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

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Galaxio"

Disusun oleh:

Shelma Salsabila	13521115
Akhmad Setiawan	13521164
Satria Octavianus Nababan	13521168

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

2023

Bab 1: Deskripsi Tugas

Galaxio adalah sebuah *game battle royale* yang mempertandingkan beberapa bot kapal. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal tetap hidup hingga akhir permainan sehingga 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.

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

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde 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 cloud*s 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 *teleport*er tersebut. Harga setiap peluncuran *teleport*er 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 dikonsumsi oleh 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. Berikut jenis-jenis dari command yang ada dalam permainan:
    A. FORWARD
    B. STOP

C. START_*AFTERBURNER*
D. STOP_*AFTERBURNER*
E. FIRE_TORPEDOES
F. FIRE_*SUPERNOVA*
G. DETONATE_*SUPERNOVA*
H. FIRE_*TELEPORT*ER
I. *TELEPORT*
J. ACTIVATE_SHIELD

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

Bab 2: Landasan Teori

2.1. Gambaran Algoritma *Greedy* secara Umum

Algoritma *greedy* merupakan sebuah algoritma yang mengimplementasikan konsep "*greedy*" dalam pendefinisian solusinya. Algoritma *greedy* biasanya digunakan untuk mencari solusi dari persoalan optimisasi (*optimization problem*) untuk memaksimumkan atau meminimumkan suatu parameter. Algoritma *greedy* dibentuk dengan memecah permasalahan dan membentuk solusi secara langkah per langkah (*step by step*). Pada setiap langkah tersebut, terdapat banyak langkah yang dapat dievaluasi. Pada algoritma *greedy*, pemrogram harus menentukan keputusan terbaik pada setiap langkahnya. Tetapi, di dalam algoritma *greedy* tidak diperbolehkan adanya *backtracking* (tidak dapat melihat mundur ke solusi sebelumnya untuk menentukan solusi langkah sekarang). Oleh karena itu, diharapkan pemrogram memilih solusi yang merupakan optimum lokal (*local optimum*) pada setiap langkahnya. Hal ini bertujuan bahwa langkah-langkah optimum lokal tersebut mengarah pada solusi dengan optimum global (*global optimum*).

Suatu persoalan dapat diselesaikan dengan algoritma *greedy* apabila persoalan tersebut memiliki dua sifat berikut:

- Solusi optimal dari persoalan dapat ditentukan dari solusi optimal subpersoalan tersebut.
- Pada setiap persoalan, terdapat suatu langkah yang dapat dilakukan dimana langkah tersebut menghasilkan solusi optimal pada subpersoalan tersebut. Langkah ini juga dapat disebut sebagai *greedy choice*.

Terdapat beberapa elemen/kompenen yang perlu didefinisikan di dalam algoritma *greedy*. Beberapa elemen/komponen algoritma *greedy* tersebut adalah:

- Himpunan kandidat (C): Berisi kandidat yang mungkin dipilih pada setiap langkahnya.
- Himpunan solusi (S): Berisi kandidat yang sudah terpilih sebagai solusi.
- Fungsi solusi (*solution function*): Menentukan apakah himpunan solusi yang dikumpulkan sudah memberikan solusi. (Domain: himpunan objek, Range: boolean).

- Fungsi seleksi (*selection function*): Memilih kandidat berdasarkan strategi *greedy* tertentu. Fungsi ini memiliki sifat heuristik (fungsi dirancang untuk mencari solusi optimum dengan mengabaikan apakah fungsi tersebut terbukti paling optimum secara matematis). (Domain: himpunan objek, Range: objek).
- Fungsi kelayakan (*feasibility function*): Memeriksa apakah kandidat yang terpilih oleh fungsi seleksi dapat dimasukkan ke dalam himpunan solusi. (Domain: himpunan objek, *Range*: boolean).
- Fungsi objektif (*objective function*): Memaksimumkan atau meminimumkan suatu parameter pada suatu persoalan. (Domain: himpunan objek, Range: himpunan objek).

Dengan menggunakan elemen/komponen di atas, algoritma *greedy* dapat didefinisikan sebagai berikut: "Algoritma *greedy* merupakan pencarian sebuah himpunan bagian S dari himpunan kandidat C, dimana S memenuhi kriteria kelayakan sebagai solusi paling optimum, yaitu S merupakan suatu himpunan solusi dan S dioptimisasi oleh fungsi objektif."

Skema umum algoritma *greedy* menggunakan *pseudocode* dengan pendefinisian elemen/komponennya adalah 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
```

Gambar 2.1 *Pseudocode* algoritma *greedy*

Pada akhir tiap iterasi, solusi yang terbentuk adalah optimum lokal, dan paka akhir *loop while-do* akan ditemukan optimum global, namun optimum global yang ditemukan ini

belum tentu merupakan solusi yang terbaik karena algoritma *greedy* tidak melakukan operasi secara menyeluruh kepada semua kemungkinan yang ada.

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

2.2. Garis Besar Cara Kerja Bot Permainan Galaxio

Adapun cara kerja bot dalam *game* galaxio ini adalah ketika semua bot sudah terkoneksi pada runner , maka bot-bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang penting adalah *RecieveGameState* karena memberikan status *game*. Selain itu bot juga dapat mengirim aksi-aksi yang ingin dilakukan seperti pergi kesuatu arah tertentu, aksi ini dikirim ke runner. Antara bot satu dan bot lain saling beradu untuk memperoleh kemenangan. Setelah semua bot kalah dan menyisakan satu bot yang masih hidup disebut “winner” maka akan terbuat dua file json yang berisi kronologi match. Dua file ini digunakan jika kita ingin melihat permainan berlangsung secara lebih jelas dengan cara menginputkan kedua file ini ke visualizer. Mengenai runner dan visualizer akan dibahas di bagian 2.4.

2.3. Implementasi Algoritma *Greedy* ke dalam Bot Permainan Galaxio

Untuk membuat bot untuk *game* galaxio ini bisa digunakan beberapa algoritma. Contohnya dengan algoritma brute force atau *greedy*. Kedua algoritma ini mempunyai kelebihan dan kekurangannya masing-masing. Algoritma brute force dengan semua ketepatan pemilihan solusi namun kompleksitas algoritmanya cukup besar. Adapun algoritma *greedy* yang bisa dikatakan merupakan algoritma yang lebih mangkus dibandingkan brute force hanya saja solusi yang dihasilkan tidak selalu tepat. Algoritma *greedy* hanya memikirkan bagaimana menghasilkan solusi terbaik untuk saat ini tanpa memikirkan apakah itu solusi yang terbaik secara global atau hanya lokal.

Pada pembuatan bot galaxio kali ini yang digunakan adalah algoritma *greedy*. Pemilihan algoritma *greedy* karena penulis merasa algoritma ini cukup lebih mangkus dibanding brute force meskipun solusi yang dihasilkan tidak selalu optimum global. Pemilihan algoritma *greedy* ini juga didasarkan pada aturan yang ada dispesifikasi tugas.

2.4 Garis Besar *Game Engine* Permainan Galaxio

Untuk menjalankan *game* galaxio bisa mengunduh starter pack yang ada di github *Entelectchallenge*. Di dalam starter pack itu ada beberapa folder yang cukup penting untuk dipahami yaitu *engine-publish ,logger-publish, runner-publish, starter*-bots, serta *visualiser*. Isi dari folder di atas adalah.

- **Folder *Engine-Publish***

  Sebelum menjalankan bot *game* galaxio ada tiga komponen penting yang harus dijalankan terlebih dahulu yakni, Di dalam folder engine-publish terdapat komponen yang berperan dalam mengimplemen dan rules *game*. Engine ini merupakan komponen yang cukup penting di dalam menjalankan *game* galaxio. Engine ini berfungsi untuk menghubungkan bot dengan runner, engine akan menunggu semua bot terhubung dengan runner.

- **Folder *Runner-Publish***

  Didalam folder runner-publish terdapat komponen yang berperan dalam menggelar sebuah *match* serta menghubungkan bot dengan *engine.* Ketika runner ini dijalankan maka runner akan meng-host sebuah match pada sebuah hostname tertentu. Untuk koneksi lokal runner akan meng-host pada localhost:5000.

- **Folder *Logger-Publish***

  Didalam folder logger-publish komponen yang ada berperan untuk mencatat log permainan sehingga kita dapat mengetahui hasil permainan. Logger juga akan digunakan sebagai input dari visualizer. Logger juga berfungsi untuk melakukan koneksi dengan runner.

Selain file *game engine*, terdapat beberapa file dan folder yang perlu ditambahkan pada *starter-pack* oleh pemrogram. Beberapa file dan folder yang wajib ada, agar *game engine* dapat dijalankan adalah diantaranya:

├── engine-publish
├── logger-publish
├── reference-bot-publish
├── runner-publish
├── starter-bots
├── visualiser
├── building-a-bot.md
├── README.md
└── run.sh

Sebelum penulis mendefinisikan algoritma pada file BotService.java, penulis terlebih dahulu melengkapkan pendefinisian file diatas. Untuk dapat menjadikan kode menjadi bot yang dapat dipakai, source code harus di-*build* menjadi sebuah file *.jar* terlebih dahulu. Salah satu caranya adalah dengan menggunakan maven ataupun intelliJ. Apabila menggunakan maven, dapat diakukan dengan meng-*install* maven terlebih dahulu. Kemudian, untuk *build source code*-nya, dilakukan dengan mengubah *directory* (cd) ke *folder java*Bot kemudian memasukkan *command* berikut di terminal: *mvn clean package*. Setelah beberapa saat, maka folder target akan terbentuk yang berisi *executable* file *jar*. *Rename* file *.jar* tersebut menjadi nama kelompok terlebih dahulu. Kemudian, untuk menggunakan file *.jar* tersebut ke permainan, dapat mengubah *dotnet Reference*Bot*.dll* pada *script* menjadi *java -jar path.* Dengan path adalah path menuju file nama_kelompok.jar hasil *build.*

Selain itu, pada program ini juga telah disediakan *build tools* dari Apache Maven Project. Intellij IDEA telah menyediakan *build tools* Maven sehingga penulis tidak perlu menambahkan *extension* apapun di dalam IDEnya. Terdapat beberapa *command* dari *build lifecycle* yang dapat dijalankan oleh Maven, diantaranya adalah berikut ini:

Pada program Galaxio ini, untuk melakukan kompilasi perubahan *source code* dan kemudian mengubahnya ke dalam bentuk file jar, pemrogram hanya perlu menjalankan perintah *compile* dan *install* saja. Pemrogram tidak perlu melakukan semua perintah pada *build lifecycle*.

- **Folder *Starter*-Bots**

  Didalam folder starter-bots ada beberapa bot yang dibuat dengan beberapa bahasa. Diantaranya ada dalam bahasa java, c# dan sebagainya. Bot yang dibuat oleh penulis

juga disimpan di folder ini, secara spesifik disimpan di dalam folder “Service” dengan nama file BotService.java karena bot yang dibuat menggunakan bahasa Java. Di dalam folder ini juga terdapat beberapa file yang berisi fungsi-fungsi untuk mendukung dalam proses membuat bot.

- **Visualiser**

  Visualiser berfungsi untuk menjalankan *game* melalui suatu interface.

Adapun cara menjalankan *game* galaxio adalah sebagai berikut.

- Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON ”appsettings.json” dalam folder “runner-publish” dan “engine-publish”
- Buka terminal baru pada folder runner-publish.
- Jalankan runner menggunakan perintah “dotnet *Game*Runner.dll”
- Buka terminal baru pada folder engine-publish
- Jalankan engine menggunakan perintah “dotnet Engine.dll”
- Buka terminal baru pada folder logger-publish
- Jalankan engine menggunakan perintah “dotnet Logger.dll”
- Jika ingin menggunakan reference bot maka bisa gunakan perintah ”dotnet ReferenceBot.dll” namun jika ingin menggunakan bot buatan maka jar terlebih dahulu bot buatan dengan maven atau intelejID. Kemudian jika menggunakan jar maka perintah yang ditulis adalah java -jar <path-name-jar>.
- Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON “*Game*StateLog_{Timestamp}” dalam folder “logger-publish”. Kedua file tersebut diantaranya *Game*Complete (hasil akhir dari permainan) dan proses dalam permainan tersebut.
- *Game*StateLog yang dihasilkan dapat diinputkan ke visualizer untuk melihat permainan dengan lebih jelas, karena jika menggunakan terminal kurang bisa dipahami.

Bab 3: Aplikasi Strategi *Greedy*

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

3.1.1. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Penentuan *Command* untuk *General* Bot

Dalam permainan Galaxio, tujuan setiap bot pemain berusaha untuk menjadi yang paling terakhir untuk bertahan hidup sehingga memenangkan permainan. Dalam prosesnya, cara yang bisa dilakukan ialah dengan mengonsumsi *food* sehingga ukuran membesar lalu menabrak musuh yang lebih kecil selagi menghindar dari musuh yang lebih besar, menembak musuh, menghindari *gas cloud* dan tembakan musuh, atau bahkan menunggu lawan melakukan kesalahan sehingga lawan mati dengan sendirinya. Implementasi algoritma pada permasalahan ini dapat dilihat pada tabel berikut.

Tabel 3.1.1 Algoritma *Greedy* Permasalahan General Bot

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	*Command-command* yang tersedia untuk menggerakkan aksi bot, dengan setiap *tick* satu *command*
Himpunan solusi	Urutan comma*nd-command* yang terpilih pada setiap tick
Fungsi solusi	Memeriksa apakah pemilihan *command* sudah tepat sesuai dengan prioritas bertahan hidup bot
Fungsi seleksi	Memilih *command* yang terbaik pada *gamestate* terbaru di setiap *tick* dengan mempertimbangkan prioritas aksi yang paling terpenting untuk menjaga bot tetap bertahan hidup
Fungsi kelayakan	Memeriksa apakah *command* yang dituliskan oleh bot merupakan *command* yang valid seperti yang telah ditetapkan

Fungsi objektif	Memilih *command-command* yang mampu menggerakkan aksi bot sehingga bot masih terus bertahan hidup untuk memenangkan permainan

3.1.2. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Bot agar Tidak Keluar dari Radius Arena

Agar membuat bot dapat memenangkan permainan maka, bot tidak boleh keluar dari area yang ada yang semakin mengecil. Area berbentuk lingkaran dengan titik pusat di tengah posisi koordinat (0, 0). Menjadi bagian hal yang penting untuk kemenangan bot dalam permainan ini, strategi *greedy* untuk permasalahan bot agar tidak keluar arena harus dibuat juga dan menjadi prioritas pertama dan dijelaskan pada tabel berikut.

Tabel 3.1.2 Algoritma *Greedy* Menjauhi Batas Arena

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	*Command* untuk menjauhi batas luar dari arena permainan
Himpunan solusi	Batas area permainan yang paling dekat dengan bot
Fungsi solusi	Melakukan pengecekan apakah jarak antara bot dengan batas arena terdekat kurang dari radius arena dikurang tiga per dua ukuran bot, jika ya maka dia akan bergerak ke arah pusat (0, 0)
Fungsi seleksi	Menyeleksi jika jarak memenuhi apa yang disebutkan disolusi maka pergi ke pusat arena permainan, jika jaraknya aman maka dapat melakukan *command* lain.
Fungsi kelayakan	Batas area luar layak dijauhi ketika jarak antara bot dan batas luar kurang dari radius area dikurangi tiga per dua ukuran bot

Fungsi objektif	Mencari command yang tepat ketika bot berdekatan dengan area luar arena agar tidak bergerak keluar batas

3.1.3. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Menjauhi *Gas cloud*

Gas cloud adalah objek yang dapat membuat ukuran bot berkurang 1 setiap 1 tick ketika berada di area gas. Menjauhi *gas cloud* adalah salah satu cara untuk mempertahankan ukuran bot agar tidak mengecil. *Gas cloud* tersebar banyak di peta. Untuk itu, algoritma *greedy* dibutuhkan untuk menghindari *gas cloud* terdekat dengan bot kita dan menjadi prioritas kedua agar tidak membuat bot menjadi kecil. Algoritmanya dijelaskan pada tabel berikut.

Tabel 3.1.3 Algoritma *Greedy* Menjauhi *Gas cloud*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Himpunan *gas cloud* yang ada di sekitar bot
Himpunan solusi	*Gas cloud* yang ada di sekitar bot yang berjarak kurang dari ukuran bot terhitung dari jarak *gas cloud* ke bot
Fungsi solusi	Memeriksa apakah jarak bot pemain terhadap jarak *gas cloud* sudah dianggap dekat (kurang dari ukuran bot)
Fungsi seleksi	Bot akan menjauhi *gas cloud* pergi ke arah yang berlawanan ketika jarak antara *gas cloud* dan bot kurang dari ukuran bot (dianggap sudah dekat). Jika tidak itu bukan suatu hal yang harus dipertimbangkan.
Fungsi kelayakan	*Gas cloud* layak dijauhi ketika jaraknya kurang dari ukuran bot
Fungsi objektif	Mencari *command* yang membuat bot pemain tidak terkena *gas cloud* sehingga tidak mengecil

3.1.4. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Mengejar Musuh Terdekat dengan Ukuran Lebih Kecil dan Menghindari Musuh Terdekat dengan Ukuran Lebih Besar

Salah satu cara untuk memenangkan pertandingan adalah dengan bermain secara agresif, yakni menabrakkan bot ke kapal musuh. Tentunya, jika bot berukuran lebih besar daripada ukuran kapal musuh. Jika berhasil menabrak bot lawan yang lebih kecil, ukuran bot akan bertambah jauh lebih cepat dibandingkan dengan hanya memakan makanan biasa. Namun sebaliknya, jika bertemu dengan kapal yang lebih besar, bot harus menghindar agar tidak ditabrak musuh. Oleh karenanya hal ini menjadi prioritas ketiga dengan penjelasan seperti pada tabel berikut.

Tabel 3.1.4 Algoritma *Greedy* Aksi terhadap Kapal Musuh

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	*Command* untuk mendekati kapal musuh yang kecil atau menjauhi kapal musuh yang besar
Himpunan solusi	Musuh-musuh terdekat beserta perbandingan ukurannya dengan kapal bot kita
Fungsi solusi	Mengecek apakah size bot kita lebih kecil atau lebih besar dari musuh-musuh terdekat
Fungsi kelayakan	Memeriksa apakah musuh berada dalam radius terjangkau oleh kapal bot
Fungsi objektif	Mencari *command* yang membuat bot mengejar musuh terdekat dengan ukuran yang lebih kecil dan melarikan diri dari musuh terdekat dengan ukuran yang lebih besar agar bertahan hidup dan memenangkan pertandingan

3.1.5. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Penggunaan *Afterburner*

Salah satu fitur menarik lainnya ialah bot pemain dapat mengaktifkan *Afterburner*, fitur yang dapat membuat bot bergerak dengan kecepatan 2x kecepatan semula. Penggunaan fitur ini dilakukan jika mengejar musuh yang memiliki ukuran lebih kecil (kecepatan musuh lebih besar) sehingga dapat menabrak musuh dengan lebih cepat.

Tabel 3.1.5 Algoritma *Greedy Afterburner*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Himpunan kondisi yang mengakibatkan penggunaan *afterburner*
Himpunan solusi	Kapal musuh terdekat memiliki ukuran yang lebih kecil (kapal musuh kecepatannya lebih besar)
Fungsi solusi	Memeriksa apakah kapal musuh yang terdekat memiliki ukuran yang lebih kecil dari bot pemain
Fungsi seleksi	Jika kapal musuh berada pada radius yang terdekat dan ukuran bot > 2x ukuran kapal musuh, bot pemain akan mengaktifkan *afterburner* untuk mengejar musuh tersebut
Fungsi kelayakan	Jika ukuran bot pemain > 60 untuk menghindari kapal terlalu kecil akibat penggunaan *afterburner*
Fungsi objektif	Menjalankan *command* yang membuat bot pemain mengaktifkan *afterburner* dan menghentikannya jika kondisi telah selesai digunakan

3.1.6. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Torpedo salvo shoot* (Menembak Musuh)

Bot dapat menembakkan *torpedo salvo* dan membuat musuh yang terkena ukurannya berkurang dan bot kita ukurannya bertambah. Namun, penggunaan *torpedo salvo* menyebabkan ukuran bot kita awalnya juga berkurang, dan tidak

akan bertambah lagi jika torpedo tidak terkena musuh. Oleh karenanya, agar tidak sia-sia, bot akan menembakkan *torpedo salvo* jika berada pada radius tertentu saja yang diuraikan pada tabel berikut.

Tabel 3.1.6 Algoritma *Greedy* Menembakkan *Torpedo Salvo*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Himpunan musuh-musuh yang ukurannya lebih besar
Himpunan solusi	Musuh-musuh yang berada pada radius < 100 agar *chance* tembakan tidak meleset semakin besar
Fungsi solusi	Memeriksa apakah kapal musuh berada di radius < 100, dan apakah ukuran kapal musuh lebih besar daripada bot pemain. Jika ya, menembakkan torpedo dan jika tidak akan melakukan *command* lain
Fungsi seleksi	Jika kapal musuh berada pada radius yang ditentukan, bot pemain akan menembakkan *torpedo salvo* untuk melawan musuh
Fungsi kelayakan	Jika *torpedo salvo* count lebih dari 1 dan ukuran bot kita tidak kurang dari 10
Fungsi objektif	Menjalankan *command* yang membuat bot pemain menembakkan *torpedo salvo* sehingga membuat musuh kalah

3.1.7. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Penggunaan *Wormhole*

Selanjutnya, subpermasalahan yang dapat diselesaikan dengan algoritma *greedy* adalah subpermasalahan bagaimana cara memanfaatkan *wormhole* sehingga memberikan efisiensi waktu maupun gerakan bot dalam menghindari kejaran musuh yang lebih besar dan mengejar musuh berukuran lebih kecil pada jarak yang cukup jauh. Hal ini dapat menjadi salah satu strategi yang untuk bot bisa bertahan hingga akhir dan memenangkan permainan.

Tabel 3.1.7 Algoritma *Greedy* Menggunakan *Wormhole*

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Himpunan dari *wormhole* yang terdekat dengan bot
Himpunan solusi	Penggunaan *wormhole* disaat kondisi yang tepat yaitu ketika menghindari musuh yang lebih besar atau mengejar bot musuh dengan jarak yang cukup jauh
Fungsi solusi	Melakukan pengecekan apakah *wormhole* terdekat akan membantu bot pemain menghindari kejaran musuh yang lebih besar dan mendekatkan bot pemain ke musuh yang ukurannya lebih kecil
Fungsi seleksi	Memeriksa salah satu *wormhole* yang terdekat dengan bot pemain dan apakah letak *wormhole* yang lain berada di area yang aman
Fungsi kelayakan	Memeriksa apakah ukuran *wormhole* lebih besar dari ukuran bot pemain sehingga *wormhole* dapat dimasuki
Fungsi objektif	Mencari *command* untuk dapat menggunakan *wormhole* dan membantu bot pemain menghindari kejaran musuh atau mengejar musuh

3.1.8. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Penggunaan *Teleport*

Selanjutnya, terdapat fitur *teleport* yang memungkinkan bot untuk berpindah ke suatu titik lokasi yang dituju dengan lebih cepat. *Teleport* memungkinkan bot untuk menghindar lebih cepat jika sedang dikejar musuh maupun mengejar musuh dengan lebih cepat. Pada intinya pengunaan *teleport* akan memindahkan bot ke suatu titik lokasi yang dituju dalam satu tick. Bot dapat menembakkan *teleport*er lalu dapat mengaktifkan *command teleport* untuk langsung berpindah.

Tabel 3.1.8 Algoritma *Greedy Teleport*

Nama	Definisi Elemen/Komponen

Elemen/Komponen	
Himpunan kandidat	Himpunan titik-titik lokasi yang dituju dan kumpulan aksi penggunaan *teleport*
Himpunan solusi	Titik lokasi yang aman yang dituju oleh bot untuk berpindah menggunakan *teleport*er
Fungsi solusi	Melakukan pengecekan apakah titik lokasi yang dituju terdapat *gas cloud*, asteroid atau musuh yang lebih besar yang malah akan memberikan kekalahan bagi bot
Fungsi seleksi	Memilih titik lokasi yang semakin mendekatkan kepada tujuan yang menguntungkan bot pemain, titik lokasi yang dianggap bot pemain dipilih untuk sebagai lokasi tujuan pemindahan bot dengan *teleport* jika mendekatkan kepada musuh jika sedang mengejar musuh dan memilih titik lokasi yang semakin menghindari musuh jika sedang dikejar musuh
Fungsi kelayakan	Memeriksa apakah ukuran bot > 100 untuk menghindari pengecilan ukuran bot pemain secara drastis, dan apakah jumlah *teleport*er yang dimiliki > 0
Fungsi objektif	Mencari *command* untuk menggunakan *teleport* sehingga dapat memindahkan bot ke titik lokasi tujuan

3.1.9. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Mengaktifkan *Shield*

Bot dapat mengaktifkan *shield*, yaitu salah satu fitur dalam permainan untuk mengeluarkan pelindung dari serangan torpedo lawan. Dalam penggunaannya, *shield* akan diaktifkan ketika ada serangan musuh dengan radius tertentu. Uraian lengkapnya dapat dilihat pada tabel berikut.

Tabel 3.1.9 Algoritma *Greedy Shield*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	*Command* yang memungkinkan bot pemain dapat mengaktifkan shield
Himpunan solusi	Kondisi yang mengakibatkan bot pemain lebih baik menggunakan shield pada tick itu juga
Fungsi solusi	Memeriksa apakah terdapat kondisi di mana bot pemain diuntungkan jika mengaktifkan *shield*, yakni jika ada torpedo musuh dalam radius 50 ataupun jika ukuran bot pemain dikurangi 20 masih lebih besar dari ukuran musuh
Fungsi seleksi	Jika musuh menyerang dengan torpedo dan jarak bot ke tembakan torpedo kurang dari radius 50 dan ukuran bot masih dianggap besar, maka *command shield* diaktifkan
Fungsi kelayakan	Memeriksa apakah ukuran bot lebih besar daripada 50, dan apakah recharge time shield dapat diaktifkan
Fungsi objektif	Mencari *command* untuk dapat menggunakan *shield* untuk berlindung dari tembakan *torpedo salvo* milik musuh

3.1.10. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Mencari dan Mengambil *Food* dan *Superfood*

Salah satu hal yang harus diperhatikan ketika ingin memenangkan suatu

pertandingan oleh sebuah bot adalah dengan mengomsumsi *food* sebanyak-banyaknya agar memperoleh ukuran yang besar dan bertahan hidup. Adapun, jika ingin mendapatkan suatu efek tambahan bisa dengan cara memakan super*food*. Dengan super*food* ini makanan yang dikonsumsi selama 5 tick nilainya dari makanan itu menjadi dua kali lipatnya. Uraian algoritma *greedy* untuk makanan dapat dilihat pada tabel berikut.

Tabel 3.1.10 Algoritma *Greedy* Pemilihan *Food* atau Super*food*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Himpunan dari aksi mendekati dan memakan *food* atau super*food*
Himpunan solusi	Himpunan *food* ataupun super*food* yang terdekat dengan bot pemain
Fungsi solusi	Melakukan pengecekan apakah ada *food*/super*food* terdekat
Fungsi seleksi	Bot akan mem-prioritaskan super *food* jika jarak ke super*food* kurang dari atau sama dengan 1,5 jarak ke *food* biasa, dengan tujuan memperoleh efek 2x lipat mengonsumsi *food* biasa. Namun, jika lebih jauh dari itu maka bot akan memprioritaskan *food* biasa saja
Fungsi kelayakan	*Food* dianggap layak dimakan jika jarak antara *food* cukup aman dari bahaya seperti musuh yang lebih besar ataupun bahaya *gas cloud* dan torpdeo salvo musuh. Hal ini berlaku juga untuk super*food*
Fungsi objektif	Mencari command yang membuat bot mendapatkan super*food*/*food*

3.1.11. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan Pengambilan dan Penggunaan *Supernova*.

Untuk dapat memenangkan permainan dan mendapatkan *score* yang tinggi maka dapat memanfaatkan fitur *supernova* yang dapat diledakkan dan memberi *damage* ke *player* yang berada dalam zona. Area ledakan akan berubah menjadi *gas*

cloud. Akan tetapi, *supernova* hanya muncul satu kali ada permainan di antara quarter pertama dan quarter terakhir, sehingga dibutuhkan keakuratan dalam penggunaan *supernova* sehingga efisiensi dari pengunaannya dapat mengalahkan musuh dan memberikan kemenangan bagi bot kita. Dalam hal ini, *supernova* akan ditembakkan ke pusat arena dengan harapan zona efek *gas cloud* dari *supernova* dapat mengenai banyak musuh.

Tabel 3.1.11 Algoritma *Greedy* Penggunaan *Supernova*

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan kandidat	Kumpulan aksi pengambilan dan penggunaan *supernova*
Himpunan solusi	Aksi dikumpulkan ketika tertentu antara bot dengan *supernova* cukup dekat dan bot pemain berjarak aman dari musuh
Fungsi solusi	Melakukan pengambilan jika terdapat *supernova* pada radius tententu dari bot (terdekat) dan memastikan keakuratannya untuk ditembakkan ke pusat arena dengan anggapan banyak musuh berkumpul dalam zona pusat arena
Fungsi seleksi	Menyeleksi radius bot terhadap *supernova*, jika jaraknya diluar dari radius yang ditentukan maka tidak mengambil *supernova* dan menyeleksi bahwa arah tembakan *supernova* tepat di pusat arena
Fungsi kelayakan	Mengambil *supernova* dalam radius tertentu dan menembakkan *supernova* jika sudah akurat terhadap pusat arena
Fungsi objektif	Mencari *command* untuk mengambil *supernova* yang terdekat dan *command* untuk menembakkan *supernova* jika sudah akurat menuju pusat arena

3.2 Eksplorasi Alternatif Solusi Algoritma *Greedy* pada Bot Permainan Galaxio

Terdapat banyak sekali alternatif solusi algoritma *greedy* yang dapat diimplementasikan pada bot permainan Galaxio. Dalam permainan galaxio suatu bot dapat memenangkan pertandingan ketika dia memiliki ukuran yang besar dan dapat bertahan hingga akhir permainan. Mewujudkan hal itu tidaklah mudah beberapa strategi algoritma *greedy* perlu diterapkan. Strategi *greedy* yang diterapkan dalam permainan adalah sebagai berikut.

3.2.1 Strategi Menghindari dan Mengejar musuh

Seperti yang telah diketahui sebelumnya, bot akan memenangkan pertandingan ketika *score*-nya lebih besar dan dapat bertahan hingga akhir, adapun ketika ukuran bot mengecil dan kurang dari 5 maka bot akan musnah. Untuk itu ketika bot yang dibuat ingin menang ukuran dari bot harus dipertahankan.

Beberapa hal dapat mempengaruhi ukuran bot salah satunya adalah ketika bot bisa menabrakan kapalnya dengan kapal musuh yang memiliki ukuran lebih kecil dari kapal dirinya maka bot akan bertambah ukurannya. Namun sebaliknya ketika bot ditabrak oleh bot musuh yang memiliki ukuran yang lebih besar dari bot kapal maka kapal akan hancur dan kapal musuh akan mendapatkan tambahan ukuran.

Tentunya hal yang diinginkan adalah bot dapat menghindar dari bot musuh yang memiliki ukuran yang lebih besar namun, mendekati bot yang memiliki ukuran yang lebih kecil. Strategi *greedy* yang dilakukan adalah bot secara *greedy* berusaha mencari optimum lokal.

Ada 2 strategi yang diterapkan

- Strategi *greedy* mengejar musuh. Bot akan berusaha mencari musuh yang ukurannya lebih kecil dari bot serta jarak diantara keduanya cukup dekat (secara spesifik paling dekat diantara bot dengan musuh yang lain dimana ukuran musuh itu lebih kecil dari bot. Namun, tidak semena-mena bot akan mengejar musuh itu, ada suatu batasan yang tidak boleh dilalui oleh bot yaitu jika ada musuh yang lebih besar dari bot yang sama-sama mengejar musuh yang sama maka bot akan mundur hal ini dilakukan untuk

mengurangi resiko dibunuh oleh bot yang lebih besar. Jika jarak antara bot dengan musuh cukup jauh namun jika dirasa dengan mempercepat kecepatannya dua kali dapat menempuh bisa memanfaatkan *afterburner.*

- Strategy *greedy* menghindari musuh. Bot akan berusah menghindari musuh yang ukurannya lebih besar daripada bot. Bot akan selalu menjaga jarak dengan musuh yang lebih besar. Hal ini dilakukan untuk menghindari bot yang berniat ingin membunuh bot.

Penulis berpendapat strategi ini cukup baik diterapkan karena pada dasarnya untuk menghindar ataupun mengejar kita tidak punya terlalu banyak pilihan cara selain memanfaatkan peluang yang pertama kali ditemui.

3.2.2 Strategi Bot Tidak Keluar Arena

Hal lain yang harus diperhatikan agar bot memenangkan sebuah pertandingan adalah menjaga bot agar tetap di arena permainan. Arena permainan *game* galaxio ini berbentuk layaknya lingkaran. Arena ini memiliki suatu radius tertentu yang akan terus berkurang menyesuaikan dengan ukuran bot yang ada dalam permainan. Berkaitan dengan ini menjadi suatu hal penting untuk menentukan strategi dari algoritma *greedy* untuk membuat bot agar tidak keluar dari arena permainan. Strategi yang dilakukan adalah, bot akan berusaha sebaik mungkin tidak mendekati batas paling luar permainan, jika terlalu dekat maka arah bot akan dirubah ke pusat dari arena. Suatu bot dikatakan terlalu dekat dengan batas luar arena permainan jika jarak antara bot dengan batas arena terdekat kurang dari radius arena dikurang tiga per dua ukuran bot. Adapun strategi yang diterapkan untuk menghindari keluar dari arena atau area permainan adalah dengan selalu menjaga jarak antara bot dengan jarak luar arena agar tidak terlalu dekat. Ketika sudah terlalu dekat maka bot akan diarahkan menuju pusat arena.

Menurut Penulis strategi ini sudah cukup baik diterapkan karena pada dasarnya tidak ada pilihan lain yang bisa kita lakukan jika berhubungan dengan garis luar permainan selain berusaha untuk menjauh dari garis luar itu. Strategi

3.2.3 Menjauhi *Gas cloud*

Salah satu bagian lain yang harus dihindari adalah *gas cloud*. *Gas cloud* dapat membuat ukuran bot berkurang 1 setiap 1 tick. Hal ini cukup merugikan bot untuk itu menghindari *gas cloud* adalah hal yang penting untuk dilakukan. Demi mencapai tujuan itu perlu dibuat sebuah strategi yang tepat. Sama seperti hal nya mencegah agar bot tidak keluar arena, untuk menghindari *gas cloud* ini juga bot akan diatur ketika jarak antara bot dengan *gas cloud* tidak kurang dari ukuran bot maka tidak akan dilakukan apa-apa. Ketika ukurannya kurang dari itu maka bot akan menjauh dengan pergi ke arah yang berlawanan. Penerapan *greedy* pada strategi ini adalah ketika penentuan langkah apakah akan menjauhi atau diam saja.

Menurut penulis, strategi ini merupakan strategi yang cukup ideal untuk memperlakukan objek *gas cloud*, tidak ada pilihan selain menjauhi atau tetap berada disana dengan suatu jarak yang konstan.

3.2.4 Strategi pada Permasalahan *Afterburner*

Afterburner dapat membuat kecepatan bot dua kali lebih cepat dibanding kecepatan aslinya. Hal ini dapat dimanfaatkan untuk mengejar musuh yang lebih kecil. Pada dasarnya ukuran musuh yang lebih kecil punya kecepatan yang lebih besar dibanding bot. Untuk mengimbangi itu bisa digunakan *afterburner*. Ada suatu prasyarat yang harus dipenuhi ukuran bot harus lebih dari 60 karena penggunaan *afterburner* ini akan mengurangi ukuran dari bot. Meskipun begitu dari analisis penulis ketika berhasil mendapatkan lawan, ukuran bot yang berkurang akibat *afterburner* ini dapat tergantikan bahkan berkali-kali lipat.

3.2.5 Strategi pada Permasalahan *Torpedo salvo shoot* (Menembak Musuh)

Strategi berikutnya adalah strategi untuk menembak musuh. *Torpedo salvo shoot* merupakan salah satu fitur yang ada pada *game* galaxio. Fitur ini membuat kapal satu dengan yang lainnya dapat menembakkan sebuah tembakan. Tembakan ini dapat membuat ukuran kapal yang terkena tembakkan menjadi berkurang ukurannya dan kapal yang menembakkan akan bertambah ukurannya. Fitur ini cukup penting dalam meberikan kemenangan kepada bot. Dengan fitur ini, bot

tidak harus selalu menjauhi musuh yang ukurannya lebih besar. Tapi, dapat melakukan perlawanan dengan melakukan penembakkan.

Bot dapat menembak musuh dengan suatu aturan tertentu, bot harus memiliki *torpedo salvo* terlebih dahulu. Bot akan memiliki *torpedo salvo* 10 tick sekali dengan maksimal jumlah *torpedo salvo* yang dimiliki hanya 5 buah. Jika telah mencapai 5 buah maka 10 tick berikutnya bot tidak akan mendapatkan torpedo salvo.

Torpedo salvo shoot harus digunakan dengan baik dan efisien. Oleh karena itu maka dibuatlah sebuah strategi secara *greedy* untuk mengatur ini. Ketika bot bertemu dengan musuh yang ukurannya lebih besar serta jarak diantaranya kurang dari 100 maka bot akan menambakkan *Torpedo salvo shoot* ini. *Torpedo salvo shoot* tidak akan ditembakkan ketika ukuran musuh lebih kecil, karena jika ukuran musuh lebih kecil lebih efisien ketika mengejarnya dibanding membuang jatah torvedo salvo. Ada hal lain yang harus diperhatikan sebelum meluncurkan *Torpedo salvo shoot* ini yaitu ukuran dari bot harus lebih besar dari 10.

Menurut penulis, strategi ini sudah tepat digunakan karena ketika kita menargetkan musuh yang lebih kecil kemungkinan meleset lebih besar karena kapal kecil akan cenderung menjauhi kapal yang lebih besar. Berbeda dengan kapal yang besar, kapal-kapal itu akan cenderung mendekati kapal kecil sehingga peluang peluru meleset cukup kecil. Dengan begitu penggunaan *torpedo salvo* juga menjadi lebih efisien.

3.2.6 Strategi Mencari/Mengambil *Food* atau *SuperFood*

Untuk menambah ukuran bot tidak hanya harus menabrak kapal musuh tapi bisa dengan mengambil *food* yang ada di arena permain. Penambahan ukuran ketika kita memakan *food* adalah tiga. Selain *food* ada juga super*food*, super*food* tidak secara langsung menambah ukuran tubuh bot. Hanya saja selama 5 detik ketika dia mendapat super *food* maka makanan yang diambil dari awalnya bernilai tiga menjadi bernilai enam.

Pada hakikatnya, jika ditinjau dari segi penambahan ukuran mengumpulkan

makanan yang banyak tidak akan menjadikan bot menjadi pemenang. Hanya saja untuk selingan ketika bot tidak mengejar atau tidak menjauhi maka makanan menjadi suatu hal yang layak dipilih untuk memperbesar ukuran bot dan sampai akhirnya dapat memenangkan pertandingan.

Sama halnya dengan kasus lain strategi yang diterapkan untuk ini adalah dengan *greedy*. Adapun strateginya adalah sebagai berikut. Bot memiliki sebuah list, dimana list itu merupakan sebuah kumpulan makaknan yang telah diurutkan berdasarkan jarak dari yang terdekat hingga yang terjauh antara bot dan makanan. Bot akan memakan makanan yang paling dekat dengan dirinya. Adapun super*food* akan diambil oleh bot ketika memang jaraknya dekat maximalnya dua kali jarak bot ke *food* yang paling terdekat. Strategi ini menggambarkan bahwa bot akan selalu mencari solusi yang terbaik, bot mencari makanan yang memang paling dekat dengan bot tanpa memikirkan kedepannya akan seperti apa.

Menurut penulis strategi ini merupakan strategi yang cukup baik. Karena bot diatur untuk memakan makanan yang paling dekat sehingga bot tidak mengeluarkan suatu usaha yang terlalu besar untuk mendapatkan nilai tambahan ukuran yang cenderung cukup kecil.

3.2.7 Strategi Permasalahan Penggunaan *Wormhole*

Objek lain yang menjadi bagian dari *game* galaxio adalah *wormhole*. *Wormhole* ada secara berpasangan dan memperbolehkan kapal dari *player* untuk memasukinya dan keluar di pasangan satu lagi. Dapat dikatakan *wormhole* merupakan suatu langkah untuk memasuki dimensi lain secara cepat. *Wormhole* menjadi sebuah fitur yang menarik yang dapat dimanfaatkan untuk mendapatkan superniva, untuk mengejar musuh, menghindari musuh dan hal lainnya. Fitur ini cukup bisa diandalkan untuk bot mendapatkan kemenangan. Hanya saja *wormhole* ini hanya dua dalam sebuah area permainan selain itu penggunaan *wormhole* juga tidak selamanya memberikan suatu manfaat yang berarti untuk bot. Untuk itu, dibuatlah strategi *greedy* untuk menentukan menggunakan *wormhole* atau tidak ketika sedang berada di dekat *wormhole*. Strategi yang dibuat adalah strategi ini mengatur bot untuk menggunakan *wormhole* atau tidak

dilihat dari keefisienan waktu dan kegunaannya.

Menurut penulis strategi ini cukup baik mengingat jika kita memasuki *wormhole* secara asal ketika sampai di sisi lain area ditakutkan ada bot besar yang siap menabrak dan membuat bot tereliminasi dari permainan.

3.2.8 Strategi Permasalahan Pengambilan dan Penggunaan *Supernova*.

Senjata yang paling ampuh pada *game* galaxio ini adalah *Supernova*. *Supernova* dapat menghancurkan beberapa bot kapal musuh dalam satu waktu. *Supernova* memberikan sebuah efek ledakan yang dapat membuat semua kapal musuh di area itu hancur. Namun *supernova* ini tidak diketahui posisinya ada dimana, untuk itu bot hanya mengandalkan keberuntungan suatu saat berada dekat dengan *supernova* sehingga bisa mendapatkannya. Walaupun begitu, ada strategi yang bisa dilakukan untuk mendapatkan ini serta ketika telah mendapatkan dan melakukan peledakan *supernova*. Adapun strateginya adalah jika *supernova* didalam radius yang cukup dekat maka ambil *supernova* itu. Jika tidak jangan ambil. Langkah berikutnya misalkan bot mendapatkan *supernova* maka tembakan *supernova* ketika banyak musuh di dalamnya jika tidak akurat jangan ditembakkan.

Menurut penulis strategi ini cukup baik tidak terlalu memaksakan diri untuk mengambil upernova serta berhati-hati dalam menembakannya.

3.2.9 Strategi Penggunaan *Teleport*.

Player dapat memanfaatkan fitur *teleport* untuk memindahkan bot ke suatu titik dan dengan arah yang dituju pada peta. *Teleport*er 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 dari size bot. Setiap 100 tick *player* akan mendapatkan 1 *teleport*er dengan jumlah maximum adalah 10. Sehingga diperlukan keakuratan dalam penggunaanya untuk efisiensi dari fitur *teleport* itu sendiri. *Teleport* akan sangat berguna untuk menghindari musuh ketika sedang dikejar maupun ketika sedang mengejar musuh.

Menurut penulis, fitur *teleport* hanya diaktifkan ketika keadaan yang benar-benar darurat saja, karena penggunaan *teleport* akan mengurangi size dari bot

dengan cukup signifikan. Sehingga, sangat disarankan untuk menggunakan *teleport* hanya ketika terdesak dikejar musuh dan butuh untuk berpindah secepat mungkin, pada suatu titik dan dengan arah yang dituju di peta.

3.2.10 Strategi Penggunaan Shield.

Shield digunakan untuk menghindari serangan torpedo lawan. Shield ini tidak bisa selalu digunakan karena ketika menggunakan shield ini maka ukuran bot berkurang sebesar 20. Adapun shield sangat dibutuhkan ketika ukuran bot dikurangi 20 masih besar dan posisi tembakan torpedo sangat dekat kurang dari 50. Sehingga, mau tidak mau shield harus diaktifkan.

Penulis menganggap penggunaan shield pada waktu-waktu mendesak adalah suatu hal yang cukup baik.

3.3 Analisis Efisiensi dari Kumpulan Solusi Algoritma *Greedy*

Pada permainan Galaxio, banyak *gamestate* yang bisa kita ketahui dengan mudah seperti posisi pemain, posisi lawan, posisi objek-objek dalam permainan, dan lain-lain. Tentu hal tersebut memudahkan program agar berjalan dengan efektif.

Pada strategi menghindari dan mengejar musuh, kita membutuhkan data perbandingan jarak antara bot dengan semua musuh. Untuk mendapatkan nilai perbandingan ini dilakukan sebanyak n kali sehingga kompleksitasnya adalah O(n). Adapun untuk perbandingan ukuran antar bot dengan musuk kompleksitasnya adalah O(1). Sehingga, untuk kompleksitas algoritma ini yaitu O(n) dirasa sudah cukup baik.

Pada strategi bot tidak keluar arena hanya dilakukan perhitungan jarak diantara keduanya. Maksimal jarak diantara keduanya adalah batas arena terdekat kurang dari radius arena dikurang tiga per dua ukuran bot. Jika jaraknya kurang dari itu maka bergerak ke arah pusat area. Kompleksitas waktunya adalah O(1).

Pada strategi menjauhi *gas cloud ,* pertama menentukan jarak antara bot dengan semua *gas cloud* kemudian yang paling dekat dengana bot adalah objek yang akan diperhatikan. Untuk melakukan ini diperlukan kompleksitas algoritma O(n). Kemudian menghitung apakah bot terlalu dekat dengan *gas cloud* atau tidak kompleksitasnya O(1). Sehingga kompleksitas algortima untuk strategi ini adalah O(n). Penulis berpendapat kompleksitas waktu ini cukup baik.

Strategi selanjutnya adalah *Torpedo salvo shoot* (Menembak Musuh), strategi ini sama membutuhkan jarak antara bot dengan musuh kompleksitasnya O(n) dan kompleksitas menembakkan *Torpedo salvo shoot* sebesar O(1). Sehingga kompleksitas algoritma ini adalah O(n).

Strategi berikutnya adalah Mencari dan Mengambil *Food* dan Super*food* kompleksitasnya juga O(n). Terdiri dari menentukan jarak semua objek *food* dan objek super*food* dengan bot ditambah proses memakan *food* dan super*food* itu.

Strategi berikutnya adalah penggunaan *wormhole.* Kompleksitasnya O(n) yaitu terdiri dari proses menentukan jarak semua *gas cloud* dengn bot ditambah proses menjauhinya.

Strategi pengambilan dan penggunaan *supernova*. Kompleksitas algoritma untuk menyelesaikan mengambil *supernova* adalah O(1) hal ini karena *supernova* hanya ada satu sehingga tidak ada penentuan perbandingan jarak anta *supernova* dengan bot yang lain. Adapun pada proses penembakannya O(n) karena bot mempertimbangkan jara musuh sebelum menembakkan *supernova*.

Strategi yang terakhir adalah *teleport*. Untuk *teleport* ini kompleksitasnya hanya O(1). Karena hanya menembakkan *teleport* ke arah mana tidak diperlukan sstrategi perbandingan dan sebagainya.

3.4 Analisis Efektivitas dari Kumpulan Solusi Algoritma *Greedy*

3.4.1 Efektivitas Strategi Menghindari dan Mengejar Musuh

Strategi *greedy* mengejar musuh cukup efektif dalam memperbanyak *score* dan memperbesar ukuran bot, akan tetapi strategi ini harus diimplementasi dengan hati-hati karena terdapat beberapa syarat yang harus dipenuhi jika bot menggunakan strategi ini dalam memberikan kemenangan, beberapa syaratnya diantaranya, ukuran bot yang kita miliki harus lebih besar daripada bot musuh, hal ini lantaran jika kita mengejar musuh yang ukurannya lebih besar dari bot kita malah akan menghancurkan bot itu sendiri. Syarat kedua yaitu jarak antara bot kita dengan bot musuh masih dalam radius jangkauan yang realistis, karena jika kita mengejar bot musuh dengan memenuhi syarat pertama yaitu ukuran bot kita lebih besar dari musuh maka kecepatan dari bot kita akan lebih lambat daripada bot musuh, hal ini

merupakan konsekuensi daripada pemenuhan syarat pertama yang disebutkan sebelumnya. Syarat lainnya jika kita mengejar bot lawan yaitu memastikan tidak adanya *gas cloud* dan asteroid yang malah akan menjadi penghalang dan membuat ukuran bot kita mengecil.

Strategi selanjutnya yaitu menghindar dari kejaran musuh, strategi ini dibutuhkan ketika terdapat musuh yang memiliki ukuran lebih besar daripada bot yang kita miliki, dan bot musuh tersebut mengejar kita. Maka menghindar dari kejarannya merupakan pilihan terbaik. Selain karena ukuran kita lebih kecil yang mana akan hancur jika ditabrak olehnya, bot kita juga memiliki kecepatan yang lebih cepat daripadanya, sehingga kita dapat memanfaatkan kelebihan tersebut untuk bertahan hingga akhir.

3.4.2 Efektivitas Strategi Bot Tidak Keluar Arena

Strategi agar bot tidak keluar arena cukup efektif untuk mempertahankan bot hingga akhir, karena arena galaxio akan semakin mengecil sesuai dengan ukuran bot yang semakin membesar. Strategi ini sangat dibutuhkan untuk menjaga bot tetap dalam arena galaxio dan menghindari bot tidak keluar arena. Dapat dikatakan bahwa strategi ini merpakan strategi dasar untuk bermain karena jika strategi ini tidak diimplementasikan maka akan membuat bot berjalan tanpa batas hingga keluar arena galaxio yang mana hal tersebut berarti bot kita kalah.

Implementasi dari strategi ini dapat di sesuaikan dengan preferensi masing-masing kelompok, yang mana kelompok kami menerapkan agar bot segera menjauhi arena dan berbalik arah menuju pusat arena jika dirasa sudah terlalu dekat yaitu jika jarak antara bot dengan batas arena terdekat kurang dari radius arena dikurang tiga per dua ukuran bot.

3.4.3 Efektivitas Strategi Menjauhi *Gas cloud*

Gas cloud akan semakin mengurangi ukuran bot sebanyak 1 setiap 1 tick jika terkena, maka dari itu strategi ini diperlukan untuk diimplementasikan dengan efektif agar bot menghindar dan tidak terkena *gas cloud* karena akan sangat merugikan bot dan membuat bot semakin mengecil sehingga mudah untuk dikalahkan. Mirip seperti implementasi pada strategi tidak keluar arena, pada

strategi ini juga bot harus menjauhi *gas cloud* dalam radius sejauh ukuran bot. Jika bot sudah menjauhi *gas cloud* maka efek yang diberikan juga akan menghilang.

3.4.4 Efektivitas Strategi *Torpedo salvo shoot* (Menembak Musuh)

Efektivitas strategi *Torpedo salvo shoot* cukup baik dengan diterapkannya strategi ini. Bot menembakkan torpedo ke arah lawan yang memiliki ukuran lebih besar sehingga tingkat ketidaktepatan tembakan itu cukup kecil. Strategi ini cukup efektif untuk diterapkan karena dapat mendapatkan pertambahan ukuran yang cepat disbanding memakan *food* yang jumlah tambahan size nya jauh lebih sedikit.

3.4.5 Efektivitas Strategi *Afterburner*

Afterburner diterapkan untuk mengejar musuh yang lebih kecil. Hal yang telah diketahui antara ukuran dan kecepatan berbanding terbalik. Semakin besar ukuran bot maka kecepatannya semakin lambat. Pada dasarnya ketika tidak menggunakan *afterburner* bot yang berukuran lebih besar tidak akan pernah berhasil mengejar bot yang ukurannya lebih kecil. Untuk itu efek *afterburner* ini dibutuhkan. Efek ini dapat membuat bot memiliki kecepatan dua kali lebih cepat. Sehingga hal yang awalnya tidak memungkinkan bot yang berukuran lebih besar mengejar bot yang berukuran lebih kecil menjadi suatu hal yang mungkin. Strategi in cukup efektif diterapkan untuk mendapatkan kemenangan bot. Karena sebenarnya bot jauh lebih cepat besar dengan menabrak musuh disbanding dengan memakan *food* atau super*food*.

3.4.6 Efektivitas Strategi Mencari dan Mengambil *Food* dan Super*food*

Hal lain yang bisa dipertimbangkan ketika ingin memperbesar ukuran bot adalah dengan mengonsumsi *food* atau super*food*. *Food* tersebar cukup banyak di area permainan. Untuk itu mendapatkan *food* adalah suatu hal yang mudah. Namun, pertambahan ukuran yang diakibatkan mengonsumsi *food* tidak terlalu signifikan. Sehingga ini menjadi hal yang bisa dikatakan sebuah pendamping tidak menjadi bagian paling utama. Namun, ketika bot mengonsumsi super *food* baru mengonsumsi *food* ukuran bot akan bertambah jauh lebih banyak dua kali dari normalnya. Untuk itu strategi yang diterapkan adalah mencari super*food* terlebih

dahulu jika dekat baru *food*. Efektivitas dari strategi ini ketika diterapkan ke permainan cukup baik. Karena penulis telah melakukan eksplorasi kesimpulan yang didapat penulis adalah strategi ini yang dirasa paling baik untuk diterapkan di bot.

3.4.7 Efektivitas Strategi pengambilan dan penggunaan *supernova*

Supernova hanya ada 1 dalam setiap pertandingan, sehingga ini dapat menjadi salah satu senjata yang ampuh untuk memenangkan permainan. Strategi pengambilan *supernova* diperlukan sebagai prioritas jika terdapat dalam radius jangkauan yang cukup dekat dengan bot. Setelah mendapatkan *supernova* maka selanjutnya *supernova* tersebut akan diledakkan, yang dalam hal ini, kelompok kami meledakkannya ke arah pusat arena galaxio, karena biasanya musuh akan berkumpul di bagian tengah arena dan ledakkan *supernova* akan mengeluarkan *gas cloud* yang cukup banyak, sehingga diharapkan ledakannya dapat mengenai sekaligus banyak musuh untuk mengurangi ukuran bot mereka, dan bot kita dapat lebih mudah mengalahkannya.

3.4.8 Efektivitas Strategi *Wormhole*

Wormhole digunakan ketika memang dekat dengan bot serta ketika digunakan user tidak mendapatkan suatu masalah ketika berpindah ke sisi lain area permainan. Ketika strategi ini diterapkan ini cukup efektiv untuk membuat bot dapat memenangkan pertandingan. Karena bot hanya akan menggunakan *wormhole* ketika dekat dan tidak menimbulkan kerugian yang berarti kedepannya.

3.4.9 Efektivitas Strategi *Teleport*

Strategi *teleport* dapat digunakan untuk memindahkan bot ke suatu titik ketitik lainnya dengan arah yang diinginkan oleh bot. Sehingga penerapan dari strategi ini diharapkan dapat membantu ketika sedang dikejar oleh musuh yang memiliki ukuran lebih besar maupun ketika mengejar musuh yang memiliki kecepatan yang lebih cepat dari bot kita.

Strategi *teleport* ini akan sangat berguna ketika bot sedang terdesak, yaitu ketika dikejar oleh musuh yang hendak menabrak kapal bot yang kita miliki, *teleport* dapat membuat bot kita berpindah ke titik dan dengan arah yang kita

inginkan, *teleport*er tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun.. Namun dalam penggunaan perlu memerhatikan beberapa hal karena *teleport* akan mengurangi ukuran bot dengan cukup signifikan yaitu sebesar 20. Setiap 100 tick *player* akan mendapatkan 1 *teleport*er dengan jumlah maximum adalah 10.

3.4.10 Efektivitas Strategi Shield

Shield digunakan ketika bot ingin bertahan dari tembakan musuh. Shield mengurangi ukuran bot sekitar 20 size sehingga penggunaan shield ini tidak bisa dilakukan secara sering. Shield akan digunakan hanya ketika memang diperlukan yaitu ketika jarak tembakan torpedo shoot kurang dari 50 serta ketika ukuran bot-20 lebih besar dibandingkan ukuran bot yang menembakan torpedo shoot. Efektivitas dari diterapkannya strategi ini ke kemenangan bot pada suatu permainan cukup baik. Dengan adanya penggunaan shield ini bot tidak hanya pasrah saat ditembakkan torpedo shoot, tetapi akan melawan bahkan dapat mendapatkan keuntungan dengan terus berjalan dan menabrak penembaj torpedo shoot.

3.5 Analisis dari Desain Solusi Algoritma *Greedy* yang Diusulkan untuk di I mplementasikan Dalam Program

Strategi yang penulis ambil sebagai algoritma *greedy* utama dari program bot permainan galaxio ini adalah penggabungan dari seluruh strategi yang telah dipaparkan pada subbab 3.2. Penulis mengambil seluruh strategi untuk setiap *mechanism* pada permainan galaxio agar program bot dapat menangangi seluruh kemungkinan kasus dari *game state* secara efektif dan tepat sasaran. Agar seluruh strategi dapat digabungkan menjadi suatu algoritma *greedy*, penulis harus mendefinisikan urutan dari strategi mana yang harus dieksekusikan terlebih dahulu. Salah satu cara mendefinisikan urutan tersebut adalah dilihat dari seberapa besar prioritas suatu strategi jika dibandingkan dengan strategi lainnya. Urutan prioritas tersebut memiliki sifat dimana pemrogram harus menentukan bagaimana urutannya agar menghasilkan algoritma *greedy* yang paling optimum. Pemrogram dapat menggunakan logika atau mengecek urutan strategi untuk memformulasikan urutan mana saja yang dapat diimplementasikan sebagai algoritma *greedy*.

Setelah dipikirkan dan kemudian mencobanya pada *game engine*, penulis telah berhasil memformulasikan urutan pengeksekusian strategi yang menghasilkan algoritma *greedy* paling optimum. Penulis berusaha mencari permutasi urutan pengeksekusian strategi yang memiliki peluang menang paling besar bila ditandingkan dengan bot lawan. Algoritma bot lawan yang digunakan sebagai tolak ukur adalah *reference* bot bawaan. Urutan prioritas dari pengeksekusian strategi heuristik yang telah diformulasikan oleh penulis adalah sebagai berikut:

- Strategi tidak keluar arena.
- Strategi menjauhi *gas cloud*.
- Strategi mengejar musuh dan menghindari musuh
- Strategi *Torpedo salvo shoot* (menembak musuh)
- Strategi *Afterburner*
- Strategi penggunaan *wormhole*
- Strategi Penggunaan *Teleport*
- Strategi Penggunaan Shield
- Strategi mencari dan mengambil *food* dan super*food*
- Strategi penggunaan *supernova*

Strategi *greedy* yang dibuat penulis dirancang untuk bisa mengatasi semua kemungkinan yang ada sehingga membuat bot dapat memenangkan pertandingan. Dalam strategi *greedy* ini bot melakukan apa yang dianggap terbaik pada saat itu tanpa mempertimbangkan jalan selanjutnya yang diambil.

Bab 4: Implementasi dan Pengujian

4.1. Implementasi Algoritma *Greedy* pada Bot Permainan Galaxio

Implementasi algoritma *greedy* pada program terdapat pada file BotService.java

A. Pseudocode

Implementasi algoritma *Greedy* pada program terdapat pada file BotService.java dalam method computeNextPlayerAction seperti pseudocode dan penjelasannya berikut.

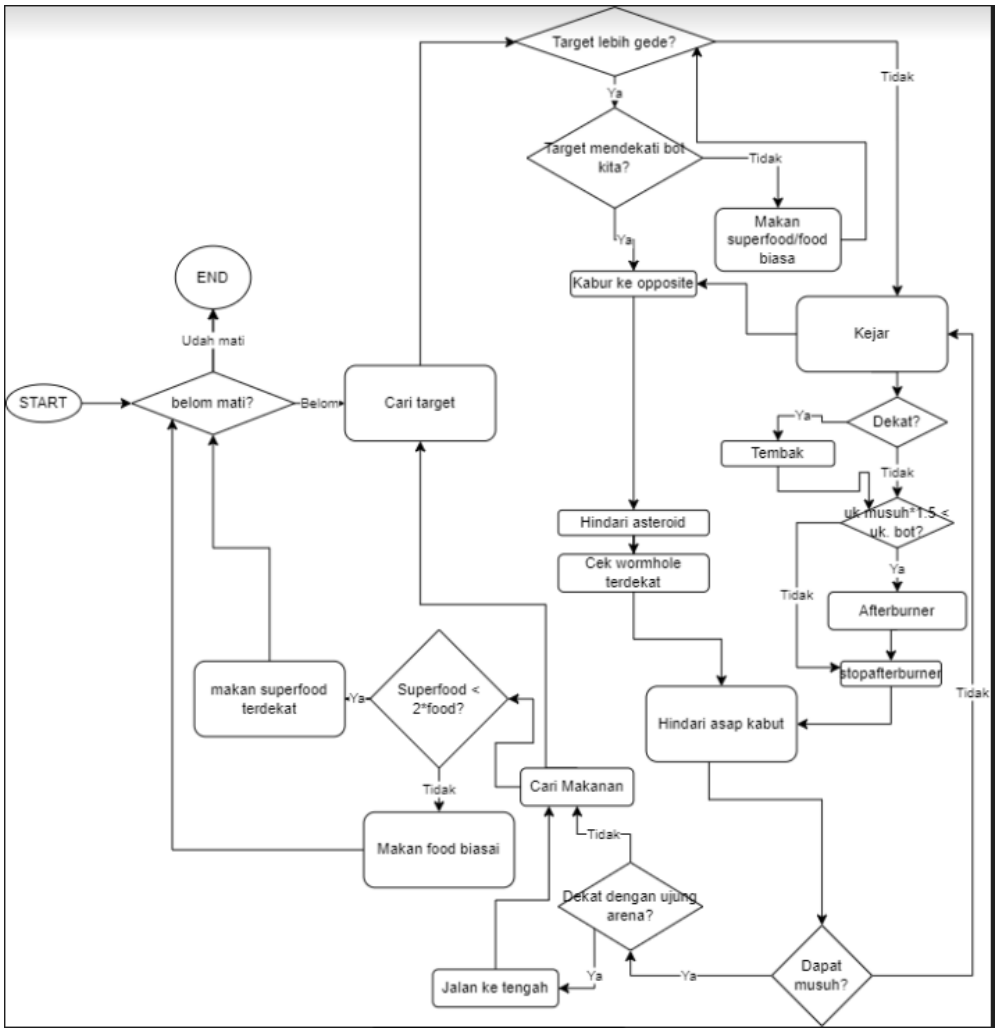
```
Procedure computeNextPlayerAction(input/output: PlayerAction playerAction)
{I.S: prosedur menerima input playerAction dari gameState yang terbaru
 F.S: playerAction terupdate dan menerima command untuk menggerakkan bot}

KAMUS LOKAL
flag : int
bot, food, superfood, gascloud, asteroid, enemy, wormhole : gameObjectType
playerAction : PlayerAction
gameState : GameState
isAfterBurnerOn : boolean

ALGORITMA
if (is not gameState.getGameObjects().isEmpty()) then {bot masih hidup}
   {prosedur akan dipanggil setiap tick dengan bantuan flag}
   if (gameState.world.getCurrentTick() = flag) then

      {jika jarak bot terlalu dekat dengan pinggir arena, menjauh}
      if (getDistanceWorldCenter(bot) + 2*bot.size > gameState.world.getRadius()) then
      playerAction.setHeading(getHeadingWorldCenter())
      playerAction.setAction(PlayerActions.FORWARD)
      else
         {Inisialisasi objek-objek yang dibutuhkan}
         food <- nearestFood(gameState)
         superfood <- nearestSuperFood(gameState)
         gascloud <- nearestGascloud(gameState)
         asteroid <- nearestAsteroid(gameState)
         enemy <- nearestEnemy(gameState)
```

```
wormhole <- nearestWormhole(gameState)

{jika dekat dengan gascloud, menjauh}
if (gascloud is not null and (getDistanceBetween(bot, gascloud) - (bot.size)
- (gascloud.size)) < (gascloud.size*1.5)) then
    playerAction.setHeading((getHeadingBetween(gascloud) + 180) % 360)
    playerAction.setAction(PlayerActions.FORWARD)

else
    {jika dekat dengan asteroid, menjauh}
    if (asteroid is not null and (getDistanceBetween(bot, asteroid) -
        (bot.size) - (asteroid.size)) < (asteroid.size*1.5)) then
       playerAction.setHeading((getHeadingBetween(asteroid) + 180) % 360)
       playerAction.setAction(PlayerActions.FORWARD)
    else
        if ((bot.size > enemy.size)) then
           if (((getDistanceBetween(bot, enemy) - (bot.size) - (enemy.size))
                 <= 0.3*gameState.world.getRadius())) then
               if (bot.size > 70)then
                    {mengejar musuh menggunakan afterburner}
                    if (not isAfterBurnerOn) then
                         playerAction.setHeading(getHeadingBetween(enemy))
                         playerAction.setAction(PlayerActions.STARTAFTERBURNER)
                         this.afterburnerOn <- true
                    else
                         playerAction.setHeading(getHeadingBetween(enemy))
                         playerAction.setAction(PlayerActions.FORWARD)
               else
                  if (not isAfterBurnerOn) then
                        playerAction.setHeading(getHeadingBetween(enemy))
                        playerAction.setAction(PlayerActions.FORWARD)
                  else {menghentikan afterburner jika telah selesai digunakan}
                        playerAction.setHeading(getHeadingBetween(enemy))
                        playerAction.setAction(PlayerActions.STOPAFTERBURNER)
                        this.afterburnerOn <- false
           else
               if (isAfterBurnerOn) then
                    {menghentikan afterburner jika telah selesai digunakan}
                    playerAction.setAction(PlayerActions.STOPAFTERBURNER)
```

```
                        this.afterburnerOn <- false
                    else
                        playerAction.setHeading(getHeadingBetween(enemy))
                        playerAction.setAction(PlayerActions.FORWARD)

            else
                if (isAfterBurnerOn) then
                    playerAction.setAction(PlayerActions.STOPAFTERBURNER)
                    this.afterburnerOn <- false
                else
                    if ((getDistanceBetween(bot, enemy) - (bot.size) -
                        (enemy.size)) <= 0.3*gameState.world.getRadius()) then
                        if (bot.size > 150) then
                        {Menggunakan shield untuk berlindung dari musuh}
                            if (bot.shieldCount == 1){
                                playerAction.setHeading((getHeadingBetween(enemy) + 180) % 360)
                                playerAction.setAction(PlayerActions.ACTIVATESHIELD)
                            else
                                playerAction.setHeading((getHeadingBetween(enemy) + 180) % 360)
                                playerAction.setAction(PlayerActions.FORWARD)
                        else
                            if (getDistanceBetween(bot, wormhole) <
                                0.15*gameState.world.getRadius())then
                                if (wormhole is not null) then
                                    if (wormhole.getSize() > bot.size) then
                                        playerAction.setHeading((getHeadingBetween(wormhole)))
                                        playerAction.setAction(PlayerActions.FORWARD)
                                    else
                                        playerAction.setHeading((getHeadingBetween(enemy)+180)%360)
                                        playerAction.setAction(PlayerActions.FORWARD)
                                else
                                    playerAction.setHeading((getHeadingBetween(enemy)+180)%360)
                                    playerAction.setAction(PlayerActions.FORWARD)
                            else
                                if((bot.size+(bot.getTorpedoSalvoCount()*5)) < enemy.size){
                                    playerAction.setHeading((getHeadingBetween(enemy)+180)%360)
                                    playerAction.setAction(PlayerActions.FORWARD)
                                else {menembakkan torpedo ke arah musuh}
                                    playerAction.setHeading(getHeadingBetween(enemy))
                                    playerAction.setAction(PlayerActions.FIRETORPEDOES)
                    else
                    {membandingkan food dan superfood untuk memilih prioritas}
                        if ((getDistanceBetween(bot, food)-(bot.size)-
                            (food.size))>=(1.5)*(getDistanceBetween(bot, superfood)-
                            (bot.size)- (superfood.size))) then
                            {mencari superfood}
                            playerAction.setHeading(getHeadingBetween(superfood))
                            playerAction.setAction(PlayerActions.FORWARD)
                        else {mencari food biasa}
                            playerAction.setHeading(getHeadingBetween(food))
                            playerAction.setAction(PlayerActions.FORWARD)
    else
       {waiting tick selanjutnya}
```

Algoritma kode legkap dapat dilihat pada lampiran pranala github di bawah.

B. FlowChart


Target lebih gede?
Tidak
Ya
Target mendekati bot kita?
Tidak
Ya
Makan superfood/food biasa
END
Kabur ke opposite
Kejar
Udah mati
START
belom mati?
Belom
Cari target
Dekat?
Ya
Tembak
Tidak
uk. musuh*1.5 < uk. bot?
Hindari asteroid
Cek wormhole terdekat
Ya
Tidak
Afterburner
stopafterburner
Tidak
makan superfood terdekat
Ya
Superfood < 2*food?
Hindari asap kabut
Tidak
Cari Makanan
Tidak
Makan food biasai
Dekat dengan ujung arena?
Ya
Ya
Jalan ke tengah
Dapat musuh?

4.2. Penjelasan Struktur Data pada Bot Permainan Galaxio

Dalam bot galaxio ini ada tiga elemen penting yang dipisahkan dalam 3 folder

A. Folder Enum

Di dalam folder enum ada dua file. File-file itu mendeklarasikan objek-objek yang ada dalam permainan serta action yang dapat dilakukan oleh bot. Dua kelas itu adalah.

- ObjectTypes.java

Di dalam file ini ada sebuah class enum ObjectTypes yang didalamnya terdapat objek-objek yang ada dalam permainan, dalam kelas enum juga terdapat 2 methode.

- Objek yang ada adalah *Player*(1), *Food*(2), *Wormhole*(3), GasCloud(4), AsteroidFields(5), TorpedoSalvo(6), Super*food*(7), *Supernova*Pickup(8), *Supernova*Bomb(9), *Teleport*er(10), Shield(11).
- Method yang terdapat dalam file ini yaitu

Methods	Description
ObjectTypes(Integer value)	Ctor untuk membentuk suatu objek yang ada dalam *game*
public static ObjectTypes valueOf(Integer value)	Mengembalikan objek yang dipanggil dalam parameter

- PlayerActions.java

Di dalam file ini ada sebuah class enum *Player*Actions di dalamnya ada beberapa objek dan methode.

- Objek yang ada dalam file ini berisi tentang aksi-aksi yang bisa dilakukan oleh bot. Ada 10 aksi yaitu, Forward(1), Stop(2), Start*Afterburner*(3), Stop*Afterburner*(4), FireTorpedoes(5), Fire*Supernova*(6), Detonate*Supernova*(7), Fire*Teleport*(8), *Teleport*(9), dan ActivateShield(10)

- Terdapat satu methode yang ada dalam file ini

Methods	Description
private PlayerActions(Integer value)	Ctor untuk menyimpan aksi yang akan dilakukan oleh bot.

B. Folder Models

Folder models memiliki peran penting untuk menjalankan bot. Ada 5 file didalamnya.

- *Game*Object.Java

  Di dalam file ini ada class dengan nama *Game*Object. Di dalamnya ada beberapa methode

Methods	Description
public *Game*Object	Ctor untuk membuat sebuah tipe data *Game*Object dengan semua atribut *Game*Object diinisialisasi dengan input yang diberikan pengguna di parameter fungsi ctor ini.
public UUID getId()	Mengembalikan atribut id dari *Game*Object
public void setId(UUID id)	Men-set atribut id *game* objek dengan input parameter dari user yaitu id
public int getSize()	Mengembalikan ukuran dari *game* objek
public void setSize(int size)	Men-set atribut size pada atribut *Game*Object dengan suatu input pengguna.
public int getTorpedoSalvoCount()	Mengembalikan jumlah torpedoSalvoCount

public void setTorpedoSalvoCount(int torpedoSalvoCount)	Men-set atribut torpedoSalvoCount dengan suatu nilai inpu dari pengguna
public int get*Supernova*Available()	Mengembalikan suatu keadaan dimana ketika terdapat *supernova*Avaible ketika iya maka return 1 jika tidak 0
public void set*Supernova*Available(int *supernova*Available)	Menset ada dan tidaknya atribut *game* objek yakni atribut *supernova*Avaible
public int get*Teleport*erCount()	Mengeluarkan jumlah dari salah satu atribut *Game*Object yaitu *Teleport*erCount
public void set*Teleport*erCount(int *teleport*erCount)	Menset suatu nilai dari input pengguna terhadap salah satu atribut *Game*Object *Teleport*erCount
public int getEffect()	Men-return jumlah efek yang sedang terjadi pada bot
public void setEffect(int effect)	Men-set salah satu atribut *Game*Object dengan effect yang merupakan input dari luar.
public int getShieldCount()	Mengembalikan salah satu jumlah atribut *Game*Object yaitu ShieldCount()
public void setShieldCount(int shieldCount)	Men-set atribut ShieldCount pada *game* objek dengan input pengguna
public int getSpeed()public void setSpeed(int speed)	Mengembalikan kecepatan dari *Game*Object
public Position getPosition()	Mengembalikan posisi dari *Game*Object
public void setPosition(Position position)	Men-set atribut dari posisi *Game*Object
public ObjectTypes get*Game*ObjectType()	Mengembalikan salah satu atribut *Game*Object yaitu *game*ObjectType

public void set*Game*ObjectType (ObjectTypes *game*ObjectType)	Men-set salah satu atribut *Game*Object yaitu *game*ObjectType dengan input dari user.
public static *Game*Object FromStateList(UUID id, List<Integer> stateList)	Mengembalikan status *Game*State yang terjadi pada bot sekarang

- *Game*State.Java

  Di dalam file ini ada class *Game*State yang memiliki beberapa method serta data/objek

  - Data/Objek itu ada World dengan tipe World, *game*Objects dalam suatu list bertipe *GameObject* dan *playerGame*Objects dalam suatu list bertipe *Game*Object.
  - Adapun method yang ada dalam kelas ini adalah

Methods	**Description**
public *Game*State()	Ctor untuk membentuk *Game*State dengan pendefinsian alokasi baru dari sebuah array yang menjadi atribut-atribut *Game*State.
public *Game*State	Ctor dengan parameter input dari user.
public World getWorld()	Mengembalikan output salah satu atribut *Game*State yaitu World
public void setWorld(World world)	Men-set atribut World pada *Game*State dengan world
public List<*Game*Object> get*Game*Objects()	Mengembalikan *Game*Object dari sebuah list bertipe *Game*Object
public void set*Game*Objects(List<*Game*Obj	Mens-set nilai elemen list *Game*Object dengan sebuah nilai *Game*Object

ect> *game*Objects)	
public List<*Game*Object> get*PlayerGame*Objects()	Mereturn elemen dari atribut *Game*State yaitu list *playerGame*Object
public void set*PlayerGame*Objects(List<*Game*Object> *playerGame*Objects)	Men-set nilai pada elemen list *playerGame*Object

- *Game*StateDto.java

  Ada 3 objek/data yaitu World world, Map<String, List<Integer>> *game*Objects, Map<String, List<Integer>> *player*Objects. Serta ada beberapa methode di dalamnya yaitu

Methods	**Description**
public Models.World getWorld()	Menghasilkan salah satu atribut *Game*StateDto yaitu World
public void setWorld(Models.World world)	Meng-set nilai world terhadap atribut World di *Game*StateDto
public Map<String, List<Integer>> get*Game*Objects()	Me-return *Game*Object dari sebuah data bertipe Map
public void set*Game*Objects(Map<String, List<Integer>> *game*Objects)	Men-set elemen Map dengan data bertipe *Game*Object
public Map<String, List<Integer>> get*Player*Objects()	Me-return *Game*Object yang berupa *player*
public void set*Player*Objects(Map<String, List<Integer>> *player*Objects)	Mereturn list *player*Objects yang merupakan bagian dari atribut *Game*StateDto
public void set*Player*Objects(Map<String,	Men-set elemen list *player*Objects yang merupakan bagian dari atribut

List<Integer>> *player*Objects)	*Game*StateDo dengan suatu nilai, inoutan dari luar.

- *Player*Action.java

  Di dalam file ini ada sebuah kelas *Player*Action dimana di dalamnya ada 3 objek UUID *player*Id, *Player*Actions action, int heading. Ada beberapa methode juga yakni

Methods	**Description**
public UUID get*Player*Id()	Menghasilkan output atribut *player*Id dari class *Player*Action
public void set*Player*Id(UUID *player*Id)	Men-set nilai *player*Id untuk atribut *player*Id yang merupakan bagian dari *Player*Action
public *Player*Actions getAction()	Menghasilkan output atribut action dari class *Player*Action
public void setAction(*Player*Actions action)	Men-set nilai action untuk atribut action yang merupakan bagian dari *Player*Action
public void setHeading(int heading)	Men-set nilai heading untuk atribut heading yang merupakan bagian dari *Player*Action
public int getHeading()	Menghasilkan output atribut heading dari class *Player*Action

- Position.java

  Di dalam file ada sebuah kelas bernama Position yang memilik atribut x yang bertipe integer dan y yang bertipe integer serta beberapa methode yaitu

Methods	Description
public Position()	Ctor yang membentuk position dengan nilai default x = 0 dan y = 0
public Position(int x, int y)	Ctor yang membentuk position dengan input dari parameter nilai x = x dan y = y
public int getX()	Menghasilkan output posisi pada atribut yaitu position x
public void setX(int x)	Men-set posisi pada atribut position x dengan nilai x input dari parameter.
public int getY()	Menghasilkan output posisi pada atribut yaitu position y
public void setY(int y)	Men-set posisi pada atribut position y dengan nilai y input dari parameter.

- World.java

Pada file ini ada sebuah kelas World yang mana memiliki tiga atribut data centerPoint bertipe position, radius bertipe integer, dan currentTick bertipe integer. Di dalam kelas ini juga ada beberapa method

Methods	Description
public Position getCenterPoint()	Me-return nilai dari atribut World yaitu centerPoint.
public void setCenterPoint(Position centerPoint)	Men-set nilai dari atribut World yaitu centerPoint menjadi centerPoint yang merupakan input dari luar.
public Integer getRadius()	Me-return nilai dari atribut World yaitu radius.
public void setRadius(Integer radius)	Men-set nilai dari atribut World yaitu radius menjadi radius yang merupakan

	input dari luar.
public Integer getCurrentTick()	Me-return nilai dari atribut World yaitu currentTick.
public void setCurrentTick(Integer currentTick)	Men-set nilai dari atribut World yaitu currentTick menjadi currenTick yang merupakan input dari luar.

C. Folder Service

Ada satu file di dalam folder ini dengan nama BotService.java tapi merupakan bagian terpenting karena algoritma utama ada disana. Ada beberapa method yang ada dalam file ini

Methods	Description
public BotService()	Menginisiasi aksi yang akan dilakukan oleh bot dan *game* state
public *Game*Object getBot()	Me-return bot yang akan digunakan dalam permainan
public void setBot(*Game*Object bot)	Men-set nilai atribut dari bot service yaitu bot dengan sebuah nilai bot yang merupakan input dari parameter
public *Player*Action get*Player*Action()	Me-return aksi-aksi yang akan dilakukan oleh pemain
public void set*Player*Action(*Player*Action *player*Action)	Men-set aksi-aksi yang akan dilakukan pemain terhadap bot
public *Game*State get*Game*State()	Me-return *game* state permainan
public void set*Game*State(*Game*State *game*State)	Me-set *game* state permainan dan memanggil procedure updateSelfState()
private void updateSelfState()	Men-set jika terdapat perubahan pada bot, seperti perubahan ukuran, kecepatan dan lain sebagainya.

private double getDistanceBetween (*Game*Object object1, *Game*Object object2)	Me-return jarak antara object1 dengan object2
public void computeNext*Player*Action(*Player*Action *player*Action)	Prosedure ini berguna untuk men-set setiap perubahan aksi yang dilakukan oleh bot
public Boolean is*Afterburner*On()	Me-return status *Afterburner* jika diaktifkan
private double getDistanceWorldCenter(*Game*Object object1)	Me-return jarak dari object1 ke center dari area permainan
private int getHeadingBetween(*Game*Object otherObject)	Me-return derajat antara bot dengan otherObject
private int getHeadingWorldCenter()	Me-return derajat antara bot dengan center area permainan
private int toDegrees(double v)	Menconvert radian ke derajat
private *Game*Object nearest*Food*(*Game*State *game*State)	Me-return object *food* yang posisinya paling dekat dengan bot
private *Game*Object nearestSuper*Food*(*Game*State *game*State)	Me-return object super*food* yang posisinya paling dekat dengan bot
private *Game*Object nearestEnemy(*Game*State *game*State)	Me-return object enemy yang posisinya paling dekat dengan bot
private *Game*Object nearestGascloud(*Game*State *game*State)	Me-return objek *gas cloud* yang posisinya terdekat dengan bot

D. Main.java

Pada file ini semua fungsi dan prosedur yang telah diimplementasikan pada folder enums, models, service dipangggil untuk dijalankan yang nantinya hasil pemanggilan pada file ini akan terlihat pada terminal dotnet yang sedang berjalan.

Methods	**Description**

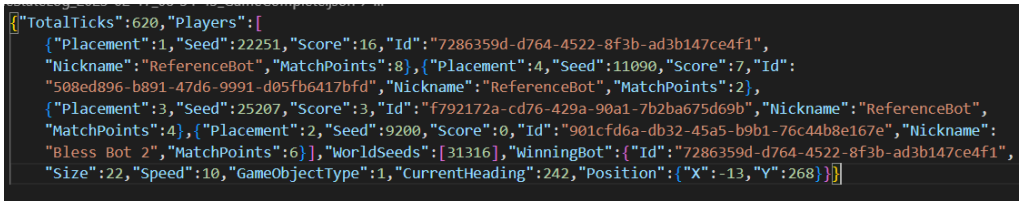
public static void main(String[] args) throws Exception	Pada intinya procedure ini akan memanggil dan mengkoneksikan ke local host untuk merunner dan memainkan *game* galaxio

4.3. Pengujian Bot serta Analisis Performansi Bot Permainan Galaxio

Pengujian bot dilakukan dengan melawankan bot dengan reference bot yang ada pada repository Entelect Challenge. Reference bot menggunakan bahasa c#. Dari beberapa kali pengujian didapatkan perolehan kemenangan sebagai berikut.

- Match 1

  Pada match ini dipertandingkan 3 referenceBot melawan 1 bot buatan penulis. Pada pertandingan ini dimenangkan oleh referenceBot. Pada match ini penulis berada di urutan ke-2.



- Match 2

  Pada match ini dipertandingkan 4 bot buatan penulis dengan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi keempat.

- Match 3

  Pada match ini 3 reference bot Kembali dipertandingkan dengan 1 bot pemain. Hasilnya adalah ReferenceBot Kembali menang menjadi juara 1. Dan posisi bot buatan yang dibuat berada pada posisi 2.

```
{"TotalTicks":215,
"Players":[{"Placement":1,"Seed":21866,"Score":60,
"Id":"b1eef4d8-9b3e-471a-891d-b59da5af5e09","Nickname":"ReferenceBot","MatchPoints":8},
{"Placement":3,"Seed":25484,"Score":18,"Id":"c684c768-618b-47ac-8087-267d6941480b",
"Nickname":"ReferenceBot","MatchPoints":4},
{"Placement":4,"Seed":14387,"Score":9,"Id":"09e8cd3e-b134-4c5f-b94d-168b14a13b81",
"Nickname":"ReferenceBot","MatchPoints":2},{"Placement":2,"Seed":23987,"Score":2,
"Id":"da230011-78f4-46b9-b7d1-8e4fc4efe421","Nickname":"Bless Bot 2","MatchPoints":6}],
"WorldSeeds":[35627],"WinningBot":{"Id":"b1eef4d8-9b3e-471a-891d-b59da5af5e09","Size":244,
"Speed":1,"GameObjectType":1,"CurrentHeading":276,"Position":{"X":125,"Y":-24}}}
```


- Match 4

  Pada match ini 3 reference bot Kembali dipertandingkan dengan 1 bot pemain. Hasilnya adalah ReferenceBot Kembali menang menjadi juara 1. Dan posisi bot buatan yang dibuat berada pada posisi 2.

logger-publish > {} GameStateLog_2023-02-17_08-37-50_GameComplete.json > ...

```
{"TotalTicks":363,"Players":[{"Placement":1,"Seed":517,"Score":62,
"Id":"3ff4337f-ec20-4d03-bbf2-e8fa52b8346b","Nickname":"ReferenceBot","MatchPoints":8},
{"Placement":3,"Seed":2901,"Score":20,"Id":"8ff92dc3-40a6-46b6-8d03-fd715ea63afc",
"Nickname":"ReferenceBot","MatchPoints":4},{"Placement":2,"Seed":13002,"Score":15,
"Id":"d70d79f7-3a91-43ac-a67f-184a085a6040","Nickname":"Bless Bot 2","MatchPoints":6},
{"Placement":4,"Seed":954,"Score":11,"Id":"f38e5000-148e-4090-bf8e-679bda193796",
"Nickname":"ReferenceBot","MatchPoints":2}],"WorldSeeds":[42336],
"WinningBot":{"Id":"3ff4337f-ec20-4d03-bbf2-e8fa52b8346b","Size":257,"Speed":1,
"GameObjectType":1,"CurrentHeading":297,"Position":{"X":-88,"Y":204}}}
```


- Match 5

  Pada match ini dipertandingkan 4 bot buatan penulis dengan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.

logger-publish > {} GameStateLog_2023-02-17_04-40-45_GameComplete.json > ...

```
{"TotalTicks":426,"Players":[{"Placement":1,"Seed":9394,"Score":81,
"Id":"26b3f10a-36a4-4385-98e2-f3eada093cf6","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":3,"Seed":12354,"Score":26,
"Id":"0b0f37dd-8a6f-42f5-9c0c-514843af51ae","Nickname":"Bless Bot","MatchPoints":4},
{"Placement":4,"Seed":25030,"Score":0,"Id":"c3fdbab1-8487-42c0-96b7-5e23ce49b499",
"Nickname":"Bless Bot","MatchPoints":2},{"Placement":2,"Seed":23914,"Score":0,
"Id":"1d2c163e-4a67-46b1-adc6-a15b2a6dc601","Nickname":"Bless Bot","MatchPoints":6}],
"WorldSeeds":[35021],"WinningBot":{"Id":"26b3f10a-36a4-4385-98e2-f3eada093cf6",
"Size":105,"Speed":6,"GameObjectType":1,"CurrentHeading":240,"Position":{"X":232,"Y":-110}}}
```


- Match 6

  Pada match ini dipertandingkan 4 bot buatan penulis dengan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.

logger-publish > {} GameStateLog_2023-02-17_04-53-43_GameComplete.json > ...

```
{"TotalTicks":131,"Players":[{"Placement":1,"Seed":20039,"Score":67,
"Id":"849047e5-3325-4614-ad4e-9417057bd50a","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":2,"Seed":8209,"Score":16,
"Id":"400ef04f-5a6d-4929-9269-f35de7447747","Nickname":"Bless Bot",
"MatchPoints":6},{"Placement":3,"Seed":241,"Score":11,
"Id":"cdcb74c5-1c50-447c-9c9d-50956b941847","Nickname":"Bless Bot","MatchPoints":4},
{"Placement":4,"Seed":1107,"Score":3,"Id":"b0cb7080-1f16-48fa-9d26-fe634350e2ff",
"Nickname":"Bless Bot","MatchPoints":2}],"WorldSeeds":[41968],
"WinningBot":{"Id":"849047e5-3325-4614-ad4e-9417057bd50a","Size":200,"Speed":6,
"GameObjectType":1,"CurrentHeading":248,"Position":{"X":-134,"Y":147}}}
```


- Match 7

  Pada match ini dipertandingkan 4 bot buatan penulis dengan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.


```
{"TotalTicks":147,"Players":[{"Placement":1,"Seed":3280,"Score":41,
"Id":"5e93da1c-bbb6-40c8-ac49-4c9806e75847","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":3,"Seed":18210,"Score":16,
"Id":"bbe39a3e-77ed-475f-9afb-afa5145c973b","Nickname":"Bless Bot",
"MatchPoints":4},{"Placement":4,"Seed":26297,"Score":18,
"Id":"c8e79a79-b9e5-417c-9b5a-188885bf3a57","Nickname":"Bless Bot",
"MatchPoints":2},{"Placement":2,"Seed":17897,"Score":30,
"Id":"e7e325b1-e3e6-400a-97f3-f1445855e997","Nickname":"Bless Bot",
"MatchPoints":6}],"WorldSeeds":[25927],
"WinningBot":{"Id":"5e93da1c-bbb6-40c8-ac49-4c9806e75847","Size":154,"Speed":2,
"GameObjectType":1,"CurrentHeading":13,"Position":{"X":-257,"Y":144}}}
```


- Match 8

  Pada match ini dipertandingkan 3 bot buatan penulis dengan satu referencebot dan yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.


```
{"TotalTicks":206,"Players":[{"Placement":1,"Seed":6475,"Score":47,
"Id":"d074845e-3306-4f72-89ad-9801374216e8","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":2,"Seed":575,"Score":32,
"Id":"bbfcb394-8b49-4601-ae67-eef81e9334bc","Nickname":"Bless Bot",
"MatchPoints":6},{"Placement":3,"Seed":7322,"Score":16,
"Id":"aaaed3b4-14ab-4582-b9f0-73b8ca49f68a","Nickname":"ReferenceBot",
"MatchPoints":4},{"Placement":4,"Seed":17439,"Score":11,
"Id":"c0394dd6-d347-4dd3-9d9a-1d01b51612d7","Nickname":"Bless Bot",
"MatchPoints":2}],"WorldSeeds":[10280],
"WinningBot":{"Id":"d074845e-3306-4f72-89ad-9801374216e8","Size":112,"Speed":2,
"GameObjectType":1,"CurrentHeading":239,"Position":{"X":-243,"Y":-139}}}
```

- Match 9

Pada match ini dipertandingkan 3 bot buatan penulis dengan satu referencebot dan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.


```
{"TotalTicks":412,"Players":[{"Placement":1,"Seed":25463,"Score":68,
"Id":"e4411697-fcef-407d-894b-1d9124767090","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":2,"Seed":11970,"Score":27,
"Id":"a930ec27-feb0-490a-8f72-476bc1a443b7","Nickname":"Bless Bot",
"MatchPoints":6},{"Placement":3,"Seed":11676,"Score":11,
"Id":"35487a9d-121f-4cbc-b696-12901b19e284","Nickname":"Bless Bot",
"MatchPoints":4},{"Placement":4,"Seed":15708,"Score":10,
"Id":"9e0979cf-f9a8-44b4-b0dc-b7f00ec9bbbb","Nickname":"ReferenceBot",
"MatchPoints":2}],"WorldSeeds":[17215],
"WinningBot":{"Id":"e4411697-fcef-407d-894b-1d9124767090","Size":284,"Speed":6,
"GameObjectType":1,"CurrentHeading":266,"Position":{"X":168,"Y":-28}}}
```


- Match 10

Pada match ini dipertandingkan 3 bot buatan penulis dengan satu referencebot dan bot yang dijadikan bot utama bagi penulis adalah bot dengan id yang pertama. Pada pertandingan bot penulis berada di posisi pertama.


```
{"TotalTicks":164,"Players":[{"Placement":1,"Seed":16817,"Score":18,
"Id":"938c7847-5c99-4040-bd40-baeb24243fee","Nickname":"Bless Bot",
"MatchPoints":8},{"Placement":4,"Seed":14115,"Score":6,
"Id":"76054dc9-9a09-4f88-bafb-8a39f2615018","Nickname":"Bless Bot",
"MatchPoints":2},{"Placement":3,"Seed":26018,"Score":9,
"Id":"be7bfb31-202d-407e-b81a-0c391f8583c8","Nickname":"Bless Bot",
"MatchPoints":4},{"Placement":2,"Seed":6083,"Score":24,
"Id":"354b25ec-0131-48ef-b0e7-7a9b92b957b7","Nickname":"ReferenceBot",
"MatchPoints":6}],"WorldSeeds":[36338],"WinningBot":{"Id":"938c7847-5c99-4040-bd40-baeb24243fee",
"Size":127,"Speed":7,"GameObjectType":1,"CurrentHeading":20,"Position":{"X":-85,"Y":234}}}
```


Dari 10 pertandingan yang dilakukan bot yang dibuat oleh penulis berhasil memenangkan pertandingan sebanyak 6 kali. Keoptimalan dari algoritma *greedy* yang dilakukan sekitar 60% . Hal ini cukup optimal, meskipun beberapa hal yang harus dipertimbangkan adalah tidak variatifnya lawan yang ada. Hal ini dikarenakan penulis cukup kesulitan mencari bot lain. Meskipun begitu, menurut penulis dengan lawan reference bot yang cukup kuat serta fitur-fitur yang di set dapat dijalankan dengan baik oleh bot menjadi sebuah tolak ukur algoritma *greedy* ini optimal.

Adapun jika dianalisis lebih jauh, bot yang dibuat oleh penulis ketika mengalami suatu kekalahan posisinya tidak terlalu jauh. Hampir setiap kekalahan bot penulis ada di posisi kedua. Hal ini menguatkan alas an penulis menyebut algoritma *greedy* ini optimal.

Bab 5: Kesimpulan dan Saran

5.1. Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma *greedy* untuk membuat bot permainan Galaxio yang bertujuan untuk bertahan tidak ditabrak oleh musuh dan berusaha untuk terus menerus memperbesar ukuran tubuhnya. Beberapa strategi diatur untuk mewujudkan ini.

Penulis mengatur strategi untuk tiap kemungkinan yang akan didapat, apakah itu terkait dengan keuntungan atau kelebihan yang didapat bot. Semua kemungkinan itu diatur dengan suatu strategi *greedy* tertentu.

Program yang penulis buat tidak sempurna seutuhnya karena karakteristik *greedy* yang tidak selalu mendapatkan hasil yang terbaik secara global membuat terkadang apa yang dihasilkan oleh bot tidak menjadi sebuah keputusan terbaik sehingga menguntungkan bot dan membuat bot dapat tetap bertahan di arena. Terkadang ada suatu masa dimana bot justru memilih suatu pilihan yang salah sehingga membuat bot harus tereliminasi dari permainan. Tapi sejauh ini dari percobaan-percobaan yang dilakukan hasilnya mendekati kemenangan. Secara spesifik dari hasil percobaan yang penulis telah lakukan 60% dari semua pertandingan yang dilakukan dimenangkan oleh bot penulis.

5.2. Saran

Terkait dengan topik ini, berikut beberapa saran yang bisa kita ajukan untuk selanjutnya:

- Proses diskusi harus dilakukan di awal-awal agar memiliki gambaran mengenai pengerjaan tugas.
- Baiknya dilakukan pembagian tugas terlebih dahulu agar *workload*

  masing – masing anggota jelas dan pengerjaan lebih terstruktur

Daftar Pustaka

Munir, Rinaldi. 2020-2021. *Algoritma Greedy Bag 1*. https://informatika.stei.itb.ac.id. 10 Februari 2023

Munir, Rinaldi. 2020-2021. *Algoritma Greedy Bag 2*. https://informatika.stei.itb.ac.id. 10 Februari 2023

Munir, Rinaldi. 2020-2021. *Algoritma Greedy Bag 3*. https://informatika.stei.itb.ac.id. 10 Februari 2023

LINK PENTING

Link Github Kode : https://github.com/satrianababan/Tubes_bless.git

Link Video : https://youtu.be/ZBlliMWq9bc