

## **Tugas Besar I - IF2211 Strategi Algoritma**

### **Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”**



**Disusun Oleh :**

Bintang Hijriawan Jachja / 13521003 / K-03

Henry Anand Septian Radityo / 13521004 / K-03

Christophorus Dharma Winata / 13521009 / K-03

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

## DAFTAR ISI

<b>Halaman Cover</b>	<b>1</b>
<b>BAB I-DESKRIPSI TUGAS</b>	<b>3</b>
<b>BAB II-LANDASAN TEORI</b>	<b>6</b>
<b>BAB III-APLIKASI STRATEGI GREEDY</b>	<b>8</b>
3.1 Mapping persoalan Galaxio menjadi elemen algoritma Greedy	8
3.2 Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Galaxio dan analisis efektif dan efisiensi alternatif solusi tersebut	8
3.3 Strategi Greedy yang dipilih dan pertimbangannya	10
<b>BAB IV-IMPLEMENTASI DAN PENGUJIAN</b>	<b>17</b>
4.1 Pseudocode	17
4.1.1 Prosedur computeNextPlayerAction(PlayerAction playerAction)	17
4.1.2 Function Terkepung(List of Game Object nearestPlayer, GameObject bot, GameObject scanMusuh)	21
4.1.3 Prosedur Idle (List of GameObject nearestTorpedos, List of GameObject nearstGasCloud, List of GameObject foodList, List of GameObject superFoodList)	22
4.2 Struktur Data	23
4.2.1 Game Object	23
4.2.2 Game State	23
4.2.3 Game State DTO	23
4.2.4 Player Action	24
4.2.5 Position	24
4.2.6 World	24
4.2.7 Object Types	24
4.2.8 Player Action	24
4.3 Pengujian	25
4.3.1 Test Case I	25
4.3.2. Test Case II	28
4.3.3 Test Case III	30
<b>BAB V-KESIMPULAN DAN SARAN</b>	<b>32</b>
5.1 Kesimpulan	32
5.2 Saran	32
5.3 Komentar dan Refleksi	32
<b>DAFTAR PUSTAKA</b>	<b>33</b>
<b>LAMPIRAN LINK BONUS VIDEO</b>	<b>34</b>

## **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 agar bot kapal anda yang tetap hidup hingga akhir permainan. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.

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

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

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

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

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Galaxio pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di integer x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu *Players*, *Food*, *Wormholes*, *Gas Clouds*, *Asteroid Fields*. Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan x. Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. Heading dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek afterburner akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 *salvo charge*. Penembakan *salvo*

*torpedo* (ukuran 10) mengurangkan ukuran kapal sebanyak 5.

3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi oleh kapal player. Apabila player mengkonsumsi *Food*, maka *Player* akan bertambah ukuran yang sama dengan *Food*. *Food* memiliki peluang untuk berubah menjadi *Super Food*. Apabila 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 melewatinya dengan syarat wormhole lebih besar dari kapal player.

5. *Gas Clouds* akan tersebar pada peta. Kapal dapat melewati *gas cloud*. Setiap kapal bertabrakan dengan *gas cloud*, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan *gas cloud*, maka efek pengurangan akan hilang.

6. *Torpedo Salvo* akan muncul pada peta yang berasal dari kapal lain. *Torpedo Salvo* berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. *Torpedo Salvo* dapat mengurangi ukuran kapal yang ditabraknya. *Torpedo Salvo* akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.

7. *Supernova* merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakkannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi *gas cloud*.

8. *Player* dapat meluncurkan *teleporter* pada suatu arah di peta. *Teleporter* tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. *Player* tersebut dapat berpindah ke tempat *teleporter* tersebut. Harga setiap peluncuran *teleporter* adalah 20. Setiap 100 tick player akan mendapatkan 1 *teleporter* dengan jumlah maximum adalah 10.

9. Ketika kapal *player* bertabrakan dengan kapal lain, maka kapal yang lebih besar akan

dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran *maximum* dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

10. Terdapat beberapa *command* yang dapat dilakukan oleh player. Setiap *tick*, player hanya dapat memberikan satu *command*. Berikut jenis-jenis dari *command* yang ada dalam permainan:

- a. FORWARD
- b. STOP
- c. START\_AFTERRBURNER
- d. STOP\_AFTERRBURNER
- e. FIRE\_TORPEDOES
- f. FIRE\_SUPERNOVA
- g. DETONATE\_SUPERNOVA
- h. FIRE\_TELEPORTER
- i. TELEPORTUSE\_SHIELD

11. Setiap *player* akan memiliki *score* yang hanya dapat dilihat jika permainan berakhir. *Score* ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal *player* lain, maka *score* bertambah 10, jika mengonsumsi *food* atau melewati *wormhole*, maka *score* bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan *score* tertinggi.

## **BAB II**

### **LANDASAN TEORI**

Algoritma *greedy* merupakan algoritma yang digunakan untuk memecahkan masalah yang berhubungan dengan optimasi. Optimasi dibagi menjadi dua jenis yaitu maksimum dan minimum. Sesuai namanya yaitu greedy atau rakus, algoritma ini bekerja dengan langkah demi langkah yang mengambil solusi yang paling ideal pada setiap langkahnya. Setelah mengambil solusi terbaik di tiap langkahnya, maka diharapkan algoritma *greedy* dapat menghasilkan solusi yang terbaik dalam skala global atau keseluruhannya.

Pada pembuatan algoritma untuk bot game Galaxio, algoritma greedy dimanfaatkan untuk menentukan aksi dari bot pada galaxio pada setiap satuan waktu. Pengambilan keputusan ini didasari oleh hal apa saja yang terdapat pada sekitar bot dengan mendeteksi objek - objek yang ada pada arena. Objek - objek tersebut antara lain seperti *torpedo*, *player* lain, *asteroid*, dan *gas cloud*. Selain mendeteksi objek - objek, bot juga mendeteksi jarak pada arena agar dapat mempertahankan posisi di dalam *safe zone*.

Secara garis besar, bot memiliki beberapa state untuk melakukan sesuatu yang diaktifkan dengan syarat tertentu. Bot memiliki 2 state utama yaitu *combat mode* dan *non-combat mode*, sesuai namanya *combat mode* digunakan agar bot dapat bertahan atau menyerang melawan bot lain dan combat mode menjadi prioritas dari bot untuk melakukan aksi. Pada *non-combat mode*, bot dapat melakukan sesuatu yang berhubungan dengan makanan dan menghindar, seperti memakan *superfood*, *food*, menghindari *gas cloud* dan berbagai objek yang dapat merugikan bot.

Implementasi algoritma ini disusun dengan menggunakan barisan kode if-else sehingga dapat memudahkan urutan prioritas yang harus dilakukan. Perintah tersebut ditambahkan ke dalam fungsi `computeNextPlayerAction` yang berada pada file `BotService.java`. Selain itu, penulis juga menambahkan fungsi - fungsi lain pada file `BotService.java` untuk memudahkan pemanggilan dan mengurangi repetisi.

Untuk menjalankan bot dengan algoritma yang sudah dibuat dibutuhkan file eksekusi bertipe .jar yang dapat dibuat dengan menggunakan bantuan perangkat lunak IntelliJ Idea atau menggunakan maven dengan mengetikkan mvn clean package pada direktori Javabot. Setelah file .jar selesai dibuat, maka file dapat dijalankan dengan membuat suatu file .bat untuk windows dan .sh untuk linux. Untuk isi file .bat dapat dilihat pada kode berikut

```
@echo off
:: Game Runner
cd ./runner-publish/
start "" dotnet GameRunner.dll

:: Game Engine
cd ../engine-publish/
timeout /t 1
start "" dotnet Engine.dll

:: Game Logger
cd ../logger-publish/
timeout /t 1
start "" dotnet Logger.dll

:: Bots
timeout /t 3
start "" java -jar ::direktori file jar, sesuaikan jumlah kode start java -jar
dengan jumlah bot
cd ../

pause
```

## BAB III

### APLIKASI STRATEGI GREEDY

#### **3.1 Mapping persoalan Galaxio menjadi elemen algoritma Greedy**

- Himpunan Kandidat: Aksi yang dapat dilakukan dalam game Galaxio
- Himpunan Solusi: Aksi yang diambil setelah melalui kode yang dibuat
- Fungsi Solusi: Fungsi size() dari game Galaxio yang menunjukkan nyawa dari bot
- Fungsi Seleksi: Mencari aksi yang paling mendukung ketahanan hidup bot
- Fungsi Kelayakan: Fungsi computeNextPlayerAction() yang dibuat penulis sebagai pemilih aksi yang terbaik yang bisa dilakukan. Fungsi ini akan dijelaskan lebih lengkap selanjutnya.
- Fungsi Obyektif: Nyawa yang dimiliki maksimal dan nyawa yang dimiliki lawan minimum

#### **3.2 Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Galaxio dan analisis efektif dan efisiensi alternatif solusi tersebut**

Di dalam persoalan Galaxio, terdapat alternatif solusi yang juga menerapkan strategi Greedy. Berikut adalah alternatif-alternatif solusi tersebut:

1. Greedy untuk maksimasi makan makanan dengan jarak yang terkecil

Strategi greedy ini merupakan strategi untuk mencari makan sebanyak-banyaknya dalam waktu dan jarak terkecil. Strategi ini mengabaikan aspek lain yang terdapat di game seperti menembak torpedo, teleportasi, shield, supernova, bahkan keberadaan musuh. Dengan mengabaikan sebagian besar aspek dari persoalan, strategi ini rentan terhadap kekalahan karena bot dapat bertabrakan dengan musuh yang berada dekatnya. Selain itu, terdapat ancaman lain seperti tembakan torpedo, gas cloud, dan supernova.

- Himpunan Kandidat: Berjalan menuju makanan
- Himpunan Solusi: Berjalan ke makanan dengan jarak yang terdekat
- Fungsi Solusi: Fungsi size() dari game Galaxio yang menunjukkan nyawa dari bot
- Fungsi Seleksi: Mencari jarak makanan
- Fungsi Kelayakan: Memeriksa jika makanan yang dituju adalah yang paling dekat
- Fungsi Obyektif: Nyawa yang dimiliki maksimal

## 2. Greedy untuk maksimasi eliminasi musuh dengan pengejaran musuh terdekat

Strategi greedy ini merupakan strategi yang lebih agresif untuk mengalahkan musuh dengan mengejar musuh. Strategi ini lebih efektif dari strategy greedy makanan karena bot dengan sendirinya mendapat makanan selama mengejar. Strategi ini memperbesar kemungkinan bot untuk menang dengan mengalahkan musuh, namun terdapat kelemahan dimana musuh memiliki ukuran yang besar.

- Himpunan Kandidat: Berjalan mengejar musuh-musuh yang ada di game
- Himpunan Solusi: Mengejar musuh yang paling dekat
- Fungsi Solusi: Fungsi `!gameState.getGameObjects().isEmpty()` dari game Galaxio yang menghasilkan `false` jika game masih berjalan. Keberjalan game pun dihentikan jika sudah tidak ada player di dalam game.
- Fungsi Seleksi: Mencari jarak musuh
- Fungsi Kelayakan: Memeriksa jika musuh yang dikejar adalah yang paling dekat
- Fungsi Obyektif: Minimasi jumlah musuh

## 3. Greedy untuk maksimasi jarak dengan player musuh

Strategi greedy ini merupakan strategi defensif untuk menghindari musuh dengan lari sejauh mungkin dari musuh-musuh. Strategi ini memungkinkan bot untuk bertahan hidup lebih lama, namun strategi ini tidak memanfaatkan fitur-fitur lain dari bot yang seharusnya bisa dimanfaatkan untuk memperbesar kemungkinan kemenangan. Strategi ini juga tidak mempertimbangkan kasus dimana musuh menembakkan torpedo dari jauh atau teleportasi ke tempat bot berada.

- Himpunan Kandidat: Aksi berjalan menjauhi musuh.
- Himpunan Solusi: Aksi berjalan kearah berlawanan dari arah musuh terdekat
- Fungsi Solusi: Fungsi `getHeadingBetween()` dari game galaxio menghasilkan arah dari bot ke suatu *GameObject*.
- Fungsi Seleksi: Mencari arah menjauhi musuh terdekat
- Fungsi Kelayakan: Arah yang dituju berlawanan dengan arah ke musuh terdekat
- Fungsi Obyektif: Berada di posisi terjauh dari musuh

## 4. Greedy untuk maksimasi eliminasi musuh dengan mengincar musuh terkecil

Strategi ini adalah strategi yang bersifat agresif dengan mengincar musuh terkecil. Terdapat banyak kelemahan dari strategi ini seperti musuh yang ada di game dapat

berubah ukuran dengan cepat baik mengecil atau membesar. Bot dengan strategi ini akan sering terjebak dalam menentukan target selanjutnya jika musuh-musuh di dalam game sedang bertempur dimana sering terjadi perubahan ukuran.

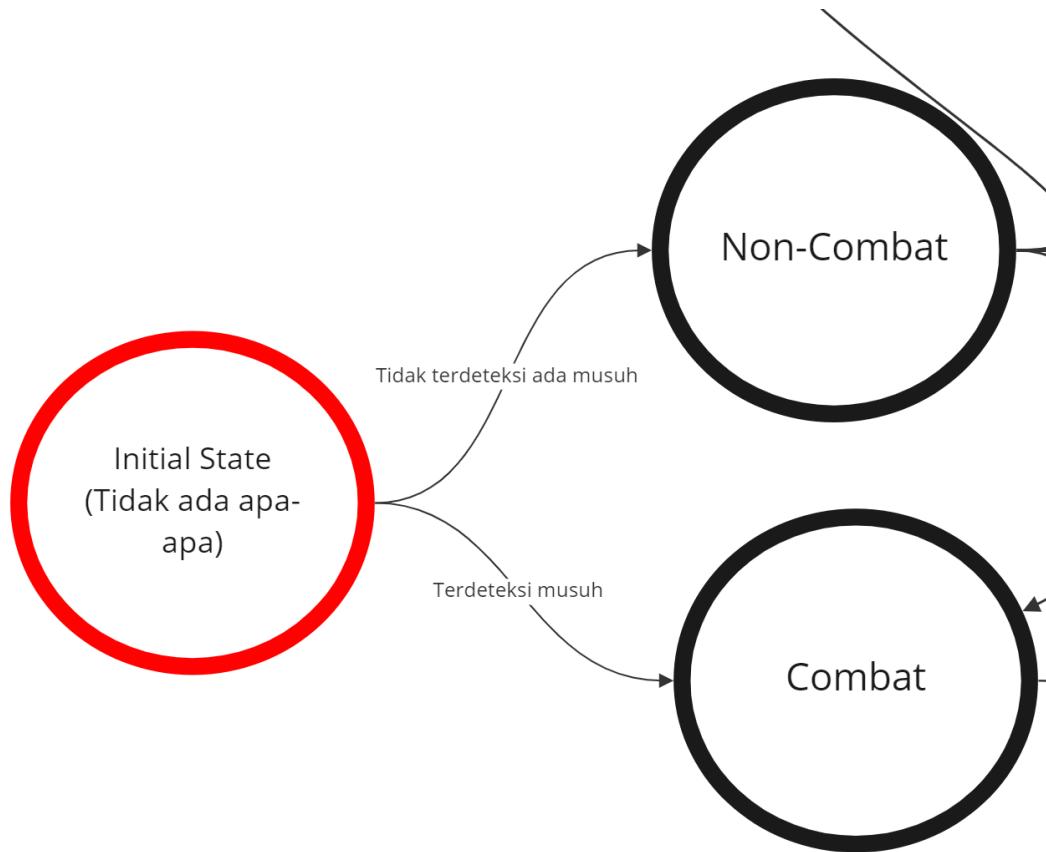
- Himpunan Kandidat: Berjalan mengejar musuh-musuh yang ada di game
- Himpunan Solusi: Mengejar musuh yang paling kecil
- Fungsi Solusi: Fungsi `gameState.getGameObjects()` dari game Galaxio yang menandakan jika game masih berjalan. Keberjalan game pun dihentikan jika sudah tidak ada objek di dalam game. Fungsi tersebut juga dapat dibuat menyaring khusus untuk objek *Player* dan diurutkan berdasarkan ukuran objek tersebut dalam implementasi bahasa Java.
- Fungsi Seleksi: Mencari ukuran musuh
- Fungsi Kelayakan: Memeriksa jika musuh yang dikejar adalah yang paling kecil
- Fungsi Obyektif: Minimasi jumlah musuh

### 3.3 Strategi Greedy yang dipilih dan pertimbangannya

Strategi Greedy yang dipilih adalah gabungan dari algoritma-algoritma greedy dengan aplikasi heuristik. Strategi yang disusun mempergunakan semua aksi yang dapat dilakukan di dalam game, mengevaluasi kondisi-kondisi yang dapat terjadi, serta membuat batasan heuristik dalam setiap eksekusi-eksekusi aksi.

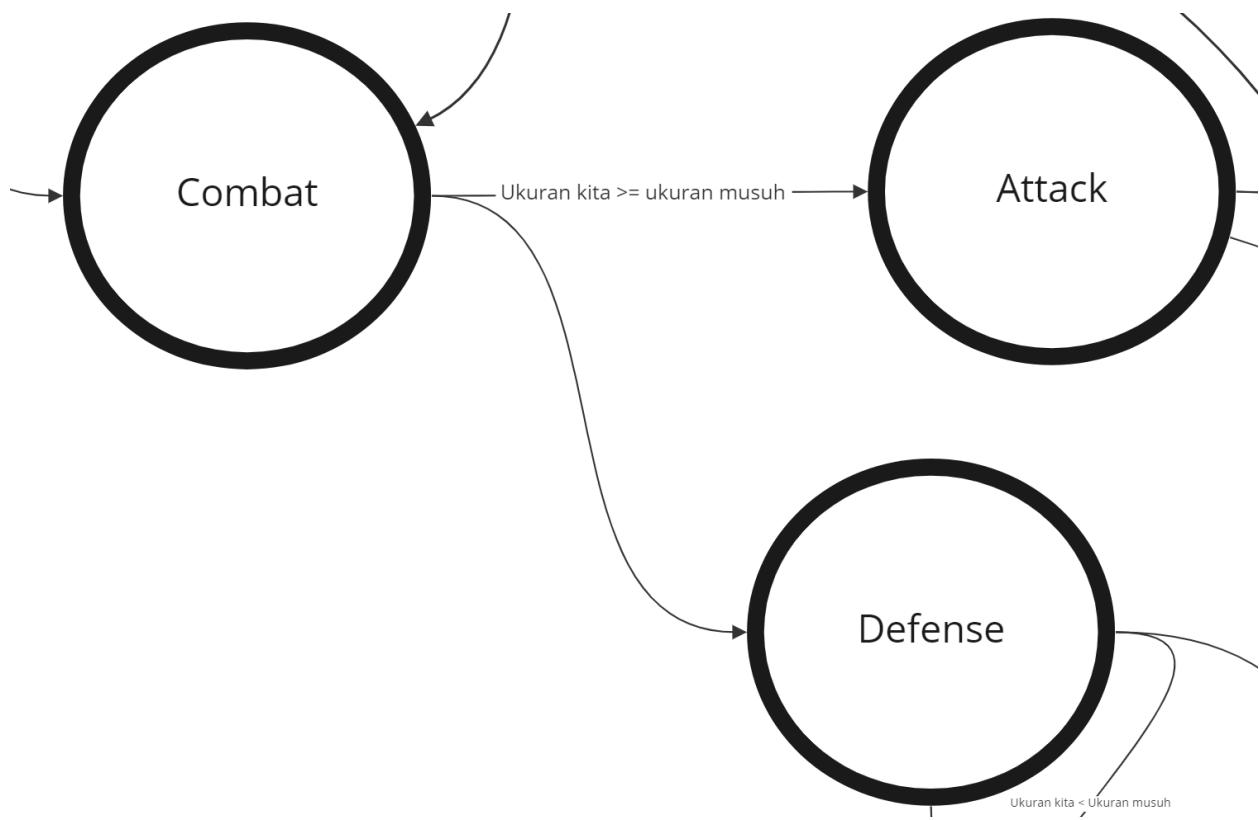
Untuk mempermudah penjelasan, penulis menggunakan *state diagram* untuk merumuskan rancangan strategi yang penulis buat. Pada diagram tersebut, garis transisi yang tidak bertulisan berarti state secara *default* langsung berpindah ke ke state setelah transisi tanpa input atau evaluasi.

*Initial state* atau state awal adalah ketika game belum dimulai, sedangkan *final state* adalah state ketika game telah berakhir. Ketika game dimulai, bot akan masuk kedalam salah satu diantara *combat* state dan *non-combat* state. *Combat* state adalah state ketika terdapat musuh di sekitar bot dalam jarak tertentu, yaitu 7 kali ukuran bot ditambah ukuran player terdekat. Penulis mengutamakan state ini daripada *non-combat* state karena penulis memprioritaskan bot untuk bertahan hidup.

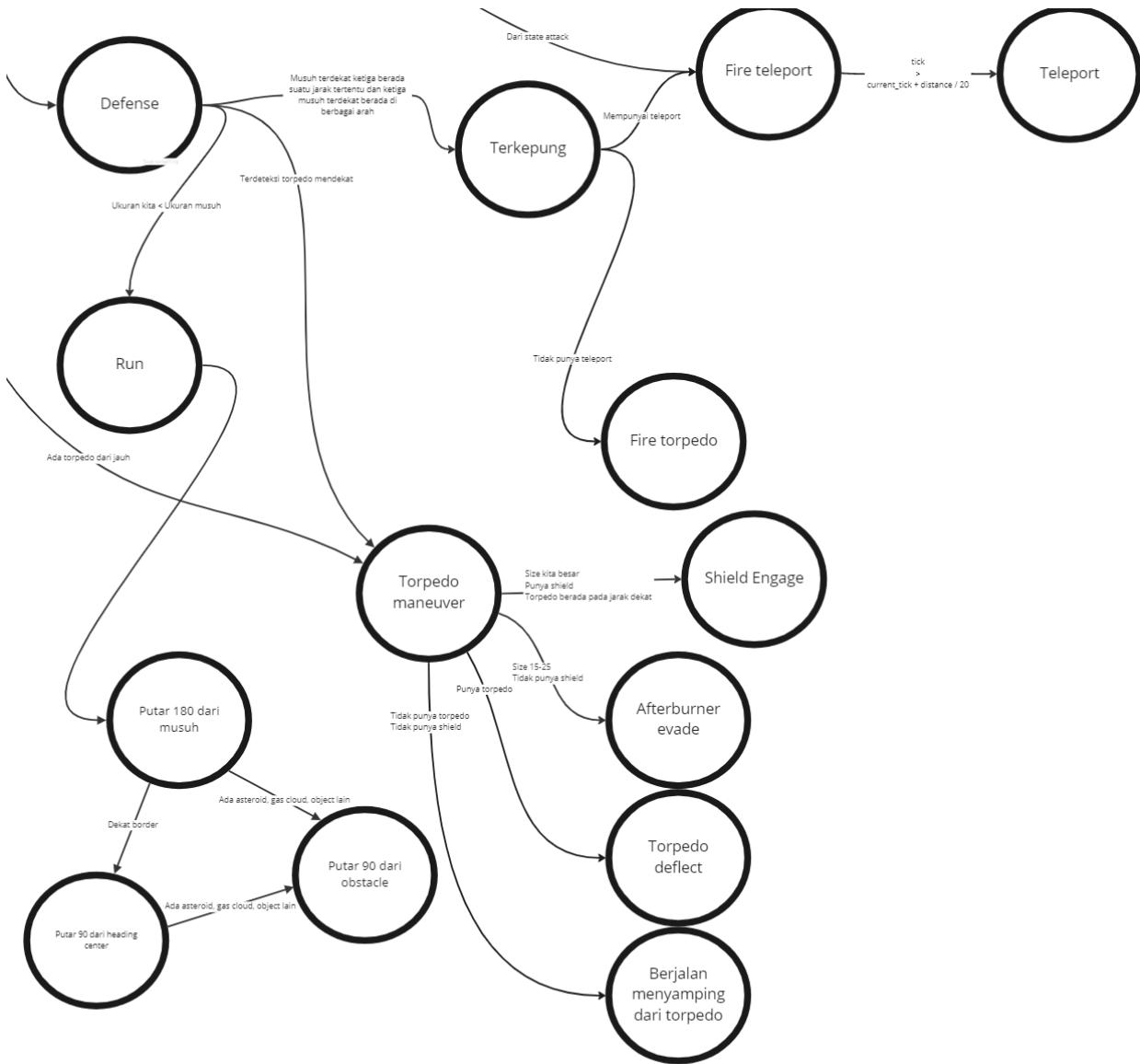


Gambar 3.1 State combat dan non-combat

Setelah masuk state *combat*, bot akan masuk kedalam state *attack* atau *defense*. Bot akan diutamakan masuk ke state *defense* karena tujuan permainan adalah bertahan hingga akhir. Bot akan masuk ke state *attack* ketika ada suatu kondisi yang terpenuhi. Jika tidak, maka bot akan masuk kedalam state *defense*.



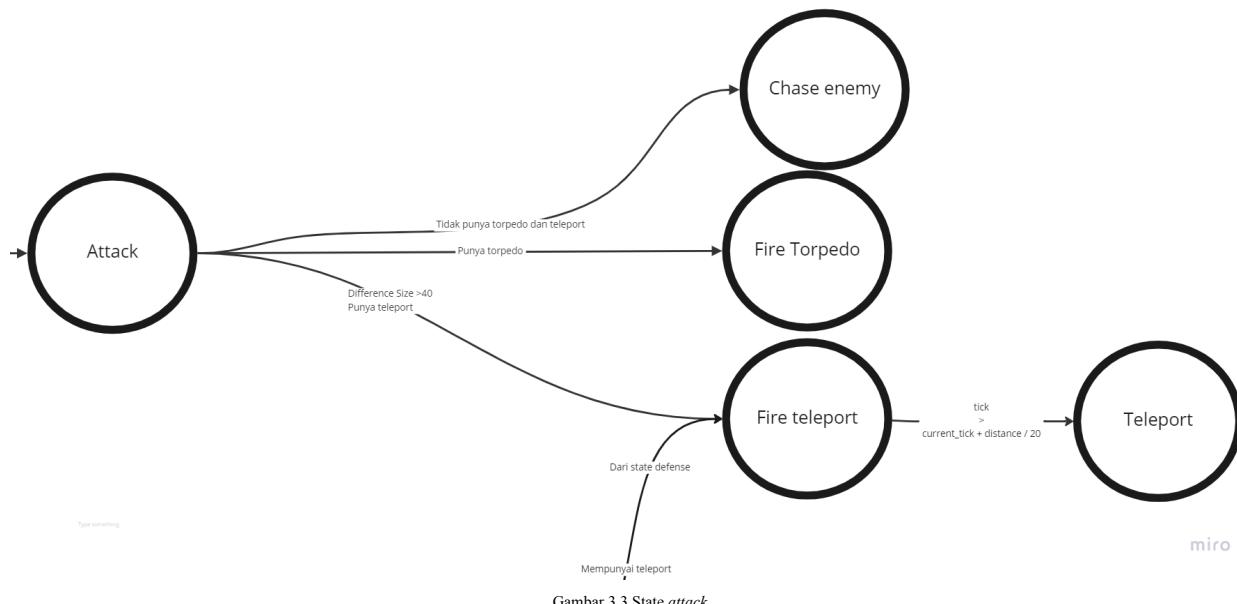
Gambar 3.2 State Combat



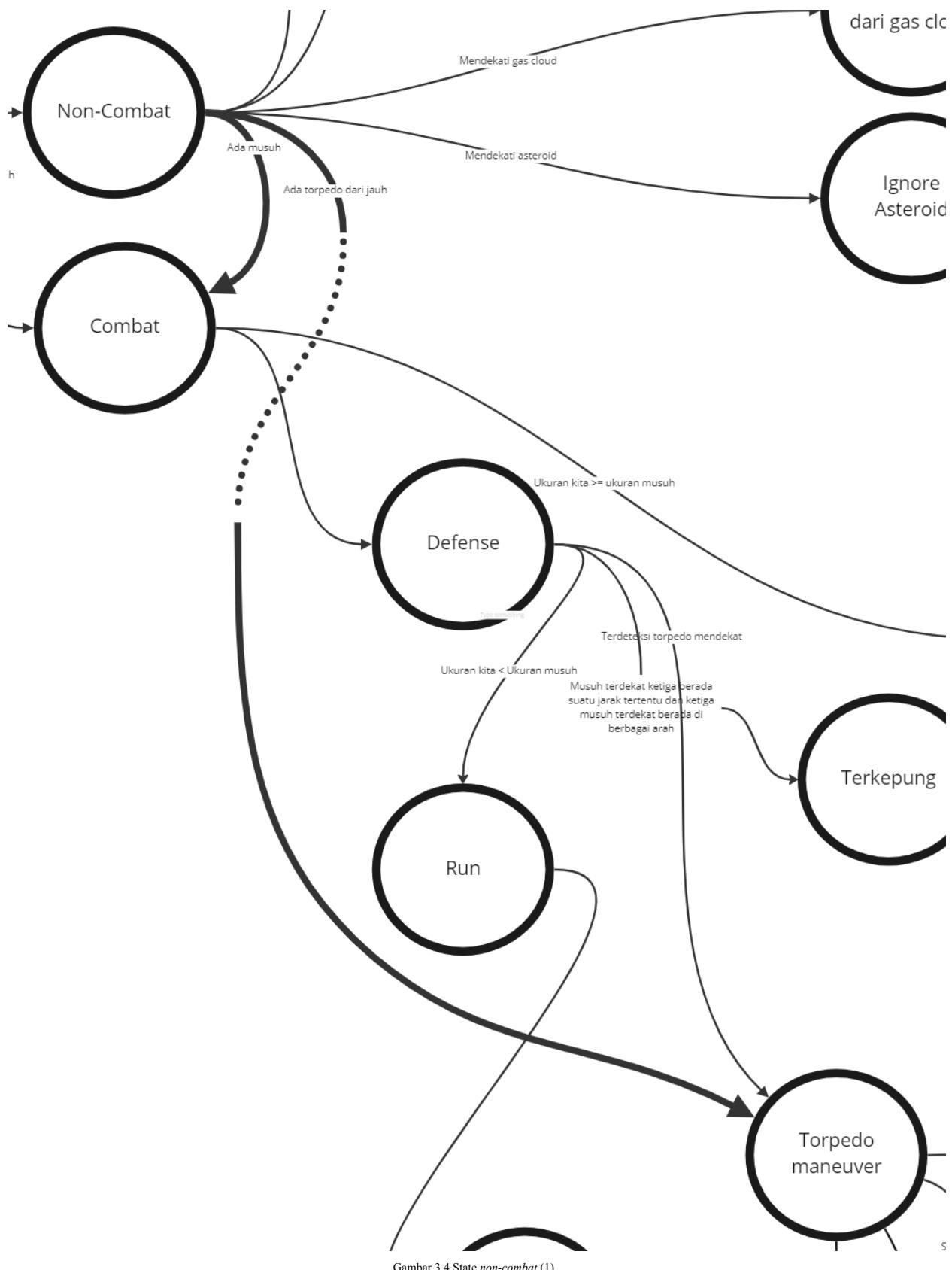
Gambar 3.3 State defense

Setelah masuk state *combat*, bot akan mendeteksi keadaan sekitar dan memeriksa apakah bot harus masuk kedalam state *defense* atau tidak. Kondisi yang diperiksa adalah apakah bot harus masuk ke salah satu dari 3 state yaitu run, torpedo maneuver, dan terkepung. Bot akan memasuki state *defense* dan state terkepung ketika ada 3 atau lebih player musuh dalam jarak 4 kali ukuran bot. Jika bot sudah masuk state terkepung, maka bot akan melakukan salah satu aksi antara menembakan teleport untuk lari jika memiliki teleport atau menembakan torpedo kearah player terdekat. Jika bot mendeteksi torpedo yang mengarah ke arah bot, maka bot akan masuk ke state *defense* dan state torpedo maneuver. Bot akan melakukan salah satu aksi antara menyalakan

shield, menghindari torpedo atau melawan dengan torpedo sendiri. Jika bot tidak memasuki kedua state tadi, bot akan memeriksa ukuran player terdekat. Jika ukuran player terdekat lebih besar dari ukuran bot, maka bot akan masuk ke state *defense* dan kemudian ke state *run*. Bot akan mencoba lari dari player terdekatnya sambil mencoba menghindari hal berbahaya lainnya.

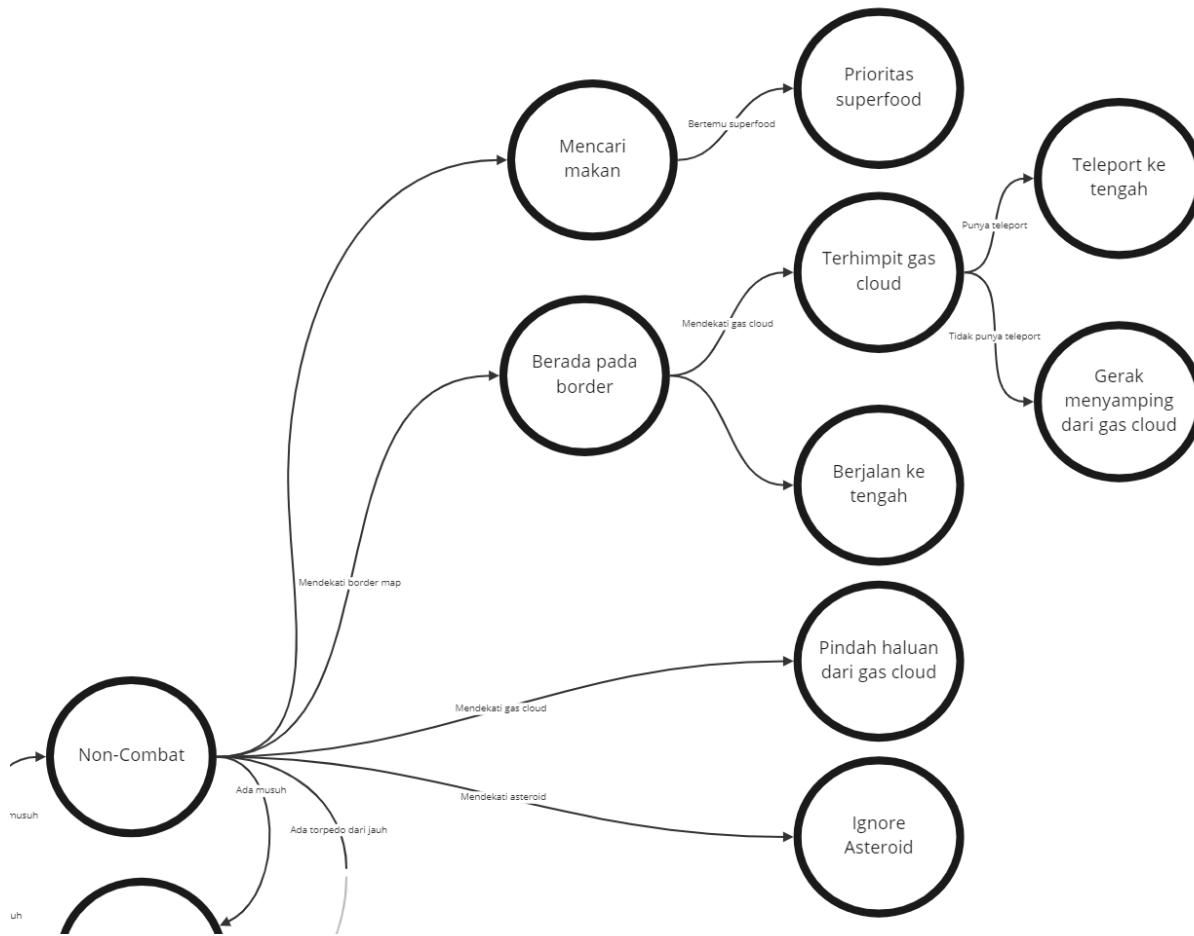


Jika kondisi-kondisi tersebut tidak terpenuhi, bot akan masuk ke state *attack*, dimana bot akan mencoba mengejar, menembak, dan mencoba teleport ke arah musuh terdekat yang lebih kecil. Bot akan mengutamakan aksi menembak dengan teleport jika kondisinya memungkinkan, yaitu ukuran bot dikurang 40 lebih besar dari ukuran player terdekat dan bot memiliki peluru teleport. Jika kondisi tersebut tidak dipenuhi, bot akan mencoba menembakan torpedo ke arah musuh jika bot memiliki peluru torpedo dan ukuran bot lebih dari 26. Jika kedua kondisi tersebut tidak dipenuhi, bot akan bergerak kearah player terdekat.



Gambar 3.4 State non-combat (1)

Jika tidak ada player musuh terdeteksi dalam radius arena, maka bot akan masuk ke dalam state *non-combat*. Bot akan tetap mendeteksi torpedo yang mengarah ke bot dan musuh di sekitar bot. Jika ada musuh yang mendekat, bot bisa masuk ke state *combat* kapan saja dan bot bisa masuk ke state torpedo maneuver jika bot mendeteksi ada torpedo yang mengarah ke bot. Jika kedua kondisi itu tidak terpenuhi, bot akan mendeteksi border dan gas cloud pada map dan akan menghindarinya.



Gambar 3.5 State *non-combat* (2)

Jika bot berada pada border dan terhimpit oleh *gas cloud*, bot akan mencoba *teleport* ke tengah map jika bisa. Jika semua kondisi tersebut tidak dipenuhi, bot akan mencari makanan terdekat, baik itu makanan biasa ataupun *superfood*.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Pseudocode

##### 4.1.1 Prosedur computeNextPlayerAction(PlayerAction playerAction)

```
Procedure computeNextPlayerAction(input : PlayerAction playerAction)
    If gameState.getWorld().getCurrentTick() != null then
        playerAction.action ← FORWARD
        playerAction.heading ← getHeadingBetween(worldCenter)
        If(not gameState.getGameObjects().isEmpty()) then
            foodList ← [for each getGameObjectType() = FOOD, sorted by
            getDistanceBetween(bot,gameObject) ]
            nearestPlayer ← [for each playerGameObject and id != bot.id, sorted by getDistanceBetween(bot,gameObject) ]
            nearestGasCloud ← [for each getGameObjectType() =
            GAS_CLOUD, sorted by getDistanceBetween(bot,gameObject) ]
            nearestAsteroid ← [for each getGameObjectType() =
            ASTEROID_FIELD, sorted by
            getDistanceBetween(bot,gameObject) ]
            superFoodList ← [for each getGameObjectType() = SUPER_FOOD,
            sorted by getDistanceBetween(bot,gameObject) ]
            nearestTorpedos ← [for each getGameObjectType() =
            TORPEDO_SALVO and abs(getHeadingBetween(item) -
            item.currentHeading + 180) + 360 mod 360 < 60 , sorted by
            getDistanceBetween(bot,gameObject) ]
            supernova_list ← [for each getGameObjectType() =
            SUPERNOVA_PICKUP, sorted by
            getDistanceBetween(bot,gameObject) ]
            a ← 5
            I traversal [0..nearestPlayer.size] do
                If (a<nearestPlayer[0].getSize()) then
                    a ← nearestPlayer[0].getSize()
                    biggestPlayer ← nearestPlayer[i]
                scanMusuh ← nearestPlayer[0]
                terkepung ← terkepung(nearestPlayer,bot,scanMusuh)
                If(getDistanceBetween(bot,scanMusuh) <7*bot.getSize() +
                sccanMusuh.getSize()) then
                    If (alrdFire and currentTick > getTime) then
                        if (bot.getSize()>teleTarget.getSize()) then
                            playerAction.action ← TELEPORT
                            alrdFire ← false
                            teleTarget ← NIL;
                            fireTele ← true
                        Else
                            alrdFire ← false
                            fireTele ← true
                            teleTarget ← NIL
                    else if (getCurrentTick() > evadeTick + 5 and evade) then
                        playerAction.action ← STOPAFTERBURNER
                        Evade ← false
                    Else if (alrdFireSupernova and detonatetick =
                    gameState.getWorld().getCurrentTick()) then
                        playerAction.action ← DETONATESUPERNOVA
```

```

        alrdFireSupernova ← false
    Else if (bot.superNovaAvailable = 1) then
        playerAction.heading ←
        getHeadingBetween(biggestPlayer)
        playerAction.action ← FIRESUPERNOVA
        Detonatetick ← getCurrentTick() +
        getDistanceBetween(bot,biggestPlayer) / 20
        alrdFireSupernova ← true
    Else if (super_nova.size() != 0 and
    getDistanceBetween(supernova_list[0],bot)<50) then
        playerAction.heading ←
        getHeadingBetween(supernova_list[0])
        playerAction.action ← FORWARD
    else if (getDistanceBetween(bot,scanMusuh) <=
    7*bot.getSize() + scanMusuh.getSize() and scanMusuh != NIL)
    then
        If (terkepung) then
            If (bot.fireTeleport>0) then
                playerAction.heading ←
                getHeadingBetween(worldCenter)
                playerAction.action = FIRETELEPORT
                alrdFire ← true
                getTime =
                getDistanceBetween(bot,worldCenter)/20 +
                getCurrentTick()
        Else
            playerAction.heading ←
            getHeadingBetween(scanMusuh)
            playerAction.action ← FIRETORPEDOES
        else if (not nearestTorpedos.isEmpty()) then
            If (bot.getSize() > 26 and
            getDistanceBetween(scanMusuh,bot) < 65 +
            bot.getSize() and bot.shield <0) then
                if(getCurrentTick - ctick > 20) then
                    Valid ← true
                if(valid) then
                    playerAction.action ←
                    ACTIVATESHIELD
                    ctick ← getCurrentTick()
                    Valid ← false
                else if (bot.getSize() >= 15 and
                bot.getSize<=25 and bot.fireTeleport > 0) then
                    Temp1 ←
                    getHeadingBetween(nearestTorpedos[0] -
                    getHeadingBetween(nearestTorpedos[1]))
                    playerAction.heading ←
                    getHeadingBetween(nearestTorpedos[0]) +
                    MAX(ABS(temp1),ABS(360-temp1))/2
                    playerAction.action ← STARTAFTERBURNER
                    Evadetick ← getCurrentTick()
                    Evade ← true
                else if (nearestTorpedos.size() > 1) then
                    Temp1 =
                    getHeadingBetween(nearestTorpedos[0] -
                    getHeadingBetween(nearestTorpedos[1]))
                    playerAction.heading ←

```

```

        getHeadingBetween(nearestTorpedos[0]) +
        MAX(ABS(temp1), ABS(360-temp1))/2
Else
    playerAction.heading ←
    getHeadingBetween(nearestTorpedos[0]) +
    90
else if(scanMusuh != NIL and scanMusuh.getSize() >
bot.getSize() and bot.getSize()>25) then
    If (bot.torpedoSalvo > 0) then
        playerAction.heading ←
        getHeadingBetween(scanMusuh)
        playerAction.action ← FIRETORPEDOES
    else if (scanMusuh != NIL and
getDistanceBetween(bot,scanMusuh) <
5*bot.getSize() + scanMusuh.getSize()) then
        playerAction.action ← FORWARD
        playerAction.heading ←
        getHeadingBetween(nearestPlayer[0] + 540
mod 360
        If (getHeadingBetween(nearestPlayer[0] >=
270 or getHeadingBetween(nearestPlayer[0]
>= 90 and
getHeadingBetween(nearestPlayer[0]) <180)
then
            tempHeading ←
            getHeadingBetween(nearestPlayer[0])
            + 90 mod 360
        Else
            tempHeading ←
            getHeadingBetween(nearestPlayer[0])
            -90 mod 360
        playerAction.heading = tempHeading
        playerAction.action = FORWARD
    else if
        (getDistanceBetween(nearestGasCloud.get(0),
bot) <
        (bot.getSize()+nearestGasCloud.get(0).getSize()
+60)) then
            if
                (getDistanceBetween(nearestGasCloud[0]),
bot)<=(1+bot.getSize()+nearestGasCloud[0]
.getSize()))then
                    playerAction.heading ←
                    getHeadingBetween(nearestGasCloud[0]
] + 180) mod 360
            Else
                If
                    getHeadingBetween(nearestGasCloud.g
et(0))>=270 ||
                    (getHeadingBetween(nearestGasCloud.
get(0))>=90 &&
                    getHeadingBetween(nearestGasCloud.g
et(0))<=180)) then

    else if
        (getDistanceBetween(nearestAsteroid[0],bot)<30

```

```

+ bot.getSize() + nearestAsteroid[0].getSize())
then
    playerAction.heading =
    playerAction.heading + 90 mod 360
else if(scanMusuh != NIL and scanMusuh.getSize() >
bot.getSize()) then
    If (bot.torpedoSalvo > 0) then
        playerAction.heading ←
        getHeadingBetween(scanMusuh)
        playerAction.action ← FIRETORPEDOES
else if (scanMusuh != NIL and
getDistanceBetween(bot,scanMusuh) <
5*bot.getSize() + scanMusuh.getSize()) then
    playerAction.action ← FORWARD
    playerAction.heading ←
    getHeadingBetween(nearestPlayer[0] + 540
mod 360
    If (getHeadingBetween(nearestPlayer[0] >=
270 or getHeadingBetween(nearestPlayer[0]
>= 90 and
getHeadingBetween(nearestPlayer[0]) <180)
then
    tempHeading ←
    getHeadingBetween(nearestPlayer[0])
    + 90 mod 360
Else
    tempHeading ←
    getHeadingBetween(nearestPlayer[0])
    -90 mod 360
    playerAction.heading = tempHeading
    playerAction.action = FORWARD
else if
(getDistanceBetween(nearestGasCloud.get(0),
bot) <
(bot.getSize() + nearestGasCloud.get(0).getSize()
+60)) then
    playerAction.heading ←
    getHeadingBetween(nearestGasCloud[0] +
90) mod 360
else if
(getDistanceBetween(nearestAsteroid[0],bot)<30
+ bot.getSize() + nearestAsteroid[0].getSize())
then
    playerAction.heading =
    playerAction.heading + 90 mod 360
Else
    idle(nearestTorpedos, nearestGasCloud,
foodList, superFoodList)
else if (scanMusuh != NIL and scanMusuh.getSize() <=
bot.getSize()) then
    If (nearestPlayer[0].getSize() < bot.getSize() - 40
and fireTele ) then
        playerAction.heading ←
        getHeadingBetween(nearestPlayer[0])
        playerAction.action ← FIRETELEPORT
        alrdFire ← true

```

```

        fireTele ← false
        getTime ← (getDistanceBetween(nearestPlayer[0],
        bot)-bot.getSize() - nearestPlayer[0].getSize()
        + 0.3*bot.getSize())/20 + getCurrentTick();
else if (bot.torpedoSalvo > 0) then
    playerAction.heading ←
    getHeadingBetween(scanMusuh)
    playerAction.action ← FIRETORPEDOES
Else
    playerAction.heading ←
    getHeadingBetween(scanMusuh)
    playerAction.action ← FORWARD
Else
    idle(nearestTorpedos, nearestGasCloud, foodList,
    superFoodList, supernova_list)
Else
    idle(nearestTorpedos, nearestGasCloud, foodList,
    superFoodList, supernova_list)
Else
    idle(nearestTorpedos, nearestGasCloud, foodList, superFoodList,
    supernova_list)
Wt ← getCurrentTick()
playerAction ← playerAction

```

#### 4.1.2 Function Terkepung(List of Game Object nearestPlayer, GameObject bot, GameObject scanMusuh)

```

Function terkepung (List of Game Object nearestPlayer, GameObject bot,
GameObject scanMusuh) → boolean
ALGORITMA
If (nearestPlayer.getSize() > 3) then
    If (geDistanceBetween(bot,nearestPlayer[2] <= 2*bot.getSize())
    and
    getHeadingBetween(scanMusuh)-getHeadingBetween(nearestPlayer[1])<=270
    and
    getHeadingBetween(scanMusuh)-getHeadingBetween(nearestPlayer[1])>=90
    and
    getHeadingBetween(nearestPlayer[1])-getHeadingBetween(nearestPlayer[2])<
    =270
    and
    getHeadingBetween(nearestPlayer[1])-getHeadingBetween(nearestPlayer[2])>=
    90
    and getHeadingBetween(scanMusuh) - getHeadingBetween(nearestPlayer[2])
    <=270
    and
    getHeadingBetween(scanMusuh) - getHeadingBetween(nearestPlayer[2]) >=
    90) then
        → true
→ true

```

#### 4.1.3 Prosedur Idle (List of GameObject nearestTorpedos, List of GameObject nearstGasCloud, List of GameObject foodList, List of GameObject superFoodList)

```

Procedure idle (List of GameObject nearestTorpedos, List of GameObject
nearestGasCloud, List of GameObject foodList, List of GameObject superFoodList,
List of supernova_pickup supernova_list)
ALGORITMA
playerAction.action ← FORWARD
If (alrdFire and getCurrentTick > getTime) then
    playerAction.action ← TELEPORT
    alrdFire ← false
else if ((getCurrentTick() > evadetick + 5) and evade) then
    playerAction.action ← STOPAFTERBURNER
    Evade ← false
else if (not nearestTorpedos.isEmpty()) then
    If (getDistanceBetween(nearestTorpedos[0],bot) < 85 + bot.getSize())
    then
        If (bot.getSize()>26) then
            If (getCurrentTick() - ctick > 20) then
                Valid ← true
        If (valid and bot.shield > 0 ) then
            playerAction.action ← ACTIVATESHIELD
            playerAction.heading ← getHeadingBetween(worldCenter)
            ctick ← getCurrentTick()
            valid ← false
        Else
            playerAction.action ← STARTAFTERBURNER
            Evade ← true
            Evadetick ← getCurrentTick()
            playerAction.heading ← getHeadingBetween(nearestTorpedos[0]
            + 450 mod 360)
    Else
        playerAction.heading ← getHeadingBetween(nearestTorpedos[0]) +
        450 mod 360
else if (getDistanceBetween(worldCenter,bot)+2.5*bot.getSize()>getRadius() and
getDistanceBetween(nearestGasCloud[0],bot) < 80 + bot.getSize() +
nearestGasCloud[0].getSize()) then
    If (bot.getSize()>40 and bot.fireTeleport>0) then
        playerAction.heading ← getHeadingBetween(worldCenter)
        playerAction.action ← FIRETELEPORT
        alrdFire ← true
        getTIme ← getDistanceBetween(worldCenter,bot) / 20 +
        getCurrentTick()
    Else
        If (getHeadingBetween(nearestGasCloud[0])>=270 or
getHeadingBetween(nearestGasCloud[0])>=90 and
getHeadingBetween(nearestGasCloud[0])<=180) then
            playerAction.heading ← getHeadingBetween(nearestGasCloud) +
            90 mod 360
        Else
            playerAction.heading ← getHeadingBetween(nearestGasCloud) -
            90 mod 360
else if (getDistanceBetween(worldCenter,bot)+2.5*bot.getSize()>gatRadius())
then
    playerAction.heading ← getHeadingBetween(worldCenter)

```

```

else if (getDistanceBetween(nearestGasCloud[0],bot) < (80 + bot.getSize() +
nearestGasCloud[0].getSize()) then
    If (getDistanceBetween(nearestGasCloud[0],bot) <= (1+bot.getSize() +
nearestGasCloud[0].getSize()) then
        playerAction.heading ← getHeadingBetween(nearestGasCloud[0]) +
180 mod 360
    else
        playerAction.heading ← getHeadingBetween(nearestGasCloud[0] + 90
mod 360
else if (getDistanceBetween(superFoodList[0],bot) <
getDistanceBetween(foodList[0],bot)) then
    playerAction.heading ← getHeadingBetween(superFoodList[0])
Else
    playerAction.heading ← getHeadingBetween(foodList[0])

```

## 4.2 Struktur Data

### 4.2.1 Game Object

Game Object merupakan struktur data kelas yang memiliki beberapa atribut diantaranya seperti berikut

```

UUID id;
Integer size;
Integer speed;
Integer currentHeading;
Position position;
ObjectTypes gameObjectType;
Integer torpedoSalvo;
Integer shield;
Integer afterBurner;
Integer fireTeleport;
Integer teleport;

```

Kelas ini juga memiliki konstruktor yang membutuhkan atribut atribut seperti diatas. *Game Object* terdiri dari 2 jenis yaitu objek yang memiliki 7 parameter dan 11 parameter, jenis *game object* seperti *asteroid*, *gas cloud*, *food*, dan sejenisnya memiliki 7 parameter sedangkan untuk *player* memiliki 11 parameter.

### 4.2.2 Game State

Game State merupakan suatu kelas yang menunjukkan apa saja yang terjadi pada dunia. Pada kelas ini terdapat list dari *game object*, *player*, dan *world*. Kelas ini memiliki konstruktor yang mengandung atribut seperti diatas.

### 4.2.3 Game State DTO

Game State DTO adalah suatu kelas yang menyimpan data *map* dari *world*. Pada kelas ini terdapat fungsi - fungsi yang dapat mengeluarkan *game object* dan juga objek *player*.

#### 4.2.4 Player Action

Pada kelas ini terdapat semua kelas yang menyimpan semua hal yang berhubungan dengan player. Contoh dari penggunaan kelas ini adalah *setter* dan *getter* untuk aksi, id player dan *heading*.

#### 4.2.5 Position

Kelas position adalah kelas yang digunakan untuk menyimpan posisi player. Posisi didefinisikan memiliki 2 atribut yaitu x dan y. Selain itu diimplementasikan pula *getter* dan *setter* untuk posisi.

#### 4.2.6 World

Pada kelas world diimplementasikan beberapa fungsi yang berhubungan dengan radius pada arena, seperti perubahan border pada setiap tick-nya dan juga center poin untuk menentukan titik tengah.

#### 4.2.7 Object Types

Object Types adalah suatu objek enum yang menyimpan semua yang jenis objek yang ada di game. Pada objectTypes terdapat tipe objek sebagai berikut

```
PLAYER(1),  
FOOD(2),  
WORMHOLE(3),  
GAS_CLOUD(4),  
ASTEROID_FIELD(5),  
TORPEDO_SALVO(6),  
SUPER_FOOD(7),  
SUPERNova_PICKUP(8),  
SUPERNova_BOMB(9),  
TELEPORTER(10),  
SHIELD(11);
```

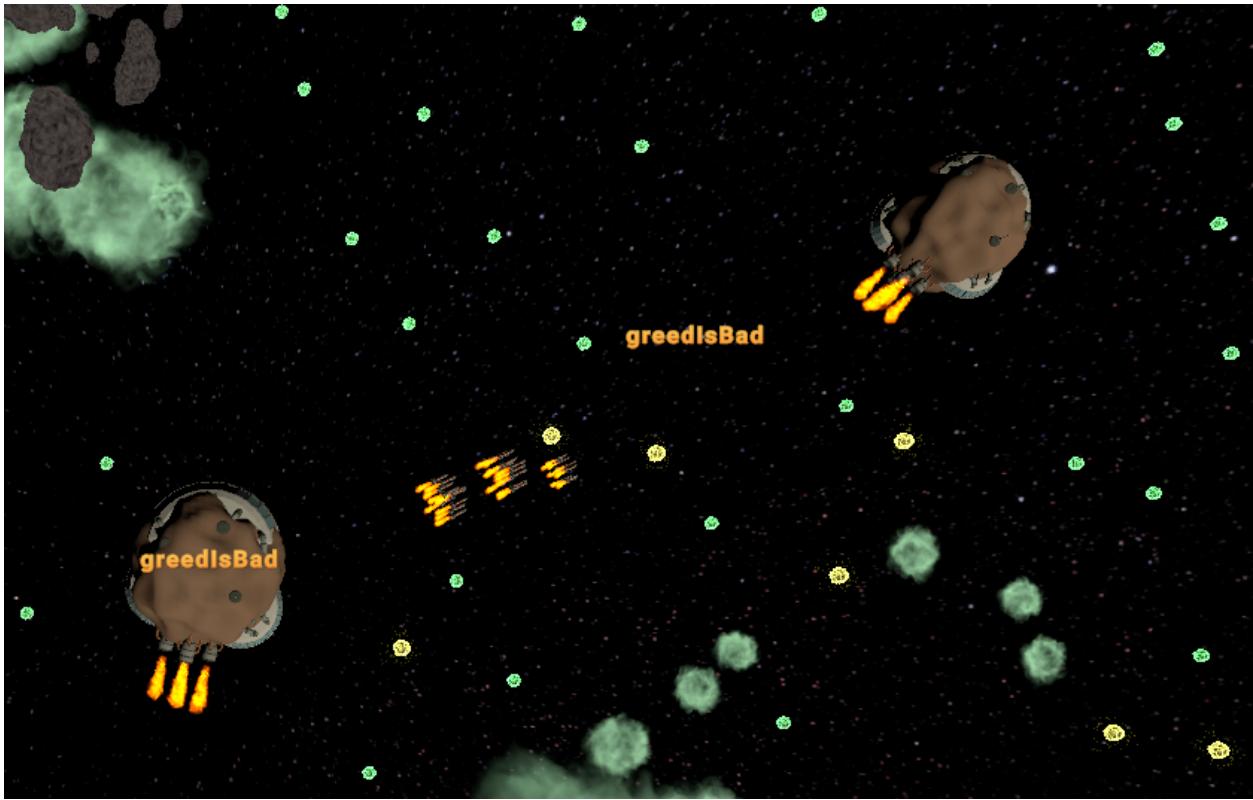
#### 4.2.8 Player Action

*Player Action* adalah suatu objek enum yang menyimpan jenis aksi yang dapat dilakukan oleh *player*. Aksi - aksi yang dapat dilakukan player antara lain

```
FORWARD(1),  
STOP(2),  
STARTAFTERBURNER(3),  
STOPAFTERBURNER(4),  
FIRETORPEDOES(5),  
FIRESUPERNova(6),  
DETONATESUPERNova(7),  
FIRETELEPORT(8),  
TELEPORT(9),  
ACTIVATESHIELD(10);
```

### 4.3 Pengujian

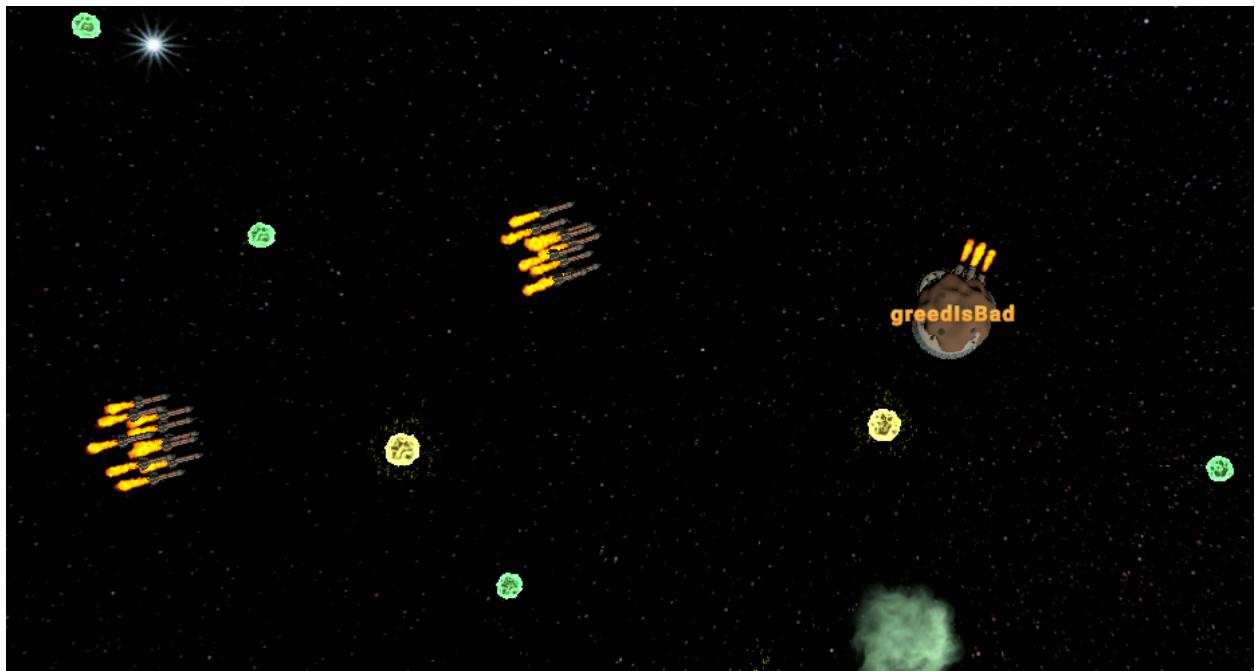
#### 4.3.1 Test Case I



Gambar 4.3.1

Ketika musuh berada di area threshold dan bot memiliki torpedo maka bot akan menembakkan torpedo, pergerakan dari bot sesuai dengan algoritma yang telah dituliskan pada kode berikut

```
else if (bot.torpedoSalvo > 0 && bot.getSize()>26 && scanMusuh != null){  
    System.out.println("fire torpedo");  
    playerAction.heading = getHeadingBetween(scanMusuh);  
    playerAction.action = PlayerActions.FIRETORPEDOES;
```



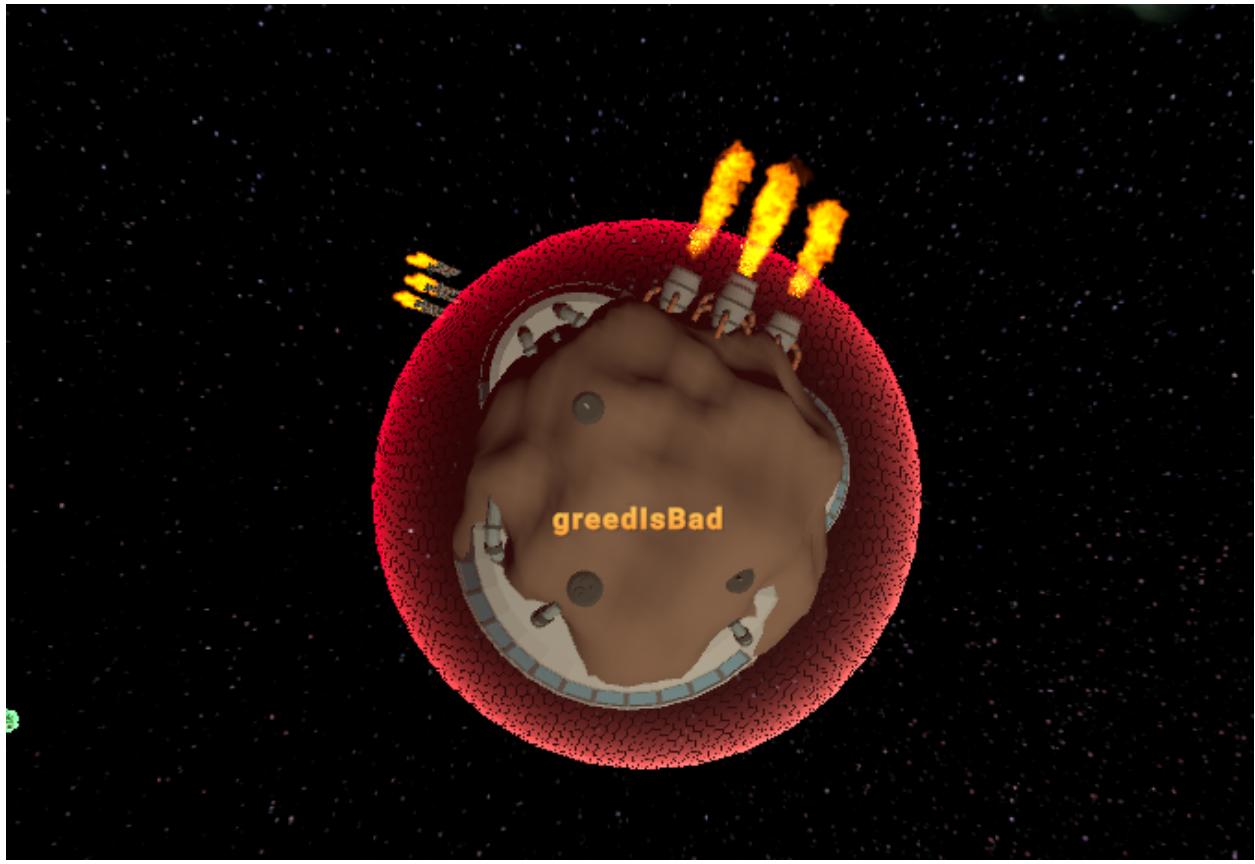
Gambar 4.3.1.1

```
Torpedo Count : 5
Shield Count : 1
Teleport Count : 1
Current Radius : 911
Current Size : 13
Current Tick : 89
```

Gambar 4.3.1.2

Apabila bot ditembak oleh torpedo dan batasan ukuran dari bot tidak memenuhi untuk menyalakan shield ataupun menyalakan afterburner maka bot akan menghindari arah torpedo seperti yang telah dituliskan pada kode berikut

```
else if(nearestTorpedos.size()>1){
    System.out.println("Evasive maneuvers! -Tanpa afterburner");
    int templ = getHeadingBetween(nearestTorpedos.get(0)) -
    getHeadingBetween(nearestTorpedos.get(1));
    playerAction.heading = getHeadingBetween(nearestTorpedos.get(0)) +
    (Math.max(Math.abs(templ),Math.abs(360-templ))/2);
```



Gambar 4.3.1.3

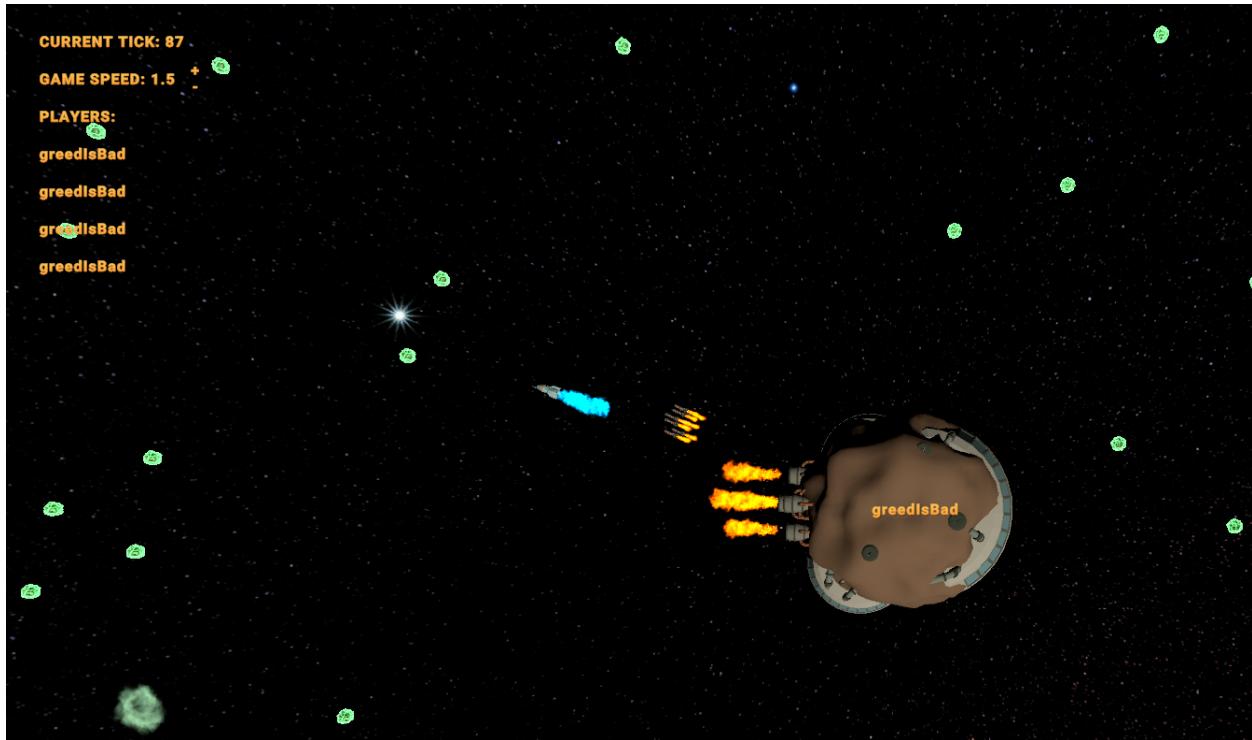
```
Torpedo Count : 1
Shield Count : 1
Teleport Count : 2
Current Radius : 839
Current Size : 47
Current Tick : 161
```

Gambar 4.3.1.4

Apabila bot ditembak dengan torpedo dan ukuran dari bot memenuhi syarat untuk menyalakan shield maka bot akan menyalakan shield. Syarat shield menyala adalah ukuran lebih dari 25.

```
if (bot.getSize() > 25 && getDistanceBetween(nearestTorpedos.get(0), bot) <
65+bot.getSize() && bot.shield > 0)
{
System.out.println("Able to shield");
playerAction.action = PlayerActions.ACTIVATESHIELD;
System.out.println("Shield Engaged!");
ctick = gameState.getWorld().getCurrentTick();
valid = false;
```

#### 4.3.2. Test Case II



Gambar 4.3.2.1

```
Torpedo Count : 5
Shield Count : 1
Teleport Count : 1
Current Radius : 913
Current Size : 26
Current Tick : 87
```

Gambar 4.3.2.2

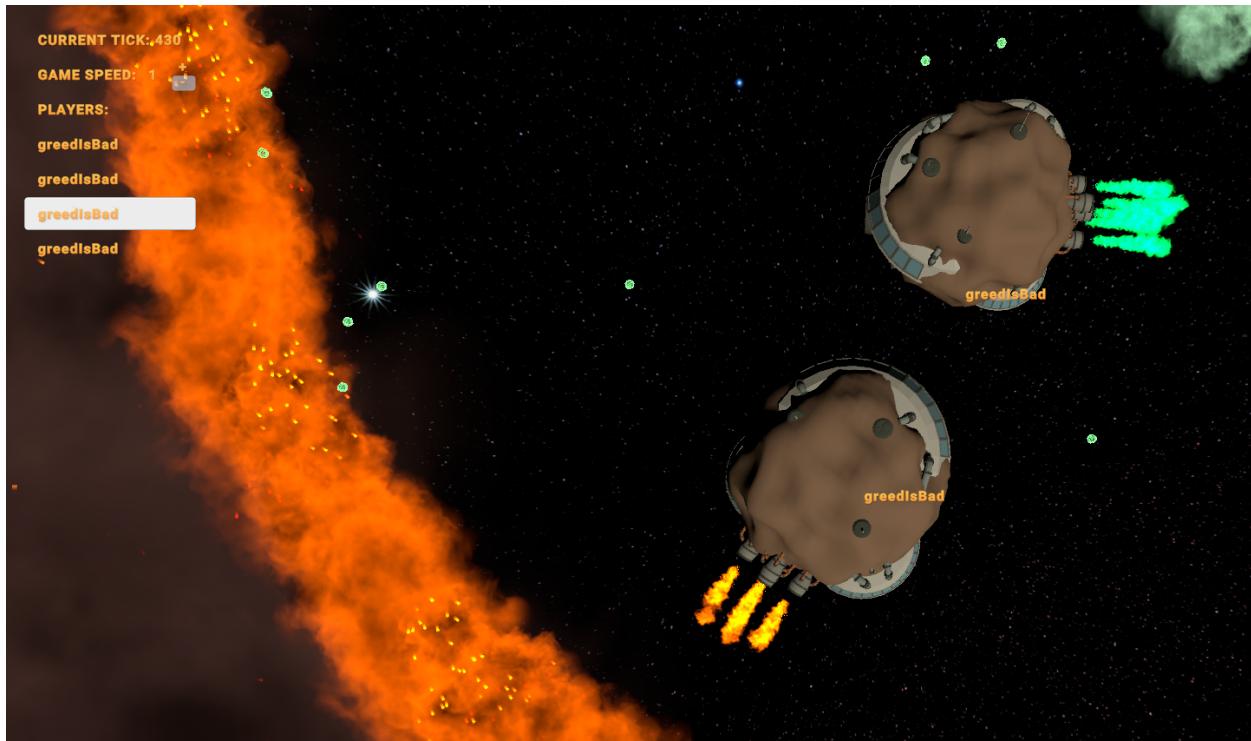
Bot yang memiliki teleport dan memenuhi memenuhi selisih batas ukuran dengan musuh lebih dari 40 maka akan menembakkan teleport sesuai dengan kode berikut.

```
if (nearestPlayer.size() != 0 && nearestPlayer.get(0).getSize() <
bot.getSize()-40 && fireTele)
{
playerAction.heading = getHeadingBetween(nearestPlayer.get(0));
playerAction.action = PlayerActions.FIRETELEPORT;
teleTarget = nearestPlayer.get(0);
alrdFire = true;
fireTele = false;
```

```

getTime = (getDistanceBetween(nearestPlayer.get(0), bot)-bot.getSize() -
nearestPlayer.get(0).getSize() + 20)/20 +
gameState.getWorld().getCurrentTick();
System.out.println("fire tele");
}

```



Gambar 4.3.2.3

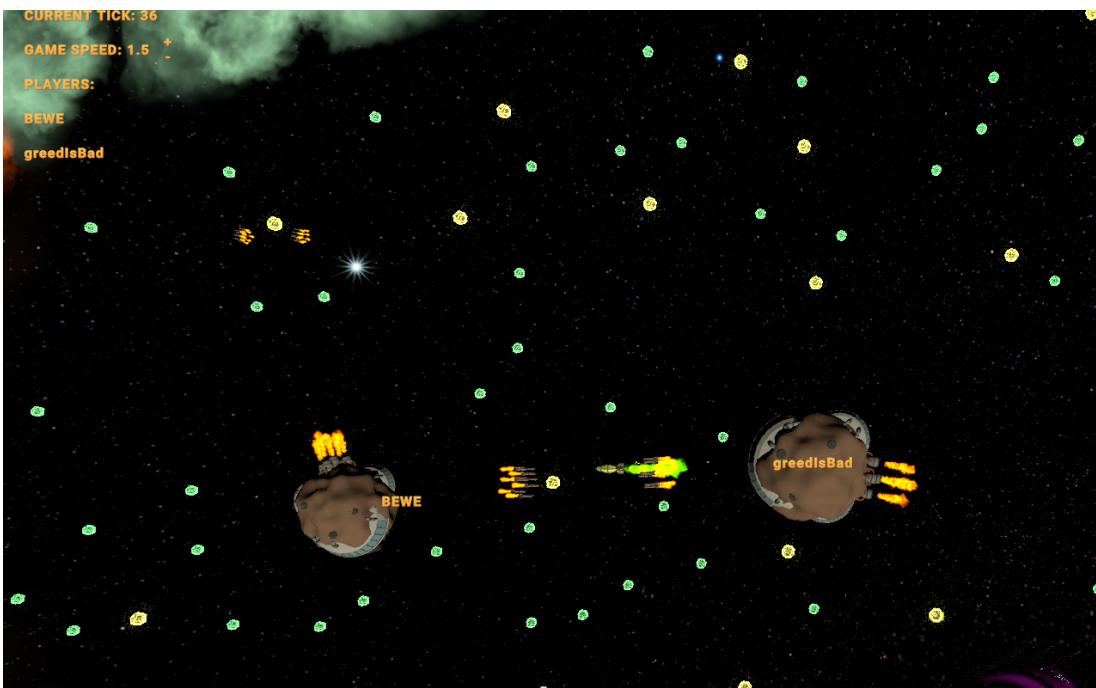
Bot yang tidak memiliki teleport dan memiliki ukuran lebih kecil dari musuh serta jarak yang dekat akan lari dengan menggunakan afterburner.

```

else if (bot.getSize() >=15 && bot.getSize() <= 25 && bot.fireTeleport == 0) {
System.out.println("Evasive maneuvers!");
int temp1 = getHeadingBetween(nearestTorpedos.get(0)) -
getHeadingBetween(nearestTorpedos.get(1));
playerAction.heading = getHeadingBetween(nearestTorpedos.get(0)) +
(Math.max(Math.abs(temp1),Math.abs(360-temp1))/2);
playerAction.action = PlayerActions.STARTAFTERBURNER;
evadetick = gameState.getWorld().getCurrentTick();
evade = true;
}

```

### 4.3.3 Test Case III



Gambar 4.3.3.1



Gambar 4.3.3.2

Apabila supernova muncul pada arena pertandingan dan jarak dari bot dan supernova kurang dari 50 maka bot akan mengambil supernova. Supernova yang diambil akan langsung ditembakkan menuju player dengan ukuran paling besar.

```
else if (alrdFireSupernova && detonatetick ==
gameState.getWorld().getCurrentTick())
{
playerAction.action = PlayerActions.DETONATESUPERNOVA;
alrdFireSupernova = false;
}
else if (bot.superNovaAvailable==1)
{
playerAction.heading = getHeadingBetween(biggestPlayer);
playerAction.action = PlayerActions.FIRESUPERNOVA;
detonatetick = gameState.getWorld().getCurrentTick() +
(int)(getDistanceBetween(biggestPlayer, bot))/20;
alrdFireSupernova = true;
}
else if(supernova_list.size()!=0 &&
getDistanceBetween(supernova_list.get(0),bot)<50)
{
playerAction.heading = getHeadingBetween(supernova_list.get(0));
playerAction.action = PlayerActions.FORWARD;
}
```

## BAB V

### KESIMPULAN DAN SARAN

#### **5.1 Kesimpulan**

Algoritma *Greedy* dapat diterapkan pada pembuatan bot pada game Galaxio. Pada bot yang dibuat penulis dalam game ini, keuntungan terbesar dinilai sebagai keadaan yang membuat bot dapat bertahan hidup. Implementasinya dilakukan dengan menerapkan implementasi *greedy* pada hal-hal yang lebih kecil, contohnya pada saat bot bergerak mendapatkan makanan yang paling dekat pada waktu tertentu, menentukan apakah perlu atau tidak memperhitungkan untuk menghindar atau mengejar musuh jika terdapat musuh di dekat kita, dan memutuskan untuk menggunakan atau tidak peralatan (*teleport*, *torpedo salvo*, dll) di saat tertentu. Implementasi-implementasi tersebut disusun menjadi jalan pemikiran untuk bot dalam suatu automata untuk mendapatkan suatu aksi bot pada setiap kondisi game yang akhirnya memberikan keuntungan yang disebutkan sebelumnya, yaitu bot dapat bertahan hidup.

#### **5.2 Saran**

Implementasi bot ini didesain dengan memanfaatkan determinite state automata sehingga memudahkan implementasi algoritma *greedy* untuk menentukan apa aksi terbaik yang dilakukan pada suatu tick. Dalam kode, implementasi ini berbentuk if - else dan memiliki beberapa persyaratan untuk memasuki setiap aksinya. Penulis menyarankan untuk menyempurnakan kembali syarat yang dilakukan untuk setiap aksi sehingga didapatkan syarat yang terbaik. Penulis juga menyarankan untuk mengimplementasikan algoritma untuk menggunakan supernova karena pada bot ini, penggunaan supernova masih belum diimplementasikan.

#### **5.3 Komentar dan Refleksi**

Tugas ini dapat menjadi sarana untuk berpraktek penggunaan algoritma serta memadukannya dengan programming. Pemahaman dan pembuatan algoritma juga turut dilatih untuk menentukan suatu algoritma. Implementasi kode if - else juga dapat menjadi gambaran mengenai susunan prioritas yang harus dilakukan oleh bot untuk melakukan suatu aksi sehingga setiap tick atau satuan waktunya bot akan melakukan aksi yang terbaik. Melalui tugas ini pula, pemahaman mengenai konsep pemrograman berorientasi objek ditingkatkan kembali sehingga kemampuan tiap individu dalam memecahkan masalah maupun membuat program dapat diasah.

## **DAFTAR PUSTAKA**

Munir, R. (2023, February). *Strategi Algoritma Greedy*. Informatika ITB.

<https://informatika.stei.itb.ac.id/>

Wessels, J. -. (2021). *EntelectChallenge/2021-Galaxio*. GitHub. Retrieved February 17, 2023,

from <https://github.com/EntelectChallenge/2021-Galaxio>

## **LAMPIRAN LINK**

Youtube : <https://youtu.be/CI9oj6J8z4g>

Github : [https://github.com/henryanandsr/Tubes1\\_greedIsBad.git](https://github.com/henryanandsr/Tubes1_greedIsBad.git)