAN	50
5.1 Kesimpulan	50
5.2 Saran	50
DAFTAR PUSTAKA	51
LAMPIRAN	52
Lampiran 1 Link Repository GitHub	52
Lampiran 2 Link Video Bonus	52

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Permasalahan

Galaxio adalah sebuah *game battle royale* yang mempertandingkan *bot* kapal seseorang dengan beberapa *bot* kapal yang lain. Tujuan dari permainan ini adalah agar *bot* kapal yang digunakan tetap hidup hingga akhir permainan.

Agar dapat memenangkan pertandingan, setiap *bot* harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan. Pada tugas ini, mahasiswa diminta untuk mengimplementasikan *bot* kapal dalam permainan Galaxio dengan menggunakan algoritma *greedy* untuk memenangkan permainan.

1.2 Spesifikasi Program

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine* Galaxio. Beberapa aturan umum dari Galaxio 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 memperkecil 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.
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* mengonsumsi *food*, maka Player akan bertambah ukuran yang sama dengan *food*. *Food* memiliki peluang untuk berubah menjadi *super food*. Apabila *super food* dikonsumsi maka setiap makan *food*, efeknya akan 2 kali dari *food* yang dikonsumsi. Efek dari *super food* bertahan selama 5 *tick*.

4. *Wormhole* ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. *Wormhole* akan bertambah besar setiap *tick game* hingga ukuran maksimum. 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 clouds*, 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 kuarter pertama dan kuarter 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 arah 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 maksimum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maksimum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa *command* yang dapat dilakukan oleh *player*. Setiap *tick*, *player* hanya dapat memberikan satu *command*.
11. Setiap *player* akan memiliki *score* yang hanya dapat dilihat jika permainan berakhir. *Score* ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal player lain, maka *score* bertambah 10, jika mengonsumsi food atau melewati *wormhole*, maka *score* bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan *score* tertinggi.

Detail lebih lanjut mengenai aturan permainan dapat dilihat [di sini](#).

Dengan memanfaatkan pengetahuan tentang aturan-aturan di atas, mahasiswa diminta untuk membuat algoritma untuk *bot game* Galaxio. Bahasa pemrograman yang digunakan untuk menuliskan

algoritma *bot* adalah bahasa Java. Permainan dijalankan melalui *game engine* yang sudah dibuat oleh Entellect Challenge yang dapat dilihat [di sini](#). Selain itu, panduan untuk menjalankan permainan, membuat *bot*, *build src code*, dan melihat *visualizer* dapat dilihat [di sini](#).

BAB II

LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*). Pada setiap langkahnya, algoritma memilih pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi kedepannya (prinsip “*take what you can get now!*”) dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma *greedy* digunakan untuk memecahkan persoalan optimasi, yaitu persoalan mencari solusi optimal (maksimisasi atau minimisasi).

Secara umum, algoritma *greedy* memiliki elemen-elemen yang dijabarkan pada Tabel X.X di bawah ini.

Tabel X.X Elemen-Elemen Algoritma *Greedy*

No.	Elemen	Penjelasan
1.	Himpunan kandidat (C)	Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah
2.	Himpunan solusi (S)	Himpunan yang berisi kandidat yang sudah dipilih
3.	Fungsi solusi	Fungsi yang menentukan apakah himpunan kandidat yang dipilih merupakan solusi
4.	Fungsi seleksi (<i>selection function</i>)	Fungsi yang menyeleksi kandidat berdasarkan strategi <i>greedy</i> tertentu yang bersifat <i>heuristic</i>
5.	Fungsi kelayakan (<i>feasible</i>)	Fungsi yang memeriksa apakah kandidat yang dipilih layak dimasukkan ke himpunan solusi
6.	Fungsi objektif	Fungsi yang mengoptimasi tujuan

Penjelasan elemen-elemen di atas dapat diringkas dalam satu kalimat: “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.”

Algoritma *greedy* memiliki skema umum sebagai berikut.

```

function greedy(C : himpunan_kandidat) -> himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
x : kandidat
S : himpunan_solusi

Algoritma:
    S <- {} {inisialisasi S dengan kosong}
    while (not SOLUSI(S)) and (C != {}) do
        x <- SELEKSI(C) { pilih sebuah kandidat dari C}
        C <- C - {x} {buang x dari C karena sudah dipilih}
        if LAYAK(S ∪ {x}) then { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
            S <- S ∪ {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

```

Pada akhir setiap iterasi, akan terbentuk solusi optimum lokal, sedangkan pada akhir *loop while-do* diperoleh solusi optimum global (jika ada). Optimum global belum tentu merupakan solusi paling optimal. Optimum global bisa jadi merupakan solusi sub-optimal atau *pseudo-optimum*.

Dari penjelasan sebelumnya, dapat dilihat bahwa algoritma ini tidak selalu memberikan solusi paling optimal. Hal ini terjadi karena:

1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada.
2. Terdapat beberapa fungsi seleksi yang berbeda. Fungsi seleksi yang dipilih harus tepat untuk menghasilkan solusi yang optimal.

Meskipun demikian, algoritma *greedy* dapat memberikan solusi yang sub-optimal atau *pseudo-optimum*.

2.2 Cara Kerja Program

Galaxio merupakan permainan yang memanfaatkan data yang terjadi pada waktu tertentu (disebut dengan tick). Pertama-tama, bot menyambungkan koneksi kepada Game Engine. Setelah terhubung, *game engine* akan menginisialisasi dan menerima *game state*. Bot kemudian mengambil data yang tersedia pada *gamestate*. Berikut merupakan rincian data yang terdapat pada *game state*.

A. World

World merupakan peta yang dihasilkan oleh *game engine*. Di dalam World, terdapat data yang dapat diakses, yaitu:

- a. *centerPoint*, nilai posisi titik tengah dari peta,
- b. *radius*, nilai radius peta pada saat tertentu, dan
- c. *currentTick*, nomor tick pada saat tertentu.

B. GameObjects dan PlayerObject

GameObjects menangani objek-objek yang terdapat pada permainan. Objek permainan terdiri dari PLAYER, FOOD, WORMHOLE, GAS_CLOUD, ASTEROID_FIELD, TORPEDOSALVO, SUPERFOOD, SUPERNOVAPICKUP, SUPERNOVABOMB, TELEPORTER, dan SHIELD.

Objek tersebut memiliki beberapa atribut yang dapat diakses:

- a. *id*, nomor bernilai unik untuk setiap objek,
- b. *size*, nilai ukuran objek,
- c. *speed*, nilai kecepatan objek pada saat tertentu,
- d. *currentHeading*, nilai heading atau arah objek pada saat tertentu,
- e. *position*, posisi objek pada saat tertentu.

PlayerObject dapat memungkinkan akses ke data *bot* pemain ataupun *bot* lawan. *PlayerObject* mengandung atribut yang sama seperti *GameObjects*, tetapi dengan beberapa atribut tambahan yang dapat diakses. Berikut merupakan rinciannya.

- a. *Effects*, nilai enkripsi efek yang berlaku pada objek,
- b. *TorpedoSalvoCount*, jumlah torpedo salvo yang dimiliki oleh objek,
- c. *SupernovaAvailable*, jumlah supernova yang dimiliki oleh objek,
- d. *TeleporterCount*, jumlah teleporter yang dimiliki oleh objek, dan
- e. *ShieldCount*, jumlah tameng pelindung yang dimiliki oleh objek.

Pembacaan data-data tersebut memungkinkan *bot* untuk mengimplementasikan strategi Algoritma Greedy pada *bot*. Strategi ini didasari pilihan yang paling sesuai dan optimum untuk setiap *tick* pada permainan Galaxio. Setelah sebuah strategi dipilih, *bot* akan mengembalikan respon aksi ke game *game engine*. Berikut merupakan daftar COMMAND yang bisa dipilih.

- a. FORWARD, perintah agar bot maju dengan kecepatan yang dimilikinya
- b. STOP, perintah untuk berhenti dan tidak melakukan apa-apa,
- c. STARTAFTERBURNER, perintah untuk memulai efek afterburner yang dapat meningkatkan kecepatan bot.
- d. STOPAFTERBURNER, perintah untuk menghentikan efek afterburner yang dapat meningkatkan kecepatan bot.
- e. FIRETORPEDOES, perintah untuk meluncurkan torpedo dari *bot*
- f. FIRESUPERNOVA, perintah untuk meluncurkan supernova dari *bot*,
- g. DETONATESUPERNOVA, perintah untuk meledakkan supernova yang telah diluncurkan,
- h. FIRETELEPORTER, perintah untuk meluncurkan teleporter dari *bot*,
- i. TELEPORT, perintah untuk teleportasi diri ke posisi teleporter yang sudah diluncurkan,
- j. ACTIVATESHIELD, perintah untuk menyalakan tameng pelindung untuk menghindari *damage*

Bersama dengan nilai heading (arah objek tersebut), perintah tersebut kemudian dijalankan. Pemilihan perintah tersebut memerlukan algoritma yang handal. Sebuah keadaan pada setiap *tick* dapat dianalisis dan mencari COMMAND untuk *tick* berikutnya. Algoritma yang diimplementasikan berbentuk fungsi objektif dari algoritma *greedy*. Selain mengirimkan COMMAND, *bot* juga mengirimkan arah hadap *bot* yang selanjutnya akan disebut HEADING. Hal ini menjadi hal yang krusial dalam permainan. *Bot* akan dilombakan selama masih ada minimal dua *bot* yang tersisa atau mencapai *tick* maksimum.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Pemetaan Elemen Algoritma *Greedy* Bot Permainan Galaxio

Subbab ini menjelaskan mengenai bagaimana elemen anggota greedy diimplementasikan pada *bot* Galaxio. Adapun terdapat beberapa penyelesaian persoalan Galaxio.

3.1.1. Pemetaan Elemen Algoritma Greedy pada Permasalahan General *Bot*

Pada permainan Galaxio, *bot* mempunyai tujuan agar menjadi *bot* terakhir yang berada pada permainan. Pada bot yang kami buat, terdapat STATETYPES yang digunakan sebagai acuan bagaimana *bot* tersebut akan berperilaku selama *tick* tersebut. STATETYPES tersebut yaitu:

- A. DEFAULT_STATE, keadaan normal ketika *bot* berfokus mencari makanan dan terhindar dari objek-objek yang dapat membahayakan,
- B. ATTACK_STATE, keadaan ketika *bot* difokuskan untuk menyerang lawan,
- C. ESCAPE_STATE, keadaan ketika *bot* difokuskan untuk melarikan diri, dan
- D. DODGE_STATE, keadaan ketika *bot* difokuskan untuk menghindar dari suatu objek.

STATETYPES ini kemudian dipilih sedemikian rupa sehingga diharapkan dapat mencapai objektif.

Tabel 3.1.1.1 Elemen Algoritma *Greedy* General Bot

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari STATETYPES untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari STATETYPES yang dapat membuat <i>bot</i> bertahan pada permainan.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari STATE TYPES tersebut dapat membuat <i>bot</i> bertahan pada permainan.
4.	Fungsi seleksi	Memilih STATETYPES berdasarkan data keadaan saat tersebut serta fungsi heuristik tingkat prioritas STATETYPES yang harus diikuti. Tingkat prioritas bersifat statik selama permainan berlangsung.
5.	Fungsi kelayakan	Memeriksa apakah STATETYPES yang digunakan merupakan STATETYPES yang valid.
6.	Fungsi objektif	STATETYPES yang dijalankan mengakibatkan <i>bot</i> melakukan aksi yang efektif serta bertahan dalam permainan.

3.1.2. Pemetaan Elemen Algoritma Greedy pada Permasalahan Default State *Bot*

Default State merupakan keadaan dimana *bot* tidak mendeteksi ancaman musuh ataupun target musuh (*idle*). Pada keadaan ini, dipilih COMMAND yang mengimplementasikan algoritma *greedy*.

Tabel 3.1.2.2 Elemen Algoritma *Greedy* Mencari Makanan

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari COMMAND untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari COMMAND yang dapat membuat <i>bot</i> bertahan pada permainan.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari COMMAND tersebut dapat membuat <i>bot</i> bertahan pada permainan.
4.	Fungsi seleksi	Memilih COMMAND berdasarkan data keadaan saat tersebut serta fungsi heuristik tingkat prioritas COMMAND yang harus diikuti. Tingkat prioritas bersifat statik selama permainan berlangsung.
5.	Fungsi kelayakan	Memeriksa apakah COMMAND yang digunakan merupakan COMMAND yang valid. Daftar COMMAND yang valid pada strategi ini adalah: STOP, FORWARD atau <i>bot</i> tidak melakukan COMMAND.
6.	Fungsi objektif	Mencari permutasi COMMAND yang dijalankan mengakibatkan <i>bot</i> tidak mati.

Adapun dalam Default State *Bot*, terdapat implementasi algoritma greedy untuk pencarian makanan. Sub-permasalahan ini berfokus pada pengarahan HEADING *bot* secara efektif dan efisien. Pencarian makanan dilakukan dengan tujuan mencapai makanan terbanyak dengan jarak terkecil (optimum) dari posisi *bot* pada *tick* tertentu.

Tabel 3.1.2.2 Elemen Algoritma *Greedy* Mencari Makanan

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari HEADING <i>bot</i> untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari HEADING <i>bot</i> yang dapat membuat <i>bot</i> mencari makan sebanyak-banyaknya.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari HEADING <i>bot</i> tersebut dapat membuat <i>bot</i> memperoleh makanan sebanyak-banyaknya.
4.	Fungsi seleksi	Memilih HEADING <i>bot</i> berdasarkan titik berat paling optimum secara lokal dari FOOD terdekat dengan <i>bot</i> pada setiap tick. Harapannya adalah HEADING yang dipilih dapat mencapai makanan sebanyak-banyaknya.

5.	Fungsi kelayakan	Memeriksa apakah HEADING <i>bot</i> yang dipilih merupakan HEADING <i>bot</i> yang valid. HEADING yang valid adalah $0 < \text{HEADING} < 360$.
6.	Fungsi objektif	HEADING <i>bot</i> yang dipilih membuat <i>bot</i> mencapai makanan terbanyak dengan jarak terkecil (optimum) dari posisi <i>bot</i> pada tick tersebut.

3.1.3. Pemetaan Elemen Algoritma Greedy pada Permasalahan Escape State *Bot*

Escape State Bot merupakan keadaan dimana *bot* mendeteksi *bot* lain yang lebih besar dari ukuran *bot* sendiri yang berada di dalam radar threat *bot*. *Bot* berfokus untuk bertahan pada permainan dengan cara kabur dari *bot* besar lainnya. Pada keadaan ini, dipilih aksi yang diharapkan dapat mencapai tujuan tersebut. Aksi tersebut beragam, tetapi berfokus kepada COMMAND FORWARD dan pemilihan HEADING optimal untuk melarikan diri. HEADING dipilih dengan menggunakan implementasi algoritma *greedy*.

Tabel 3.1.3.1 Elemen Algoritma *Greedy Defense State Bot*

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari HEADING <i>bot</i> untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari HEADING <i>bot</i> yang dapat membuat <i>bot</i> kabur dari <i>bot</i> besar lain yang terdeteksi di dalam radar.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari HEADING <i>bot</i> tersebut dapat membuat <i>bot</i> bertahan pada permainan dan kabur dari <i>bot</i> besar lain yang terdeteksi di dalam radar.
4.	Fungsi seleksi	Memilih HEADING <i>bot</i> berdasarkan data keadaan saat tersebut serta fungsi heuristik tingkat prioritas HEADING <i>bot</i> yang harus diikuti. Tingkat prioritas bersifat statik selama permainan berlangsung. Fungsi tersebut mengikuti strategi penghindaran paling optimum untuk setiap tick.
5.	Fungsi kelayakan	Memeriksa apakah HEADING <i>bot</i> yang dipilih merupakan HEADING <i>bot</i> yang valid. HEADING yang valid adalah $0 < \text{HEADING} < 360$.
6.	Fungsi objektif	HEADING <i>bot</i> yang dipilih membuat <i>bot</i> bertahan pada permainan dan kabur dari <i>bot</i> besar lain yang terdeteksi di dalam radar.

3.1.4. Pemetaan Elemen Algoritma Greedy pada Permasalahan Dodge State *Bot*

Dodge State Bot merupakan keadaan dimana *bot* mendeteksi objek lain (TORPEDO, SUPERNOVABOMB) yang mendekati atau berada dalam radar *bot*.

Tabel 3.1.3.1 Elemen Algoritma *Greedy* Dodge State *Bot*

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari COMMAND untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari COMMAND yang dapat membuat <i>bot</i> menghindari objek yang menuju <i>bot</i> dan tetap bertahan.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari COMMAND tersebut dapat menuju posisi yang paling jauh dari objek yang menuju <i>bot</i> .
4.	Fungsi seleksi	Memilih COMMAND berdasarkan data keadaan saat tersebut serta fungsi heuristik tingkat prioritas COMMAND yang harus diikuti. Tingkat prioritas bersifat statik selama permainan berlangsung.
5.	Fungsi kelayakan	Memeriksa apakah COMMAND yang digunakan merupakan COMMAND yang valid. Daftar COMMAND yang valid pada strategi ini adalah : ACTIVATESHIELD, FORWARD atau <i>bot</i> tidak melakukan COMMAND.
6.	Fungsi objektif	Mencari permutasi COMMAND yang dijalankan mengakibatkan <i>bot</i> menghindari objek secara efektif dan efisien dan bertahan pada permainan.

3.1.4. Pemetaan Elemen Algoritma Greedy pada Permasalahan Attack State *Bot*

Attack State Bot merupakan keadaan dimana *bot* mendeteksi *bot* lain yang ukurannya lebih kecil dari *bot* pemain yang berada dalam radar *attack bot*.

Tabel 3.1.3.1 Elemen Algoritma *Greedy* Dodge State *Bot*

No.	Elemen	Pemetaan
1.	Himpunan kandidat (C)	Seluruh permutasi dari COMMAND untuk setiap tick.
2.	Himpunan solusi (S)	Kemungkinan permutasi dari COMMAND yang dapat membuat <i>bot</i> melawan <i>bot</i> yang terdeteksi dan tetap bertahan.
3.	Fungsi solusi	Melakukan pengecekan apakah permutasi dari COMMAND tersebut dapat melawan <i>bot</i> yang terdeteksi oleh radar <i>bot</i> .
4.	Fungsi seleksi	Memilih COMMAND berdasarkan data keadaan saat tersebut serta fungsi heuristik tingkat prioritas COMMAND yang harus diikuti. Tingkat prioritas bersifat statik selama permainan berlangsung.

5.	Fungsi kelayakan	Memeriksa apakah COMMAND yang digunakan merupakan COMMAND yang valid. Daftar COMMAND yang valid pada strategi ini adalah : FIRETORPEDOES, ACTIVATESHIELD, FORWARD, FIRETELEPORTER, TELEPORT atau <i>bot</i> tidak melakukan COMMAND.
6.	Fungsi objektif	Mencari permutasi COMMAND yang dijalankan mengakibatkan <i>bot</i> melawan <i>bot</i> yang terdeteksi dengan cara paling efektif dan efisien dan tetap bertahan.

3.2 Alternatif Solusi Strategi *Greedy*

Strategi algoritma Greedy yang diimplementasikan dalam bot pada permainan Galaxio berfokus kepada keadaan dan elemen yang dapat diakses oleh bot dan kemudian mengubah STATETYPES.

3.2.1 Strategi Heuristik General Bot

Pada setiap langkah (*tick*), pilihlah STATETYPES dengan prioritas tertinggi. Prioritas sudah disusun berdasarkan keadaan yang paling menguntungkan bagi *bot*. *Bot* pada permainan membutuhkan penanganan pada setiap keadaan. Hal ini diharapkan dapat memberikan *damage* minimum dan *attack* maksimum untuk setiap tick.

Strategi pertama yang dapat diimplementasikan adalah memilih STATETYPES ATTACK_STATE. Keadaan ini berfungsi khusus untuk menyerang dan mengejar lawan. Keadaan ini memberi keuntungan terbesar, yaitu mengakibatkan *bot* musuh mati. Keuntungan kedua adalah menambah ukuran *bot* pemain dan mengurangi ukuran *bot* lawan (sebanyak 1-10 unit). Pada keadaan ini, *bot* tidak memperhatikan posisi makanan sehingga meminimalisir penambahan ukuran. Selain itu, menjalankan strategi penyerangan pada lawan yang besar akan sangat membahayakan *bot* pemain.

Strategi kedua yang dapat diimplementasikan adalah memilih STATETYPES DODGE_STATE. Keadaan ini berfungsi untuk menghindari segala jenis objek yang dapat menjadi rintangan bagi *bot* pemain. Pada strategi ini, prioritas penghindaran adalah sebagai berikut: EDGE, TORPEDO, GAS_CLOUD, WORMHOLE. Keuntungan pada keadaan ini adalah *bot* pemain terhindar dari objek yang berpotensi menyebabkan pengurangan ukuran *bot*.

Strategi ketiga yang dapat diimplementasikan adalah memilih STATETYPES ESCAPE_STATE. Keadaan ini berfungsi untuk melarikan diri dari objek-objek yang sangat membahayakan *bot* pemain. Keuntungan pada keadaan ini adalah *bot* pemain tidak berada dalam situasi dimana berpotensi bahaya. Hal ini sangat krusial karena *bot* berpotensi untuk mati pada keadaan ini. Ukuran *bot* pemain tidak berkurang ataupun bertambah.

Strategi keempat yang dapat diimplementasikan adalah memilih STATETYPES DEFAULT_STATE. Keadaan ini berfokus pada pencarian makanan, dimana *bot* pemain menambah

ukurannya. Pada keadaan ini, *bot* tidak memperhitungkan algoritma untuk menghindari ataupun menyerang musuh.

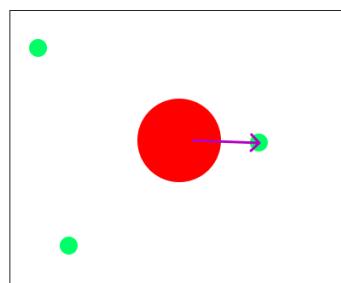
Strategi kelima merupakan gabungan dari semua *state* yang sudah dipaparkan sebelumnya. *Bot* akan mengkomputasi keadaan manakah yang paling optimal untuk dipilih pada *tick*. Berikut merupakan prioritas pada *General Bot*: ATTACK_STATE, DODGE_STATE, ESCAPE_STATE, dan DEFAULT_STATE. Dimisalkan ada *bot* lain dengan ukuran lebih besar berada dalam radar, maka ATTACK_STATE tidak layak untuk dijalankan karena potensi menang melawan *bot* tersebut rendah. Tetapi, ATTACK_STATE menjadi prioritas utama pemilihan apabila semua kondisi memenuhi. Strategi ini merupakan strategi yang dipilih penulis dalam menyelesaikan subpermasalahan *General Bot*.

3.2.2 Strategi Heuristik Default State Bot

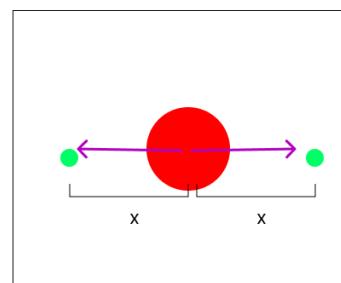
Pada setiap langkah (*tick*), pilihlah COMMAND yang menghasilkan tujuan dengan prioritas tertinggi. Tujuan telah disusun berdasarkan keadaan yang paling menguntungkan bagi *bot* pemain. Hal ini diharapkan dapat memberikan *damage* minimum untuk *bot*.

Strategi pertama yang dapat diimplementasikan adalah mendapatkan SUPERNOVA_PICKUP. Strategi ini berfokus pada pencarian SUPERNOVA yang dapat menyebabkan *damage* paling besar kepada *bot* lawan lainnya (*greedy by value*). Alasan lainnya strategi ini penting adalah karena SUPERNOVA_PICKUP hanya muncul satu kali, di kuarter pertama atau akhir permainan.

Strategi kedua yang dapat diimplementasikan adalah mencari makanan dengan jarak paling pendek dari posisi *bot* pemain (*greedy by distance*). Strategi ini berfokus pada pencarian HEADING menuju arah makanan terdekat. Adapun permasalahan yang diamati dari strategi ini adalah adanya recalling ketika ditemukan kasus makanan terdekat mempunyai jarak yang sama terhadap *bot* pemain.



Gambar 3.2.1. Pencarian makanan dengan Naive menurut jarak terdekat



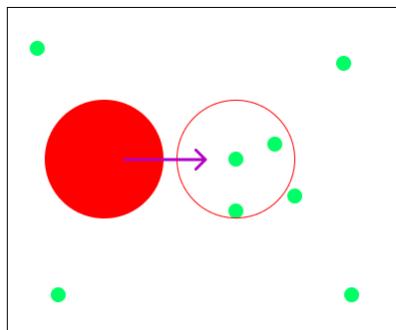
Gambar 3.2.2. Permasalahan re-calling pada Pencarian Makanan

Strategi ketiga yang dapat diimplementasikan adalah pencarian HEADING menuju titik tengah *cluster* makanan. Selain memanfaatkan algoritma *greedy by distance*, strategi ini juga memanfaatkan

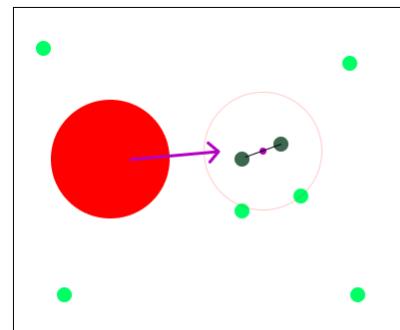
greedy by value karena diharapkan mendapatkan makanan terbanyak dengan satu kali jalan ke heading tersebut. Berikut merupakan penjelasan algoritma mencari makanan tersebut.

3.2.2.1. Strategi Greedy Pencarian Makanan

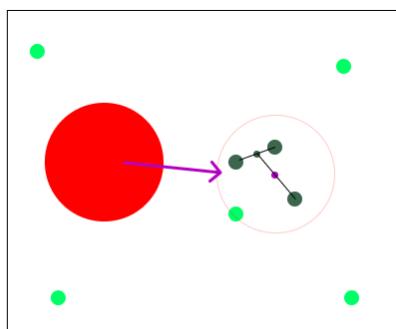
Pada setiap langkah, pilih HEADING yang menuju titik tengah *cluster* terbesar yang dapat terambil oleh *bot* dari makanan terdekat. Hal ini diharapkan mendapatkan *distance* minimum dari makanan dan *value* makanan terbanyak dalam satu kali jalan. Pada strategi ini, diimplementasikan teorema titik berat pada fisika. *Bot* pertama-tama akan memilih HEADING menuju makanan utama. Setelah mendapat titik tengah dari dua HEADING, dilakukan kembali pengukuran ke makanan terdekat. Dilakukan pengulangan sampai ditemukan makanan dengan jarak lebih besar daripada threshold. *Bot* pemain diharapkan akan melewati titik rengah ini dan bisa melahap makanan sebanya-banyaknya.



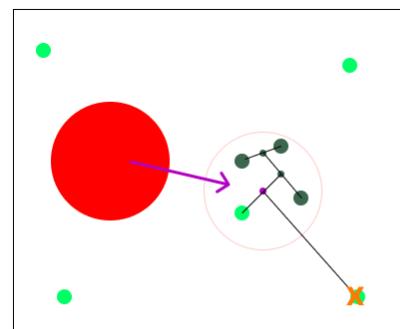
Gambar 3.2.2.1. Pencarian makanan dengan jarak terdekat dari bot, membuat **threshold** sebesar ukuran bot disekitarnya



Gambar 3.2.2.2. Pencarian titik tengah dengan makanan terdekat dari titik pertama



Gambar 3.2.2.3. Pencarian titik tengah dari makanan terdekat dengan titik sebelumnya



Gambar 3.2.2.4. Iterasi dilakukan sampai keadaan tidak layak: jarak berada diluar threshold

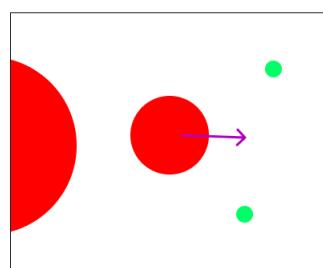
3.2.3 Strategi Heuristik Escape State Bot

Pada setiap langkah (tick), pilihlah COMMAND yang menghasilkan tujuan dengan prioritas tertinggi. Tujuan telah disusun berdasarkan keadaan yang paling menguntungkan bagi *bot* pemain. Hal ini diharapkan dapat memberikan *damage* minimum untuk *bot*.

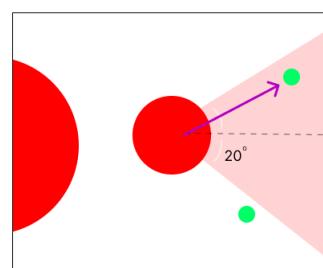
Strategi pertama yang dapat diimplementasikan adalah menembakkan TORPEDOES kepada lawan dengan jarak jauh jika lawan lebih besar daripada ukuran *bot* pemain. Lawan yang dipilih tentu adalah lawan yang paling menciptakan ancaman bagi pemain (*greedy by urgency*). Hal ini dilakukan jika TorpedoSalvoCount lebih dari nol dan ukuran *bot* tidak kurang dari 10. Ukuran *bot* yang kecil akan menyebabkan potensi bunuh diri pada *bot* dan mengakibatkan *bot* lebih terekspos.

Strategi kedua yang dapat diimplementasikan adalah mencari HEADING dengan arah berlawanan 180 derajat dengan *bot* musuh yang sedang mengejar. Ini merupakan strategi yang paling efisien dalam menghindari musuh karena menciptakan jarak yang paling jauh dari musuh.

Strategi ketiga yang dapat diimplementasikan adalah mencari HEADING dengan arah berlawanan dengan *bot* musuh tetapi memperhitungkan posisi makanan terdekat. Ambang yang dipilih untuk posisi makanan ini adalah ± 20 derajat dari arah yang berlawanan dengan musuh. Pengambilan makanan dapat menambah ukuran *bot* pemain sehingga jika selisih antara kedua *bot* yang sedang berlawanan adalah kecil akan mendapat kesempatan lebih besar untuk melawan balik. Hal ini memanfaatkan algoritma *greedy by distance*.



Gambar 3.2.2.1. Pengarahan HEADING berlawanan dengan musuh



Gambar 3.2.2.2. Pengarahan HEADING menjauhi musuh, mendekati suatu makanan (Ambang 20 derajat)

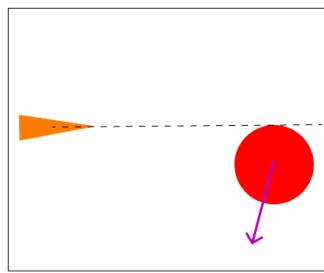
3.2.4 Strategi Heuristik Dodge State Bot

Pada setiap langkah (tick), pilihlah COMMAND yang menghasilkan tujuan dengan prioritas tertinggi. Tujuan telah disusun berdasarkan keadaan yang paling menguntungkan bagi *bot* pemain. Hal ini diharapkan dapat memberikan *damage* minimum untuk *bot*.

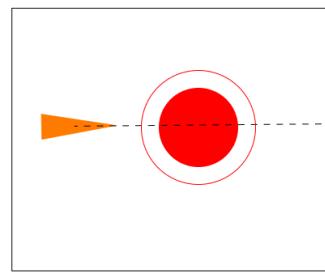
Strategi pertama yang dapat diimplementasikan adalah mencari TORPEDO terdekat yang menjadi ancaman bagi *bot* pemain. Jika TORPEDO memenuhi suatu kondisi tertentu, maka *bot* pemain memilih HEADING yang memungkinkan untuk menghindari objek tersebut. Hal ini memberikan

keuntungan maksimal bagi *bot* pemain karena jika terhindar, tidak mengurangi ukuran *bot* pemain ataupun menambah *bot* musuh.

Strategi kedua yang dapat diimplementasikan adalah mencari TORPEDO yang paling menjadi ancaman bagi *bot* pemain dikarenakan jaraknya paling dekat. TORPEDO tersebut kemudian diinterpolasi untuk didapatkan trajektorinya. Jika trajektori TORPEDO sudah sangat dekat maka COMMAND ACTIVATESHIELD akan aktif. Hal ini memberikan kerugian 20 unit kepada ukuran *bot* tetapi dapat membuat *bot* lawan tidak mendapatkan ukuran tambahan.



Gambar 3.2.4.1. Pengarahan HEADING untuk menghindari dari trajektori TORPEDO

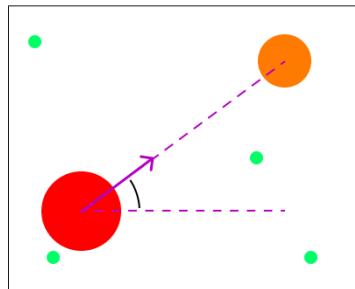


Gambar 3.2.4.2. Pengaktifan SHEILD untuk menghindari tabrakan dari trajektori TORPEDO

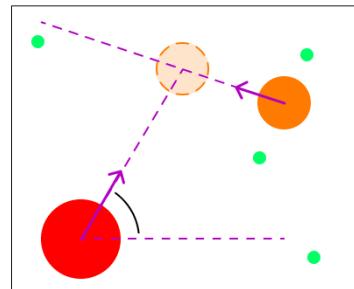
3.2.5 Strategi Heuristik Attack State Bot

Pada setiap langkah (tick), pilihlah COMMAND yang menghasilkan tujuan dengan prioritas tertinggi. Tujuan telah disusun berdasarkan keadaan yang paling menguntungkan bagi *bot* pemain. Hal ini diharapkan dapat memberikan *damage* maksimum untuk *bot*. *Command-command* yang digunakan untuk melakukan *attack* yaitu *command-command* terkait dengan *teleporter* (FIRETELEPORTER, TELEPORT), supernova (FIRESUPERNOVA, DETONATESUPERNOVA, torpedo (FIRETORPEDOES), dan pengejaran (FORWARD).

Secara umum, untuk melakukan *attack*, *bot* perlu melakukan *aiming attack*. Dalam hal ini, penulis menyiapkan empat strategi *aiming* yang dapat digunakan saat memberikan *command* bertipe *attack*: versi 0, versi 1, versi 2, dan versi 3. Pada *aim* versi 0, algoritma langsung memberikan HEADING yang mengarah tepat ke posisi lawan pada *tick* tersebut. Pada *aim* versi 1, algoritma akan memberikan HEADING yang mengarah ke posisi yang diperkirakan akan dilewati oleh lawan. Versi 2 dan 4 merupakan strategi *aiming* yang jauh lebih khusus. Pada *aim* versi 2, *attack* berusaha diarahkan untuk melakukan *blocking* terhadap lawan yang berada di ujung dunia. Pada *aim* versi 3, *attack* berusaha diarahkan untuk melakukan *blocking* terhadap lawan yang berada di dekat *gas cloud*.



Gambar 3.2.5.1. Menentukan attack aim tanpa prediksi (aim versi 0)



Gambar 3.2.5.1. Menentukan attack aim dengan prediksi (aim versi 1)

Pada implementasi prioritasnya, penulis mengurutkan prioritas *attack* (strategi dan *command*) dengan mempertimbangkan jenis *attack* yang memberikan *damage* terbesar kerugian terkecil terhadap bot yang digunakan. Penulis juga mempertimbangkan ukuran dari *bot* penulis, ukuran dari *bot* lawan, jarak antara *bot* penulis dengan *bot* lawan, serta kecepatan gerak *bot* penulis dan kecepatan gerak lawan.

3.3 Analisis Efisiensi dari Kumpulan Solusi Alternatif Strategi *Greedy*

Permainan Galaxio memungkinkan banyak data gameState yang diketahui seperti posisi objek, HEADING objek, dan kecepatan objek tersebut. Hal tersebut membuat program berjalan dengan efektif. Hal yang banyak digunakan pada bot pemain adalah menyortir sebuah list berisikan objek berdasarkan jaraknya ke bot pemain. Pada tugas besar ini, penulis menggunakan library bawaan Java yang menggunakan mergesort sehingga memiliki kompleksitas waktu $O(n \log(n))$.

Pada strategi State Base Bot, pertama-tama dilakukan pencarian posisi *bot* lawan dengan terurut berdasarkan jaraknya ke *bot* pemain. Algoritma ini memiliki kompleksitas waktu $O(n \log(n))$. Setelah itu, dilakukan beberapa kondisional untuk menentukan STATETYPES yang perlu dilakukan berdasarkan tingkat prioritas yang paling menguntungkan bagi *bot*. Hal ini memiliki kompleksitas waktunya konstan atau $O(1)$. Maka strategi Dodge State Bot memiliki kompleksitas $O(n \log(n))$.

Pada strategi Default State Bot, pertama-tama dilakukan pencarian posisi makanan dengan terurut berdasarkan jaraknya ke bot pemain. Algoritma ini memiliki kompleksitas waktu $O(n \log(n))$. Setelah itu, dilakukan pembuatan cluster makanan dengan iterasi dari makanan terdekat sampai makanan dengan jarak sama dengan ukuran bot pemain. Algoritma ini memiliki kompleksitas waktu $O(n)$. Maka strategi default state bot memiliki efisiensi $O(n \log(n)) + O(n)$ atau jika disimplifikasi menjadi $O(n \log(n))$.

Pada strategi Dodge State Bot, pertama-tama dilakukan pencarian posisi torpedo dengan terurut berdasarkan jaraknya ke *bot* pemain. Algoritma ini memiliki kompleksitas waktu $O(n \log(n))$. Setelah itu, dilakukan beberapa kondisional untuk menentukan COMMAND yang perlu dilakukan berdasarkan

tingkat prioritas yang paling menguntungkan bagi *bot* yang kompleksitas waktunya konstan atau $O(1)$. Maka strategi Dodge State Bot memiliki kompleksitas $O(n \log(n))$.

Pada strategi Escape State Bot, pertama-tama dilakukan pencarian posisi *bot* lawan dengan terurut berdasarkan jaraknya ke *bot* pemain. Algoritma ini memiliki kompleksitas waktu $O(n \log(n))$. Setelah itu, dilakukan beberapa kondisional untuk menentukan COMMAND yang perlu dilakukan berdasarkan tingkat prioritas yang paling menguntungkan bagi *bot*. Hal ini memiliki kompleksitas waktunya konstan atau $O(1)$. Maka strategi Dodge State Bot memiliki kompleksitas $O(n \log(n))$.

Pada strategi terakhir yaitu Attack State Bot, pertama-tama dilakukan pencarian posisi *bot* lawan, GAS_CLOUD, TELEPORTER dengan terurut berdasarkan jaraknya ke *bot* pemain. Algoritma ini memiliki kompleksitas waktu $O(n \log(n))$. Setelah itu, dilakukan beberapa kondisional untuk menentukan COMMAND yang perlu dilakukan berdasarkan tingkat prioritas yang paling menguntungkan bagi *bot*. Hal ini memiliki kompleksitas waktunya konstan atau $O(1)$. Maka strategi Attack State Bot memiliki kompleksitas $O(n \log(n))$.

3.4. Efektivitas Alternatif Solusi Strategi *Greedy*

3.3.1 Efektivitas Strategi Default State Bot

Strategi pertama menyebabkan *bot* pemain mendapatkan senjata terkuat dalam permainan Galaxio. Senjata ini memberikan keuntungan yang paling besar untuk *bot* pemain, namun tidak menambah ukuran *bot* pemain sehingga jika ukurannya masih kecil akan menyebabkan potensi bahaya.

Strategi kedua efektif secara kompleksitas dan merupakan algoritma yang sederhana. Secara garis besar, strategi *greedy* ini merupakan strategi paling baik, tetapi menurut penulis masih menyisakan ruang untuk pengembangan lebih lanjut. Setelah beberapa kali dilakukan tes, strategi ini akan menyebabkan masalah ketika terdapat dua FOOD dengan jarak yang sama. *Bot* pemain akan kebingungan dalam menentukan HEADING.

Strategi ketiga merupakan strategi modifikasi dari strategi kedua, dimana *bot* pemain mendapatkan posisi dari sebuah titik tengah dari sebuah *cluster* FOOD . Strategi ini sangat efektif karena dalam menuju ke satu titik, *bot* pemain mendapatkan lebih dari satu FOOD. Menurut penulis, strategi ketiga digabungkan dengan strategi pertama merupakan strategi paling efisien untuk menyelesaikan subpermasalahan dimana *bot* pemain bersifat *idle* atau tidak mendeteksi ancaman ataupun target.

3.3.2 Efektivitas Strategi Escape State Bot

Strategi pertama dengan menembakkan TORPEDO efektif karena menambahkan ukuran *bot* pemain dan mengurangi ukuran *bot* lawan. Lawan yang dipilih tentu adalah lawan yang paling

menciptakan ancaman bagi pemain (*greedy by urgency*). Namun, strategi ini risikan jika *bot* pemain memiliki ukuran yang sangat kecil.

Strategi kedua merupakan strategi yang cukup efektif, karena menghindari *bot* lawan dengan menciptakan jarak yang paling jauh. Menurut penulis, strategi ini memerlukan modifikasi lebih jauh karena jika *bot* pemain hanya membelok dengan statis sebesar 180 derajat, dapat membelok ke sebuah objek dan tidak memberikan keuntungan bagi *bot* pemain.

Strategi ketiga merupakan strategi modifikasi dari strategi kedua. Dengan mencari arah ke FOOD terdekat sambil melarikan diri (dengan syarat berada di dalam radar), *bot* pemain mendapatkan keuntungan dari makanan tersebut sehingga menimbulkan potensi ukuran *bot* pemain lebih besar dari ukuran *bot* lawan. Menurut penulis, strategi ketiga digabungkan dengan strategi pertama merupakan strategi yang paling efisien dalam penyelesaian subpermasalahan melarikan diri dari *bot* lawan. Dengan beberapa kondisional seperti ukuran *bot* pemain serta jarak *bot* pemain dengan lawan, penulis dapat menentukan mana strategi yang paling menguntungkan.

3.3.2 Efektivitas Strategi Dodge State Bot

Strategi pertama merupakan strategi naif paling efisien yang dapat diterapkan dalam Dodge State Bot. Strategi ini memungkinkan *bot* pemain untuk terhindar dari objek yang membahayakan. Tetapi, probabilitas *bot* pemain terhindar dari objek-objek tersebut sangat kecil, sehingga menurunkan efektivitas strategi ini.

Strategi kedua merupakan strategi trayektori TORPEDO, jika sangat dekat maka *bot* pemain akan mengeluarkan SHIELD. Walaupun sangat efektif, hal ini memberikan kerugian 20 unit kepada ukuran *bot* tetapi dapat membuat *bot* lawan tidak mendapatkan ukuran tambahan. Menurut penulis, strategi yang efisien adalah memperhitungkan trayektori TORPEDO dan memilih strategi mana yang paling memberikan keuntungan pada *tick* saat itu.

3.3.2 Efektivitas Strategi Attack State Bot

Strategi pertama merupakan aim versi 0. Aim ini memberikan heading yang mengarah tepat pada posisi lawan pada tick tersebut. Meskipun terlihat efisien dan efektif, jika aim ditembakkan secara jauh maka akan meleset sehingga dianggap kurang efektif.

Strategi kedua merupakan aim versi 1. Aim ini memberikan HEADING ke arah posisi yang diperkirakan akan dilewati oleh lawan. Strategi ini lebih efektif daripada strategi sebelumnya, karena *bot* dapat memperkirakan kemana arah lawan pada *tick* selanjutnya.

Strategi ketiga merupakan aim versi 2. Aim ini berusaha untuk melakukan *blocking* terhadap lawan yang ada di ujung dunia. Pada tembakan ini, diharapkan lawan akan membelok ke ujung dunia.

Strategi keempat merupakan aim versi 3. Aim ini berusaha untuk melakukan *blocking* lawan yang berada di dekat GAS_CLOUD. GAS_CLOUD memberikan *damage* yang besar bagi *bot*, maka strategi ini dianggap efektif dalam melawan musuh.

Namun, tidak dapat dipungkiri bahwa setiap *aim* dapat meleset. Oleh karena itu, penulis menambahkan faktor koreksi pada perhitungan *aiming* dengan memperkirakan arah belokan lawan.

3.4 Strategi *Greedy* yang Dipilih

Strategi heuristik yang diambil sebagai algoritma utama merupakan penggabungan dari seluruh strategi yang telah dipaparkan pada subbab sebelumnya. Untuk setiap keadaan tertentu, penulis mengambil seluruh strategi dengan harapan *bot* pemain dapat menangani seluruh kemungkinan secara efektif dan efisien. Agar seluruh strategi dapat digabungkan menjadi algoritma *greedy* utama, penulis mendefinisikan prioritas dari strategi-strategi tersebut. Prioritas ditentukan dari pilihan yang memberikan keuntungan terbanyak dan *damage* minimum bagi *bot* pemain. Pada tugas besar ini, digunakan logika dan perhitungan untuk mengecek urutan strategi tersebut.

Setelah prioritas tersebut ditentukan, dicari permutasi dari setiap urutan pengeksekusian strategi dimana strategi memiliki peluang paling besar untuk membuat *bot* pemain bertahan pada permainan. Karena penulis tidak berhasil mencari *bot* pemenang Entelect Challenge 2022 GALAXIO, algoritma *bot* lawan yang digunakan sebagai tolak ukur adalah *ReferenceBot* bawaan, iterasi *bot* penulis sebelumnya, serta *bot* kolega yang juga memanfaatkan strategi *greedy*.

Berikut merupakan urutan prioritas pada *bot* pemain:

- A. Strategi membelokkan HEADING sebesar ± 15 derajat menuju ke titik tengah peta ketika *bot* sudah mencapai ujung dari peta
- B. Strategi ESCAPE_STATE ketika *bot* berada dalam keadaan sangat terancam oleh *bot* lawan dengan ukuran lebih besar
- C. Strategi DODGE_STATE ketika *bot* mendeteksi *bot* lawan yang lebih besar
- D. Strategi DODGE_STATE ketika *bot* menemukan objek yang menjadi ancaman
- E. Strategi ATTACK_STATE ketika *bot* mendeteksi *bot* lawan yang lebih kecil
 1. Strategi teleporter diprioritaskan jika keadaan dan kondisi memenuhi. Kondisi yaitu *bot* pemain memiliki ukuran yang lebih besar dari *bot* lawan dan *bot* pemain memiliki teleporter.

2. Strategi supernova diprioritaskan jika keadaan dan kondisi memenuhi. Kondisi yaitu *bot* lawan berada pada posisi yang jauh dan *bot* pemain memiliki supernova.
3. Strategi AIMV2 digunakan jika *bot* lawan dekat dengan ujung peta dunia
4. Strategi AIMV3 digunakan jika *bot* lawan dekat dengan objek GAS_CLOUD
5. Strategi AIMV1 digunakan

F. Strategi DEFAULT_STATE ketika *bot* tidak mendeteksi ancaman ataupun target

Strategi tersebut disusun sedemikian rupa sehingga diharapkan dapat memenuhi berbagai situasi dan kondisi yang dihadapi oleh *bot* pemain.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy*

```
{ Algoritma Greedy }

{ Algoritma Perpindahan State pada Default }
procedure runState() {
    { Kamus Lokal }
    defaultAction : boolean
    nearestEnemy : gameObject

    { Inisialisasi }
    defaultAction <- true
    playerList <- getGameObjects().stream()
        .filter(getGameObjectType() == ObjectType == PLAYER)
        .sorted(Comparator(getDistanceBetween(self, item))

    { PEMILIHAN PRIORITAS }
    if (not playerList.empty) then
        nearestEnemy < playerList[1] {karena index 0 adalah diri sendiri}
    { PRIORITAS 1 }
        if (supernovaAvailable()) then
            assign(STATETYPES.ATTACK_STATE)
            assign(PlayerActions.FIRESUPERNOVA)
            defaultAction <- false
    { PRIORITAS 2 }
        else if (detectEnemy(self, radarSize) then
            { PRIORITAS 2.1}
            if (enemyisBig()) then
                assign(STATETYPES.ESCAPE_STATE)
                assign(PlayerActions.FORWARD)
                defaultAction <- false
            { PRIORITAS 2.2}
            else if (enemyisSmall()) then
                assign(STATETYPES.ATTACK_STATE)
                assign(PlayerActions.FORWARD)
                defaultAction <- false
            { PRIORITAS 2.1}
            else if (enemyissameSize()) then
                assign(STATETYPES.DEFAULT_STATE)
                assign(PlayerActions.DO NOTHING)
    { PRIORITAS 3 }
        else if (selfisBig)then
            { PRIORITAS 3.1}
            if (self.size - 30 > nearestEnemy.size AND self.Telopoter.count > 0)
                assign(STATETYPES.ATTACK_STATE)
                assign(PlayerActions.FORWARD)
                assign(AMV1(self, nearestEnemy, 20)
    { PRIORITAS 3.2}
        else if (self.TorpedoSalvoCount > 0)
            assign(STATETYPES.ATTACK_STATE)
            assign(PlayerActions.FORWARD)
            assign(AMV1(self, nearestEnemy, 20)
}
```

```

{ Algoritma Default State Bot }

...
if (defaultAction) then
    teleporterPrepped <- false
    supernovaList <- getGameObjects().stream()
        .filter(getGameObjectType() == ObjectType == SUPERNOVA_PICKUP)
        .sorted(Comparator(getDistanceBetween(self, item)))
    output("Supernova count " + supernovaList.size

{ PEMILIHAN PRIORITAS }
if (not supernovaList.empty) then
    if ((getDistanceBetween(supernovaList[0], self) < 300 AND self.size > 50)
        OR (getDistanceBetween(supernovaList[0], self) < 50))
{ PRIORITAS 1}
    assign(getHeadingBetween(supernovaList[0], self)
{ PRIORITAS 2}
    else then hoardingFood()
else then hoardingFood()

...

```

```

{ Algoritma Pencarian Makanan }
procedure hoardingFood(){
    { Kamus Lokal }
    foodList : List
    sizeSelf : double
    newHeading, xAwal, yAwal : integer
    i : integer

    { Inisialisasi }
    sizeSelf <- self.size()
    foodThreshold <- 0
    foodList <- getGameObjects().stream()
        .filter(getGameObjectType() == ObjectType == FOOD)
        .sorted(Comparator(getDistanceBetween(self, item)))

    { Pilihlah FOOD dengan jarak terdekat dari bot (greedy by distance) }
    if ( not foodList.isEmpty()) then
        newHeading <- getHeadingBetween(foodList[0], self)
        xAwal <- (foodList[0].position.x)
        yAwal <- (foodList[0].position.y)
        Position awal <- new Position(xAwal, yAwal)

        { Pilihlah posisi baru terdekat dari posisi yang telah dipilih sebelumnya }
        { Pada tahap ini, diharapkan mendapatkan banyak FOOD dalam satu kalo jalan }
        { Greedy by Value }
        traversal i [0..foodList.size]
            if (getDistanceBetween(awal,foodList[i]) < (sizeSelf - foodThreshold)) then
                xAwal <- (foodList[0].position.x + xAwal) / 2
                yAwal <- (foodList[0].position.y + yAwal) / 2
                awal.setX(xAwal)
                awal.setY(yAwal)
            else then break
        Position tujuan <- new Position(xAwal, yAwal)
        newHeading <- getHeadingBetween(tujuan, self)

```

```

{ Algoritma Dodge State Bot }

...
    if (defaultAction) then
        dodging <- false

    { PEMILIHAN PRIORITAS }
    if (critical AND shieldCount > 0 AND size > 25 AND torpedoList.size > 2) then
        assign(ACTIVATESHIELD)
    else then
        if (not outsideBound() and self.size < 100) then
            if (NavigateDodging) then
                if (dodging) then
                    cachedDirection <- decideTurnDir(getHeading(), self, gameState)
                    cachedHeading <- getHeading()
                    if (cachedDirection = 1) then
                        assign(cachedHeading + 300) % 360
                    else then
                        assign(cachedHeading + 60) % 360
                    dodging <- true
                else then
                    if (cachedDirection = 1) then
                        assign(cachedHeading + 300) % 360
                    else then
                        assign(cachedHeading + 60) % 360
                assign(PlayerActions.FORWARD)
            if (not hit) then
                assign(STATETYPES.DEFAULT_STATE)
                dodging <- false
            else then
                assign(PlayerActions.FORWARD)
        else then
            assign(PlayerActions.FORWARD)
    ...
}

{ Algoritma Perpindahan State Escape State Bot }

...
{ PEMILIHAN PRIORITAS }
{ Strategi Greedy Attack }

{ PRIORITAS 1: ENEMY OUT OF SIGHT}
if (not (detectEnemy(playerList[1], self, radarsize) then
    assign(STATETYPES.DEFAULT_STATE)
    assign(PlayerActions.FORWARD)

{ PRIORITAS 2: ENEMY IS SMALL}
else if (isSmall(playerList[1], self, radarsize) then
    assign(STATETYPES.ATTACK_STATE)
    assign(PlayerActions.FORWARD)

{ PRIORITAS 3: FIRE TORPEDO }
{ FIRE TORPEDO}
if (self.TorpedoSalvoCount > 0 AND self.size > tpdsizeTH)
    assign(STATETYPES.ATTACK_STATE)
    assign(PlayerActions.FORWARD)
    assign(AIMV1(self, nearestEnemy, 20))

{PRIORITAS 4: DEFAULT ACTION}
else
    defaultAction(enemyDirection)

```

```

{ Algoritma Escape State Bot }

procedure defaultAction (enemyDirection: integer){
    { Kamus Lokal }
    notfoundfood : boolean
    closestFoodDirection : integer
    i : integer

    { Inisialisasi }
    notfoundfood <- true
    i <- 0
    foodList <- getGameObjects().stream()
        .filter(getGameObjectType() == ObjectType == FOOD)
        .sorted(Comparator(getDistanceBetween(self, item)))

    { Escape dan mencari makanan secara Greedy By Distance }
    if (not foodList.empty()) then
        while (not foodfound AND i<MIN(10, foodList.size)) do
            closestFoodDirection <- getHeadingBetween(foodList[i], self)
            if(aroundDegrees(closestFoodDirection, enemyDirection)) then
                notfoundfood <- false
                assign(closestFoodDirection)
            else then i = i + 1
    { Jika tidak ada makanan, maka melarikan diri secara efektif atau 180 derajat }
    if (not foundfood) then
        assign(enemydirection +180)
    assign(PlayerActions.FORWARD)
}

{ Algoritma Attack State Bot }

...
{ PEMILIHAN PRIORITAS }
{ Strategi Greedy Attack }

{ PRIORITAS 1: FIRING TELEPORTER }
if (teleporterPrepped) then
    assign(STATETYPES.ATTACK_STATE)
    assign(PlayerActions.FIRETELEPORT)

{PRIORITAS 2: JIKA UKURAN BOT BESAR}
if (self.isBig) then
    {PRIORITAS 2.1. PREP TELEPORT}
    if (self.size - 30 > nearestEnemy.size AND self.Teleporter.count > 0)
        assign(STATETYPES.ATTACK_STATE)
        assign(PlayerActions.FORWARD)
        assign(AIMV1(self, nearestEnemy, 20))
    {PRIORITAS 2.2. FIRE TORPEDO}
    else if (self.TorpedoSalvoCount > 0)
        assign(STATETYPES.ATTACK_STATE)
        assign(PlayerActions.FORWARD)
        assign(AIMV1(self, nearestEnemy, 20))
    {PRIORITAS 2.3. GO TO DEFAULT}
    else then assign(STATETYPES.DEFAULT_STATE)

{PRIORITAS 3: JIKA UKURAN BOT KECIL}
else then

    {PRIORITAS 3.1. ENEMY SUDAH JAUH}
    if (attackRange(self, nearestEnemy) == 4 then

```

```

assign(STATETYPES.DEFAULT_STATE)
assign(PlayerActions.FORWARD)

{PRIORITY 3.2. MEMILIKI SUPERNOVA}
else if (supernovaavailable > 0) then
    { Greedy by value dengan mencari Enemy terBesar }
    traversal i [1..playerList.size]
        nearestEnemy <- playerList[i]
        if(attackRange(self, nearestEnemy) >= 2) then
            assign(STATETYPES.ATTACK_STATE)
            assign(PlayerActions.FIRESUPERNOVA)

{PRIORITY 3.3. JIKA ENEMY TERLALU KECIL}
else if (attackRange(self, nearestEnemy) == 1 then
    {FIRE TELEPORT}
    if (self.size - 30 > nearestEnemy.size AND self.Teleporter.count > 0)
        assign(STATETYPES.ATTACK_STATE)
        assign(PlayerActions.FORWARD)
        assign(AIMV1(self, nearestEnemy, 20)
    {FIRE TORPEDO}
    else if (self.TorpedoSalvoCount > 0)
        assign(STATETYPES.ATTACK_STATE)
        assign(PlayerActions.FORWARD)
        assign(AIMV1(self, nearestEnemy, 20)
    {GO TO DEFAULT}
    else then
        assign(STATETYPES.DEFAULT_STATE)
        assign(PlayerActions.FORWARD)

{PRIORITY 3.4. ENEMY IS BIGGER}
else if (isBig(nearestEnemy, self+10) then
    if(attackRange(self, nearestEnemy) >= 2) then
        {FIRE TORPEDO}
        if (self.TorpedoSalvoCount > 0)
            assign(STATETYPES.ATTACK_STATE)
            assign(PlayerActions.FORWARD)
            assign(AIMV1(self, nearestEnemy, 20)
        {GO TO DEFAULT}
        else then
            assign(STATETYPES.DEFAULT_STATE)
            assign(PlayerActions.FORWARD)
    else { ESCAPE }
        assign(STATETYPES.ESCAPE_STATE)
        assign(PlayerActions.FORWARD)

{PRIORITY 3.5. ENEMY IS SMALLER}
else if (isSmall(nearestEnemy, self) then
    if(attackRange(self, nearestEnemy) >= 2) then
        {FIRE TELEPORT}
        if (self.size - 30 > nearestEnemy.size AND self.Teleporter.count > 0)
            assign(STATETYPES.ATTACK_STATE)
            assign(PlayerActions.FORWARD)
            assign(AIMV1(self, nearestEnemy, 20)
    else then
        switch(attackRange(self, nearestEnemy)
            case 1:
                {FIRE TORPEDO}
                if (self.TorpedoSalvoCount > 0 AND self.size > tpdsizeth)
                    assign(STATETYPES.ATTACK_STATE)
                    assign(PlayerActions.FORWARD)

```

```

        assign(AIMV1(self, nearestEnemy, 20)
else
    defaultAction(fixAim)

case 2:
{FIRE TORPEDO}
if (self.TorpedoSalvoCount > 0 AND self.size > tpdsizETH)
    if(outsideBound(nearestEnemy) AND aimv2 != 9000 then
        fixAim <- aimv2(self, nearestEnemy, nearestGasCloud)
    else if aimv3(self, nearestEnemy, nearestCloud) != -9999 then
        fixAim <- aimv3(self, nearestEnemy, nearestGasCloud)
    else then
        fixAim <- aimv1(self, nearestEnemy, nearestGasCloud)
    else then
        defaultAction(fixAim)
else
    assign(STATETYPES.DEFAULT_STATE)
    assign(PlayerActions.FORWARD)
case 3:
{FIRE TORPEDO}
if (self.TorpedoSalvoCount > 0 AND self.size > tpdsizETH)
    fixAim <- aimv1(self, nearestEnemy, nearestGasCloud)
else
    assign(STATETYPES.DEFAULT_STATE)
    assign(PlayerActions.FORWARD)
default:
    assign(STATETYPES.DEFAULT_STATE)
    assign(PlayerActions.FORWARD)

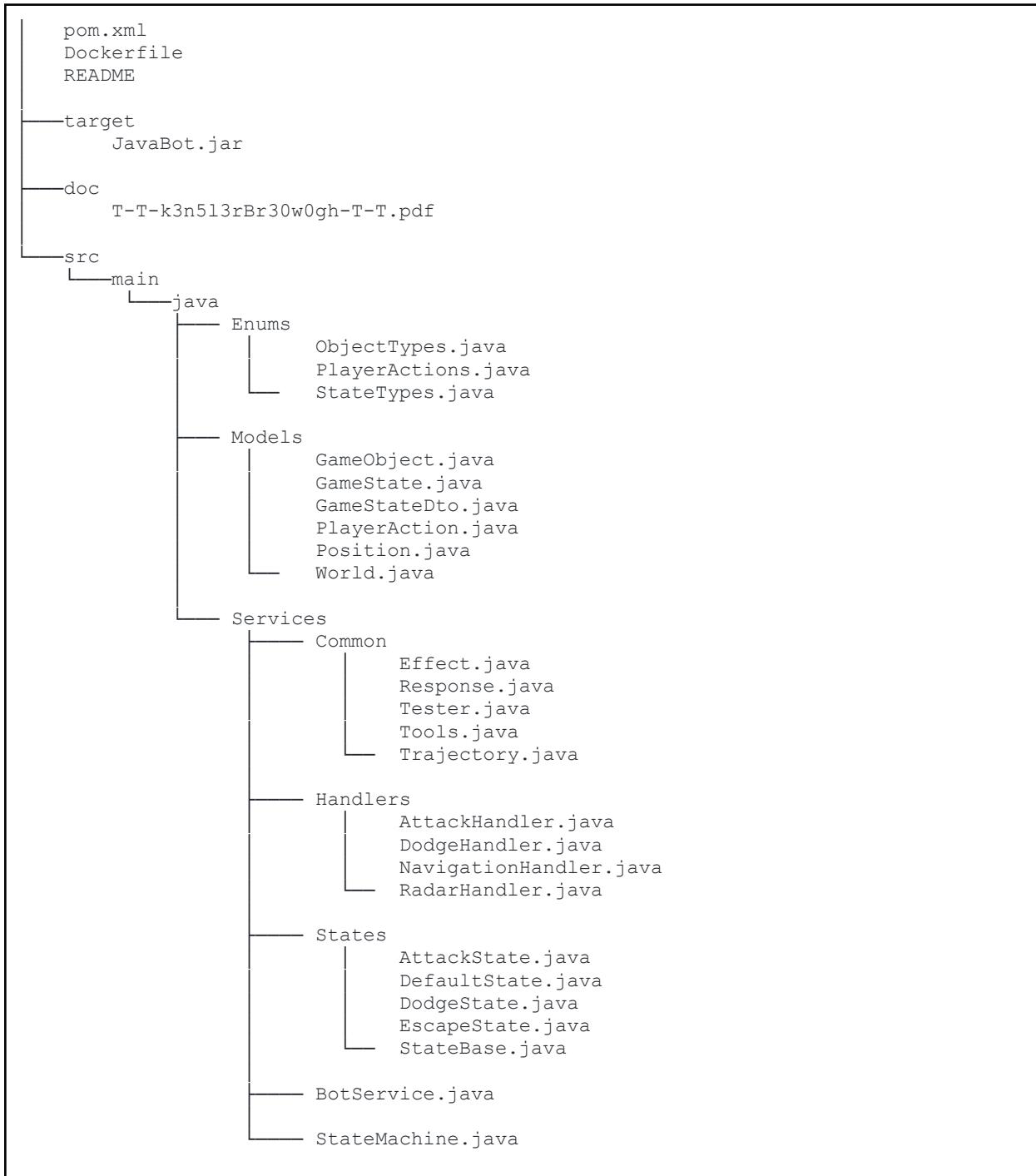
{PRIORITAS 3.6. SIZE NANGGUNG}
{FIRE TELEPORT}
if (self.size - 30 > nearestEnemy.size AND self.Teleporter.count > 0)
    assign(STATETYPES.ATTACK_STATE)
    assign(PlayerActions.FORWARD)
{GO TO DEFAULT}
else then
    assign(STATETYPES.DEFAULT STATE)
    assign(PlayerActions.FORWARD)

...

```

4.2 Struktur Data yang Digunakan

Dalam tugas besar ini, terdapat 3 folder utama dan 1 file utama: Enums, Models, Services, dan Main.java. Berikut merupakan rincian serta deskripsi (atribut dan metode) dari setiap folder tersebut.



4.2.1. Folder Enums

Folder ini berisi file yang memuat enumerasi yang dapat memudahkan kami dalam mengacu pada suatu indeks tertentu. Pada file in folder ini tidak terdapat atribut dan *method* karena hanya berisi enumerasi.

4.2.2. Folder Models

Folder Models berisi file yang memuat perkakas pembantu pemodelan *bot* pada permainan Galaxio. Folder ini berisikan 6 file. Pada Tugas Besar ini, perubahan pada folder ini hanya menambahkan atribut pada GameObject.java karena terdapat kekurangan atribut.

4.2.3. Folder Services

4.2.3.1. Folder Common

Folder ini berisi tools dan utilities yang digunakan secara umum pada pengembangan bot. Pada folder ini terdapat 5 file, yaitu Response.java, Tools.java, Effects.java, Tester.java dan Trajectory.java. Kelas Response merupakan kelas yang dibuat untuk memudahkan mengirimkan response ke bot. Response ini berupa aksi yang akan *bot* lakukan serta dalam keadaan apa bot tersebut berada (contohnya adalah ATTACK_STATE).

Tabel 4.2.3.1.1. Class Response

Nama Class : Response	
ATRIBUT	
newState	Tipe <u>StateTypes</u> yang dibuat untuk menyimpan keadaan baru pada bot.
newAction	Tipe <u>PlayerAction</u> yang dibuat untuk menyimpan aksi yang akan dilakukan bot.
METHOD	
public Response()	Metode konstruktor untuk kelas Response.
public void assign (StateTypes state, PlayerAction action)	Metode untuk menetapkan atribut pada kelas Response berupa keadaan dan aksi bot.
public void assign (StateTypes state)	Metode untuk menetapkan atribut keadaan pada kelas Response (overload).
public void assign (PlayerAction action)	Metode untuk menetapkan atribut aksi pada kelas Response (overload).
public void assign	Metode untuk menetapkan nilai HEADING

(int direction)	sebagai newAction pada kelas Response (overload).
public void assign (PlayerAction actions)	Metode untuk menetapkan atribut aksi lebih dari satu pada kelas Response (overload).
public PlayerAction getNewAction()	Metode untuk mengembalikan nilai PlayerAction yang mendeskripsikan aksi bot yang sedang berjalan.
public StateTypes getState()	Metode untuk mengembalikan StateTypes yang mendeskripsikan keadaan bot saat ini.
public int getHeading()	Metode untuk mengembalikan nilai HEADING bot yang merupakan arah bot tersebut dalam game ($0 < \text{HEADING} < 360$).

Kelas Tools merupakan kelas yang digunakan untuk menyimpan metode-metode yang digunakan sebagai alat dalam pengembangan bot. Kelas tool tidak memiliki atribut sendiri.

Tabel 4.2.3.1.2. Class Tools

Nama Class : Tools	
ATRIBUT	
-	-
METHOD	
public static double getDistanceBetween (GameObject object1, GameObject object2)	Metode yang digunakan untuk mengembalikan jarak antara 2 objek.
public static double getDistanceBetween (Position pos1, Position pos2)	Metode yang digunakan untuk mengembalikan jarak antara 2 posisi (overload).
public static int getHeadingBetween (Position pos1, Position pos 2)	Metode yang digunakan untuk mengembalikan selisih heading antar 2 objek.
public static int toDegree(double v)	Metode yang mengembalikan nilai sudut berupa derajat (semula radian).
public static boolean aroundDegrees (int currentDegree, int targetDegree, int range)	Metode yang digunakan untuk mendapatkan jangkauan derajat arah berdasarkan target.
public static double getXbyDistance (int heading, Double distance, gameObject obj)	Metode yang digunakan untuk mendapatkan posisi X dari jarak antara objek.
public static double getYbyDistance (int heading, Double distance, gameObject obj)	Metode yang digunakan untuk mendapatkan posisi X dari jarak antara objek.

Kelas Trajectory merupakan kelas berupa trayektori dari *bot* yang digunakan untuk meramalkan serta memperhitungkan arah *bot* jika ingin mencapai target tertentu selama permainan.

Tabel 4.2.3.1.3. Class Trajectory

Nama Class : Trajectory	
ATRIBUT	
gradient	tipe long yang menyimpan gradien
constant	tipe long yang menyimpan konstanta
vel	tipe long yang menyimpan kecepatan
vy	tipe double yang menyimpan kecepatan pada sumbu-y
vx	tipe double yang menyimpan kecepatan pada sumbu-x
tetha	tipe int yang menyimpan sudut
init	tipe <u>Position</u> yang menyimpan posisi awal
METHOD	
public Trajectory (GameObject object)	Metode konstruktor untuk kelas Trajectory
public Trajectory (long gradient, long constant)	Metode konstruktor dari gradien dan konstanta, atribut lain dianggap 0 (overload).
public Position interpolate (int time)	Metode yang digunakan untuk interpolasi posisi berdasarkan waktu dan kecepatan objek.

Kelas Effect merupakan kelas berupa efek yang dikenakan pada *bot* pemain. Kelas ini dibuat untuk memudahkan dalam mendekripsikan enkripsi yang sudah diberikan oleh *Game Engine*.

Tabel 4.2.3.1.4. Class Effect

Nama Class : Effect	
ATRIBUT	
decoded	String yang berguna untuk menyimpan hasil dekripsi efek yang dikenakan pada <i>bot</i> pemain
METHOD	
public Effect(int hashed)	Metode yang digunakan untuk mendekripsi efek yang dikenakan pada <i>bot</i> .
public boolean isAftBurn()	Metode yang mengembalikan true apabila <i>bot</i>

	memiliki efek afterburner
public boolean isAstField()	Metode yang mengembalikan true apabila <i>bot</i> terkena efek asteroid field
public boolean isGasCld()	Metode yang mengembalikan true apabila <i>bot</i> terkena efek gas cloud
public boolean isSupFood()	Metode yang mengembalikan true apabila <i>bot</i> terkena efek superfood
public boolean isShield()	Metode yang mengembalikan true apabila <i>bot</i>

Class Tester merupakan kelas yang diperuntukkan untuk *debugging* dan menganalisis keberjalanannya *bot*.

Tabel 4.2.3.1.5. Class Tester

Nama Class : Tester	
ATRIBUT	
-	-
METHOD	
appendFile(String line, String filename)	Metode yang digunakan untuk mengeprint kondisi pada <i>bot</i> dan menaruh hasil tersebut dalam file .txt. Tujuannya adalah agar penulis mampu menganalisis dengan presisi aksi yang dilakukan <i>bot</i> pada suatu tick tertentu.

4.2.3.2. Folder Handlers

Folder ini berisi *handlers* yang merupakan kelas yang menangani suatu keadaan spesifik. Pada folder ini terdapat 4 file: DodgeHandler.java, NavigationHandler.java, RadarHandler.java, dan AttackHandler.java. Kelas DodgeHandler merupakan kelas yang dibuat khusus untuk penanganan kasus menghindari suatu objek.

Tabel 4.2.3.2.1. Class DodgeHandler

Nama Class : DodgeHandler	
ATRIBUT	
critical	Boolean yang menyimpan apakah ada ancaman yang dekat di teritori atau trayektorinya menuju <i>bot</i> .
dodging	Boolean yang menyimpan apakah <i>bot</i> sedang melakukan manuver penghindaran terhadap torpedo.

hit	Boolean yang menyimpan apakah bot berada pada lintasan sebuah torpedo.
METHOD	
public static Position intercept (Trajectory line1, Trajectory line2)	Metode yang digunakan untuk memperkirakan sebuah trayektori dan membuat titik potong terhadap trayektori tersebut.
public static int timeToIntercept (Trajectory line1, Trajectory line2)	Metode yang digunakan untuk mengembalikan perkiraan waktu (dalam ticks) yang harus ditempuh untuk membuat titik potong terhadap trayektori tersebut.
public static double closestDistance (Trajectory line1, Trajectory line2)	Metode yang digunakan untuk mengembalikan jarak terdekat antara dua trayektori.
public static double distanceAfterTime (Trajectory line1, Trajectory line2, int time)	Metode yang digunakan untuk mengembalikan perkiraan jarak berdasarkan masukan trayektori dan waktu.
public static boolean inTrajectory (GameObject bot, List<GameObject> torpedoList)	Metode yang digunakan untuk mengembalikan true jika terdeteksi sebuah tornado berada dalam radar inTrajectory.

Kelas NavigationHandler merupakan kelas yang dibuat khusus untuk penanganan kasus navigasi objek, terutama objek *bot*.

Tabel 4.2.3.2.2. Class NavigationHandler

Nama Class : NavigationHandler	
ATRIBUT	
cachedHeading	integer yang menyimpan heading dari bot
dodging	boolean yang menyimpan apakah sedang menghindari objek atau tidak
temp	integer yang menyimpan konfirmasi arah yang dituju (kanan atau kiri)
cachedObject	Tipe GameObject yang sedang ditangani oleh <i>bot</i> .
METHOD	
public static boolean outsideBound (GameState gameState, GameObject obj)	Metode yang digunakan untuk mengembalikan true jika objek bot berada di ujung world.
public static boolean outsideBound (GameState gameState, Position toBe, GameObject obj)	Metode yang digunakan untuk mengembalikan true jika objek bot berada di ujung world

	(overload).
public static int decideTurnDir (int currentHeading, GameObject obj)	Metode yang mengembalikan integer yang mendeskripsikan arah yang dituju bot (kanan atau kiri)
public static boolean decideTurnDir (int currentHeading, GameObject self, GameObject avoidObj)	Metode yang mengembalikan boolean yang mendeskripsikan arah yang dituju bot (kanan atau kiri) khusus jika ada objek yang dihindari (overload).
public static StateTypes dodgeEnemy()	Metode yang digunakan untuk menghindari bot lain dengan ukuran lebih besar daripada bot sendiri.
public static int dodgeObjects(int currentHeading, GameState gameState, GameObject self, List <GameObject> objectList)	Metode yang digunakan untuk menghindari objek-objek yang berada dalam List objectList.
public static int dodgeEdge (GameObject bot, GameState gameState)	Metode yang digunakan untuk menghindari ujung world.

Kelas RadarHandler merupakan kelas yang dibuat khusus untuk penanganan radar pada objek *bot*.

Tabel 4.2.3.2.3. Class RadarHandler

Nama Class : RadarHandler	
ATRIBUT	
-	-
METHOD	
public static boolean detectEnemy (GameObject otherObject, GameObject bot, Double size)	Metode yang mengembalikan true jika terdeteksi <i>bot</i> lain, terlepas dari ukuran bot tersebut.
public static boolean detectThreat (GameState gameState, GameObject bot, Double size)	Metode yang mengembalikan true jika terdeteksi enemy di dalam radar yang ukurannya lebih besar daripada <i>bot</i> pemain.
public static boolean isSmall (GameObject otherObject, Double threshold)	Metode yang mengembalikan true jika GameObject berukuran lebih kecil daripada <i>bot</i> pemain dengan ambang sebesar <i>threshold</i> .
public static boolean isBig (GameObject otherObject, Double threshold)	Metode yang mengembalikan true jika GameObject berukuran lebih besar daripada <i>bot</i> pemain dengan ambang sebesar <i>threshold</i> .
public static boolean detectSupernova (GameObject otherObject, Double threshold)	Metode yang mengembalikan true jika terdapat SUPERNOVA_PICKUP dan berada di dalam radar sebesar <i>threshold</i> .

Kelas AttackHandler merupakan kelas yang dibuat khusus untuk penanganan penyerangan terhadap *bot* lainnya.

Tabel 4.2.3.2.4. Class AttackHandler

Nama Class : NavigationHandler	
ATRIBUT	
METHOD	
longRange	integer yang menyimpan jarak jauh.
midRange	integer yang menyimpan jarak tengah.
closeRange	integer yang menyimpan jarak dekat.
supernovaFired	boolean yang menyimpan true jika supernova sudah ditembakkan.
supernovaEmpty	boolean yang menyimpan true jika supernova sudah kosong.
teleporterFired	boolean yang menyimpan true jika teleporter sudah ditembakkan.
teleporterPrepped	boolean yang menyimpan true jika teleporter sudah siapkan.
teleporterEmpty	boolean yang menyimpan true jika teleporter tidak digunakan dan peta tidak memiliki teleporter aktif.
teleporterDelay	Integer yang menyimpan tick setelah aksi penembakan teleporter.
bigSize	Integer yang menyimpan <i>threshold</i> ukuran besar suatu bot.
smallSize	Integer yang menyimpan <i>threshold</i> ukuran kecil suatu bot.
public static double realDist (GameObject o1, GameObject o2)	Metode yang mengembalikan jarak diantara dua objek dengan tipe double.
public static double realDist (Position p1, Position p2, int r1, int r2)	Metode yang mengembalikan jarak diantara dua objek dengan tipe double (overload).
public static int detAttackRange (GameObject o1, GameObject o2)	Metode yang mengembalikan integer yang mendeskripsikan strategi penyerangan yang akan dipilih oleh <i>bot</i>

public static int aimv0 (GameObject self, GameObject enemy)	Metode penyerangan yang dasar, bidikan hanya berdasarkan mengembalikan HEADING antar objek.
public static int aimv1 (GameObject self, GameObject enemy, int attackSpeed)	Metode penyerangan naive, bidikan berdasarkan prediksi kecepatan objek yang ditarget.
public static int aimv2 (GameObject self, GameObject enemy)	Metode penyerangan apotema, bidikan berdasarkan prediksi garis potong dan prediksi gerakan target.
public static int aimv3 (GameObject self, GameObject enemy, GameObject danger)	Metode penyerangan bidikan pembelok, bidikan memiliki tujuan untuk membelokkan musuh keujung peta dunia.
public static int attackFood (GameState gameState, GameObject self, GameObject enemy)	Metode penyerangan sambil mencari makanan.

4.2.3.4. Folder States

Folder ini berisi *handlers* yang merupakan kelas yang menangani suatu keadaan spesifik. Pada folder ini terdapat 4 file: AttackState.java, DefaultState.java, DodgeState.java, dan EscapeState.java. Kelas AttackState merupakan kelas yang menangani navigasi pada bot.

Tabel 4.2.3.4.1. Class AttackState

Nama Class : AttackState	
ATRIBUT	
-	-
METHOD	
public static Response runState()	Metode yang digunakan untuk mengembalikan Response berisikan aksi dan HEADING yang dituju oleh <i>bot</i>
public static void defaultAction (int enemyDirection)	Metode yang digunakan untuk determinasi aksi <i>default</i> yang dilakukan bot pada keadaan ini
public static void detectSupernova()	Metode yang digunakan untuk mendeteksi supernova yang berada di dalam radar <i>bot</i>
public static void detectTeleporter()	Metode yang digunakan untuk mendeteksi teleporter yang berada di dalam radar <i>bot</i>

Class defaultState merupakan kelas yang menangani keadaan *default* pada *bot*. Pada keadaan ini, *bot* memiliki fokus utama untuk mencari makanan dan terhindar dari semua rintangan yang ada di dalam peta.

Tabel 4.2.3.4.2. Class DefaultState

Nama Class : DefaultState	
ATRIBUT	
-	-
METHOD	
public static Response runState()	Metode yang digunakan untuk mengembalikan Response berisikan aksi dan HEADING yang dituju oleh <i>bot</i>
public static void hoardingFood()	Metode yang digunakan untuk mencari makanan pada peta. Pada metode ini diharapkan dapat mencari makanan secara optimum.

Class dodgeState merupakan kelas yang menangani keadaan dimana *bot* harus menghindar. Pada keadaan ini, *bot* berfokus dalam mencari jalur yang optimal dalam menghindari objek.

Tabel 4.2.3.4.3. Class dodgeState

Nama Class : DodgeState	
ATRIBUT	
dodging	boolean yang menyimpan true apabila <i>bot</i> sedang menghindari suatu objek tertentu.
cachedHeading	integer yang menyimpan HEADING <i>bot</i>
METHOD	
public static Response runState()	Metode yang digunakan untuk mengembalikan Response berisikan aksi dan HEADING yang dituju oleh <i>bot</i>
public static Response detectTorpedoes()	Metode yang digunakan untuk mendeteksi apakah ada torpedo yang mendekati <i>bot</i> pemain

Class escapeState merupakan kelas yang menangani keadaan dimana *bot* harus milarikan diri. Pada keadaan ini, *bot* berfokus dalam mencari jalur yang optimal dalam milarikan diri dari musuh.

Tabel 4.2.3.4.4. Class escapeState

Nama Class : escapeState	
ATRIBUT	
-	-
METHOD	

<code>public static Response runState()</code>	Metode yang digunakan untuk mengembalikan Response berisikan aksi dan HEADING yang dituju oleh <i>bot</i> .
<code>public static void defaultAction(int enemyDirection)</code>	Metode yang digunakan untuk mencari jalur optimum dalam melarikan diri dari musuh.

Class stateBase merupakan kelas yang menangani keadaan prioritas utama *bot*. Pada keadaan ini, *bot* berfokus dalam mencari jalur yang optimal dalam menghadapi keadaan.

Tabel 4.2.3.4.5. Class escapeState

Nama Class : escapeState	
ATRIBUT	
<code>gameState</code>	Tipe <u>GameState</u> yang menyimpan keadaan <i>bot</i>
<code>self</code>	Tipe <u>GameObject</u> yang menyimpan object <i>bot</i>
<code>currentAction</code>	Tipe <u>PlayerAction</u> yang menyimpan aksi <i>bot</i>
<code>retval</code>	Tipe <u>Response</u> yang menyimpan respon <i>bot</i>
<code>radarsize</code>	double yang menyimpan ukuran radar <i>bot</i>
<code>smallThreshold</code>	double yang menyimpan threshold <i>bot</i>
METHOD	
<code>public static void updateState (GameState newGameState, GameObject bot, PlayerAction action)</code>	Metode yang digunakan untuk memperbarui keadaan <i>bot</i> sehingga berubah ke keadaan lainnya
<code>public static void pathfind (int currentHeading)</code>	Metode yang digunakan untuk mencari jalur optimum dalam pergerakan <i>bot</i>
<code>public static void fireTorpedoes (int direction)</code>	Metode yang digunakan untuk menembak torpedo berdasarkan arah <i>bot</i>

4.2.2.5. Class BotService

Class botService pada folder ini merupakan kelas bawaan yang memiliki tujuan untuk mengkomputasi hal-hal yang berkaitan dengan *fetching* data pada *bot*. Ada tiga atribut yang diolah pada botService, yaitu GameObject bot, PlayerAction, dan GameState.

4.2.2.6. Class State Machine

Class stateMachine pada folder ini merupakan kelas yang memiliki tujuan untuk mengubah STATETYPES yang berkaitan dengan perubahan keadaan pada *bot*.

Tabel 4.2.2.3.2. Class stateMachine

Nama Class : StateMachine	
ATRIBUT	
CURRENTSTATE	StateTypes yang menyimpan keadaan <i>bot</i>
response	Response pada <i>bot</i> , termasuk COMMAND dan HEADING
METHOD	
public void changeState (StateTypes NewState)	Metode yang mengubah <i>state</i> ke keadaan baru
public PlayerAction determineAction (GameState gameState, PlayerAction currentAction, GameObject self)	Metode yang mendeterminasi aksi yang dilakukan berdasarkan keadaan saat ini.

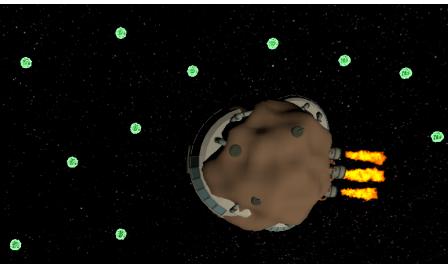
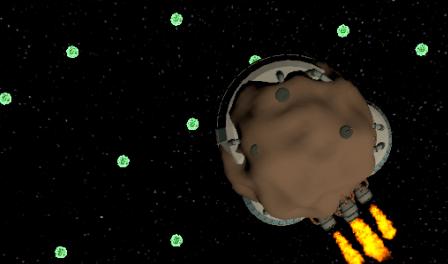
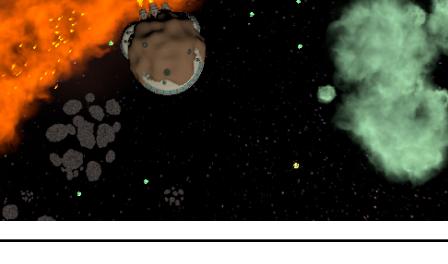
4.2.3. Main.java

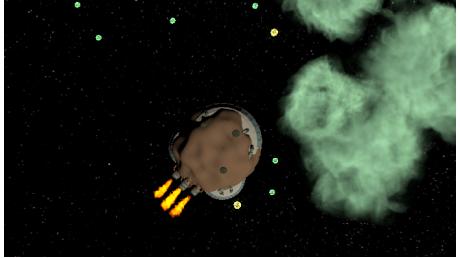
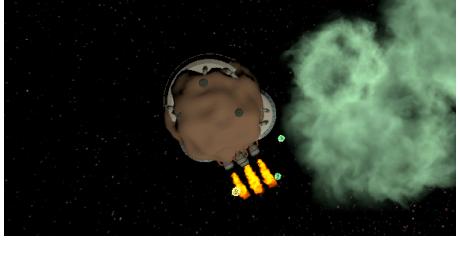
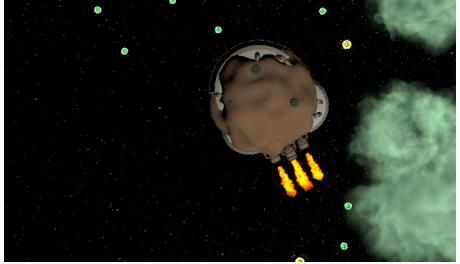
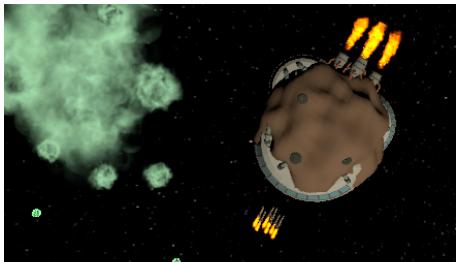
File ini berisi keberjalanannya *bot* secara keseluruhan. Pertama, dilakukan sambungan koneksi ke server untuk menyambungkan koneksi *bot* ke *game engine*. Setelah *bot* tersambung, dilakukan *fetching* determinasi aksi yang kemudian diarahkan ke Game Engine.

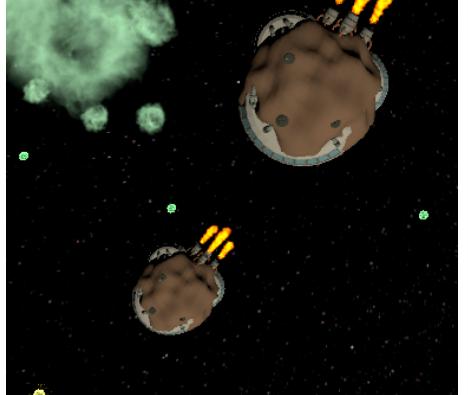
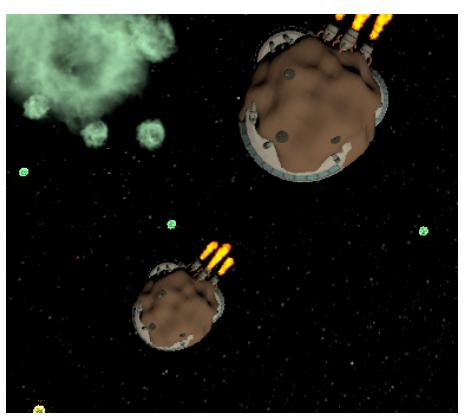
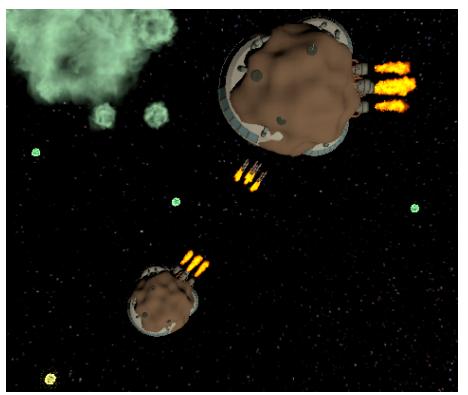
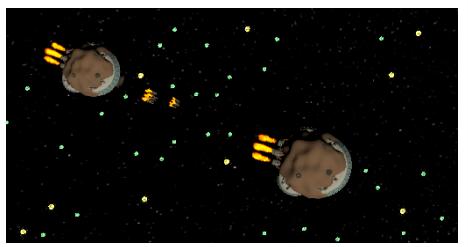
4.3. Eksperimen Hasil Implementasi

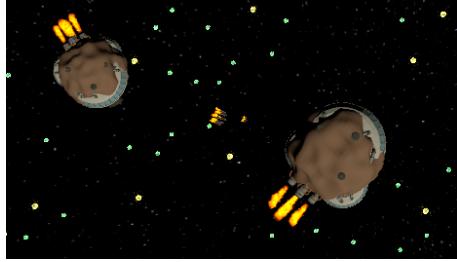
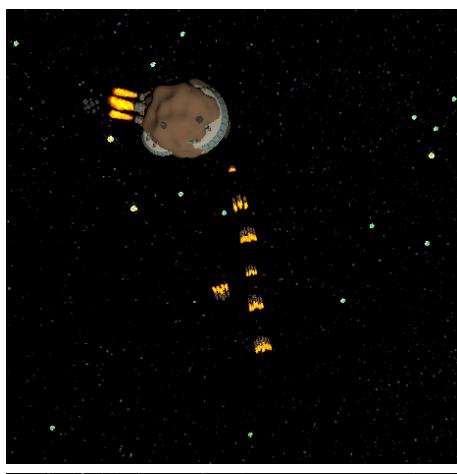
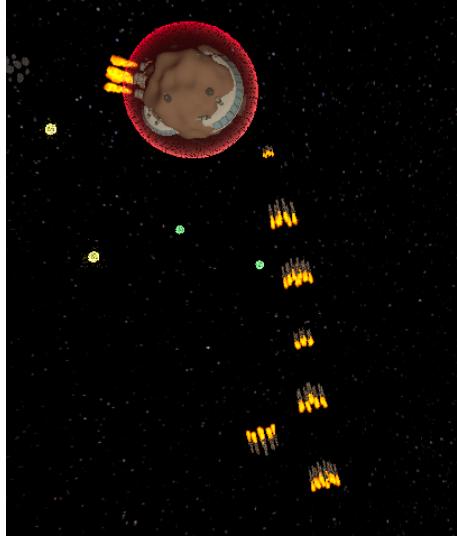
Berikut merupakan hasil implementasi dari beberapa kasus *bot* dalam permainan Galaxio.

Tabel 4.3.1. Eksperimen Kasus

Kasus	Screenshot	Analisis
<i>Bot</i> pemain mencari makanan ke makanan terdekat.	 	<p><i>Bot</i> mencari makanan dengan menghitung titik tengah dengan makanan lain yang masih berjarak kurang dari ukuran <i>bot</i> saat ini.</p> <p>Gambar pertama menunjukkan <i>bot</i> baru mengambil makanan.</p> <p>Gambar kedua menunjukkan <i>bot</i> mengarah pada titik tengah antara makanan terdekat dengan yang masih berjarak kurang dari ukuran <i>bot</i> saat ini.</p>
<i>Bot</i> pemain menghindari Edge World pada peta.	 	<p><i>Bot</i> akan melakukan perubahan haluan 180 derajat ± 15 dengan mengarah berlawanan terhadap <i>gas cloud</i> atau <i>bot</i> lain yang berukuran besar terdekat.</p> <p>Gambar pertama menunjukkan <i>bot</i> tengah mencari makanan pada ujung peta.</p> <p>Gambar kedua menunjukkan <i>bot</i> mengarah ke tengah dan cenderung menjauhi <i>gas cloud</i> terdekat.</p>

<p>Bot pemain menghindari <i>wormhole</i> dan <i>gas cloud</i> pada peta.</p>	  	<p>Bot akan melakukan manuver berupa gerakan berputar sampai terdapat makanan dengan arah menjauhi <i>wormhole</i> ataupun <i>gas cloud</i> atau sudah melewati objek-objek tersebut. Arah pembelokan ditentukan terhadap pusat peta.</p> <p>Gambar pertama menunjukkan bot hendak mengarah pada <i>gas cloud</i>.</p> <p>Gambar kedua menunjukkan bot mengubah haluan menjadi berputar pada <i>gas cloud</i> tersebut.</p> <p>Gambar ketiga menunjukkan bot berhenti berputar dan kembali mengarah pada makanan.</p>
<p>Bot pemain akan menjauhi bot lain yang berukuran lebih besar yang mendekati bot pemain.</p>		<p>Bot akan menjauhi bot lain yang berukuran lebih besar dengan bergerak 180 derajat terhadap posisi lawan sampai jarak tertentu sudah dipenuhi.</p> <p>Gambar pertama menunjukkan sebuah bot didekati oleh bot yang lebih besar.</p> <p>Gambar kedua menunjukkan bot kecil mengubah haluan menjauhi bot besar tersebut.</p>

		
<i>Bot</i> pemain akan menembak torpedo kepada <i>bot</i> yang lebih besar ketika <i>bot</i> yang lebih besar tersebut mendekat.		<i>Bot</i> akan menembak torpedo pada <i>bot</i> yang lebih besar jika <i>bot</i> tersebut didekati. Gambar pertama menunjukkan sebuah <i>bot</i> yang tengah didekati <i>bot</i> lain yang lebih besar
		Gambar kedua menunjukkan <i>bot</i> tersebut menembakkan torpedo pada arah <i>bot</i> yang lebih besar.
<i>Bot</i> pemain akan menghindari torpedo yang akan mengenai <i>bot</i> pemain.		Jika interpolasi torpedo menghasilkan kena dan tidak memenuhi kondisi untuk menyalakan shield maka <i>bot</i> akan melakukan perubahan haluan sebesar ± 60 dengan mengarah pada pusat peta. Gambar pertama menunjukkan dua buah torpedo yang mengarah pada <i>bot</i> yang di sebelah kanan.

		Gambar kedua menunjukkan <i>bot</i> yang di sebelah kanan melakukan perubahan haluan sebesar 60 derajat mendekati pusat peta.
Bot pemain menyalakan <i>shield</i> jika sedang diserang.	 	<p>Bot melakukan prediksi kena dengan interpolasi berdasarkan jarak bot terhadap torpedo yang paling dekat lalu akan menghitung jumlah torpedo di sekitar torpedo tersebut.</p> <p>Jika interpolasi menghasilkan kena dan jumlah di sekitar torpedo ada 3 dan waktu hasil interpolasi dibawah nilai tertentu maka bot akan menyalakan <i>shield</i>.</p> <p>Gambar pertama menunjukkan sebuah <i>bot</i> yang hendak dikenai lebih dari 3 buah torpedo.</p> <p>Gambar kedua menunjukkan <i>bot</i> tersebut menyalakan <i>shield</i>.</p>

Bot pemain akan menembakkan *teleport* untuk menyerang *bot* lain jika berukuran tertentu.

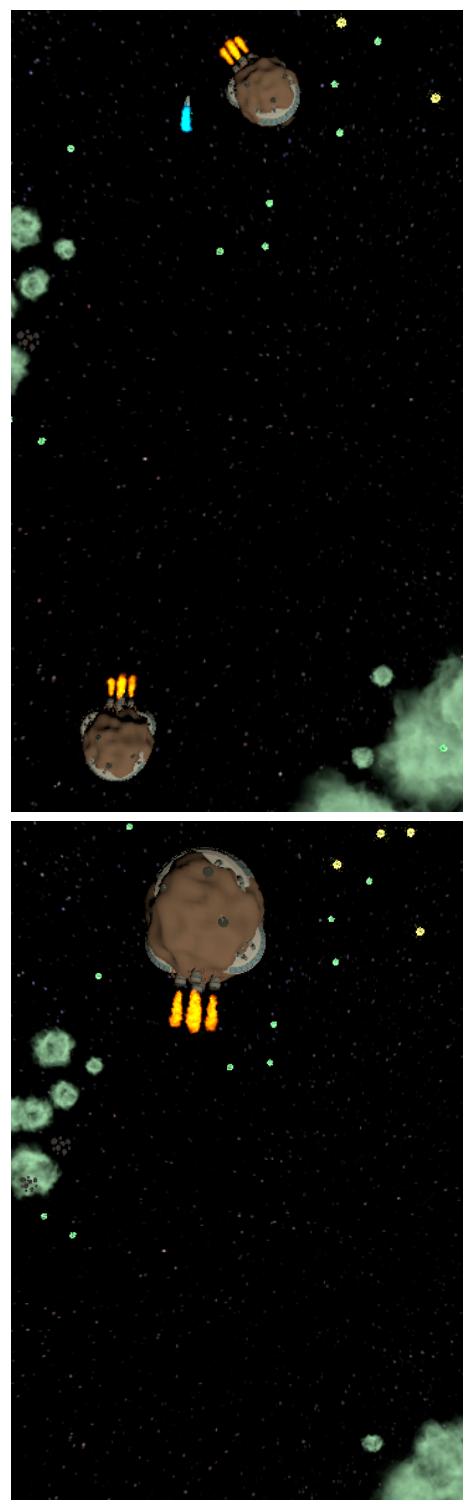


Bot melakukan perbandingan terhadap ukurannya dan ukuran *bot* terdekat, jika ukuran *bot* melebihi ukuran *bot* terdekat dengan nilai tertentu maka *bot* akan menembakkan *teleport*.

Gambar pertama menunjukkan *bot* lain yang diserang.

Gambar kedua menunjukkan *tick* selanjutnya dengan *teleport* telah ditembakkan.

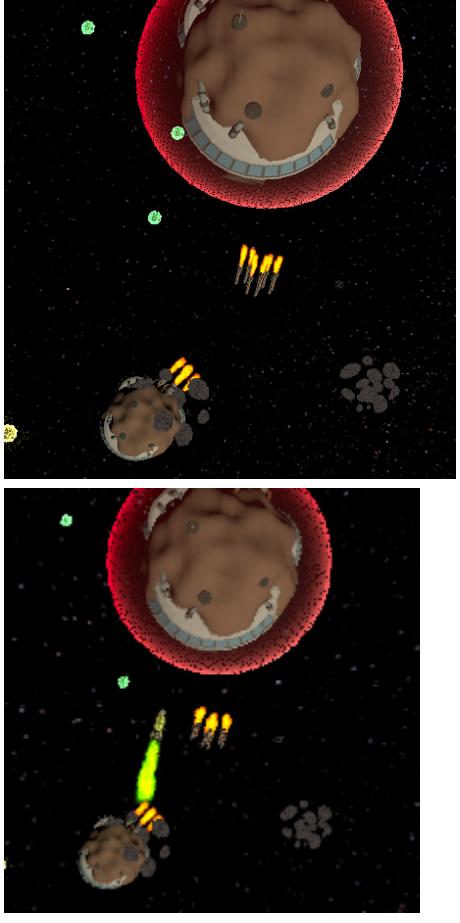
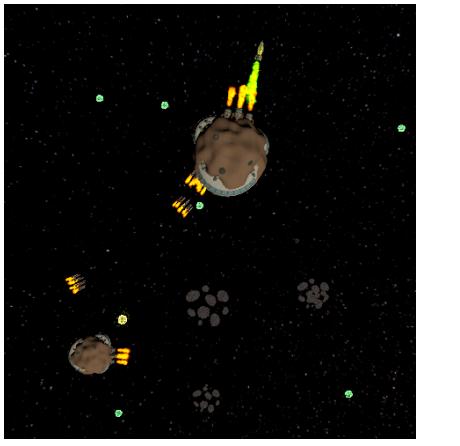
Bot pemain akan melakukan *teleport* jika *teleport* yang ditembakkan mendekati bot lain.

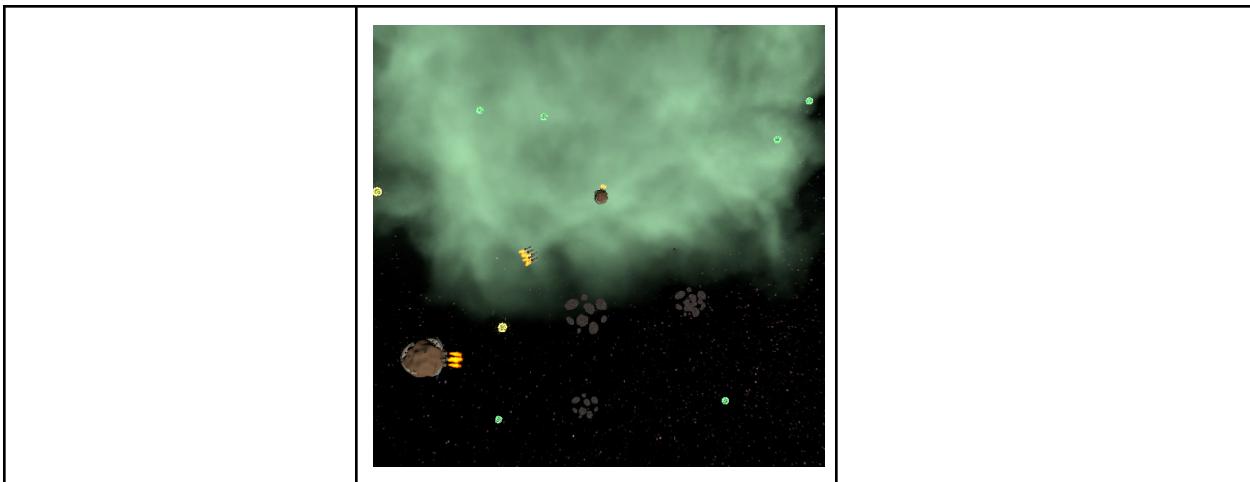


Bot mendeteksi keberadaan *teleport* yang pada peta lalu menghitung jaraknya. Bot akan melakukan *teleport* jika mendekati bot lain yang berukuran lebih kecil, namun hal ini berarti untuk *teleport* yang telah ditembakkan memiliki kemungkinan untuk tidak dilakukan *teleport* jika arah penembakan bot meleset.

Gambar pertama menunjukkan *teleport* mendekati bot lain.

Gambar kedua menunjukkan bot telah melakukan *teleport* dan memakan bot lain tersebut.

<p><i>Bot</i> pemain akan menembakkan <i>supernova</i> untuk menyerang <i>bot</i> lain jika memilikinya.</p>	 <p>The top screenshot shows a player bot (brown sphere) with a red energy shield and a blue ring around its base. It is facing another bot (brown sphere) which has just fired a yellow energy beam at it. The bottom screenshot shows the same scene after the beam hit, with the player bot's shield partially depleted and the other bot's shield also partially depleted.</p>	<p>Jika memiliki <i>supernova</i>, <i>bot</i> akan langsung menembakkannya ke <i>bot</i> lain dengan jarak terdekat.</p> <p>Gambar pertama menunjukkan <i>bot</i> telah memiliki <i>supernova</i> tengah dikejar oleh <i>bot</i> lain yang lebih besar.</p> <p>Gambar kedua menunjukkan <i>bot</i> menembakkan <i>supernova</i> terhadap <i>bot</i> lain.</p>
<p><i>Bot</i> pemain akan meledakkan <i>supernova</i> jauh dari <i>bot</i> dan dekat dengan <i>bot</i> lain.</p>	 <p>The top screenshot shows a player bot (brown sphere) with a red energy shield and a blue ring around its base. It is facing another bot (brown sphere) which has just fired a yellow energy beam at it. The bottom screenshot shows the same scene after the beam hit, with the player bot's shield partially depleted and the other bot's shield also partially depleted.</p>	<p><i>Bot</i> mendekati keberadaan <i>supernova</i> yang telah ditembakkan pada peta lalu menghitung jaraknya. <i>Supernova</i> akan diledakkan ketika jauh dari <i>bot</i> dan dekat dengan <i>bot</i> lain, namun hal ini berarti <i>supernova</i> memiliki kemungkinan untuk tidak diledakkan jika arah penembakan <i>bot</i> meleset.</p> <p>Gambar pertama menunjukkan <i>supernova</i> telah mendekati <i>bot</i> lain namun belum cukup jauh dari <i>bot</i> penembak.</p> <p>Gambar kedua menunjukkan <i>bot</i> penembak telah melakukan peledakan <i>supernova</i> karena jarak sudah cukup jauh.</p>



BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas Besar I IF2211 Strategi Algoritma ini telah diimplementasikan algoritma berbasis *greedy* beserta fungsi-fungsi pendukung dalam tujuan untuk membuat *bot* yang paling lama bertahan pada permainan Galaxio. Dari eksperimen yang telah dilakukan, didapatkan bahwa penggunaan strategi *greedy* cukup optimal dalam kasus ini. Pemilihan dengan keuntungan paling besar atau tamak pada setiap langkahnya kemungkinan besar merupakan kemungkinan yang terbaik.

Namun, perlu diperhatikan bahwa untuk mencapai hasil yang optimal, implementasi algoritma *greedy* ini perlu dilengkapi dengan pendekatan lain seperti heuristik agar strategi *greedy* yang digunakan lebih optimal dan cocok untuk berbagai kondisi. Hal ini juga diperkuat dengan data pada Tabel 4.3.1. dimana *bot* dapat menangani berbagai kasus dan keadaan. *Bot* kami berhasil membuktikan dengan *winrate* yang besar melawan ReferenceBot. Dengan demikian, penulis menyimpulkan bahwa melalui Tugas Besar I IF2211 Strategi Algoritma ini, dapat dibuat sebuah algoritma berbasis *greedy* diimplementasikan dalam suatu program yang dapat menghasilkan *bot* menyelesaikan Galaxio.

5.2 Saran

Tugas kecil IF2211 Strategi Algoritma Semester II Tahun 2022/2023 menjadi salah satu tugas yang memberikan pelajaran baru bagi penulis. Berdasarkan pengalaman penulis mengerjakan tugas ini, berikut merupakan saran untuk pembaca yang ingin melakukan atau mengerjakan hal yang serupa.

1. Perancangan algoritma dan struktur program perlu diperhatikan dalam mengerjakan tugas ini. Pada tugas ini, bot penulis berkali-kali terputus sambungannya dengan game engine. Setelah beberapa kali percobaan, hal ini sepertinya disebabkan oleh penggunaan heap memory yang sangat besar sehingga game engine menjadi kewalahan.
2. Keefektifan dalam kerja sama tim merupakan hal yang penting dalam mengerjakan tugas ini. Tugas ini sangat terbantu oleh pemakaian real-time collaboration app. Selain itu, pemakaian aplikasi pengelola version control seperti Github sangat disarankan agar memudahkan untuk mengelola pekerjaan secara asinkron.
3. Mengingat bahwa program implementasi pada tugas ini sudah termasuk ke dalam pemrograman berorientasi objek, penggunaan Struktur Data yang tepat merupakan hal yang penting untuk diperhatikan. Pada Tugas Besar I ini, kami menggunakan tipe buatan agar membuat fungsi menjadi lebih rapi dan dapat digunakan dengan mudah ketika *fetching* dan *transferring* ke *game engine*

DAFTAR PUSTAKA

- Munir, R. (2022). Strategi Algoritma #1 Algoritma Greedy (1). Retrieved from Homepage Rinaldi Munir: [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, R. (2022). Strategi Algoritma #1 Algoritma Greedy (1). Retrieved from Homepage Rinaldi Munir: [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, R. (2022). Strategi Algoritma #1 Algoritma Greedy (1). Retrieved from Homepage Rinaldi Munir: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)- Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)- Bag3.pdf)

LAMPIRAN

Lampiran 1 Link Repository GitHub

https://github.com/MuhamadAjiW/Tubes1_T-T-k3n5l3rBr30w0gh-T-T.git

Lampiran 2 Link Video Bonus

https://youtu.be/6JiZ_UQ8zxo