

LAPORAN TUGAS BESAR 1

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”



Disusun Oleh:
Kelompok 33

Anggota Kelompok:

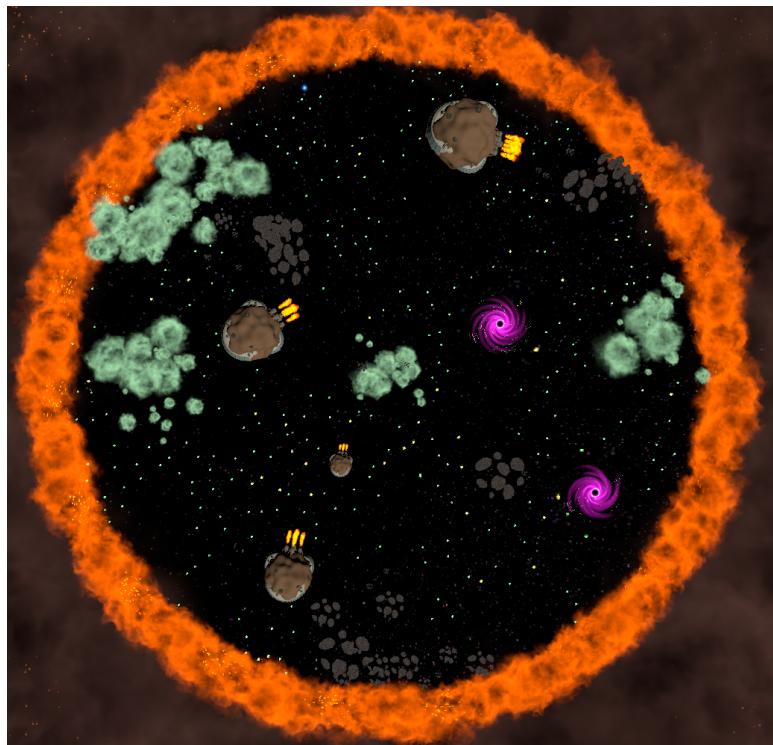
Ezra Maringen Christian Mastra Hutagaol - 13521073
Christian Albert Hasiholan - 13521078
Tobias Natalio Sianipar - 13521090

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
FEBRUARI 2023

Bab 1

Deskripsi Tugas

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



Gambar 1. Ilustrasi permainan *Galaxio*

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>

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 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 *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 melewatkannya 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 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 mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa *command* yang dapat dilakukan oleh *player*. Setiap tick, *player* hanya dapat memberikan satu *command*. Untuk daftar *commands* yang tersedia, bisa merujuk ke tautan [panduan](#) di spesifikasi tugas
11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal *player* lain, maka score bertambah 10, jika mengonsumsi *food* atau melewati *wormhole*, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan score tertinggi.

Adapun peraturan yang lebih lengkap dari permainan *Galaxio*, dapat dilihat pada laman :

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

Bab 2

Landasan Teori

A. Algoritma Greedy

Algoritma greedy adalah algoritma yang membentuk solusi langkah per langkah dengan mencari solusi terbaik lokal pada setiap langkahnya. Pada kebanyakan kasus, algoritma greedy tidak menghasilkan solusi yang paling optimal, namun menemukan solusi yang mendekati optimum dengan waktu yang cukup cepat. Algoritma Greedy memiliki prinsip “take what you can get now” artinya adalah algoritma akan mengambil solusi yang optimal pada langkah tersebut tanpa mempertimbangkan konsekuensinya pada langkah berikutnya.

Elemen- elemen algoritma greedy :

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah (misal : simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
 2. Himpunan solusi, S : berisi kandidat yang sudah dipilih.
 3. Fungsi solusi : menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
 4. Fungsi seleksi (selection function) : memilih kandidat berdasarkan strategi greedy tertentu.
- Strategi greedy ini bersifat heuristik
5. Fungsi kelayakan (feasible) :
 6. Fungsi Obyektif : memaksimumkan atau memminimumkan

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat C yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dipotimasi oleh fungsi obyektif.

B. Permainan Galaxio

Dalam permainan Galaxio akan terdapat beberapa bot kapal yang bertanding secara battle royale dengan beberapa bot kapal lain. Command yang tersedia berupa perlakuan pada bot kapal. Bot kapal terakhir yang bertahan adalah bot kapal yang akan jadi pemenang dari permainan ini.

The Map

Peta permainan akan berbentuk kartesius yang memiliki arah negatif dan positif. Peta hanya menangani angka bulat. Pusat peta adalah 0.0 dan ujung dari peta merupakan radius. Jumlah ronde maksimum pada game sama dengan ukuran radius. Ukuran peta akan mengecil seiring batasan peta mengecil.

Pada peta, akan terdapat 5 objek yaitu:

- Players : Pemain dan musuh
- Food : objek kecil yang dibutuhkan agar kapal bisa membesar
- Wormholes : 2 titik pada peta yang terhubung dari kedua arah
- Gas Clouds : Zona berbahaya yang akan merusak bot kapal jika dilewati oleh bot kapal

- Asteroid Fields : Zona berbahaya yang akan memperlambat bot kapal jika dilewati oleh bot kapal

1. Visibility

Peta akan selalu terlihat dan akan mengecil selagi batas peta berkurang

2. Boundary

Ukuran dari peta akan mengecil pada setiap game tick. Object yang terjatuh di luar peta game tick berakhir akan dihilangkan dari peta. Hal ini tidak berlaku untuk player, berada diluar peta akan membuat ukuran player berkurang sebesar 1 pada setiap game tick

Objects

Semua Objek direpresentasikan oleh bentuk melingkar dan memiliki titik pusat dengan X dan Y sebagai koordinat

1. Food

Food akan tersebar pada peta dengan ukuran 3 yang bisa dikonsumsi oleh player.

- Food tidak akan bergerak
- Food akan menghilang jika terjatuh di luar peta
- Jika player menabrak food maka player akan memakan seluruh *food* tersebut dan *player* akan bertambah besar sebesar size food

2. Super Food

Ketika food muncul di peta, food memiliki kemungkinan untuk memunculkan super food.

Super food akan meningkatkan “absorption rate” dari player.

3. Wormholes

Wormholes akan ada sepasang pada peta, sehingga memungkinkan player untuk masuk dari suatu sisi dan akan keluar di sisi yang satu lagi. Wormholes akan bertambah besar ketika game tick dibuat menjadi maksimum. Wormhole akan mengecil menjadi setengah dari ukurannya ketika ada kapal yang melalui wormhole tersebut.

4. Gas Clouds

Gas Clouds akan tersebar di sekitar peta menyatu menjadi clouds yang lebih kecil. Kapal bisa melewati *gas clouds* tetapi akan membuat ukuran kapal berkurang sebesar 1 untuk setiap game tick

5. Asteroid Fields

Asteroid fields akan tersebar di sekitar peta menyatu menjadi clouds yang lebih kecil. Kapal bisa melewati asteroid fields tetapi akan memperlambat kapal sebesar faktor dari 2.

6. Torpedo Salvo

Torpedo Salvo akan mengurangkan size dari object yang ditabrak sebesar size torpedo salvo

- Torpedo Salvo bergerak secara lurus

- Torpedo Salvo bergerak dengan speed 60
- Torpedo Salvo memiliki size 10
- Torpedo Salvo akan terus bergerak selama mereka masih memiliki size

7. Supernova

Supernova merupakan senjata yang sangat kuat. Supernova hanya muncul sekali dalam permainan. Supernova biasanya muncul di antara quarter pertama dan quarter terakhir permainan. Pemain yang mengambil supernova bisa menembakkan supernova bomb. Supernova bomb bergerak seperti torpedo, tapi tidak menabrak apapun ketika bergerak. Pemain yang menembakkan supernova bomb bisa memberi command untuk meledakkan supernova bomb, menyebabkan damage yang sangat besar kepada pemain yang berada di zona ledakan.

8. Teleport

Pemain bisa meluncurkan teleporter ke suatu arah di peta. Teleporter bergerak dengan speed 20 dan tidak akan menabrak apapun. Pemain bisa menggunakan command untuk teleport ke lokasi teleporter mereka. Teleporter akan hancur jika sampai ujung map. Menggunakan teleporter akan membuat size berkurang sebanyak 20

The Ship

Bot akan bermain sebagai sebuah kapal luar angkasa, yang akan memakan objek planet dan kapal lain untuk bisa membesar. Berikut merupakan settingan awal kapal :

- Speed : kapal akan bergerak dengan X posisi pada setiap game tick, dimana x merupakan speed dari kapal. Mula - mula speed kapal akan dimulai dari 20 dan akan berkurang ketika kapal mulai membesar
- Size : size dari ship akan dimulai dengan radius 10
- Heading : kapal akan bergerak dengan arah heading, diantara 0 sampai 359 derajat

1. Speed

Speed menentukan seberapa cepat kapal bisa bergerak pada setiap game tick. Kapal tidak akan bergerak ketika stopped atau command FORWARD tidak diberikan, tetapi nilai dari speed kapal akan tetap sama. Speed berbanding terbalik dengan ukuran kapal. Semakin besar kapal maka semakin lambat speed kapal. Speed ditentukan dengan rumus berikut :

$$\begin{aligned} \text{speedRatio} &= 200 \\ \text{speedRatio}/\text{bot.size} &= \text{speed} \end{aligned}$$

Hasil dibulatkan ke atas dengan minimal 1.

2. Afterburner

Afterburner akan meningkatkan speed kapal sebesar faktor 2. Namun, afterburner juga akan membuat size kapal berkurang sebanyak 1 setiap tick. Afterburner akan menghancurkan kapal sendiri jika size dari ship kurang dari 5

3. Dark Matter Torpedoes

Dark matter torpedoes bisa digunakan untuk memanipulasi space di sekitar kapal. Torpedoes yang mengenai object di world akan menghancur object tersebut. Ketika torpedoes mengenai musuh, maka kapal akan mencuri matter dari musuh proporsional dengan damage yang diberikan ke musuh

4. Shield

Untuk bertahan kapal bisa mengaktifkan shield. Ketika shield dinyalakan, shield bisa mementalkan torpedo lawan. Ketika diaktifkan, shield bisa bertahan selama 20 tick dan akan menggunakan size sebesar 20.

Collisions

Collision terjadi ketika ada 2 objek tumpang tindih setidaknya 1 unit world space. Ini berarti object bisa menyentuh satu sama lain, tetapi ketika mereka tumpang tindih maka akan dianggap sebagai bertabrakan dan mekanisme bertabrakan akan berlaku

1. Ship to Ship Collisions

Ketika kapal pemain bertabrakan, kapal yang memiliki size lebih besar akan mengonsumsi kapal yang lebih kecil dengan rate 50% dari size kapal yang lebih besar hingga maksimum sebesar kapal yang lebih kecil. Setelah terjadi ship to ship collisions, heading kapal akan dibalik 180 derajat dan akan terpisah sejauh 1 unit world space serta akan bergerak maju dengan arah yang baru.

2. Food Collision

Ketika kapal menabrak food particle, kapal akan mengonsumsi semua makanan itu dan akan meningkatkan size kapal sebesar size food yang ditabrak.

GameObjects

GameObjects berisi semua object yang ada di peta. Isi dari GameObjects seperti berikut :

- Size : radius dari object
- Speed : kecepatan object bisa bergerak
- Heading : arah dari object
- GameObjectType : tipe dari object seperti,
 1. Food
 2. Wormhole
 3. Gas Cloud
 4. Asteroid Field
- X Position : posisi X pada bidang kartesius
- Y Position : posisi Y pada bidang kartesius

PlayerObjects

PlayerObjects berisi semua object pemain yang ada di peta. Isi dari PlayerObjects seperti berikut:

- Size : radius dari object

- Speed : Kecepatan object bisa bergerak
- Heading : arah dari object
- GameObjectType : tipe dari object seperti,
 1. Player
- X Position : posisi X pada bidang kartesius
- Y Position : posisi Y pada bidang karterius
- Active Effects : Bitwise effect yang sedang memengaruhi bot, direpresentasikan oleh :
 - 0 = No Effect
 - 1 = Afterburner active
 - 2 = Asteroid Field
 - 4 = Gas Cloud

The Commands

Dalam setiap game tick pemain hanya bisa memberikan 1 command kepada kapal.

Command yang bisa digunakan antara lain :

1. FORWARD : Command ini akan menggerakkan kapal
2. STOP : Command ini akan memberhentikan kapal yang sedang bergerak
3. START_AFTERCLOUD : Command ini akan mengaktifkan afterburner kapal
4. STOP_AFTERCLOUD : Command ini akan memberhentikan afterburner kapal
5. FIRE_TORPEDOES : Command ini akan mengonsumsi 1 salvo charge dan 5 size untuk menembakkan salvo torpedoes ke arah yang diinginkan
6. FIRE_SUPERNOVA : Command ini akan mengonsumsi 1 supernova pickup dan menembakkan supernova ke arah yang diinginkan
7. DETONATE_SUPERNOVA : Command ini akan meledakkan supernova yang ada
8. FIRE_TELEPORTER : Command ini akan mengonsumsi 1 teleport charge dan 20 size untuk menembakkan teleporter ke arah yang diinginkan
9. TELEPORT : Command ini akan melakukan teleport ke teleporter yang ada jika ada teleporter
10. USE_SHIELD : Command ini akan menggunakan shield untuk menangkis serangan torpedoes jika pemain punya shield

Cara Kerja Program

1. Runner –saat dijalankan– akan meng-host sebuah *match* pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
4. Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar *match* dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut BotCount yang dimiliki file JSON “appsettings.json”. File tersebut terdapat di dalam folder “runner-publish” dan “engine-publish”.

5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah ReceiveGameState karena memberikan status game.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi *match*.

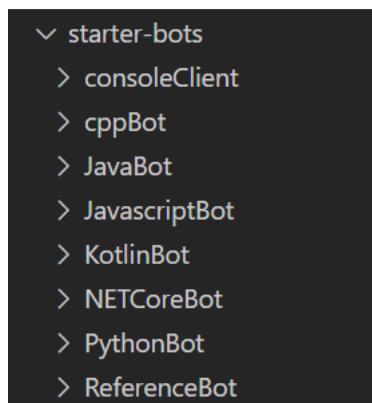
Cara Menjalankan Game

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.

Starter bot

Starter pack menyediakan *starter bot* sebagai titik awal dari pengembangan bot milik kita sendiri. Starter bot sudah dibekali kode yang dibutuhkan oleh bot untuk melakukan koneksi dengan runner sehingga kita dapat fokus kepada strategi permainan. Gambar dibawah menunjukkan bahwa starter-pack memberikan beberapa pilihan bahasa untuk starter bot. Namun pada tugas besar ini, kita akan menggunakan bot yang berbahasa Java.



Untuk memulai, silahkan salin folder JavaBot (atau tidak usah, terserah kalian) dan beri nama bot kalian dengan cara mengubah string “Coffee Bot” pada file Main.java baris ke-61 menjadi apapun terserah kalian.

Visualizer

Cara menjalankan permainan melalui visualizer:

1. Lakukan ekstrak pada file zip Galaxio dalam folder “visualiser” sesuai dengan OS kalian
2. Jalankan aplikasi Galaxio
3. Buka menu “Options”
4. Salin path folder “logger-publish” kalian pada “Log Files Location”, lalu “Save”
5. Buka menu “Load”
6. Pilih *file* JSON yang ingin diload pada “Game Log”, lalu “Start”
7. Setelah masuk ke visualisasinya, kalian dapat melakukan *start*, *pause*, *rewind*, dan *reset*
8. Silahkan buat bot terbaik kalian dan selamat menikmati permainan 😊

Bab 3

Aplikasi Strategi Greedy

A. Elemen - Elemen Algoritma Greedy Pada Permainan Galaxio

Dalam permainan Galaxio, pemain akan bertanding menggunakan bot kapal untuk bertanding dengan beberapa bot kapal lain secara battle royale. Setiap pemain yang bermain galaxio akan memiliki sebuah bot kapal untuk dipertandingkan. Pemenang dari game ini adalah pemain dengan bot kapal yang bisa tetap bertahan hidup hingga akhir permainan. Berikut merupakan elemen greedy dalam permainan galaxio :

a. Himpunan Kandidat :

Himpunan perintah yang dapat dilakukan yaitu,

- FORWARD
- STOP
- START_AFTERBURNER
- STOP_AFTERBURNER
- FIRE_TORPEDOES
- FIRE_SUPERNOVA
- DETONATE_SUPERNOVA
- FIRE_TELEPORTER
- TELEPORT

b. Himpunan Solusi :

himpunan perintah yang terpilih dan akan dilakukan

c. Fungsi Solusi :

Memeriksa apakah perintah pada himpunan kandidat dapat dijalankan dengan baik

d. Fungsi Seleksi :

Memilih perintah terbaik untuk digunakan dalam permainan

e. Fungsi kelayakan :

Memeriksa apakah perintah yang dipilih dapat dilakukan

f. Fungsi Objektif :

Menangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup

B. Alternatif Solusi Greedy

1. Greedy by profit

Pada strategi ini, dalam memilih perintah terbaik, bot kapal akan mempertimbangkan weight terberat untuk setiap area. Setelah masuk ke dalam area dengan weight terberat, maka bot kapal akan bergerak ke arah food terdekat.

Algoritma ini akan membagi peta menjadi 12 bagian berdasarkan weight. Setelah membagi peta menjadi 12 bagian, bot kapal akan bergerak menuju area yang memiliki weight terberat. Setelah memasuki area tersebut maka bot kapal akan bergerak menuju food terdekat dari bot kapal tersebut. Efisiensi algoritma ini $O(n^2)$ karena dalam algoritma ini metode greedy yang kami gunakan membuat dua loop. Loop pertama digunakan untuk membagi objek-objek di sekitar ship menjadi beberapa area. Kemudian, loop kedua digunakan untuk memberi value pada tiap objek yang terdapat pada area tersebut sehingga hasilnya akan menjadi weight dari area tersebut dan akan dibandingkan dengan area lain untuk ditemukan jalan terbaik

Efektivitas dari algoritma ini cukup baik karena bot akan mencari food terdekat dan semakin banyak food yang dimakan maka akan semakin besar ukuran bot kapal

2. GreedybyHorde

Pada strategi ini, bot akan mencari food terdekat dengan mempertimbangkan jarak food-food tersebut.

Algoritma ini dilakukan dengan cara bot akan mencari 30 food terdekat pertama pada bot. Untuk setiap food terdekat pertama itu akan dicari lagi 20 food terdekat kedua yang berdekatan dengan setiap food terdekat pertama. Setelah itu total jarak food terdekat kedua dan jadikan total itu sebagai weight. Bot akan mengarah ke weight terberat. Efisiensi algoritma ini $O(N^2)$ karena dalam algoritma ini menggunakan double loop

Efektivitas bot ini sangat baik karena bot akan mencapai daerah dengan sekumpulan makanan yang paling banyak

C. Strategi Greedy Terbaik yang Dipilih

Dari beberapa alternatif greedy yang ada kelompok kami memilih menggunakan algoritma greedy by profit. Kelompok kami tidak memilih algoritma GreedybyHorde karena pada greedybyhorde ketika bot menemukan dua tempat yang memiliki weight yang sama bot tidak bisa menentukan harus pergi ke tempat yang mana

Algoritma yang digunakan dalam mengimplementasikan strategi greedy yang dipilih yaitu greedy by profit ialah sebagai berikut

1. BestHeading

Fungsi ini akan membagi area di sekitar bot menjadi 12, kemudian menghitung value tiap objek yang ada di tiap area dan dijadikan weight untuk tiap area. Setelah itu akan dicari area dengan weight terbesar. Kemudian akan dikembalikan dari nomor area tersebut

2. searchHeadInArea

Fungsi ini akan mencari heading terbaik pada suatu area dengan cara membuat list food dan superfood yang terdapat pada area tersebut kemudian mengurutkannya berdasarkan distance lalu memilih yang terdekat

Bab 4

Implementasi dan Pengujian

A.Pseudocode

```
package
Service
s;

import Enums.*;
import Models.*;
import com.fasterxml.jackson.dataformat.xml.XmlAnnotationIntrospector;
import org.ietf.jgss.GSSManager;

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

public class BotService {
    private GameObject bot;
    private PlayerAction playerAction;
    private GameState gameState;
    private GameObject firedTarget;
    private GameObject prevTarget;
    private int prevSize;
    private boolean first;
```

```
private boolean justFired;  
private boolean normalShot;  
  
  
public BotService() {  
    this.playerAction = new PlayerAction();  
    this.gameState = new GameState();  
}  
  
  
  
  
public GameObject getBot() {  
    return this.bot;  
}  
  
  
  
public void setBot(GameObject bot) {  
    this.bot = bot;  
    this.first = true;  
    this.justFired = false;  
    this.prevSize = bot.getSize();  
    this.normalShot = true;  
}  
  
  
  
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);
    //
    //      this.playerAction = playerAction;

    if (!gameState.getGameObjects().isEmpty()) {
        System.out.println("calculating...");

        if(justFired) {
            playerAction.action = PlayerActions.FORWARD;
            playerAction.heading = getHeadingBetween(prevTarget);

            //
            playerAction = didntSeeThatComing(playerAction);

            justFired = false;
        } else {
```

```
        playerAction = farmingMethod(playerAction);

        playerAction = greedyArentYou(playerAction);

        playerAction = deathUponYou(playerAction);

        //          playerAction = ambulanceNotForMe(playerAction);

    }

System.out.println(gameState.getWorld().getCurrentTick());
System.out.println("Done!");

System.out.println(this.bot.getSize());

first = false;

}

this.playerAction = playerAction;

}

//*****



public PlayerAction farmingMethod(PlayerAction playerAction) {

    if(first || prevSize != bot.getSize()) {

        //          playerAction = gotoBestFoodHorde(playerAction);

        prevTarget = getBestFoodHorde();

        playerAction.action = PlayerActions.FORWARD;

        playerAction.heading = getHeadingBetween(prevTarget);
```

```
        System.out.println(getDistanceBetween(bot, prevTarget));
        System.out.println("heading to food.");
        System.out.println(prevSize);

        prevSize = bot.getSize();
    }

    System.out.println("farming method called.");
    return playerAction;
}

public PlayerAction deathUponYou(PlayerAction playerAction) {

    GameObject bestPrey = getUrgentPrey();
    double distance = getDistanceBetween(bestPrey, this.bot);

    if(bestPrey != this.bot && (this.bot.getSize() -
bestPrey.getSize() > 5 ||
        this.bot.getSize() - bestPrey.getSize() > -20 &&
distance < 70)) {

        if(normalShot) {
            playerAction.heading = getHeadingBetween(bestPrey);
            normalShot = false;
        } else {
            playerAction.heading = getFutureHeading(bestPrey);
        }
    }
}
```

```
        normalShot = true;

    }

    playerAction.action = PlayerActions.FIRETORPEDOES;
    System.out.println("\\"death upon you\\" called.");
}

justFired = true;
firedTarget = bestPrey;

return playerAction;
}

public PlayerAction greedyArentYou(PlayerAction playerAction) {

    var shipList = gameState.getPlayerGameObjects();
    double avgSpd = 0;
    for(int i = 0; i < shipList.size(); i++) {
        avgSpd += shipList.get(i).getSpeed();
    }

    avgSpd *= 1/ shipList.size();

    GameObject bestPrey = getSpamablePrey();

    if(bestPrey.getSpeed() < avgSpd * 3 / 2 && this.bot.getSize() > 30) {
        if(normalShot) {

            playerAction.heading = getHeadingBetween(bestPrey);
            normalShot = false;
        }
    }
}
```

```
        } else {
            playerAction.heading = getFutureHeading(bestPrey);
            normalShot = true;
        }
        playerAction.action = PlayerActions.FIRETORPEDOES;

        justFired = true;
        firedTarget = bestPrey;
        System.out.println("\"greedy arent you\" called.");
    }

    return playerAction;
}

private PlayerAction didntSeeThatComing(PlayerAction playerAction) {

    if(firedTarget.effects.value >= Effects.SHIELD.value &&
    getDistanceBetween(firedTarget, this.bot) < this.bot.getSize() * 2
    && this.bot.effects.value < Effects.SHIELD.value){
        playerAction.action = PlayerActions.ACTIVATESHIELD;
    }

    return playerAction;
}

private PlayerAction ambulanceNotForMe(PlayerAction playerAction) {
    GameObject urgentPrey = getNearestShip();
}
```

```

        var torpedoList = gameState.getGameObjects()

                .stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.TORPEDOSALVO
                        && item.currentHeading != (getHeadingBetween(item)
+ 180) % 360)
                .sorted(Comparator

                        .comparing(item -> getDistanceBetween(bot, item)))
                .collect(Collectors.toList());
}

double distance = getDistanceBetween(urgentPrey, this.bot);

if(distance < Math.pow(distance,0.5) * 60 / this.bot.getSpeed() &&
!torpedoList.isEmpty()) {
    playerAction.action = PlayerActions.ACTIVATESHIELD;
}
}

return playerAction;
}

private int getFutureHeading(GameObject otherObject) {

    double futureConstant = otherObject.getSpeed() *
getDistanceBetween(this.bot, otherObject) / 60;

    double futureX = Math.cos(otherObject.currentHeading) *
futureConstant;
    double futureY = Math.sin(otherObject.currentHeading) *
futureConstant;

    var direction = toDegrees(Math.atan2(otherObject.getPosition().y +
futureY - bot.getPosition().y,
otherObject.getPosition().x + futureX -
bot.getPosition().x));
}

```

```

        return (direction + 360) % 360;
    }

    public GameObject getNearestFood() {
        var foodList = gameState.getGameObjects()
            .stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.FOOD)
            .sorted(Comparator
                .comparing(item -> getDistanceBetween(bot, item)))
            .collect(Collectors.toList());
    }

    return foodList.get(0);
}

public GameObject getBestFoodHorde() {
    var foodList = gameState.getGameObjects()
        .stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.FOOD
        || item.getGameObjectType() ==
ObjectTypes.SUPERFOOD)
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList());
}

List<Map.Entry<GameObject, Double>> bestFoodList = new
ArrayList<>();

for (int i = 0; i < 30; i++) {
    GameObject currentFood = foodList.get(i);

    var temp = gameState.getGameObjects()

```

```

        .stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.FOOD)
            .sorted(Comparator
                .comparing(item ->
getDistanceBetween(currentFood, item)))
            .collect(Collectors.toList());
    }

    double oneOverDistances = 0;

    for (int j = 0; j < 20; j++) {
        double distances = getDistanceBetween(currentFood,
temp.get(j));
        oneOverDistances += Math.pow(distances,-2) * 1000000;
    }

    if(currentFood.getGameObjectType() == ObjectTypes.SUPERFOOD) {
        bestFoodList.add(new
AbstractMap.SimpleEntry<>(currentFood,
oneOverDistances *
Math.pow(getDistanceBetween(currentFood, bot), -1.05)
* 36));
    } else {
        bestFoodList.add(new
AbstractMap.SimpleEntry<>(currentFood,
oneOverDistances *
Math.pow(getDistanceBetween(currentFood, bot), -1.05)));
    }
}

var bestFood = bestFoodList.get(0);

for (int i = 0; i < bestFoodList.size(); i++){
    if(bestFoodList.get(i).getValue() > bestFood.getValue()){
}
}

```

```
        bestFood = bestFoodList.get(i);

    }

}

return bestFood.getKey();
}

public GameObject getSpamablePrey(){

    var preyList = gameState.getPlayerGameObjects()

        .stream().filter(item -> item !=

this.bot).sorted(Comparator

        .comparing(item -> Math.pow(item.getSpeed(), 1.2)

            * Math.pow(getDistanceBetween(bot, item), 0.5)))

        .collect(Collectors.toList());



    return preyList.get(0);
}

public GameObject getUrgentPrey(){

    GameObject bestPrey = getNearestShip();

    double bestPreyDistance = getHeadingBetween(bestPrey);

    if(bestPreyDistance - this.bot.getSize() <

Math.pow(bestPreyDistance,0.5) * 60 / bestPrey.getSpeed()){

        return bestPrey;

    } else {

        return this.bot;
    }
}

public GameObject getNearestShip(){
```

```

        var shipList = gameState.getPlayerGameObjects()

                .stream().sorted(Comparator
                        .comparing(item -> getDistanceBetween(bot, item)))
                .collect(Collectors.toList());
    }

    if(gameState.getPlayerGameObjects().size() > 1) { return
shipList.get(1); } else { return shipList.get(0); }

}

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));  
}  
}
```

B.Struktur data

1. GameObject

Struktur data GameObject berisi atribut yang dimiliki oleh object-object pada permainan ini. Misalnya seperti ID, Size, Speed, Heading, Position. GameObject pada player dan object lainnya sedikit berbeda dimana pada player GameObject memiliki 11 atribut sedangkan object lainnya memiliki 7 atribut.

2. PlayerAction

Struktur data PlayerAction berisi ID PLAYER, Action dan heading. Dimana struktur data ini digunakan untuk mengirimkan command untuk menggerakkan objek yang dimainkan.

Playeraction tersebut yaitu,

1. Forward
2. Stop
3. StartAfterburner
4. StopAfterburner
5. FireTorpedos
6. FireSupernova
7. DetonateSupernova
8. FireTeleport
9. ActivateShield

3. GameState

Struktur data GameState berisi kondisi saat ini dari game seperti kondisi world game, list object pada game serta list player pada game.

4. Position

Struktur data Position berisi posisi sebuah objek dalam bidang kartesian

5. World

Struktur data world berisi center point, radius permainan dan current tick

6.ObjectTypes

ObjectTypes merupakan kumpulan tipe tipe objek yang terdapat pada permainan ini. Tipe tipe objek tersebut yaitu,

1. Player
2. Food
3. Wormhole
4. GasCloud
5. AsteroidField
6. TorpedoSalvo
7. Superfood
8. SupernovaPickup
9. SupernovaBomb
10. Teleporter

11. Shield

C.Pengujian

Untuk menguji bot yang sudah kami buat, kami mencoba bertanding dengan reference bot yang telah disediakan oleh Entellect. Pertandingan pertama kami mencoba 1 vs 1 dengan reference bot, pertandingan kedua kami mencoba 3 vs 1 melawan reference bot, pertandingan ketiga kami mencoba 2 vs 2 melawan reference bot, pertandingan ketiga kami mencoba 4 vs 4 melawan reference bot. Pada pertandingan 1 vs 1 dengan reference bot bot kami memperoleh persentase kemenangan 100%. pada pertandingan 3 vs 1 dengan reference bot bot kami memperoleh persentase kemenangan 50%. Pada pertandingan 2 vs 2 dengan reference bot bot kami memperoleh persentase kemenangan 100%. Pada pertandingan 4 vs 4 dengan reference bot bot kami memperoleh persentase kemenangan 100%. Hasil terbaik yaitu bot menang ketika dalam pertandingan jumlah bot dan jumlah bot musuh seimbang seperti pada pengujian diatas ketika 2 vs 2, 1 vs 1, dan 4 vs 4. Hasil terburuk yaitu bot kalah ketika dalam pertandingan jumlah bot musuh lebih banyak dari bot pemain seperti pada pengujian diatas ketika bot pemain dikeroyok 3 vs 1.

Bab 5

Kesimpulan dan Saran

Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma greedy untuk membuat bot permainan Galaxio yang bisa mencapai tujuan objektif yaitu memenangkan permainan dengan menjadi kapal yang bertahan hidup sampai akhir. Kelompok kami membuat strategi greedy berdasarkan profit sehingga bot kami akan bergerak kearah area yang memiliki weight tertinggi. Bot yang kami buat dapat menyelesaikan permainan galaxio dengan cukup optimal.

Saran

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

- a. Akan lebih baik jika dilakukan pembagian tugas terlebih dahulu agar workload masing - masing anggota jelas dan pengerjaan lebih terstruktur
- b. Kami menyarankan untuk berikutnya, pembuatan laporan dapat dilakukan lebih cepat lagi dan tidak terlalu mepet dengan tanggal pengumpulan.

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)-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/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf)

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

Link Github : https://github.com/ChrisAlberth/Tubes1_JannaWessels.io