Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Galaxio"

Disusun oleh :

Fajar Maulana Herawan	13521080
Mohammad Farhan Fahrezy	13521106
Muhammad Abdul Ghazali	13521128

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

2022

Daftar Isi

Bab 1 : Deskripsi Tugas

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

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 Entellect Challenge yang terdapat pada *repository* githubnya. *Game engine* yang dibuat oleh Entellect Challenge menggunakan bahasa Scala sebagai bahasa pemrograman utama dalam pembuatannya.

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

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di *integer* x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu *Players, Food, Wormholes, Gas Clouds, Asteroid Fields.* Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan x. Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. *Heading* dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek *afterburner* akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. *Food* akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan oleh kapal *player*. Apabila *player* mengkonsumsi *Food,* maka *Player* akan bertambah ukuran yang sama dengan *Food. Food* memiliki peluang untuk berubah menjadi *Super Food.* Apabila *Super Food* dikonsumsi maka setiap makan *Food,* efeknya akan 2 kali dari *Food* yang dikonsumsi. Efek dari *Super Food* bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari *player* untuk memasukinya dan keluar di pasangan satu lagi. *Wormhole akan* bertambah besar setiap tick game hingga ukuran maximum. Ketika *Wormhole* dilewati, maka *wormhole* akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat *wormhole* lebih besar dari kapal *player.*
5. *Gas Clouds* akan tersebar pada peta. Kapal dapat melewati *gas cloud*. Setiap kapal bertabrakan dengan *gas cloud,* ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan *gas cloud*, maka efek pengurangan akan hilang.
6. *Torpedo Salvo* akan muncul pada peta yang berasal dari kapal lain. *Torpedo Salvo* berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada

lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. *Torpedo Salvo* akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.

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

Bab 2 : Landasan Teori

2.1 Gambaran Algoritma *Greedy* secara Umum

Algoritma *greedy* adalah algoritma yang memakai konsep "*greedy*" atau serakah dalam solusi dari persoalan optimisasi yang memaksimumkan atau meminimumkan suatu parameter. Algoritma greedy diimplementasi dengan memecah permasalahan dan membentuk solusi setiap langkahnya. Algoritma *greedy* akan menentukan keputsusan terbaik pada setiap langkahnya. Akan tetapi, algoritma tidak boleh melakukan *backtracking* (tidak boleh munduk ke solusi sebelumnya untuk menentukan solusi langkah sekarang). Dengan mementukan solusi terbaik di setiap langkahnya, algoritma diharapkan dapat memberikan solusi yang terbaik secara menyeluruh (global).

Sebuah persoalan dapat diselesaikan dengan algoritma *greedy* jika persoalan tersebut memiliki dua sifat ,yakni

1. Solusi optimal dari sebuah persoalan dapat ditentukan dari solusi optimal dari masalah yang lebih kecil
2. Pada setiap persoalan, terdapat beberapa langkah dan dari langkah tersebut didapatkan langkah yang merupakan solusi terbaik. Langkah tersebut disebut *greedy choice*.

Elemen di algoritma *greedy* meliputi :

a. Himpunan kandidat (C) : Berisi kandidat yang mungkin dipilih pada setiap langkahnya.
b. Himpunan solusi (S) : Berisi kandidat yang sudah terpilih sebagai solusi
c. Fungsi solusi (*solution function)*: Menentukan apakah himpunan kandidat yang terpilih adalah solusi (domain : himpunan solusi, range : boolean)
d. Fungsi seleksi adalah fungsi yang memilih kandidat dengan menerapkan startegi *greedy* tertenty, yang tidak selalu menjamin hasil optimal secara matematis. Fungsi ini mengambil himpunan objek sebagai input dan mengembalikan objek sebagai output.
e. Fungsi kelayakan adalah fungsi yang memeriksa apakah kandidat yang dipilih oleh fungsi seleksi dapat dimasukkan ke dalam himpunan solusi. Fungsi ini mengambil himpunan objek sebagai input dan mengembalikan nilai Boolean (benar atau salah) sebagai output.
f. Fungsi objektif adalah fungsi yang bertujuan untuk memaksimumkan atau meminimumkan suatu parameter pada suatu persoalan. Fungsi ini mengambil himpunan objek sebagai input dan mengembalikan himpunan objek sebagai output.

Dengan menggunakan elemen-elemen di atas, algortima greedy akan mencari sebuah subset *S* dari kandidat *C*. Subset *S* harus memenuhi syarat sebagai solusi paling optimal dengan mengoptimalkan fungsi objektif, *S* merupakan himpunan solusi yang layak. Kelemahan dari algoritma ini sendiri yakni karena kita tidak melaukan operasi secara menyeluruh kepada semua kemungkinan yang ada maka ada kemungkinan sousi yang di berikan bukanlah yang terbaik.

Dengan demikian, Algoritma *greedy* cocok digunakan untuk masalah yang hanya mendekati optimal dan tidak tepat optimal. Terkadang, solusi yang dihasilkan lebih baik daripada algoritma

yang menghasilkan solusi eksak dengan kebutuhan waktu yang lama. Dalam contoh ini, *Traveling Salesman Problem*, algoritma *greedy* akan jauh lebih cepat dibandingkan dengan algoritma *brute-force*,walaupun solusi yang ditemukan biasanya hanya mendekati solusi optimal.

2.2 Garis Besar Cara Kerja Bot Permainan Galaxio

Secara garis besar, cara kerja *bot* dari permainan Galaxio ini adalah pertama-tama *bot* akan menganalisis objek – objek di dalam game yang tersedia pada GameState.java. Objek – objek tersebut antara lain :

- Player :

Semua bot yang ada di dalam game.

- Food :

Makanan yang jika di makan akan menambah size bot. *Food* memiliki peluang untuk berubah menjadi *Super Food*

- SuperFood :

Jika dikonsumsi maka setiap makan *Food,* efeknya akan 2 kali dari *Food* yang dikonsumsi dan hanya bertahan 5 tick.

- WormHole :

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.*

- GasCloud :

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.

- AsteroidField

Jika Bot terkena maka akan melambat

- TorpedoSalvo

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.

- SupernovaPickUp

Supernova yang dapat diambil

- SupernovaBomb

Supernova yang sedang diledakkan

- Teleporter

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.

- Shield

Melindungi Bot dari TorpedoSalvo

Setelah itu, Bot akan melakukan analisis dengan menggunakan algoritma untuk mencari aksi paling tepat.Aksi – aksi yang dilaksanakan meliputi :

- Forward
- Stop
- StartAfterBurner
- StopAfterBurner
- FireTorpedoes
- FireSupernova
- DetonateSupernova
- FireTeleport
- Teleport
- ActivateShield

Algoritma yang dituliskannya tersebut akan menggunakan algoritma *greedy*.

2.3 Implementasi Algoritma *Greedy* ke dalam Bot Permainan Galaxio

Beberapa algoritma seperti *brute-force, minimax, greedy,* dan lain-lain. Setiap algoritma memiliki kelebihan dan kekurangannya masing – masing. *Brute-force* memerlukan banyak waktu dan sumber daya untuk mengevaluasi setiap kemungkinan *command*, tetapi dapat menghasilkan solusi optimum global. *Greedy* tidak memerlukan banyak waktu atau sumber daya, tetapi bergantung pada teknik heurisitk yang mungkin tidak menghasilkan solusi optimum global atau sulit dibuktikan kebenarannya. Minimax umum digunakan pada bot game, akan tetapi algoritma *greedy* lebih cepat dalam menghasilkan solusi karena hanya mempertimbangkan tindakan terbaik dalam jangka pendek.

Oleh karena itu, penulis menggunakan algoritma *greedy* sebagai algoritma pembentukan bot disebabkan algoritma ini tidak membutuhkan waktu atau memori yang terlalu besar. Selain itu, terdapat banyak kemungkinan solusi *greedy* yang dapat di buat oleh penulis. Akan tetapi, teradapat peluang penulis tidak dapat menciptakan algoritma *greedy* yang paling optimal.

2.4 Garis Besar *Game Engine* Permainan Galaxio

Game engine adalah kerangka kerja perangkat lunak yang dirancang sebagai kumpulan alat dan fitur untuk membantu pengembangan game. Biasanya, game engine memiliki beberapa alat pengembangan inti, seperti rendering engine (untuk merender grafik 2D atau 3D), physics engine atau collision engine (untuk mengatur aturan fisika pada game), sound engine, scripting language, animation engine, kecerdasan buatan, jaringan, streaming, manajemen memori, dukungan threading, dukungan lokal, grafik animasi, dan dukungan video. Pada game ini, tidak semua alat disediakan oleh programmer, hanya beberapa alat esensial seperti rendering engine untuk command prompt, collision engine, dan scripting language yang tersedia di repositori. Beberapa alat tambahan, seperti rendering engine 2D, dapat diunduh dari perangkat lunak pihak ketiga.

Pada permainan Galaxio, *game engine* yang digunakan dibuat menggunakan bahasa C# dan sudah tersedia di *repository* yang diunggah sebelumnya oleh Entelect.terdapat tiga komponen penting dari game ini ,antara lain

1. Engine

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

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

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

Folder starter pack terdiri dari engine-publish,logger-publish,referencebot-publish,runner-publish,starter-bots,visualizer. Engine-publish berfungsi untuk engine dari gamenya, logger-publish untuk menyimpan hasil run time, referencebot-publish adalah bot default game, runner-publish untuk menjalankan game, dan visualizer untuk menvisualkan game. Sebelumnya untuk menjalankan *game engine,* Kita harus memiliki JDK (Java Delopment Kit) Versi 11 dan .NET versi 3.1. Kemudian, untuk menjalankan game mengetikan run.bat untuk windows dan sh run.sh untuk linux. Sebelumnya, terdapat langkah-langkah untuk menjalankan game

1. Tentukan jumlah bot yang ingin dimainkan dengan mengkonfigurasi file JSON "appsettings.json" pada folder "runner-publish" dan "engine-publish".
2. Buka terminal baru pada folder "runner-publish".
3. Jalankan runner menggunakan perintah "dotnet GameRunner.dll".
4. Buka terminal baru pada folder "engine-publish".
5. Jalankan engine menggunakan perintah "dotnet Engine.dll".
6. Buka terminal baru pada folder "logger-publish".
7. Jalankan logger menggunakan perintah "dotnet Logger.dll".
8. Jalankan seluruh bot yang ingin dimainkan.
9. Setelah permainan selesai, riwayat permainan akan disimpan pada 2 file JSON dengan nama "GameStateLog_{Timestamp}" yang terletak pada folder "logger-publish". Kedua file tersebut mencakup hasil akhir dari permainan dan seluruh proses dalam permainan

Pada saat dijalankan, Runner akan berfungsi sebagai host untuk sebuah pertandingan pada sebuah hostname tertentu dan secara lokal di-host pada localhost:5000. Engine kemudian terhubung dengan Runner dan menunggu bot pemain terhubung ke Runner. Logger juga melakukan hal yang sama dan terhubung dengan Runner. Jumlah bot yang dibutuhkan untuk memulai pertandingan ditentukan oleh atribut BotCount pada file JSON "appsettings.json" di folder "runner-publish" dan "engine-publish". Pertandingan dimulai ketika jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi. Bot mendengarkan event dari Runner, terutama event RecieveGameState yang memberikan status game, dan mengirimkan event kepada Runner yang berisi tindakan bot. Pertandingan berakhir saat selesai, dan dibuat dua file JSON yang berisi riwayat pertandingan.

Bab 3 : Penggunaan Algoritma Greedy

3.1 Pemetaan Komponen Algoritma Greedy Bot Galaxio

3.1.1 Pemetaan Komponen Algoritma Greedy pada Persoalan General Bot

Pemenang dalam *game* Galaxio adalah satu-satunya bot yang dapat bertahan hingga akhir permainan dimana semua bot lain sudah mati. Galaxio memberikan beberapa cara bagi bot untuk menjadi pemenang, seperti memakan *food* hingga dapat menjadi berukuran besar, mengarahkan bot ke arah tertentu agar terhindar dari *gas clouds* dan *asteroid field*, menembak bot lain dengan *torpedo*, dan masih banyak lagi. Secara umum, bot dapat mengkonfigurasi arah *heading* dan *action* yang akan dilakukan.

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action* yang dapat dilakukan pada seluruh *tick* dalam permainan
Himpunan Solusi	*Action* dan *heading* dari bot yang menjadikan bot sebagai *survivor* terakhir.
Fungsi Solusi	Mengecek apakah permutasi dari *action* dan *heading* dapat menjadikannya pemenang
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik ini akan melihat kondisi optimal pada suatu kasus yang telah terbagi-bagi. Fungsi ini dibuat dengan logika, intuisi, dan pengalaman programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid dapat dilihat pada bab sebelumnya
Fungsi Objektif	Mencari permutasi *action* dan *heading* yang membuat bot terhindar dari *obstacle* yang membuat *size* bot berkurang dan tidak termakan bot lain serta melakukan penyerangan pada musuh dengan optimal.

3.1.2 Pemetaan Komponen Algoritma Greedy pada Persoalan *Damage Mechanism*

Galaxio memungkinkan bot untuk menerima *damage* berupa pengurangan *size* dari serangan musuh dan obstacle pada *map*. Bot akan berkurang *size*-nya apabila terkena *torpedo* atau *supernova* dari musuh. Bot juga akan berkurang *sizenya* apabila terkena gas clouds dan *border* dari permainan. Bot dapat menggunakan shield dalam durasi tertentu untuk bertahan dari *torpedo* dan *supernova* dengan cost *size* bot. Selain itu *torpedo* dari musuh juga dapat ditembak dengan *torpedo* bot untuk mengurangi size dan damage dari *torpedo* musuh tetapi akan mengurangi size dari *bot*. Bot juga dapat merubah arah *heading* agar tidak terkena *gas*

clouds dan *border*. *Damage* yang paling kuat dihasilkan dari musuh yang memiliki size lebih besar dari bot dan melakukan *collision* dengan bot. Hasil dari *collision* adalah bot yang lebih kecil *sizenya* akan langsung mati

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action ACTIVATESHIELD, FIRETORPEDOS, FORWARD* yang dapat dilakukan pada seluruh *tick* dalam permainan
Himpunan Solusi	*Action ACTIVATESHIELD, FIRETORPEDOS, FORWARD* dan *heading* dari bot yang menjadikan bot terhindar dari pengurangan *size.*
Fungsi Solusi	Mengecek apakah permutasi dari *action Action ACTIVATESHIELD, FIRETORPEDOS, FORWARD* dan *heading* dapat menghindari bot dari pengurangan size dari *damage*.
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik pada persoalan *damage mechanism* ini didasarkan dengan strategi programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid adalah *heading* yang sesuai dan melakukan *Action ACTIVATESHIELD, FIRETORPEDOS, FORWARD* atau tidak sama sekali
Fungsi Objektif	Mencari permutasi *action* dan *heading* yang membuat bot mendapat *damage* seminimum mungkin dari serangan lawan dan *obstacle* pada *map* dengan mempertimbangkan ketersediaan *shield,torpedoes* size bot.

3.1.3 Pemetaan Komponen Algoritma Greedy pada Persoalan Menyerang Musuh

Galaxio memungkinkan bot untuk menyerang bot lain agar *size* dari musuh semakin berkurang sehingga akan mati (*size = 0)*. Meskipun bot dapat memenangkan permainan tanpa menyerang, akan didapatkan kemenangan dengan lebih pasti apabila bot menyerang musuh. Bot dapat menyerang dengan *torpedoes* dan *supernova* yang dapat diarahkan dengan *heading* bot.

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action FIRETORPEDOS, FIRESUPERNOVA, DETONATESUPERNOVA atau tidak*

	menyerang sama sekali yang dapat dilakukan pada seluruh *tick* dalam permainan
Himpunan Solusi	*Action FIRETORPEDOS, FIRESUPERNOVA, DETONATESUPERNOVA* dan *heading* dari bot atau tidak menyerang sama sekali yang paling menguntungkan bot.
Fungsi Solusi	Mengecek apakah permutasi dari *action Action FIRETORPEDOS, FIRESUPERNOVA, DETONATESUPERNOVA* dan *heading* dapat membuat bot optimum.
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik pada persoalan menyerang musuh ini didasarkan dengan strategi programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid adalah *heading* yang sesuai dan melakukan action *FIRETORPEDOS, FIRESUPERNOVA, DETONATESUPERNOVA* atau tidak sama sekali
Fungsi Objektif	Mencari permutasi *action* dan *heading* yang membuat bot menyerang musuh secara optimum dengan mempertimbangakan ketersediaan *torpedos* dan *supernova* serta *size* dari bot.

3.1.4 Pemetaan Komponen Algoritma Greedy pada Persoalan *Collision*

Galaxio memungkinkan bot untuk melakukan *collision* , yaitu suatu kondisi dimana bot mengenai bot lain. Bot yang lebih besar akan bertambah ukurannya sebanyak 50% dari bot yang lebih kecil dan bot yang lebih kecil akan mati. Bot yang lebih besar dapat melakukan *afterburner* dan mengejar musuh dengan lebih cepat dengan harga *size* dari bot. Sedangkan apabila bot lebih kecil, maka harus menjauh dari yang lebih besar dan apabila perlu dapat mengaktifkan *afterburner* apabila musuh yang lebih besar sudah sangat berdekatan dengan bot. *afterburner* dapat dinonaktifkan apabila sudah memenuhi tujuannya. Persoalan ini sangatlah penting karena dapat langsung membuat bot gugur dari kemenangan.

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action START_AFTERBURNER, STOP_AFTERBURNER, FORWARD atau tidak* melakukan *collision* sama sekali yang dapat dilakukan pada seluruh *tick* dalam permainan

Himpunan Solusi	*Action START_AFTERBURNER, STOP_AFTERBURNER, FORWARD* dan *heading* dari bot *atau tidak* melakukan *collision* sama sekali yang paling menguntungkan bot.
Fungsi Solusi	Mengecek apakah permutasi dari *Action START_AFTERBURNER, STOP_AFTERBURNER, FORWARD* dan *heading* dari bot *atau tidak* melakukan *collision* sama sekali dapat membuat bot optimum.
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik pada persoalan menyerang musuh ini didasarkan dengan strategi programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid adalah *heading* yang sesuai dan melakukan *Action START_AFTERBURNER, STOP_AFTERBURNER, FORWARD* dan *heading* dari bot *atau tidak* melakukan *collision* sama sekali
Fungsi Objektif	Mencari permutasi *Action START_AFTERBURNER, STOP_AFTERBURNER, FORWARD* dan *heading* dari bot *atau tidak* melakukan *collision* sama sekali dan *heading* yang membuat bot dapat melakukan *collision* , menjauh dari musuh mempertimbangkan *size* bot dan musuh serta jarak dengan musuh.

3.1.5 Pemetaan Komponen Algoritma Greedy pada Persoalan *Consumable Food* dan *Superfood*

Bot dalam permainan Galaxio dapat pula memakan *food* dan *superfood* yang ada pada map untuk memperbesar *size*. Bot dapat mengarahkan *heading* pada *food* dan *superfood* dan melakukan *action FORWARD*. Bot menambah *size*nya dengan *food* dan *superfood* agar semakin sulit untuk kalah dalam *collision* dan dapat menggunakan *action* yang memerlukan pengurangan *size*.

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali yang dapat dilakukan pada seluruh *tick* dalam permainan

Himpunan Solusi	Permutasi *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali yang memberikan efek paling optimal pada bot.
Fungsi Solusi	Mengecek apakah permutasi dari *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali adalah yang paling optimum
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik pada persoalan menyerang musuh ini didasarkan dengan strategi programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid adalah *heading* yang sesuai dan melakukan *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali
Fungsi Objektif	Mencari permutasi *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali yang membuat bot dapat memakan *food* dan *superfood* dengan bahaya yang paling minimum dan paling cepat membuat bot membesar *size*-nya.

3.1.5 Pemetaan Komponen Algoritma Greedy pada Persoalan *Teleport* dan *Wormhole*

Bot pada permainan Galaxio dapat pula melakukan *action FIRE_TELEPORTER* dan *TELEPORT* apabila tersedia *teleporter* pada bot untuk berpindah secara instan pada map. Dengan mekanisme ini, bot dapat melakukan mobilisasi dengan lebih mudah. Bot akan berpindah pada daerah yang paling menguntungkan. Semisal ada bot ingin melakukan collision pada bot yang lebih kecil, maka *teleporter* akan memudahkan bot dalam melakukan *collision*. Dan ada banyak lagi kasus penggunaan *teleporter* ini. *Wormhole* juga tersedia pada map untuk berteleportasi pada wormhole lain pada peta.

Komponen	Deskripsi Komponen
Himpunan Kandidat	Seluruh permutasi dari *heading* dan *action FIRE_TELEPORTER* dan *TELEPORT atau tidak* berteleportasi sama sekali yang dapat dilakukan pada seluruh *tick* dalam permainan
Himpunan Solusi	*heading* dan *action FIRE_TELEPORTER* dan *TELEPORT atau tidak* berteleportasi sama sekali yang memberikan efek paling optimal pada bot.

Fungsi Solusi	Mengecek apakah permutasi dari *heading* dan *action FIRE_TELEPORTER* dan *TELEPORT atau tidak* berteleportasi sama sekali adalah yang paling optimum
Fungsi Seleksi	Mencari *action* dan *heading* berdasarkan *state* dari bot itu sendiri dan *state* dari permainan yang sedang berjalan. Pengurutan prioritas dalam pencarian *action* dan *heading* dilakukan dengan fungsi heuristik. Fungsi heuristik pada persoalan menyerang musuh ini didasarkan dengan strategi programmer.
Fungsi Kelayakan	Melakukan validasi *action* dan *heading* pada bot. *action* dan *heading* yang valid adalah *heading* yang sesuai dan melakukan *heading* dan *action FORWARD atau tidak* memakan *food* dan *superfood* sama sekali
Fungsi Objektif	Mencari *heading* dan *action FIRE_TELEPORTER* dan *TELEPORT atau tidak* berteleportasi sama sekali yang membuat bot dapat berpindah pada lokasi di map yang paling optimum dengan mempertimbangkan ketersediaan *teleporter* dan obstacle yang mungkin ada pada daerah lokasi yang dituju.

3.2 Eksplorasi Berbagai Algoritma Greedy dan Strategi Heuristik yang Bersesuaian pada Bot Galaxio

Banyaknya objek yang dapat diakses bot seperti *game object* (Player lain, Obstacle, dan lain lain) dari *game state* dapat memberikan banyak alternatif pertimbangan keputusan *action* dan *heading* bot yang dilakukan pada suatu *tick*. Maka dari itu, akan ada banyak Algortima Greedy yang dihasilkan pula. Strategi heuristik yang digunakan juga akan ada banyak karena solusi yang paling optimum akan sulit dibuktikan dan hanya dapat dinilai dari keefektifannya dengan data performa bot pada permainan yang sudah terjadi, serta banyaknya hubungan interaksi antarelemen dalam permainan Galaxio. Strategi yang menurut penulis merupakan strategi yang efektif dijelaskan dalam berikut ini :

3.2.1 Strategi Heuristik *Damage Mechanism*

Damage pada bot adalah berupa berkurangnya *size* pada bot. Bot dapat menerima *damage* dari *gas clouds, asteroid fields, torpedo, supernova,* dan *border.* Dengan pertimbangan dari objek-objek tersebut dan keadaan bot, dihasilkan beberapa strategi heuristik yang bertujuan untuk meminimalisasi *damage* yang diterima bot.

Strategi heuristik pertama adalah adalah mengaktifkan *shield* pada saat terdeteksi serangan musuh yang mengarah ke bot dan sudah dekat apabila tersedia *shield* pada bot dan *size* bot mencukupi. Apabila tidak tersedia, akan meluncurkan *torpedo* pada arah tersebut

untuk menghadangnya apabila size bot masih mencukupi untuk meluncurkan *torpedo* dan *charge torpedo* masih ada pada bot. Kemudian apabila tidak ada *charge torpedo* atau *size* nya tidak mencukupi , bot akan menghindar saja dengan action *FORWARD* pada arah yang tegak lurus dari arah serangan. Apabila ada supernova yang terdeteksi mendekati bot, bot akan bergerak menjauhinya agar tidak terkeda *damage* dan *gas cloud* yang dihasilkan. Pada strategi ini, bot akan selalu menghindar dari *gas cloud* (*spawn* maupun *aftereffect supernova)* bot tidak akan menyebranginya sama sekali. Strategi ini memaksimalkan *resource* yang ada pada bot untuk sangat meminimalisasi *damage.* Namun, strategi ini tidak mengambil resiko untuk menerima *damage* yang dapat ditolerir untuk bertahan hidup sehingga dapat terjadi kasus terjebak diantara musuh dan *gas cloud*.

Strategi kedua memiliki penanganan yang sama pada *torpedos* dan *supernova* tetapi akan berperilaku berbeda terhadap *gas cloud* dan *asteroid field.* Bot akan melintasinya apabila dirasa perlu dengan pertimbangan hal-hal tertentu. Apabila *gas cloud* cukup kecil dan lokasi optimal yang akan dimiliki bot berada di seberangnya, maka bot akan melintasinya. Selain itu, apabila bot sedang dikejar musuh yang lebih besar, maka bot akan melintasi *gas cloud* untuk menghindarinya. Penyebrangan ini mempertimbangkan *size* dari bot agar tidak terlalu kecil dan dapat bertahan dari *damage gas cloud.* Strategi ini memecahkan masalah dimana bot tidak bisa menghindar karena terjebak. Namun, perhitungan damage yang dapat ditolerir cukup masih beresiko karena faktor elemen lain seperti musuh lain cukup sulit untuk diperhitungkan aksinya.

3.2.2 Strategi Heuristik Menyerang Musuh

Ada beberapa strategi untuk memaksimalkan *damage* pada musuh. Dan dengan mempertimbangkan *charge torpedo*, keadaan pada map, dan hal lain , dihasilkan strategi berikut ini :

Strategi pertama adalah apabila size dari bot tidak di bawah batas aman, tidak ada penghalang seperti *gas cloud* dan *asteorid field* antara bot dengan musuh yang ditarget, dan *charge torpedo* tersedia, maka bot akan mengubah *heading* ke arah musuh dan melakukan action *FIRE_TORPEDOES.* Kemudian apabila bot memiliki *supernova,* akan diubah *heading* ke arah musuh yang masih di atas batas jarak aman supernova dan apabila supernova sudah cukup dekat dengan musuh akan dilakukan *DETONATE_SUPERNOVA*. Namun strategi ini memungkinkan terjadinya *torpedoes* tidak mengenai musuh dan hanya membuang-buang *charge* dan *size.*

Strategi kedua adalah strategi yang sama dengan strategi pertama, namun hanya akan dilakukan penembakan *torpedoes* apabila jarak dengan musuh cukup dekat agar akurasi penembakan meningkat.

3.2.3 Strategi Heuristik *Collision*

Terdapat beberapa strategi dalam menangani persoalan *collision* bot, yaitu sebagai berikut :

Strategi pertamanya adalah menjauh dari musuh yang lebih besar dari bot dengan heading ke arah yang paling optimal dimana tidak ada *obstacle* dan *border.* Kemudian apabila bot memiliki *size* yang berada di batas aman untuk menang dalam *collision* dengan musuh, akan diubah *heading* menuju arah musuh dan action *FORWARD*. Kelemahan strategi ini adalah tidak dapat mengejar musuh yang mengaktifkan *afterburner*.

Strategi kedua dilakukan dengan strategi yang sama dengan strategi kedua tetapi menggunakan *AFTERBURNER* DAN *STOP_AFTERBURNER* hingga tujuan dari bot tercapai. Strategi ini mengorbankan *size* dari bot untuk melakukan *afterburner*.

3.2.4 Strategi Heuristik *Consumable Food* dan *Superfood*

Ada beberapa Strategi yang dapat dilakukan oleh bot untuk memaksimalkan *food* dan *superfood*. Dengan mengonsumsi objek ini, bot akan menambah ukuran dan *superfood* juga memberikan efek yang lebih menguntungkan dalam memperbesar *size* (multiplier). Strateginya adalah sebagai berikut :

Strategi pertama diimplementasikan dengan mengubah *heading* ke arah *food* dan *superfood* terdekat yang ada dengan action *FORWARD* apabila tidak terdapat prioritas lain seperti menyerang, menjauh musuh yang harus dilakukan oleh bot. Dalam pengimplementasiannya, harus diperhatikan untuk menangani jarak makanan yang sama dekatnya agar tidak terjadi *loop* dimana bot hanya akan berbolak balik arah.

Strategi kedua dilakukan dengan memprioritaskan superfood terlebih dahulu yang ada radius tertentu dan baru akan mengonsumsi *food* selanjutnya. Hal ini bertujuan agar *food* dan *superfood* yang dikonsumsi dapat memberikan penambahan *size* paling optimal.

Strategi ketiga diimplementasikan dengan membagi arah heading menjadi beberapa partisi dan dalam setiap partisi akan dihitung pembobotan banyaknya *food* dan *superfood* beserta jaraknya. Kemudian bot akan menuju ke arah yang memiliki bobo *food* dan *superfood* yang paling besar.

3.2.5 Strategi Heuristik *Teleporter*

Terdapat beberapa strategi dalam penggunaan *teleporter* dan pencarian *teleporter* yang mungkin untuk bot, yaitu sebagai berikut :

Strategi pertama adalah menuju arah *spawn teleporter* pada map untuk mendapatkan *charge*-nya apabila sudah dalam radius tertentu dari bot. Kemudian apabila bot sudah memiliki *teleporter* dan *size* nya berada di atas batas aman, bot akan menggunakan teleporter dalam kasus berikut:

- Terjebak diantara obstacle dan **musuh**
  Akan dilakukan pembagian beberapa partisi arah *heading* dan setiap heading akan dilist *obstacle* dan musuh yang ada di partisi tersebut. Apabila dalam jumlah partisi yang tidak aman dari musuh dan *obstacle* sudah melebihi batas, maka diputuskan untuk menggunakan teleporter ke arah *heading* yang aman dengan *action FIRE_TELEPORTER* dan apabila *teleporter* ditembakkan sudah berada di posisi yang memenuhi akan dilakukan *action TELEPORT*.
- Adanya spawn *supernova*
  Bot akan langusng mengubah *heading* ke arah *supernova* , lalu melakukan *action FIRE_TELEPORTER* kemudian melakukan *action TELEPORT* apabila *teleporter* sudah berada di posisi *supernova*.

Selain itu *wormhole* akan selalu dihindari karena sangat berdasarkan keberuntungan mengenai dimana bot akan dipindahkan.

Strategi kedua memiliki strategi yang sama dengan strategi pertama namun berbeda pada *wormhole*. *Wormhole* hanya akan dituju apabila *size* bot cukup kecil dan sedang dikejar

oleh musuh. Walaupun sangat mengedepankan keberuntungan, strategi ini cukup efektif pada beberapa kasus.

3.3 Analisis Efiisiensi dari Algoritma Greedy yang Ada

Pada Galaxio, setiap informasi dari Game Object dapat didapat dengan mengakses *game state* sehingga kompleksitasnya O(1) dan sangat tidak berpengaruh pada efisiensi kode algoritma *greedy.*

Kemudian pada strategi *damage mechanism* , pencarian objek musuh dan obstacle harus disortir berdasarkan yang paling dekat dengan bot sehingga kompleksitasnya O(log(N) * N*N) karena dalam java, metode Collections.sort memiliki kompleksitas O(log(N) * N)

Lalu pada strategi menyerang musuh pencarian musuh terdekat dan sorting pada jarak tertentu sehingga kompleksitasnya adalah O(log(N) * N*N).

Strategi *collision* akan mencari player yang lebih besar yang paling dekat sehingga kompleksitasnya adalah O(log(N) * N*N).

Lalu pada strategi *Consumable Food* dan *Food* dilakukan pula pencarian dengan sortir jarak yang paling dekat. Maka kompleksitasnya adalah O(log(N) * N*N).

Terakhir, strategi *teleporter* menggunakan pencarian *obstacle* dan menyorting dengan jarak terdekat sehingga kompleksitasnya adalah O(log(N) * N*N).

3.4 Analisis Efektifitas Algoritma Greedy

3.4.1 Strategi Heuristik Damage Mechanism

Strategi pertama efektif dalam meminimalisasi damage karena semua usaha telah dilakukan untuk menjauh dari misil musuh dan tidak terkena damage obstacle.

Strategi kedua dirasa penulis adalah strategi yang paling baik karena berani untuk mengambil resiko disaat kondisi bot sudah *deadend* dan bertahan.

3.4.2 Strategi Heuristik Menyerang musuh

Strategi pertama sangat mengusahakan untuk memberikan damage pada musuh meskipun musuh cukup jauh. Musuh yang jauh ini mugnkin kena atau tidak bergantung keberuntungan.

Strategi kedua sangat meningkatkan akurasi tapi tidak berani mengambil kesempatan saat jalan musuh dengan bot sudah clear.

3.4.3 Strategi Heuristik Collision

Strategi pertama dan kedua sangat baik dalam menghindari collison dari musuh yang lebih besar dan mengejar musuh yang lebih kecil.

34.5 Strategi Heuristik

3.5 Strategi Greedy Pada Bot

Strategi bot yang dipakai adalah strategi greedy damage mechanism kedua, lalu collision yang pertama, dan consumable yang pertama, teleporter yang kedua, dan menyerang musuh yang pertama seperti yang dipaparkan dalam bab 3.4

Bab 4 Implementasi dan Pengujian

4.1. Implementasi Algoritma *Greedy* pada Bot Permainan Galaxio

Implementasi algoritma *greedy* pada program terdapat pada file BotService.java, dalam *method computeNextPlayerAction* yang didalamnya ada fungsi getHeading dan fungsi getNextAction.

```
procedure computeNextPlayerAction
{I.S playerAction terdefinisi}
{F.S playerAction berisi command yang ada pada Game}
{ proses }
{ menentukan aksi pada tick selanjutnya }

Kamus Lokal
GasCloudList : list of gascloud
Foodlist : list of food
PlayerList : list of player
TorpedoList : list of torpedo
AsteroidFieldList : list of asteroidfield
Objectlist : ObjectList
GameState : gameState
mySize : integer
enemySize : integer
EnemyDistance : float

Procedure getHeading(input mySize : integer, input enemySize : integer,
input enemyDistance : float, input playerList:  list of player,input
foodList : list of food, gasCloudList : list of gasCloud)

{I.S parameter terdefinisi}
{F.S arah dari bot untuk tick selanjutnya terdefinisi}
{Proses menentukan semua kemungkinan menggunakan dan mengambil solusi
terbaik menggunakan algoritma greedy untuk arah bot pada tick selanjutnya }

Procedure getNextAction(input gasCloudList : list of gasCloud, input
playeList : list of player,input torpedoList : list of torpedo,input
asteroidFieldList : list of asteroidField)

{I.S parameter terdefinisi}
{F.S aksi dari untuk tick selanjutanya terdefinisi}
{Proses menentukan semua kemungkinan menggunakan dan mengambil solusi
terbaik menggunakan algoritma greedy untuk aksi bot pada tick selanjutnya}

Algoritma

 If ( !gamestate.getGameObjects().isEmpty()) then
    {Mengecek apakah di dalam game masih ada objek}
    mySize <-- bot.getSize()
    Output("Your size: ")
    Output(mySize)
    gasCloudList <-- objectList.getGasCloudList()
    FoodList <-- objectList.getFoodList()
    PlayerList <-- objectList.getPlayerList()
    torpedoList <-- objectList.getTorpedoList()
    asteroidFieldList <-- objectList.getAsteroidFieldList()

    enemySize <-- playerList.get(1).getSize()
```

```
    {mengambil size player terdekat}

    enemyDistance <-- getDistanceBetweenWithSize(bot, playerList.get(1))

    { Menentukan Heading untuk Next Tick }
    getHeading(mySize, enemySize, enemyDistance, playerList, foodList,
gasCloudList);

    { Menentuken PlayerAction untuk next tick }
    getNextAction(gasCloudList, playerList, torpedoList, asteroidFieldList);

   EndIf

   This.playerAction <-- playerAction
   Output("Your Action:")
   Output(playerAction.getAction())
```

```
Procedure getHeading(input mySize : integer, input enemySize : integer, input
enemyDistance : float, input playerList:  list of player,input foodList :
list of food, gasCloudList : list of gasCloud)

{I.S parameter terdefinisi}
{F.S arah dari bot untuk tick selanjutnya terdefinisi}
{Proses menentukan semua kemungkinan menggunakan dan mengambil solusi terbaik
menggunakan algoritma greedy untuk arah bot pada tick selanjutnya }

Kamus Lokal
GasCloudOnSightList : List of Gas Cloud
IsUsingAfterBurner : bool
Function GetHeadingAvoid( objek : Gameobjek) -> integer
Function GetHeadingBetween(objek : Gameobjek) -> integer
Function GetNearestObjectOnSight(heading : integer, listgascloud : list of
gas clouds) -> list of object
Function GetDistanceBetweenWithSize (objek : gameobjek,objek : gameobjek )->
Float
Function GetHeadingFrom2(heading1 : integer, heading : integer) -> integer
Function getHeadingAvoidBorder(bot : bot) -> integer
Algoritma

If enemyDistance <= 200 and enemySize > mySize then
  { kondisi kalau ada enemy yang deket dan lebih besar }
  playerAction.heading <- getHeadingAvoid(playerList.get(1))
Else if mySize > enemySize and enemyDistance <= 6*mySize then
   playerAction.heading <- getHeadingBetween(playerList.get(1))
    { conditional kalau harus menggunakan afterburner }
   If enemyDistance<=2*mySize and enemyDistance>=0.5*mySize then
      bot.isUsingAfterBurner <-- true;
   else
       If foodList.size()!= 0 then
          { conditional tidak ada makanan lagi }
          playerAction.heading <-  getHeadingBetween(foodList.get(0))
          Output("Eat food");
        else
           playerAction.heading <- getHeadingAvoid(playerList.get(1));
```

```
        gasCloudOnSightList <- getNearestObjectOnSight(playerAction.heading,
gasCloudList);
        if  !(gasCloudOnSightList.size() == 0) then
            { kondisi saat tidak ada gascloud di depan bot}
            if getDistanceBetweenWithSize(bot, gasCloudOnSightList.get(0)) <=
50 then
            { kondisi saat bot kita lebih kecil dari 50 terhadap gas cloud }
               playerAction.heading <- getHeadingFrom2(playerAction.heading,
getHeadingAvoid(gasCloudOnSightList.get(0)))
            else if (getDistanceBetweenWithSize(bot,
gasCloudOnSightList.get(0)) <= 10 then
            { kondisi saat bot kita lebih kecil dari 10 terhadap gas cloud }
                PlayerAction.heading <-
getHeadingAvoid(gasCloudOnSightList.get(0))
                Output("Avoid Gas Cloud")

        if  getDistanceFromBorder(bot) <= 1*mySize then
            { Ketika terlalu mepet ujung }
            playerAction.heading <- getHeadingFrom2(playerAction.heading,
getHeadingAvoidBorder())
            Output("Avoid Border")
            if  getDistanceFromBorder(bot) <= 0.5*mySize then
               { Ketika sudah sangat dekat dengan get heading }
                playerAction.heading <- getHeadingAvoidBorder()
                Output("Avoid Border with critical distance")
```

```
Procedure getNextAction(input gasCloudList : list of gasCloud, input
playeList : list of player,input torpedoList : list of torpedo,input
asteroidFieldList : list of asteroidField)

{I.S parameter terdefinisi}
{F.S aksi dari untuk tick selanjutanya terdefinisi}
{Proses menentukan semua kemungkinan menggunakan dan mengambil solusi
terbaik menggunakan algoritma greedy untuk aksi bot pada tick selanjutnya}

Kamus Lokal
STARTAFTERBURNER : command
STOPAFTERBURBER : command
USESHIELD : command
FIRETORPEDOES : command
FORWARD : command
Function GetHeadingBetween(objek : gameobjek) -> integer
Function isWayClear( asteroid_gascloud : list of asteroid and gascloud ,
targer : gameobjek ) -> bool
Function isTorpedoGoingToHitMe ( listtorpedoes : list of torpedo ) -> bool
IsUsingAfterBurner : bool
GetTorpedoSalvoCount : integer

Algoritma
if bot.isUsingAfterBurner then
    { cek apakah bot sedang menggunakan afterburner }
   playerAction.action <- PlayerActions.STARTAFTERBURNER
   Output("Start AfterBurner")
else if getPlayerAction().getAction() == PlayerActions.STARTAFTERBURNER then
     { Cek apakah bot sudah harus mematikan afterburner }
      playerAction.action <- PlayerActions.STOPAFTERBURNER
      Output("Stop AfterBurner")
else
     If isTorpedoGoingToHitMe(torpedoList) && bot.getSize()>=25 then
        { Cek apakah harus menggunakan shield }
         playerAction.action <- PlayerActions.USESHIELD
         OUTPUT("Using Shield")
      else if bot.getTorpedoSalvoCount() != 0 && bot.getSize() > 50  &&
isWayClear(Stream.concat(gasCloudList.stream(),
asteroidFieldList.stream()).toList(), playerList.get(1)) then
         { Cek apakah harus menembakkan torpedoes }
               playerAction.action <- PlayerActions.FIRETORPEDOES;
               playerAction.heading <- getHeadingBetween(playerList.get(1))
               Output("Firing Torpedoes")
       else
          { Cek apakah harus maju }
               playerAction.action <- PlayerActions.FORWARD
               Output("Going Forward")
```

```
Function GetHeadingBetween(otherObject : gameobjek) -> integer
{ diberikan gameobjek mengeluarkan heading antara bot kita
dengan otherobjek }
Kamus Lokal
Direction : integer
Algoritma
{ Mendapatkan heading bot ke otherObject }
direction <- toDegrees(Math.atan2(otherObject.getPosition().y -
bot.getPosition().y,otherObject.getPosition().x -
bot.getPosition().x))
-> (direction + 360) % 360;

Function GetHeadingAvoid( objek : Gameobjek) -> integer
{ diberikan gameobjek mengerluarkan heading antara bot kita
dengan otherobjek }
Kamus Lokal
Function GetHeadingBetween(otherObject : gameobjek) -> integer
Algoritma
- > getHeadingBetween(otherObject) + 180) % 360

Function getHeadingAvoidBorder(bot : bot) -> integer
 { diberikan game objek mengeluarkan heading objek ke tengah }
Kamus Lokal
Position : position
Center : gameobjek
Algoritma
        position <- new Position(0, 0);
        center = new GameObject(null, null, null, null,
position, null, null, null, null, null, null);
        -> getHeadingBetween(center);

Function GetHeadingFrom2(heading1 : integer, heading : integer)
-> integer
{ diberikan dua heading mengeluarkan resultan antar dua heading
}
Kamus Lokal
min : integer
NewHeading1 : integer
NewHeading2 : integer
MinHeading : integer
Algoritma
min <- Math.min(heading1, heading2)
NewHeading1 <- (heading2 - heading1 + 360) % 360
NewHeading2 <- (heading1 - heading2 + 360) % 360
MinHeading <- Math.min(newHeading1, newHeading2)
-> (min + minHeading / 2 + 360) % 360
```

```
Function GetDistanceBetweenWithSize (objek : gameobjek,objek :
gameobjek )-> Float

Kamus Lokal
Algoritma
-> getDistanceBetween(objek1,objek2) - (object1.getSize()
+object2.getSize())
```

4.2 Penjelasan Struktur Data pada Bot Permainan Galaxio

Struktur data pada permainan galaxio berbentuk class yang terdiri dari 4 kategori ,yakni Enums berisi konstanta – konstanta objek yang ada permainan, Models berisi realisasi dari semua yang berapa di game seperti gameobject, gamestate, playeraction, dan lain-lain , Services berisi tempat logika bot berada dan tempat pengimplementasian algoritam greedt, dan terakhir ada Main.java . Semua yang kategori di atas sudah disediakan oleh Entelect kecuali Services. Berikut rincian dari 4 kategori tersebut :

a. Enums

Kategori ini berisi kelas – kelas yang anggotanya merupakan konstanta atau suatu nilai yang akan digunakan sepanjang permainan. Isinya merupakan enumerasi – enumerasi untuk objek yang ada pada permainan.

1. Effects.java

Berisi enumerasi objek yang dapat memberikan efek khusus pada bot,yakni AFTERBURNER, ASTEROID_FIELD, GAS_CLOUD, SUPER_FOOD, dan SHIELD.

Attributes	Description
value	Untuk menjadi nilai dari enum

Methods	Description
Effects	Untuk mengest nilai enum (setter)
getEffects	Mendapatkan nilai enum (getter)

2. ObjectTypes.java

Berisi enumerasi objek apa saja yang berada di dalam game, yakni PLAYER,FOOD,WORMHOLE, GAS_CLOUD, ASTEROID_FIELD, TORPEDOSALVO, SUPER_FOOD, SUPERNOVA_PICKUP, SUPERNOVA_BOMB, TELEPORTER, SHIELD.

Attributes	Description
value	Untuk menjadi nilai dari enum

Methods	Description
ObjectTypes	Untuk mengest nilai enum (setter)
valueOf	Mendapatkan nilai enum (getter)

3. PlayerActions.java

Berisi enumerasi aksi apa saja yang dapat dilakukan player, seperti FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES,FIRESUPERNOCA,DETONATESUPERNOVA,FIRETELPORTER,TELPORT,USESHIELD

Attributes	Description
value	Untuk menjadi nilai dari enum

Methods	Description
PlayerActions	Untuk mengambil nilai enum (getter)

b. Models
berisi realisasi dari semua yang berapa di game seperti gameobject, gamestate, playeraction, dan lain-lain
1. GameObject.java
Berisi semua isi objek di dalam game

Attributes	Description
id	Id player
size	Size dari player
speed	Speed dari player
currentHeading	Arah dari player
position	Posisi dari player
gameObjectType	Enum dari objek
effects	Efek dari game
torpedoSalvoCount	Banyak torpedo salvo
supernovaAvailable	ada atau tidaknya supernova
teleportCount	Counter cool down teleport
shieldCount	Counter shield
isUsingAfterBurner	Apakah memakai afterburner

Methods	Description
GameObject	konstruktor
getId	Get Id player
getSize	Get Size player
setSize	Set Size player
getSpeed	Get speed player
setSpeed	Set speed player
getPosition	Get posisi player
setPosition	Set posisi player

getGameObjectType	Get enum objectType
setGameObject	Set objectType
getTorpedoSalvoCount	Get torpedoSalvoCount
getSupernovaAvailable	Get supernovaAvailable
setSupernovaAvailable	Set supernovaAvailble
getTeleporterCount	Get TeleporterCount
setTeleportCount	Set TeleporterCount
getShieldCount	Get ShieldCount
setShieldCount	Set ShieldCount
FromStateList	Return game objek baru

2. GameState.java
Berisi state di game

Attributes	Description
world	Deskripsi dari world
gameObjects	List of game object di game
playerGameObjects	List of player di game

Methods	Description
GameState	Konstruktor
getWorld	Get World
setWorld	Set world
getGameObjects	Get list of gameobjects
setGameObjects	Set list of gameobjects
getPlayerGameObjects	Get list of playergameobjects
setPlayerGameObjects	Set list of playergameobjects

3. GameStateDTO
Berisi posisi setiap objek di dalam game

Attributes	Description
private World	Deskripsi dari world
private Map<String, List<Integer>> gameObjects	List of posisi game object di game
private Map<String, List<Integer>> playerObjects	List of posisi player di game

Methodnya berisi setter dan getter untuk setiap atribut tersebut

4. PlayerAction.java
Berisi aksi yang bisa dilakukan bot

Attributes	Description
public UUID playerId;	ID dari player
public PlayerActions action;	Action dari player
public int heading;	Arah heading dari player

Methodnya berisi setter dan getter untuk setiap atribut tersebut

5. Position.java
Berisi kelas posisi atribute terdiri dari sumbu x dan y , methodnya setter dan getter untuk setiap attributenya

6. World.java
Berisi deskripsi dari world

Attributes	Description
public Position centerPoint	Posisi tengah dari dunia
public Integer radius	Radius dari dunia
public Integer currentTick	Tick dari game

B. Services
1. BotService.java
Berisi logika dari bot
Memiliki atribute
private GameObject bot;
private PlayerAction playerAction;
private GameState gameState;

Dan selain method setter dan getter ada method
ComputeNextPlayerAction : mengisi heading dan aksi bot
GetDistanceBetween : mengeluarkan jarak antar dua objek
GetDistanceBetweenWithSize : mengerluarkan jarak antar dua objek dengan ukurannya
GetDistanceFromBorder : jarak anatr objek dengan border
GetHeadingBetween : mengeluarkan arah dari dua objek
GetHeadingAvoid : mengeluarkan arah untuk menghindar
GetHeadingavoidBorder : mengeluarkan arah untuk menghindari border
GetHeadingFrom2 : mencari resultan
GetNearestObjectOnSight : mencari objek atau player terdekat
IsCollide : mengeluarkan apakah mau nabrak atau tidak
IsInsideGas : apakah di dalam gas cloud atau tidak

IsWayClear : kalau target itu kosong
IsTorpedoGoingToHitMe : apakah torpedo mengenai player
ToDegrees : mengubah rad ke derajat

2. ObjectList.java
Semua objek yang dibutuhkan dalam penentuan strategi greedy.terdapat atribut

```
private List<GameObject> gasCloudList;
private List<GameObject> foodList;
private List<GameObject> playerList;
private List<GameObject> torpedoList;
private List<GameObject> asteroidFieldList;
```

Dan methodnya haya getter dari setiap attributenya.

C. Main.java
Berisi main program dari game
Methodnya berisi setter dan getter untuk setiap atributnya

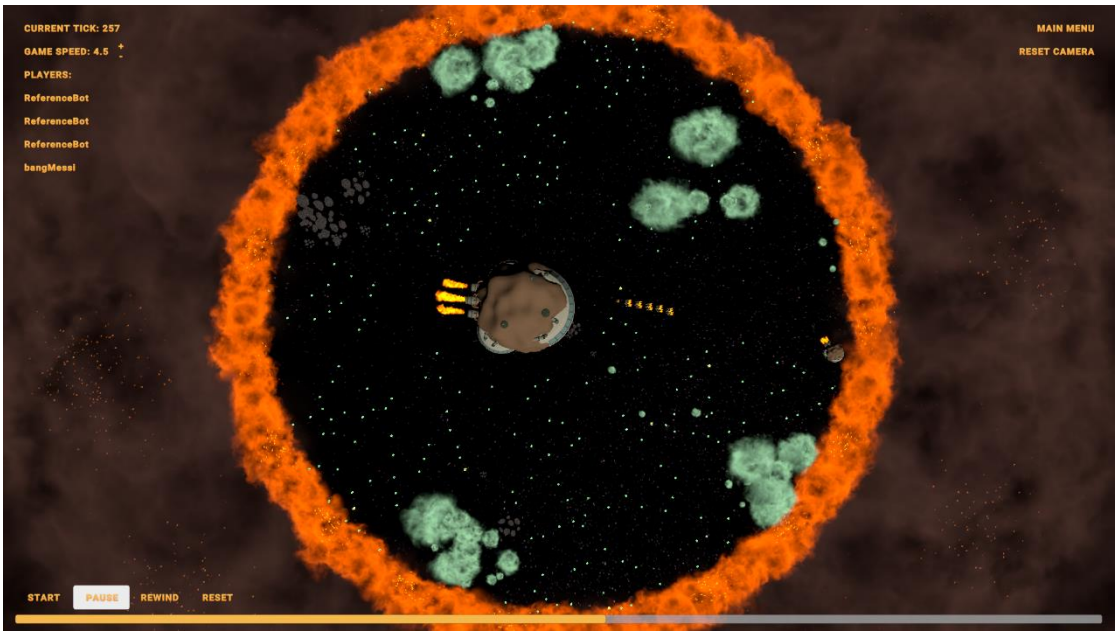
4.3 Pengujian Bot serta Analisis Performansi Bot Permainan Galaxio

Pengujian bot untuk mendapatkan analisis performansi kami lakukan dengan melakukan adu tanding bot bangMessi dengan beberapa bot lainnya yang bisa kami dapatkan. Bot yang kami gunakan diantaranya adalah ReferenceBot (default), EnlivenSquad, eres-el-mejor dan Yasin_bot.

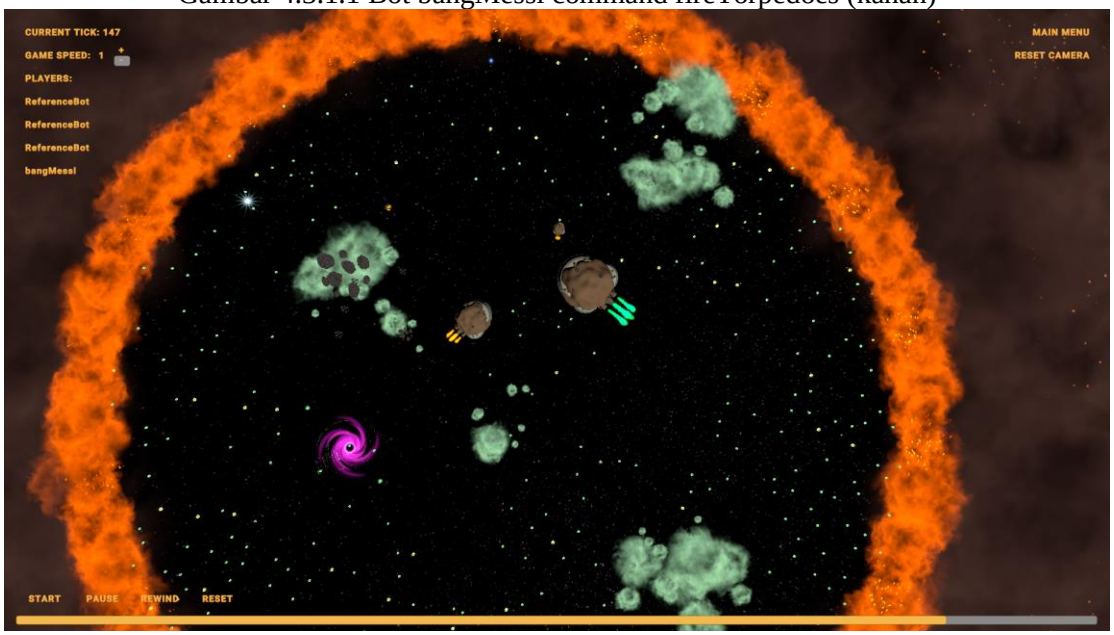
4.3.1 Melawan 3 ReferenceBot

Pada percobaan pertama, penulis melakukan percobaan dengan menggunakan 4 bot dalam satu match Galaxio. Bot yang digunakan pada percobaan pertama adalah:
Bot 1 : ReferenceBot
Bot 2 : ReferenceBot
Bot 3 : ReferenceBot
Bot 4 : bangMessi

Dari 10 kali percobaan, bot penulis (bangMessi) mampu memenangkan 7 dari 10 match yang ditandingkan dengan 3 ReferenceBot. Bot bangMessi mampu memenangkan 7 dari 10 match disebabkan karena bot bangMessi memiliki keunggulan fitur jika dibandingkan dengan ReferenceBot, diantaranya adalah adanya command fireTorpedoes dan startAfterBurner. Kedua command tersebut menurut penulis sudah cukup untuk memenangkan setidaknya peringkat dua jika ditandingkan dengan ReferenceBot.



Gambar 4.3.1.1 Bot bangMessi command fireTorpedoes (kanan)



Gambar 4.3.1.2 Bot bangMessi menggunakan command startAfterBurner

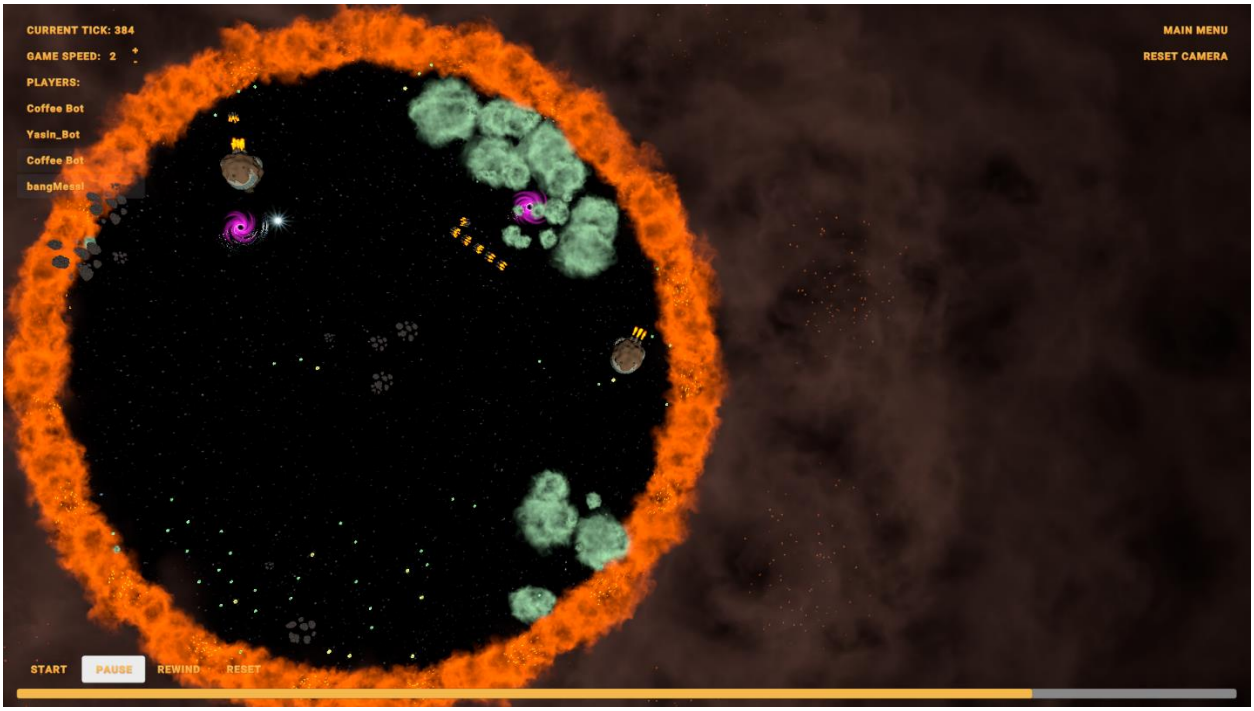
Pada gambar 4.3.1.1 dapat dilihat bahwa meskipun bot bangMessi kalah secara ukuran (kanan), ia dapat mengatur untuk menjauhkan dirinya dari ReferenceBot dan melakukan command fireTorpedoes untuk mengurangi ukuran size ReferenceBot. Pada gambar 4.3.1.2 dapat dilihat bahwa bot bangMessi menggunakan command startAfterBurner untuk mengejar ReferenceBot yang memiliki ukuran lebih kecil dan memenangkan match.

Strategi greedy yang digunakan dapat mencapai kondisi optimum karena bot bangMessi tidak mendapatkan disturbansi dari command ReferenceBot, sehingga strategi greedy dapat memenangkan lebih dari setengah match yang diujikan.

4.3.2 Melawan EnlivenSquad, eres-el-mejor, Yasin_bot

Pada percobaan kedua, penulis melakukan percobaan dengan menggunakan 4 bot dalam satu match Galaxio. Bot yang digunakan pada percobaan pertama adalah:
Bot 1 : EnlivenSquad
Bot 2 : Yasin_bot
Bot 3 : eres-el-mejor
Bot 4 : bangMessi

Dari 10 kali percobaan, bot penulis (bangMessi) hanya memenangkan 3 dari 10 match yang ditandingkan dengan 3 bot lainnya . Bot bangMessi mampu memenangkan 3 dari 10 match disebabkan karena bot bangMessi memiliki sudah memiliki lawan yang sepadan. Berbeda dengan percobaan sebelumnya, bot lawan sekarang sudah bisa melakukan fireTorpedoes dan startAfterBurner.



Gambar 4.3.2.1 Bot bangMessi menembakkan fireTorpedoes sekaligus menjaga jarak

Pada gambar 4.3.2.1 bot bangMessi menembakkan fireTorpedoes kearah yang tidak ada object lain (gas cloud/asteroid) agar torpedoes tidak menabrak object yang dapat mengurangi size bot jika mengenainya.

Solusi optimum tidak selalu dapat tercapai jika melakukan pengujian dengan 3 bot custom yang dibuat oleh kelompok lain. Hal tersebut terjadi karena bot custom yang kami ujikan memiliki strategi yang cukup mirip dengan bot bangMessi, yaitu dengan mengandalkan command fireTorpedoes. Dengan seluruh bot menggunakan command fireTorpedoes, kemungkinan sekeliling bot bangMessi akan dipenuhi dengan torpedoes sehingga dapat menghalangi potensi bot bangMessi untuk melakukan action lain. Solusi optimal dapat tercapai jika bot bangMessi mendapatkan spawn awal yang cukup bersih dari object penghalan (gas cloud/asteroid) sehingga dapat mendapatkan *early-buff* karena memiliki keuntungan dari segi size.

Bab 5

5.1 Kesimpulan

Kelompok kami berhasil mengimplementasikam algoritma *greedy* untuk membuat bot permainan galaxio yakni bisa mencapai tujuan objektif bertahan dari serangan bot lainnya. Dari hasil yang didapatkan, dapat dilihat bahwa penggunakan strategi *greedy* cukup optimal dalam kasus ini, dikarenakan pada permainan galaxio pemilihan paling menguntungkan dapat membuat bot pada game bertahan dari serangan maupun menghindari objek objek yang ada di dalam game.Untuk mengilustrasikan, misal pada suatu langkah kita melakukan *greedy* dengan mencoba meraih makanan terdekat dan mempriotaskan jika disekitar makanan tersebut ada lawan yang lebih besar maka kita akan memprioitaskan menghindar dari lawan tersebut.

5.2 Saran

beberapa saran yang bisa kita ajukan untuk selanjutnya:

1. Penulis menyarankan untuk berikutnya, pembuatan laporan dapat dilakukan lebih Efisien dan efektif agar tidak mendekati tanggal pengumpulan
2. Penulis menyarankan agar memilih bot yang dapat compatible di semua device

Daftar Pustaka

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)-Bag2.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

Link Penting

Link Github : https://github.com/fajarmhrwn/Tubes1_bangMessi