LAPORAN TUGAS BESAR I

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan Mata Kuliah Strategi Algoritma (IF2211)

Disusun oleh:
Kelompok 32
“Shota Lovers 6969”

Anggota:
Muhammad Farrel Danendra Rachim (13521048)
Haidar Hamda (13521105)
Reza Pahlevi Ubaidillah (13521165)

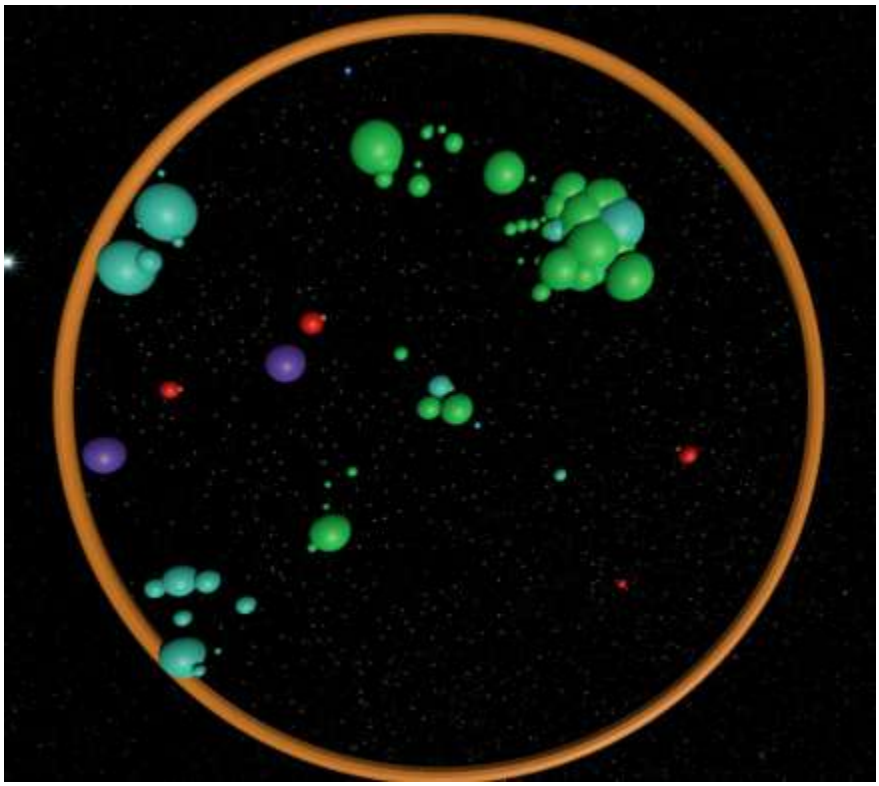
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2022/2023

Daftar Isi

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 disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan 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 menembakannya 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_AFTERBURNER
    d. STOP_AFTERBURNER
    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.

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* adalah algoritma yang digunakan untuk memecahkan persoalan secara langkah per langkah, di mana pada masing-masing langkah algoritma mengambil hasil yang terbaik pada saat itu juga dengan harapan bahwa pemilihan optimum lokal tersebut akan berakhir dengan optimum global. Namun, optimum global belum tentu menjadi solusi terbaik, karena algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada dan terdapat beberapa fungsi seleksi yang berbeda. Sehingga, solusi algoritma *greedy* dapat dianggap sebagai hampiran solusi optimal. Algoritma *greedy* sangat populer dan sederhana dalam memecahkan persoalan optimasi, yaitu persoalan mencari solusi optimal yang terdiri atas maksimasi dan minimasi. Beberapa contoh persoalan yang diselesaikan dengan algoritma *greedy* yaitu persoalan penukaran uang, persoalan *knapsack*, pohon merentang minimum, kode Huffman, dan *Traveling Salesman Problem*.

Terdapat beberapa elemen algoritma *greedy*:

1. Himpunan kandidat (C): Berisi kandidat yang akan dipilih pada setiap langkah, misal simpul/sisi dalam graf, koin, benda, karakter, dsb.
2. Himpunan solusi (S): Berisi kandidat yang sudah dipilih.
3. Fungsi solusi: Menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi (*selection function*): Memilih kandidat berdasarkan strategi *greedy* tertentu yang bersifat heuristik.
5. Fungsi kelayakan (*feasible*): Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi obyektif: Memaksimumkan atau meminimumkan.

Dari elemen-elemen tersebut, algoritma *greedy* melibatkan pencarian sebuah himpunan bagian S dari himpunan kandidat C, dengan syarat S harus memenuhi beberapa kriteria yang ditentukan di mana S menyatakan suatu solusi dan S dioptimisasi oleh fungsi objektif.

Berikut adalah *pseudocode* dari skema umum algoritma *greedy*:

```
function greedy(C: himpunan) -> himpunan_solusi
{Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy}

DEKLARASI
    x: kandidat
    S: himpunan_solusi

ALGORITMA
    S <- {}
    while (not SOLUSI(S)) and (C != {}) do
        x <- SELEKSI(C)
        C <- C - {x}
        if LAYAK(S U {x}) then
            S <- S U {x}
        endif
    endwhile
    {SOLUSI(S) or C = {}}

    if SOLUSI(S) then {solusi sudah lengkap}
        return S
    else
        write('Tidak ada solusi)
    endif
```

2.2. Cara Kerja Program

Untuk mengimplementasikan algoritma greedy pada bot permainan galaxio ini, bot melakukan akan aksi sesuai dengan PlayerAction yang diterima. Pada program ini, keputusan untuk menentukan aksi dipilih berdasarkan PlayerState pada kondisi itu.

Berikut *prerequisite* yang diperlukan sebelum merancang program:

- Java versi 11 atau lebih:
  https://www.oracle.com/java/technologies/downloads/#java8
- NodeJS
  https://www.jetbrains.com/idea/
- .Net Core 3.1
  https://www.jetbrains.com/idea/
- Apache Maven
  https://maven.apache.org/

Pada program, bot melakukan aksinya dengan mengembalikan sebuah *command* yang akan dilakukan di setiap *tick* dalam sebuah permainan, bergantung strategi *greedy* yang dipilih bot untuk mendapatkan hasil optimal. Aksi ini dilakukan di botService.java dan Algorithm.java. *Command* ini nantinya akan diantar ke Main.java untuk diproses ke bot asli di *game*-nya.

Untuk membangun dan menjalankan program ini, diperlukan proses berikut:

1. Clone repository yang terdapat di lampiran
2. Buka terminal (bisa berupa Git Bash) di dalam folder hasil clone (dalam hal ini di folder paling luar)
3. Jalankan perintah mvn clean package
4. Pergi ke folder starter-pack
5. Edit file run.bat di folder starter-pack tersebut jika perlu, lalu run file .bat tersebut.
6. Setelah selesai, klik Galaxio.exe di folder visualizer untuk melihat animasi *game* Galaxio tersebut.
7. Load *Game* dengan file .json terbaru yang seharusnya sudah disediakan di autofill.
8. Jalankan *game*.

BAB 3
Aplikasi Strategi Greedy

3.1. Mapping

Himpunan kandidat:

1. Himpunan bilangan bulat diantara 0 sampai 359 (*heading)*
2. Himpunan *PlayerActions* (seluruh *command* yang dapat digunakan pemain pada game Galaxio)

Himpunan solusi:

1. Himpunan bilangan bulat dan PlayerActions yang terpilih

Fungsi solusi:

Memeriksa apakah bilangan bulat tepat mengarah objek terbaik tanpa memotong rintangan.

Fungsi seleksi:

Memilih *heading* yang tepat berdasarkan keadaan *gameState* saat itu.

Fungsi kelayakan:

Mengecek apabila torpedo, teleport, dan *afterburner* yang digunakan kapal tidak akan membuat ukuran kapal terlalu kecil sehingga rentan tereliminasi.

Fungsi obyektif:

Jika strategi Greedy fokus kepada *Fetching*, maka bot memaksimalkan banyaknya *food* yang dikonsumsi.

3.2. Alternatif Solusi Greedy

Kami mengimplementasikan beberapa alternatif strategi *Greedy* untuk menentukan solusi yang terbaik berdasarkan kondisi sekitar bot pada waktu itu juga. Berikut adalah alternatif-alternatif strategi yang diurutkan berdasarkan prioritas dari yang tertinggi dalam tiap *tick*, antara lain:

a. Strategi *Greedy Escaping*

Menurut kami, bertahan hidup adalah objektif yang paling penting dalam *game* bertipe *battle royale* ini. Sehingga, kami memprioritaskan kapal pemain agar "kabur" dengan *damage* yang minimal dari berbagai macam hal yang dapat mengurangi ukuran kapal kami yang akan berujung ke eliminasi. Terdapat beberapa objek "berbahaya" yang perlu diperhatikan kapal: awan gas, torpedo lawan, dan kapal lawan yang ukurannya lebih besar dari kapal pemain. Kami mengutamakan untuk menghindari torpedo terlebih dahulu akibat *damage* yang disebabkannya.

Untuk melakukan metode *escaping* ini, terdapat tiga buah cara:

- Jika ukuran bot di antara 50 dan 70, serta radius batas *game* lebih dari 30, maka kapal mengubah *heading* sebesar 130 dari objek yang ingin dihindari, menembak teleporter, dan melakukan teleportasi.

- Jika ukuran bot di antara 10 dan 30, ada baiknya tidak menembak teleporter karena akan mengorbankan ukuran kapal, sehingga kapal akan mengubah *heading* sebesar 130 dari objek yang ingin dihindari dan mengaktifkan *afterburner* untuk kabur dari objek tersebut.
- Jika ukuran bot di luar *range-range* tersebut, bot hanya akan mengubah *heading* sebesar 130 dari objek yang ingin dihindari dan maju ke arah tersebut (mengaktifkan *command* FORWARD).

b. Strategi *Greedy Chasing*

Strategi ini membuat kapal memprioritaskan menyerang kapal lawan menggunakan torpedo untuk memperkecil ukuran kapal lawan dan memperbesar ukuran kapal pemain. Torpedo akan ditembakkan kapal pemain jika kapal pemain sudah menghadap kapal lawan. Kami mempertimbangkan jarak antara bot kami dengan bot lawan serta kecepatan torpedo agar torpedo memiliki kemungkinan terbaik untuk mengenai bot lawan. Selain itu, kapal pemain dapat mengejar kapal lawan yang ukurannya lebih kecil dari ukuran kapal pemain agar dapat mengeliminasi kapal tersebut dengan mengkonsumsinya. Kami perlu mempertimbangkan agar bot pemain tidak terlalu dekat ke awan gas atau *wormhole* untuk menghindari pengurangan ukuran kapal pemain. *State* ini dimulai jika ada "mangsa" dan diakhiri jika "mangsa" telah dikenai torpedo atau dimakan, atau "mangsa" di luar jangkauan.

c. Strategi *Greedy Fetching*

Strategi ini membuat kapal mengutamakan mengambil *food* atau *super food* sebanyak-banyaknya di wilayah sekitar kapal tersebut dengan mengubah *heading* sesuai makanan terdekat untuk memperbesar ukuran kapal tersebut. Semakin besar ukuran kapal, semakin besar *advantage* yang dimiliki kapal tersebut (kapal tersebut dapat mengonsumsi kapal lawan yang lebih kecil). *Food* yang dikonsumsi tidak akan muncul kembali, sehingga kapal akan berusaha mengambil *food* sebanyak mungkin jika terdeteksi. Kapal juga akan berusaha untuk tidak menabrak awan gas atau asteroid saat mengambil *food*.

3.3. Analisis Efisiensi

Berikut analisis efisiensi terhadap tiap strategi *greedy* yang sudah dibahas, per satu *tick* permainan.

a. Strategi *Greedy Escaping*

Efisiensi waktu: O(n) dengan n objek "berbahaya" di sekitar bot.

Strategi ini mempertimbangkan apakah kapal perlu "kabur" berdasarkan objek-objek di sekitarnya, seperti awan gas dan torpedo. Algoritma akan memeriksa berbagai objek di sekitarnya dengan iterasi objek terdekat, mencari objek yang dianggap "berbahaya" bagi bot, dan memilih *command* yang cocok (START_AFTERBURNER, FIRE_TELEPORTER, FORWARD) sesuai kondisi *game* tersebut pada waktu tertentu.

b. Strategi *Greedy Chasing*

Efisiensi waktu: O(n) dengan n objek "mangsa" di sekitar bot.

Strategi ini mempertimbangkan apakah kapal perlu mengejar "mangsa" jika kapal lawan terdeteksi. Algoritma akan memeriksa berbagai objek di sekitarnya yang dapat dikejar atau ditembak dengan iterasi pengecekan objek berdasarkan jaraknya, mencari objek yang dapat dijadikan "mangsa" bagi bot, dan memilih *command* yang cocok (FORWARD, FIRE_TORPEDOES) sesuai kondisi kapal lawan.

c. Strategi *Greedy Fetching*

Efisiensi waktu: O(n) dengan n objek *food* di sekitar bot.

Strategi ini.menerapkan pengambilan objek *food* di sekitarnya dengan iterasi tiap objek di sekitar bot tersebut dan memakan *food* yang paling dekat terhadap *bot* tersebut agar penambahan ukuran bot dapat berjalan seefisien dan secepat mungkin.

3.4. Analisis Efektivitas

Berikut analisis efektivitas terhadap tiap strategi *greedy* yang sudah dibahas.

a. Strategi *Greedy Escaping*

Strategi ini efektif apabila:

Kapal menghadapi satu atau dua objek "berbahaya" dalam radius *game* yang luas, sehingga kapal memiliki ruang yang memadai untuk bergerak ke posisi di peta yang relatif lebih aman. Selain itu, cara kapal melakukan *escaping* juga bersifat dinamis, yang berarti metode *escaping* dapat berubah-ubah sesuai ukuran kapal dan radius peta pada saat itu juga.

Strategi ini tidak efektif apabila:

Kapal menghadapi objek "berbahaya" dalam radius *game* yang terbatas, sehingga kapal tidak memiliki ruang yang cukup untuk melakukan sesuatu. Selain itu, juga terdapat potensi terjadi konflik antara aksi *escaping* yang akan dilakukan dalam tiap *tick* karena kondisi-kondisi terjadinya aksi tersebut akan bertabrakan seiring mengecilnya radius peta. Hal ini akan membuat gerakan kapal tidak fokus.

b. Strategi *Greedy Chasing*

Strategi ini efektif apabila:

Kapal mendeteksi kapal lawan yang ukurannya lebih kecil dengan selisih yang lumayan besar dibandingkan ukuran kapal pemain. Selain itu, strategi ini cocok digunakan saat kapal lawan berada pada garis tembakan kapal pemain, sehingga torpedo dapat digunakan untuk mengurangi ukuran kapal lawan tersebut, atau kapal pemain dapat bergerak dengan cepat untuk mengkonsumsi kapal lawan jika *margin* ukuran antar kapal lumayan besar.

Strategi ini tidak efektif apabila:

Kapal mendeteksi kapal lawan yang ukurannya lebih besar atau sama dengan kapal pemain. Lagi-lagi, objektif utama dari *game* ini menurut kami yaitu bertahan hidup, sehingga strategi ini tidak menganjurkan kapal untuk mengejar kapal lawan yang lebih besar. Walaupun begitu, kapal tetap dapat menembak kapal lawan agar ukuran kapal lawan mengecil.

c. Strategi *Greedy Fetching*

Strategi ini efektif apabila:

Kapal dikelilingi oleh banyak *food* di sekitarnya, sehingga kapal dapat mengkonsumsi *food* sebanyak mungkin dalam suatu ruang terbatas secara efektif dan memperbesar ukuran kapal secara lebih cepat.

Strategi ini tidak efektif apabila:

Tidak banyak *food* yang tersebar di sekitar kapal, sehingga kapal tidak mampu menambah ukurannya dengan cepat.

3.5. Strategi Greedy yang Dipilih

Pada program ini, kami memanfaatkan ketiga alternatif *greedy* yang telah dibahas di subbab-subbab sebelumnya. Namun, lagi-lagi, kami memiliki prioritas terhadap strategi apa saja yang akan diterapkan terlebih dahulu. Kami mengutamakan strategi *escaping* karena kami berpendapat bahwa keselamatan kapal kami adalah yang paling penting dalam *game battle royale* seperti Galaxio. Jika dihadapkan pada suatu bahaya oleh objek seperti torpedo, algoritma akan berusaha untuk mencari jalan keluar agar *damage* yang diperoleh sekecil mungkin, daripada membalas balik bahaya tersebut yang akan berisiko besar bagi kapal kami.

Setelah bahaya hilang, barulah kapal kami mulai menyerang kapal lawan jika kapal kami memiliki keuntungan yang relatif besar terhadap kapal tersebut (*chasing*). Namun, jika kapal kami dalam posisi yang menyusahkan (seperti ukuran yang lebih kecil dari kapal lawan) dan tidak ada bahaya pada *tick* tertentu, kapal akan berusaha melakukan *feeding* terhadap *food* sebanyak mungkin untuk memperbesar ukuran kapal secepat mungkin. Dengan urutan prioritas seperti ini, kami dapat mengoptimalkan kinerja bot dalam bertahan hidup selama mungkin berdasarkan *rescource* dan rintangan yang ada.

BAB 4
Implementasi dan Pengujian

4.1. Implementasi Algoritma Greedy

Berikut adalah implementasi dari algoritma *greedy* yang digunakan. Algoritma yang diperlihatkan hanya mencakup bagian-bagian penting program saja dan tidak sama seutuhnya seperti kode program pada source code.

```
function selectTarget (ObjectTypes objectType) -> GameObject
{
    mengembalikan GameObject terbaik yang bisa dituju oleh bot untuk pengumpulan FOOD/SUPERFOOD
    dilihat dari prioritas SUPERFOOD>FOOD dan apakah untuk mencapai FOOD/SUPERFOOD tersebut
    harus memotong GAS_CLOUD/ASTEROID_FIELD
}

DEKLARASI
nearestfood: GameObject
nearestfoodnotcollidingGas,nearestFoodNotCollidingAsteroid: list of GameObject
WORLD: list of GameObject


ALGORITMA
nearestfood<-getObjectsByDistance(bot,objectType,gameState,false)[0]
{
    fungsi getObjectsByDistance mengembalikan list of GameObject dengan elemen list
    yang memuliki objectType yang sama dengan parameter secara terurut berdasarkan jarak
    antara bot dan object
}
WORLD<-gameState.getGameObjects()
if(cekInside(WORLD,ObjectTypes.GAS_CLOUD)) then {
    {fungsi cekInside akan menerima parameter list of GameObject dan ObjectTypes.
    mengembalikan true jika object dengan ObjectTypes yang sama dengan parameter
    terdapat di dalam parameter list.}
    nearestfoodnotcollidingGas <-notCollidingObjects(objectType,
    getObjectsByDistance(bot,ObjectTypes.GAS_CLOUD,gameState,false))
    if(cekInside(gameState.getGameObjects(), ObjectTypes.ASTEROID_FIELD)) then
        nearestFoodNotCollidingAsteroid = NotCollidingObjects(objectTypes,
        Algorithm.getObjectsByDistance(bot,ObjectTypes.ASTEROID_FIELD,gameState,false))
        if (nearestFoodNotCollidingAsteroid != null) then
            if (nearestFoodNotCollidingGas != null) then
                l1 = {
                    nearestFood;
                    nearestFoodNotCollidingAsteroid[0];
                    nearestFoodNotCollidingGas[0];
                }
                -> getBestWeight(l1);
                {
```

```
                {
                    getBestWeight mengembalikan object yang terbaik
                    untuk dijadikan terget
                }
            else
                if(checkCollision(bot, nearestFood,
                getObjectsByDistance(bot,ObjectTypes.ASTEROID_FIELD,
                gameState,false))) then
                    {
                        checkCollision mengembalikan true jika suatu objek
                        akan memotong suatu objek untuk mencapai objek
                        target
                    }
                    if (getDistanceBetween(bot, nearestFood) * 1.5 <
                    getDistanceBetween(bot, nearestFoodNotCollidingAsteroid[0]))  then
                        -> nearestFoodNotCollidingAsteroid[0]
                    else
                        -> nearestFood

                else
                    -> nearestFood

         else
             if (nearestFoodNotCollidingGas != null) then
                 if (checkCollision(bot, nearestFood,
                 getObjectsByDistance(bot,ObjectTypes.GAS_CLOUD,gameState,false))) then
                     if (getDistanceBetween(bot, nearestFood) * 2 < getDistanceBetween(bot, nearestFoodNotCollidingGas[0])) then
                         -> nearestFood
                     else
                         -> nearestFoodNotCollidingGas[0]
                 else
                     -> nearestFood
             else
                 -> nearestFood

     else
         if (nearestFoodNotCollidingGas != null) then
             if (checkCollision(bot, nearestFood, getObjectsByDistance(bot,ObjectTypes.GAS_CLOUD,gameState,false))) then
                 if (getDistanceBetween(bot, nearestFood) * 2 < getDistanceBetween(bot, nearestFoodNotCollidingGas[0])) then
                     -> nearestFood
```

```
               else
                   -> nearestFoodNotCollidingGas[0];


           else {-> nearestFood}
         else
            -> nearestFood

      else if (cekInside(gameState.getGameObjects(), ObjectTypes.ASTEROID_FIELD)) then
         nearestFoodNotCollidingAsteroid = NotCollidingObjects(objectTypes, getObjectsByDistance(bot,ObjectTypes.ASTEROID_FIELD,gameState,false))
         if (nearestFoodNotCollidingAsteroid != null) then
             if (checkCollision(bot, nearestFood, getObjectsByDistance(bot,ObjectTypes.ASTEROID_FIELD,gameState,false))) then
                 if (getDistanceBetween(bot, nearestFood) * 1.5 < getDistanceBetween(bot, nearestFoodNotCollidingAsteroid[0])) then
                     -> nearestFoodNotCollidingAsteroid[0];
                  else
                     -> nearestFood;

              else
                 -> nearestFood;

         else
            -> nearestFood;

    else
      ->nearestFood;
```

```
procedure getPrey(){
        Melakukan update pada variabel statik class Chase yaitu aimPrey dan chasePrey, berupa mangsa untuk ditembak dan manga unntuk dikejar
secara berturut-turut.
}

DEKLARASI
aimPrey, chasePrey: GameObject

ALGORITMA
aimPrey <- findShootable()
if(aimPrey != null) then
        print("Shooting")
chasePrey <- findChaseable()
if(chasePrey != null) then
        print("Chasing")
```

```
procedure needAfterBurner(){
        Melakukan update pada variabel static class starting dan stopping, untuk menentukan apakah afterburner diperlukan.
}

DEKLARASI
time: double
starting, isBoosting, stopping: boolean
MAX_CHASE_TIME: double

ALGORITMA
MAX_CHASE_TIME <- 5.0
time <- Algorithm.calcCollisionTime(Chase.bot, Chase.chasePrey)
if(MAX_CHASE_TIME < time && time <= MAX_CHASE_TIME*2) then
        starting <- true
else
        if(isBoosting) then
                stopping <- true


function findChaseable() -> GameObject
{
       Mencari player yang dapat menjadi incaran (menjadi mangsa) untuk dikejar
}

DEKLARASI
objListByBorderDistance, gasClouds, wormholes: List of GameObject
bot: GameObject
MAX_CHASE_TIME: double

ALGORITMA
MAX_CHASE_TIME <- 5.0
objListByBorderDistancce <- Algorithm.getObjectsByBorderDistance(bot, ObjectTypes.PLAYER, Chase.gameState)
objListByBorderDistance.remove(Chase.bot)

gasClouds <- Algorithm.getObjectsByDistance(bot, ObjectTypes.GAS_CLOUD, gameState, false)
wormholes <- Algorithm.getObjectsByDistance(bot, ObjectTypes.WORMHOLE, gameState, false)

for(bot of objListByBorderDistance) do
       if((Algorithm.calcCollisionTime(Chase.bot, bot) <- MAX_CHASE_TIME or objListByBorderDistance.size()-1) and
              bot.size*2 <= Chase.bot.size) do
              if(not Algorithm.checkCollision(Chase.bot, bot, gasClouds) and not Algorithm.checkCollision(Chase.bot, bot, wormholes))  do
                     -> bot
-> null


function findShootable() -> GameObject
{
        Mencari player yang dapat menjadi incaran (menjadi mangsa) untuk ditembak
}

DEKLARASI
objListBySize, gasClouds, asteroids, wormholes: List of GameObject
bot: GameObject
distance, torpedoTravelTime: double

ALGORITMA
objListBySize <- Algorithm.getObjectBySize(ObjectTypes.PLAYER, gameState)
{Mendapatkan objek sesuai enum ObjectTypes, terurut berdasarkan ukuran objek}
objListBySize.remove(Chase.bot)
{Menghapus elemen dari dalam list}

gasClouds <- Algorithm.getObjectsByDistance(bot, ObjectTypes.GAS_CLOUD, gameState, false)
asteroids <- Algorithm.getObjectsByDistance(bot, ObjectTypes.ASTEROID_FIELD, gameState, false)
wormholes <- Algorithm.getObjectsByDistance(bot, ObjectTypes.WORMHOLE, gameState, false)
{Mendapatkan objek sesuai enum ObjectTypes, terurut berdasarkan jarak dari terkecil}

for(bot of objListBySize) do
        distance <- Algorithm.getDistanceBetween(Chase.bot, bot)
        {Mencari jarak antara dua GameObject}
        torpedoTravelTime <- Algorithm.toTime(TORPEDO_SPEED, distance)
        {Menghitung waktu tempuh}
        if(Algorithm.calculateCenterToBorderTime(bot, true) >= torpedoTravelTime) do
                if(not Algorithm.checkCollision(Chase.bot, bot, gasClouds) and
                not Algorithm.checkCollision(Chase.bot, bot, asteroids) and
                not Algorithm.checkCollision(Chase.bot, bot, wormholes))  do
                {Mengecek apakah ada suatu objek yang termasuk gasClouds, asteroids, atau wormholes di antara dua objek}
                        -> bot
-> null
```

```
procedure escape(input/output avoidedObjects:GameObject, input/output playerAction: PlayerActions,
input/output heading: integer, input bot:GameObject)
{
    mengubah heading dan player action berdasarkan ancaman yang ada
}
DEKLARASI
torpedo,gasCloud,other_ship:GameObject


ALGORITMAl
torpedo <- detectTorpedo();
{fungsi detectTorpedo mengembalikan bot jika terdapat torpedo yang mengarah ke bot}
gasCloud <- detectGasCloud()
{fungsi detectGasCloud mengembalikan bot jika jarak antara bot dan gasCloud dekat}
if (torpedo != null or gasCloud != null) then
    // prioritas torpedo terlebih dahulu
    if (gasCloud != null) then
        avoidedObjects <- gasCloud
    if (torpedo != null) then
        avoidedObjects <- torpedo
    if (getSize(bot) > 10 and getSize(bot) < 30) then
        {fungsi getSize mengembalikan size dari object parameter}
        heading <- getHeadingBetween(bot, avoidedObjects) + 130
        playerAction <- PlayerActions.START_AFTERBURNER
    else if (getRadius(bot) > 30 and getSize(bot) > 50 and getSize(bot) < 70) then
        heading <- getHeadingBetween(bot, avoidedObjects) + 130
        {fungsi getHeadingBetween mengembalikan heading dari object1 untuk menuju object2}
        playerAction <- PlayerActions.FIRETELEPORT
        playerAction <- PlayerActions.TELEPORT
    else
        heading <- getHeadingBetween(bot, avoidedObjects) + 130
        playerAction <- PlayerActions.FORWARD

else
    other_ship <- detectShip()
    if (other_ship != null) then
            avoidedObjects <- other_ship
        if (getSize(bot) > 10 and getSize(bot) < 30) then
            heading <- getHeadingBetween(bot, avoidedObjects) + 130
            playerAction <- PlayerActions.START_AFTERBURNER
        else if (getRadius(bot) > 30 and getSize(bot) > 50 && getSize(bot) < 70) then
            heading <- getHeadingBetween(bot, avoidedObjects) + 130
            playerAction <- PlayerActions.FIRETELEPORT
            playerAction <- PlayerActions.TELEPORT
        else
            heading <- getHeadingBetween(bot, avoidedObjects) + 130
            playerAction <- PlayerActions.FORWARD
```

4.2. Penjelasan Struktur Data

a. Main.java
   Main.java merupakan program utama yang akan dijalankan. Kelas main akan meminta data *command* pada kelas BotService untuk menjalankan bot.
b. Package Enums

Struktur Data dan Kelas	Penjelasan Singkat
ObjectTypes.java	Deklarasi tipe-tipe objek dalam *game*, termasuk pusat *game*, makanan, awan gas, dan lain-lain.
PlayerActions.java	Deklarasi nama-nama *command* yang dapat digunakan pemain, seperti FORWARD, STOP, FIRE_TORPEDOES, dan lain-lain.
PlayerState.java	Deklarasi state yang terjadi pada bot pemain, yaitu FETCHING, CHASING, dan ESCAPING. Enumerasi ini akan berguna untuk memilih strategi *Greedy* yang tepat.

c. Package Models

Struktur Data dan Kelas	Penjelasan Singkat
BotService.java	Kelas yang memutuskan segala aksi bot berikutnya. Elemen *private* dari kelas ini mencakup bot, aksi pemain, gameState, target berupa objek lain, dan apakah bot mengarah ke batas peta. Dalam kelas ini, ditentukan apa gameState berikutnya: *fetching, escaping,* atau *chasing*. Terdapat berbagai fungsi untuk menghitung jarak antara bot dengan objek lain, mendeteksi objek terdekat, dan lain-lain.
Chase.java	Kelas yang akan mengimplementasikan strategi *Greedy chasing.*
Fetch.java	Kelas yang akan mengimplementasikan strategi *Greedy fetching.*
Escape.java	Kelas yang akan mengimplementasikan strategi *Greedy escaping.*

d. Package Services

Struktur Data dan Kelas	Penjelasan Singkat
Algorithm.java	Kelas ini berisi berbagai metode yang berguna mendeteksi objek-objek sekitar bot pemain seperti torpedo, serta mengkalkulasikan heading, jarak, dan waktu tabrakan antara dua buah objek untuk melancarkan

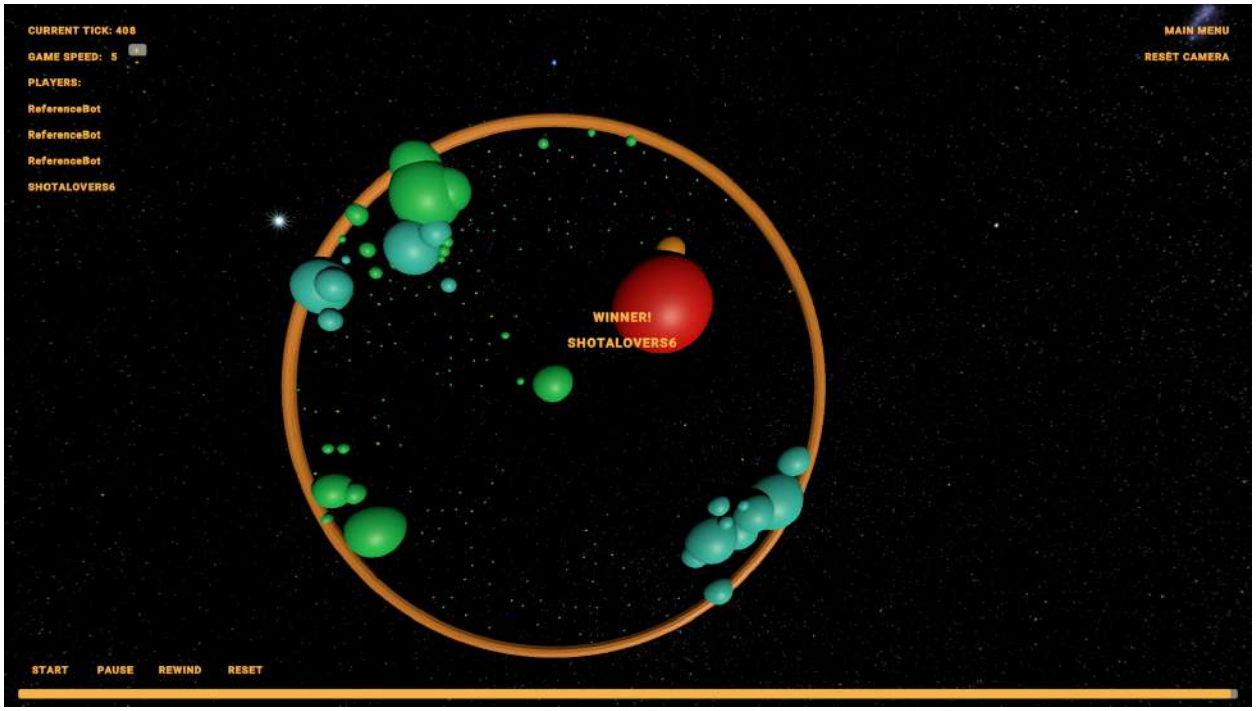
	implementasi strategi *greedy* tertentu.
GameObject.java	Kelas yang akan merepresentasikan objek pada game. Kelas ini memiliki: - UUID id: ID sebuah objek - int size: Ukuran sebuah objek - int speed: Kecepatan sebuah objek, jika bergerak - int currentHeading: *Heading* terkini objek tersebut - Position position: Posisi terkini objek tersebut dalam (x, y) - int radius: Radius yang dimiliki objek tersebut (misal bot memiliki radius awal 10) - gameObjectType gameObjectType: Nama tipe objek
GameState.java	Kelas yang akan merepresentasikan state game pada waktu tertentu. Kelas ini memiliki: - World world: Data seluruh peta berisi titik tengah, radius, dan *tick* terkini - List<GameObject> gameObjects: List yang berisi semua objek dalam *game* tersebut - List<GameObject> playerGameObjects: List yang berisi semua objek yang dimiliki pemain
GameStateDto.java	Kelas yang akan merepresentasikan state game dengan pemetaan oleh map pada waktu tertentu. Kelas ini memiliki: - World world: Data seluruh peta berisi titik tengah, radius, dan tick terkini - Map<String, List<Integer>> gameObjects: Map yang memetakan integer dengan gameObject - Map<String, List<Integer>> playerObjects: Map yang memetakan integer dengan objek pemain
PlayerAction.java	Kelas yang akan merepresentasikan aksi pemain. Kelas ini memiliki: - UUID playerId: ID pemain - PlayerActions action: *Command* yang digunakan pemain - int heading: *Heading* yang dimiliki pemain pada saat itu
Position.java	Kelas yang akan merepresentasikan posisi dalam bidang Kartesius. Kelas ini memiliki: - int x: Absis - int y: Ordinat

World.java	Kelas yang akan merepresentasikan data yang penting dalam berjalannya *game*. Kelas ini memiliki: - Position centerPoint: Posisi titik tengah pada peta - int radius: Radius batas permainan - int currentTick: Menandai *tick* tertentu

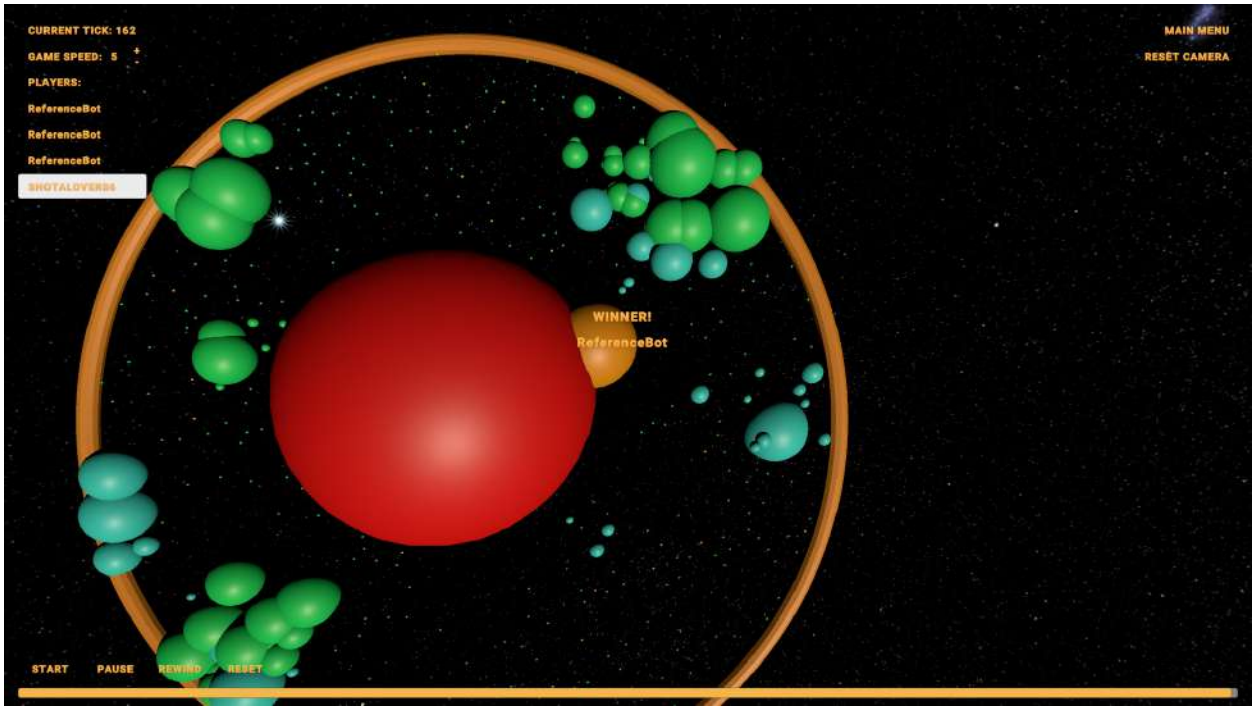
4.3. Analisis Desain Solusi Algoritma Greedy

Pengujian bot dilakukan dengan melawankan bot kami dengan bot ReferenceBot. Game galaxio ini memiliki banyak faktor random. Sehingga perlu dilakukan banyak pengujian untuk menyimpulkan hasil. Kammi memutuskan untuk melakukan pengujian sebanyak 10 kali. Berikut hasil akhir 10 match tersebut:

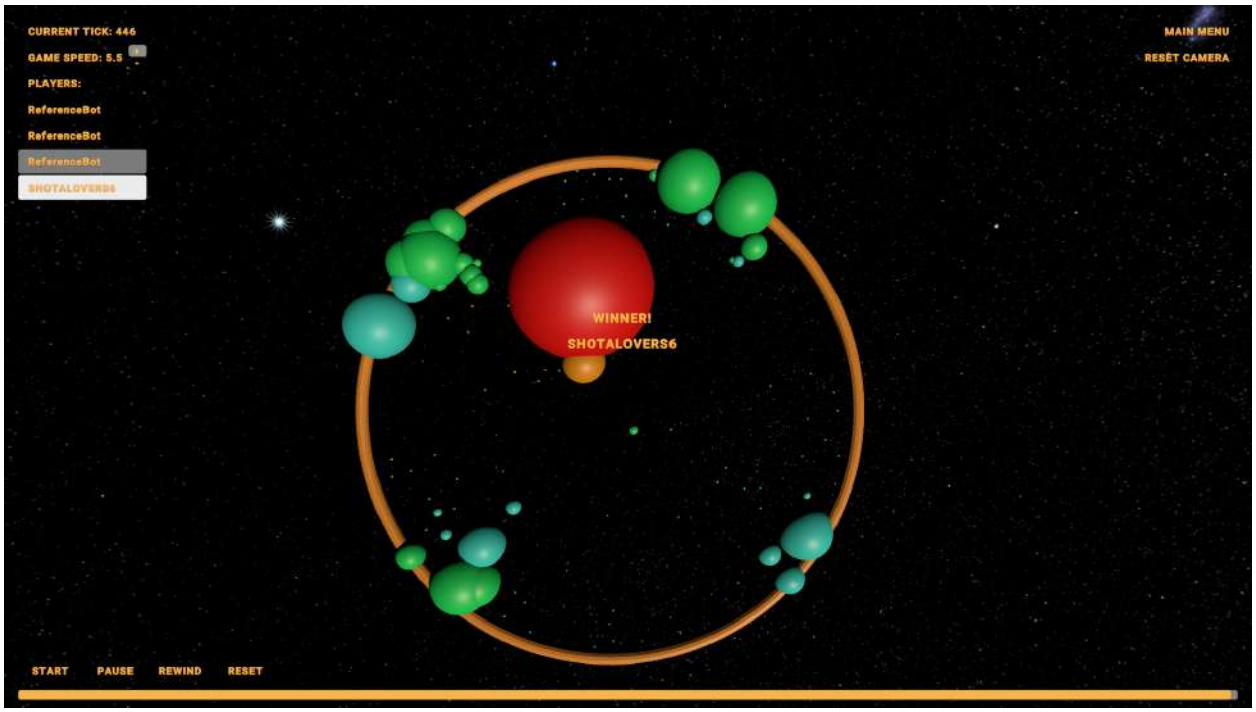
- Pengujian 1:
  Optimal: Kapal mampu menghindari ukuran ReferenceBot yang lebih besar dan menembaknya secara akurat agar lebih kecil
  Tidak optimal: -



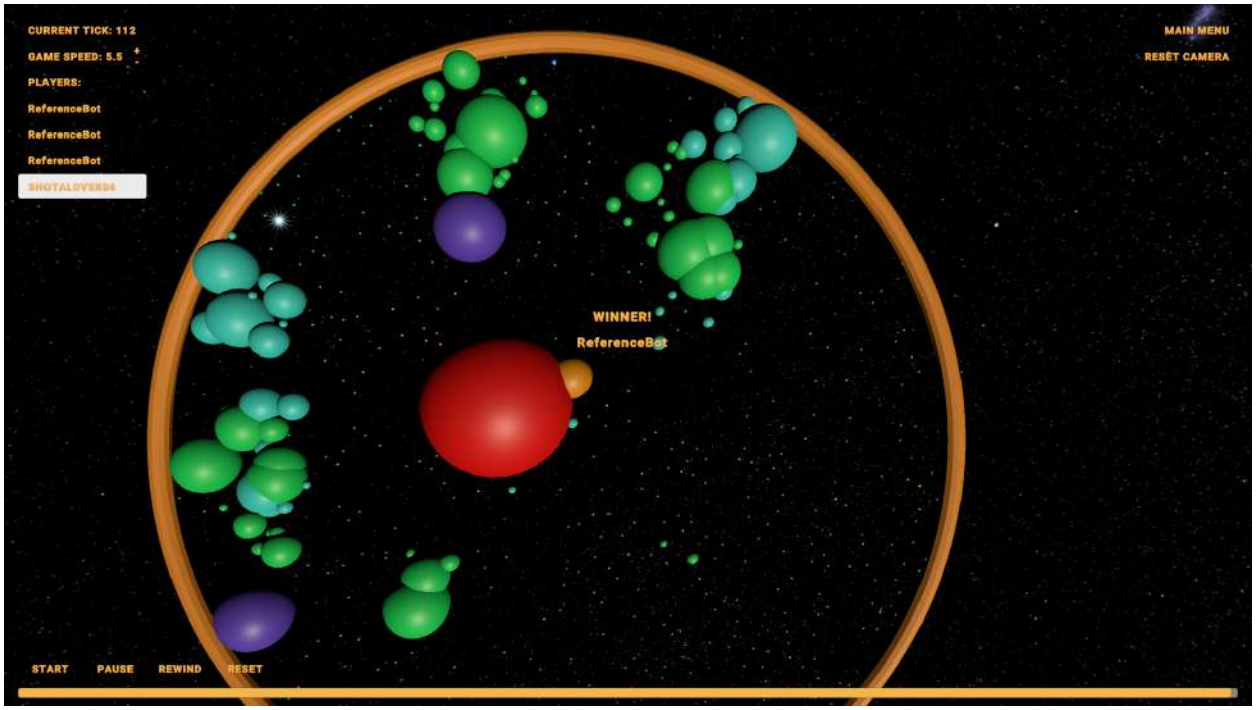
- Pengujian 2:
  Optimal: Kapal mampu menembak ReferenceBot secara akurat agar lebih kecil
  Tidak optimal: Kapal langsung menabrak ke kapal dengan ukuran yang lebih besar



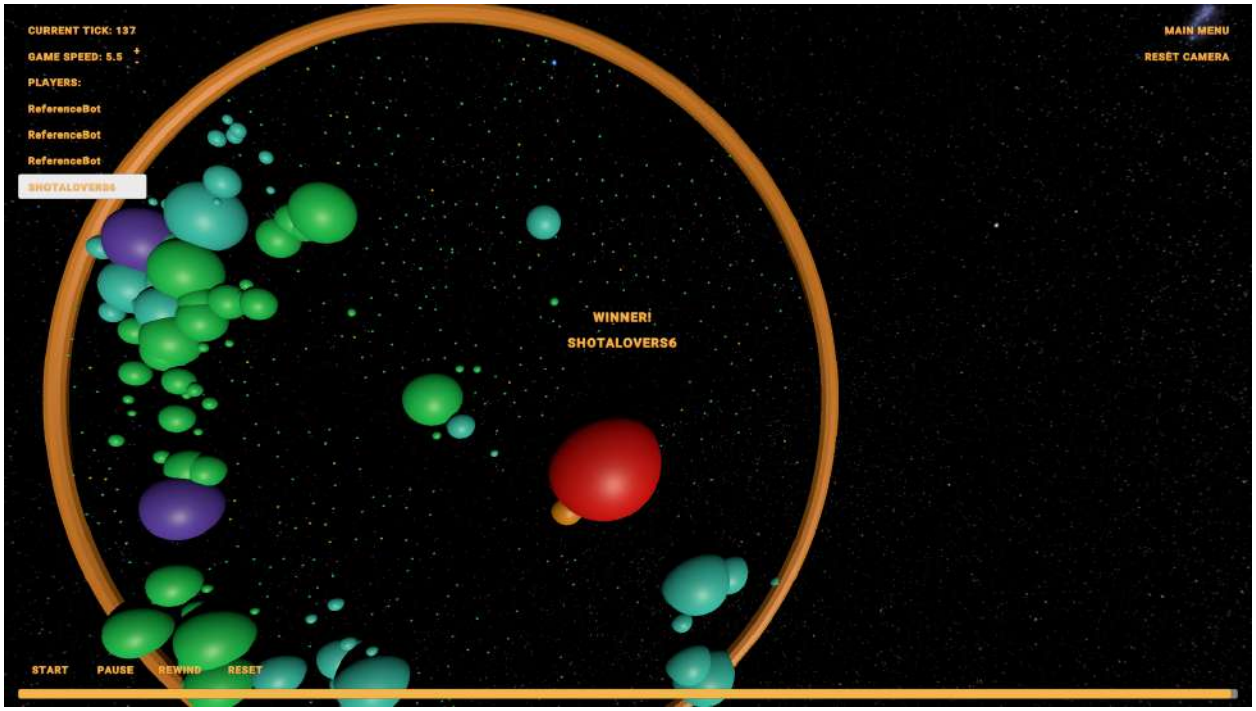
- Pengujian 3:
  Optimal: Kapal mampu menembak ReferenceBot secara akurat agar lebih kecil
  Tidak optimal: Kapal agak lambat dalam memakan *food*



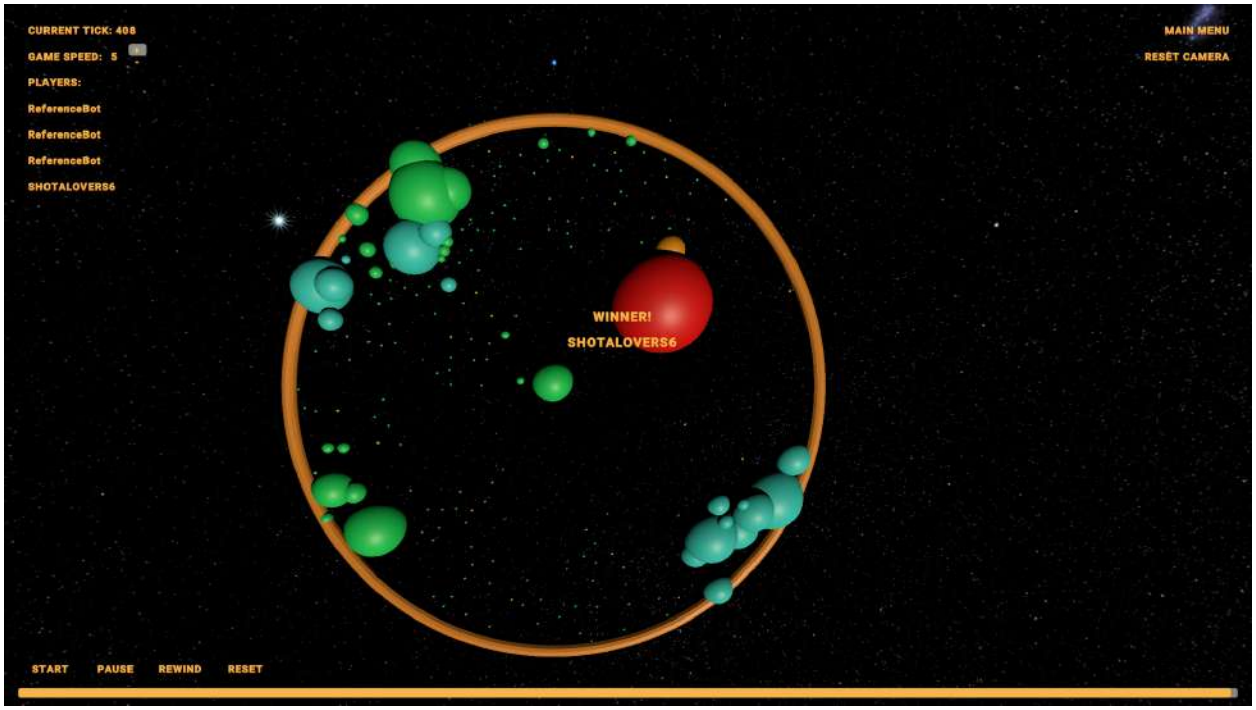
- Pengujian 4:
  Optimal: Kapal mampu memakan *food* dengan banyak
  Tidak optimal: Kapal menghabiskan terlalu banyak waktu di awan gas



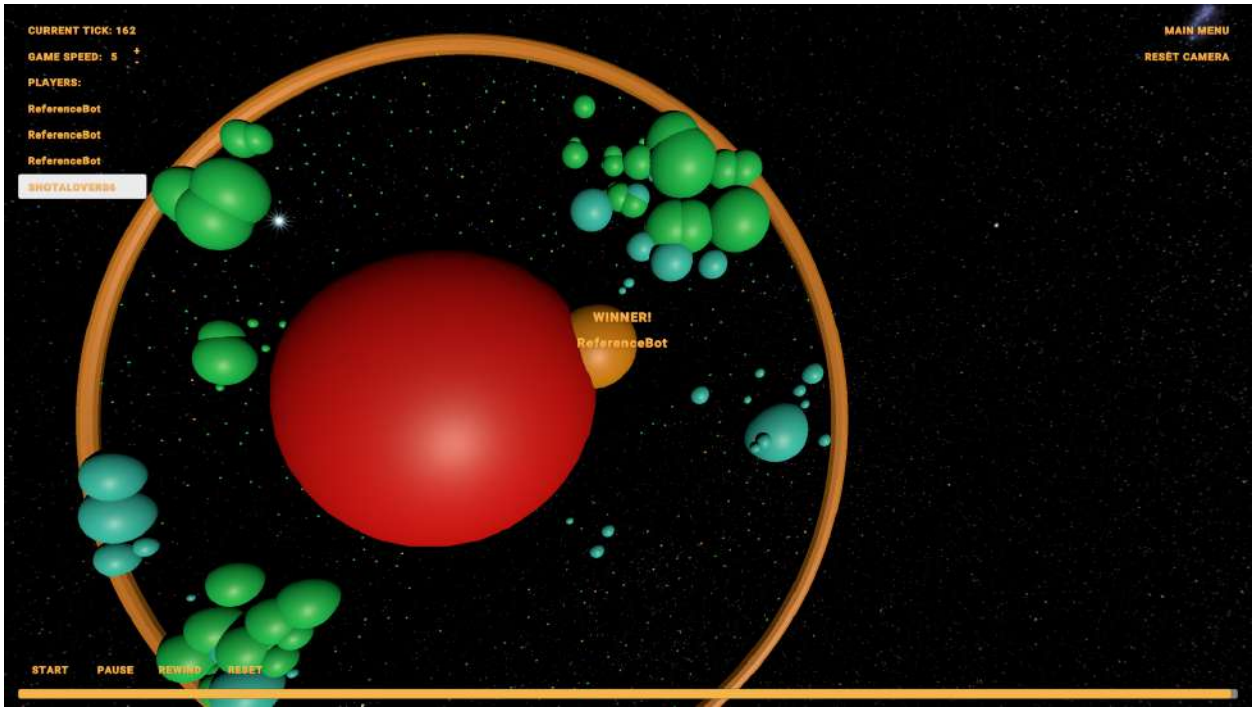
- Pengujian 5:
  Optimal: Kapal mampu menembak ReferenceBot secara akurat agar lebih kecil, kapal dapat memakan *food* secara maksimal dan menghindari bahaya seperti awan gas
  Tidak optimal: -



- Pengujian 6:
  Optimal: Kapal mampu menembak ReferenceBot secara akurat agar lebih kecil
  Tidak optimal: -

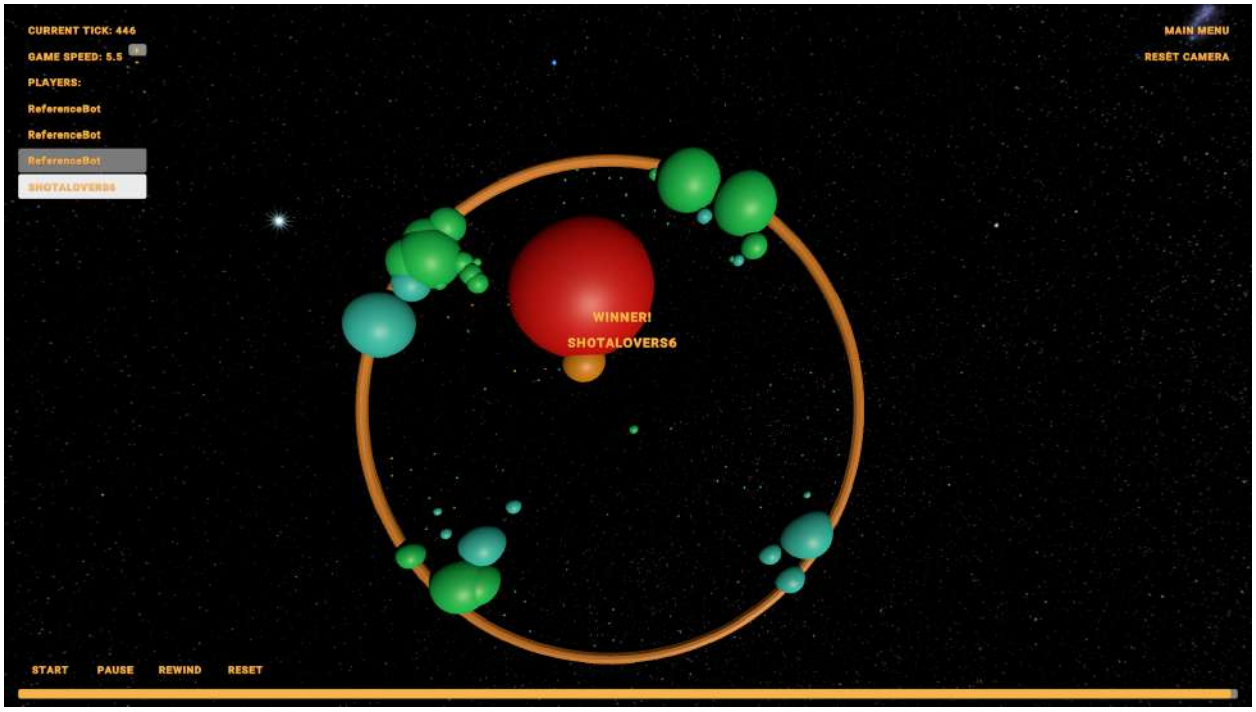


- Pengujian 7:
  Optimal: Kapal mampu memakan *food* dengan banyak
  Tidak optimal: Kapal menghabiskan terlalu banyak waktu di awan gas, kapal banyak menembak torpedo dan tidak mengenai lawan

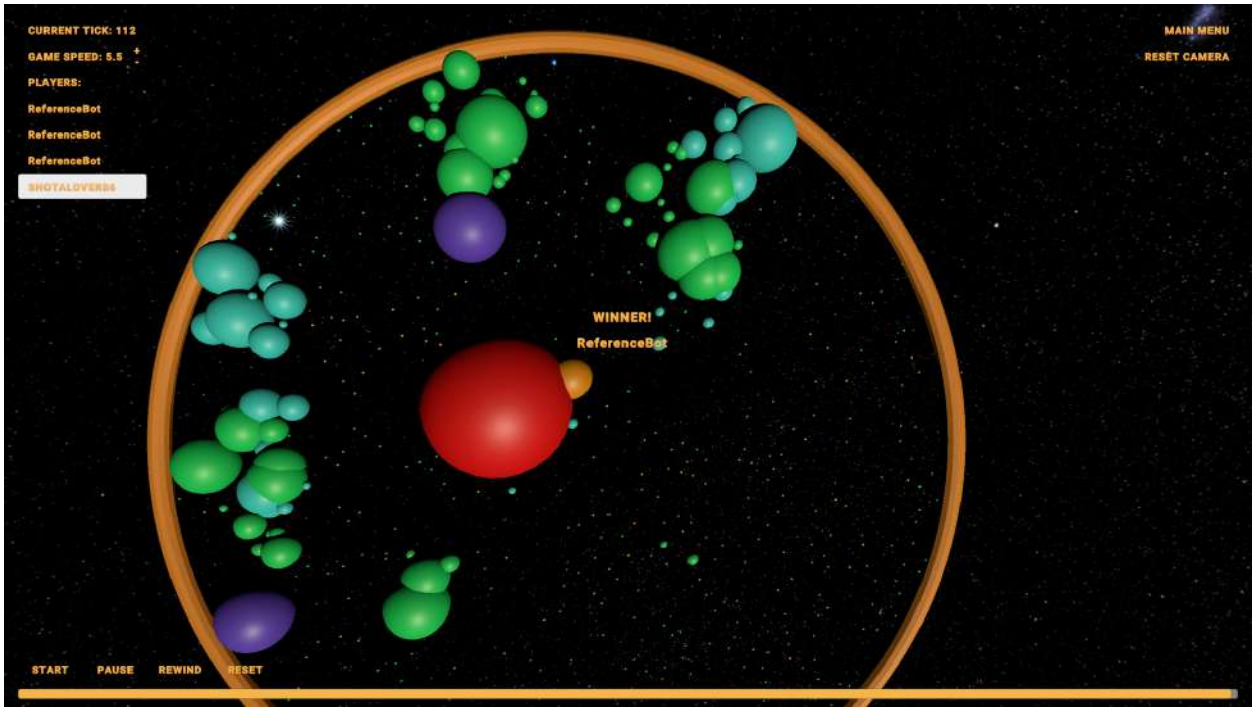


- Pengujian 8:
  Optimal: Kapal mampu memakan *food* dengan banyak, kapal dapat menembak lawan dengan lancar

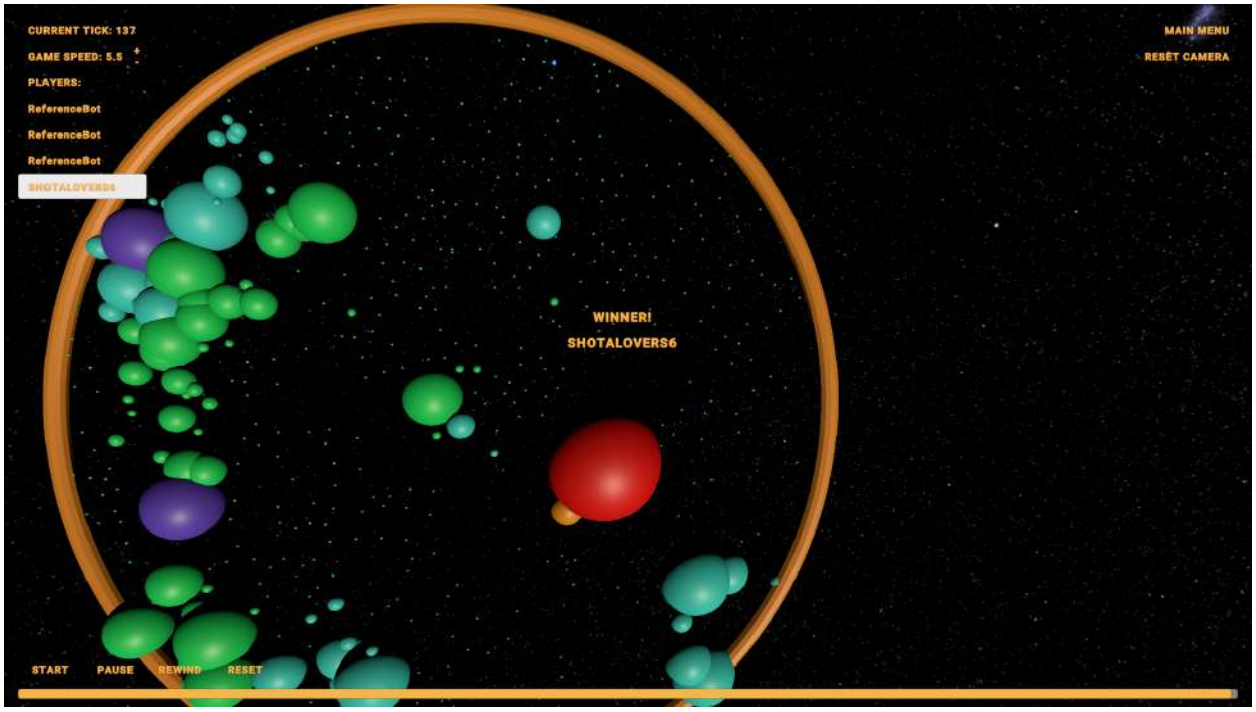
Tidak optimal: Kapal menghabiskan terlalu banyak waktu di awan gas



- Pengujian 9:
  Optimal: Kapal mampu memakan *food* dengan banyak
  Tidak optimal: Kapal banyak menembak torpedo dan tidak mengenai lawan



- Pengujian 10:
  Optimal: Kapal mampu menembak ReferenceBot secara akurat agar lebih kecil
  Tidak optimal: -



Hasil yang didapat dari melakukan 10 pengujian adalah bot kami berhasil memenangkan 6 dari 10 pertandingan. Dari sini didapat bahwa algoritma *greedy* yang kami buat cukup optimal dilihat dari persentase menang sebesar 60%.

BAB 5
Kesimpulan dan Saran

5.1. Kesimpulan

Dari Tugas Besar 1 IF2211 Strategi Algoritma Semester II 2022/2023 ini kami menyimpulkan bahwa kami dapat membangun bot dalam permainan Galaxio ini menggunakan algoritma *greedy*. Rupanya, dengan memanfaatkan algoritma *greedy*, terdapat berbagai alternatif strategi berkaitan dengan berbagai aspek permainan yang dapat digunakan oleh bot untuk memenangkan pertandingan: *Escaping, Chasing,* dan *Fetching*. Masing-masing dari alternatif tersebut memiliki kelebihan dan kekurangan. Oleh karena itu, kami mengoptimalkan tiap alternatif tersebut sesuai kondisi di sekitar kapal tersebut secara tepat agar kami mencapai hasil yang maksimal dan menguntungkan bagi kami. Walaupun begitu, algoritma *greedy* tidaklah sempurna untuk memecahkan setiap permasalahan pada waktu tertentu. Tiap strategi yang kami buat pun juga mungkin tidak memberi hasil maksimal seperti yang diharapkan.

5.2. Saran

Kami memiliki beberapa saran untuk disampaikan berkaitan dengan Tugas Besar 1 IF2211 Strategi Algoritma Semester II 2022/2023, yaitu:

- Walaupun sudah dibuat dokumen panduan untuk memodifikasi bot Galaxio, alangkah baiknya jika terdapat video tutorial untuk memudahkan kami untuk mengetes bot tersebut.
- Efektivitas dalam implementasi bot ini tentunya masih terdapat kekurangan dan keterbatasan dikarenakan fokus diarahkan kepada algoritma *greedy*.

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)-Bag2.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf
- https://github.com/EntelectChallenge/2021-Galaxio
- https://www.embibe.com/exams/intersection-between-circle-and-line/

Lampiran

- Link repository: https://github.com/haidarhamda/Tubes1_ShotaLovers6969
- Link YouTube: https://youtu.be/LHyQ8M92IwE