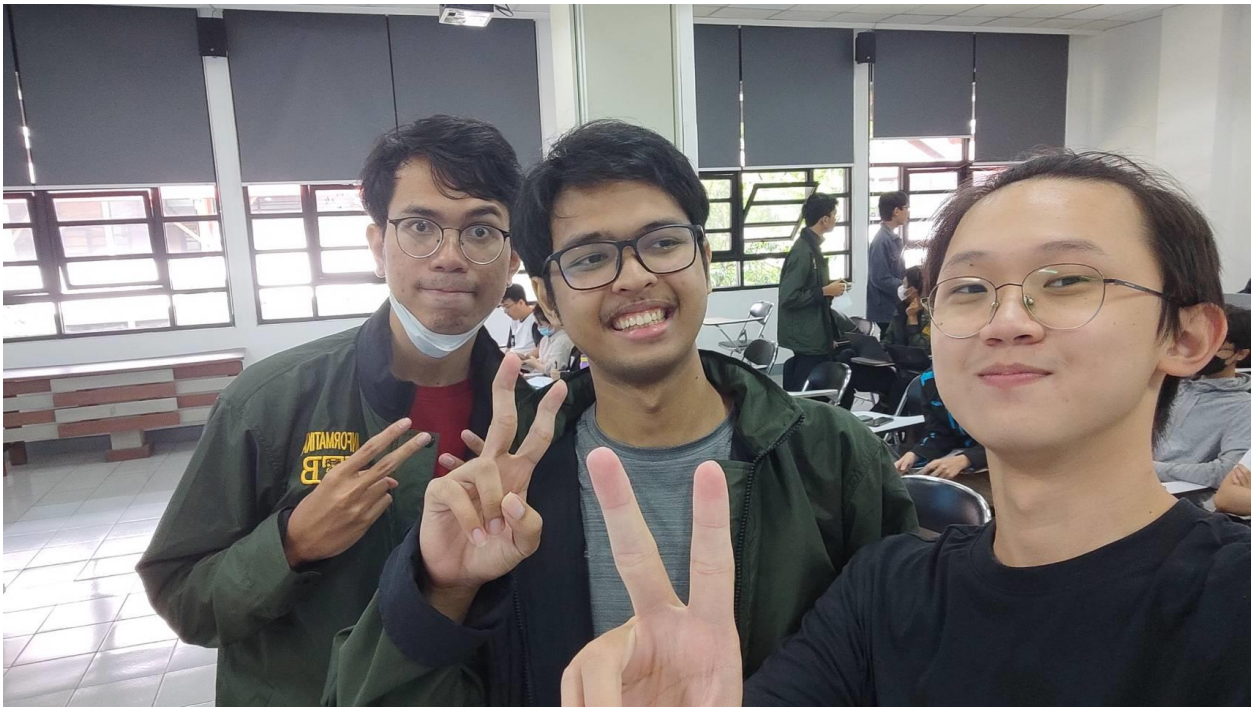
# LAPORAN TUGAS BESAR I

# IF2211 STRATEGI ALGORITMA

Pemanfaatan Algortima *Greedy* dalam Aplikasi Permainan *Galaxio*




Disusun oleh:

| | |
|---|---|
| Fakhri Muhammad Mahendra | 13521045 |
| Kenneth Ezekiel Suprantoni | 13521089 |
| Arsa Izdihar Islam | 13521101 |

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung


2023

# DAFTAR ISI

# BAB I
# DESKRIPSI MASALAH

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 Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Galaxio. Game engine dapat diperoleh pada laman berikut:

https://github.com/EntelectChallenge/2021-Galaxio

Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan Galaxio dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada starter-bots di dalam starter-pack pada laman berikut ini:

https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2

Permainan Galaxio berada didalam sebuah peta yang adalah bidang kartesian bilangan bulat pada arah positif dan negatif, sehingga pemain hanya dapat berada pada posisi (x, y) bilangan bulat, dimana tengah dari peta adalah (0, 0) dan ujungnya berada sesuai *radius* yang akan terus mengecil dari tengah peta. Terdapat beberapa objek permainan yang tersebar di peta dan direpresentasikan dengan bentuk bulat, antara lainnya:

- *Food* : Objek statis yang jika dikonsumsi pemain akan menambahkan 3 satuan *size*
- *Super Food* : Objek *Food* dengan kesempatan beberapa persen menjadi *super food*, yang akan meningkatkan tingkat konsumsi sebesar 2 kali
- *Wormhole* : Sepasang objek yang akan men-*transport* objek yang memasuki nya ke lokasi pasangan-nya jika objek tersebut lebih kecil dari *Wormhole*
- *Asteroid Field* : Kumpulan objek yang jika menyentuh pemain, akan mengurangi kecepatan pemain sebesar 2 kali kecepatan-nya sekarang

- *Torpedo Salvo* : Objek yang bisa ditembakkan oleh pemain, yang akan mengonsumsi 5 satuan *size* pemain dan mengeluarkan objek *torpedo* sebesar 10 satuan *size*, yang akan mentransfer satuan *size* dari pemain yang terkena kepada pemain yang menembakkan
- *Supernova* : Objek yang muncul sekali saat permainan, yang serupa dengan torpedo, dan dapat ditembakkan, saat ditembakkan lalu diledakkan, akan menghasilkan *Gas Clouds* yang berukuran besar
- *Teleport* : Objek yang bisa ditembakkan oleh pemain dan saat di

*Command* yang dapat dipanggil oleh pemain antara lain:

- *FORWARD* : Menggerakan pemain ke sebuah arah
- *STOP* : Menghentikan pemain
- *START_AFTERBURNER* : Mengaktifkan *afterburner*
- *STOP_AFTERBURNER* : Menghentikan *afterburner*
- *FIRE_TORPEDOES* : Menembakkan *torpedo salvo*
- *FIRE_SUPERNOVA* : Menembakkan *supernova*
- *DETONATE_SUPERNOVA* : Meledakkan *supernova*
- *FIRE_TELEPORTER* : Menembakkan *teleporter*
- *TELEPORT* : Mengaktifkan *teleport*
- *USE_SHIELD* : Mengaktifkan *shield*

# BAB II
# LANDASAN TEORI

## A. Algoritma *Greedy*

Algoritma *Greedy,* adalah algoritma yang umumnya digunakan untuk optimasi, dimana algoritma tersebut melibatkan memilih pilihan yang dinilai paling optimal dalam saat tersebut, tanpa memikirkan apa yang akan terjadi pada masa depan. Sehingga, algoritma *greedy* berlawanan dengan pilihan-pilihan konservatif yang mementingkan efeknya di masa depan. Umumnya, algoritma *greedy* akan diterapkan dengan pendekatan heuristik, dimana pendekatan tersebut akan mendefinisikan, untuk persoalan tertentu, parameter apa yang dipentingkan, atau dalam kata lain, yang menjadi dasar untuk pemilihan solusi yang paling optimal. Sehingga, semua algoritma yang memecahkan masalah secara heuristik dengan selalu mengambil pilihan yang optimal pada saat tersebut dapat dikategorikan sebagai algoritma *greedy*. Walaupun algoritma *greedy* tidak selalu menjamin solusi yang optimal, namun solusi yang diberikan algoritma ini dapat menghasilkan solusi yang optimal secara lokal, dimana solusi tersebut dapat mendekati solusi optimal yang global jika diberikan waktu yang cukup.

## B. Cara Kerja Program

Eksekusi dari aksi bot pada kelompok kami masih mengikuti struktur BotService.java yang diberikan dalam template JavaBot, dimana bot akan memilih suatu aksi beserta arah dari aksi tersebut yang disebut *heading* berdasarkan suatu algoritma yang ditetapkan, dalam kasus ini, adalah algoritma *greedy*.

Perubahan yang dilakukan oleh kelompok kami adalah pada bagaimana bot menentukan aksi selanjutnya, dimana kami membuat sebuah kelas bernama Processor untuk menjadi parent dari prosesor fitur-fitur yang akan diimplementasikan. Sehingga, prosesor-prosesor tersebut akan menghitung semua kemungkinan aksi dan akan memberikan nilai bagi aksi-aksi tersebut yang dilabelkan sebagai *weight,* dan pemilihan yang dilakukan oleh bot adalah *Greedy by weight*. (Ini lebih ke bab 3)

Implementasi dari algoritma *greedy* ke dalam bot dapat dilakukan lewat sebuah *method* pada *class BotService* yang bernama *ComputeNextPlayerAction* dimana di dalam *method* tersebut, akan

dijalankan semua *processing* yang perlu dilakukan, yang akan menghasilkan suatu *list of actions* beserta *weight* dari semua *actions*-nya. Algoritma *greedy* untuk mendapatkan aksi yang paling optimal pada *game state* tersebut adalah *greedy by weight*, dimana algoritma akan memilih *weight* yang terbesar sebagai aksi yang dinilai paling optimal pada saat itu dan menjalankannya. Batasan yang kami terapkan pada algoritma *greedy* kami, dimana kami rasa jika batas ini terlewat, algoritma kami tidak bisa diklasifikasikan sebagai algoritma *greedy* adalah basis waktu yang digunakan, dimana bot kami akan selalu melihat *game state* sekarang, tidak bisa belajar dari hasil aksi yang dilakukan pada *game state* yang sudah lalu, dan juga tidak bisa memprediksi *game state* yang akan datang.

*Game engine* yang digunakan adalah *game engine* yang sudah disediakan didalam *starter-pack* bot. *Game engine* itu sendiri terbagi menjadi dua bagian, *game engine* yang menjalankan *game* nya dan *visualiser* yang memvisualisasikan *game* nya. Di dalam folder *JavaBot*, sudah tersedia *file run.sh* yang siap dijalankan oleh pengguna, umumnya menggunakan terminal linux, atau bisa juga dengan git bash. Secara *default*, pengguna diperlukan untuk selalu melakukan *build* pada *package JavaBot* yang dibuat menjadi .jar dengan perintah *mvn package*, tetapi kelompok kami sudah mengintegrasikan perintah tersebut ke dalam *file run.sh* sehingga saat mengeksekusi *file run.sh*, *package JavaBot* juga akan secara automatis ter-*build* menjadi .jar.

Untuk menyimpulkan:

1. Build JavaBot melalui command **mvn package**
2. Jalankan *game-engine* dengan **sh run.sh**
3. Setelah *game-engine* selesai, *log* permainan akan dimasukkan ke dalam folder *logger*
4. Buka *Galaxio visualizer* di ***Galaxio.exe***
5. Pilih *file log* dari permainan yang baru saja dimainkan, *file* tersebut akan secara automatis dinamakan dengan tanggal dan waktu dari akhir *game*

Untuk tambahan, mengutip dari *file Get Started Galaxio* yang diberikan,

Berdasarkan gambaran cara kerja program game yang telah disebutkan sebelumnya, berikut merupakan cara menjalankan game secara lokal di **Windows**:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada *file* JSON ”appsettings.json” dalam folder “runner-publish” dan “engine-publish”

2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah “dotnet GameRunner.dll”
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah “dotnet Engine.dll”
6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah “dotnet Logger.dll”
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 *file* JSON “*GameStateLog_*{*Timestamp*}” dalam folder “logger-publish”. Kedua file tersebut diantaranya *GameComplete* (hasil akhir dari permainan) dan proses dalam permainan tersebut.

**Note:** Untuk menjalankan Game Runner, Engine, atau Logger pada **UNIX-based** OS dapat memodifikasi atau langsung menjalankan “run.sh” yang tersedia pada starter-pack. Pada windows dapat menggunakan atau memodifikasi batch script seperti 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 "" dotnet ReferenceBot.dll
cd ../

pause
```

Bagian :: Bots dapat dimodifikasi dari menjalankan Reference Bot menjadi Bot kalian sendiri. Untuk langkah build *source code* bot kalian dan menjalankannya dapat dibaca pada bagian Starter Bot di bawah ini.

# BAB III
# APLIKASI STRATEGI GREEDY

## A. *Mapping* persoalan Galaxio

Algoritma greedy memiliki beberapa elemen atau istilah yang sering menyertainya, diantaranya:

1. Himpunan kandidat (C): berisi kandidat yang akan dipilih pada setiap langkah. (*Filter* yang dilakukan dalam masing-masing prosesor)
2. Himpunan solusi (S): berisi kandidat yang sudah dipilih. (Hasil dari prosesor)
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi. (Modul prosesor)
4. Fungsi seleksi: memilih kandidat berdasarkan strategi greedy tertentu. Strategi ini bersifat heuristik. (*greedy by XXX)*
5. Fungsi kelayakan: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi. (Optimasi)
6. Fungsi obyektif: memaksimumkan atau meminimumkan (Seleksi aksi optimum)

Persoalan di dalam permainan Galaxio dibagi menjadi fitur-fitur *game*-nya dan *game objects* yang ada, sehingga *mapping* yang dihasilkan dari persoalan-persoalan tersebut dapat dirangkum menjadi:

| No | Persoalan | Kegunaan | Fungsi proses |
|---|---|---|---|
| **1** | *Food* | • Menambahkan *size* bot yang memakannya sebanyak 3 satuan<br>• Diproses secara *greedy by nearest distance* | *Food Processor* |
| **2** | *Superfood* | • Menambahkan *size* bot yang memakannya sebanyak 3 satuan<br>• Membuat bot yang memakannya memasuki efek dimana semua *food* yang dimakan akan bernilai | *Food Processor* |

| | | | |
|---|---|---|---|
| | | dua kali lipat selama jangka waktu tertentu<br>• Diproses secara *greedy by nearest distance* | |
| **3** | *Wormholes* | • Membuat bot yang *collide teleport* ke *wormhole* pasangannya<br>• Tidak diproses karena dinilai tidak dapat dipetakan secara *greedy* | - |
| **4** | *Gas Clouds* | • Membuat bot yang *collide* berkurang *size* nya terus menerus sampai tidak *collide*<br>• Diproses secara *greedy by distance*, dimana jika sudah memasuki *threshold distance* tertentu, prioritas untuk menjauh akan tinggi<br>• Diproses juga secara *greedy by distance and position*, dimana jika bot berada diantara *Gas Cloud* dan *Edge*, bot akan menembak *Gas Cloud* | *Obstacle Processor, Torpedo Processor Obstacle* |
| **5** | *Asteroid Fields* | • Membuat bot yang *collide* berkurang *speed* nya sampai tidak *collide*<br>• Tidak diproses karena dianggap kurang signifikan | - |
| **6** | *Torpedo Salvo* | • Dapat ditembakkan oleh bot, dengan mengorbankan 5 satuan | *Torpedo Processor Player,* |

| | | | |
|---|---|---|---|
| | | *size*, tetapi akan mengeluarkan 10 *torpedo* dimana 1 *torpedo* yang mengenai bot lawan akan mengurangi *size* lawan sebanyak 1 satuan dan menambahkan *size* bot penembak sebanyak 1 satuan<br>• Diproses secara *greedy by guarantee value*, dimana dibuat sebuah algoritma untuk menghitung apakah *torpedo* akan mengenai lawan menggunakan vektor<br>• Diproses sebagai *key* dari pemrosesan *Gas clouds* sebagai *obstacle*<br>• Diproses juga untuk menghindari, dengan *greedy by distance and heading* | *Torpedo Processor Obstacle, Dodging Processor* |
| **7** | *Supernova* | • Dapat diambil dan ditembakkan oleh bot, yang akan menghasilkan *area Gas Clouds* yang luas<br>• Akan diproses untuk diambil dengan *greedy by nearest distance* | *Food Processor* |
| **8** | *Teleport* | • Dapat ditembakkan oleh bot untuk *teleport* ke posisi dimana *teleporter*-nya berada<br>• Diproses secara *greedy by guarantee hit* dan juga *greedy by* | *Teleport Processor, Run From Teleporter Processor* |

| | | | |
|---|---|---|---|
| | | *position*, dimana bot akan menembakkan *teleporter* jika dirasa akan mengenai lawan dan lebih besar dari lawan, dan akan mengaktifkannya jika dirasa dapat memakan lawan | |
| **9** | *Afterburner* | • Dapat dinyalakan untuk meningkatkan *speed* dari bot, dimana *afterburner* akan mengonsumsi 1 satuan *size* per *tick* | - |
| **10** | *Shield* | • Dapat diaktifkan oleh bot untuk men-*deflect torpedo* yang datang, dimana *shield* akan aktif selama 20 *tick*, menghabiskan 20 satuan *size*, dan akan *cooldown* selama 20 *tick*<br>• Diproses secara *greedy by torpedo value, torpedo distance, and torpedo heading* | *Shield Processor* |
| **11** | *Map* | • Akan mengecil selama permainan berlangsung<br>• *Edge* akan mengurangi *size* dari bot yang mengenainya<br>• *Edge* diproses secara *greedy by distance and position* | *Edge Processor* |
| **12** | *Players* | • Objek yang akan menjadi lawan dari bot | *Enemy Processor* |

| | | | |
|---|---|---|---|
| | | • Diproses untuk menghindari player yang lebih besar secara *greedy by value and distance* | |

## B. Eksplorasi Alternatif Solusi

Menurut kelompok kami, terdapat beberapa alternatif solusi selain penerapan algoritma *greedy* pada persoalan optimasi bot. Pertama, ada kemungkinan untuk menerapkan solusi berdasarkan konsep *Machine learning* dalam *artificial intelligence*, dimana bot akan mencoba suatu strategi dasar, mencoba suatu variasi, dan mempertahankan variasi yang menang, sampai tercipta suatu strategi kompleks yang bisa secara konsisten memenangkan pertandingan jika diberikan suatu keadaan *game* yang optimal. Kedua, pendekatan prediksi juga bisa dilakukan, dimana pendekatan ini adalah gabungan antara algoritma *greedy* dan *backtracking*, karena bot akan melihat suatu *game state,* mencoba perhitungan *course of actions* yang akan di *chain* untuk semua kemungkinan *actions*, dengan memprediksi beberapa *game state* kedepan. Pendekatan ini *greedy* karena akan mengambil *best course of actions*, tetapi *backtracking* juga karena diperlukan suatu algoritma untuk mencari *best course of actions* yang baru jika prediksi gagal dengan *backtracking* dari beberapa *actions* yang sudah di-*chain* tetapi belum dijalankan.

Selain alternatif *non-greedy*, terdapat juga pendekatan alternatif yang masih dapat dikategorikan sebagai algoritma *greedy*. Alternatif pertama yang dipertimbangkan adalah *greedy by position*, dimana bot akan selalu memprioritaskan posisi paling tengah dari *map*, alternatif ini memiliki keuntungan dimana jika bot berhasil bertahan sampai peta mengecil, bot akan tetap hidup, tetapi alternatif ini kurang didukung karena dinilai terlalu simpel dan juga tidak memperhitungkan faktor-faktor lain yang terdapat dalam permainan. Alternatif kedua yang dipertimbangkan adalah *greedy by game condition*, dimana bot akan memprioritaskan mencari makanan terdekat saat awal game, dan saat akhir game, lebih fokus ke pertahanan dan tidak keluar dari peta, alternatif ini tidak dipilih karena dianggap terlalu pasif dan juga sangat tidak fleksibel terhadap kondisi bot relatif terhadap bot-bot lainnya. Alternatif selanjutnya adalah *greedy by distance* dimana bot akan memprioritaskan memakan hal terdekat yang bisa dimakannya, alternatif ini juga tidak dipilih karena kurang fleksibel dan kurang mempertimbangkan fitur-fitur lain yang terdapat di dalam permainan.

Selain itu, *mapping* persoalan yang disebutkan sebelumnya, masing-masing persoalan dapat diatasi dengan algoritma *greedy*-nya masing-masing, yaitu:

1. Food: *greedy by nearest food, greedy by safe location*
2. Superfood: *greedy by nearest super* food, *greedy by safe location*
3. Gas Clouds: *greedy by distance, greedy by position*
4. Torpedo Salvo: *greedy by guarantee, greedy by distance and heading*
5. Supernova: *greedy by nearest distance*
6. Teleport: *greedy by guarantee, greedy by position*
7. Shield: *greedy by torpedo value, distance, and heading*
8. Map: *greedy by distance, greedy by position*
9. Players: *greedy by value*, *greedy by distance*

### C. Strategi Greedy yang Dipilih

Pada akhirnya, diputuskan untuk menggunakan algoritma *greedy* yang dinilai paling fleksibel, dan bisa mempertimbangkan semua pendekatan *greedy* yang sudah disebutkan, yaitu ***greedy by weight***, dimana di dalamnya akan terdapat perhitungan-perhitungan yang menggunakan algoritma *greedy* yang sudah disebutkan seperti *greedy by position*, *game condition,* dan *distance* untuk menghitung *weight* atau prioritas aksi tersebut. Dengan kata lain, algoritma ini akan menggabungkan seluruh algoritma-algoritma greedy pada tiap persoalan yang akan diselesaikan secara terpisah terlebih dahulu, baru dibandingkan *action* apa dan bagaimana yang paling menguntungkan untuk dilakukan. Algoritma *greedy by weight* ini dinilai paling fleksibel karena kemampuannya untuk *mapping* semua persoalan fitur di dalam permainan menjadi suatu nilai prioritas, berdasarkan kondisi game, jarak, dan hal-hal sebagainya. Seperti contoh, untuk perhitungan aksi, terdapat banyak prosesor algoritma *greedy* yang diterapkan dalam mencari aksi yang dinilai paling optimal saat itu, ditandakan dengan *weight* terbesar, prosesor makanan atau *default* akan, seperti algoritma *greedy* pada biasanya, memfilter dan *sort* semua makanan berdasarkan jaraknya dari bot secara menaik, dan diambil yang paling dekat, sehingga algoritma pemilihan tersebut dapat dikategorikan sebagai *greedy by nearest distance*, yang ditranslasikan menjadi suatu *weight*, yang akan juga digunakan untuk pemilihan aksi dalam algoritma utama *greedy by weight*.

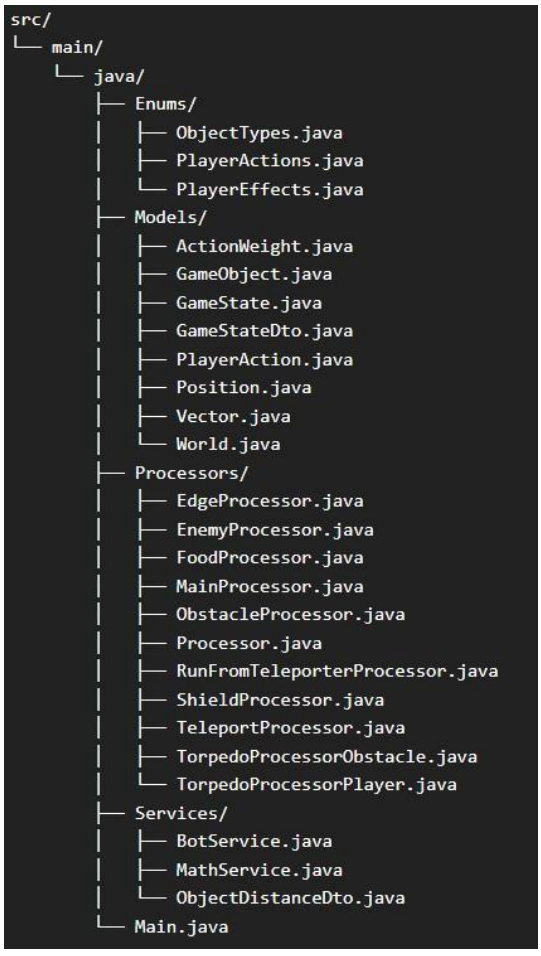
# BAB IV
# IMPLEMENTASI DAN PENGUJIAN

## A. Repository Github

Seluruh kode program bot Galaxio dari kelompok kami dapat diakses melalui repository Github pada tautan berikut: https://github.com/arsaizdihar/Tubes1_Team-AFK

## B. Struktur Data Program

Secara keseluruhan, struktur folder program yang kami buat untuk bot Galaxio adalah sebagai berikut:




Seperti yang terlihat pada struktur folder tersebut, terdapat empat modul utama. Modul Enums berfungsi untuk menyimpan enum-enum yang berkaitan dengan objek *game.* Modul Models berfungsi untuk mengatur dan menyimpan objek dan *state game* (Class GameObject,

GameState, GameStateDto, dan World) beserta properti yang mendukungnya (Class ActionWeight, PlayerAction, Position, dan Vector). Modul Services berfungsi untuk mengurus fungsi dan proses dalam membantu perhitungan (MathService) dan berjalannya bot.

Didalam Modul MathService, proses-proses yang kami implementasikan adalah sebagai berikut:

- getDistanceBetween : Menghasilkan jarak antara dua benda atau posisi di dalam permainan
- getHeadingBetween : Menghasilkan arah dari pemain ke benda
- toDegrees : Mengonversikan radian ke derajat
- reverseHeading : Membalikkan arah *heading* sebesar 180 derajat
- getObjectsInArea : Mengembalikan sebuah list yang berisi seluruh objek di dalam suatu radius tertentu dengan posisinya sebagai titik tengah
- calcObjectValueBetweenObjects : Mengembalikan *value* satuan *size* dari objek-objek yang beririsan dengan suatu vektor diantara dua objek
- guaranteeHitTorpedo : Sebuah fungsi untuk memastikan apakah suatu *torpedo* diperkirakan akan mengenai sasaran atau tidak
- isCollide : Mengembalikan apakah dua objek berkolisi atau tidak
- getDegreeDifference : Mengembalikan selisih arah antara dua arah sudut
- isInTorpedoPath : Mengembalikan apakah suatu objek didalam jalur *torpedo* atau tidak
- torpedoEscapeHeading : Mengembalikan arah untuk kabur dari *torpedo*
- distanceToTorpedoPath : Mengembalikan jarak ke jalur torpedo
- calcObjectValueInArea : Mengembalikan suatu nilai dari semua objek yang berada di dalam suatu area
- getPositionFromAPoint : Mengembalikan suatu posisi yang dihitung dari jaraknya dan sudutnya berdasarkan suatu sisi

Modul Processors merupakan bagian yang mengatur garis besar program bot, dengan garis bawah dalam modularitas. Class yang ada pada modul ini merupakan *inheritance* dari *abstract class* Processor yang memiliki properti “bot” yang menyimpan *state* bot, “gameState” yang menyimpan *state game*, dan “data” yang berguna menyimpan hasil proses algoritma greedy dari tiap prosesor. Selain itu terdapat *method abstract* “process” yang merupakan prosedur utama untuk

melakukan *processing* algoritma. Semua Class *children* dari Processor ini akan memiliki algoritma greedy masing-masing. Pendekatan ini mengaplikasikan *design principle* "Separation of concerns" yaitu setiap proses yang memiliki fokus permasalahan tertentu dipisah dalam class processor tersendiri, contohnya ObstacleProcessor hanya akan berfokus pada algoritma greedy untuk menghindari *disadvantage* akibat *obstacle.* Setelah tiap processor selesai dijalankan, setiap "weight" atau pembobotan urgensi tiap command dari seluruh processor akan dibandingkan dan dicari yang tertinggi. Maka dari itu, pada bagian implementasi algoritma, kami hanya akan menjelaskan algoritma dari *method* process tiap Processor.

C. **Implementasi dalam Pseudocode**

```
Class FoodProcessor
procedure process()

Algoritma
nearest = makanan terdekat yang berada pada posisi aman
close = makanan terdekat kedua
if (nearest exists)
      proses mencegah bot berpindah-pindah makanan yang diincar
      heading = arah bot ke makanan terdekat
      weight = 3
      tambahkan action forward dengan heading = heading dan weight = weight ke
dalam data prosessor
```

```
Class EdgeProcessor
procedure process()

Algoritma
distanceToCenter = jarak bot ke tengah world
heading = arah bot ke tengah world
if (jarak bot ke tengah cukup jauh hingga cukup dekat ke ujung world) then
      weight = 1000
else
      weight = 2
tambahkan action forward dengan heading dan weight ke dalam data prosesor
```

```
Class ShieldProcessor
procedure process()
```

Algoritma

```
if (shield bot sedang aktif) then return {tidak perlu proses shield}
nearbyHeadingTorpedos = torpedo pada jarak dekat tertentu dan kemungkinan akan kena bot
for (torpedo : nearbyHeadingTorpedos)
      if (jarak bot ke torpedo <= batas tertentu) then
            existCriticalTorpedos = true
if (existCriticalTorpedos && neadbyHeadingTorpedos.size() >= jumlah minimum untuk shield && bot.getSize() >= ukuran minimum untuk shield) then
      masukkan action activate shield dengan weight yang cukup tinggi
```

```
Class TeleportProcessor
procedure process()

if (bot tidak dalam keadaan menembakkan teleport && ukuran bot > batas yang ditentukan)
      for (ply: listPlayerMusuh)
            heading = arah bot ke musuh
            guarantee = MathService.guaranteeHitTorpedo(bot.position, ply)
            hitRate = guarantee ? 1 : 0.2
            weight = ply.getSize() * hitRate
            tambahkan action fire teleport dengan heading dan weight
else {sedang ada teleport yang dikeluarkan}
      playerInRadius = player yang akan termakan apabila teleport digunakan
      if (playerInRadius exist) then
            tambahkan aksi teleport dengan weight yang cukup tinggi
```

```
Class TorpedoProcessorPlayer
procedure process()
```

Algoritma

```
if (bot.getSize() > batas tertentu && salvoCount > 0) then
      for (obj : listPlayerMusuh)
            obstacleValue = banyaknya obstacle yang akan tertabrak sebelum terkena musuh
            guarantee = MathService.guaranteeHitTorpedo(bot.getPosition(), obj)
```

```
            hitRate = guarantee ? 1 : 0
            weight = (10 − obstacleValue − 5) * hitRate
            heading = arah bot ke musuh
            masukkan action fire torpedo dengan weight dan heading
```

```
Class TorpedoProcessorObstacle
procedure process()

Algoritma
gasCloudList = list game object bertipe gas cloud
for (obj: gasCloudList)
      if (bot.getSize() > 20 && salvoCount > 0) then
            heading = arah bot ke obj
            biggerThanHalf = obj.getSize() > bot.getSize() * 0.5
            canHead = obj.getSize() * 2.0 / bot.getSpeed() <= 0.5 *
bot.getSize()
            canDestroy = !biggerThanHalf
            distanceToCenter = jarak bot ke tengah world
            if (worldRadius − distanceToCenter + bot.getSize() >
thresholdDistance && heading == arah bot ke tengah world) then
                  if (canHead) then
                        weight = 601
                        action = forward
                        heading = arah bot ke tengah world
                  else if (canDestroy)
                        weight = 601
            else
                  weight = 2
            masukkan action, weight dan heading ke data
```

```
Class ObstacleProcessor
procedure process()

Algoritma
obstacleList = list gas cloud pada world
for (obj : obstacleList)
      distance = jarak bot ke obj
      if (distance <= 6) then
```

```
            weight = 500
            if (distance < 3 and distance >= 0) then
                  weight = 600
                  heading = arah sebaliknya dari arah bot ke obstacle
            else if (distance < 0) // didalam gas cloud
                  heading = arah titik 0,0
                  defVal = value objek yang berada sejauh radius dari titik
dengan arah ke 0,0
                  // cari kiri atau kanan, ada yang lebih baik kah daripada ke
tengah
                  altLeft = arah ke kiri bot, jika dekat ( beda < 30 derajat)
dengan heading, maka altLeft = heading
                  altRight = arah ke kanan bot, jika dekat ( beda < 30 derajat)
dengan heading, maka altRight  = heading
                  leftVal = value objek yang berada sejauh radius dari titik
yang di arah altLeft
                  rightVal = value objek yang berada sejauh radius dari titik
yang di arah altRight
                  heading dipilih = heading dengan value terbesar

      masukkan action, weight dan heading ke data
```

```
Class RunFromTeleporterProcessor
Procedure process()

Algoritma
BiggestPlayer = pemain terbesar di map selain bot sendiri
tlpList = list semua teleporter di map yang dekat dan mengarah ke bot pemain
if (tlpList.size > 0)
    obj = teleporter terdekat
    if (jarak antara bot dan obj <= BiggestPlayer.size + suatu threshold number)
        altLeft = arah kiri dari bot
        altRight = arah kanan dari bot
        pointDistance = parameter distance ke area
        leftVal = value semua objek didalam area sebesar radius dengan titik
tengah sebuah titik sejauh pointDistance dengan arah altLeft dari posisi bot
```

```
        rightVal = value semua objek didalam area sebesar radius dengan titik tengah sebuah titik sejauh pointDistance dengan arah altRight dari posisi bot
        heading = pilih heading altLeft jika leftVal > rightVal dan sebaliknya
        if (!bot.hasJustFireTeleport and bot.teleporterCount > 0 and belum ada teleporter milik bot and bot.getSize() > 35)
            masukkan heading yang dipilih dan weight beserta action FireTeleport ke data
    if not (belum ada teleporter milik bot)
        if (jarak antara obj dan teleporter bot >= bot.size + suatu threshold + BiggestPlayer.size)
            masukkan weight dan aksi Teleport ke data
```

```
Class EnemyProcessor
Procedure process()

Algoritma
If ukuranBot < 50
      detectedTorpedo = daftar torpedo yang berjarak  suatu threshold tertentu dari bot
      nearestTorpedos = torpedo terdekat dengan bot dari list detectedTorpedo
      if (jarak lintasan nearestTorpedo dengan bot < botSize + 10 do
            heading = menjauhi garis lintasan torpedo
            weight = 100
            masukkan action Forward dengan weight dan heading
```

## D. Pengujian

| Nama yang diuji | Implementasi Strategi | Hasil |
|---|---|---|
| **Pengambilan makanan** | Strategi diimplementasikan dengan mencari makanan dengan jarak terdekat dari bot | Strategi dalam kebanyakan kasus sudah bisa dengan efektif mengambil makanan yang ada di sekitarnya,<br><br>Namun terdapat kasus spesial dimana bot bisa terjebak karena strategi ini, |

| | | |
|---|---|---|
| | | yaitu ketika bot berada tepat di tengah-tengah dua makanan. |
| **Menghindari *obstacle* (*gas cloud)*** | Strategi diimplementasikan dengan bergerak ke arah menjauh dari *obstacle* apabila terdapat *obstacle* yang cukup dekat. | Strategi berjalan dengan baik pada mayoritasi kasus, namun masih terdapat permasalahan:<br>*1.* Ketika bot berada di antara gas cloud dan edge, maka yang terjadi adalah bot terjepit dan tidak bisa kabur dari situasi tersebut. |
| **Menjauhi center pada awal game** | Pada awal game bot akan memprioritaskan makanan yang mengarahkan tujuan ke bagian luar map. | Strategi berjalan dengan baik, dimana bot bergerak keluar dari tengah map pada awal game untuk menghindari pertempuran awal game yang terjadi di sekitar tengah map.. |
| **Menghindari Edge** | Jika bot sudah berada di dekat pinggiran peta dengan jarak tertentu, maka bot akan bergerak ke tengah peta. | Strategi berjalan dengan baik pada kebanyakan kasus, namun masih terdapat beberapa permasalahan:<br>*1.* Ketika bot berada di antara gas cloud dan edge, maka yang terjadi adalah bot terjepit dan tidak bisa kabur dari situasi tersebut.<br>*2.* Ketika bot sangat kecil size nya, seakan-akan aturan ini diabaikan dan bot malah terus berjalan keluar peta. |
| **Mengaktifkan Shield** | Strategi diimplementasikan dengan mendeteksi apakah terdapat cukup banyak torpedo | Strategi berjalan dengan baik |

| | | |
|---|---|---|
| | yang memenuhi kualifikasi berikut<br>1. Torpedo musuh mengarah ke bot.<br>2. Torpedo cukup dekat dengan bot sehingga jika shield diaktifkan sekarang, torpedo masih bisa di ditahan oleh shield pada sebelum 20 tick terjadi.<br>Ketika sudah ada minimal 2 torpedo yang memenuhi kriteria tersebut, di cek kembali dari torpedo-torpedo tersebut apakah ada yang cukup dekat sehingga pada tick selanjutnya torpedo akan *collide* dengan bot.<br><br>Ketika semua kondisi sudah terpenuhi maka baru bot akan mengirimkan pesan nyalakan shield.<br><br>Pesan untuk menyalakan shield hanya dikirimkan apabila bot tidak sedang dalam keadaan shield menyala. | |

| | | |
|---|---|---|
| **Menembakkan torpedo** | Strategi diimplementasikan dengan memperhatikan beberapa variabel<br>1. Jarak dari bot ke posisi musuh<br>2. Kecepatan gerak musuh<br>3. Besar musuh<br>4. Kecepatan torpedo<br>5. Banyak obstacle yang diantara bot dan musuh<br>6. Banyak amunisi torpedo yang dimiliki<br>7. Biaya untuk menembakkan teleport | Strategi sudah berjalan dengan baik. |
| **Menembakkan teleport** | Strategi teleport diimplementasikan dengan strategi yang mirip dengan menembakkan torpedo, namun ada beberapa konsiderasi tambahan, seperti:<br>1. Biaya tambahan untuk menembak teleport.<br>2. Besar selisih bot dengan target yang harus memenuhi kuota tertentu. | Strategi sudah berjalan dengan baik, namun untuk beberapa kejadian yang dirasa cukup tepat untuk menembakkan teleport, bot masih belum bisa mengenali situasi tersebut dengan baik. |
| **Melakukan teleport** | Strategi digunakan dengan mengkonsiderasi posisi teleport pada saat ini, apakah di posisi tersebut dengan bisa collide | Strategi sudah berjalan dengan baik. |

| | | |
|---|---|---|
| | dengan bot musuh yang size nya lebih kecil atau tidak | |
| **Menembakkan obstacle** | Strategi ini diimplementasikan sebagai pencegahan dari kondisi *stuck* atau *deadlock* diantara obstacle, dimana player akan menembak obstacle jika obstacle tersebut dinilai bisa dihancurkan | Strategi ini berjalan cukup baik, hanya saja memang kondisi ini sudah jarang terjadi karena sudah ter-*handle* dengan baik oleh modul Menghindari *obstacle* |
| **Menghindari teleport** | Strategi ini diimplementasikan untuk men-*counter teleport* yang ditembakkan oleh lawan, sehingga bot dapat bertahan lebih lama | Strategi ini dapat dinilai 60/100 karena belum 100% selalu menghindari *teleport* dari lawan |
| **Menghindari torpedo** | Strategi yang digunakan adalah mendeteksi torpedo yang yang sudah memasuki area deteksi tertentu. Jika sudah akan dicek jarak bot dengan garis lintasan torpedo. Jika jarak bot dengan garis masih terlalu dekat dengan garis lintasan torpedo, maka bot akan bergerak menjauhi garis lintasan torpedo. | Selama pengujian masih ditemukan kasus bot tidak menghindari torpedo seperti ekspektasi, terutama ketika ukuran bot sangat kecil. |

# BAB V
# KESIMPULAN, SARAN, KOMENTAR DAN REFLEKSI

## A. Kesimpulan

Kesimpulan yang dapat diambil oleh kelompok kami adalah bahwa algoritma *greedy* dapat digunakan untuk menentukan solusi lokal maksimum/optimum dari kondisi sekarang. Tetapi, lokal maksimum yang didapatkan dari kondisi sekarang belum pasti menjadi global maksimum. Selain itu, algoritma *greedy* juga dapat disimpulkan sebagai algoritma optimasi yang cukup baik dalam *decision making* dalam pembuatan sebuah bot, karena, dalam kebanyakan kasus, pengambilan solusi yang optimal di saat tersebut atau optimal lokal sudah cukup mendekati optimal global. Solusi optimal lokal akan berbeda dari optimal global hanya jika ada perhitungan atau prediksi, dan juga penyimpanan data yang lalu yang terlibat.

## B. Saran

Saran untuk kelompok kami diantaranya:

- Lebih mendalami *game engine*, karena banyak isu-isu yang tidak bisa di-*solve* karena kurang pengetahuan tentang *game engine* yang digunakan
- Lebih teratur dalam menjadwalkan pengerjaan Tugas besar
- Lebih banyak sesi *brainstorming* diantara sesi pengerjaan

## C. Refleksi

Refleksi yang kami dapatkan dari tugas ini adalah untuk lebih meningkatkan lagi kinerja dalam pengerjaan tugas besar, seperti dalam perancangan model, perancangan *timeline* pengerjaan, dan juga memperbaiki *time management* sehingga dapat menyelesaikan dan memperbaiki perancangan algoritma lebih jauh sebelum *deadline.* Hal lain yang kami dapatkan adalah pengalaman berharga tentang *bot development* dan perancagan algoritma untuk pengambilan keputusan bot.

# DAFTAR PUSTAKA


https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

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

https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md

https://docs.google.com/document/d/1Ym2KomFPLIG_KAbm3A0bnhw4_XQAsOKzpTa70IgnLNU/edit

https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2

Link video : https://youtu.be/S6gkOi8NH_A

Link GitHub : https://github.com/arsaizdihar/Tubes1_Team-AFK