

**Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II Tahun 2022/2023**

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”



Disusun oleh:

Azmi Hasna Zahrani	13521006
Matthew Mahendra	13521007
Syarifa Dwi Purnamasari	13521018

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023**

DAFTAR ISI

DAFTAR ISI	1
BAB I	3
DESKRIPSI TUGAS	3
BAB II	4
LANDASAN TEORI	4
2.1 Algoritma Greedy	4
2.2. Galaxio Challenge	4
2.3. Cara Menjalankan Permainan	5
BAB III	7
APLIKASI STRATEGI GREEDY	7
3.1 Proses Mapping Persoalan Galaxio Menjadi Elemen-Element Algoritma Greedy	7
3.2 Eksplorasi Alternatif Solusi Greedy yang Mungkin Dipilih dalam Persoalan Galaxio	8
3.2.1 Greedy by Size	8
3.2.2 Greedy by Obstacle	8
3.2.3 Greedy by Food	8
3.2.4 Greedy by Attack and Defense	8
3.3 Analisis Efisiensi dan Efektivitas dari Kumpulan Alternatif Solusi Greedy	9
3.4 Strategi Greedy yang Dipilih	10
BAB IV	11
IMPLEMENTASI DAN PENGUJIAN	11
4.1 Implementasi Algoritma Greedy yang Digunakan	11
4.2 Struktur Data yang Digunakan	17
4.3 Pengujian dan Analisis Solusi Algoritma Greedy	18
4.3.1 Kondisi di Dekat Food	18
4.3.2 Kondisi di Dekat Lawan Aksi Menghindar	18
4.3.3 Kondisi di Dekat Lawan Aksi Menyerang	18
4.3.4 Kondisi di Dekat Lawan Aksi Menembak	19
4.3.5 Kondisi Menghindari Serangan Lawan	19
4.3.6 Kondisi Tersisa Dua Bot	20
4.3.7 Kondisi Menemukan Superfood	20
4.3.8 Catatan Kasus Khusus	21
BAB V	22
KESIMPULAN DAN SARAN	22
5.1 Kesimpulan	22

5.2 Saran	22
5.3 Refleksi	22
DAFTAR PUSTAKA	23
LAMPIRAN	24

BAB I

DESKRIPSI TUGAS

Galaxio adalah sebuah game battle royale yang mempertandingkan bot kapal anda dengan beberapa bot kapal yang lain. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah mempertahankan bot kapal masing-masing agar tetap hidup hingga akhir permainan. Agar dapat memenangkan permainan, setiap bot harus mengimplementasikan strategi tertentu. Pada tugas besar 1 ini, dispesifikkan harus menggunakan algoritma greedy untuk pengimplementasian tersebut.

Game Galaxio memiliki beberapa komponen permainan yang ada di dalamnya. Beberapa komponen tersebut antara lain, peta permainan berbentuk kartesius yang memuat objek-objek permainan berupa Players, Food, Wormholes, Gas Clouds, dan Asteroid Fields. Dalam permainan ini juga memiliki beberapa objek yang berlaku sebagai senjata, yaitu Torpedo Salvo dan Supernova. Selain itu, Players dalam game ini juga dapat melakukan berbagai hal seperti Teleport dan menyerang Players lain dengan berbagai cara yang berlaku. Secara khusus dijelaskan pada bagian *command* yang dapat dilakukan oleh players, *command-command* tersebut antara lain FORWARD, STOP, START_AFTERBURNER, STOP_AFTERBURNER, FIRE_TORPEDOES, FIRE_SUPERNOVA, DETONATE_SUPERNOVA, FIRE_TELEPORTER, dan TELEPORTUSE_SHIELD.

Tugas besar 1 IF2211 Strategi Algoritma dikerjakan secara berkelompok dengan menggunakan bahasa pemrograman Java serta difasilitasi oleh *game engine* yang diberikan bersama spesifikasi.

BAB II

LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* merupakan salah satu metode pemecahan masalah optimasi (*optimization problem*) untuk memaksimalkan dan meminimumkan suatu parameter yang populer dan sederhana. Alur utama dalam pemecahan masalah dengan menggunakan algoritma ini adalah dengan menyelesaikan persoalan secara *step by step* dan pada setiap langkah, pilihan yang terbaik diambil tanpa memperhatikan konsekuensi ke depan serta berharap bahwa dengan memilih optimum lokal, optimum global dapat tercapai di akhir. Suatu permasalahan dapat diselesaikan dengan algoritma *greedy* apabila solusi optimal dari persoalan tersebut dapat ditentukan dari solusi dari optimal subpersoalan tersebut dan pada setiap permasalahan, terdapat suatu langkah yang dapat dilakukan dimana langkah tersebut menghasilkan solusi optimal pada subpersoalan tersebut (*greedy choice*).

Pada algoritma *greedy*, beberapa elemen-elemen yang digunakan antara lain:

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 (*solution function*) : 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*) : memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif (*objective function*) : memaksimalkan atau meminimumkan

Akan tetapi, optimum global belum tentu merupakan solusi optimum terbaik dan bisa jadi merupakan solusi *sub-optimum* atau *pseudo-optimum*. Hal ini dikarenakan algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada (sebagaimana pada metode *exhaustive search*) dan terdapat beberapa fungsi seleksi yang berbeda, sehingga untuk mendapatkan solusi optimal diperlukan fungsi yang tepat. Maka dari itu, pada sebagai persoalan, algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal, tetapi solusi sub-optimal.

2.2. Galaxio Challenge

Galaxio Challenge merupakan sebuah pertandingan bot pada tahun 2021 yang dibuat oleh Entelect Challenge. Ide dasar permainan ini adalah sebuah peta yang diisi dengan berbagai objek dan juga pemain-pemain lainnya. Pemain akan membuat sebuah bot yang akan digunakan untuk mengeliminasi pemain lainnya dalam permainan ini hingga tersisa satu pemain yang akan dinyatakan sebagai juara.

Permainan ini didukung oleh beberapa elemen yang dapat digunakan oleh pemain, seperti penembakan torpedo salvo untuk mengurangi ukuran musuh dan menambah ukuran pemain untuk setiap penembakan yang berhasil, aktivasi shield dari torpedo salvo, teleportation torpedo untuk berpindah tempat, afterburner untuk menambah kecepatan, dan food untuk bertambah besar. Untuk penggunaan senjata, maka akan ada konsekuensi berupa pengurangan ukuran.

Penyerangan dapat dilakukan antar pemain dengan cara saling memakan pemain. Jika ada pemain yang ukurannya lebih kecil dan dimakan oleh pemain yang lebih besar, maka pemain tersebut akan tereliminasi. Pengurangan ukuran hingga di bawah 5 juga dapat membuat seorang pemain tereliminasi. Pengurangan yang dilakukan selain oleh pemain adalah melalui elemen permainan seperti asteroid, gas cloud, dan jika bot berada di luar area pertandingan.

2.3. Cara Menjalankan Permainan

Untuk menjalankan permainan ini, dapat mengikuti langkah sebagai berikut,

1. Unduh seluruh repository ini sebuah folder dan letakkan dalam engine yang telah diunduh pada folder starter-bots
2. Akan terbentuk sebuah file dengan ekstensi .jar dengan nama KMMBot.jar pada folder target. File ini merupakan bot yang akan digunakan
3. Jika belum melakukan pengaturan jumlah bot pada game engine, lakukan pengaturan pada file appsettings.json pada folder runner-publish dan logger-publish
4. Pada runner game engine, masukkan command java -jar path dengan path yang mengarah pada ./target/KMMBot.jar lalu jalankan runner. Contoh runner.bat yang dapat digunakan adalah

```
@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 ../starter-bots/JavaBot/target/JavaBot.jar
timeout /t 3
start "" java -jar ../starter-bots/Tubes1_KMM/target/KMMBot.jar
timeout /t 3
```

```
start "" java -jar ../starter-bots/Tubes1_KMM/target/KMMBot.jar  
timeout /t 3  
start "" java -jar ../starter-bots/Tubes1_KMM/target/KMMBot.jar  
cd ../  
  
pause
```

Runner ini menggunakan 4 bot dengan target adalah KMMBot sebanyak 3 bot dan 1 JavaBot yang merupakan reference bot

5. Setelah menjalankan game, visualizer dapat dibuka untuk melihat bot yang telah digunakan

Selain itu, diperlukan pula *prerequisites* sebagai berikut,

1. Java SDK (Bot dikembangkan pada versi 19.0.1)
2. .NET 3.1.0 dan .NET Runtime 5.0 untuk menjalankan game engine
3. Maven untuk membuat file .JAR
4. Game Engine yang dapat diunduh pada tautan:
<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

BAB III

APLIKASI STRATEGI GREEDY

3.1 Proses Mapping Persoalan Galaxio Menjadi Elemen-Elemen Algoritma Greedy

Permainan Galaxio merupakan permainan yang mengharuskan Players untuk mempertahankan diri sendiri sampai akhir. Beberapa strategi bisa digunakan untuk dapat mencapai kemenangan, seperti meninjau dari posisi, ukuran, obstacle, dan lain-lain.

Nama Elemen	Deskripsi
Himpunan Kandidat	Himpunan kandidat yang digunakan untuk strategi memenangkan permainan Galaxio adalah {posisi, ukuran, obstacle}.
Himpunan Solusi	Himpunan solusi yang diambil dari himpunan kandidat adalah {posisi, ukuran}
Fungsi Solusi	Fungsi solusi digunakan untuk menyeleksi himpunan kandidat untuk membentuk himpunan solusi. Fungsi solusi ini merupakan pertimbangan dan perhitungan untuk menentukan strategi yang digunakan.
Fungsi Seleksi	Fungsi seleksi digunakan untuk memilih kandidat berdasarkan strategi greedy yang dipilih. Dalam hal ini, strategi yang dipilih adalah posisi dan ukuran. Fungsi seleksi diimplementasikan dalam permainan untuk memilih jalan players agar bisa survive sampai akhir permainan.
Fungsi Kelayakan	Fungsi kelayakan digunakan untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi, dalam kata lain layak atau tidak untuk digunakan. Fungsi kelayakan langsung diimplementasikan pada program sambil waktu berjalan. Apabila tidak layak ataupun tidak memberikan hasil yang diinginkan, maka kandidat langsung dieliminasi.
Fungsi Objektif	Fungsi objektif digunakan untuk mencari permutasi dari solusi yang membuat player dapat memenangkan permainan.

3.2 Eksplorasi Alternatif Solusi Greedy yang Mungkin Dipilih dalam Persoalan Galaxio

Dalam mengerjakan Tugas Besar 1 IF2211 Strategi Algoritma, kelompok mengeksplorasi beberapa alternatif solusi greedy yang dapat diaplikasikan dalam memenangkan permainan Galaxio.

3.2.1 Greedy by Size

Greedy by Size merupakan salah satu strategi yang dapat digunakan untuk menyelesaikan permainan Galaxio. Strategi ini bekerja dengan memerhatikan ukuran pesawat. Apabila pesawat masih berukuran kecil, pesawat akan memilih untuk memakan Food di sekitarnya untuk menambah ukuran. Ketika ukurannya besar, pesawat baru akan menyerang lawan menggunakan senjata yang dapat digunakannya. Strategi ini tidak dipilih untuk menjadi solusi karena memiliki kelemahan yaitu mengabaikan objek-objek lain selain makanan dan pesawat musuh.

3.2.2 Greedy by Obstacle

Greedy by Obstacle bekerja dengan memerhatikan objek-objek di sekitar pesawat yang dapat mengurangi ukuran pesawat seperti Asteroid, Gas, dan Edge. Pesawat otomatis akan menghindari obstacle-obstacle tersebut agar ukurannya tidak berkurang. Namun, strategi ini memiliki kelemahan yaitu pesawat hanya akan memerhatikan obstacle tanpa peduli dengan Food dan Players lain.

3.2.3 Greedy by Food

Strategi ini berfokus hanya pada proses bertambah besar dan tidak memikirkan algoritma lainnya. Pada proses memakan ini, bot hanya mengandalkan pada keuntungan bahwa akan ada suatu momen ketika ada bot yang ukurannya lebih kecil berada di dekatnya sehingga termakan ketika bot juga sedang mencari makan. Kelemahan dari strategi ini sangat jelas karena fungsi objektif global yang ingin dicapai, yaitu untuk memenangkan permainan menjadi sangat kecil kemungkinannya. Kecuali jika semua bot pada suatu pertandingan menggunakan algoritma ini, maka hanya menunggu waktu saja agar terjadi proses eliminasi hingga tersisa satu anggota.

3.2.4 Greedy by Attack and Defense

Strategi ini merupakan modifikasi dari 3.2.3. Pada kondisi normal, menyerang bisa saja menjadi langkah yang terbaik agar mengurangi jumlah lawan dan perlindungan juga langkah yang terbaik agar dapat melindungi dari tembakan torpedo lawan. Akan tetapi, karena sifat dari permainan ini, setiap melakukan penyerangan berupa penembakan torpedo, baik teleport maupun salvo, serta aktivasi shield, akan mengurangi ukuran dari bot.

3.3 Analisis Efisiensi dan Efektivitas dari Kumpulan Alternatif Solusi Greedy

Solusi Greedy	Efisiensi	Efektivitas	Deskripsi
Greedy by Size	Langkah umum hanya dua yaitu memakan atau menyerang sehingga membuat algoritma ini efisien	Kurang efektif	Tidak memerhatikan objek-objek lain yang dapat membuat fungsi objektif global tidak tercapai, i.e. mengecil karena terkena torpedo/gas cloud dan asteroid. Bot juga tidak menggunakan perangkat-perangkat lain seperti teleport, torpedo, dan shield karena hanya mengandalkan ukuran.
Greedy by Obstacle	Harus memerhatikan radius (edge), asteroid dan atau gas cloud. Mencoba menghindari lokasi yang terdapat objek-objek tersebut. Kurang efisien.	Tidak efektif	Pada dasarnya hanya bergerak mencoba menghindari obstacle saja tanpa ada usaha untuk menyerang musuh. Fungsi objektif global sulit untuk dicapai.
Greedy by Position	Membuat deretan objek yang paling dekat. Lalu mengambil langkah untuk setiap objek yang terdekat. Juga membuat prioritas. Algoritma cukup efisien	Efektif	Algoritma efektif dalam menentukan langkah-langkah yang harus diambil berikutnya karena berdasarkan posisi bot terdekat. Juga mempertimbangkan seluruh aspek objek
Greedy by Food	Tidak efisien dalam meraih fungsi objektif global karena fokus hanya menambah ukuran pada bot	Tidak efektif	Bot hanya akan menambah makanan. Proses menyerang bergantung pada keberuntungan akan bot lain yang

			bertabrakan dengan bot pemain.
Greedy by Attack and Defence	Tidak efisien karena setiap penyerangan dan perlindungan yang terlalu agresif akan berdampak negatif pada pemain	Tidak efektif	Bot akan terus menyerang dan berindung dengan shield. Jika tidak disertai dengan proses penambahan besar, maka bot dapat tereliminasi jika bot menjadi agresif.

3.4 Strategi Greedy yang Dipilih

Strategi greedy yang dipilih oleh kelompok merupakan campuran dari strategi greedy by position dan greedy by size. Strategi ini mengambil setiap langkah pada permainan dengan memerhatikan posisi bot, yaitu benda yang ada di dekatnya. Apabila terdapat obstacle di sekitar bot, maka bot akan melakukan beberapa aksi yang dikelompokkan sesuai dengan obstacle yang dihadapi. Apabila obstacle yang ada di sekitar bot berupa Asteroid, Gas, dan Edge, maka bot akan menghindari obstacle tersebut. Apabila obstacle yang ada di sekitar bot berupa Food maka bot akan melakukan aksi memakan obstacle tersebut. Apabila obstacle yang ada di sekitar bot berupa bot lawan, maka bot akan melakukan aksi sesuai kondisi bot lawan. Apabila bot lawan yang di dekat bot berukuran lebih besar, maka bot akan menghindar atau menembaknya. Tindakan penembakan torpedo salvo dilakukan apabila ukuran dari bot mencukupi sehingga tidak akan mati karena pengurangan ukuran (konsekuensi dari penembakan torpedo). Apabila bot lawan yang di dekat bot berukuran lebih kecil, maka bot akan memakannya. Apabila terdapat tembakan Torpedo Salvo dari lawan, dan tembakan tersebut berada di dekat bot maka bot akan mengaktifkan shield, tetapi dengan melihat terlebih dahulu apakah torpedo salvo terdapat pada permainan atau tidak (ada yang ditembakkan atau tidak). Aktivasi dari shield juga memerhatikan ukuran dari bot karena aktivasi dari shield akan mengurangi ukuran dari bot.

Strategi ini dipilih karena dinilai paling efektif dari strategi yang ada pada himpunan kandidat yang ada. Selain itu, strategi ini juga memiliki hasil yang paling optimal untuk bertahan hingga akhir permainan karena tidak hanya berfokus pada satu objek saja, tetapi juga kepada beberapa objek sekaligus selama objek tersebut ada di dekat bot, sekaligus melakukan *handling* terhadap . Oleh karena itulah, strategi Greedy by Position ini dipilih.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy yang Digunakan

Algoritma Greedy
untuk Galaxio Bot
Kelompok KMM

Greedy yang digunakan: Greedy by Position blend dengan Greedy by Size

Kamus Lokal
bot : PlayerObject
playerAction : PlayerAction
gameState : GameState
worldCenter : worldCenter
centerPoint : Position

```
Algoritma
centerPoint <- makeNewPosition(0,0)
worldCenter <- makeGameObject(null, null, null, null, centerPoint, null, null,
null)

/* Mencari Posisi terdekat dari setiap tipe objek */
for each gameState.gameObject
    if gameState.gameObject.type = FOOD then
        foodlist <- gameState.gameObject

sort foodlist by distanceBetween(bot, foodlist)

for each gameState.gameObject
    if gameState.gameObject.type = ASTEROID or gameState.gameObject.type =
GAS_CLOUD then
        obstacleList <- gameState.gameObject

sort obstacleList by distanceBetween(bot, obstacleList)

for each gameState.gameObject
    if gameState.gameObject.type = SUPERFOOD then
        superFoodList <- gameState.gameObject

sort superFoodList by distanceBetween(bot, superFoodList)

for each gameState.gameObject
    if gameState.gameObject.type = PLAYER then
        playerList <- gameState.gameObject

sort playerList by distanceBetween(bot, playerList)

for each gameState.gameObject
    if gameState.gameObject.type = TORPEDO then
        torpedoList <- gameState.gameObject
```

```

sort torpedoList by distanceBetween(bot, torpedoList)

/* Buatlah aksi default dan juga aksi-aksi lainnya berdasarkan posisi objek
terdekat */
/* .heading artinya arah dari bot akan menuju objek dalam fungsi
getHeadingBetween */
/* .action artinya aksi yang dilakukan bot seperti menembak, aktivasi shield,
dll. */

/* Default: Makan */
playerAction.heading <- getHeadingBetween(foodlist[0])

/* Jika ada superfood di dekat bot */
if(distanceBetween(bot,superFoodList(0)) < distanceBetween(bot,foodList(0)))
then
    playerAction.heading <- getHeadingBetween(superFoodList[0])

/* Jika ada obstacle di dekat bot */
if(getDistanceBetween(bot, gasList[0]) < 100 ) then
    playerAction.heading <- (playerAction.currentHeading + 90) % 360

if(getDistanceBetween(bot, astList[0]) < 100 or getDistanceBetween(bot,
astList[0]) < getDistanceBetween(bot, gasList[0])) then
    playerAction.heading <- (playerAction.currentHeading + 90) % 360

/* Jika ada player */
/* Kondisi Player lebih dari satu */
if(getDistanceBetween(this.bot, enemyList[0]) - bot.size -
enemyList.get[0].size <= 200) then
    if(burner) then
        burner <- false
        playerAction.action <- STOPAFTERBURNER

    /* Periksa ukuran bot dengan musuh */
    if(enemyList[0].size < bot.size) then
        playerAction.heading <- getHeadingBetween(enemyList[0])

        /* Periksa apakah memungkinkan untuk menembak atau tidak */
        if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size < 100 and bot.size > 50) then
            playerAction.heading <- getHeadingBetween(enemyList[0])
            playerAction.action <- FIRETORPEDOES;

        /* Periksa apakah memungkinkan untuk teleport menuju musuh atau tidak
*/
        if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size > 50 and bot.size > 100) then
            depend on teleportTick:
                teleportTick = 0:
                    playerAction.heading <- getHeadingBetween(enemyList[0])
                    playerAction.action <- FIRETELEPORT
                    teleportTick <- 1;
                teleportTick >= 1 and teleportTick < 10:
                    teleportTick <- teleportTick + 1
                teleportTick = 10:
                    teleportTick <- 0

```

```

        playerAction.action <- TELEPORT

    else if(enemyList[0].size >= bot.size) then
        playerAction.heading <- (getHeadingBetween(enemyList[0]) + 90) % 360;
        if( bot.size > 85 and getDistanceBetween(enemyList[0], bot) -
bot.size - enemyList[0].size <= 150) then
            if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size <= 100) then
                playerAction.heading <- enemyList[0].currentHeading

                /* Jika jarak terlalu dekat, aktifkan afterburner */
                playerAction.action <- STARTAFTERBURNER;
                burner <- true;

                /* Jika memungkinkan tembak enemy */
                if(this.bot.getSize() > 150 and enemyList[0].size - bot.size < 30
and enemyList[0].size - bot.size > 30 ) then
                    playerAction.heading <- getHeadingBetween(enemyList[0])
                    playerAction.action <- FIRETORPEDOES

/* Kondisi Player hanya dua (bot dan enemy) */
if(playerList = 1)then
    if(burner) then
        burner <- false
        playerAction.action <- STOPAFTERBURNER

/* Sama seperti jika masih banyak, memeriksa ukuran */
if(bot.size < playerList[0].size) then
    playerHeading.heading <- getHeadingBetween(playerList[0])
    if(bot.size > 85) then
        playerAction.heading <- getHeadingBetween(enemyList[0])
        playerAction.action <- FIRETORPEDOES
    if(bot.size >100 and getDistanceBetween(bot, playerList[0]) > 85) then
        if(teleportTick = 0) then
            playerAction.heading <- getHeadingBetween(enemyList[0])
            playerAction.action <- FIRETELEPORT;
            teleportTick <- 1
        }else if(teleportTick >= 1 and teleportTick < 10){
            teleportTick <- teleportTick + 1
        }else if(teleportTick = 10){
            teleportTick <- 0
            playerAction.action <- TELEPORT
        }
    else
        playerAction.heading <- (getHeadingBetween(enemyList[0]) + 90) % 360;
        if((getHeadingBetween(superFoodList[0]) + 180) %360 !=
enemyList[0].currentHeading){
            playerAction.heading <- getHeadingBetween(superFoodList[0]);
        }
        if(getDistanceBetween(enemyList[0], bot) < 50){
            playerAction.heading <- enemyList.get(0).currentHeading;
            playerAction.action <- STARTAFTERBURNER;
            burner <- true;
        }
        if(this.bot.getSize() > 85) {
            playerAction.heading <- getHeadingBetween(enemyList[0]);

```

```

        playerAction.action <- FIRETORPEDOES
    }

/* Cek untuk torpedo */
if(torpedoList.size() > 0)then
    if(getDistanceBetween(bot,torpedoList[0]) < 150) then
        torpedoDirection <- (((torpedoList[0].currentHeading + 180) % 360) -
getHeadingBetween(torpedoList[0]));
        /* jika ada torpedo yang searah dg bot, nyalakan shield */
        if(( torpedoDirection != 180 or torpedoDirection != -180 )and bot.size
> 85) then
            playerAction.action <- PlayerActions.ACTIVATESHIELD

/* Cek apakah bot akan atau sudah berada di luar peta */
if(getDistanceBetween(bot, worldCenter) + (1.5 * bot.size) >=
gameState.getWorld().radius) then
    playerAction.heading <- getHeadingBetween(worldCenter)

/* OLD */
Algoritma Greedy
untuk Galaxio Bot
Kelompok KMM

Greedy yang digunakan: Greedy by Position

Kamus Lokal
bot : PlayerObject
playerAction : PlayerAction
gameState : GameState
worldCenter : worldCenter
centerPoint : Position

Algoritma
centerPoint <- makeNewPosition(0,0)
worldCenter <- makeGameObject(null, null, null, null, centerPoint, null, null,
null)

/* Mencari Posisi terdekat dari setiap tipe objek */
for each gameState.gameObject
    if gameState.gameObject.type = FOOD then
        foodlist <- gameState.gameObject

sort foodlist by distanceBetween(bot, foodlist)

for each gameState.gameObject
    if gameState.gameObject.type = ASTEROID or gameState.gameObject.type =
GAS_CLOUD then
        obstacleList <- gameState.gameObject

sort obstacleList by distanceBetween(bot, obstacleList)

for each gameState.gameObject
    if gameState.gameObject.type = SUPERFOOD then
        superFoodList <- gameState.gameObject

```

```

sort superFoodList by distanceBetween(bot, superFoodList)

for each gameState.gameObject
  if gameState.gameObject.type = PLAYER then
    playerList <- gameState.gameObject

sort playerList by distanceBetween(bot, playerList)

for each gameState.gameObject
  if gameState.gameObject.type = TORPEDO then
    torpedoList <- gameState.gameObject

sort torpedoList by distanceBetween(bot, torpedoList)

/* Buatlah aksi default dan juga aksi-aksi lainnya berdasarkan posisi objek
terdekat */
/* .heading artinya arah dari bot akan menuju objek dalam fungsi
getHeadingBetween */
/* .action artinya aksi yang dilakukan bot seperti menembak, aktivasi shield,
dll. */

/* Default: Makan */
playerAction.heading <- getHeadingBetween(foodlist[0])

/* Jika ada superfood di dekat bot */
if(distanceBetween(bot,superFoodList(0)) < distanceBetween(bot,foodList(0)))
then
  playerAction.heading <- getHeadingBetween(superFoodList[0])

/* Jika ada obstacle di dekat bot */
if(getDistanceBetween(bot, gasList[0]) < 100 ) then
  playerAction.heading <- (playerAction.currentHeading + 90) % 360

if(getDistanceBetween(bot, astList[0]) < 100 or getDistanceBetween(bot,
astList[0]) < getDistanceBetween(bot, gasList[0])) then
  playerAction.heading <- (playerAction.currentHeading + 90) % 360

/* Jika ada player */
/* Kondisi Player lebih dari satu */
if(getDistanceBetween(this.bot, enemyList[0]) - bot.size -
enemyList.get[0].size <= 200) then
  if(burner) then
    burner <- false
    playerAction.action <- STOPAFTERBURNER

  /* Periksa ukuran bot dengan musuh */
  if(enemyList[0].size < bot.size) then
    playerAction.heading <- getHeadingBetween(enemyList[0])

    /* Periksa apakah memungkinkan untuk menembak atau tidak */
    if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size < 100 and bot.size > 50) then
      playerAction.action <- FIRETORPEDOES;

  /* Periksa apakah memungkinkan untuk teleport menuju musuh atau tidak

```



```

*/
    if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size > 50 and bot.size > 100) then
        depend on teleportTick:
            teleportTick = 0:
                playerAction.action <- FIRETELEPORT
                teleportTick <- 1;
            teleportTick >= 1 and teleportTick < 10:
                teleportTick <- teleportTick + 1
            teleportTick = 10:
                teleportTick <- 0
                playerAction.action <- TELEPORT

    else if(enemyList[0].size >= bot.size) then
        playerAction.heading <- (getHeadingBetween(enemyList[0]) + 90) % 360;
        if( bot.size > 100 and getDistanceBetween(enemyList[0], bot) -
bot.size - enemyList[0].size <= 150) then
            if(getDistanceBetween(enemyList[0], bot) - bot.size -
enemyList[0].size <= 100) then
                playerAction.heading <- enemyList[0].currentHeading

        /* Jika jarak terlalu dekat, aktifkan afterburner */
        playerAction.action <- STARTAFTERBURNER;
        burner <- true;

        /* Jika memungkinkan tembak enemy */
        if(this.bot.getSize() > 150 and enemyList[0].size - bot.size < 30
and enemyList[0].size - bot.size > 30 ) then
            playerAction.heading <- getHeadingBetween(enemyList[0])
            playerAction.action <- FIRETORPEDOES

/* Kondisi Player hanya dua (bot dan enemy) */
if(playerList = 1) then
    if(burner) then
        burner <- false
        playerAction.action <- STOPAFTERBURNER

    /* Sama seperti jika masih banyak, memeriksa ukuran */
    if(bot.size < playerList[0].size) then
        playerAction.heading <- getHeadingBetween(playerList[0])
        if(bot.size > 100) then
            playerAction.action <- FIRETORPEDOES
        if(bot.size > 100 and getDistanceBetween(bot, playerList[0]) > 50) then
            if(teleportTick = 0) then
                playerAction.action <- FIRETELEPORT
                teleportTick <- 1
            else if(teleportTick >= 1 and teleportTick < 10)
                teleportTick <- teleportTick + 1
            else if(teleportTick = 10)
                teleportTick <- 0
                playerAction.action <- TELEPORT

    else
        playerAction.heading <- (getHeadingBetween(enemyList[0]) + 180) % 360;
        if((getHeadingBetween(superFoodList[0]) + 180) % 360 !=
enemyList[0].currentHeading) {

```

```

        playerAction.heading <- getHeadingBetween(superFoodList[0]);
    }
    if(getDistanceBetween(enemyList[0], bot) < 50){
        playerAction.action <- STARTAFTERBURNER
        burner <- true
    }
    if(this.bot.getSize() > 100) {
        playerAction.heading <- getHeadingBetween(enemyList[0])
        playerAction.action <- FIRETORPEDOES
    }

/* Cek untuk torpedo */
if(torpedoList.size() > 0)then
    if(getDistanceBetween(bot,torpedoList[0]) < 150) then
        torpedoDirection <- (((torpedoList[0].currentHeading + 180) % 360) -
getHeadingBetween(torpedoList[0]))
        /* jika ada torpedo yang searah dg bot, nyalakan shield */
        if(torpedoDirection != 180 or torpedoDirection != -180) then
            playerAction.action <- PlayerActions.ACTIVATESHIELD

/* Cek apakah bot akan atau sudah berada di luar peta */
if(getDistanceBetween(bot, worldCenter) + (1.5 * bot.size) >=
gameState.getWorld().radius) then
    playerAction.heading <- getHeadingBetween(worldCenter)

```

4.2 Struktur Data yang Digunakan

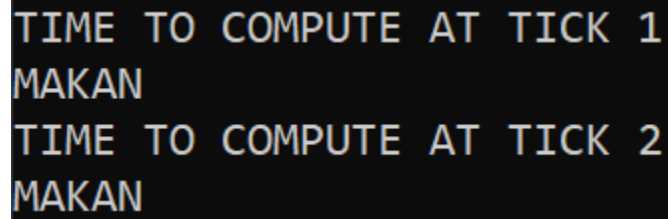
Struktur data yang digunakan pada pembuatan bot ini adalah,

- foodList, superFoodList, gasList, astList, torpedoList: Array of GameObjects
- enemyList : Array of GameObjects where ObjectType = 1
- bot : GameObject
- playerAction : PlayerAction
- worldCenter : GameObject
- burner : boolean
- teleportTick : integer
- centerPoint : Position
- type GameObjects : <UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, Integer torpedoSalvo, Boolean supernovaAvailable>
- type PlayerAction : <UUID playerId, PlayerActions action, int heading>
- type Position : <int x, int y>

4.3 Pengujian dan Analisis Solusi Algoritma Greedy

Pengujian terhadap bot yang digunakan akan menggunakan hasil suatu pertandingan bot. Gambar-gambar yang tertera di bawah ini merupakan salah satu contoh hasil *run* sebuah game.

4.3.1 Kondisi di Dekat Food

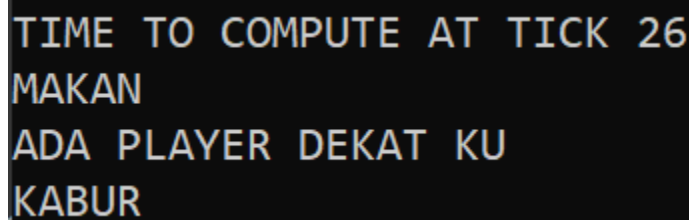


```
TIME TO COMPUTE AT TICK 1  
MAKAN  
TIME TO COMPUTE AT TICK 2  
MAKAN
```

Gambar 4.3.1 Fungsi Makan

Pada kondisi bot berada di dekat makanan, maka bot akan melakukan proses memakan makanan tersebut. Pada kasus ini, bot selalu mengambil makanan.

4.3.2 Kondisi di Dekat Lawan Aksi Menghindar

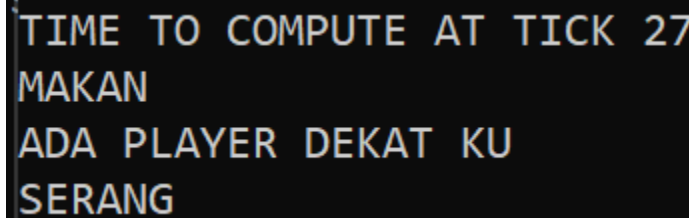


```
TIME TO COMPUTE AT TICK 26  
MAKAN  
ADA PLAYER DEKAT KU  
KABUR
```

Gambar 4.3.2 Fungsi Menghindar dari Lawan

Apabila bot berada pada kondisi di dekat lawan yang memiliki ukuran lebih besar, maka bot akan menghindari lawan. Proses menghindari dilakukan dengan berbelok sebesar 90 derajat yang dapat membuat lawan mengejar bot. Akan tetapi, jika terus dilakukan, dapat menjadi terkunci dalam kejar-mengejar secara berputar.

4.3.3 Kondisi di Dekat Lawan Aksi Menyerang



```
TIME TO COMPUTE AT TICK 27  
MAKAN  
ADA PLAYER DEKAT KU  
SERANG
```

Gambar 4.3.3 Fungsi Menyerang Lawan

Apabila bot berada pada kondisi di dekat lawan dan bot memiliki ukuran yang lebih besar dari lawan, maka bot akan memilih aksi menyerang dengan cara memakan lawan. Penyerangan pada kasus ini adalah penyerangan dengan bergerak ke arah pemain yang paling dekat. Proses ini selalu berhasil mendeteksi dan melakukan perhitungan yang tepat.

4.3.4 Kondisi di Dekat Lawan Aksi Menembak

```
TIME TO COMPUTE AT TICK 58  
MAKAN  
ADA PLAYER DEKAT KU  
FIRE TORPEDOES!  
SERANG
```

Gambar 4.3.4 Fungsi Menembak Lawan

Apabila bot berada pada kondisi di dekat lawan dan bot memiliki ukuran yang besar dengan lawan yang juga besar, maka bot akan menyerang lawan dengan cara menembak Torpedo Salvo. Penembakkan terkadang tidak akurat jika antar pemain sedang bergerak atau jika musuh lain sangat kecil sehingga bergerak dengan cepat. Selain itu, karena adanya *flaw* dalam program yang dibuat, penembakkan terkadang kurang akurat dan menjadi membuat ukuran bot kecil tanpa efek yang berarti (menembak musuh).

4.3.5 Kondisi Menghindari Serangan Lawan

```
TIME TO COMPUTE AT TICK 63  
MAKAN  
ADA PLAYER DEKAT KU  
SERANG  
61  
242  
Activate Shield
```

Gambar 4.3.5 Fungsi Mengaktifkan Shield

Apabila bot mendeteksi ada Torpedo Salvo di sekitarnya, maka bot akan mengaktifkan shield untuk melindunginya. Aktivasi shield ini tidak hanya terjadi ketika bot mendeteksi Torpedo Salvo dari lawan saja, tapi ketika bot menembakkan Torpedo Salvo kepada lawan. Hal ini menjadi kelemahan karena aktivasi shield dapat mengurangi ukuran bot. Namun, hal tersebut sulit dihindari karena tidak ada pembeda di antara Torpedo Salvo yang berasal dari bot maupun lawan.

4.3.6 Kondisi Tersisa Dua Bot

```
TIME TO COMPUTE AT TICK 101  
MAKAN  
ADA PLAYER DEKAT KU  
SERANG  
Tinggal berdua nih!  
SERANG
```

Gambar 4.3.6 Fungsi Pengirim Pesan Sisa Dua Pemain

Ketika tersisa dua pemain di dalam peta, maka akan dicetak keterangan tersisa dua pemain dan bot akan mulai menyerang lawan dengan menembak dan melakukan teleport. Proses yang diambil ketika bot tersisa dua hanya melakukan penyerangan ataupun perlindungan, tergantung kondisi ukuran tubuh dari bot serta jarak dari musuh. Proses penyerangan dapat dilakukan selalu, sedangkan dalam proses menghindari terkadang bot akan mengaktifkan afterburner lalu tereliminasi karena perhitungan dari bot lain yang mempertimbangkan perbedaan kecepatan dari bot dan atau ukuran musuh yang lebih besar.

4.3.7 Kondisi Menemukan Superfood

```
TIME TO COMPUTE AT TICK 157  
MAKAN  
ADA PLAYER DEKAT KU  
KABUR  
Tinggal berdua nih!  
KABUR, CARI MAKAN DULU  
MAKAN SUPER FOOD
```

Gambar 4.3.7 Fungsi Makan Superfood

Ketika menemukan super food, bot akan mengirim pesan dengan mencetak ke layar bahwa bot telah memakan super food. Super food lebih diutamakan daripada food karena dapat menambah ukuran bot lebih cepat. Dalam implementasinya apabila bot menemui food dan super food secara bersamaan, maka bot akan memilih super food terlebih dahulu.

4.3.8 Catatan Kasus Khusus

Pada beberapa pertandingan yang dilakukan saat pengujian oleh kelompok, ada ditemukan hal-hal yang patut menjadi catatan sebagai berikut,

1. Pada saat bot hanya tersisa dua dan atau ketika tick game sudah sangat tinggi, bot terkadang akan mengambil tindakan untuk maju lurus hingga keluar map dan akhirnya tereleminasi. Akan tetapi, pada beberapa komputer yang memiliki spesifikasi yang lebih baik, hal ini lebih jarang ditemukan. Hal ini dapat disebabkan karena kompleksitas program ataupun respons dari hardware komputer
2. Proses penembakan torpedo untuk teleport, untuk jumlah pemain > 2 , terkadang tidak melakukan teleportasi. Hal ini dapat disebabkan karena kondisi yang sudah berbeda sehingga diambil langkah lainnya yang lebih baik. Contohnya ketika melakukan penembakan, ada pemain yang lebih besar sedang mendekat sehingga teleportasi diabaikan dan mengambil langkah untuk menghindar
3. Pada proses mencari makanan, jika terdapat dua makanan yang jaraknya sama, maka bot terkadang menjadi bingung, manakah yang harus diambil. Hal ini dapat teratasi jika ada kondisi lain yang mengharuskan bot untuk berpindah. Kasus ini disebabkan karena logika untuk makan bot yaitu mencari yang terdekat dari posisi bot saat itu
4. Pada saat ada asteroid atau gas cloud yang dekat, pada umumnya bot akan selalu mencoba untuk menghindar. Akan tetapi, pada beberapa kasus, misalnya wormhole yang *spawn* di dalam lingkup asteroid atau gas cloud, maka bot akan kebingungan untuk berpindah sehingga memerlukan beberapa saat agar bot keluar dari area tersebut. Waktu tersebut disebabkan oleh logika bot yaitu berputar sebanyak 90 derajat lalu maju untuk melakukan tindakan selanjutnya

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari tugas ini dapat disimpulkan hal-hal sebagai berikut,

1. algoritma greedy dapat diterapkan dalam persoalan gim *battle royale*
2. dalam penggunaan algoritma greedy, algoritma greedy yang digunakan dapat lebih dari satu algoritma untuk menghasilkan hasil yang lebih maksimal
3. penentuan elemen yang digunakan serta langkah dalam membangun fungsi objektif lokal sangat penting dalam pembangunan algoritma greedy agar dapat menggunakan elemen-elemen tersebut secara baik hingga fungsi objektif lokal terpenuhi
4. dalam menyusun bot permainan, diperlukan pengujian terhadap nilai *threshold* yang digunakan agar dapat menentukan nilai yang paling tepat
5. karena batasan bot adalah pada algoritma greedy, maka beberapa hal seperti keakuratan dapat menjadi kurang karena perubahan kondisi yang terjadi

5.2 Saran

Saran kami untuk penggunaan algoritma greedy ataupun dalam pembuatan bot permainan adalah,

1. dalam pembuatan bot, ada baiknya algoritma greedy digabung dengan algoritma maupun perhitungan matematis lainnya agar bot dapat menjadi lebih akurat dalam melakukan aksi
2. dalam menggunakan algoritma greedy, pertimbangkan dengan lebih dalam *threshold* yang akan digunakan untuk menentukan aksi. Penentuan *threshold* sangat penting dan akan memengaruhi tindakan yang diambil oleh bot

5.3 Refleksi

Melalui tugas ini, kelompok belajar dan semakin memahami mengenai algoritma greedy. Algoritma greedy memang pada dasarnya hanya memikirkan langkah terbaik pada suatu kondisi tertentu. Kelompok belajar untuk bekerja sama dan bekerja keras memikirkan algoritma greedy. Kelompok belajar untuk membagi tugas di tengah kesibukannya masing-masing. Selain itu, melalui batasan yang diberikan pada tugas ini, kelompok belajar untuk mengikuti arahan yang diberikan sesuai spesifikasi

DAFTAR PUSTAKA

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

Algoritma Greedy pada Perkuliahan Strategi Algoritma IF2211 Program Studi Teknik Informatika
Institut Teknologi Bandung Tahun 2023

LAMPIRAN

Tautan Github: https://github.com/MHEN2606/Tubes1_KMM

Tautan Video Pendukung (BONUS 1): <https://youtu.be/TQM4bUWjono>