

IF2211-03 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN “GALAXIO”

LAPORAN TUGAS BESAR 1



Oleh Kelompok 3 (beweGanteng)

Kelvin Rayhan Alkarim - 13521005

Kenny Benaya Nathan - 13521023

Ahmad Nadil - 13521024

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

DAFTAR ISI

BAB I - PENDAHULUAN	3
BAB II - LANDASAN TEORI	6
2.1. Algoritma Greedy	6
2.2. Elemen-elemen algoritma greedy	7
2.3. Game Engine	8
2.4. Game Engine Galaxio	8
2.5. Struktur Game Engine	9
2.6. Alur Permainan Galaxio	11
2.6.1. Menjalankan permainan Galaxio	11
2.6.2. Mengembangkan bot	13
BAB III - APLIKASI STRATEGI GREEDY	17
3.1. Alternatif Solusi Greedy	17
3.1.1. Greedy by size	17
3.1.2. Greedy by Attack	18
3.1.3. Greedy by Defense	19
3.2. Solusi Greedy yang Diterapkan dan Pertimbangannya	20
BAB IV - IMPLEMENTASI DAN PENGUJIAN	24
4.1. Implementasi dalam Pseudocode	24
4.2. Struktur Data Program	29
4.3. Analisis dan Pengujian	33
BAB V - PENUTUP	38
5.1. Kesimpulan	38
5.2. Saran	38
5.3. Refleksi	38
LAMPIRAN	39
DAFTAR PUSTAKA	40

BAB I

PENDAHULUAN

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.

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 salvo 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 disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi 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 Superfood 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 IF2211 Strategi Algoritma – Tugas Besar 1 3 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 menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam 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
- b. STOP
- c. STARTAFTERBURNER
- d. STOPAFTERBURNER
- e. FIRETORPEDOES
- f. FIRESUPERNOVA
- g. DETONATESUPERNOVA
- h. FIRETELEPORT
- i. TELEPORT
- j. ACTIVATESHIELD

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

2.1. Algoritma Greedy

Algoritma greedy adalah metode pemrograman yang memecahkan masalah optimasi dengan cara memilih pilihan terbaik pada setiap langkah. Ini berbeda dari metode backtracking atau pencarian heuristik yang memeriksa setiap solusi mungkin. Dalam algoritma greedy, kita memilih opsi yang paling baik saat ini dan melanjutkan dengan harapan bahwa pilihan-pilihan ini akan mengarah pada solusi optimal dalam jangka panjang. Algoritma ini sering digunakan dalam masalah optimasi yang melibatkan pemilihan elemen dari himpunan dan penempatannya dalam solusi.

Contohnya, dalam masalah knapsack, algoritma greedy akan memilih barang dengan rasio nilai-berat tertinggi dan memasukkannya ke dalam tas sampai tas penuh. Ini juga dapat digunakan dalam masalah pemilihan maksimal, dimana kita memilih elemen-elemen dengan nilai tertinggi sampai batas kapasitas tertentu tercapai. Algoritma greedy juga digunakan dalam masalah penjadwalan, dimana kita memilih tugas dengan deadline paling dekat untuk dikerjakan pertama.

Meskipun algoritma greedy sering memberikan hasil yang baik, itu tidak selalu memberikan solusi optimal. Terkadang, kita mungkin membuat pilihan yang salah pada langkah awal yang mengarah pada solusi yang kurang baik dalam jangka panjang. Oleh karena itu, penting untuk memvalidasi solusi yang ditemukan dengan algoritma greedy dengan membandingkannya dengan solusi yang diketahui benar.

Namun, salah satu keuntungan utama dari algoritma greedy adalah kecepatannya. Karena hanya memilih pilihan terbaik pada setiap langkah, algoritma ini sering berjalan lebih cepat daripada metode pencarian heuristik atau backtracking yang memeriksa setiap solusi mungkin. Ini juga relatif mudah dipahami dan diterapkan karena memiliki pendekatan yang sederhana dan intuitif.

Secara keseluruhan, algoritma greedy adalah alat yang berguna dalam memecahkan berbagai masalah optimasi. Meskipun tidak selalu menghasilkan solusi optimal, algoritma ini sering memberikan solusi yang baik dengan kecepatan yang sangat baik dan mudah dipahami. Oleh karena itu, algoritma ini sangat penting bagi para pemrogram agar mendapatkan alur program yang efektif. Dalam kasus ini, kami membuat program bot dalam game dengan menggunakan algoritma greedy agar mendapat aksi bot yang paling efektif dan efisien.

2.2. Elemen-elemen algoritma greedy

Algoritma greedy terdiri dari beberapa komponen, yaitu:

1. Himpunan kandidat (C)

Himpunan ini berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)

2. Himpunan solusi (S)

Himpunan yang berisi kandidat yang sudah dipilih

3. Fungsi solusi

Fungsi ini akan menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi

4. Fungsi seleksi (*selection function*)

Fungsi yang akan memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.

5. Fungsi kelayakan (*feasible*)

Fungsi akan memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)

6. Fungsi obyektif

Fungsi ini menjelaskan apakah untuk memaksimumkan atau meminimumkan

2.3. Game Engine

Game engine merupakan sebuah *Software Development Environment* (SDE) di mana kegunaannya adalah untuk membantu pembuatan *video game*. Sesuai dengan fungsinya, *game engine* terdiri dari berbagai komponen perangkat keras maupun lunak. Beberapa di antaranya seperti, 2D dan 3D *graphics rendering*, animasi, *Artificial Intelligence* (AI), manajemen memori, audio, dan lain-lain. Alat-alat ini digunakan untuk membantu semua proses desain dan pengembangan *video game* sekaligus mengurangi biaya pengembangan *video game*, hal ini dikarenakan pengembang tidak perlu membuatnya dari nol. Cukup memakai *game engine* yang sudah tersedia sebagai sebuah pondasi. Satu *game engine* bisa dipakai untuk mengembangkan berbagai *video game* yang berbeda tanpa melakukan perubahan yang banyak.

2.4. Game Engine Galaxio

Game engine dari permainan Galaxio dapat ditemukan pada tautan <https://github.com/EntelectChallenge/2021-Galaxio>. Ada beberapa komponen yang diperlukan untuk menjalankan permainan ini, yaitu:

1. *Engine*

Engine bertanggung jawab untuk mengaplikasikan *command* dari setiap *bot* yang sudah dibuat ke dalam *game state* dengan memperhatikan aturan yang sudah dibuat pada permainan Galaxio.

2. *Logger*

Logger bertanggung jawab untuk mencatat semua data beserta perubahan pada *Game State* setiap *tick*. *Logger* akan terhubung dengan *runner*, tempat di mana *bot* dan *engine* pun juga terhubung. Semua hal yang dicatat oleh *logger* akan dimasukkan ke dalam sebuah *log file* setiap kali sebuah permainan selesai berjalan. *Logger* juga melakukan tugas *exception logger*. File *log* yang dibuat oleh *logger* akan menjadi *input* untuk dibaca oleh *visualizer*.

3. *Runner*

Runner bertanggung jawab untuk menjalankan permainan dan mempertandingkan setiap peserta yang masuk ke dalam permainan. Selama menjalankan pertandingan, *Runner* akan memanggil *command* yang sudah

diberikan pada setiap *bot* kemudian memberikannya kepada *engine* untuk dieksekusi.

4. Starter Bots

Starter Bot inilah yang akan dipakai dalam permainan. *User* akan memberikan sejumlah *command* yang tentu saja terbatas kepada *bot* sebagai logika/alur berpikir yang akan dipakai oleh *bot* selama permainan berlangsung.

5. Visualiser

Visualiser adalah tempat untuk menampilkan visualisasi dari apa yang terjadi selama sebuah sesi permainan berlangsung berdasarkan semua hal yang terjadi di setiap *game state* dengan membaca file *log* terkait yang sudah dibuat oleh *Game Logger*.

6. Engine Compose

Engine Compose yang disediakan oleh Galaxio memberikan cara yang mudah dan nyaman untuk memproses dan menjalankan seluruh komponen pada *Game Engine* Galaxio di dalam docker. Pada tugas besar kali ini, pengembang tidak akan menggunakan *engine compose* karena *engine compose* ini tidak masuk ke dalam *starter-pack release* terbaru, sehingga game tidak akan dijalankan di dalam *docker container*.

2.5. Struktur Game Engine

Komponen-komponen yang terdapat di dalam *Game Engine* jelas akan sangat membantu untuk mengembangkan *bot* untuk permainan galaxio. *Game Engine* dari Galaxio dapat diunduh pada *starter-pack release* terbaru yang dikeluarkan oleh pengembang permainan Galaxio, yaitu *release* versi 2021.3.2. *Starter-pack* ini memiliki beberapa komponen, strurnya sendiri terdiri dari:

1. Folder engine-publish: Folder yang berisi *Game Engine*.
2. Folder logger-publish: Folder yang berisi *Game Logger*.
3. Folder reference-bot-publish: Galaxio menyediakan sebuah *template bot* yang bisa menjadi referensi untuk membangun ataupun sebagai lawan selama membangun *bot* sendiri.
4. Folder runner-publish: Folder yang berisi *Game Runner*

5. Folder starter-bots: Folder yang berisi berbagai folder bot dengan bahasa yang berbeda-beda sehingga bisa dimodifikasi sebagai tempat untuk mengembangkan *bot* sendiri.
6. Folder visualiser: Folder berisi *visualiser* dalam bentuk masih *zipped*. Ada 3 File zip sesuai dengan OS nya masing-masing (Linux, Windows, MacOS)
7. File README.md: ReadMe dari *repository Game Engine* Galaxio
8. File run.sh: File untuk menjalankan seluruh *Game Engine*
9. File building-a-bot.md: File penjelasan singkat cara membuat *bot*.

Karena tugas besar ini menggunakan bahasa Java, maka pengembang akan menggunakan folder JavaBot yang berada di dalam folder starter-bots sebagai tempat untuk mengembangkan *bot* dengan bahasa Java. Struktur dari folder JavaBot sendiri terdiri dari:

1. Folder .github
2. Folder src/main/java: Seluruh Source Code untuk membangun *bot*
3. Folder target: Program akan mengeksekusi file .jar yang terdapat di dalam folder ini.
4. File Dockerfile
5. File pom.xml: File konfigurasi untuk Apache Maven untuk melakukan proses *build project* secara otomatis

Seperti yang sudah dideskripsikan, pengimplementasian *bot* dilakukan di dalam folder src/main/Java. Ada beberapa folder dan file yang harus diperhatikan di dalam folder ini agar dapat mengimplementasikan *bot* dengan benar, yaitu:

1. Folder Enums

Folder ini berisi semua tipe objek yang ada dalam permainan sekaligus *action* yang bisa dilakukan pada *bot*. Data pada kelas yang ada di dalam Enums memiliki nilai konstan

2. Folder Models

Secara garis besar, folder Models berpengaruh pada perpindahan dan perubahan *Game State*. Mulai dari *state world* pada Galaxio hingga *state* pada setiap objek.

3. Folder Services

Di dalam folder ini, terdapat file BotService.java. Di sinilah tempat untuk mengimplementasikan *bot*. Tempat untuk menaruh logika berpikir dari *bot* beserta perintah-perintahnya bisa dilakukan di dalam procedure computeNextplayerAction yang menerima parameter variabel bertipe PlayerAction.

4. File Main.java

File program utama dari bot ini, di mana akan menghubungkan *bot* dengan *runner* beserta *engine*. Seluruh perintah bot akan dikirimkan oleh file main.java ini ke *engine*.

2.6. Alur Permainan Galaxio

Sebelum mulai mengembangkan *bot* hingga menjalankan permainan, ada beberapa prasyarat yang harus dipenuhi/diunduh sebelumnya. Hal-hal tersebut adalah:

- Java (minimal versi 11)
- NodeJS
- .NET Core 3.1
- .NET Core 5 (untuk menjalankan bot referensi)
- IDE (opsional)
- Apache Maven (apabila tidak menggunakan IntelliJ IDEA)

2.6.1. Menjalankan permainan Galaxio

Sebelum menjalankan permainan, ada beberapa hal yang bisa di konfigurasi, salah satunya adalah jumlah bot yang akan dimasukkan yang terdapat pada appsettings.json pada folder runner-publish dan appsettings.json pada folder engine-publish.

```
// ./engine-publish/appsettings.json

"RunnerConfig": {
    "BotCount": 4,
    "ComponentTimeoutInMs": 240000,
    "BotTimeoutInMs": 500000
}
```

```
// ./runner-publish/appsettings.json

{
  "RunnerUrl": "http://localhost",
  "RunnerPort": "5000",
  "BotCount": 4,
  "TickRate": 75,
  "MapRadius": 1000,
  "MapRadiusRatio": 250,
  "StartRadius": 300,
  "StartRadiusRatio": 75,
  "StartingPlayerSize": 10,
  "MinimumPlayerSize": 5,
  "ConsumptionRatio": {
    "1": 0.5,
    "3": 0.5,
    "7": 2
  },
}
```

Jumlah bot dapat diubah pada kedua file di bagian “BotCount”, kemudian ubahlah angkanya sesuai dengan keinginan.

Kemudian, untuk menjalankan permainan, bisa langsung mengeksekusi file run.sh yang sudah tersedia di root folder jika anda menjalankannya di platform *UNIX-based* OS. Apabila permainan dijalankan di Windows, maka anda dapat membuat file run baru bertipe *Windows Batch File* (misalnya run.bat) yang berisi:

```
// run.bat

@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 java -jar { lokasi target file .jar }
timeout /t 3
start java -jar { lokasi target file .jar }
timeout /t 3
start java -jar { lokasi target file .jar }
timeout /t 3
start java -jar { lokasi target file .jar }
cd ../

pause

```

2.6.2. Mengembangkan bot

Sesuai dengan deskripsi struktur *Game Engine*, pengembangan bot bisa dilakukan pada folder `./starter-bots/JavaBot/src/main/java`. Untuk mengimplementasikan algoritma atau alur berpikirnya bisa dilakukan pada file `BotService.java` yang ada di dalam folder `Services`. Berikut ini adalah file `BotService.java` awal yang sudah tersedia pada starter-pack *release v2021.3.2*.

```

// BotService.java
package Services;

import Enums.*;
import Models.*;

import java.util.*;
import java.util.stream.*;

public class BotService {
    private GameObject bot;
    private PlayerAction playerAction;

```

```

private GameState gameState;

public BotService() {
    this.playerAction = new PlayerAction();
    this.gameState = new GameState();
}

public GameObject getBot() {
    return this.bot;
}

public void setBot(GameObject bot) {
    this.bot = bot;
}

public PlayerAction getPlayerAction() {
    return this.playerAction;
}

public void setPlayerAction(PlayerAction playerAction) {
    this.playerAction = playerAction;
}

public void computeNextPlayerAction(PlayerAction playerAction) {
    playerAction.action = PlayerActions.FORWARD;
    playerAction.heading = new Random().nextInt(360);

    if (!gameState.getGameObjects().isEmpty()) {
        var foodList = gameState.getGameObjects()
            .stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.FOOD)
            .sorted(Comparator
                .comparing(item -> getDistanceBetween(bot,
item)))
            .collect(Collectors.toList());

        playerAction.heading = getHeadingBetween(foodList.get(0));
    }

    this.playerAction = playerAction;
}

```

```

public GameState getGameState() {
    return this.gameState;
}

public void setGameState(GameState gameState) {
    this.gameState = gameState;
    updateSelfState();
}

private void updateSelfState() {
    Optional<GameObject> optionalBot =
gameState.getPlayerGameObjects().stream().filter(gameObject ->
gameObject.id.equals(bot.id)).findAny();
    optionalBot.ifPresent(bot -> this.bot = bot);
}

private double getDistanceBetween(GameObject object1, GameObject
object2) {
    var triangleX = Math.abs(object1.getPosition().x -
object2.getPosition().x);
    var triangleY = Math.abs(object1.getPosition().y -
object2.getPosition().y);
    return Math.sqrt(triangleX * triangleX + triangleY * triangleY);
}

private int getHeadingBetween(GameObject otherObject) {
    var direction = toDegrees(Math.atan2(otherObject.getPosition().y -
bot.getPosition().y,
                                    otherObject.getPosition().x - bot.getPosition().x));
    return (direction + 360) % 360;
}

private int toDegrees(double v) {
    return (int) (v * (180 / Math.PI));
}

}

```

Pada file tersebut, sudah tersedia beberapa fungsi dan prosedur yang sudah bisa diimplementasikan. Prosedur `computeNextPlayerAction` akan memberikan *output* pada `PlayerAction`, di mana *bot* akan memutuskan apa

action yang akan dilakukan pada *tick* berikutnya. Di dalam prosedur tersebut, terdapat kondisi `if (!gameState.getGameObjects().isEmpty())`. Kondisi ini menyatakan bahwa kode di dalam blok *if* tersebut akan dijalankan apabila masih terdapat data bertipe Game Object dalam suatu *tick*. Artinya, permainan masih berjalan. Jadi, jika permainan sudah selesai, maka blok dalam kondisi tersebut tidak dijalankan.

Prosedur tersebut menerima parameter *playerAction* bertipe data *PlayerAction* dari kelas *PlayerAction* yang memiliki atribut *playerId(UUID)*, *action(PlayerActions)*, dan *heading (int)*. Untuk menetapkan *action* dari *bot*, kita cukup melakukan *assignment* pada variabel *action* yang merupakan atribut dari variabel parameter *playerAction*. Hal yang sama juga berlaku jika ingin menetapkan *heading* dari *bot*. Pada akhirnya, apabila seluruh algoritma sudah selesai, isi dari variabel parameter *playerAction* akan dimasukkan ke dalam *bot* pada perintah `this.playerAction = playerAction;`.

BAB III

APLIKASI STRATEGI GREEDY

3.1. Alternatif Solusi Greedy

3.1.1. Greedy by size

Greedy by size adalah pendekatan algoritma greedy dengan memprioritaskan penambahan size serta mempertahankan size dari bot. Salah satu cara untuk menambahkan size bot yang pertama ada food. Food terbagi menjadi dua jenis yaitu food dan superfood. Sedangkan untuk mempertahankan size bot dapat dilakukan dengan menghindari obstacle yang terdapat dalam game (seperti gas cloud, edge).

- a. Mapping Elemen Greedy
 - i. Himpunan kandidat : Semua Action yang tersedia.
 - ii. Himpunan solusi : Action yang terpilih.
 - iii. Fungsi solusi : Memeriksa apakah action yang dipilih menambahkan size terbanyak.
 - iv. Fungsi seleksi : pilih action yang menambahkan size yang paling efisien
 - v. Fungsi kelayakan : Memeriksa apakah action yang dipilih valid dan dapat dijalankan.
 - vi. Fungsi objektif : Mencari output penambahan size yang maksimum.

- b. Analisis Efisiensi Solusi

Terdapat dua action yang diperhatikan dalam pendekatan algoritma ini yaitu heading dan forward. Keduanya merupakan action yang digunakan untuk bot bergerak ke suatu arah. Untuk skema pencarian makanan yang paling efisien, program akan menjalankan quick sort untuk memasukkan data food yang terdapat dalam world kedalam suatu list secara terurut berdasarkan jarak terdekat. Begitu juga dengan obstacle, program akan menjalankan quick sort untuk memasukkan data gas cloud yang terdapat dalam world kedalam suatu list secara terurut berdasarkan jarak terdekat. Sehingga, list tersebut dapat dipakai untuk bot dapat menghindari obstacle tersebut ketika sudah mencapai jarak tertentu. Kompleksitas dari algoritma ini adalah :

$$T(n) = O(n \log n). \text{ (dengan } n = \text{jumlah elemen pada list)}$$

c. Analisis Efektivitas Solusi

Greedy by size berasal dari adanya elemen size yang menjadi kunci kemenangan untuk bisa mengalahkan bot-bot yang lain. Namun pendekatan algoritma ini bukan yang paling efektif dikarenakan terdapat banyak faktor lain yang dapat mempengaruhi size pada bot (torpedo, teleport, supernova).

Strategi ini efektif jika :

- Posisi bot relatif jauh dengan bot musuh yang lain
- Terdapat banyak food dalam area spawn bot.

Strategi ini tidak efektif jika :

- Posisi bot di kelilingi bot musuh yang lain.
- Sedikit food yang ada pada area spawn bot
- Bot musuh menggunakan strategi attack

3.1.2. Greedy by Attack

Greedy by attack adalah pendekatan algoritma greedy dengan memprioritaskan pemberian damage pada musuh dengan cara memakan, menembak, atau melakukan teleportasi ke lokasi musuh. Penyerangan yang berhasil dapat mengurangi atau bahkan mengeliminasi musuh dan menambahkan size dari bot.

a. Mapping Elemen Greedy

- i. Himpunan kandidat : Semua action yang tersedia
- ii. Himpunan solusi : Action yang terpilih
- iii. Fungsi solusi : Memeriksa action yang dipilih memiliki cost terkecil dan jarak musuh terdekat
- iv. Fungsi seleksi : Pilih action yang dilakukan dengan jarak terdekat.
- v. Fungsi kelayakan : Memeriksa apakah bot dapat melakukan action yang terpilih
- vi. Fungsi Objektif : Mencari output damage maksimum dengan cost minimum.

b. Analisis Efisiensi Solusi

Strategi ini memprioritaskan aksi-aksi seperti menembakkan torpedo, mengejar musuh yang lebih kecil dengan cara bergerak biasa, afterburner atau teleport. Sama seperti strategi sebelumnya strategi ini menggunakan quick sort untuk menemukan bot musuh terdekat dengan kompleksitas :

$$T(n) = O(n \log n). \text{ (dengan } n = \text{jumlah elemen pada list)}$$

c. Analisis Efektivitas Solusi

Greedy by attack berasal dari terdapatnya aksi untuk menyerang lawan yang bisa menambahkan size sendiri dan mengurangi size musuh. Tapi strategi ini belum cukup efektif karena dengan kita menembakkan torpedo, size kita juga mengurang jika peluru torpedo tidak mengenai musuh.

Strategi ini efektif jika :

- Bot musuh tidak menggunakan strategi Defense
- Jarak antara bot lain tidak jauh sehingga akurasi dari tembakan lebih besar

Strategi ini tidak efektif jika :

- Bot musuh menggunakan strategi Defense
- Jarak antara bot lain jauh

3.1.3. Greedy by Defense

Greedy by defense merupakan pendekatan algoritma *greedy* di mana bot akan memprioritaskan *action* yang mampu mempertahankan ukurannya dengan menghindari serangan-serangan musuh.

a. Mapping Elemen Greedy

- I. Himpunan kandidat : Semua *action* yang tersedia
- II. Himpunan solusi : *Action* yang terpilih
- III. Fungsi solusi : Memeriksa apakah *action* yang dipilih bisa menghindari kontak dari serangan-serangan musuh
- IV. Fungsi seleksi : pilih *action* yang mampu menghindari serangan musuh pada *tick* selanjutnya

- V. Fungsi kelayakan : memeriksa apakah ukuran *bot* mampu untuk melakukan *action* tersebut
- VI. Fungsi objektif : meminimumkan hingga meniadakan kemungkinan terkena *damage* akibat serangan musuh

b. Analisis Efisiensi Solusi

Strategi ini memprioritaskan aksi-aksi untuk melindungi diri dari serangan musuh. Aksi-aksi yang dapat dilakukan adalah Shield, menghindar dari peluru, menghindar dari musuh yang sizenya lebih besar. Strategi ini juga menggunakan quick sort untuk mendeteksi musuh terdekat dan peluru terdekat agar bisa mengaktifasi shield atau menghindar. Kompleksitas Strategi ini adalah :

$$T(n) = O(n \log n). \text{ (dengan } n = \text{jumlah elemen pada list)}$$

c. Analisis Efektivitas Solusi

Greedy by defense berasal dari adanya *action* lain dari musuh yang mampu mengurangi ukuran *bot*. Oleh karena itu, *bot* akan memprioritaskan aksi-aksi yang mampu untuk melindungi diri dan menghindari serangan-serangan musuh. Dengan menghindari pengurangan ukuran *bot* dari musuh, tidak menutup kemungkinan akan menang. Caranya bisa dengan menyalakan *shield* ataupun bisa juga dengan kabur, entah dari musuh ataupun dari *torpedo* lain. Namun, strategi ini tidak efektif karena adanya faktor lain yang mampu mengurangi ukuran *bot* seperti *gas cloud*, *edge of world*, ataupun

Strategi ini efektif jika :

- Tidak ada objek lain di sekitar yang mampu mengurangi ukuran *bot*

Strategi ini tidak efektif jika :

- Ada banyak *obstacle* lain di sekitar *bot* yang mampu mengurangi ukuran *bot*.
- Jarak antara *bot* lain jauh

3.2. Solusi Greedy yang Diterapkan dan Pertimbangannya

Berdasarkan solusi-solusi yang telah dijelaskan sebelumnya, terdapat kelebihan dan kekurangan dari masing-masing solusi. Dikarenakan pada setiap ronde game mempunyai situasi yang tidak tentu, maka kami menentukan *bot*

dapat mempertimbangkan pendekatan greedy yang sesuai dengan situasi yang sedang terjadi dan menentukan action apa yang akan dipilih.

Pada implementasi program bot kami, kami memutuskan untuk menggunakan alur pemrograman dengan urutan prioritas dari tertinggi hingga terendah sebagai berikut :

1. Mengambil dan menembakkan *supernova* kepada musuh dengan ukuran terbesar
2. Menghindar dari *edge* jika jarak terhadap bot dekat.
3. Mengaktifkan *shield* jika terdapat peluru datang ke arah bot.
4. Melarikan diri dari bot lain yang sizenya lebih besar.
5. Melakukan *teleport* menuju bot lain dengan size lebih kecil.
6. Mengejar bot lain dengan size lebih kecil.
7. Menembakkan *torpedo* kepada musuh jika kondisi diatas tidak memungkinkan.
8. Mendapatkan *food* atau *super food* terdekat jika tidak memasuki kondisi diatas.

Poin-poin diatas dibuat berdasarkan solusi-solusi greedy sebelumnya. Poin 2 dan 8 merepresentasikan *greedy by size*. Poin 3 dan 4 merepresentasikan *greedy by defend*. Poin 1, 5, 6, dan 7 merepresentasikan *greedy by attack*.

Prioritas tertinggi yang akan dipertimbangkan oleh bot adalah mengambil dan menembakkan *supernova*. Kemunculan objek *supernova* adalah kondisi yang sangat jarang untuk ditemukan dalam permainan. Objek ini hanya muncul 1 kali, yaitu di antara tahap awal dan akhir permainan. Selain itu, objek ini mampu memberikan *damage* yang cukup besar pada lawan, sehingga akan memberikan keuntungan bagi pemain. Kondisinya yang jarang dan sangat menguntungkan ini memberikan tempat sebagai prioritas tertinggi untuk dilakukan bot.

Bot akan selalu memprioritaskan untuk menghindar dari *edge* setelah *supernova*. Hal ini dilakukan agar bot tidak keluar dari map dan tidak mengurangi ukuran bot. Tidak ada hal yang bisa dilakukan apabila letak bot berada di luar *map*, karena seluruh objek permainan ada di dalam *map*.

Selain itu, bot juga memprioritaskan untuk mengaktifkan *shield* jika ada peluru yang datang ke arah bot. *Shield* aktif selama 10 *tick*. Artinya, *shield* aktif cukup lama tidak hanya untuk melindungi bot dari peluru pertama, namun juga rentetan peluru lainnya yang mungkin datang, mengingat setiap bot mampu menembak 5 peluru dalam waktu yang singkat. Selain untuk melindungi bot, *shield* juga mampu untuk melakukan serangan balik dengan memantulkan peluru yang dari lawan sehingga bisa kembali ke arah lawan dan mampu memberikan *damage* kepada lawan. Apabila tidak memiliki *shield*, maka bot akan menghindar dari *torpedo*.

Setelah prioritas mengaktifkan *shield*, bot akan memprioritaskan pula untuk melarikan diri dari bot yang lebih besar.

Setelah *greedy by defense*, bot akan melanjutkan prioritasnya ke menembak dan melakukan *teleport* pada bot yang lebih kecil. Kami mendapatkan bahwa aksi menyerang ini cukup efektif selama permainan. Dengan melakukan *teleport*, akan menjadi lebih mudah untuk mengejar bot yang lebih kecil bahkan mampu mengonsumsi lawan sehingga bisa mendapatkan penambahan ukuran yang cukup signifikan.

Apabila jarak cukup jauh dari lawan yang ingin diserang, ukuran bot cukup kecil, dan tidak ada *teleporter* yang tersimpan dalam bot, akan menjadi sulit untuk melakukan serangan ke lawan yang lebih kecil. Oleh karena itu, bot akan lebih memprioritaskan untuk mengejar terlebih dahulu lawan-lawan yang lebih kecil, baru mencoba untuk menembakkan *torpedo* ke arah lawan.

Kedua strategi *Greedy by defense* (poin 3 dan 4) diletakkan di atas *greedy by attack* (poin 5, 6, dan 7), karena strategi ini lebih mementingkan untuk mempertahankan ukuran, mengingat aksi pada *greedy by attack* juga memiliki kemungkinan mengurangi ukuran dari bot apabila gagal dieksekusi dengan baik.

Terakhir, aksi untuk mencari *food* ataupun *super food* ditempatkan sebagai prioritas terendah. Didapati bahwa keuntungan yang didapat juga tidak terlalu signifikan. Selain itu, *food* dan *super food* juga tersebar di berbagai tempat. Selama bergerak, kemungkinan untuk mendapatkan *food* ataupun *super food*

tetap besar walaupun bot sedang melakukan aksi lain. Aksi ini juga merupakan aksi *default* apabila kondisi-kondisi di atas tidak terpenuhi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi dalam *Pseudocode*

Procedure computeNextPlayerAction

Procedure yang berisi strategi algoritma bot

```
procedure computeNextPlayerAction(input/output : playerAction):  
  
    IF gameState.getGameObjects() is not empty THEN  
        IF tickCount ≠ gameState.getWorld().getCurrentTick() THEN  
            worldRadius ← gameState.getWorld().getRadius();  
            worldCenter ← new GameObject(UUID.randomUUID(), 0, 0, 0, new  
position(0, 0), null, 0, 0, 0, 0, 0);  
  
            playerAction.action ← PlayerActions.FORWARD;  
            playerAction.heading ← getHeadingBetween(worldCenter);  
  
            if (foodList.size() > 0) then  
                nearestFood ← foodList.get(0)  
                playerAction.heading ← nearestFood  
                playerAction.action = PlayerActions.FORWARD  
            Endif  
  
            if (playerList.size() > 0) then  
                nearestPlayer ← playerList.get(0)  
                if (isBotInSight(obstacleList, nearestPlayer) And bot size > 30  
And getDistanceBetween(bot, nearestPlayer) - bot.size - nearestPlayer.size  
< 500 And bot torpedoSalvoCount > 0) Then  
                    playerAction.heading ← getHeadingBetween(nearestPlayer)  
                    playerAction.action ← FIRETORPEDOES  
                    botAction ← "FIRING TORPEDO"  
                Endif  
                if (bot size - 20 > nearestPlayer size and distance between bot  
and nearestPlayer < 300 and bot shieldCount = 0 and bot torpedoSalvoCount =  
0) then  
                    playerAction.heading ← getHeadingBetween(nearestPlayer)  
                    playerAction.action ← FORWARD  
                    botAction ← "CHASING SMALLER PLAYER"  
                Endif
```

```

        if (bot size > 50 and bot size - 20 > nearestPlayer size and
bot teleporterCount > 0 and not isTeleported) then
            playerAction.heading ← getHeadingBetween(nearestPlayer)
            playerAction.action ← FIRETELEPORT
            botAction ← "FIRE TELEPORTER"
            isTeleported ← true
            teleportNear ← true
            teleportSmall ← false
            tickTeleport ← current game tick
        Endif
        if (bot size > 50 and bot size - 20 > smallestPlayer size and
bot teleporterCount > 0 and not isTeleported) then
            playerAction.heading ← getHeadingBetween(smallestPlayer)
            playerAction.action ← FIRETELEPORT
            botAction ← "FIRE TELEPORTER"
            isTeleported ← true
            teleportNear ← false
            teleportSmall ← true
            tickTeleport ← current game tick
        Endif
    Endif
    if (bot size < 30 and bot size < nearestPlayer size and distance
between bot and nearestPlayer - bot size - nearestPlayer size < 100 + bot
size) then
        playerAction.heading ← (getHeadingBetween(nearestPlayer) +
180)
        playerAction.action ← FORWARD
        botAction ← "RUNNING FROM BIGGER PLAYER"
    Endif

    if (torpedoList.size > 0 and a torpedo is coming towards the bot
and distance between bot and torpedo - bot size < 70 and bot shieldCount >
0 and bot size > 40) then
        playerAction.action ← ACTIVATESHIELD
        botAction ← "ACTIVATING SHIELD"
    Endif

    nearestPlayer = first item in playerList
    if (distance between bot and worldCenter + bot size + 100 >
worldRadius) then
        print "AVOIDING EDGE"

```

```

        playerAction.heading ← getHeadingBetween(worldCenter)
    Endif

    if (bot.size > nearestPlayer.size and isTeleported and current game
    tick - tickTeleport >= (distance between bot and nearestPlayer - bot.size -
    nearestPlayer.size) / 20 and teleportNear) then
        playerAction.heading ← getHeadingBetween(nearestPlayer)
        playerAction.action ← TELEPORT
        botAction ← "TELEPORTING"
        isTeleported ← false
        teleportNear ← false
        teleportSmall ← false
        tickTeleport ← gamestate.getWorld().getCurrentTick()
    Endif

    if (bot.size > smallestPlayer.size and isTeleported and current
    game tick - tickTeleport >= (distance between bot and nearestPlayer - bot
    size - nearestPlayer.size) / 20 and teleportNear) then
        playerAction.heading ← getHeadingBetween(nearestPlayer)
        playerAction.action ← TELEPORT
        botAction ← "TELEPORTING"
        isTeleported ← false
        teleportNear ← false
        teleportSmall ← false
        tickTeleport ← gamestate.getWorld().getCurrentTick()
    Endif

If (superNova.size > 0) Then
    output "SUPERNova DETECTED"
    If (bot.size > 50 And Not isTeleported And bot.teleporterCount > 0)
Then
    playerAction.heading ← getHeadingBetween(superNova())
    playerAction.action ← PlayerActions.FIRETELEPORT
    botAction ← "TELEPORT TO SUPERNova"
    superNovaTeleport ← True
    isTeleported ← True
    tickTeleport ← gameState.getWorld().getCurrentTick()
Endif
Endif
If (superNovaTeleport And superNova.size > 0) Then

```

```

If (isTeleported And gameState.getWorld().getCurrentTick() -
tickTeleport ≥ (getDistanceBetween(bot, superNova.get(0)) - bot.size -
superNova.get(0).size) / 20) Then
    playerAction.heading ← getHeadingBetween(superNova.get())
    playerAction.action ← PlayerActions.TELEPORT
    botAction = "TELEPORTING TO SUPERNOVA"
    isTeleported ← False
    teleportNear ← False
    teleportSmall ← False
    superNovaTeleport ← False
    tickTeleport ← gameState.getWorld().getCurrentTick()
Endif
Endif
If (superNovaFired And gameState.getWorld().getCurrentTick() -
tickSuperNova ≥ (getDistanceBetween(bot,biggestPlayer) - biggestPlayer.size
- bot.size) / 20) Then
    playerAction.heading ← getHeadingBetween(biggestPlayer)
    playerAction.action ← PlayerActions.DETONATESUPERNOVA
    botAction ← "DETONATING SUPERNOVA"
    superNovaFired ← False
Endif
If (bot.superNovaAvailable == 1 and getDistanceBetween(bot,
biggestPlayer) - bot.size - biggestPlayer.size > 150) Then
    playerAction.heading ← getHeadingBetween(biggestPlayer)
    playerAction.action ← PlayerActions.FIRESUPERNOVA
    botAction ← "FIRE SUPERNOVA"
    superNovaFired ← True
    tickSuperNova ← gameState.getWorld().getCurrentTick()
Endif
If (superNovaCloud.size > 0) Then
    If (getDistanceBetween(bot, superNovaCloud.get(0)) - bot.size -
superNovaCloud.get(0).size < 100) Then
        playerAction.heading ←
getHeadingBetween(superNovaCloud.get(0)) + 180
        playerAction.action ← PlayerActions.FORWARD
        botAction ← "RUNNING FROM SUPERNOVA"
    Endif
Endif

```

Function isWorldCenterClear

Function ini dibuat untuk memeriksa apakah center world tidak terdapat bot lain atau tidak.

```
Function isWorldCenterClear(List<GameObject> players, GameObject  
worldCenter) -> Boolean :  
    for Each player in players  
        if (getDistanceBetween(player, worldCenter) - player.size < 80)  
Then  
    Return False  
    Endif  
    Endfor  
    Return True  
endfunction
```

Function isBothInSight

Function ini berfungsi untuk memberi informasi apakah terdapat obstacle diantara bot dengan musuh

```
Function isBotInSight(obstacle, target) -> Boolean :  
    For Each obstacle in obstacles  
        If (getDistanceBetween(bot, obstacle) < getDistanceBetween(bot,  
target) And getHeadingBetween(obstacle) = getHeadingBetween(target)) Then  
            Return False  
        Endif  
    Endfor  
    Return True  
endfunction
```

Function isTorpedoComing

Function ini berfungsi untuk memberi informasi apakah terdapat peluru torpedo yang datang mengarah ke arah bot.

```
Function isTorpedoComing(GameObject torpedo) -> Boolean:  
    if (torpedo.getCurrentHeading() ≥ 0 And torpedo.getCurrentHeading() ≤  
90 And bot.getCurrentHeading() ≥ 180 And bot.getCurrentHeading() ≤ 270)  
Then  
    Return True  
    ElseIf (torpedo.getCurrentHeading() ≥ 90 And  
torpedo.getCurrentHeading() ≤ 180 And bot.getCurrentHeading() ≥ 270 And  
bot.getCurrentHeading() ≤ 360) Then  
    Return True  
    ElseIf (torpedo.getCurrentHeading() ≥ 180 And  
torpedo.getCurrentHeading() ≤ 270 And bot.getCurrentHeading() ≥ 0 And  
bot.getCurrentHeading() ≤ 90) Then  
    Return True  
    ElseIf (torpedo.getCurrentHeading() ≥ 270 And  
torpedo.getCurrentHeading() ≤ 360 And bot.getCurrentHeading() ≥ 90 And  
bot.getCurrentHeading() ≤ 180) Then  
    Return True  
    Else Then  
        Return False  
    Endif  
endfunction
```

4.2. Struktur Data Program

Dalam bot Galaxio yang dibuat dengan bahasa Java, kami menerapkan pendekatan pemrograman berbasis objek dengan mengenkapsulasi data dalam kelas-kelas. Selama proses pembuatan bot, kami menambahkan beberapa kelas dan struktur data untuk melengkapi implementasi program. Program Galaxio terdiri dari lima bagian utama, yakni kelas-kelas, Enums, Models, services, dan Main.

a. Enums

Pada bagian Enums, tiap-tiap kelasnya terpisah dalam masing-masing file.

Folder Enums ini merepresentasikan nilai-nilai konstanta yang terdapat pada game Galaxio. Kelas-kelas tersebut adalah :

- ObjectTypes

Kelas ObjectTypes berisi semua objek yang terdapat pada game tersebut yaitu

- PLAYER
- FOOD
- WORMHOLE
- GAS_CLOUD
- ASTEROID_FIELD
- TORPEDO_SALVO
- SUPER_FOOD
- SUPERNOVA_PICKUP
- SUPERNOVA_BOMB
- TELEPORTER
- SHIELD

- PlayerActions

Kelas PlayerActions berisi aksi-aksi yang tersedia untuk bot yaitu

- FORWARD
- STOP
- STARTAFTERBURNER
- STOPAFTERBURNER
- FIRETORPEDOES
- FIRESUPERNOVA
- DETONATESUPERNOVA
- FIRETELEPORT
- TELEPORT
- ACTIVATESHIELD

b. Models

Dalam folder models, terdapat kelas-kelas yang berisi entitas yang terdapat pada permainan Galaxio beserta dengan atribut yang dimilikinya. Kelas - kelas tersebut antara lain :

- GameObject

Kelas GameObject merupakan kelas yang berisikan atribut-atribut Dimiliki oleh bot dan objek-objek pada permainan Galaxio.

Atribut-atribut tersebut adalah :

- Id
- size
- speed
- currentHeading
- position
- gameObjectType
- Effects
- torpedoSalvoCount
- superNovaAvailable
- teleporterCount
- shieldCount

- GameState

Kelas GameState merupakan kelas yang berisikan atribut-atribut yang melambangkan kondisi pada permainan Galaxio setiap kali permainannya dimainkan. Atribut-atribut tersebut adalah :

- World
- List<GameObject> gameObjects
- List<GameObject> playerGameObjects

- GameStateDto

Kelas GameStateDto merupakan kelas yang berisikan atribut-atribut yang melambangkan kondisi pada permainan Galaxio setiap kali permainannya dimainkan. Atribut-atribut tersebut adalah :

- World
- List<GameObject> gameObjects
- List<GameObject> playerGameObjects

- PlayerAction

Kelas PlayerAction merupakan kelas yang berisikan atribut-atribut yang melambangkan kondisi player dalam permainan Galaxio. Atribut-atribut tersebut adalah :

- playerId
- action

- heading

- Position

Kelas Position merupakan kelas yang berisikan atribut-atribut yang melambangkan letak posisi pada permainan Galaxio. Atribut-atribut tersebut adalah :

- x
- y

- World

Kelas World merupakan kelas yang berisikan atribut-atribut dari world pada permainan Galaxio. Atribut-atribut ini melambangkan kondisi dari world tersebut. Atribut-atribut tersebut adalah :

- centerPoint
- radius
- currentTick

c. Services

Hanya terdapat satu kelas dalam folder Services yaitu services. Kelas ini berisi implementasi dari bot. Kelas ini terdapat atribut-atribut yang dapat memudahkan jalannya program dari bot itu sendiri. Terdapat atribut yang sudah sediakan oleh starter-pack permainannya yaitu bot, playerAction, gameState, dan botAction. Kami juga menambahkan beberapa atribut untuk mempermudah keberjalanannya program diantaranya yaitu :

- tickCount

Atribut ini berfungsi untuk menandakan tick sekian yang berlangsung pada ronde permainan.

- isTeleported

Atribut ini berfungsi untuk memeriksa apakah bot sudah menembakan teleporter atau belum.

- teleportNear

Atribut ini berfungsi untuk menandakan kondisi mana aksi bot menembakan teleporter

- teleportSmall

Atribut ini berfungsi untuk menandakan kondisi mana aksi bot menembakan teleporter

- tickTeleport
Atribut ini berfungsi untuk menyimpan tick saat bot menembakkan teleporter
- tickSuperNova
Atribut ini berfungsi untuk menyimpan tick saat bot menembakkan supernova
- superNovaTeleport
Atribut ini berfungsi untuk memeriksa apakah bot sudah menembakan teleporter menuju supernova atau belum.
- superNovaFired
Atribut ini berfungsi untuk memeriksa apakah bot sudah menembakan supernova atau belum

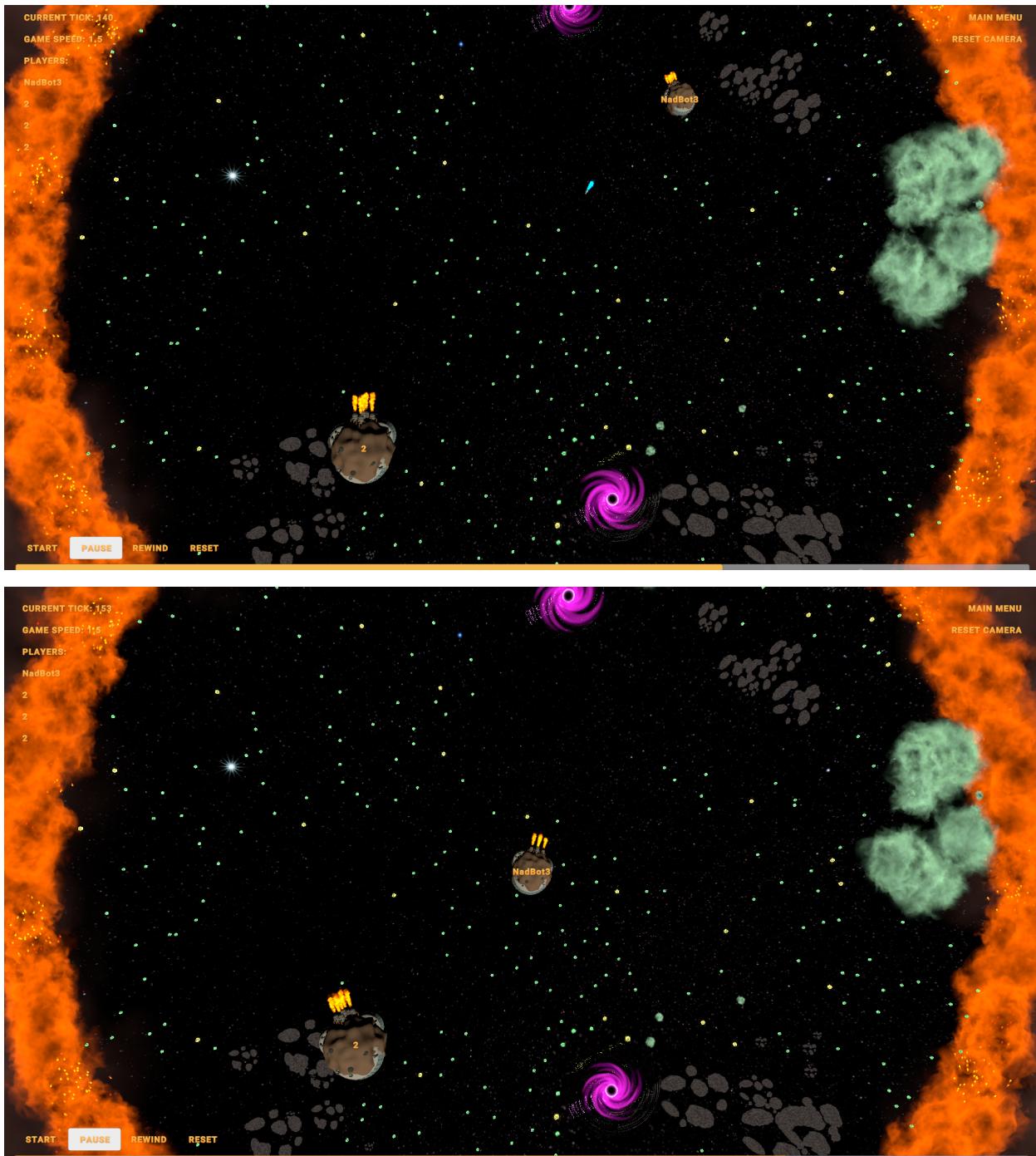
d. Main

Pada bagian Main, terdapat program inisialisasi bot saat dipanggil dari executable. Terdapat juga program inisialisasi yang berkaitan dengan game state dan penerjemahan action dari bot menuju *game engine*.

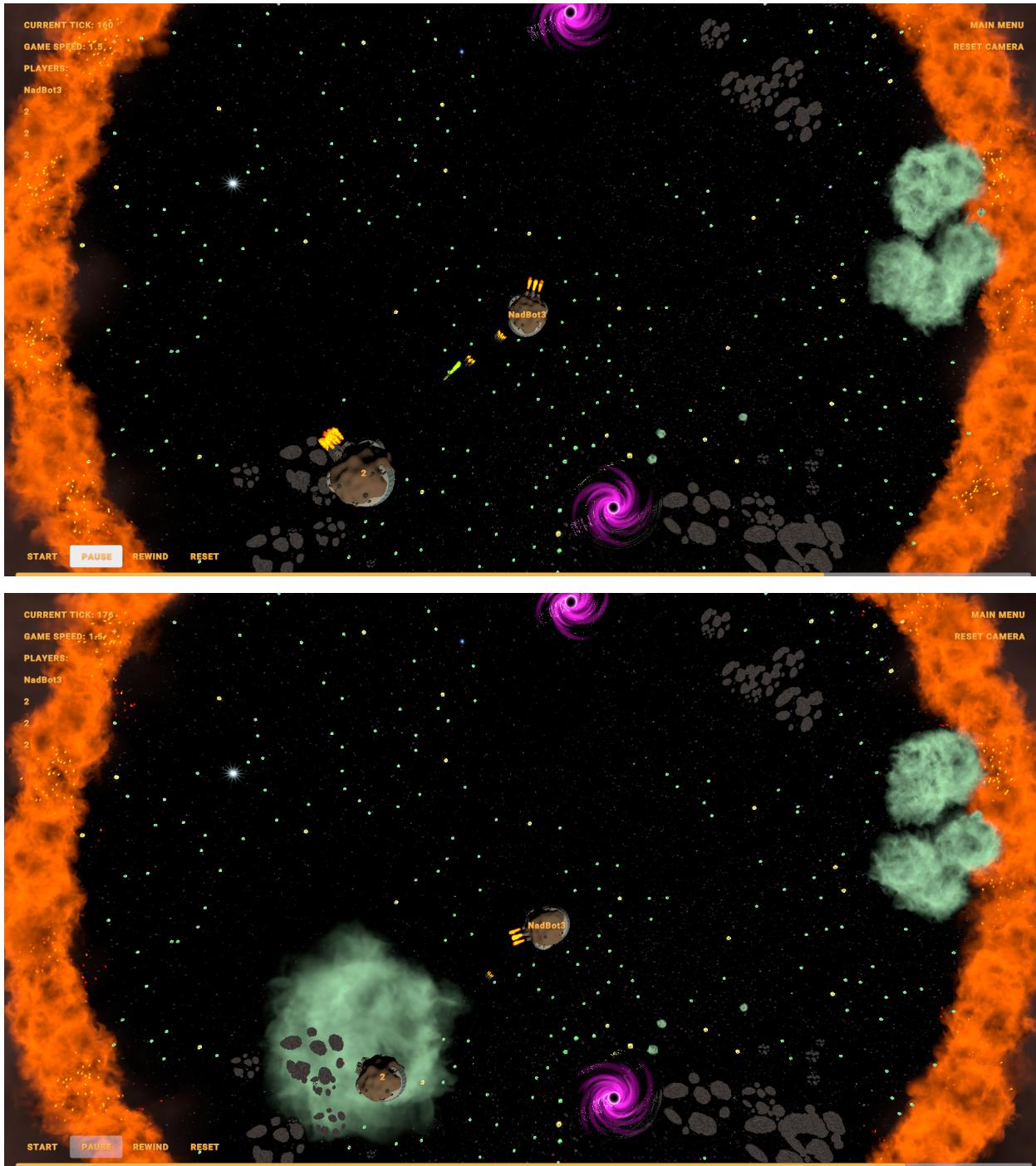
4.3. Analisis dan Pengujian

Kami melakukan analisis dan pengujian dengan cara menjalankan permainan dengan memasukkan bot kami dengan bot kami sendiri dan reference bot yang telah kami sedikit kembangkan. Setelah kami menjalankan permainan, kami melakukan pengamatan terhadap *behaviour* atau semua aksi-aksi yang bot kami lakukan menggunakan *visualizer* yang tersedia.

Berikut adalah analisis yang kami lakukan terhadap suatu pertandingan bot kami.

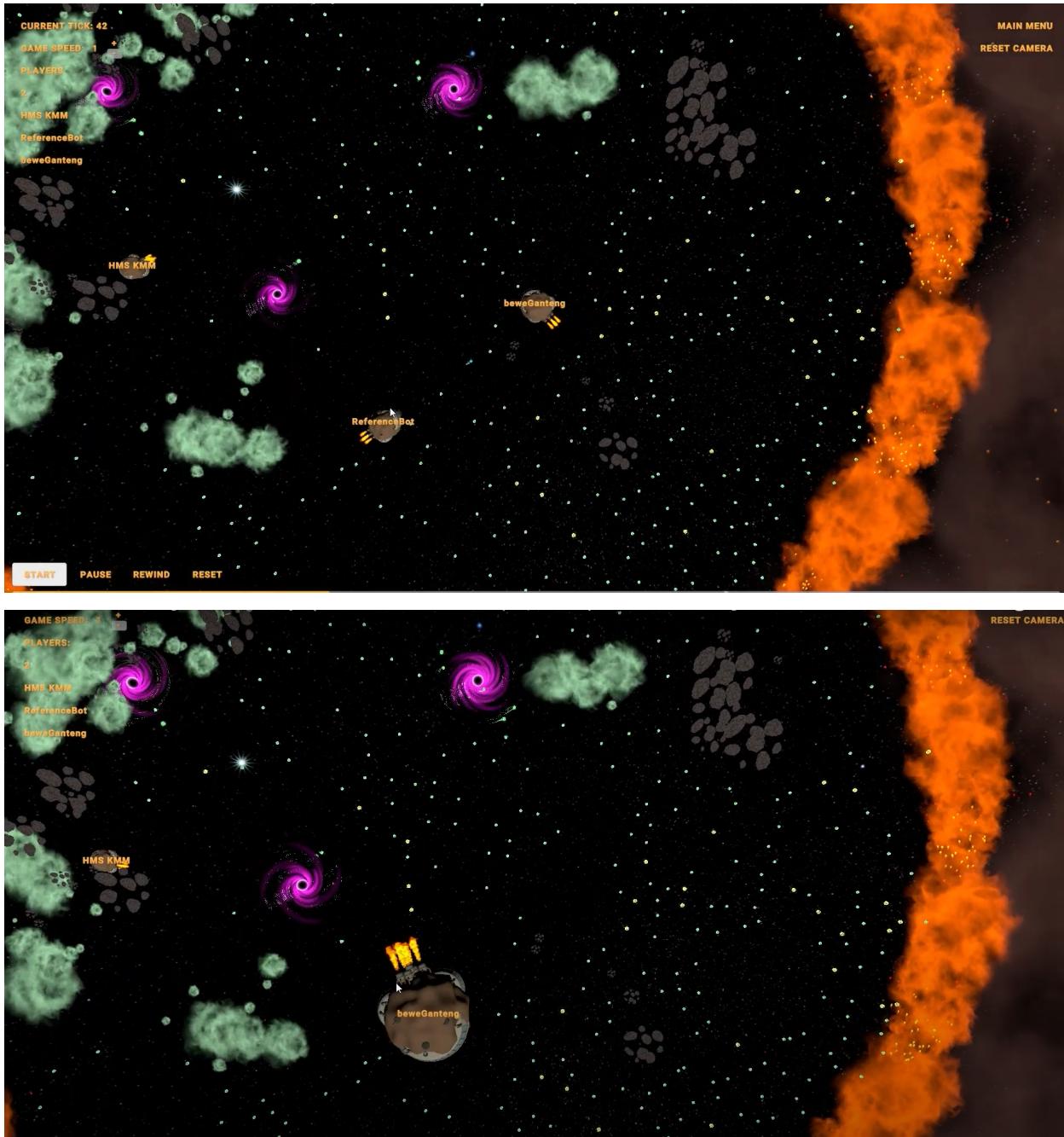


Pada tick 140 dan 152, terdapat supernova yang berada di *world center*. Disini kita bisa melihat bahwa bot mengganti prioritas dari mencari food menjadi teleport untuk mengambil supernova tersebut.

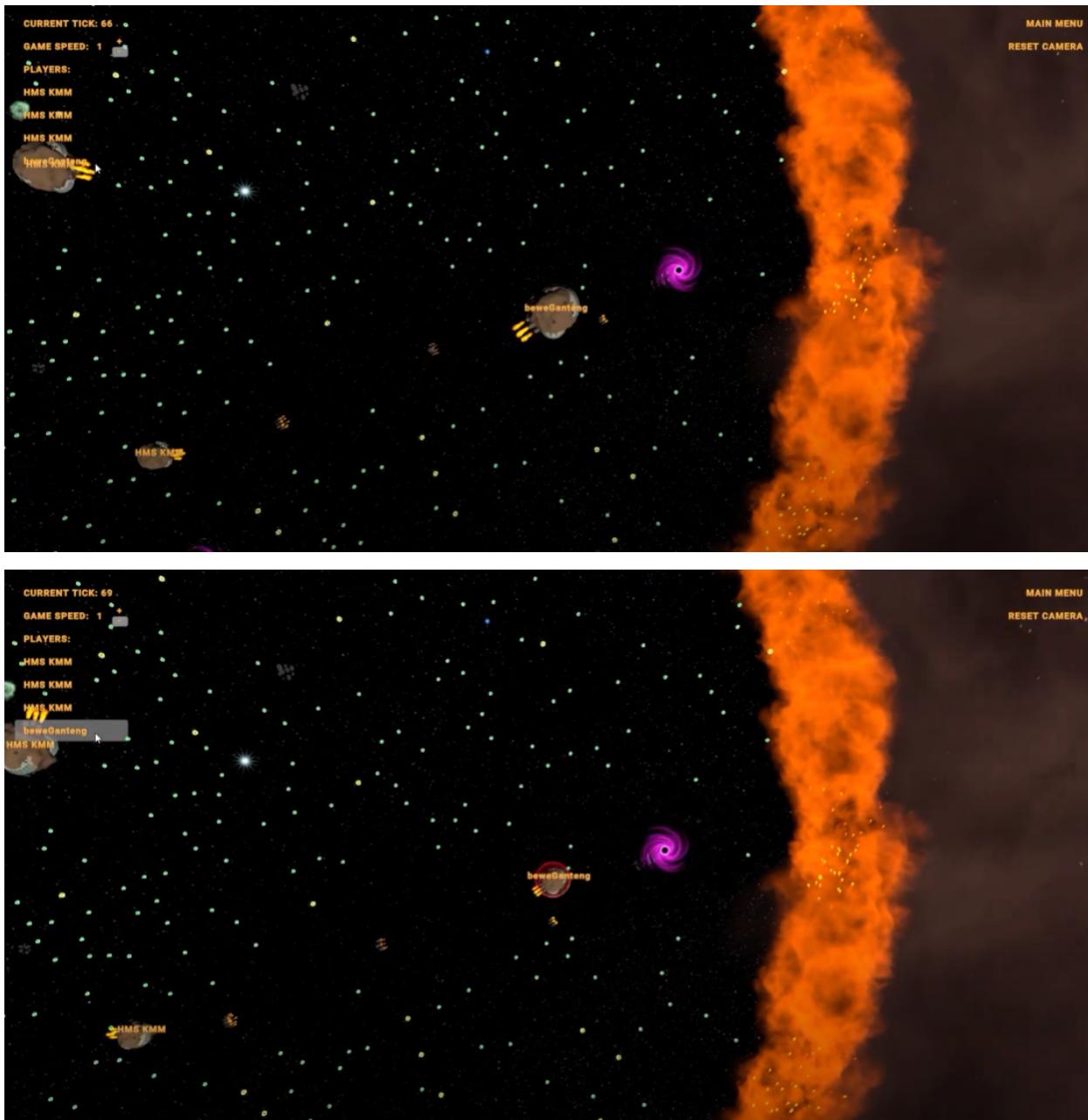


Pada tick 160 dan 176, setelah bot mengambil supernova, bot tersebut menembakan supernova kepada bot musuh dengan size paling besar. Setelah peluru supernova telah melewati distance yang telah ditentukan sesuai rumus, supernova diledakkan dan mengenai musuh. Pada tick ini kita juga bisa melihat setelah bot meledakkan supernova dan torpedoCount kosong, bot akan kembalikan prioritas menjadi pencarian food. Behavior ini merupakan

implementasi dari solusi *Greedy by attack* karena memprioritaskan melakukan *damage* kepada lawan.



Pada tick 42 dan 55, kita bisa melihat bot menembakan peluru teleporter terhadap musuh terdekat karena sudah memasuki kondisi yaitu size dari bot kita lebih besar daripada size bot musuh. *Behaviour* ini juga merupakan implementasi dari solusi *greedy by attack* dan juga *greedy by size* karena memprioritaskan memakan lawan dan menambah size bot.



Pada *tick* 66, kita bisa melihat bahwa terdapat peluru torpedo yang bergerak menuju bot. Pada *tick* 69, bot mengubah prioritas aksi dari mencari food dengan mengaktifasi shield untuk menghindari *damage* yang diberikan dari torpedo. *Behaviour* ini merupakan implementasi dari solusi *greedy by defense*.

BAB V

PENUTUP

5.1. Kesimpulan

Dengan ini, kami berhasil membuat strategi bot untuk permainan Galaxio dengan pendekatan algoritma greedy. Program akhir yang kami buat merupakan hasil dari penggabungan beberapa solusi greedy yang tersedia dan sudah kami uji untuk menghasilkan strategi terbaik versi penulis.

5.2. Saran

Saran untuk *developer* lain yang akan mencoba mengembangkan *bot* Galaxio:

- Gabungkan solusi-solusi algoritma Greedy yang tersedia agar mampu menciptakan strategi bot yang efektif di setiap kondisi

Saran untuk penurunan tugas besar selanjutnya :

- Pada saat awal penurunan tugas besar ini, terdapat informasi yang kurang sehingga kami kebingungan untuk memulai pengujian dan menganalisis strategi.

5.3. Refleksi

Refleksi untuk kelompok kami di antaranya :

- Strategi algoritma yang telah kami buat dapat dikembangkan kembali efektivitas strateginya agar mendapat strategi yang lebih baik.
- Pembagian tugas kelompok dapat dilakukan lebih baik agar pembagian waktu tugas lebih merata.

LAMPIRAN

Tautan *Repository Github* :

https://github.com/IceTeaXXD/Tubes1_beweGanteng

Tautan *video penjelasan strategi bot* :

<https://www.youtube.com/watch?v=Na2bnxRj6lw>

DAFTAR PUSTAKA

Gregory, J. (2018). *Game Engine Architecture, Third Edition*. CRC Press LLC.

Rinaldi, M. (n.d.). *Algoritma Greedy (2021) Bag1*.

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

Rinaldi, M. (n.d.). *Algoritma Greedy (2021) Bag2*.

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

Rinaldi, M. (n.d.). *Algoritma Greedy (2021) Bag3*.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf)