PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN "GALAXIO"

LAPORAN TUGAS BESAR

Disusun untuk memenuhi salah satu tugas besar
mata kuliah Strategi Algortima
IF2211
Oleh

Bernardus Willson	**13521021**
Raynard Tanadi	**13521143**
Kenneth Dave Bahana	**13521145**

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB I
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.

Pada tugas besar pertama IF2211 Strategi Algoritma ini, kami menggunakan bahasa pemrograman Java untuk membuat algoritma pada *bot.* Selain itu, kami juga menggunakan maven untuk mengubah *bot* yang kami buat menjadi .jar supaya dapat di-*compile*. Untuk menjalankan permainan, kami menggunakan *game engine* yang dibuat oleh Entelect Challenge pada *repository* github yang berjudul 2021-Galaxio.

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 dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar

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

10. Terdapat beberapa *command* yang dapat dilakukan oleh *player.* Setiap tick, *player* hanya dapat memberikan satu *command.* Berikut jenis-jenis dari *command* yang ada dalam permainan:
    a. FORWARD : *Command* ini akan membuat kapal bergerak ke arah yang dituju.
    b. STOP : *Command* ini akan membuat kapal berhenti secara langsung sampai terdapat *command* bergerak lainnya.
    c. STARTAFTERBURNER: *Command* ini akan mengaktifkan *afterburner* yang akan mempercepat kecepatan kapal jika sedang bergerak, tetapi akan memberi *damage* pada kapal.
    d. STOPAFTERBURNER : *Command* ini menghentikan *afterburner*
    e. FIRETORPEDOES : *Command* ini membutuhkan 1 *salvo charge* untuk setiap torpedo yang ditembakkan ke arah yang dituju, tetapi akan memberi *damage* pada kapal.
    f. FIRESUPERNOVA : *Command* ini membutuhkan *supernova pickup* untuk menembak *supernova* yang ditembakkan ke arah yang dituju.
    g. DETONATESUPERNOVA : *Command* ini meledakkan *supernova* yang telah ditembakkan.
    h. FIRETELEPORTER : *Command* ini membutuhkan 1 *teleport charge* untuk setiap *teleporter* yang ditembakkan ke arah yang dituju.
    i. TELEPORT : *Command* ini akan memindahkan posisi kapal ke posisi *teleporter* yang telah ditembakkan.
    j. USESHIELD : *Command* ini mengaktifkan perisai untuk memantulkan torpedo yang ditembakkan ke arah kapal (jika perisai tersedia).
11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal *player* lain, maka score bertambah 10, jika mengonsumsi *food* atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan score tertinggi.

BAB II
LANDASAN TEORI

A. Dasar Teori Algoritma *Greedy* Secara Umum

Greedy atau serakah berarti sikap yang selalu merasa tidak puas dari hal yang dimilikinya. Algoritma yang menggunakan elemen serakah ini atau algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah selalu mengambil pilihan yang terbaik pada setiap saat, dengan memilih pilihan optimum lokal dengan harapan berakhir pada optimum global.

Terdapat beberapa elemen algoritma *greedy*, seperti :

- Himpunan Kandidat (C) : himpunan dari yang akan dipilih pada setiap langkah.
- Himpunan Solusi (S) : himpunan dari kandidat yang sudah dipilih.
- Fungsi Solusi : menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
- Fungsi Seleksi :memilih kandidat berdasarkan strategi *greedy* tertentu (bersifat heuristik).
- Fungsi Kelayakan : memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
- Fungsi Obyektif : memaksimumkan atau meminimumkan.

Dengan elemen - elemen di atas, dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian S dari himpunan kandidat C di mana S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan dioptimasi oleh fungsi obyektif. Berikut merupakan skema umum algoritma *greedy* :

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
   x : kandidat
   S : himpunan_solusi

Algoritma:
   S ← {}      { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
        x ← SELEKSI(C)     { pilih sebuah kandidat dari C}
        C ← C – {x}        { buang x dari C karena sudah dipilih }
        if LAYAK(S ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
            S ← S ∪ {x}     { masukkan x ke dalam himpunan solusi }
        endif
   endwhile
   {SOLUSI(S) or C = {} }

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

Pada algoritma tersebut, akan terbentuk solusi yang optimum lokal pada akhir setiap iterasi dan akan diperoleh solusi optimum global (jika ada) pada akhir while-do. Optimum global sendiri belum tentu merupakan solusi terbaik. Hal ini terjadi karena algoritma *greedy* tidak memeriksa semua kemungkinan solusi yang ada, melainkan terdapat beberapa fungsi seleksi yang berbeda-beda sehingga harus memilih fungsi yang tepat jika ingin menghasilkan solusi yang optimal.

Terdapat banyak persoalan yang dapat diselesaikan dengan menggunakan algoritma greedy. Salah satu persoalan yang dapat diselesaikan algoritma greedy adalah persoalan *knapsack*. Persoalan *knapsack* adalah memilih objek-objek yang dimasukkan ke dalam *knapsack* sedemikian sehingga memaksimumkan keuntungan. Total bobot objek yang dimasukkan ke dalam knapsack tidak boleh melebihi kapasitas knapsack. Objek - objek tersebut memiliki properti seperti ukuran, berat, dan properti lain - lainnya. Berikut formulasi sederhana penyelesaian *knapsack*.

Maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*)

$$\sum_{i=1}^{n} w_i x_i \leq K$$

yang dalam hal ini, $x_i = 0$ atau 1, $i = 1, 2, \ldots, n$

Maksimasi hasil yang diinginkan, yaitu suatu properti F diselesaikan dengan perbatasan suatu kapasitas atau *constraint* yang dinyatakan k untuk seluruh kemungkinan objek (x) yang ada. Keserakahan dari persoalan ini dapat ditentukan dari salah satu propertinya seperti mencari ukuran yang sekecil mungkin, mencari berat yang sekecil mungkin, atau bahkan memprioritaskan keduanya secara bersamaan untuk mendapatkan hasil yang terbaik

B. Cara Kerja Bot Galaxio Secara Umum

Secara umum, *Bot Galaxio* bertujuan untuk memenangkan permainan dengan menjadi bot terakhir yang hidup. Bot ini secara umum bermain sebagai kapal dengan cara mencari makanan atau menembak musuh untuk bertambah lebih besar, kemudian memakan musuh lain yang lebih kecil untuk memenangkan permainannya. Terdapat beberapa nilai yang dipunyai oleh kapal, seperti *speed* yang melambangkan kecepatan dari kapal pada permainan dengan kecepatan awal sebesar 20 dan terus berkurang seiring dengan bertambahnya ukuran, *size* yang melambangkan ukuran dari kapal dengan ukuran awal sebesar 10 , dan *heading* yang melambangkan arah tujuan dari kapal untuk bergerak yang berkisar dari 0 derajat sampai dengan 359 derajat. Kapal tidak akan bergerak sampai diberikan perintah untuk bergerak dan jika sedang bergerak, tidak akan berhenti bergerak sampai diberikan perintah untuk berhenti. Kapal juga harus memiliki *size* di atas 5, jika tidak, maka kapal akan dihilangkan dari peta dan dianggap telah tereliminasi.

Kapal dapat melakukan *perintah* berupa aksi yang ada seperti:

- Aksi *Forward* untuk bergerak maju sesuai arah menghadap (*heading*)
- Aksi *Stop* untuk menghentikan aksi *Forward*
- Aksi *Start Afterburner* untuk bergerak maju dengan tambahan kecepatan (*boost*)
- Aksi *Stop Afterburner* untuk menghentikan aktivasi dari *afterburner*.

- Aksi *Fire Torpedoes* untuk menembakkan peluru yang bisa mengenai bot lain
- Aksi *Fire Supernova* untuk menembakkan peluru *supernova* yang bisa meledak
- Aksi *Detonate Supernova* untuk meledakkan peluru yang sedang ditembak
- Aksi *Fire Teleport* untuk menembakkan peluru yang digunakan untuk berpindah tempat
- Aksi *Teleport* untuk mengaktivasi *Fire teleport* dan bot berpindah tempat ke tempat peluruh sedang berada
- Aksi *Activate Shield* untuk mengaktivasi perlindungan yang mencegah tembakan *torpedo* mengenai bot tersebut.

Selain itu, kapal juga dapat mengambil data dari *game object* yang ada, seperti :

- Objek *Food* yang dimakan bot untuk bertambah besar
- Objek *Super Food* yang dimakan bot dan bertambah ukuran sangat besar
- Objek *Wormholes* yang merupakan entitas untuk berpindah tempat bot jika mengenainya dan berpindah ke *wormhole* lain yang ada sesuai pilihan
- Objek *Gas Clouds* yang menyebabkan bot mengecil setiap *tick*-nya apabila berada di dalamnya
- Objek *Asteroid Fields* yang menyebabkan bot bergerak lebih lambat didalamnya
- Objek *Torpedo Salvo* yang merupakan amunisi penembakan *torpedo*
- Objek *Supernova* yang merupakan amunisi penembakkan *supernova*
- Objek *Teleport* yang merupakan amunisi untuk berpindah ke suatu tempat.
- Objek *Shield* yang merupakan pertahanan bot.

Untuk setiap objek yang ada, kapal dapat mengambil data mengenai objek tersebut yang meliputi :

- *Size* sebagai ukuran objek
- *Speed* sebagai kecepatan objek
- *Heading* sebagai arah menghadap objek
- *GameObjectType* sebagai tipe suatu objek, yaitu *food, wormhole, gas cloud,* dan *asteroid field*
- *X Position* sebagai keterangan letak bot pada X-Axis
- *Y Position* sebagai keterangan letak bot pada Y-Axis

Kapal juga dapat mengambil data dari setiap bot yang masih hidup dan terdapat pada peta, antara lain adalah :

- *Size* sebagai ukuran bot
- *Speed* sebagai kecepatan bot
- *Heading* sebagai arah menghadap bot
- *GameObjectType* sebagai tipe suatu objek, yaitu *player*
- *X Position* sebagai keterangan letak bot pada X-Axis
- *Y Position* sebagai keterangan letak bot pada Y-Axis
- *Active Effects* sebagai keterangan efek yang mempengaruhi bot pada saat itu.

Dengan berbagai aksi yang dapat dilakukan dan data yang diambil, bot dapat mengambil keputusan pergerakan bot setiap *tick* atau setiap *frame* pada permainan dikomputasikan melalui algoritma yang dikondisikan dalam *BotService,* yaitu diutamakan pada fungsi *computeNextPlayerAction* yang mengatur setiap keputusan aksi yang dilakukan untuk setiap *tick* tertentu.

C. Cara Menjalankan *Game Engine* Permainan Galaxio

Untuk menjalankan *Game Engine*, terdapat tiga hal utama yang harus dijalankan terlebih dahulu, yaitu *runner*, *engine*, dan *logger game* tersebut. Pertama, harus dijalankan secara berurutan *runner* game terlebih dahulu dengan menjalankan “dotnet GameRunner.dll” pada *directory folder* runner-publish, kemudian menjalankan “dotnet Engine.dll” pada engine-publish, dan terakhir menjalankan “dotnet Logger.dll” sehingga pemetaan permainan sudah siap dan telah dijalankan.

Setelah permainan berjalan, bot - bot baik berbentuk referensi maupun buatan (digunakan dalam bahasa Java), dimasukkan ke dalam permainan dengan cara mengakses *directory* langsung hasil buatan program yang telah dimasukkan dalam arsip dalam bentuk *jar* pada command “java -jar <*directory*-nya>”. Apabila menggunakan referensi bot yang ada, maka jalankan “dotnet ReferenceBot.dll” pada directory reference-bot-publish. Seluruh tahapan tersebut dapat dijalankan langsung dalam sebuah file run.bat dalam folder *starter-pack* dan dijalankan seperti lampiran berikut.

```
@echo off
:: Game Runner
cd ./runner-publish/
start "" dotnet GameRunner.dll

:: Game Engine
cd ../engine-publish/
timeout /t 1
start "" dotnet Engine.dll

:: Game Logger
cd ../logger-publish/
timeout /t 1
start "" dotnet Logger.dll

:: Bots
cd ../reference-bot-publish/
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" dotnet ReferenceBot.dll
timeout /t 3
start "" java -jar "C:\Users\Dave Bahana\Downloads\Full program STIMA1\starter-pack\starter-bots\JavaBot\target\JavaBot.jar"
cd ../

pause

```

Contoh berikut menggunakan 3 bot referensi dan diakhiri dengan buatan bot dalam bahasa Java yang telah dimasukkan dalam sebuah arsip jar hasil pemrograman aksi - aksi *bot* tersebut. Menjalankan permainan tersebut cukup dengan membuka arsip *run.bat* yang dibuat tersebut dan secara langsung akan dipanggil setiap perintah yang dibutuhkan hingga game berjalan. Setelah terminal menjalankan permainan tersebut, dalam folder *visualizer*, *unzip* terlebih dahulu visualisasi sesuai sistem operasi gawai yang digunakan dan jalankan aplikasi *Galaxio*. Pertama, aturkan opsi terlebih dahulu *directory* yang digunakan adalah folder *logger-publish* dalam untuk mengakses riwayat permainan yang telah jalan pada terminal, kemudian buka arsip pada opsi *load* dalam game sesuai dengan keterangan waktu permainan dijalankan.

BAB III
APLIKASI STRATEGI GREEDY

A. Proses Mapping Persoalan Galaxio menjadi Elemen - Elemen Algortima Greedy

1. Greedy by Food
   Ini pengertian
   a. Mapping Elemen Greedy

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi melakukan aksi FORWARD dan perubahan arah *heading* atau tidak melakukan aksi tersebut dalam setiap *tick*-nya
Himpunan solusi	Hasil dari variasi penggunaan aksi - aksi yang ada menyebabkan penambahan ukuran bot semaksimal mungkin dengan cara rakus akan mencari area yang memiliki paling banyak makanan.
Fungsi solusi	Melakukan pengecekkan dengan sistem *scanning* serta dilakukan *scoring* akan daerah yang memiliki poin terbesar sebagai area terbaik untuk bot tempati
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik hasil kumpulan data *scoring* dan *scanning*, kemudian memilih jalur yang tepat berdasarkan hasil tersebut dengan melakukan pemindahan heading (arah menghadap) dan melakukan aksi FORWARD berdasarkan hasil tersebut
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid. Daftar command yang valid pada strategi ini adalah bot melakukan command FORWARD, dan mengubah arah *heading*, atau hanya melakukan pengubahan arah *heading* saja.
Fungsi objektif	Mencari permutasi dari perintah yang membuat bot berukuran sebesar mungkin dengan cara memakan *food* sebanyak mungkin.

b. Analisa efisiensi

Greedy by *food* melakukan pengecekan area dengan cara pembagian area untuk setiap 30 derajat dari bot. Hasil pengecekkan tersebut menghasilkan daftar area ataupun makanan terdekat yang ada dekat bot pada saat itu. Untuk kasus terburuk, apabila algoritma mencari area yang terbaik untuk dijelajahi bot berdasarkan hasil *scoring*, maka kompleksitas algoritma tersebut adalah O(n) dari perumusan n+1 dimana 1 tersebut merupakan melakukan aksi FORWARD dan n adalah sejumlah tuple yang dihasilkan proses scanFood, yaitu pada algoritma bersifat konstan 12 area (360 derajat dibagi 30 derajat untuk setiap daerah dan dibandingkan satu per satu dalam sebuah *for loop* untuk mencari jarak terdekat. Apabila algoritma menggunakan algoritma *default food* yang hanya mengejar makanan, karena *best area* sudah tidak memenuhi kondisi atau sudah tidak ada, maka hal tersebut merupakan kasus terbaik dalam kompleksitas, yaitu O(1)

c. Analisa efektivitas

Greedy by food ini karena mengutamakan memakan makanan sebanyak mungkin selama permainan berlangsung, tentunya keuntungan besar utama yang didapatkan adalah ukuran yang sebesar mungkin bot dapatkan melalui makanan. Selain itu, *greedy by food* ini bersifat sederhana sehingga mudah untuk diimplementasikan beriringan dengan strategi lain dan strategi ini tidak akan menghalangi, bahkan menjadi cadangan yang membantu dalam strategi *greedy* Namun, kendala dari *greedy by food* sendiri adalah tingkat agresifnya yang sangat kurang. Dalam permainan ini, penting sekali membutuhkan agresi untuk bersaing dengan kompleksitas dan strategi *greedy* yang lain. Oleh karena itu, hanya dengan *greedy by food* sendiri membatasi kesempatan - kesempatan yang mungkin jauh lebih baik untuk bot pada setiap saat.

2. Greedy by Time

Greedy by Time adalah algoritma greedy yang memprioritaskan ketahanan atau durasi bertahan bot dalam permainan dengan cara menjauh dari segala kemungkinan halangan (*obstacle*) yang ada dalam permainan. Apabila ada halangan seperti GAS CLOUD, maka algoritma greedy ini akan berjalan menghindari halangan tersebut dan apabila ada TORPEDOSALVO dari bot lawan mengarah ke bot, maka dilakukan aksi ACTIVESHIELD untuk menangkis tembakan *torpedo* tersebut.

a. Mapping Elemen Greedy

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi melakukan aksi FORWARD, aksi ACTIVATESHIELD, FIRETORPEDOES, perubahan arah *heading*, atau tidak melakukan aksi - aksi tersebut dalam setiap *tick*-nya.
Himpunan solusi	Hasil dari variasi penggunaan aksi - aksi yang ada menyebabkan pengurangan ukuran bot sekecil

	mungkin baik dari hadangan maupun perlawanan dari lawan sehingga menghasilkan waktu bertahan di dalam permainan lama.
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot berjalan seperti seharusnya. Jika ada halangan sekitar bot menghindarinya atau jika ada musuh yang lebih besar, maka arah pergerakan bot dengan aksi FORWARD mengarah menjauh dari bot atau berlawanan. Jika ditembak FIRETORPEDOES oleh bot lawan, maka dilakukan aksi ACTIVATESHIELD, dan jika bot lawan lebih besar mendekat, maka dilakukan FIRETORPEDOES sebagai *defensive shooting* atau mencoba melindungi diri dengan menembakkan *torpedo* pada musuh dengan *fire rate* yang tinggi.
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik hasil kumpulan data halangan (*obstacle*) dekat bot pada jarak tertentu seperti GASCLOUD, ASTEROIDFIELD, WORMHOLE, tembakan TORPEDOSALVO dari bot lawan, serta bot lawan yang lebih besar.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid. Daftar command yang valid pada strategi ini adalah bot melakukan command FORWARD, ACTIVATESHIELD, mengubah arah *heading*, atau hanya melakukan pengubahan arah *heading* saja.
Fungsi objektif	Mencari permutasi dari perintah yang membuat bot hidup selama mungkin dengan menerima pengurangan ukuran bot sekecil mungkin atau tidak sama sekali.

b. Analisa efisiensi

Greedy by time melakukan pengecekan posisi bot dengan lawan yang lebih besar, bot dengan *obstacles* atau halangan tertentu seperti *gas cloud*, *asteroid*, dan *wormhole* hingga jarak bot dengan batasan api pada permainan. Hasil pengecekan tersebut kemudian digunakan untuk kesimpulan pemilihan aksi apabila diprioritaskan untuk menjauh dari suatu halangan yang paling dekat dengan bot dan dapat berdampak pada pengancaman bot kalah, atau perlindungan dari penembakkan musuh yang mendekat, serta penembakkan *defensive shooting*

dalam *fire rate* yang tinggi apabila musuh besar mendekat mencoba memakan dari jarak dekat. Maka suatu aksi yang mungkin antara FORWARD ACTIVATESHIELD, atau FIRETORPEDOES dilakukan. Sehingga, kompleksitasnya adalah O(1).

c. Analisa efektivitas

Greedy by time yang mengutamakan durasi bertahan selama mungkin akan dibatasi dengan proses *shrinking* atau mengecilnya ukuran medan permainan pada setiap *tick*.

Algoritma ini menguntungkan apabila ukuran bot sangat kecil sehingga kurangnya kesempatan bot untuk menyerang dan mencegah perlawanan dari bot lawan yang mungkin memprioritaskan perlawanan terhadap bot yang kecil. Algoritma ini juga menguntungkan apabila masih tersedia banyaknya makanan dalam meda permainan sehingga menjauh dari halangan yang merugikan dan memiliki banyak kesempatan dari segi lain, baik dalam hal penambahan ukuran maupun penyerangan bot lawan.

Prioritas terhadap waktu juga memiliki keterbatasannya dalam beberapa kondisi. Algoritma ini kurang efektif apabila terlalu banyak halangan yang hampir mengelilingi bot sehingga tidak ada keputusan yang terbaik untuk menghindar, melainkan mementingkan aspek lain. Perkembangan dalam segi ukuran dan mengambil kesempatan dalam menyerang menjadi berkurang dan dapat menyebabkan bot terpuruk hingga kalah walaupun disisi lain berkesempatan juga untuk mengambil kesempatan di lain waktu.

3. Greedy by Enemy Size

Greedy by Enemy Size adalah algoritma greedy yang memiliki prioritas untuk membuat kapal lawan terdekat menjadi sekecil mungkin. Hal ini dilakukan dengan beberapa cara, seperti menembakan torpedo, memakan dengan cara melakukan teleport atau mengejar seperti biasa, dan menembakkan supernova. Dengan mekanisme permainan di mana size bot dapat bertambah dengan mengurangi atau memakan kapal lawan, hal ini dapat menguntungkan bot dalam hal melemahkan kapal lawan sekaligus memperkuat bot.

a. Mapping Elemen Greedy

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi melakukan aksi FIRETORPEDOES, aksi FIRESUPERNOVA, aksi FIRETELEPORT, aksi DETONATESUPERNOVA, aksi TELEPORT, aksi FORWARD disertai dengan perubahan arah heading ke kapal lawan, atau tidak

	melakukan aksi - aksi tersebut dalam setiap *tick*-nya.
Himpunan solusi	Hasil dari variasi penggunaan aksi - aksi yang ada menyebabkan pengurangan size kapal lawan sampai sekecil mungkin sehingga size kapal lawan berkurang menjadi lebih kecil atau tereliminasi.
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot berjalan seperti seharusnya. Jika tidak terdapat ammo untuk torpedo, teleport, dan supernova, maka bot tidak melakukan aksi yang menembakkan objek tersebut.
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas yang didasarkan dengan strategi pengurangan size kapal lawan yang paling optimum.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid. Daftar command yang valid pada strategi ini adalah bot melakukan command FIRETORPEDOES, FIRESUPERNOVA, FIRETELEPORT, DETONATESUPERNOVA, TELEPORT, dan FORWARD.
Fungsi objektif	Mencari permutasi dari perintah yang membuat bot dapat mengurangi size kapal lawan sampai sekecil mungkin atau tereliminasi.

b. Analisa efisiensi

Greedy by Enemy Size melakukan pengecekan posisi bot dengan kapal musuh, size bot dengan musuh, dan ammo untuk melakukan penembakkan objek. Hasil pengecekan tersebut kemudian digunakan untuk kesimpulan pemilihan aksi yang diprioritaskan untuk mengurangi size dari kapal musuh, maka suatu aksi yang mungkin antara FIRESUPERNOVA dengan DETONATESUPERNOVA atau FORWARD atau FIRETELEPORT dengan ACTIVESHIELD dilakukan. Sehingga, untuk kasus terburuk, yaitu hampir semua kondisi yang memungkinkan kompleksitasnya adalah O(n) dari perumusan n+1 dimana 1 merupakan melakukan aksi baik FIRETORPEDOES, FORWARD, FIRETELEPORT, maupun ACTIVATESHIELD serta n merupakan jumlah tuple hasil scanning dalam list, yaitu scan objek - objek yang ada di depan musuh serta sekitarnya, dikomparasikan dengan *minDistance* atau jarak terdekat. Namun, terdapat juga kasus penembakkan FIRESUPERNOVA

dilakukan. Sehingga, untuk kasus terburuk, yaitu hampir semua kondisi yang memungkinkan kompleksitasnya adalah O(n) dari perumusan n+1 dimana 1 merupakan melakukan aksi baik FIRETORPEDOES, FORWARD maupun ACTIVATESHIELD serta n merupakan jumlah tuple hasil scanning dalam list, yaitu scan objek - objek yang ada di depan musuh serta sekitarnya, dikomparasikan dengan *minDistance* atau jarak terdekat. Namun, apabila terjadi aksi FIRESUPERNOVA dalam strategi *greedy by enemy size* ini, maka hal tersebut memberikan kasus terbaik kompleksitas yaitu O(1).

c. Analisa efektivitas

Greedy by Enemy Size memiliki prioritas untuk membuat lawan sekecil mungkin atau tereliminasi. Hal ini dapat dicapai dengan menggunakan beberapa aksi yang telah disebutkan di atas, seperti menembakkan torpedo, supernova, teleport, dan mengejar musuh.

Algoritma ini menguntungkan apabila algoritma musuh terkait penggunaan shield dan penghindaran buruk. Algoritma ini juga menguntungkan apabila terdapat tidak begitu banyak object yang terdapat di antara kapal lawan yang disasar dan bot sehingga penembakan torpedo tidak terhalang. Selain itu, algoritma ini juga menguntungkan apabila ukuran dari bot tidak terlalu kecil sehingga saat melakukan aksi - aksi yang mengurangi size bot, bot tidak rentan tereliminasi karena size yang terlalu kecil.

Prioritas terhadap enemy size juga memiliki keterbatasannya dalam beberapa kondisi. Algoritma ini kurang efektif apabila algoritma musuh terkait penggunaan shield dan penghindaran baik, lalu terlalu banyak halangan yang terdapat diantara kapal lawan yang disasar dan bot. Apabila tembakkan dari bot berhasil dihindari atau terhalang suatu objek maka algoritma greedy by enemy size akan menjadi kurang efektif. Selain itu, jika terdapat banyak gas clouds di sekitar bot atau terdapat kapal - kapal musuh lain yang menyerang bot dan mengurangi size bot, maka algoritma greedy by enemy size juga akan menjadi kurang efektif karena rentan tereliminasi.

B. Eksplorasi Alternatif Solusi Greedy yang Mungkin Dipilih dalam Persoalan Galaxio

a. Strategi Heuristik Food

Strategi Heuristik dalam Food memiliki satu faktor penting yaitu berkaitan dengan pengecekkan daerah sekitar yang membuat strategi mencari makanan itu sendiri bekerja. Dalam implementasi *greedy by food*, diperlukan aspek *scanning* dan *scoring* yang memilah data makanan sekitar menjadi bagian - bagian area dengan data makanan terbanyak sekaligus halangan terkecil dari proses *scoring*nya sendiri. Strategi ini penting sekali ketika bot mencari makanan yang mungkin pada standarnya dapat mengejar

makanan terdekat dengan bot pada saat itu. Namun, harus diperhatikan juga arah bot tersebut apakah mengarah ke tempat yang banyak makanan atau malah banyak halangan, sehingga strategi heuristik ini mencari penopang dari strategi serakah akan makanan.

b. Strategi Heuristik Time

Intinya: diperlukan heurestik buat bikin kondisi kapan harus activateshield,

Strategi Heuristik ini sangat penting dalam memilih serta memilah hasil *scanning* dan *game state* dari data berbagai halangan yang muncul disekitar bot untuk memilih aksi yang tepat berdasarkan jenis halangan yang diprioritaskan untuk dihindari bot. Strategi ini memberikan sejumlah kondisi tertentu yang dikategorikan berdasarkan jarak halangan yang dihasilkan data serta jenis halangan itu sendiri untuk menentukan aksi yang tepat apabila memasuki kondisi tersebut. Untuk memformulasikan algoritma ini, diperlukan beberapa strategi heuristik untuk menangani penghindaran suatu halangan.

Strategi Heuristik yang pertama adalah apabila ada suatu halangan berupa objek yang sifatnya tidak bergerak ataupun menumbuh perlahan (tanpa perpindahan signifikan) berupa *GAS CLOUD*, *WORMHOLE*, dan *ASTEROIDFIELD*, maka informasi halangan tersebut dijadikan area yang poinnya relatif sedikit (dikurangi berdasarkan jumlah dan proporsi dari keberadaannya) dalam sistem *scanning* sehingga strategi ini mengutamakan hasil pengarahan heading yang menghindari serta aksi FORWARD untuk menjauh, baik dengan berlawanan arah atau hanya berbelok berdasarkan hasil *scanning* yang dilakukan.

Strategi Heuristik yang kedua adalah untuk halangan yang sifatnya bergerak dengan cukup cepat dan utamanya terfokuskan untuk menghalang bot, maka strategi untuk mengambil informasi data pada kondisi tersebut dan mengesekusi aksi ACTIVATESHIELD untuk menangkis halangan tersebut yang berupa TORPEDOSALVO hasil penembakkan bot lawan. ACTIVATESHIELD ini dilakukan supaya hasil tabrakan torpedo tersebut dengan bot ditangkis dan tidak menyebabkan mengecilnya ukuran bot.

Strategi Heuristik yang ketiga adalah untuk informasi posisi pada jarak tertentu yang relatif dekat terdapat bot musuh yang mendekati dan lebih besar, maka bot melakukan aksi FORWARD dengan arah *heading* menjauh dari bot lawan, baik berlawanan maupun berbelok untuk mencegah keluar dari perbatasan medan permainan ataupun mencegah dari halangan lain yang mungkin ada di sekitarnya.

Strategi Heuristik yang terakhir adalah aksi yang diperlukan dari informasi perbatasan medan perang. Apabila bot berada cukup dekat dengan perbatasan medan perang dan terancam untuk keluar dari medan dan kalah dalam permainan, aksi FORWARD disertakan dengan heading dengan implementasi menuju ke posisi (0,0) dunia, yaitu titik tengah medan pertarungan supaya bot tidak termakan oleh daerah luar medan perang yang secara signifikan mengecilkan ukuran bot.

c. Strategi Heuristik Enemy Size

Strategi Heuristik dibutuhkan dalam memilih aksi yang diprioritaskan berdasarkan data dari game state yang didapatkan untuk mengurangi size dari kapal musuh. Dalam strategi ini, terdapat sejumlah kondisi tertentu yang didasarkan oleh jarak bot dengan kapal musuh, size bot dibandingkan dengan size kapal musuh, dan ketersediaan ammo. Untuk memformulasikan algoritma ini, diperlukan beberapa strategi heuristik untuk menangani beberapa kondisi berbeda.

Pada strategi heuristik yang pertama, bot akan menembakkan supernova dengan melakukan aksi FIRESUPERNOVA dengan heading ke arah kapal musuh apabila bot memiliki ammo untuk menembakkan supernova atau supernovaAvaible ==1, jarak antara bot dengan kapal musuh berbeda lebih dari setengah kali worldradius saat itu, dan kapal musuh merupakan kapal dengan size terbesar saat itu dan lebih besar minimal sepuluh persen dari size bot. Setelah ditembakkan, supernova akan didetonasi dengan menggunakan aksi DETONATESUPERNOVA dengan rumus jarak antara bot dengan kapal musuh dibagi dengan kecepatan supernova. Dengan menembakkan supernova dan mendetonasinya, kapal musuh yang terkena akan mengecil secara signifikan.

Pada strategi heuristik yang kedua, bot akan mengejar kapal musuh dengan melakukan aksi FORWARD dengan heading yang diarahkan ke kapal musuh, apabila kapal musuh yang terdekat dengan bot memiliki jarak dibawah 100 dan berukuran lebih kecil dua puluh persen dari bot, serta bot tidak sedang menembakkan teleporter. Dengan kondisi jarak yang dekat diantara bot dengan kapal musuh, strategi ini dapat membuat bot memakan kapal musuh sehingga musuh tereliminasi.

Pada strategi heuristik yang ketiga, bot akan menembakkan teleporter ke arah kapal musuh dengan melakukan aksi FIRETELEPORT dengan heading yang diarahkan ke kapal musuh, apabila kapal musuh yang terdekat dengan bot memiliki jarak diatas 150 dan dibawah 500, kapal musuh berukuran lebih kecil sepuluh persen ditambah 40 dari bot, bot memiliki size diatas 100, bot memiliki ammo untuk menembakkan teleport dan menggunakan shield yaitu teleporterCount >0 dan shieldCount > 0, serta bot tidak sedang menembakkan teleporter ataupun menggunakan shield. Setelah teleporter ditembakkan, bot akan mengaktifkan shield dengan aksi ACTIVESHIELD guna menjaga size supaya tidak berkurang saat menunggu teleporter didetonasi. Lalu, bot akan mendetonasi teleporter dengan aksi TELEPORT dengan rumus jarak antara bot dengan kapal musuh dibagi dengan kecepatan teleporter, apabila kapal musuh yang dituju tetap memiliki size yang lebih kecil dari bot dan terdapat teleporter yang masih aktif. Selama teleporter ditembakkan dan belum didetonasi, bot tidak dapat menembakkan torpedo maupun teleporter lain. Dengan strategi ini, bot dapat memakan kapal musuh yang lebih kecil sampai tereliminasi secara cepat.

Pada strategi heuristik yang keempat, bot akan menembakkan torpedo secara ofensif ke arah kapal musuh dengan melakukan aksi FIRETORPEDOES dengan heading ke arah kapal musuh, apabila terdapat kapal musuh yang memiliki size lebih kecil minimal dua puluh persen dari bot, jarak antar kapal musuh dengan bot dibawah 150, bot

memiliki ammo untuk menembakkan torpedo atau torpedoSalvoCount >0, bot memiliki size di atas 50, dan bot tidak sedang menembakkan teleporter. Sebelum melakukan aksi FIRETORPEDOES, bot akan memeriksa apakah terdapat objek yang menghalangi bot dengan kapal musuh, jika terdapat objek yang menghalangi, bot tidak jadi melakukan aksi FIRETORPEDOES. Dengan strategi ini, bot dapat mengurangi size kapal musuh dengan akurasi yang cukup bagus karena jarak yang cukup dekat dan tidak terdapat objek yang menghalangi.

Pada strategi heuristik yang terakhir, bot akan menembakkan torpedo ke arah kapal musuh terdekat dengan melakukan aksi FIRETORPEDOES dengan heading ke arah kapal musuh, apabila tidak terdapat makanan yang tersisa di medan perang. ebelum melakukan aksi FIRETORPEDOES, bot akan memeriksa apakah terdapat objek yang menghalangi bot dengan kapal musuh, jika terdapat objek yang menghalangi, bot tidak jadi melakukan aksi FIRETORPEDOES. Dengan strategi ini, bot dapat mengurangi size kapal musuh tanpa terhalang objek sebagai opsi terakhir untuk memenangkan permainan.

d. Pembobotan objek

Pada keseluruhan entitas seperti objek dan atribut dari pemain yang ada, informasi sekitar pada bot dapat di lakukan *scanning* dan hasil keseluruhan *scanning* tersebut dibentuk dalam proses *scoring* dimana setiap objek yang ditemukan memiliki nilai dan menjadi penentuan area prioritas untuk bot kunjungi. Hasil proses poin ini kemudian digunakan dalam pencarian makanan pada peta permainan yang memiliki makanan terbanyak serta halangan berupa *gas cloud, asteroid, wormhole,* musuh yang lebih besar, serta seluruh kemungkinan halangan lainnya yang berjumlah paling minimal, didasarkan dengan *scoring system* yang dibuat.

C. Strategy Greedy yang dipilih

Pada poin - poin sebelumnya, terdapat banyak strategi heuristik yang diimplementasikan tergantung kondisi dan keadaan yang ada. Masing - masing dari strategi tersebut memiliki kelebihan dan kekurangannya masing - masing sehingga algoritma greedy utama dari program bot permainan Galaxio ini adalah penggabungan dari seluruh strategi yang terdapat di poin - poin sebelumnya. Hal ini dilakukan supaya bot dapat melakukan aksi yang optimal di berbagai kondisi yang berbeda. Untuk menjadikan seluruh strategi heuristik menjadi algoritma greedy, dibutuhkan urutan prioritas dari strategi - strategi tersebut.

Urutan prioritas dari pengeksekusian strategi - strategi heuristik yang ada sesuai dengan kondisi masing - masing adalah sebagai berikut :

1. Menembak supernova jika kapal musuh terbesar berjarak jauh dari bot (Strategi Heuristik Enemy Size pertama)
2. Mengaktifkan shield saat terdapat TORPEDOSALVO hasil tembakkan kapal musuh dekat (Strategi Heuristik Time kedua)

3. Menembak kapal musuh jika lebih besar dari bot atau defensive shooting (Strategi Heuristik Time kelima)
4. Kembali ke tengah jika bot cukup dekat dengan perbatasan medan perang (Strategi Heuristik Time keempat)
5. Menjauh dari kapal musuh atau kabur jika size yang dimilikinya lebih besar dari bot dan dekat (Strategi Heuristik Time ketiga)
6. Mengejar kapal musuh yang berukuran lebih kecil dari bot jika dekat (Strategi Heuristik Enemy Size kedua)
7. Teleportasi ke posisi kapal musuh yang lebih kecil jika berada di jangkauan yang ditentukan (Strategi Heuristik Enemy Size ketiga)
8. Menembak torpedo ke arah kapal musuh jika berukuran lebih kecil dari bot dan dekat atau offensive shooting (Strategi heuristik Enemy Size keempat)
9. Mengambil makanan saat dikejar (Strategi Heuristik Food Pertama)
10. Mencari area terbaik untuk dituju (Strategi Pembobotan Objek & Strategi Heuristik Time pertama)
11. Menembak ke kapal musuh terdekat jika sudah tidak terdapat makanan di medan perang (Strategi Heuristik Enemy Size kelima)

Algoritma greedy yang dibuat harus selalu menghasilkan suatu aksi untuk setiap ticknya agar bot tidak diam pada kondisi tertentu sehingga dibuat aksi default berupa bot bergerak maju dengan aksi FORWARD, serta dibuat aksi mencari makanan terdekat untuk 5 tick pertama untuk menghindari hal tersebut.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma Greedy pada Program Bot

1. Repository Github

Bot yang kami buat dapat diakses melalui repository Github pada tautan berikut: https://github.com/bernarduswillson/Tubes1_paSTI-MAh-rank-1.git

2. Implementasi Dalam Pseudocode

Implementasi algoritma greedy pada program terdapat pada file BotService.java, dalam method BotService. Berikut kami lampirkan beberapa fungsi dan prosedur yang telah kami buat dalam method tersebut.

2.1 Prosedur computeNextPlayerAction

```
PROCEDURE computeNextPlayerAction(playerAction : PlayerAction)
```

2.1.1 Tembak Supernova

```
{Shoot supernova when enemy is the biggest and far away}

    // get biggest enemy size
    LET isEnemyBiggest ←    gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .filter(bot -> bot.size*1.1 > this.bot.size)
                            .max(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .orELSE(NULL)

    IF (
    isEnemyBiggest NOT NULL AND
    getDistanceBetween(isEnemyBiggest, this.bot) > (0.5*this.gameState.world.radius) + this.bot.size + bot.size AND
    this.bot.supernovaAvailable = 1 AND
    isAction = 0
    ) THEN
        playerAction.heading ← getHeadingBetween(isEnemyBiggest)
        playerAction.action ← PlayerActions.FIRESUPERNOVA
        supernovaTick ← this.gameState.world.getCurrentTick()
        supernovaActiveIn ← (getDistanceBetween(isEnemyBiggest, this.bot) - (isEnemyBiggest.size * 0.5) -
this.bot.size) / 20
        supernovaWasShot ← supernovaWasShot + 1
        isAction = 1
    END IF

    // detonate supernova
    IF (
    supernovaTick + supernovaActiveIn = currentTick AND
    supernovaWasShot >= 1 AND
    isAction = 0
```

```
    ) {
        playerAction.action ← PlayerActions.DETONATESUPERNOVA
        isAction ← 1
        supernovaWasShot ← 0
    }
```

2.1.2 Shield

```
{Shield when enemytorpedo is close}

    // get torpedoes heading to this bot
    LET enemyTorpedoes ←    gameState.getGameObjects().stream()
                            .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.TORPEDOSALVO)
                            .filter(gameObject -> isEnemyTorpedo(gameObject) = TRUE)
                            .filter(gameObject -> getDistanceBetween(gameObject, this.bot) < 100 + this.bot.size)
                            .sorted(Comparator
                                    .comparing(gameObject -> getDistanceBetween(this.bot, gameObject)))
                            .collect(Collectors.toList())

    IF (
    this.bot.size > 50 AND
    enemyTorpedoes.size() > 1 AND
    shieldOn = 0 AND
    isAction = 0
    ) THEN
        playerAction.action ← PlayerActions.ACTIVATESHIELD
        isAction ← isAction + 1
        shieldOn ← shieldOn + 1
    END IF
```

2.1.3 Defensive Shoot

```
{Shoot enemy when enemy is bigger, defensive shooting}

    // is enemy bigger with margin, for defensive shooting
    LET isEnemyBigger ←     gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .filter(bot -> bot.size > this.bot.size*0.9)
                            .orELSE(NULL)

    // check IF there is an object that can block shooting (defensive)
    LET isObjectInFrontD ←  gameState.getGameObjects().stream()
                            .filter(gameObject -> isEnemyBigger NOT NULL)
                            .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.WORMHOLE OR
gameObject.gameObjectType = ObjectTypes.ASTEROIDFIELD OR gameObject.gameObjectType = ObjectTypes.GASCLOUD)
                            .filter(gameObject -> (isInSight(getHeadingBetween(isEnemyBigger), gameObject, 30)))
                            .collect(Collectors.toList())

    IF (
    isEnemyBigger NOT NULL AND
    getDistanceBetween(isEnemyBigger, this.bot) < 350 + this.bot.size + isEnemyBigger.size AND
    this.bot.size > 30 AND
    this.bot.torpedoSalvoCount > 0 AND
    teleWasShot = 0 AND
    isAction = 0
    ) THEN
        minDistance ← 1000
        IF (isObjectInFrontD.size() > 0) THEN
            FOR (i ← 0 : i < isObjectInFrontD.size()) DO
                IF (getDistanceBetween(isObjectInFrontD.get(i), this.bot) < minDistance) THEN
                    minDistance ← getDistanceBetween(isObjectInFrontD.get(i), this.bot)
```

```
                END IF
                i ← i + 1
            END FOR
        END IF
        // check IF there is an object that can block shooting
        IF (minDistance - this.bot.size < getDistanceBetween(isEnemyBigger, this.bot) - this.bot.size -
isEnemyBigger.size) THEN
            minDistance ← 1000
        END IF
        // high fire rate when enemy is close
        ELSE IF (getDistanceBetween(isEnemyBigger, this.bot) < 150 + this.bot.size + isEnemyBigger.size) THEN
            IF (closeFire = 0) THEN
                playerAction.heading ← getHeadingBetween(isEnemyBigger)
                playerAction.action ← PlayerActions.FIRETORPEDOES
                closeFire ← closeFire + 1
                isAction ← 1
            END IF
            ELSE IF (closeFire = 1) THEN
                closeFire ← 0
            END IF
        END IF
        // low fire rate when enemy is far
        ELSE IF (getDistanceBetween(isEnemyBigger, this.bot) <= 350 + this.bot.size + isEnemyBigger.size) THEN
            IF (longFire = 0) THEN
                playerAction.heading ← getHeadingBetween(isEnemyBigger)
                playerAction.action ← PlayerActions.FIRETORPEDOES
                longFire ← longFire + 1
                isAction ← 1
            END IF
            ELSE IF (
            longFire = 1 OR
            longFire = 2 OR
            longFire = 3
            ) THEN
                longFire ← longFire + 1
            END IF
            ELSE IF (longFire >= 4) THEN
                longFire ← 0
            END IF
        END IF
    END IF
```

2.1.4 Near Edge

```
{Go to center when edge is near}

    // is enemy bigger with margin, for running away
    LET isEnemyBigger2 ←    gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .filter(bot -> bot.size*1.1 > this.bot.size)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .orElse(NULL)

    IF (
    distanceFromWorldCenter*1.2 + (1.7*this.bot.size) > worldRadius AND
    isAction = 0
    ) THEN
        // if chased by enemy in the edge, escape
        IF (
        isEnemyBigger2 NOT NULL AND
        getDistanceBetween(isEnemyBigger2, this.bot) < 200 + this.bot.size + isEnemyBigger2.size
        ) THEN
            IF (squeezeL = 0) THEN
```

```
                playerAction.heading ← getHeadingBetween(isEnemyBigger2) + 90
                playerAction.action ← PlayerActions.FORWARD
                squeezeL ← squeezeL + 1
                isAction ← 1
            END IF
            ELSE IF (squeezeL = 1) THEN
                squeezeL ← 0
            END IF
        END IF
        // IF not chased, go to center
        ELSE
            IF (center = 0) THEN
                playerAction.heading ← getHeadingToCenter()
                playerAction.action ← PlayerActions.FORWARD
                center ← center + 1
                isAction ← 1
            END IF
            ELSE IF (center = 1) THEN
                center ← 0
            END IF
    END IF
```

2.1.5 Kabur

```
{Escape if enemy is bigger and close}

    // is enemy bigger with margin, for running away
    LET isEnemyBigger2 ←    gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .filter(bot -> bot.size*1.1 > this.bot.size)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .orElse(NULL);

    IF (
    isEnemyBigger2 NOT NULL AND
    getDistanceBetween(isEnemyBigger2, this.bot) < 200 + this.bot.size + bot.size AND
    isAction = 0
    ) THEN
        IF (runAway = 0) THEN
            playerAction.heading ← getHeadingBetween(isEnemyBigger2) + 180
            playerAction.action ← PlayerActions.FORWARD
            runAwayF ← runAwayF + 1
            runAway ← runAway + 1
            isAction ← 1
        END IF
        ELSE IF (runAway = 1) THEN
            runAwayF ← runAwayF + 1
            runAway ← 0
        END IF
    END IF
```

2.1.6 Kejar musuh

```
{Chase when enemy is smaller and close}

    // is enemy smaller with margin, for chasing and offensive shooting
    LET isEnemySmaller2 ←   gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .filter(bot -> bot.size*1.2 < this.bot.size)
                            .orElse(NULL)
```

```
    IF (
    isEnemySmaller2 NOT NULL AND
    getDistanceBetween(isEnemySmaller2, this.bot) < 100 + this.bot.size + bot.size AND
    teleWasShot = 0 AND
    isAction = 0
    ) THEN
        IF (chase = 0) THEN
            playerAction.heading ← getHeadingBetween(isEnemySmaller2)
            playerAction.action ← PlayerActions.FORWARD
            chase ← chase + 1
            runAwayF ← runAwayF + 1
            isAction ← 1
        END IF
        ELSE IF (chase = 1) THEN
            runAwayF ← runAwayF + 1
            chase ← 0
        END IF
    END IF
```

2.1.7 Teleport

```
{Shoot teleport when enemy is smaller and in range}

    // is enemy smaller with margin, and far, for teleporter
    LET isEnemySmaller3 ←   gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .filter(bot -> bot.size*1.1 + 40 < this.bot.size)
                            .orElse(NULL)

    IF (
    isEnemySmaller3 NOT NULL AND
    getDistanceBetween(isEnemySmaller3, this.bot) >= 150 + this.bot.size + bot.size AND
    getDistanceBetween(isEnemySmaller3, this.bot) < 500 + this.bot.size + bot.size AND
    this.bot.size > 100 AND
    this.bot.teleporterCount > 0 AND
    this.bot.shieldCount > 0 AND
    shieldOn = 0 AND
    teleWasShot = 0 AND
    isAction = 0
    ) THEN
        playerAction.heading ← getHeadingBetween(isEnemySmaller3)
        playerAction.action ← PlayerActions.FIRETELEPORT
        teleTick ← this.gameState.world.getCurrentTick()
        teleActiveIn ← (getDistanceBetween(isEnemySmaller3, this.bot)-(isEnemySmaller3.size*0.5)-this.bot.size)/20
        teleWasShot ← teleWasShot + 1
        activateShield ← 1
        isAction ← 1
    END IF

    // activate shield after shooting teleporter
    IF (
        activateShield >= 1 AND
        isAction = 0
    ) THEN
    playerAction.action ← PlayerActions.ACTIVATESHIELD
    activateShield ← 0
    shieldOn ← 1
    isAction ← 1
    END IF

    // is enemy smaller with no margin, for detonate
    LET isEnemySmaller ←    gameState.getPlayerGameObjects().stream()
```

```
                            .filter(bot -> bot.id NOT this.bot.id)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .filter(bot -> bot.size < this.bot.size)
                            .orElse(NULL)

    // detonate teleporter when the enemy is smaller
    IF (
    teleTick + teleActiveIn = currentTick AND
    isEnemySmaller NOT NULL AND
    teleWasShot >= 1 AND
    isAction = 0
    ) THEN
        playerAction.action ← PlayerActions.TELEPORT
        isAction ← 1
        teleWasShot ← 0
    END IF
    ELSE IF (
    teleTick + teleActiveIn = currentTick AND
    isEnemySmaller = NULL AND
    teleWasShot >= 1  AND
    isAction = 0
    ) THEN
        teleWasShot ← 0
    END IF
```

2.1.8 Offensive Shoot

```
{Shoot torpedo when enemy is smaller and close, offensive shooting}

    // is enemy smaller with margin, for chasing and offensive shooting
    LET isEnemySmaller2 ←   gameState.getPlayerGameObjects().stream()
                            .filter(bot -> bot.id NOT this.bot.id)
                            .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                            .filter(bot -> bot.size*1.2 < this.bot.size)
                            .orElse(NULL)

    // check if there is an object that can block shooting (offensive)
    LET isObjectInFrontO ←  gameState.getGameObjects().stream()
                            .filter(gameObject -> isEnemySmaller2 NOT NULL)
                            .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.WORMHOLE OR
gameObject.gameObjectType = ObjectTypes.ASTEROIDFIELD OR gameObject.gameObjectType = ObjectTypes.GASCLOUD)
                            .filter(gameObject -> (isInSight(getHeadingBetween(isEnemySmaller2), gameObject, 30)))
                            .collect(Collectors.toList());

    IF (
    isEnemySmaller2 NOT NULL AND
    getDistanceBetween(isEnemySmaller2, this.bot) < 150 + this.bot.size + bot.size AND
    this.bot.torpedoSalvoCount > 0 AND
    this.bot.size > 50 AND
    teleWasShot = 0 AND
    isAction = 0
    ) THEN
        double minDistance ← 1000
        IF (isObjectInFrontO.size() > 0) THEN
            FOR (i ← 0 : i < isObjectInFrontO.size()) DO
                IF (getDistanceBetween(isObjectInFrontO.get(i), this.bot) < minDistance) THEN
                    minDistance ← getDistanceBetween(isObjectInFrontO.get(i), this.bot);
                END IF
                i ← i + 1
            END FOR
        END IF
        // check if there is an object that can block shooting
```

```
        IF (minDistance - this.bot.size < getDistanceBetween(isEnemySmaller2, this.bot) - this.bot.size -
isEnemySmaller2.size) THEN
            minDistance ← 1000
        END IF
        ELSE
            IF (attack = 0) THEN
            playerAction.heading ← getHeadingBetween(isEnemySmaller2)
            playerAction.action ← PlayerActions.FIRETORPEDOES
            attack ← attack + 1
            isAction ← 1
            END IF
            ELSE IF (attack = 1) THEN
                attack ← 0
            END IF
        END IF
    END IF
```

2.1.9 Get Food ketika sedang dikejar

```
{Get food while being chased}

    IF (
    closestFood NOT NULL AND
    runAwayF > 0 AND
    isAction = 0
    ) THEN
        playerAction.action ← PlayerActions.FORWARD
        runAwayF ← 0
        isAction ← 1
    END IF
```

2.1.10 Bergerak ke area terbaik

```
{Finding best area to move to}

    // get food heading based on scanning
    var scanFood ←  gameState.getGameObjects().stream()
                    .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.FOOD OR gameObject.gameObjectType
= ObjectTypes.SUPERFOOD)
                    .filter(gameObject -> (isInSight(headingPref, gameObject, 30)))
                    .collect(Collectors.toList());

    IF (
    (scanFood NOT NULL AND
    this.gameState.world.getCurrentTick() < 600 AND
    prevsize < this.bot.size AND
    runAwayF = 0 AND
    isAction = 0)
    ) THEN
        minDistanceF ← 1000
        index ← 0
        IF (scanFood.size() > 0) THEN
            FOR (i ← 0 : i < scanFood.size()) DO
                IF (getDistanceBetween(scanFood.get(i), this.bot) < minDistanceF) THEN
                    minDistanceF ← getDistanceBetween(scanFood.get(i), this.bot)
                    index ← i
                END IF
                i ← i + 1
            END FOR
            LET closestScanFood ← scanFood.get(index)
            playerAction.heading ← getHeadingBetween(closestScanFood)
            playerAction.action ← PlayerActions.FORWARD
            minDistanceF ← 1000
```

```
            isAction ← 1
        END IF
    END IF
```

2.1.11 Jika sudah tidak ada food, fokus tembak lawan

```
{Kamikaze when there is no more food}

    // closest enemy var
    LET closestEnemy ←  gameState.getPlayerGameObjects().stream()
                        .filter(bot -> bot.id NOT this.bot.id)
                        .min(Comparator.comparing(bot -> getDistanceBetween(bot, this.bot) - bot.size -
this.bot.size))
                        .orElse(NULL)

    // check if there is an object that can block shooting (kamikaze)
    LET isObjectInFrontK ←  gameState.getGameObjects().stream()
                            .filter(gameObject -> closestEnemy NOT null)
                            .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.WORMHOLE OR
gameObject.gameObjectType = ObjectTypes.ASTEROIDFIELD OR gameObject.gameObjectType = ObjectTypes.GASCLOUD)
                            .filter(gameObject -> (isInSight(getHeadingBetween(closestEnemy), gameObject, 30)))
                            .collect(Collectors.toList())

    IF (
    closestFood = NULL AND
    isAction = 0
    ) THEN
        minDistance ← 1000
        IF (isObjectInFrontK.size() > 0) THEN
            FOR (i ← 0 : i < isObjectInFrontK.size()) DO
                IF (getDistanceBetween(isObjectInFrontK.get(i), this.bot) < minDistance) THEN
                    minDistance ← getDistanceBetween(isObjectInFrontK.get(i), this.bot)
                END IF
                i ← i + 1
            END FOR
        END IF
        // check if there is an object that can block shooting
        IF (minDistance - this.bot.size < getDistanceBetween(closestEnemy, this.bot) - this.bot.size -
closestEnemy.size) THEN
            minDistance ← 1000
        END IF
        ELSE
            playerAction.heading ← getHeadingBetween(closestEnemy)
            playerAction.action ← PlayerActions.FIRETORPEDOES
            isAction ← 1
    END IF
```

2.1.12 Food untuk mencegah bug

```
{Default food prevent bug under 5 ticks}

    // closest food
    LET closestFood ←   gameState.getGameObjects().stream()
                        .filter(gameObject -> gameObject.gameObjectType = ObjectTypes.FOOD OR
gameObject.gameObjectType = ObjectTypes.SUPERFOOD)
                        .min(Comparator.comparing(gameObject -> getDistanceBetween(gameObject, this.bot)))
                        .orElse(NULL)

    IF (
    currentTick < 5 AND
    prevsize > this.bot.size AND
    closestFood NOT NULL
    ) THEN
```

```
        playerAction.heading ← getHeadingBetween(closestFood)
        playerAction.action ← PlayerActions.FORWARD
        isAction ← 1
    END IF


{Default action}
    IF (isAction = 0) THEN
        playerAction.action ← PlayerActions.FORWARD
        isAction ← 1
    END IF
```

2.2 Fungsi jarak pandang bot

```
// function if object is in sight or no
    FUNCTION isInSight(heading : integer, gameObject : GameObject, vision : integer) → boolean
        vision RETURN vision / 2
        IF (heading >= vision AND heading <= 360 - vision) THEN
            IF (getHeadingBetween(gameObject) >= (heading - vision)%360 AND getHeadingBetween(gameObject) <=
(heading + vision)%360) THEN
                RETURN TRUE
            END IF
        END IF
        ELSE IF (heading < vision) THEN
            IF ((getHeadingBetween(gameObject) <= (heading + vision) AND getHeadingBetween(gameObject) >= 0) OR
(360 - vision + heading <= getHeadingBetween(gameObject) AND getHeadingBetween(gameObject) <= 360)) THEN
                RETURN TRUE
            END IF
        END IF
        ELSE IF (heading > 360 - vision) THEN
            IF ((getHeadingBetween(gameObject) >= (heading - vision) AND getHeadingBetween(gameObject) <= 360) OR
(360 - vision + heading <= getHeadingBetween(gameObject) AND getHeadingBetween(gameObject) <= 360)) THEN
                RETURN TRUE
            END IF
        END IF
        RETURN FALSE;
```

2.3 Fungsi mendetect peluru musuh

```
// detect if torpedo is heading towards us
FUNCTION isEnemyTorpedo(gameObject) -> boolean
    // Get our position
    LET x ← this.bot.getPosition().x
    LET y ← this.bot.getPosition().y

    // Get torpedo position
    LET x2 ← gameObject.getPosition().x
    LET y2 ← gameObject.getPosition().y

    // Get torpedo heading
    LET heading ← gameObject.currentHeading

    // Check IF the torpedo is heading towards us
    IF (x2 > x AND y2 > y) THEN
        IF (heading >= 180 AND heading <= 270) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 < x AND y2 > y) THEN
        IF (heading >= 270 AND heading <= 360) THEN
            RETURN TRUE
        END IF
```

```
    ELSE IF (x2 < x AND y2 < y) THEN
        IF (heading >= 0 AND heading <= 90) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 > x AND y2 < y) THEN
        IF (heading >= 90 AND heading <= 180) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 > x AND y2 = y) THEN
        IF (heading = 180) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 < x AND y2 = y) THEN
        IF (heading = 0) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 = x AND y2 > y) THEN
        IF (heading = 270) THEN
            RETURN TRUE
        END IF
    ELSE IF (x2 = x AND y2 < y) THEN
        IF (heading = 90) THEN
            RETURN TRUE
        END IF
    END IF

    RETURN FALSE
```

2.4 Fungsi scan area

```
// scan list of object
FUNCTION scanObject() -> integer
    vision ← 30
    List<Double> scoringList ← new ArrayList<Double>()
    FOR (i ← 0 : i < 360/vision) DO
        headingStart ← i * vision
        headingEnd ← headingStart + vision
        scoreSum ← 0

        //collect food list
        List<GameObject> foodList ← gameState.getGameObjects()
                                    .stream().filter(item -> item.getGameObjectType() = ObjectTypes.FOOD)
                                    .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                    .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                    .sorted(Comparator
                                            .comparing(item -> getDistanceBetween(this.bot, item)))
                                    .collect(Collectors.toList())
        FOR (j ← 0 : j < foodList.size()) DO
            scoreSum ← scoreSum + 20 / getDistanceBetween(this.bot, foodList.get(j))
            j ← j + 1
        END FOR

        //collect superfood list
        List<GameObject> superfoodList ←    gameState.getGameObjects()
                                            .stream().filter(item -> item.getGameObjectType() =
ObjectTypes.SUPERFOOD)
                                            .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                            .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                            .sorted(Comparator
                                                    .comparing(item -> getDistanceBetween(this.bot, item)))
```

```
                                        .collect(Collectors.toList())
        FOR (int j = 0 : j < superfoodList.size()) DO
            scoreSum ← scoreSum + 21 / getDistanceBetween(this.bot, superfoodList.get(j))
            j ← j + 1
        END FOR

        //collect obstacles list
        List<GameObject> gasCloudList ← gameState.getGameObjects()
                                        .stream().filter(item -> item.getGameObjectType() = ObjectTypes.GASCLOUD OR
item.getGameObjectType() = ObjectTypes.SUPERNOVABOMB)
                                        .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                        .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                        .sorted(Comparator
                                                .comparing(item -> getDistanceBetween(this.bot, item)))
                                        .collect(Collectors.toList())
        FOR (int j = 0 : j < gasCloudList.size()) DO
            scoreSum ← scoreSum - 10 / getDistanceBetween(this.bot, gasCloudList.get(j))
            j ← j + 1
        END FOR

        //collect asteroid list
        List<GameObject> asteroidList ← gameState.getGameObjects()
                                        .stream().filter(item -> item.getGameObjectType() =
ObjectTypes.ASTEROIDFIELD || item -> item.getGameObjectType() = ObjectTypes.TORPEDOSALVO)
                                        .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                        .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                        .sorted(Comparator
                                                .comparing(item -> getDistanceBetween(this.bot, item)))
                                        .collect(Collectors.toList())
        for (j ← 0 : j < asteroidList.size()) DO
            scoreSum ← scoreSum - 5 / getDistanceBetween(this.bot, asteroidList.get(j))
            j ← j + 1
        END FOR

        //collect wormhole list
        List<GameObject> wormholeList ← gameState.getGameObjects()
                                        .stream().filter(item -> item.getGameObjectType() = ObjectTypes.WORMHOLE)
                                        .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                        .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                        .sorted(Comparator
                                                .comparing(item -> getDistanceBetween(this.bot, item)))
                                        .collect(Collectors.toList())
        FOR (j ← 0 : j < wormholeList.size()) DO
            scoreSum ← scoreSum - 10 / getDistanceBetween(this.bot, wormholeList.get(j))
            j ← j + 1
        END FOR

        // collect bigger enemy list
        List<GameObject> enemyBiggerList ←  gameState.getGameObjects()
                                            .stream().filter(item -> item.getGameObjectType() = ObjectTypes.PLAYER)
                                            .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + item.size)
                                            .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                            .sorted(Comparator
                                                    .comparing(item -> getDistanceBetween(this.bot, item)))
                                            .collect(Collectors.toList())
        FOR (j ← 0 : j < enemyBiggerList.size()) DO
            scoreSum ← scoreSum - 20 / getDistanceBetween(this.bot, enemyBiggerList.get(j))
```

```
            j ← j + 1
        END FOR

        // collect smaller enemy list
        List<GameObject> enemySmallerList ← gameState.getGameObjects()
                                            .stream().filter(item -> item.getGameObjectType() = ObjectTypes.PLAYER)
                                            .filter(item -> getDistanceBetween(this.bot, item) <
(this.gameState.world.radius*0.5) + this.bot.size + bot.size)
                                            .filter(item -> getHeadingBetween(item) >= headingStart AND
getHeadingBetween(item) <= headingEnd)
                                            .filter(item -> item.size < this.bot.size)
                                            .sorted(Comparator
                                                    .comparing(item -> getDistanceBetween(this.bot, item)))
                                            .collect(Collectors.toList());
        FOR (j ← 0 : j < enemySmallerList.size()) DO
            scoreSum ← scoreSum - 10 / getDistanceBetween(this.bot, enemySmallerList.get(j))
            j ← j + 1
        END FOR

        scoringList.add(scoreSum);
        i ← i + 1
    END FOR

    //calculate the best direction
    maxScore ← 0
    maxScoreIndex ← 0
    FOR (i ← 0 : i < scoringList.size()) DO
        IF (scoringList.get(i) > maxScore) THEN
            maxScore ← scoringList.get(i)
            maxScoreIndex ← i
        END IF
        i ← i + 1
    END FOR
    RETURN (maxScoreIndex+1) * vision - (vision/2)
```

B. Penjelasan Struktur Data yang Digunakan dalam Program bot Galaxio

Struktur data pada permainan galaxio diimplementasikan dalam beberapa *class*. *Class* tersebut dibagi menjadi 4 kategori utama, yaitu *Enums* yang berisi konstan – konstan objek yang merupakan bagian dari permainan serta konstan - konstan aksi yang menjadi daftar gerakan untuk pemain, *Models* yang berisi informasi keseluruhan permainan mulai dari *game state* berisi informasi permainan setiap *tick*-nya, beserta mengakses setiap *enumerates* yang ada seperti fungsi - fungsi pemilihan aksi dilakukan oleh player, fungsi mendapatkan data player seperti ID dan atribut lainnya, serta mengakses data ataupun pengukuran objek dengan objek lain ataupun dengan player. *Services* yang berisi algoritma dinamika yang digunakan dalam pemilihan keputusan *bot* selama permainan berlangsung, dan *Main.java* yang mengatur semuanya mulai dari membaca permainan dan menjalankan *service bot*.

Berikut pemaparan berbagai struktur kelas yang digunakan dalam bot *Galaxio*:

1. Kategori *Services*

   Kelas *BotService* dalam kategori ini berperan sebagai strategi utama dalam permainan *Galaxio*, yaitu strategi permainan yang dilakukan bot sendiri melalui implementasi kombinasi aksi *Player* serta pemanfaatan informasi *Objects* yang ada.

Atribut	Deskripsi
public class BotService	Kelas utama yang memegang atribut serta metode dari *BotService* itu sendiri.
private GameObject bot;	Atribut penanda tipe dari bot itu sendiri sebagai objek dari permainan
private PlayerAction playerAction	Atribut aksi yang dilakukan bot
Private Gamestate gameState	Atribut state game yang dapat digunakan oleh bot
static int currentTick	Atribut statis berisi tick game pada saat itu juga
static int shieldOn	Atribut statis sebagai penanda shield sedang aktif sehingga tidak dapat memanggil fungsi shield lagi
static int prevsize	Atribut statis berisi data besar bot (bot size) pada tick sebelumnya
static int tickX	Atribut statis berisi data tick sebelumnya (currentTick - 1)
static int activateShield	Atribut statis sebagai penanda untuk menyalakan shield, dipakai pada saat menembak peluru teleport
static int teleTick	Atribut statis berisi data tick pada saat teleport ditembakkan
static int teleActiveIn	Atribut statis berisi data berapa tick yang dibutuhkan agar peluru teleport sampai ke target
static int teleWasShot	Atribut statis sebagai penanda bahwa peluru teleport sudah ditembakkan, agar bot tidak melakukan command yang membuat size nya mengecil
static int closeFire	Atribut statis yang berguna untuk mengatur fire rate dari bot, untuk tembak jarak dekat fire ratenya tinggi
static int longFire	Atribut statis yang berguna untuk mengatur fire rate dari bot, untuk tembak jarak jauh fire ratenya rendah
static int center	Atribut statis yang berguna agar bot dapat

	melakukan multitasking pada saat dekat dengan edge
static int runAway	Atribut statis yang berguna agar bot dapat melakukan multitasking pada saat dikejar musuh
static int chase	Atribut statis yang berguna agar bot dapat melakukan multitasking pada saat mengejar musuh
static int attack	Atribut statis yang berguna agar bot dapat melakukan multitasking pada saat menembak musuh
static int squeeze	Atribut statis yang berguna agar bot dapat melakukan multitasking pada saat terjepit dengan musuh dan edge
static int runAwayF	Atribut statis sebagai penanda untuk masuk ke conditional mencari makan saat dikejar
static int fixDetonate	Atribut statis sebagai penanda untuk masuk ke conditional detonate 2 kali, karena seringkali fungsi detonate tidak terpanggil
static int supernovaTick	Atribut statis berisi data tick pada saat supernova ditembakkan
static int supernovaActiveIn	Atribut statis berisi data berapa tick yang dibutuhkan agar peluru supernova sampai ke target
static int supernovaWasShot	Atribut statis sebagai penanda bahwa peluru supernova sudah ditembakkan

Metode	**Deskripsi**
public BotService()	Konstruktor metode pembuatan bot
public GameObject getBot()	Fungsi yang mengembalikan atribut dari bot
public void setBot(GameObject bot)	Prosedur yang mengubah atribut dari bot
public PlayerAction getPlayerAction()	Fungsi yang mengembalikan aksi pemain
public void setPlayerAction(PlayerAction playerAction)	Prosedur yang dapat mengubah action pemain

public void computeNextPlayerAction(PlayerAction playerAction)	Prosedur yang menentukan action apa yang harus dilakukan bot selanjutnya
public boolean isInSight(int heading, GameObject gameObject, int vision)	Fungsi yang menentukan apakah suatu objek berada di pandangan kita, mengembalikan true atau false
public boolean isEnemyTorpedo(GameObject gameObject)	Fungsi yang menentukan apakah suatu torpedo merupakan torpedo musuh yang sedang mengarah ke kita, mengembalikan true atau false
public GameState getGameState()	Fungsi yang mengembalikan game state pada tick sekarang
public void setGameState(GameState gameState)	Prosedur yang dapat mengubah game state
private void updateSelfState()	Prosedur yang dapat mengupdate self state
private double getDistanceBetween(GameObject object1, GameObject object2)	Fungsi yang mengembalikan jarak antara objek 1 dengan objek 2
private int getHeadingBetween(GameObject otherObject)	Fungsi yang mengembalikan arah heading menuju ke objek
private int getHeadingToCenter()	Fungsi yang mengembalikan arah heading menuju ke titik pusat game
private int toDegrees(double v)	Fungsi yang mengembalikan arah derajat
private int scanObject()	Fungsu yang digunakan untuk melakukan scanning dan *scoring system, mengembalikan arah heading terbaik*

2. Kategori *Enums*
   Kategori enums berisi kelas - kelas yang menyimpan anggota konstanta berupa nilai integer yang melambangkan akses setiap aksi pemain serta konstanta nilai akses suatu jenis objek.
   I. ObjectTypes.java

Metode	Deskripsi
public enum ObjectTypes	Kelas utama enums yang memiliki informasi tipe objek dalam *value* penomoran nilai integer
ObjectTypes(Integer value)	Constructor nilai *ObjectType* dengan nilai

	value.
Public static ObjectTypes valueOf(Integer value)	Digunakan sebagai fungsi memberikan nilai tipe objek

II. PlayerActions.java

Metode	Deskripsi
Public enum PlayerActions	Kelas utama *PlayerActions* yang memiliki informasi setiap jenis aksi yang dapat dilakukan pemain dalam *value* penomoran nilai integer
Private PlayerActions(Integer value)	Constructor nilai *PlayerACtions* dengan nilai value.

3. Kategori Models
   Kategori Models berisi kelas - kelas pemodelan dari permainan itu sendiri mulai dari *state* permainan yaitu kondisi permainan pada setiap saatnya, spesifikasi atribut serta metode untuk mendapatkan atribut dari objek game dan aksi pemain, serta model dunia area permainan dari segi radius, pemetaan x dan y, dan *tick* permainan

I. GameObject.java

Atribut	Deskripsi
public class GameObject	Kelas utama objek permainan yang memiliki segala informasi atrbut dan metod dari setiap objek permainan yang ada.
public UUID id	ID sebuah objek
public Integer size	Ukuran sebuah objek
public Integer currentHeading	Arah menghadap objek
public Position position	Posisi objek
public ObjectTypes gameObjectType	Tipe objek
public Integer effects	Efek yang sedang dimiliki objek
public Integer torpedoSalvoCount	Jumlah amunisi torpedo objek (hanya dimiliki objek *player,* jika bukan *player* maka bersifat *null*)

public Integer supernovaAvailable	Menyatakan ada atau tidaknya satu amunisi supernova (hanya dimiliki objek *player,* jika bukan *player* maka bersifat *null*)
public Integer teleporterCount	Menyatakan jumlah amunisi penembakkan *teleporter*(hanya dimiliki objek *player,* jika bukan *player* maka bersifat *null*)
public Integer shieldCount	Menyatakan jumlah *shield* pemain(hanya dimiliki objek *player,* jika bukan *player* maka bersifat *null*)

Metode	**Deskripsi**
Public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, Integer effects, Integer torpedoSalvoCount, Integer supernovaAvailable, Integer teleporterCount, Integer shieldCount)	Constructor pembentukkan sebuah objek game.
public UUID id	Mengembalikan nilai id objek
public void setId(UUID id)	Mengubah nilai id objek
public int getSize()	Mengembalikan nilai ukuran objek
public void setSize(int size)	Mengubah nilai ukuran objek
public int getSpeed()	Mengembalikan nilai kecepatan objek
public void setSpeed(int speed)	Mengubah nilai kecepatan objek
public Position getPosition()	Mengembalikan nilai posisi objek
public void setPosition(Position position)	Mengubah nilai posisi objek
public ObjectTypes getGameObjectType()	Mengembalikan tipe objek
public void setGameObjectType(ObjectTypes gameObjectType)	Mengubah tipe objek

public static GameObject FromStateList(UUID id, List<Integer> stateList)	Mengembalikan nilai seluruh atribut suatu objek

II. GameState.java

Atribut	Deskripsi
public World world	Merupakan atribut data dari medan permainan itu sendiri.
public List<GameObject> gameObjects	Atribut yang menyimpan segala objek yang ada dalam permainan pada *state* tersebut.
public List<GamObject> playerGameObjects	Atribut yang menyimpan segala tipe objek pemain beserta objek yang dimiliki pemain pada *state* tersebut.

Metode	Deskripsi
public GameState()	Konstruktor untuk membentuk gamestate
public GameState(World world, List<GameObject> gameObjects, List<GameObject playerGameObjects)	Konstruktor dengan parameter
public World getWorld()	Mengembalikan nilai *world* (atribut dunia)
public void setWorld(World world)	Mengubah nilai *world*
public List<GameObject> getGameObjects()	Mengembalikan daftar objek permainan
public void setGameObjects(List<GameObject> gameObjects)	Mengubah daftar objek permainan
public List<GameObject> getPlayerGameObjects()	Mengembalikan daftar objek *player*
Public void setPlayerGameObjects(List<GameObject > playerGameObjects)	Mengubah daftar objek *player*

III. GameStateDto.java

Atribut	Deskripsi
private World world	Merupakan atribut data dari medan permainan itu sendiri.
private Map<String, List<Integer>> gameObjects	Atribut yang menyimpan segala objek yang ada dalam permainan pada *state* tersebut.
private Map<String, List<Integer>> playerObjects	Atribut yang menyimpan segala tipe objek pemain beserta objek yang dimiliki pemain pada *state* tersebut.

Metode	Deskripsi
public Models.World getWorld()	Fungsi yang mengembalikan suatu world
public void setWorld(Models.World world)	Prosedur yang berfungsi mengatur suatu world
public Map<String, List<Integer>> getGameObjects()	Fungsi yang mengembalikan list berisi objek-objek dari suatu peta
public void setGameObjects(Map<String, List<INteger>> gameObjects	Prosedur yang berfungsi mengatur objek-objek dari suatu peta
public Map<String, List<Integer>>	Fungsi yang mengembalikan list berisi objek-objek dari suatu pemain
public void setPlayerObjects(Map<String, List<Integer>> playerObjects)	Prosedur yang berfungsi mengatur objek-objek dari suatu pemain

IV. PlayerAction.java

Atribut	Deskripsi
public class PlayerAction	Class yang berisi tentang aksi pemain
public UUID playerId	Atribut yang berisi ID suatu pemain
public PlayerActions action	Atribut yang berisi aksi suatu pemain

public int heading	Atribut yang berisi arah heading suatu pemain

Metode	Deskripsi
public UUID getPlayerId()	Fungsi yang mengembalikan ID suatu pemain
public void setPlayerId(UUID playerId)	Prosedur yang berfungsi mengatur ID suatu pemain
public PlayerActions getAction()	Fungsi yang mengembalikan aksi suatu pemain
public void setAction(PlayerActions action)	Prosedur yang berfungsi mengatur aksi suatu pemain
public int getHeading()	Fungsi yang mengembalikan arah heading suatu pemain
public void setHeading(int heading)	Prosedur yang berfungsi mengatur arah heading suatu pemain

V. Position.java

Atribut	Deskripsi
public class Position	Class yang berisi tentang posisi suatu objek
public Position()	Konstruktor
public Position(int x, int y)	Konstruktor dengan parameter
public int getX()	Fungsi yang mengembalikan posisi X suatu objek
public void setX(int x)	Prosedur yang berfungsi mengatur posisi X suatu objek
public int getY()	Fungsi yang mengembalikan posisi Y suatu objek
Public void setY(int y)	Prosedur yang berfungsi mengatur posisi Y suatu objek

VI. World.java

Atribut	Deskripsi
public class World	Class yang berisi tentang info world
public Integer radius	Atribut yang berisi radius suatu world
public Integer currentTick	Atribut yang berisi tick sekarang dari suatu world

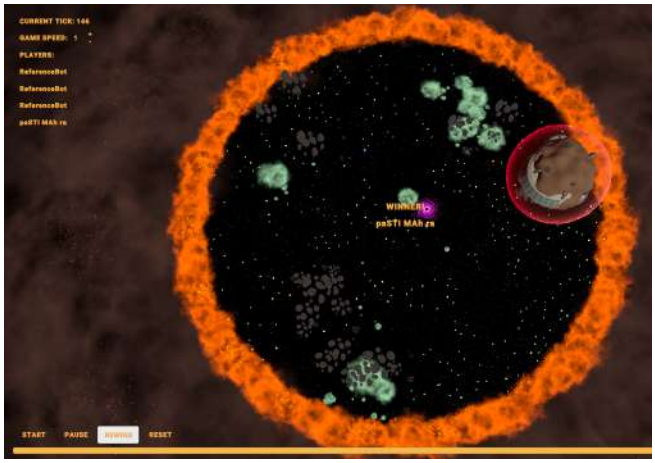
Metode	Deskripsi
public Position getCenterPoint()	Fungsi yang dapat mengembalikan titik tengah suatu map
public void setCenterPoint(Position centerPoint)	Prosedur yang dapat digunakan untuk mengatur titik tengah suatu map
public Integer getRadius()	Fungsi yang dapat mengembalikan radius suatu map
public void setRadius(Integer radius)	Prosedur yang dapat digunakan untuk mengatur radius map
public Integer getCurrentTick()	Fungsi yang dapat mengembalikan tick pada saat ini
public void setCurrentTick(Integer currentTick)	Prosedur yang digunakan untuk mengatur tick sekarang

C. Analisis dari Desain Solusi Algoritma Greedy yang Diimplementasikan pada Setiap Pengujian yang Dilakukan

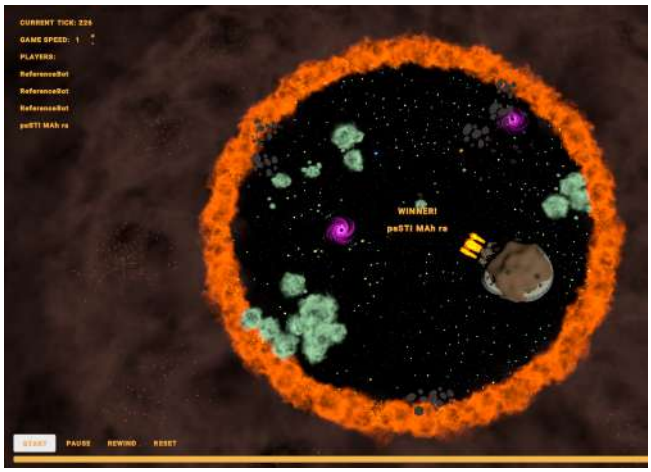
Pengujian bot dilakukan dengan melawankan bot kami, paSTI MAh rank 1, dengan reference bot. Bot ini merupakan bot default yang diberikan oleh entelect challenge, walaupun logika bot ini masih sangat buruk, bahkan beberapa command tidak digunakan sama sekali, sehingga terkadang tidak bisa melakukan pengujian untuk kasus-kasus tertentu.

Permainan Galaxio ini memiliki banyak faktor random dan kehokian, sehingga perlu dilakukan banyak pengujian untuk mendapatkan hasil yang sesuai. Diputuskan untuk melakukan pengujian sebanyak 3 kali, dan digunakan visualizer untuk memvisualisasikan hasil yang didapat. Berikut hasil akhir match match tersebut:

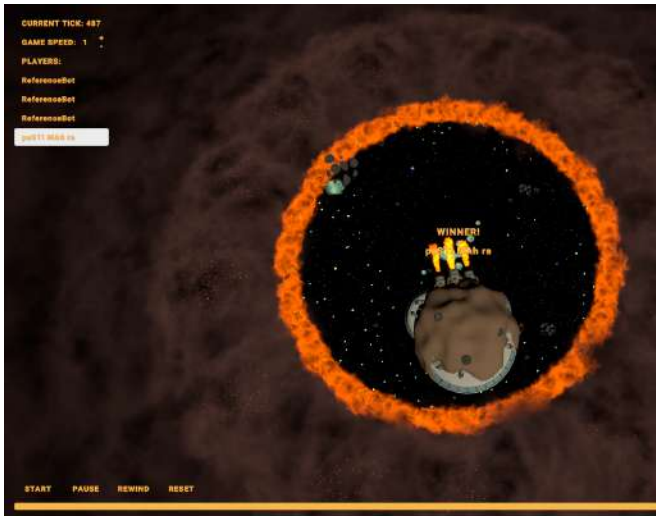
Percobaan pertama:



Percobaan kedua:



Percobaan ketiga:



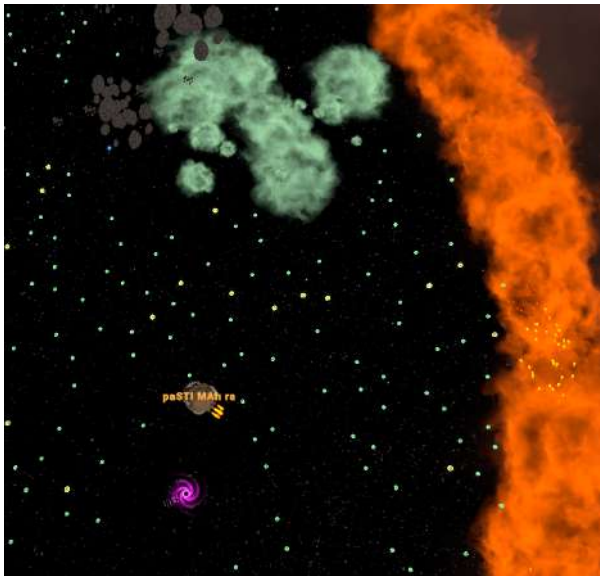
Dari ketiga percobaan tersebut, diperoleh hasil 3 kali menang oleh bot kami. Dari sebagian besar hasil percobaan, bot kami berhasil memenangkan permainan dengan durasi yang cukup lama. Hal ini disebabkan karena kami menerapkan strategi *defensive*, yaitu mencoba mencari area makanan yang jauh dari halangan dan musuh. Sehingga bot kami mengurangi peluang mati di awal dengan mencegah serangan musuh berperang dengan musuh. Pada akhirnya, bot kami diharapkan menjadi bot paling besar di game tersebut dan dengan mudah memenangkan permainan tersebut.

Untuk menunjukkan bahwa algoritma kita dapat berfungsi dengan baik dan benar, kami menggunakan 4 bot hasil implementasi algoritma kami untuk diadu, karena dengan menggunakan reference bot sebagai lawan hanya terdapat algoritma untuk makan dan mendeteksi musuh. Berikut adalah hasil implementasinya:

1. Defensive shooting

Seperti yang terlihat pada gambar, bot yang ada di posisi atas menembak musuh yang lebih besar dari besar badannya. Hal tersebut sangat krusial karena menurut kami pertahanan terbaik adalah menyerang. Terlebih lagi jika peluru bot mengenai musuh, ukuran badan musuh berkurang dan ukuran bot bertambah besar. Tentunya ini dapat menjadi titik balik agar bot dapat membalikan keadaan sehingga state bot selanjutnya adalah menyerang, bukan bertahan dan kabur lagi.

2. Scan

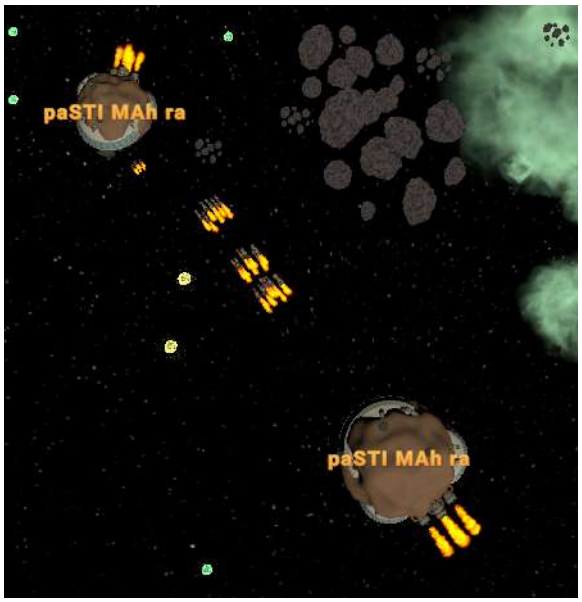


Selanjutnya adalah fitur scan yang menjadi tulang punggung keberjalanan bot kami. Tentunya, untuk memenangkan pertandingan kita membutuhkan area terbaik agar ukuran tubuh bot bertambah dan tidak berkurang. Maka dari itu kami menciptakan fitur scan yang berguna untuk mendeteksi area yang aman untuk bot kunjungi, seperti contohnya pada gambar di atas. Bot lebih memilih berjalan ke area yang penuh makanan daripada berjalan ke area yang memiliki *obstacle* seperti *gas cloud* dan *wormhole*.

3. Shield

Fitur shield ini juga tidak kalah penting bagi sebuah bot. Berdasarkan aturan dari permainannya, peluru *torpedo* yang ditembakkan ke musuh dapat mengurangi ukuran tubuh sebanyak 10 satuan dalam permainan yang artinya sangat menentukan kemenangan sebuah bot. Dengan adanya shield, bot dapat meminimalisir adanya pengurangan ukuran tubuh ketika ditembaki. Tidak hanya itu, hal ini juga dapat menjadi titik balik sebuah bot karena peluru yang musuh tembaki menjadi sia-sia dan hanya membuang-buang ukuran tubuh musuh saja.

4. Menembak dan mengejar musuh yang lebih kecil



Bot kami juga memiliki fitur untuk menembaki musuh yang berada dalam range dan ukuran tubuhnya lebih kecil dari bot. Selain menembaki, bot kami sekaligus mengejar musuh dan diharapkan musuh bisa tereliminasi dari permainan. Namun pada kondisi tertentu, fitur ini malah bisa menjadi bumerang untuk bot jika musuh memiliki keberuntungan yang lebih tinggi, misalnya ketika shieldnya aktif, atau bahkan memakan musuh sehingga tubuhnya bertambah besar secara tiba-tiba, dan bot sedang berada dalam posisi mengejar.

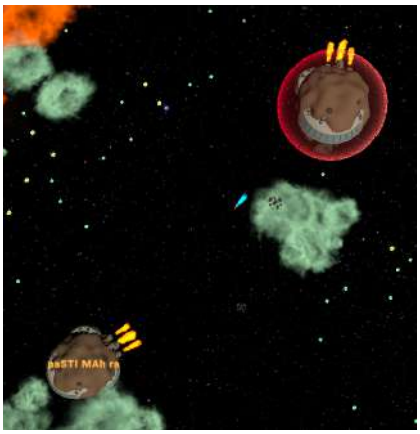
5. Kabur

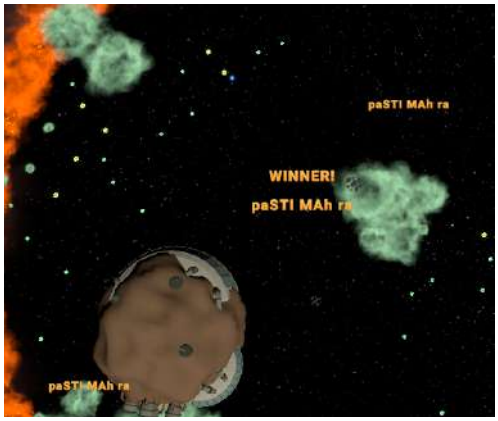
Seperti yang terlihat pada gambar, bot yang berada di posisi atas sedang berusaha untuk kabur karena sedang dikejar oleh musuh di bawah. Tentunya fitur ini sangat berguna agar bot tidak dimakan dan tereliminasi secara sia-sia oleh musuh.

6. Hindarin edge

Berdasarkan gambar tersebut, terlihat bahwa bot sedang melakukan aksi FORWARD dengan arah hadap berlawanan dari arah menghadap *edge* atau pinggiran dari medan perang, yaitu 180 derajat dari arah menghadap *edge*. Fitur ini sangat berguna apabila bot sedang mendekati ujung dari medan permainan dan mencoba menjauh dari *edge* supaya tidak mati dan hal tersebut juga berguna untuk bot dapat bertahan hingga akhir permainan ataupun permainan yang cukup lama sehingga pemetaan medannya sudah sangat kecil.

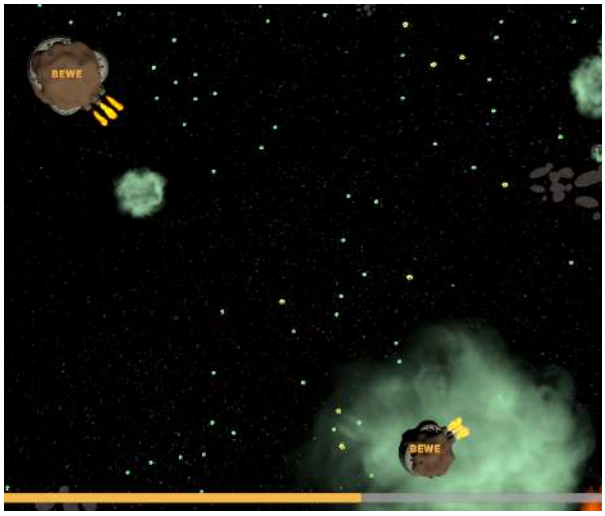
7. Teleport





Hasil *testing* kami ini menunjukkan *fire teleport* menjadi strategi penyerangan yang paling efektif dalam menyerang musuh dan mendapatkan penambahan ukuran yang cukup drastis pada umumnya. Dari gambar tersebut, ditunjukkan bahwa pertama, bot kami menembakkan fire teleport ke musuh yang cukup jauh dan lebih kecil, kemudian tepat setelah penembakkan, bot kami mengaktifkan *shield* guna mencegah marabahaya yang dapat datang seperti penembakkan *torpedo* oleh bot musuh yang bisa membuat ukuran kita lebih kecil dan berpotensi membatalkan penembakkan *teleport*.

8. Supernova



Dalam pemetaan tersebut, salah satu penyerangan yang cukup agresif dan baik dalam implementasi kami adalah penembakkan *supernova*. Penembakkan *supernova* ini hanya terjadi apabila *bot* mendapatkan sebuah supernovaAvailable ato satu amunisi *supernova* yang dalam kasus ini didapatkan dengan bot kami melewati posisi (0,0) atau titik tengah dari dunia

permainan. Hasil dari penembakkan dan detonate ini meledakkan sebuah bom berupa *gas cloud* yang mengecilkan musuh dalam kecepatan yang cukup cepat. Algoritma ini selalu efektif walaupun tingkat persentase kejadiannya sangat jarang.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Pada tugas besar strategi algoritma 1, kami berhasil membuat bot pada permainan Galaxio dengan menggunakan pendekatan algoritma greedy. Dapat dilihat, algoritma greedy tetap memerlukan strategi yang dinamis atau heuristik karena greedy murni yang selalu mengejar optimum lokal pada kasus - kasus tertentu dapat sangat terbatas dalam permainan Galaxio sehingga memiliki kemungkinan menang yang kecil karena sebuah permainan memperhatikan lebih dari satu aspek.

Maka dari itu, algoritma greedy utama yang diimplementasikan pada bot merupakan penggabungan dari sejumlah alternatif solusi greedy sehingga bot dapat mempunyai algoritma terbaik dan optimal untuk memenangkan permainan Galaxio.

B. Saran

Setelah menyelesaikan tugas besar pertama ini, kami memiliki beberapa saran untuk kelompok kami, yaitu pengerjaan laporan dapat dilakukan dengan lebih cepat supaya tidak mepet dengan tanggal pengumpulan, pemberian komentar dengan jelas pada kode yang dibuat sejak awal sehingga anggota kelompok lain dapat lebih mudah untuk mengerti, dan mekanisme pembagian dan pengerjaan tugas dapat dilakukan dengan lebih baik dan jelas.

DAFTAR PUSTAKA

1. https://dosen.perbanas.id/menyelesaikan-knapsack-problem-dengan-menggunakan-algoritma-greedy/
2. https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md#all-commands
3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
4. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf
5. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

LAMPIRAN

Link Repository Github : https://github.com/bernarduswillson/Tubes1_paSTI-MAh-rank-1.git
Link Video Bonus : https://youtu.be/dVTqu51C2d8