

**Laporan Tugas Besar 1 IF2211 Strategi algoritma
Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan
“Galaxio”**



**Disusun Oleh:
Kelompok 35 “AI-nus”**

| | |
|----------------------------|----------|
| Muhammad Equilibrie Fajria | 13521047 |
| Fakih Anugerah Pratama | 13521091 |
| Zidane Firzatullah | 13521163 |

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER II TAHUN 2022/2023**

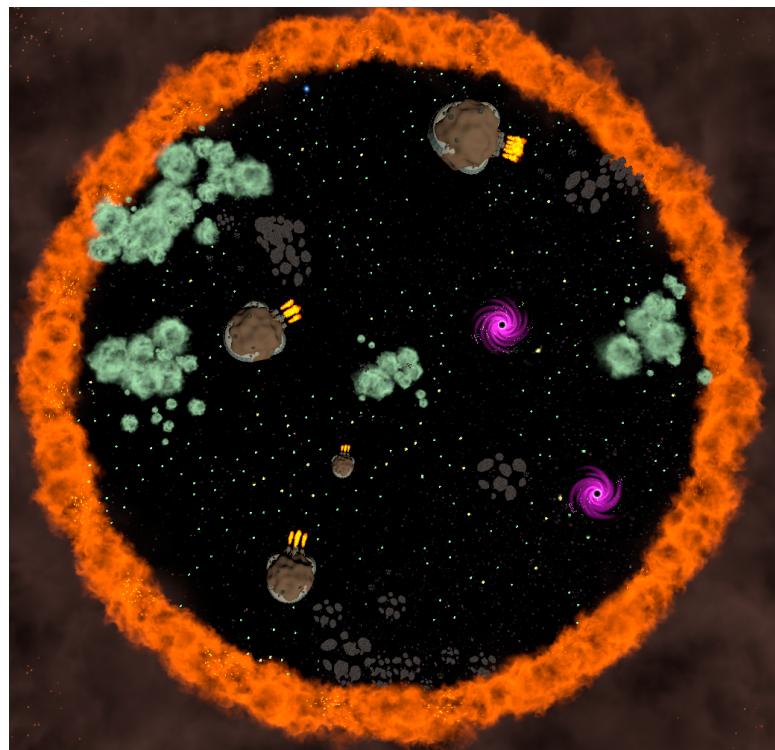
Daftar Isi

| | |
|---|-----------|
| Daftar Isi | 1 |
| Bab 1 | |
| Deskripsi Tugas | 2 |
| Bab 2 | |
| Landasan Teori | 5 |
| 2.1. Dasar Teori | 5 |
| 2.2. Cara Kerja Program | 6 |
| Bab 3 | |
| Aplikasi Strategi Greedy | 8 |
| 3.1. Proses Mapping Persoalan | 8 |
| 3.2. Alternatif Solusi Greedy | 8 |
| 3.3. Analisis Efisiensi dan Efektivitas | 9 |
| 3.4. Strategi Greedy Program | 10 |
| Bab 4 | |
| Implementasi dan Pengujian | 12 |
| 4.1. Implementasi Algoritma Greedy | 12 |
| 4.1.1. Implementasi ShieldBot | 12 |
| 4.1.2. Implementasi ShootBot | 14 |
| 4.1.3. Implementasi MoveBot | 17 |
| 4.1.4. Implementasi TeleportBot | 22 |
| 4.2. Struktur Data | 25 |
| 4.3. Pengujian | 27 |
| 4.3.1. Pengujian Fire Torpedoes (Peluru putih) | 27 |
| 4.3.2. Pengujian Fire Teleport (Peluru biru) | 28 |
| 4.3.3. Pengujian Forward menuju daerah dengan densitas makanan tinggi | 29 |
| 4.3.4. Pengujian EnemyEvade menghindari posisi musuh dengan size yang lebih besar | 29 |
| 4.3.5. Pengujian Activate Shield untuk menepis serangan torpedo musuh | 30 |
| Bab 5 | |
| Kesimpulan dan Saran | 32 |
| 5.1. Kesimpulan | 32 |
| 5.2. Saran | 32 |
| Daftar Pustaka | 33 |
| Lampiran | 34 |

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 melewatiinya 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

2.1. Dasar Teori

Algoritma *greedy* merupakan metode yang sering digunakan untuk menyelesaikan persoalan optimasi. Persoalan optimasi merupakan persoalan mencari solusi optimal. Persoalan optimasi terdiri dari dua macam, yaitu maksimasi (*maximization*) dan minimasi (*minimization*). *Greedy* merupakan kata dari bahasa inggris yang berarti rakus, tamak, atau loba. Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) dengan mengambil pilihan terbaik yang diperoleh pada saat itu tanpa mempertimbangkan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma *greedy* tidak selalu menghasilkan solusi yang optimum. Pada sebagian persoalan, algoritma *greedy* hanya dapat memberikan solusi yang sub-optimal.

Algoritma *greedy* memiliki enam elemen, yaitu:

1. Himpunan Kandidat, C
Berisi kandidat yang akan dipilih pada setiap langkah.
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. Bersifat heuristik.
5. Fungsi Kelayakan (*feasible*)
Memeriksa kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi Objektif
Memaksimumkan atau meminimumkan.

Berdasarkan elemen-elemen tersebut, algoritma *greedy* melakukan pencarian pada himpunan kandidat, C, yang akan menghasilkan sebuah himpunan bagian, S. S menyatakan suatu solusi dan S di optimisasi oleh fungsi objektif.

2.2. Cara Kerja Program

Sebelum mulai merancang program, unduhlah starter pack dari tautan berikut
<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

Agar permainan dapat dijalankan, terdapat beberapa *requirement* dasar yang diperlukan, yaitu:

1. Java (minimal 11) :
<https://www.oracle.com/java/technologies/downloads/#java8>
2. IntelliJ IDEA : <https://www.jetbrains.com/idea/>
3. .Net Core 3.1 :
 - [Windows](#)
 - [Linux](#)
 - [MacOS](#)

Untuk menjalankan permainan secara lokal di windows, dapat dilakukan dengan langkah-langkah berikut:

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.

Untuk mengimplementasikan algoritma *greedy* ke dalam bot, dapat dilakukan dengan mengubah JavaBot yang ada pada folder starter-bots. Bot melakukan aksi sesuai dengan aksi yang diterima pada playeraction. Pada program ini, keputusan untuk melakukan tiap aksi dilimpahkan kepada empat *microbot*, yaitu:

1. MoveBot
MoveBot mengatur keputusan terkait aksi player FORWARD dan STOP.
2. ShootBot
ShootBot mengatur keputusan terkait aksi player FIRETORPEDOES, FIRESUPERNOVA, dan DETONATE SUPERNOVA.

3. TeleportBot

TeleportBot mengatur keputusan terkait aksi player FIRETELEPORT dan TELEPORT.

4. ShieldBot

ShieldBot mengatur keputusan terkait aksi player ACTIVATESHIELD.

Empat *microbot* tersebut memiliki angka prioritas masing-masing. Setiap bot menentukan keputusan berdasarkan aksi yang diatur, lalu mengirimkan perintah untuk melakukan aksi tersebut beserta angka prioritasnya ke class BotProcessor menggunakan metode sendMessage. Lalu, pada BotProcessor aksi dengan prioritas tertinggi lah yang akan diperintahkan kepada bot untuk dilaksanakan.

Bab 3

Aplikasi Strategi *Greedy*

3.1. Proses Mapping Persoalan

Himpunan Kandidat

Seluruh *player action* yang ada pada permainan Galaxio, mencangkup: FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRESUPERNOVA, DETONATE SUPERNOVA, FIRETELEPORT, TELEPORT, dan ACTIVATESHIELD.

Himpunan Solusi

Berisi *player action* yang terpilih dari *player action* yang ada pada himpunan kandidat sesuai dengan fungsi seleksi.

Fungsi Solusi

Memeriksa apakah *player action* yang terpilih merupakan *player action* yang menghasilkan ukuran yang paling besar untuk bot.

Fungsi Seleksi

Memilih salah satu *player action* dengan angka prioritas paling tinggi diantara himpunan kandidat yang ada.

Fungsi Kelayakan

Memeriksa apakah *player action* yang baru dipilih bisa dilakukan oleh bot. Jika ACTIVATESHIELD, maka bot harus memiliki shield.

Fungsi Objektif

Memaksimumkan *size* bot.

3.2. Alternatif Solusi *Greedy*

Pada tugas besar ini, terdapat beberapa alternatif solusi *greedy* untuk mendapatkan solusi yang terbaik. Beberapa solusi tersebut antara lain:

1. Strategi *greedy* berdasarkan *distance*

Strategi ini sangat sederhana, yakni bot hanya perlu mencari makanan terdekat dan mendekatinya. Dengan memanfaatkan aksi FORWARD dan STOP,

bot dapat bergerak menuju makanan yang terdekat dengan bot. Strategi ini tidak memanfaatkan aksi selain FORWARD dan STOP.

2. Strategi *greedy* berdasarkan *size*

Pada strategi ini, perlu dihitung jumlah makanan terbanyak dalam lintasan tertentu. Setelah menemukan jumlah makanan terbanyak dengan lintasan x, maka bot akan bergerak melalui lintasan x tersebut. Sama dengan strategi *greedy* berdasarkan *distance*, strategi ini juga hanya memanfaatkan aksi FORWARD dan STOP.

3. Strategi *greedy* berdasarkan *damage*

Strategi ini mengandalkan torpedo dan supernova. Setiap torpedo atau supernova yang *ready* akan langsung ditembakkan ke musuh terdekat. Tujuannya agar meminimalkan *size* musuh.

4. Strategi *greedy* berdasarkan *position*

Strategi ini mengandalkan teleporter untuk menyerang musuh dari belakang. Bot akan menggunakan teleporter pada musuh yang lebih kecil. Tujuannya adalah memaksimalkan *size* bot dan meminimalkan *size* lawan.

5. Strategi *greedy* berdasarkan *size* dan *position*

Strategi ini menggabungkan strategi *greedy* berdasarkan *size* dan Strategi *greedy* berdasarkan *position*. Pada awal permainan, bot lebih fokus untuk mencari makan sampai sizenya besar. Lalu saat teleporter sudah tersedia, bot akan menggunakananya untuk teleport di belakang musuh yang lebih kecil.

3.3. Analisis Efisiensi dan Efektivitas

Untuk seluruh strategi *greedy* yang dirumuskan, berikut analisis efisiensi dan efektivitasnya:

1. Strategi *greedy* berdasarkan *distance*

Strategi ini dapat diimplementasikan dengan meminta list objek dari server dan menyaring objek yang berupa food serta mengurutkannya berdasarkan jarak, dari yang terdekat sampai ke yang terjauh dari bot. Strategi ini tidak terlalu menggunakan banyak resource, namun hasil yang didapat kurang optimal karena

food terdekat dari bot belum tentu memberikan keuntungan paling besar untuk bot. Strategi ini terbilang kurang efektif.

2. Strategi *greedy* berdasarkan *size*

Pada strategi ini proses penghitungan makanan terbanyak dalam lintasan tertentu memerlukan resource yang lebih dibanding dengan pendekatan *distance*. Namun, hasil yang didapat pun jauh lebih optimal dari strategi *greedy* berdasarkan *distance*. Kelemahan dari strategi ini, yaitu belum memanfaatkan aksi lain yang berpotensial mendapatkan keuntungan lebih. Strategi ini terbilang lumayan efektif.

3. Strategi *greedy* berdasarkan *damage*

Keefisienan strategi ini sedikit lebih membutuhkan resource dengan strategi pertama. Pada strategi ini memanfaatkan torpedo dan supernova, sehingga diperlukan adanya kalkulasi arah tembakan. Hasil yang didapat pun masih kurang apalagi mengingat setiap tembakan mengurangi ukuran bot, sehingga berpotensi merugikan untuk bot. Oleh karena itu, strategi ini kurang efektif.

4. Strategi *greedy* berdasarkan *position*

Strategi ini lebih membutuhkan resource daripada strategi pertama. Sebab perlu dilakukan kalkulasi arah penembakan teleporter dan pengaktifan teleport. Keefektivitasannya pun masih kurang optimal.

5. Strategi *greedy* berdasarkan *size* dan *position*

Strategi ini merupakan strategi yang menggunakan resource terbanyak, namun keuntungan yang diperoleh dari strategi ini berpotensi sangat besar. Dengan menggabungkan kelebihan dari strategi *size* dan strategi *position*, bot dapat memperoleh size yang maksimal, baik dari food maupun musuh.

3.4. Strategi *Greedy* Program

Pada program ini banyak diterapkan konsep *greedy by size and position* dengan perbandingan nilai size dan/atau position sebagai nilai densitas tertentu yang dimiliki masing-masing kandidat aktivitas yang akan dijalankan. Konsep ini dipilih karena dapat lebih mudah diterapkan dan disesuaikan dengan setiap aksi player yang disediakan oleh permainan.

Sebagai contoh dalam aktivitas pencarian makanan dan supernova pickup, digunakan nilai perbandingan **banyak makanan lain di sekitar sebuah makanan target (size)** dibagi

dengan **jarak yang harus ditempuh untuk mencapai target (position)**. Semakin dekat dan semakin ramainya lingkungan sebuah calon target maka nilai density akan semakin besar sehingga akan terpilih melalui algoritma greedy.

Bab 4

Implementasi dan Pengujian

4.1. Implementasi Algoritma *Greedy*

Berikut adalah implementasi algoritma *greedy* yang digunakan. Algoritma yang ditampilkan hanya mencangkup bagian-bagian yang signifikan dari program.

4.1.1. Implementasi ShieldBot

```
public class ShieldBot extends ActionCalculator implements ActionBot

    private final botProcessor : BotProcessor
    private final stateHolder : StateHolder
    private final playerAction : PlayerAction

    private function calculateTorpedoHit (torpedoList : List<GameObject>, botSize :integer)
    -> boolean
        {Mengkalkulasikan apakah torpedo akan mengenai bot atau tidak.
        Mengembalikan true jika torpedo akan mengenai bot dan false jika tidak}

        hit: boolean = false
        {variabel yang menunjukkan apakah player akan terkena torpedo atau tidak}
        hitCenterHeading : integer
        {Heading torpedo agar mengenai titik pusat bot}
        hitCenterDistance : integer
        {Jarak torpedo dengan titik pusat berdasarkan heading yang akan mengenai titik
        pusat}
        distanceToCenter : integer
        {Jarak torpedo dengan titik pusat berdasarkan header sebenarnya}
        headingDiff : integer
        {Selisih heading sebenarnya dari torpedo dengan heading agar mengenai titik
        pusat bot}
        bot : GameObject = stateHolder.getBot()
        currentTorpedo : GameObject
        i : integer = 0

        while (i < torpedoList.size() and not hit) do
            currentTorpedo <- torpedoList.get(i)
```

```

hitCenterHeading <- getHeadingBetween(currentTorpedo, bot)
hitCenterDistance <- (int) getDistanceBetween(currentTorpedo, bot)
headingDiff <- Math.abs(currentTorpedo.getCurrentHeading() -
hitCenterHeading)
if (headingDiff < 90) then
    distanceToCenter <- (integer) Math.tan(headingDiff) *
    hitCenterDistance
    hit <- (distanceToCenter < botSize)
    i++
-> hit

private hasShield (shieldCount : integer) -> boolean
{Mengembalikan true jika bot memiliki shield dan false jika tidak}
-> shieldCount > 0

private canUseTorpedo (size : integer, shieldCount : integer) -> boolean
{Mengembalikan true jika bisa memakai torpedo dan false jika tidak}
-> size > 30 and hasShield(shieldCount)

public void run ()
{Menjalankan kalkulasi secara menyeluruh dengan menggunakan fungsi-fungsi di
atas untuk menentukan apakah bot layak menggunakan shield atau tidak}

priorityNum : integer
{Angka prioritas}
gameState : GameState
botSize : integer
shieldCount : integer

if (stateHolder.getBot() = null) then
    botProcessor.sendMessage(playerAction, -1)
->

priorityNum <- 3
gameState <- stateHolder.getGameState()
playerAction.setAction(PlayerActionEn.ACTIVATESHIELD)
botSize <- stateHolder.getBot().getSize()
shieldCount <- stateHolder.getBot().getShieldCount()

List<GameObject> torpedoList <- getAll Torpedo from gameState

```

```

if (!torpedoList.isEmpty()) then
    if (canUseTorpedo(botSize, shieldCount) and
        calculateTorpedoHit(torpedoList, botSize)) then
        {Jika bisa memakai torpedo dan torpedo akan mengenai bot, maka akan
        dikirim pesan untuk melakukan ACTIVATESHIELD beserta angka
        prioritasnya ke botProcessor}
        botProcessor.sendMessage(playerAction, priorityNum)

```

4.1.2. Implementasi ShootBot

```

public class ShootBot extends ActionCalculator implements ActionBot

private final botProcessor : BotProcessor
private final stateHolder : StateHolder
private final playerAction : PlayerAction
private snFired : boolean
private largestPlayerId : UUID

public void run() {
    gameState : GameState
    bot : GameObject
    world : World
    sn : GameObject
    target : GameObject

    gameState <- stateHolder.getGameState()
    bot <- stateHolder.getBot()
    if (bot = null) then
        botProcessor.sendMessage(playerAction, -1)
        ->

    if (snFired and gameState.getGameObjects() != null and not
        gameState.getGameObjects().isEmpty()) then
        sn <- null
        for (GameObject object: gameState.getGameObjects())
            if (object.getGameObjectType() =
                ObjectTypeEn.SUPERNOVA_BOMB) then
                sn <- object
                break
        if (sn = null) then

```

```

        snFired <- false
        target <- stateHolder.getPlayerMap().get(largestPlayerId)
        world <- gameState.getWorld()
        if (sn != null and not isInRadius(bot, sn, 1.5*bot.getSize()) and
            (isInRadius(sn, target, target.getSize()) or not isInRadius(target,
            world.getCenterPoint(), 0.9 * world.radius))) then
            playerAction.setAction(PlayerActionEn.DETONATESUPERNOV
            A)
            botProcessor.sendMessage(playerAction, 5)
            snFired <- false

        if (!snFired) then
            ->

        if (bot.isSnAvailable()) then
            supernova(gameState.getPlayerGameObjects())
            ->

        if (stateHolder.getBot().getSize() < 25) then
            botProcessor.sendMessage(playerAction, -1)
            ->
        torpedoes(gameState.getPlayerGameObjects(), bot)

private void supernova(playerObjects : GameObject )
    largestPlayer : GameObject
    size : double

    if (playerObjects != null and playerObjects.isEmpty())
        ->

    largestPlayer <- null
    size <- -1, cSize
    for (GameObject player: playerObjects) {
        cSize <- player.getSize()
        if (largestPlayer = null or cSize > size) then
            if (player.getId() = stateHolder.getBot().getId()) then
                continue
            largestPlayer = player
            size = cSize
        if (largestPlayer = null) then
            ->

```

```

largestPlayerId <- largestPlayer.getId()
playerAction.setAction(PlayerActionEn.FIRESUPERNOVA)
playerAction.setHeading(getHeadingBetween(stateHolder.getBot(),
largestPlayer))
snFired <- true

botProcessor.sendMessage(playerAction, 4)

private void torpedoes(List<GameObject> playerObjects, GameObject bot)
closestPlayer : GameObject
closestDist : double

if (playerObjects != null and playerObjects.isEmpty()) then
->

closestPlayer <- null
closestDist <- -1, temp
for (GameObject player: playerObjects)
    temp <- getDistanceBetween(bot, player) - bot.getSize()
    if (closestPlayer = null or temp < closestDist) then
        if (player.getId() = bot.getId()) then
            continue
        closestPlayer <- player
        closestDist <- temp
    if (closestPlayer = null or closestDist-closestPlayer.getSize() >
torpedoThreshold(bot.getSize())) then
->
    playerAction.setAction(PlayerActionEn.FIRETORPEDOES)
    playerAction.setHeading(getHeadingBetween(stateHolder.getBot(), closestPlayer))

    botProcessor.sendMessage(playerAction, lerpInt(easeIn(100 /
clampDouble(getDistanceBetween(stateHolder.getBot(), closestPlayer) -
closestPlayer.getSize(), 100, 500)), 2, 4))

private double torpedoThreshold(double size)
if (size > 70) then
-> 1.8 * size
-> 2*size

```

4.1.3. Implementasi MoveBot

```
public class MoveBot extends ActionCalculator implements ActionBot

    private final botProcessor : BotProcessor
    private final stateHolder : StateHolder
    private final globalPlayerAction : PlayerAction

    abstract class MoveBotStrategy
        protected desireAmount : integer

        protected playerAction : PlayerAction = new PlayerAction()
        protected gameState : GameState = stateHolder.getGameState()

        {default constructor}
        public MoveBotStrategy()
            this.desireAmount <- -1 {value default}
            execute()

        public int getDesire()
            -> this.desireAmount

        abstract void execute()

    {Meng-handle pergerakan mencari food, super food, dan supernova}
    class FoodChase extends MoveBotStrategy
        private final foodAmountThreshold : integer = 40

        void execute()

            this.playerAction.action <- PlayerActionEn.FORWARD
            this.playerAction.heading <- new Random().nextInt(360)

            if (!this.gameState.getGameObjects().isEmpty()) then
                foodList : list of GameObjects
                for GameObject obj in this.gameState.getGameObjects() do :
                    if (obj.getGameObjectType() = ObjectTypeEn.FOOD or
                    obj.getGameObjectType() = ObjectTypeEn.SUPER_FOOD or
```

```

obj.getGameObjectType() = ObjectTypeEn.SUPERNOVA_PICKUP) then
    foodList.add(obj)

valueNearby : integer = 0
    tempGreedyVal : double
    maxGreedyVal : double = 0

for(GameObject item in foodList)
    for(GameObject nearbyItem : this.gameState.getGameObjects())
        if(getDistanceBetween(nearbyItem, item) > 200) then
            continue

        if(nearbyItem.getGameObjectType() = ObjectTypeEn.FOOD) then
            valueNearby <- valueNearby + 1
        if(nearbyItem.getGameObjectType() = ObjectTypeEn.SUPER_FOOD)
            valueNearby <- valueNearby + 3
        if(nearbyItem.getGameObjectType() =
ObjectTypeEn.SUPERNOVA_PICKUP) then
            valueNearby <- valueNearby + 9

tempGreedyVal <- valueNearby /
getDistanceBetween(stateHolder.getBot(), item)

if(maxGreedyVal < tempGreedyVal)then
    maxGreedyVal = tempGreedyVal
    this.playerAction.heading = getHeadingBetween(stateHolder.getBot(),
item)
    this.desireAmount = lerpInt(easeInOut((float)clampInt(valueNearby, 0,
this.foodAmountThreshold)/this.foodAmountThreshold), 2, 4)

valueNearby <- 0
tempGreedyVal <- 0

{Meng-handle pergerakan menghindari musuh yang lebih besar, gas_cloud,
asteroid_field, dan wormhole}
{wormhole dihindari karena output yang sangat random dan berbahaya bagi
pemain}
class EnemyEvade extends MoveBotStrategy

```

```

private safeSizeThreshold : int = 30

void execute()

    if(not this.gameState.getPlayerGameObjects().isEmpty()) then

        playerList : list of GameObject
        for GameObject obj in this.gameState.getPlayerGameObjects() do :
            if (obj.getGameObjectType() = ObjectTypeEn.PLAYER
                and item.getId() != stateHolder.getBot().getId()
                and item.getSize() - safeSizeThreshold > stateHolder.getBot().getSize()
                and getDistanceBetween(stateHolder.getBot(), item) - item.getSize() <
500)
                or
                (item.getGameObjectType() = ObjectTypeEn.GAS_CLOUD
                and getDistanceBetween(stateHolder.getBot(), item) - item.getSize() <
200)
                or
                (item.getGameObjectType() = ObjectTypeEn.ASTEROID_FIELD
                and getDistanceBetween(stateHolder.getBot(), item) - item.getSize() <
100)
                or
                (item.getGameObjectType() = ObjectTypeEn.WORMHOLE
                and getDistanceBetween(stateHolder.getBot(), item) - item.getSize() -
stateHolder.getBot().getSize() < 10)) then
                    playerList.add(obj)

            if(playerList.isEmpty()) then
                this.desireAmount <- -1
                ->

        midPoint : Position = Position.getCentroid(playerList)

        this.playerAction.action = PlayerActionEn.FORWARD
        this.playerAction.heading =
rotateHeadingBy(getHeadingBetween(stateHolder.getBot(), midPoint), 180)

```

```

        this.desireAmount =
        lerpInt(easeInOut(1/getDistanceBetween(stateHolder.getBot(), playerList.get(0))), 1,
        4)

    { Meng-handle pergerakan menghindari torpedo salvo dan teleporter musuh }
    class TorpedoEvade extends MoveBotStrategy
        void execute()
            if(!this.gameState.getPlayerGameObjects().isEmpty())
                torpedoList : List of GameObject

            for GameObject item in this.gameState.getGameObjects() do :
                if (item.getGameObjectType() = ObjectTypeEn.TORPEDO_SALVO
                    and getDistanceBetween(stateHolder.getBot(), item) - item.getSize() <
                    400)
                    or
                    (item.getGameObjectType() = ObjectTypeEn.TELEPORTER
                     and getDistanceBetween(stateHolder.getBot(), item) - 2 *
                     stateHolder.getBot().getSize() < 200) then
                        torpedoList.add(item)

                torpedoList.sorted(Comparator.comparing(item ->
                    getDistanceBetween(stateHolder.getBot(), item)))

            if(torpedoList.isEmpty()) then
                this.desireAmount = -1
                ->

                this.playerAction.action = PlayerActionEn.FORWARD
                this.playerAction.heading = torpedoList.get(0).getGameObjectType() =
                ObjectTypeEn.TORPEDO_SALVO ?
                (
                    getHeadingDifference(stateHolder.getBot().getCurrentHeading(),
                    rotateHeadingBy(torpedoList.get(0).getCurrentHeading(), 90)) < 180
                    ?

```

```

rotateHeadingBy(torpedoList.get(0).getCurrentHeading(), 90)
    :
rotateHeadingBy(torpedoList.get(0).getCurrentHeading(), -90)
    )
    :
    // TELEPORTER
(
rotateHeadingBy(getHeadingBetween(stateHolder.getBot(), torpedoList.get(0)), 180)
    )
this.desireAmount =
lerpInt(easeInOut(100/clampDouble(getDistanceBetween(stateHolder.getBot(),
torpedoList.get(0)), 100, 400)), 2, 4)

{Meng-handle pergerakan mengejar musuh yang lebih kecil}
class EnemyChase extends MoveBotStrategy
    private safeSizeThreshold : int = 20
    private safeMinimumSizeThreshold : int = 35

    void execute()
        if(!this.gameState.getPlayerGameObjects().isEmpty()) then

            smallerPL : List of GameObject
            for GameObject item in this.gameState.getPlayerGameObjects() do :
                if (player.getGameObjectType() = ObjectTypeEn.PLAYER and player.getId() != stateHolder.getBot().getId() and player.getSize() + safeSizeThreshold < stateHolder.getBot().getSize()) then
                    smallerPL.add(item)

            smallerPL.sorted(Comparator.comparing(player -> getDistanceBetween(stateHolder.getBot(), player)))
                .map(player -> player.getPosition())

            if(smallerPL.isEmpty() or stateHolder.getBot().getSize() <= safeMinimumSizeThreshold) then
                this.desireAmount = -1

```

```

->

    this.playerAction.action = PlayerActionEn.FORWARD
    this.playerAction.heading = getHeadingBetween(stateHolder.getBot(),
smallerPL.get(0))
    this.desireAmount =
lerpInt(easeOut(200/clampDouble(getDistanceBetween(stateHolder.getBot(),
smallerPL.get(0)), 200, 600)), 2, 4)

```

```

{runner code}
public void run()
if(stateHolder.getBot() = null)
    botProcessor.sendMessage(new PlayerAction(), -1)
->

toCalculate : List<MoveBotStrategy> = Stream.of(new FoodChase(),
new EnemyEvade(),
new EnemyChase(),
new TorpedoEvade())
.collect(Collectors.toList())

toExecute : MoveBotStrategy = toCalculate
.stream()
.sorted(Comparator.comparing(movebot -> movebot.getDesire()))
.collect(Collectors.toList()).get(toCalculate.size()-1)

botProcessor.sendMessage(toExecute.getPlayerAction(), toExecute.getDesire())

```

4.1.4. Implementasi TeleportBot

```

public class TeleportBot extends ActionCalculator implements ActionBot

private final botProcessor : BotProcessor
private final stateHolder : StateHolder
private final playerAction : PlayerAction

private prevTick : integer
private distance : double
private targetId : UUID

```

```

private prevTargetId : UUID
private oTargetPosition : Position

public void run()
    gameState : GameState
    bot : GameObject
    world : World
    distanceT : double

    gameState <- stateHolder.getGameState()
    bot = stateHolder.getBot()
    if (bot = null or gameState.getGameObjects() = null or
    gameState.getGameObjects().isEmpty() or bot.getSize() = 10 or
    bot.getTeleportCount() <= 0) then
        botProcessor.sendMessage(playerAction, -1)
    ->
    world <- gameState.getWorld()
    if (stateHolder.isTeleShot() and world.getCurrentTick() != prevTick) then
        distanceT = (world.getCurrentTick() - prevTick) * 25
        if (distanceT < distance) then
            botProcessor.sendMessage(playerAction, -1)
        ->
        tele()
    ->
    if (bot.getSize() > 100 and bot.getTeleportCount() > 0 and not
    stateHolder.isTeleShot()) then
        shootTele(gameState.getPlayerGameObjects())

private void tele()
    playerMap : Map<UUID, GameObject>
    target : GameObject

    playerAction.setAction(PlayerActionEn.TELEPORT)
    playerMap <- stateHolder.getPlayerMap()
    target <- playerMap.get(targetId), bot <- stateHolder.getBot()
    if (target = null or target.getSize() > bot.getSize()
    or not isInRadius(target, oTargetPosition, bot.getSize()) ) then
        botProcessor.sendMessage(playerAction, -1)
        stateHolder.setTeleShot(false)

```

```

->
botProcessor.sendMessage(playerAction, 5)

private void shootTele(List<GameObject> playerObjects) {
    target : GameObject

    if (playerObjects = null or playerObjects.isEmpty() or
stateHolder.getGameState().getWorld().getCurrentTick()=prevTick)
        botProcessor.sendMessage(playerAction, -1)
    ->

target <- null, bot <- stateHolder.getBot()
{set target and check for larger player around target}
for (GameObject player: playerObjects)
    if (player = null or player.getId() = bot.getId() or player.getId() =
prevTargetId or getDistanceBetween(bot, player) < 2*bot.getSize()) then
        continue
    if (player.getSize() > 0.8 * bot.getSize()) then
        if (target != null and player.getSize() > 1.2*bot.getSize() and
isInRadius(target, player, 1.5*player.getSize())) then
            target = null
        continue
    if (target = null or target.getSize() < player.getSize()) then
        target <- player
    if (target = null)
        botProcessor.sendMessage(playerAction, -1)
    ->

{check in gas cloud}
for (GameObject gameObject: stateHolder.getGameState().getGameObjects())
    if (gameObject.getGameObjectType() !=
ObjectTypeEn.GAS_CLOUD) then
        continue

    if (isInRadius(target, gameObject, gameObject.getSize())) then
        botProcessor.sendMessage(playerAction, -1)
    ->

prevTargetId <- targetId
targetId <- target.getId()
playerAction.setAction(PlayerActionEn.FIRETELEPORT)

```

```

playerAction.setHeading(getHeadingBetween(stateHolder.getBot(), target))
distance <- getDistanceBetween(bot, target)
prevTick <- stateHolder.getGameState().getWorld().getCurrentTick()
oTargetPosition <- target.getPosition()

botProcessor.sendMessage(playerAction, 4)

```

4.2. Struktur Data

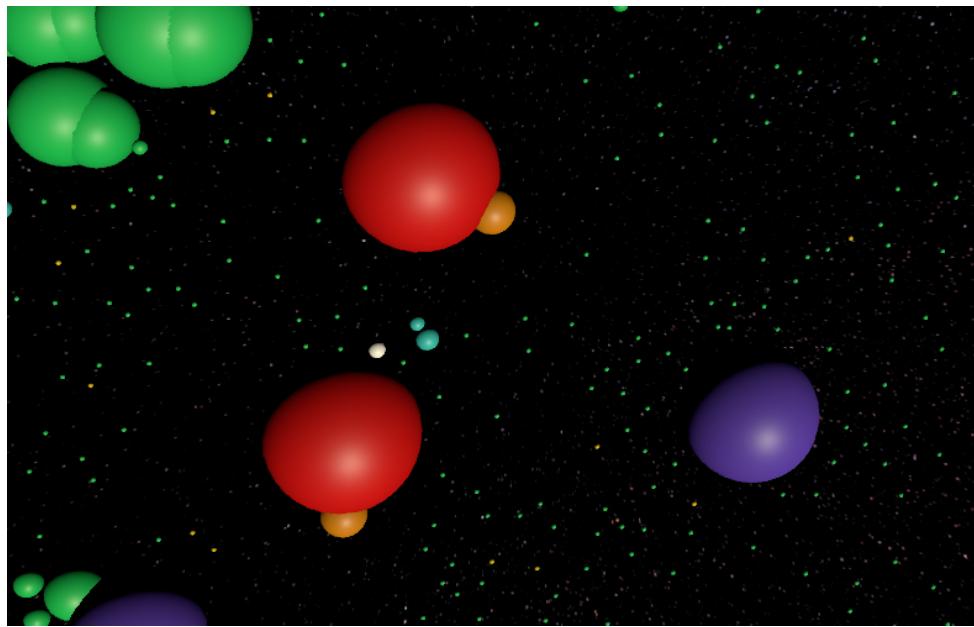
| Struktur Data dan Kelas | Penjelasan |
|--------------------------|--|
| Main.java | Program utama yang akan dijalankan. Saat dijalankan, kelas Main akan memanggil metode start() dari class ExecHandler. |
| ExecHandler.java | Class ini merupakan class yang menghubungkan algoritma <i>greedy</i> yang telah diimplementasikan dengan server untuk menjalankan galaxio. Class ini merupakan class <i>final</i> yang mengirimkan aksi yang akan dilakukan bot ke server. |
| ObjectTypeEn.java | Class ini berisikan objek-objek yang ada pada Galaxio, yaitu: PLAYER, FOOD, WORMHOLE, GAS_CLOUD, ASTEROID_FIELD, TORPEDO_SALVO, SUPERFOOD, SUPERNOVA_PICKUP, SUPERNOVA_BOM, TELEPORTER, dan SHIELD. |
| PlayerActionEn.java | Class ini berisikan aksi player, yaitu FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRESUPERNOVA, DETONATE SUPERNOVA, FIRETELEPORT, TELEPORT, dan ACTIVATESHIELD |
| PlayerEffectHandler.java | Class ini berisikan status efek pada player. |
| StateHolder.java | Class ini berisikan GameState. Selama permainan berjalan, GameState akan terus <i>diupdate</i> berdasarkan state yang |

| | |
|-----------------------|---|
| | dikirimkan oleh server. |
| GameObject.java | Class ini berisikan atribut-atribut yang ada pada sebuah objek. Terdapat 11 atribut yang dimiliki oleh sebuah objek, yaitu id, size, speed, currentHeading, position, gameObjectType, effectHashCode, torpedoCount, snAvailble, teleportCount, dan shieldCount. |
| GameState.java | Class ini menyimpan informasi state berupa world dan list semua objek yang ada di permainan. |
| GameStateDto.java | Class ini berfungsi sebagai perantara dalam mentransfer objek-objek permainan. |
| PlayerAction.java | Class ini menyimpan tiga atribut player, yaitu playerId, action, dan heading. Atribut action merupakan aksi yang akan dieksekusi oleh player saat run time. |
| Position.java | Class ini berisikan informasi terkait posisi suatu objek. Karena objek pada permainan ini dua dimensi, maka posisinya disimpan dalam variabel x dan y. |
| World.java | Class ini berisikan atribut centerPoint, radius, dan currentTick. Ketiga atribut tersebut menyusun world pada permainan Galaxio. |
| BotProcessor.java | Class ini berfungsi untuk menyimpan player action yang dikirim oleh <i>microbot</i> dan memutuskan action mana yang akan dieksekusi berdasarkan prioritasnya. |
| ActionBot.java | Class ini hanyalah merupakan interface dari metode run yang diimplementasikan oleh masing-masing <i>microbot</i> . |
| ActionCalculator.java | Class ini berisi beberapa metode yang dapat membantu membuat keputusan setiap <i>microbot</i> . Beberapa metode tersebut diantaranya getDistanceBetween, getHeadingBetween, dan |

| | |
|------------------|--|
| | rotateHeadingBy. |
| MoveBot.java | Class ini mengatur keputusan terkait aksi player FORWARD dan STOP. Mengimplementasikan metode run pada ActionBot.java dengan aksi yang diatur class ini. |
| ShootBot.java | Class ini mengatur keputusan terkait aksi player FIRETORPEDOES, FIRESUPERNOVA, dan DETONATE SUPERNOVA. Mengimplementasikan metode run pada ActionBot.java dengan aksi yang diatur class ini. |
| ShieldBot.java | Class ini mengatur keputusan terkait aksi player FIRETELEPORT dan TELEPORT. Mengimplementasikan metode run pada ActionBot.java dengan aksi yang diatur class ini. |
| TeleportBot.java | Class ini mengatur keputusan terkait aksi player ACTIVATESHIELD. Mengimplementasikan metode run pada ActionBot.java dengan aksi yang diatur class ini. |

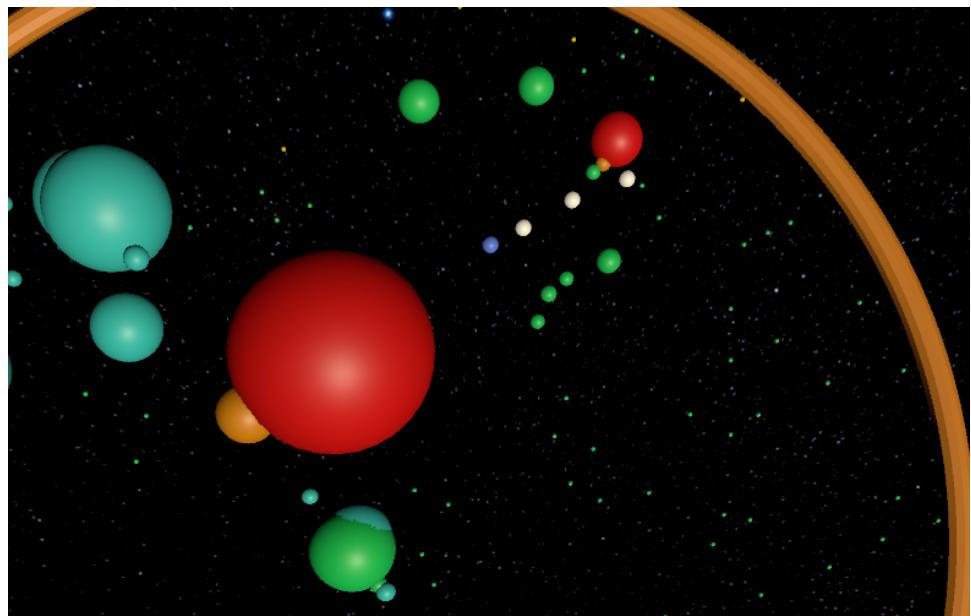
4.3. Pengujian

4.3.1. Pengujian Fire Torpedoes (Peluru putih)



Dapat dilihat player menembakkan torpedo ke arah musuh yang dekat. Algoritma greedy yang diimplementasi adalah *by position* karena terjadi pemilihan enemy terdekat dan *by size* karena terjadi pemilihan dengan kriteria size tertentu untuk target yang akan ditembak dan size tertentu untuk player boleh menembak.

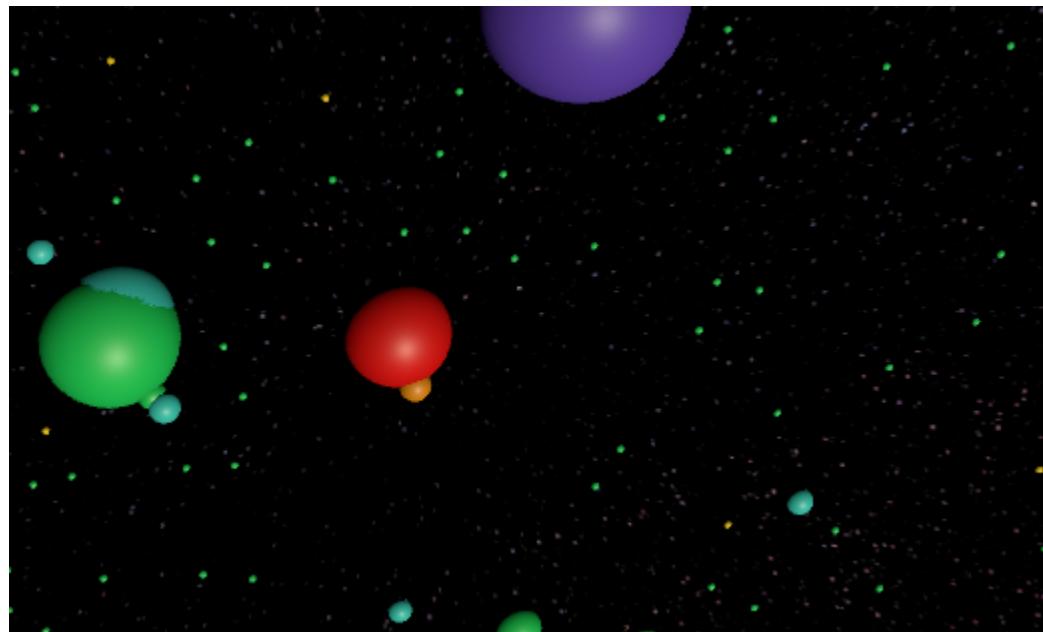
4.3.2. Pengujian Fire Teleport (Peluru biru)



Dapat dilihat player menembakkan teleporter ke arah musuh yang *viable* untuk didekati, yaitu musuh yang memiliki size sesuai kriteria dan jauh dari *hazard*, seperti gas cloud (hijau) dan asteroid (biru hijau). Secara implementasi, hal ini tercapai karena

program berhasil menembakkan teleporter ke arah enemy yang paling mencocokkan kriteria position dan size.

4.3.3. Pengujian Forward menuju daerah dengan densitas makanan tinggi



Dapat dilihat player bergerak mendekat food dengan densitas kluster terbesar, yaitu posisi yang dekat dengan super food (warna kuning) karena weight nya yang lebih besar. Secara desain algoritma, hal ini telah terpenuhi dengan penggunaan dan perhitungan nilai densitas greedy untuk masing masing food di sekitar player.

4.3.4. Pengujian EnemyEvade menghindari posisi musuh dengan size yang lebih besar



Dapat dilihat player berusaha menghindari kapal musuh yang memiliki size yang lebih besar. Dalam kasus berusaha menghindari, implementasi greed by size dan position untuk menjauhi musuh sudah terimplementasikan dengan baik meskipun terkadang tidak optimal seperti yang terlihat pada gambar di atas.

Karena nilai yang kurang optimal, player menjadi terlalu dekat dengan musuh yang mengejarnya dan juga mengarah menuju *asteroid field* yang juga seharusnya dihindari oleh player.

4.3.5. Pengujian Activate Shield untuk menepis serangan torpedo musuh



Dapat dilihat player sedang dalam posisi tembak-menembak torpedo dengan musuh. Melalui perhitungan value position torpedo, player memutuskan untuk menyalakan shield sehingga berhasil untuk menepis serangan torpedo yang dilakukan oleh kapal musuh. Beberapa optimisasi nilai yang dapat dimanfaatkan dalam skenario penggunaan shield contohnya adalah dengan menghemat penggunaan shield dengan

menyalakannya seterlambat mungkin (torpedo memiliki posisi yang dekat dengan player).

Bab 5

Kesimpulan dan Saran

5.1. Kesimpulan

Berdasarkan Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 2022/2023 berjudul Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan “Galaxio”, kami mendapati bahwa untuk merancang dan mengimplementasikan bot pada permainan “Galaxio” dapat dilakukan dengan menggunakan strategi *greedy*. Penyelesaian dengan strategi *greedy* yaitu mengoptimalkan situasi pada setiap lingkup lokal, dengan harapan dapat tercapainya kondisi yang optimum pada lingkup global. Dengan menggunakan strategi *greedy*, terdapat beberapa alternatif yang dapat digunakan, yaitu *greedy by distance*, *greedy by size*, *greedy by damage*, *greedy by position*, dan *greedy by size and position*. Setiap alternatif tersebut memiliki kelebihan dan kekurangan tersendiri. Meskipun tujuan strategi *greedy* untuk mencari global optimum, namun pada kenyataannya hasil yang didapat tidak selalu optimum. Namun, meskipun tidak optimum, hasil yang didapatkan paling tidak termasuk hasil yang sub-optimum.

5.2. Saran

Adapun saran-saran yang dapat kami berikan terkait Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 2022/2023:

1. Algoritma *greedy* yang digunakan pada Tugas Besar ini tentu belum sempurna. Masih ada beberapa kekurangan yang bisa diperbaiki terkait keoptimuman hasil yang diperoleh.
2. Beberapa implementasi program masih bisa lebih efisien, sehingga *resource* yang digunakan lebih sedikit.
3. Dokumentasi pemrograman masih bisa dilengkapi.
4. Server game engine memiliki kemungkinan untuk *overload*, sehingga banyak request yang tidak di-register di game meskipun sudah dikirim.

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)

Lampiran

Link GitHub: <https://github.com/zidane-itb/tubes1-stima.git>

Link Video: <https://www.youtube.com/watch?v=FVGjbYGB8Pw>