Tugas Besar 1 IF2211 Strategi
Algoritma Semester II tahun
2022/2023

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan“Galaxio”

Disusun oleh:

Muhammad Rizky Syaban	13521119
Ulung Adi Putra	13521122
Muhammad Zaki Amanullah	13521146

PROGRAM STUDI TEKNIK INFORMATIKA SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKAINSTITUT TEKNOLOGI BANDUNG

2023

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.

Tugas dari mahasiswa adalah diminta untuk mengimplementasikan bot permainan *galaxio* menggunakan Bahasa pemrograman Java. Dalam pembuatan bot tersebut, terdapat Batasan dalam pengimplementasian berupa mahasiswa hanya diperbolahkan menggunakan strategi algoritma *greedy* atau rakus. Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Galaxio*. 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
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*
7. *Salvo* akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
8. *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.*
9. *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.
10. 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.

11. 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:
    1) FORWARD
    2) STOP
    3) START_AFTERBURNER
    4) STOP_AFTERBURNER
    5) FIRE_TORPEDOES
    6) FIRE_SUPERNOVA
    7) DETONATE_SUPERNOVA
    8) FIRE_TELEPORTER
    9) USE_SHIELD
12. 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

2.1. Algoritma Greedy

Algoritma Greedy adalah algoritma yang mengimplementasikan sifat “rakus” dalam menyelesaikan persoalan. Algoritma greedy menyelesaikan persoalan dengan cara mengambil solusi dari setiap Langkah. Dalam tiap evaluasi akan dipilih salah satu solusi yang akan menghasilkan keputusan terbaik pada tiap Langkah tersebut. Namun, dalam algoritma greedy kita tidak diperbolehkan untuk melakukan *back tracking* atau tidak diperbolehkan untuk mengevaluasi Langkah yang sebelumnya diambil untuk menentukan Langkah sekarang. Oleh karena itu, diharapkan setiap pengambilan keputusan dalam tiap Langkah selalu memberikan solusi yang optimum local. Sehingga, diharapkan setiap solusi optimum local akan menghasilkan solusi yang optimum juga untuk keseluruhan program (optimum global).

Suatu persoalan dapat diselesaikan dengan algoritma *greedy* apabila persoalan tersebutmemiliki dua sifat berikut:

- Solusi optimal dari persoalan dapat ditentukan dari solusi optimal subpersoalan tersebut.
- Pada setiap persoalan, terdapat suatu langkah yang dapat dilakukan dimana langkah tersebut menghasilkan solusi optimal pada subpersoalan tersebut. Langkah ini juga dapat disebut sebagai *greedy choice.*

Dalam mengimplementasikan algoritma greedy, kita memerlukan beberapa kompone yang perlu didefinisikan terlebih dahulu. Komponen tersebut adalah :

- Himpunan kandidat (C): Berisi kandidat yang mungkin dipilih pada setiap langkahnya.
- Himpunan solusi (S): Berisi kandidat yang sudah terpilih sebagai solusi.
- Fungsi solusi (*solution* ***function***): Menentukan apakah himpunan solusi yang dikumpulkan sudah memberikan solusi. (Domain: himpunan objek, Range: boolean).
- Fungsi seleksi (*selection* ***function***): Memilih kandidat berdasarkan strategi *greedy* tertentu. Fungsi ini memiliki sifat heuristik (fungsi dirancang untuk mencari solusi optimum dengan mengabaikan apakah fungsi tersebut terbukti paling optimum secara

matematis). (Domain: himpunan objek, Range: objek).

- Fungsi kelayakan (*feasibility **function***): Memeriksa apakah kandidat yang terpilih oleh fungsi seleksi dapat dimasukkan ke dalam himpunan solusi. (Domain: himpunan objek, Range: boolean).
- Fungsi objektif (*objective **function***): Memaksimumkan atau meminimumkan suatu parameter pada suatu persoalan. (Domain: himpunan objek, Range: himpunan objek).

2.2 Cara Kerja Bot

Cara kerja bot pada permain Galaxio adalah, pertama tama bot akan mengambil data game state terlebih dahulu. Data yang bisa diambil dari Game state adalah sebagai berikut:

1. currentTicks : Satuan waktu pada game.
2. Radius : Ukuran dari peta permainan
3. List game object yang terdiri dari
    - Id : nomor dari objek
    - Size : ukuran objek
    - Speed : kecepatan objek
    - currentHeading : arah pergerakan dari objek
    - position : posisi object dalam map
    - gameObjectType : tipe dari game objek, bisa berupa player, gasCloud, asteroid, dan lain lain.
    - torpedoSalvoCount : Jumlah torpedo yang dimiliki oleh objek, untuk objek yang bukan bertipe player bernilai 0
    - supernovaAvaiable : apakah objek memiliki supernova atau tidak
    - teleporter Count : jumlah teleporter yang dimiliki objek, untuk objek yang bukan bertipe player bernilai 0
    - shieldCount : shield yang dimiliki oleh objek, untuk yang bertipe bukan player bernilai 0.

Setelah bot mendapatkan semua info yang ada pada game state, bot akan melakukan Analisis menggunakan algortima yang sudah dituliskan. Algoritma yang dituliskan menggunakan strategi greedy untuk menentukan action yang akan dilakukan oleh bot.

2.3 Cara menjalankan game

1. Jalankan game engine
    a. Jalankan game-runner

Formatted: Default Paragraph Font, Font: 12 pt, Complex Script Font: 12 pt,

Formatted: Default Paragraph Font, Font: 12 pt, Complex Script Font: 12 pt

Untuk menjalankan game-runner pindah direktori ke runner-publish lalu jalankan dotnet GameRunner.dll

b. Jalankan game-engine

Untuk menjalankan game-runner pindah direktori ke engine-publish lalu jalankan dotnet Engine.dll

c. Jalankan game-logger

Untuk menjalankan game-runner pindah direktori ke runner-publis lalu jalankan dotnet Logger.dll

2. Jalankan semua bot

Jumlah bot yang dijalankan secara default adalah 4 buah dan dapat diganti dalam file app.setting.json pada folder engine-publish dan runner-publish. Masuk ke lokasi dimana file excecutable berada dan jalankan file tersebut. Sebagai contoh, untuk menjalankan reference bot bisa jalankan dotnet ReferenceBot.dll di terminal

2.4 Cara menjalankan visualizer

Pada folder visualizer extraxt file visualizer sesuai dengan operating system. Buka visualizer lalu masuk option dan ubah path menuju folder dimana file game_state_log.json berada. Masuk ke load lalu pilih gamelog yang ada lalu klik start dan nikmati permainan.

Formatted: Default Paragraph Font, Font: 12 pt, Complex Script Font: 12 pt, English (Indonesia)

Formatted: Default Paragraph Font, Font: 11 pt, Complex Script Font: 11 pt,

Formatted: Default Paragraph Font, Font: 12 pt, Complex Script Font: 12 pt, English (Indonesia)

Formatted: Default Paragraph Font, Font: 11 pt, Complex Script Font: 11 pt,

Formatted: Default Paragraph Font, Font: 12 pt, Complex Script Font: 12 pt, English (Indonesia)

Formatted: Default Paragraph Font, Font: 11 pt, Complex Script Font: 11 pt,

Formatted: Default Paragraph Font

BAB III
STRATEGI GREEDY

A. Elemen-Elemen Algoritma Greedy dalam persoalan *Galaxio*

- Himpunan kandidat (C): Aksi pemain dan heading pemain yang dapat dilakukan oleh bot. Heading pemain berupa sudut bertipe integer dengan skala 0 samapai 359. Aksi pemain yang dapat dilakukan oleh bot adalah :
  1. FORWARD
  2. STOP
  3. START_AFTERBURNER
  4. STOP_AFTERBURNER
  5. FIRE_TORPEDOES
  6. FIRE_SUPERNOVA
  7. DETONATE_SUPERNOVA
  8. FIRE_TELEPORTER
  9. USE_SHIELD
- Himpunan solusi (S): Aksi dan arah heading yang dipilih
- Fungsi solusi (*solution* ***function***): Memeriksa apakah solusi yang dipilih adalah solusi yang dapat membuat bot bertahan hingga terakhir.
- Fungsi seleksi (*selection* ***function***): Memilih aksi dan heading yang dapat membuat bot mengeliminasi pemain lain atau bertahan tergantung dengan kondisi dari game state.
- Fungsi kelayakan (*feasibility* ***function***): semua solusi yang diperoleh layak, karena tidak ada kondisi yang membuat aksi tidak valid.
- Fungsi objektif (*objective* ***function***): Aksi yang dipilih dapat membuat bot bertahan sampai permainan berakhir.

Elemen – elemen algoritma greedy dalam setiap strategi yang diimplementasikan :

1. Elemen-elemen algoritma greedy dalam menghindari musuh dan obstacle terdekat saat menghindari border.
   - Himpunan kandidat (C)

     Semua heading yang mungkin dilakukan oleh bot {1..360}.
   - Himpunan solusi (S)

     Heading yang menghindari semua objek dengan tipe PLAYER dengan

ukuran yang lebih besar dengan bot dan semua objek dengan tipe GASCLOUD dan WORMHOLE di resultankan dengan heading ke center point (0, 0).

- Fungsi solusi (Solution function)

  Mengecek kesemua heading yang berlawanan dengan heading bot dengan obstacle baik itu PLAYER dengan ukuran yang lebih besar dengan bot maupun GASCLOUD dan WORMHOLE dengan melakukan iterasi untuk mengetahui setiap heading yang berlawanan dengan obstacle yang sudah diseleksi dengan fungsi seleksi lalu mencari resultannya.

- Fungsi seleksi (Selection function)

  Menyaring semua objek bertipe PLAYER dengan ukuran yang lebih besar dengan ukuran bot yang memiliki jarak dengan bot lebih besar dengan jarak bot ke makanan target dan menyaring semua WORMHOLE serta GASCLOUD yang memiliki jarak lebih dekat dengan bot daripada jarak dari bot ke border.

- Fungsi kelayakan (Feasibility function)

  Semua heading yang diperoleh layak.

- Fungsi objektif (Objective function)

  Mencari heading yang aman dari semua lawan dan semua gascloud dan wormhole serta dapat menghindari border.

2. Elemen-elemen algoritma greedy dalam menghindari musuh dan obstacle terdekat saat mencari makan.

   - Himpunan kandidat (C)

     Semua heading yang mungkin dilakukan oleh bot {1..360}.

   - Himpunan solusi (S)

     Heading yang menghindari semua objek dengan tipe PLAYER dengan ukuran yang lebih besar dengan bot dan semua objek dengan tipe GASCLOUD dan WORMHOLE.

   - Fungsi solusi (Solution function)

     Mencari semua heading yang berlawanan dengan heading bot dengan obstacle baik itu PLAYER dengan ukuran yang lebih besar dengan bot maupun GASCLOUD dan WORMHOLE dengan melakukan iterasi untuk mengetahui setiap heading yang berlawanan dengan obstacle yang sudah diseleksi dengan fungsi seleksi lalu mencari resultannya.

   - Fungsi seleksi (Selection function)

Menyaring semua objek bertipe PLAYER dengan ukuran yang lebih besar dengan ukuran bot yang memiliki jarak dengan bot lebih besar dengan jarak bot ke makanan target dan menyaring semua WORMHOLE serta GASCLOUD yang memiliki jarak lebih dekat dengan bot daripada jarak dari bot ke makanan terdekat.

- Fungsi kelayakan (Feasibility function)
  Semua heading yang diperoleh layak.
- Fungsi objektif (Objective function)
  Mencari heading yang aman dari semua lawan dan semua gascloud dan wormhole.

3. Elemen-elemen algoritma greedy dalam menembak torpedo
   - Himpunan Kandidat (C)
     Semua heading yang mungkin dilakukan oleh bot {1..360}.
   - Himpunan solusi (S)
     Heading yang mengarah ke target dan action yang dilakukan adalah FIRETORPEDOS
   - Fungsi solusi (Solution function)
     Memeriksa apakah heading yang didapat sudah sesuai atau tidak
   - Fungsi seleksi (Selection function)
     Mencari bot lawan yang memiliki ukuran yang lebih besar dengan kita dan terdekat kemudian set heading bot kita menuju ke bot target. Jika tidak ada pemain yang lebih besar dari kita, maka akan dicari pemain terdekat yang lebih kecil daripada bot kita, kemudian set heading bot menuju ke target. Setelah heading didapatkan aksi yang dilakukan adalah FIRETORPEDOS. Prioritas target toperdo adalah pemain yang lebih besar dari kita, hal ini bertujuan untuk mengikis ukuran target. Selain itu, pemain lawan dengan ukuran yang lebih besar memiliki tingkat ancaman yang lebih tinggi karena dapat mengeleminasi bot kita.
   - Fungsi kelayakan (Feasibility function)
     Semua target layak
   - Fungsi objektif (Objective function)
     Mencari heading dari bot ke torpedo.
4. Elemen-elemen algoritma greedy dalam memilih target makanan
   - Himpunan Kandidat (C)

Semua makanan yang ada di dalam world.

- Himpunan solusi (S)

  Sebuah gameObject berupa makanan yang aman.

- Fungsi solusi (Solution function)

  Mengecek apakah makanan tersebut aman atau tidak, apabila tidak aman, prioritaskan menghindar dari musuh.

- Fungsi seleksi (Selection Function)

  Kita akan selalu mengambil makanan yang layak yang terdekat dan layak.

- Fungsi kelayakan (Feasibility function)

  Semua solusi layak.

- Fungsi objektif (Objective function)

  Mencari heading dari bot ke target.

5. Elemen-elemen algoritma greedy dalam menghindari serangan teleporter lawan

- Himpunan Kandidat (C)

  Semua heading yang mungkin untuk meluncurkan teleporter.

- Himpunan solusi (S)

  Heading untuk meluncurkan teleporter yaitu heading yang ditujukan kepada makanan atau kepada pemain lain yang memiliki ukuran lebih kecil dari bot.

- Fungsi solusi (Solution function)

  Fungsi yang mengiterasi setiap kemungkinan *escape route*, yaitu heading ke player dengan ukuran yang lebih kecil atau ke makanan yang paling jauh. Apabila tidak terdapat player dengan ukuran yang lebih kecil, maka prioritaskan heading ke makanan yang paling jauh.

- Fungsi seleksi (Selection Function)

  Target yang diprioritaskan paling pertama adalah pemain dengan ukuran yang lebih kecil, apabila tidak ditemukan yang aman maka akan dilanjutkan pencarian kepada makanan yang paling aman. Selanjutnya, apabila tidak ditemukan makanan yang aman untuk target teleporter, maka kita tidak akan menggunakan teleporter.

- Fungsi kelayakan (Feasibility function)

  Semua solusi layak.

- Fungsi objektif (Objective function)

  Mencari heading dari bot ke target.

B. Alternatif Solusi Greedy dalam persoalan *Galaxio*

1. **Strategi Menghindari Edge**
    a. **Strategi Menuju Center Point**

        Dalam strategi ini, apabila kita sudah mendekati border atau ujung arena, kita akan mengubah heading kita menjadi menuju ke center point dari world. Ini merupakan strategi yang cukup efisien karena pergerakan border bisa saja lebih cepat dari kecepatan bot.

    b. **Strategi Menyusuri**

        Kita akan menyusuri border apabila sudah mendekati border. Strategi yang lebih sulit diimplementasikan karena harus memperhatikan arah gerak kapal apakah mengarah searah atau berlawanan jarum jam. Dan kita juga secara teknis tidak mempertimbangkan kecepatan mengecilnya border sehingga merupakan strategi yang lebih beresiko.

2. **Strategi Menghindari Bigger Player**
    a. **Strategi Heading Berlawanan**

        Strategi ini merupakan alternatif strategi yang lebih konservatif yakni apabila kita menemukan player dengan ukuran yang lebih besar dengan ukuran kita, kita akan langsung mengubah heading bot dengan heading berlawanan dengan arah dari bot ke musuh. Strategi ini tercoba lebih efektif menghindari musuh karena musuh yang memiliki ukuran lebih besar dengna kita tentunya memiliki kecepatan yang lebih rendah dan dengan menghindar 180 derajat kita dapat mengalahkan mereka dengan kecepatan kecuali apabila musuh menggunakan teleporter atau afterburner.

3. **Strategi Menghindari Gas Cloud**
    a. **Strategi Heading Berlawanan**

        Dalam strategi ini, kita akan langsung mengubah heading kita menjadi heading yang berlawanan dengan heading gas cloud. Alternatif ini cukup efektif apabila ukuran gas cloud dalam rentang sedang ke besar dan gas cloud mengumpul. Strategi ini akan kurang efektif apabila bot terjebak di antara dua buah kumpulan gas cloud yang akan membuat bot akan menggantikan arahnya 180 derajat berulang-ulang dan strategi ini kurang efisien apabila kita harus menghindari sebuah gas cloud yang berukuran sangat kecil.

b. Strategi Mengganti Heading Sedikit

4. Strategi Menembak Torpedo ke Semua Pemain

Dalam menggunakan strategi ini kita menggunakan torpedo untuk menembaknya kepada pemain yang berukuran lebih kecil dan pemain yang berukuran lebih besar dengan bot. Kita akan segera memasuki fase penembakan torpedo apabila kita memiliki amunisi torpedo dan kita memiliki size yang mencukupi untuk menembakkan torpedo. Strategi ini cukup efektif apabila kita ingin membunuh pemain yang lebih besar karena terdapat efek penambahan size jika torpedo mengenai pemain lain. Dengan strategi ini kita juga dapat menembakkan torpedo kepada pemain dengan ukuran yang lebih kecil untuk membunuh mereka dengan lebih mudah karena mereka memiliki kecepatan yang lebih cepat.

5. Strategi Menembak Teleporter

a. Menembakkan teleporter untuk menyerang

Dalam strategi ini kita menembakkan torpedo untuk berpindah secara instan dari posisi awal bot ke posisi teleport untuk menyerang lawan yang berukuran lebih kecil. Strategi ini merupakan strategi yang beresiko karena teleporter mengurangi size bot sebanyak 20 dan dari saat bot menembakkan teleporter hingga teleporter sampai ke tempat tujuan bisa saja bot lawan bertambah ukuran sehingga strategi ini merupakan strategi yang berbahaya.

b. Menembakkan teleporter untuk menghindari serangan dari teleporter

Kita hanya menggunakan teleporter sebagai mekanisme untuk menghindari serangan teleporter lawan. Kita akan mengecek apakah ada teleporter yang mengarah ke bot lalu kita akan melemparkan teleporter kita kepada pemain yang lebih kecil yang aman atau kepada makanan yang aman. Strategi ini akan lebih aman digunakan daripada strategi yang dijelaskan sebelumnya karena walaupun size berkurang 20 kita akan bebas dari serangan teleporter.

6. Strategi Mencari Makan

a. Strategi mencari makanan terdekat

Dalam strategi ini bot hanya akan memakan makanan yang paling dekat dan makanan yang layak untuk menghindari bot bolak-balik karena makanan yang layan memiliki jarak yang berbeda dengan makanan yang lain. Dalam mencari makanan terdekat kita juga menghindari musuh karena itu merupakan prioritas

utama bot. Dalam game, makanan yang telah dikonsumsi tidak akan muncul kembali, oleh karena itu bot harus memakan semua makanan secepat mungkin. Untuk itu, strategi ini cukup efektif.

b. Strategi mencari makanan dengan density paling banyak

Dalam implementasi strategi ini, bot akan mencari makanan dengan densitas paling tinggi, yaitu pada saat iterasi makanan paling dekat, bot akan mencari semua makanan lain dalam radius tertentu dari makanan tersebut dan menjumlahkan hasilnya untuk mendapatkan densitas di sekitar makanan tersebut. Dalam percobaan, bisa saja makanan paling padat adalah makanan yang jauh dan karena kita akan menghindari semua musuh dan obstacle yang lebih dekat dari makanan maka bisa saja pada saat mancari musuh yang lebih dekat dari pada makanan kita mencari dengan radius yang besar sehingga kemungkinan bot untuk mencari makan lebih kecil sehingga strategi ini kurang cocok untuk farming pada early game.

7. Strategi Menyalakan Shield

Dalam menggunakan strategi ini, kita menggunakan shield untuk bertahan dari serangan torpedo musuh. Sebelum memasuki mode menggunakan shield, akan dicek terlebih dahulu semua game objek yang ada pada map. Jika terdapat objek yang bertipe torpedo dengan radius 100 dari bot kita, maka akan dilakukan cek heading dari torpedo tersebut. Jika terdapat salah satu torpedo yang memiliki heading menuju ke arah bot kita, maka shield akan diaktifkan. Selain itu, terdapat prekondisi sebelum memakai shield, jika ukuran bot kita kurang dari 50 maka shield tidak dapat diaktifkan.

8. Strategi Menembakkan Supernova

Apabila supernova pick up sudah tersedia, maka supernova pick up akan dianggap sebagai food dan apabila sudah dekat maka bot akan dapat mengenalinya dan akan mengambilnya layaknya food dan superfood. Apabila kita sudah memiliki supernovapickup kita akan langsung menembakkanya pada tick selanjutnya. Setelah itu kita akan cek apabila supernova sudah cukup jauh dari jarak bot maka kita akan langsung mendetonasi supernova. Supernova yang ditembakkan akan mengarah ke arah bot lawan apapun. Untuk menghasilkan efek yang besar maka kita perlu menargetkan bot lawan yang paling banyak. Strategi ini cukup efektif apabila kita ingin menargetkan sebuah bot untuk supernova secara spesifik namun kurang efektif apabila kita ingin densitas bot

lawan paling banyak.

C. Strategi yang diimplementasikan

1. **Menghindari Edge**

Jika terdapat kondisi dimana bot kita berada di dekat batas, kelompok kami memilih untuk menggunakan strategi menuju ke center point. Hal ini dikarenakan pergi menuju center point lebih efektif daripada menyusuri batas. Kami mempertimbangkan adanya penyusutan ukuran map seiring berjalannya tik. Oleh karena itu, jika memilih strategi menyusuri map, perlahan ukuran kita akan terkikis oleh border map karena bot kita tidak menjauh dari batas map.

2. Menghindari Bigger Player

Strategi yang digunakan adalah dengan mendapatkan heading yang berlawanan dengan heading musuh karena heading tersebut menurut kami adalah yang paling aman untuk dilalui. Apabila ada lebih dari satu musuh di dekat kita, maka heading yang kita ambil adalah resultan dari semua heading opposite dari semua musuh yang mendekati.

3. Menghindari Gas Cloud

Strategi yang dipilih adalah strategi yang mengubah heading dari bot sedikit ke kanan atau ke kiri dari gas clous untuk menghindari. Strategi itu dipilih karena berdasarkan percobaan kita seringkali bot terjebak di antara banyak gas cloud sehingga apabila kita hanya mencari resultan dari gas cloud terdekat maka akan ada kemungkinan bot akan membolak-balikkan headingnya dan hal tersebut akan membuang-buang waktu untuk melakukan hal lain.

4. Menembak Torpedo

Torpedo akan selalu ditembakkan selagi masih ada amunisi torpedo salvo dan ukuran kita sudah melebihi 100. Strategi tersebut dipilih karena torpedo memiliki efek yang cukup besar dengan harga yang cukup murah yaitu 5 size. Karena torpedo akan mengurangi size bot yang terkena dan menambah size bot kami maka prioritas penembakkan torpedo adalah kepada arah bot yang memiliki ukuran yang lebih besar dari bot kita.

5. Menembak Teleporter

Dalam penggunaan teleporter, kelompok kami memilih menggunakan teleporter untuk menghindari teleporter lain. Penggunaan teleporter untuk menghindari serangan teleporter lain wajib digunakan, karena untuk antisipasi dari serangan musuh yang bisa

mengeliminasi bot kita secara tiba tiba. Hal ini perlu digunakan agar memenuhi objektif dari permainan yaitu menjadi last man standing, jadi semua algoritma untuk bertahan pasti kita pakai. Kita tidak memakai teleporter untuk menyerang karena kami menganggap hal tersebut mudah untuk dihindari oleh musuh. Kecepatan teleporter yang hanya 20 saja menjadi pertimbangan kami mengapa tidak memakai strategi ini. Untuk menyerang dengan torpedo yang memiliki kecepatan 60 atau 3 kali lipat dari teleporter saja, terkadang masih sering tidak mengenai sasaran. Selain karena ketidakakuratannya, berkurangnya ukuran bot saat menembakan teleporter menjadi salah satu alasan mengapa kelompok kami tidak menggunakannya. Selain itu, bot yang diimplementasikan hanya menggunakan strategi greedy, bukan bot artifial intelligent, sehingga kelompok kami merasa hanya akan membuang ukuran secara sia sia karena ketidakakuratan teleporter yang cukup tinggi.

6. Mencari Makan

Strategi mencari makan yang digunakan adalah memakan food atau superfood atau bahkan supernova yang terdekat. Apabila ktia tidak ada makanan yang layak maka bot akan berfokus untuk menghindari serangan dari bot lain yang lebih besar. Alasan dipilihnya strategi ini dan bukan strategi pencarian makan berdasarkan densitas adalah karena dengan mancari makanan berdasarkan densitas kita bisa saja mengganti heading ke makanan yang berjarak relatif jauh dengan kita. Pencarian berdasarkan dnesitas juga kurang sesuai dengan pencarian musuh. Mekanisme pencarian musuh dilakukan dengan mencari musuh yang lebih dekat daripada makanan target. Dengan memilih makanan yang memiliki densitas paling besar, bisa saja kita memilih makanan yang jauh dan peluang tidak ada musuh akan berkurang sehingga dengan strategi densitas kita akan lebih berfokus untuk menghindari musuh daripada mencari makan. Oleh karena itu, strategi makanan terdekat lebih cocok untuk mencari makan.

7. Mengaktifkan Shield

Startegi ini wajib untuk diimplementasikan oleh kelompok kami. Hal ini dikarenakan objektif dari permainan ini adalah untuk menjadi last man standing, sehingga kita harus memakai semua startegi yang bisa untuk bertahan.

8. Menembakkan supernova

Kita akan segera menembakkan supernova ketika kita memiliki supernova pickup. Apabila kita sudah menembakkan supernova bomb, kita akan segera mendetonasi supernova apabila jaraknya sudah jauh dari bot karena prioritas kami adalah untuk

bertahan hidup.

BAB IV
IMPELEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma Greedy pada bot

1. Class Utility Function

Class ini tidak memiliki atribut dan memiliki 23 method yang berisi fungsi-fungsi yang digunakan dalam perhitungan-perhitungan dalam algoritma berikut pseudocode dari 23 method tersebut

```
Function public static getTrueDistance(object1 : GameObject, object2 : GameObject) ->
double
{ deskripsi}

   triangleX <- object1.getPosition().x - object2.getPosition().x
   triangleY <- object1.getPosition().y - object2.getPosition().y
   -> (triangleX * triangleX + triangleY * triangleY) -object1.getSize() -
   object2.getSize()
```

```
Function public static getAngle(Bot : GameObject, target : GameObject, bigger :
GameObject) -> integer
{deskripsi}
       direction1 <- toDegrees(Math.atan2(target.getPosition().y - bot.getPosition().y,
                        target.getPosition().x - bot.getPosition().x))
    direction1 <- (direction1 + 360) % 360
       direction2 <- toDegrees(Math.atan2(bigger.getPosition().y - bot.getPosition().y,
    bigger.getPosition().x - bot.getPosition().x))
    direction2 <- (direction2 + 360) % 360
    if (direction2 > direction1) then
       -> direction2 - direction1
    else
       -> direction1 - direction2
```

```
Function public static getAngle(bot : GameObject, target : GameObject, bigger :
GameObject) -> integer
{deskripsi}

   direction1 <- toDegrees(Math.atan2(target.getPosition().y - bot.getPosition().y,
   target.getPosition().x - bot.getPosition().x))
   direction1 <- (direction1 + 360) % 360
   direction2 <- toDegrees(Math.atan2(bigger.getPosition().y - bot.getPosition().y,
   bigger.getPosition().x - bot.getPosition().x))
   direction2 <- (direction2 + 360) % 360
   if (direction2 > direction1) then
       -> direction2 - direction1
   else
       -> direction1 - direction2
```

```
Function public static getDensity (target : GameObject, foodList : List) -> integer
{deksripsi}

   val <- 0
   for(integer i <- 0 i<foodList.size() i++){
   if (getTrueDistance(target, foodList.get(i)) < 200) then
        val <- val + foodList.get(i).getSize()
   -> val
```

```
Function public static isSave(Bot : GameObject, obstacleList : List, target :
GameObject) -> boolean
{deksripsi}

     i traversal [0..obstacleList.size()]
          if (obstacleList.get(i).getGameObjectType() == ObjectTypes.PLAYER) then

               if (getAngle(target, bot, obstacleList.get(i)) < 90 and
               (getTrueDistance(obstacleList.get(i), target) < getTrueDistance(bot,
               target))) then

                    -> false
               else if (getAngle(target, bot, obstacleList.get(i)) > 90) then
                    if ((getTrueDistance(bot, target)/2) > getTrueDistance(target,
                    obstacleList.get(i))) then
                         -> false
          else
               saveAngle <-
               toDegrees(Math.asin((obstacleList.get(i).getSize()+bot.getSize())/getDista
               nce(bot, obstacleList.get(i))))
               if (Math.abs(getHeadingBetween(bot, obstacleList.get(i))
               getHeadingBetween(bot, target)) > saveAngle) then
                    if (getTrueDistance(bot, target) > getTrueDistance(bot,
                    obstacleList.get(i))) then
                         -> false
        -> true
```

```
Function public static integer countEnemyNear(Bot : GameObject, List<GameObject>
enemies,double searchRadius) -> integer
     nearEnemyCount <- 0
     for (integer i <- 0 i < enemies.size() i++) then
          if (getTrueDistance(enemies.get(i), bot) < searchRadius) then
               nearEnemyCount <- 1 + nearEnemyCount
     -> nearEnemyCount
```

```
Function public static countObstacleNear(Bot : GameObject, <GameObject> obstacles,
searchRadius :double) -> integer
{deskripsi}

   nearEnemyCount <- 0
   i traversal [0.. obstacles.size()]
         if (getTrueDistance(obstacles.get(i), bot) < searchRadius) then
              nearEnemyCount <- 1 + nearEnemyCount
   -> nearEnemyCount
```

```
Function public static distanceFromEdge(Bot : GameObject, gameState : GameState) ->
double
{deskripsi}

     -> ((double)(gameState.getWorld().getRadius()) - 100) -
     distanceFromCenterPoint(bot, gameState)
```

```
Function public static findResultant(Bot : GameObject, enemies : List, enemyCount :
integer) -> integer
{deskripsi}

     if (enemies.size() == 0) then
          -> 0
     result <- (getHeadingBetween(bot, enemies.get(0)) + 180) % 360
     i traversal [1..enemyCount]
          result <- result + ((getHeadingBetween(bot, enemies.get(i)) + 180) % 360)
          result <- result/2
          result <- result % 360
     -> result
```

```
Function public static listSmaller(Bot : GameObject, gameState : GameState) -> List
{deksripsi}

     playerList <- gameState.getGameObjects().stream().filter(item ->
     item.getGameObjectType() == ObjectTypes.PLAYER and item.getSize() <
     bot.getSize()).sorted(Comparator.comparing(GameObject::getSize)).collect(Collectors
     .toList())

     -> playerList
```

```
Function public static getHeadingBetween(Bot : GameObject, otherObject : GameObject) ->
integer
{deskripsi}

    direction <- toDegrees(Math.atan2(otherObject.getPosition().y -
    bot.getPosition().y, bot.getPosition().x - bot.getPosition().x))
    -> (direction + 360) % 360
```

```
Function public static nearEdge(Bot : GameObject, gameState : GameState) -> boolean
{deskripsi}

    -> (distanceFromCenterPoint(bot, gameState)) (double) (gameState.getWorld()
    .getRadius() - 30)
```

```
Function public static distanceFromCenterPoint(GameObject object, gameState :
GameState) -> double
{deskripsi}

    triangleX <- Math.abs(object.getPosition().x)
    triangleY <- Math.abs(object.getPosition().y)
    -> Math.sqrt(triangleX * triangleX + triangleY * triangleY) +
    (double)object.getSize()
```

```
Function public static bigestSmaller(Bot : GameObject, gameState : GameState) ->
GameObject
{deksripsi}

    playerList <- listSmaller(bot, gameState)
    -> playerList.get(playerList.size() - 1)
```

```
Function public static getHeadingToCenterPoint(Bot : GameObject, gameState : GameState)
-> integer
{deksripsi}

    direction <- toDegrees(Math.atan2(gameState.getWorld().getCenterPoint().y -
    bot.getPosition().y, gameState.getWorld().getCenterPoint().x -
    bot.getPosition().x))
```

```
Function public static getDistance(Bot : GameObject, target : GameObject) -> double
{deksripsi}

    triangleX <- Math.abs(bot.getPosition().x - target.getPosition().x)
    triangleY <- Math.abs(bot.getPosition().y - target.getPosition().y)
```

```
Function public static getDistanceFromCenterPoint (object : GameObject, gameState :
GameState) -> double
{deskripsi}

    triangleX <- Math.abs(object.getPosition().x - gameState.getWorld() .getCenterPoint
    ().x)
    triangleY <- Math.abs(object.getPosition().y - gameState.getWorld()
    .getCenterPoint().y)
    -> Math.sqrt(triangleX * triangleX + triangleY * triangleY) + (double)object
    .getSize() + 2
```

```
Function public static outOfBounds(GameObject object, GameState gameState) -> boolean
{deskripsi}

    if (getDistanceFromCenterPoint (object, gameState) > (double) gameState
    .getWorld().getRadius()) {
        -> true
    else
        -> false
```

```
Function public static getRelativeHeading(Bot : GameObject, otherObject : GameObject) -
> integer
{deksripsi}

    -> getHeadingBetween(bot, otherObject) - bot.getCurrentHeading()
```

```
Function public static getTeleporterList(gameState : GameState, bot : GameObject) ->
List<GameObject>
{deskripsi}
    teleList <- gameState.getGameObjects().stream().filter(item ->
    item.getGameObjectType() == ObjectTypes.TELEPORTER)
    .sorted(Comparator.comparing(item -> getDistance(bot, item)))
    .collect(Collectors.toList())
    -> teleList
```

```
Function public static avoidGasCloud(Bot : GameObject, GameObject gas, integer
prevHeading) -> integer
{deskripsi}

    saveAngle <- toDegrees(Math.asin((gas.getSize()+bot.getSize())/
    getDistance(bot,gas)))
    relativeHeading <- getHeadingBetween(bot, gas) - prevHeading
    if (getTrueDistance(bot, gas) <= 70) then
        if (relativeHeading < 0) then
            if (relativeHeading > -1*saveAngle) then
                -> getHeadingBetween(bot, gas)+saveAngle
        else
            if (relativeHeading < saveAngle) then
                -> getHeadingBetween(bot, gas)-saveAngle
    -> prevHeading
```

```
Function public static outOfBoundsWithId(UUID objectID, GameState gameState) -> boolean
{deskripsi}

    Teleporter <- gameState.getGameObjects().stream() .filter(item -> item
    .getGameObjectType() == ObjectTypes.TELEPORTER and item.getId() == objectID)
    .collect(Collectors.toList())
    myTeleporter <- Teleporter.get(0)
    -> outOfBounds(myTeleporter, gameState)
```

```
Function public static activateShield(Bot : GameObject, GameObject torpedo) -> boolean
{deskripsi}

    saveAngle <-toDegrees (Math.asin((bot .getSize() + torpedo.getSize()) /
    getDistance(torpedo, bot)))
    relativeHeading <- getHeadingBetween(torpedo, bot) - torpedo.getCurrentHeading()
       if (getTrueDistance(torpedo, bot) <= 100) then
        if (relativeHeading < 0) then
            if (relativeHeading > -1*saveAngle) then
                -> true
        else
            if (relativeHeading < saveAngle) then
                -> true
    -> false
```

2. method *computeNextPlayerAction* pada class botService

```
Procedure public void computeNextPlayerAction(playerAction : PlayerAction)
    botOutput <- "Random Heading"
    playerAction.action <- PlayerActions.FORWARD
    playerAction.heading <- new Random().nextInt(360)

    if (!gameState.getGameObjects().isEmpty()) then
        foodList <- gameState.getGameObjects().stream().filter(item -> item
        .getGameObjectType() == ObjectTypes.FOOD or item.getGameObjectType() ==
        ObjectTypes.SUPERFOOD or item.getGameObjectType() ==
        ObjectTypes.SUPERNOVAPICKUP)

        smallerPlayer <- gameState.getPlayerGameObjects().stream().filter(item ->

        item.getSize() < bot.getSize()) smallerPlayerList <- gameState

        .getPlayerGameObjects().stream().filter(item -> item.getSize() <

        bot.getSize()).sorted(Comparator.comparing(item -> UtilityFunctions

        .getTrueDistance(bot, item))).collect(Collectors.toList())




        foods <- Stream.concat(foodList, smallerPlayer).sorted(Comparator
        .comparing(item -> UtilityFunctions.getTrueDistance(bot, item)))
        .collect(Collectors.toList())

        biggerPlayer <- gameState .getPlayerGameObjects().stream().filter(item ->
        (item.getSize() >= bot.getSize() and UtilityFunctions.getDistance(bot, item)
        != 0)).sorted(Comparator.comparing(item ->
        UtilityFunctions.getTrueDistance(bot, item))).collect(Collectors.toList())

        obstacleList <- gameState.getGameObjects().stream().filter(item ->
        item.getGameObjectType() == ObjectTypes.GASCLOUD or item.getGameObjectType()
        == ObjectTypes.WORMHOLE)
        .sorted(Comparator.comparing(item -> UtilityFunctions.getTrueDistance(bot,

        item))).collect(Collectors.toList())



        torpedoList <- gameState.getGameObjects().stream().filter(item -> item
        .getGameObjectType() == ObjectTypes.TORPEDOSALVO).sorted(Comparator
        .comparing(item -> UtilityFunctions.getTrueDistance(bot, item)))
        .collect(Collectors.toList())

        teleporterList <- gameState.getGameObjects().stream().filter(item ->
        item.getGameObjectType() == ObjectTypes.TELEPORTER).sorted(Comparator
        .comparing(item -> UtilityFunctions.getTrueDistance(bot, item)))
        .collect(Collectors.toList())

        tempHeading <- 0
        if (UtilityFunctions.nearEdge(bot, gameState)) then
        botOutput <- "Going to center"

        centerHeading <- UtilityFunctions.getHeadingToCenterPoint(bot, gameState)

        enemiesNear <- UtilityFunctions.countEnemyNear(bot, obstacleList,
        UtilityFunctions.distanceFromEdge(bot, gameState))
```

```
obstaclesNear <- UtilityFunctions.countObstacleNear(bot, obstacleList,
UtilityFunctions.distanceFromEdge(bot, gameState))

playerAction.action <- PlayerActions.FORWARD
playerAction.heading <- centerHeading

if (enemiesNear > 0) then
     avoidEnemy <- UtilityFunctions.findResultant(bot, biggerPlayer,
     enemiesNear)
     finalHeading <- ((avoidEnemy + centerHeading) / 2) % 360
     if (obstaclesNear > 0) then
          finalHeading <- UtilityFunctions.avoidGasCloud(bot, obstacleList
          .get(0), finalHeading)
     playerAction.heading <- finalHeading
else if (obstaclesNear > 0) then
     botOutput <- "Avoiding gasCloud"
     foodHeading <- bot.getCurrentHeading()
     if (foods.size() > 0) then
          if (gameState.getWorld().getCurrentTick() % 2 == 0) then
               foodHeading <- getHeadingBetween(foods.get(0))
               tempHeading <- foodHeading
          else
               foodHeading <- bot.getCurrentHeading()
               tempHeading <- foodHeading
          botOutput <- "eating than avoid gas"
     playerAction.heading <- UtilityFunctions.avoidGasCloud(bot,
     obstacleList.get(0), foodHeading)
else if (bot.getTorpedoSalvoCount() > 0 and bot.getSize() > 100) then
     if (biggerPlayer.size() > 0) then
          heading <- getHeadingBetween(biggerPlayer.get(0))
          if (heading > centerHeading-60 && heading <centerHeading+60) then
               playerAction.action <- PlayerActions.FIRETORPEDOES
               playerAction.heading <- heading
               botOutput <- "Firing torpedoes near border"
          else
               botOutput <- "Avoiding border"
               playerAction.heading <- centerHeading
     else if (smallerPlayerList.size() > 0) then
          heading <- getHeadingBetween(smallerPlayerList.get(0))
          if (heading > centerHeading - 60 and heading < centerHeading + 60)
          then
               playerAction.action <- PlayerActions.FIRETORPEDOES
               playerAction.heading <- heading
               botOutput <- "Firing torpedoes to smaller players"
          else
               botOutput <- "Avoiding border"
               playerAction.heading <- centerHeading
     else if (smallerPlayerList.size() > 0) then
          heading <- getHeadingBetween(smallerPlayerList.get(0))
          if (heading > centerHeading - 60 && heading < centerHeading + 60)
          then
               playerAction.action <- PlayerActions.FIRETORPEDOES
```

```
                    playerAction.heading <- headin
                    botOutput <- "Firing torpedoes to smaller players"
               else
else if (bot.getTorpedoSalvoCount() > 0 and bot.getSize() > 100 or
bot.getSupernovaAvailable() == 1 or supernovaFired) then
     supernovaBomb <- gameState.getGameObjects().stream().filter(item ->
     (item.getGameObjectType() == ObjectTypes.SUPERNOVABOMB)) .sorted(Comparator
     .comparing(item -> UtilityFunctions.getTrueDistance(bot, item)))
     .collect(Collectors.toList())
     if (bot.getSupernovaAvailable() == 1) then
          playerAction.action <- PlayerActions.FIRESUPERNOVA
          supernovaFired <- true
          if (biggerPlayer.size() > 0) then
               botOutput <- "firing supernove ke bigger"
               playerAction.heading <- getHeadingBetween(biggerPlayer.get(0))
          else if (smallerPlayerList.size() > 0) then
               botOutput <- "firing supernove ke smaller"
               playerAction.heading <- getHeadingBetween  (smallerPlayerList
               .get(0))
     else if (supernovaFired && getDistanceBetween(bot, supernovaBomb.get(0)) >=
     400) then
          playerAction.action <- PlayerActions.DETONATESUPERNOVA
     else
          playerAction.action <- PlayerActions.FIRETORPEDOES
          if (biggerPlayer.size() > 0) then
               botOutput <- "firing torpedo ke bigger"
               playerAction.heading <- getHeadingBetween(biggerPlayer.get(0))
          else if (smallerPlayerList.size() > 0) then
               botOutput <- "firing torpedo ke smaller"
               playerAction.heading <- getHeadingBetween(smallerPlayerList .get(0))
else
     GameObject target <- null
     botOutput <- "Eating"
     playerAction.action <- PlayerActions.FORWARD
     if (foods.size() != 0) then
          if (foods.size() > 1) then
               if (Math.abs((int) Math.round(UtilityFunctions.getTrueDistance (bot,
               foods.get(0)) - UtilityFunctions.getTrueDistance(bot, foods.get(1))))
               < 5) then
                    if (foods.size() > 2 && Math.abs((int) Math.round
                    (UtilityFunctions.getTrueDistance(bot, foods.get(2)) -
                    UtilityFunctions.getTrueDistance(bot, foods.get(1)))) < 5) then
                         target <- foods.get(3)
                         playerAction.heading <- getHeadingBetween(foods.get(3))
                    else
                         target <- foods.get(2)
                         playerAction.heading <- getHeadingBetween(foods.get(2))
               else
                    target <- foods.get(0)
                    playerAction.heading <- getHeadingBetween(foods.get(0))
```

```
        else
            target <- foods.get(0)
                playerAction.heading <- getHeadingBetween(foods.get(0))

    if (target == null) {
        obstaclesNear <- UtilityFunctions.countObstacleNear(bot, obstacleList, 75)
        enemiesNear <- UtilityFunctions.countEnemyNear(bot, biggerPlayer, 100)
    else
        obstaclesNear <- UtilityFunctions.countObstacleNear(bot, obstacleList,
        UtilityFunctions.getDistance(bot, target))
        enemiesNear <- UtilityFunctions.countEnemyNear(bot, biggerPlayer,
        UtilityFunctions.getDistance(bot, target))

    if (enemiesNear > 0) then
        avoidEnemy <- UtilityFunctions.findResultant(bot, biggerPlayer,
        enemiesNear)
        if (obstaclesNear > 0) then
            botOutput <- "avoid enemy dan gasCloud"
            finalHeading <- UtilityFunctions.avoidGasCloud(bot,
            obstacleList.get(0), avoidEnemy)
        else
            botOutput <- "avoid enemy"
            finalHeading <- avoidEnemy
        playerAction.heading <- finalHeading
        playerAction.action <- PlayerActions.FORWARD
    else if (obstaclesNear > 0) then
        botOutput <- "Avoiding gasCloud"
        foodHeading <- new Random().nextInt(360)
        if (foods.size() != 0) then
            target <- foods.get(0)
            if (gameState.getWorld().getCurrentTick() % 2 ==0) then
              foodHeading <- getHeadingBetween(target)
            else
                foodHeading <- bot.getCurrentHeading()
            tempHeading <- foodHeading
        playerAction.heading <- UtilityFunctions.avoidGasCloud(bot,
        obstacleList.get(0), foodHeading)
```

```
        if (teleporterList.size() > 0) then
            i traversal [0..teleporterList.size()]
               if (UtilityFunctions.avoidTeleporter(bot, teleporterList.get(i)))
        then
                    if (fireTeleporter) then
                        if (teleporterLocked) then
                            botOutput <- "Teleporting"
                            if (UtilityFunctions.getDistance(teleporter, bot) <<- 20
        or UtilityFunctions.getDistance(teleporterList.get(i), bot) <<- 40) then
                                playerAction.action <- PlayerActions.TELEPORT
                                fireTeleporter <- false
                                teleporterLocked <- false
                        else
                            botOutput <- "Locking teleporter"
                            teleporter <- teleporterList.get(0)
                    else
                        if (bot.getTeleporterCount() > 0) then
                            if (smallerPlayerList.size() > 0) then
                                j traversal [smallerPlayerList.size()..0]
                                    if (UtilityFunctions.targetIsSaveToTeleport(bot,
        smallerPlayerList.get(j), gameState)) then
                                        botOutput <- "Teleporting to smaller player"
                                        playerAction.heading <-
        getHeadingBetween(smallerPlayerList.get(j))
                                        playerAction.action <-
        PlayerActions.FIRETELEPORT
                                        fireTeleporter <- true
                                        break
                             else if (foods.size() > 0) then
                                j traversal [foods.size() - 1..0]
                                    if (UtilityFunctions.targetIsSaveToTeleport(bot,
        foods.get(j), gameState)) then
                                        botOutput <- "Teleporting to ideal food"
                                        playerAction.heading <-
        getHeadingBetween(foods.get(j))
                                        playerAction.action <-
        PlayerActions.FIRETELEPORT
                                        fireTeleporter <- true
                                        break
                                break
    String aktifga <- "ga aktif"
    if (torpedoList.size()>0)then
        if (UtilityFunctions.activateShield(bot, torpedoList.get(0)) and
      bot.getSize()>50)then
            playerAction.action <- PlayerActions.ACTIVATESHIELD
            aktifga <- "aktif"

    output("Game Tick : " + gameState.getWorld().getCurrentTick())
    if (obstacleList.size()>0)then
        output("Jarak gas cloud terdekat :" + UtilityFunctions.getTrueDistance(bot,
    obstacleList.get(0)))
    if (biggerPlayer.size()>0)then
        output("Jarak enemy terdekat :" + UtilityFunctions.getTrueDistance(bot,
    biggerPlayer.get(0)))
    if (foods.size()>0)then
        output("Jarak Makanan terdekat :" + UtilityFunctions.getTrueDistance(bot,
    foods.get(0)) + ", " + foods.get(0))
    if (foods.size()>1)then
        output("Makanan kedua terdekat :"+ UtilityFunctions.getTrueDistance(bot,
    foods.get(1)))
```

```
output("Heading : " + botOutput + " " + aktifga)
output("tempHeading : " + tempHeading)
output("current Heading :" + bot.getCurrentHeading())
output("Size :" + bot.getSize())
output("Jarak ke edge :" + UtilityFunctions.distanceFromEdge(bot, gameState) + 100)
output("\n")
```

B. Analisis Desain Solusi

Strategi greedy yang menghasilkan solusi optimal seperti mencari makanan terdekat dan juga menembak torpedo ke player lein. Namun ada beberapa komponen yang kami tidak dapat gunakan karena strategi greedy tidak dapat berjalan optimal jika diimplementasikan seperti *afterburner*, dan juga teleport untuk menyerang karena sangat memungkinkan untuk gagal dalam mencapai target.

Strategi dalam menghindari gascloud juga merupakan strategi optimal dikarenakan dari pengujian yang kami lakukan sangat jarang bot kami menabrak gas cloud (jika tidak ada musuh yang lebih besar)

Namun sesekali bot terlihat bingung jika haru smenghadapi banyak pemain dan juga banyak gas cloud di radius tertentu.

Bab V

Kesimpulan dan Saran

A. Kesimpulan

Pada tugas besar kali ini kami dibuat berpikir untuk menciptakan algoritma greedy terbaik yang bisa diimplementasikan pada permainan Galaxio. Tujuan dari algoritma yang diimplementasikan adalah mengambil aksi dan heading dari bot yang paling efektif untuk memenangkan pertandingan atau menjadi pemain yang tidak gugur sampai terakhir. Kami juga belajar tentang bagaimana Kerjasama antar tim yang efektif dan saling bertukar pikiran dalam merancang strategi permainan Galaxio.

B. Saran

Untuk kedepannya jika menemuhi deskripsi tugas yang serupa dengan ini, pahami terlebih dahulu spesifikasi dan aturan yang ada pada game. Setelah itu baru diskusikan algoritma yang tepat untuk memenangkan permainan. Selain itu, sebisa mungkin untuk mengerjakan tugas ini dalam satu device dan satu tempat yang sama agar menghindari konflik pada repository serta memudahkan menyusun strategi.

Bab VI

Lampiran

A. Link Repository

https://github.com/Ulung32/Tubes1_eres-el-mejor-portero-del-mundo.git