TUGAS BESAR 1 STRATEGI ALGORITMA

IMPLEMENTASI ALGORITMA *GREEDY*

Disusun untuk memenuhi laporan tugas besar mata kuliah Strategi Algoritma semester 2 Institut Teknologi Bandung

Disusun oleh kelompok *PlayerNumberOne* :

Nigel Sahl	13521043
Hosea Nathanael Abetnego	13521057
Hanif Muhammad Zhafran	13521157

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

Kata Pengantar

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa, karena berkat-Nya begitu melimpah dalam kehidupan kami. Kami dapat menyelesaikan tugas besar pertama untuk mata kuliah Strategi Algoritma.

Berikut disajikan laporan tugas besar ini. Laporan ini berisi penjelasan terhadap persoalan dari tugas besar pertama

Dengan semua informasi yang kami peroleh

Kami berharap laporan ini dapat menjadi media pembelajaran dan eksplorasi kamu khususnya dalam bidang Strategi Algoritma dengan menggunakan Algoritma *Greedy*.

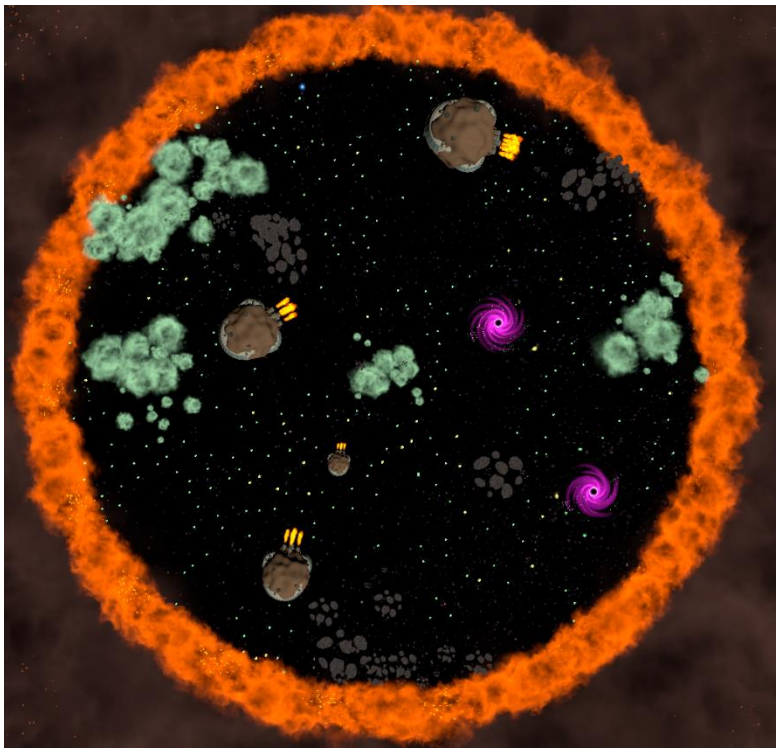
Bandung, 17 Februari 2023

Penyusun Laporan

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, tugas ini menggunakan sebuah *game* engine yang mengimplementasikan permainan Galaxio. *Game* engine diperoleh pada laman berikut:

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

Pada tugas ini, kami mengimplementasikan bot kapal dalam permainan Galaxio dengan menggunakan strategi *greedy* untuk memenangkan permainan. Untuk mengimplementasikan

bot tersebut, kami melanjutkan dan memodifikasi 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 *Player*s, 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 mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa *command* yang dapat dilakukan oleh *player*. Setiap tick, *player* hanya dapat memberikan satu *command*. Untuk daftar *command*s yang tersedia, bisa merujuk ke tautan panduan di spesifikasi tugas
11. Setiap *player* akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal *player* lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi.

Adapun peraturan yang lebih lengkap dari permainan Galaxio, dapat dilihat pada laman :

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

BAB 2
LANDASAN TEORI

Greedy adalah paradigma algoritmik yang membangun solusi sepotong demi sepotong, selalu memilih bagian berikutnya yang menawarkan manfaat paling jelas dan langsung. Jadi masalah di mana memilih optimal secara lokal juga mengarah ke solusi global adalah yang paling cocok untuk *Greedy*. Algoritma ini merupakan metode yang paling populer dan sedehana untuk memecahkan persoalan optimasi. Terdapat dua macam persoalan dalam hal ini, yaitu persoalan maksimasi dan minimisasi. Strategi *greedy* berarti membuat keputusan pada setiap langkah tanpa memperhitungkan konsekuensinya pada langkah selanjutnya. Kami menemukan langkah lokal terbaik di setiap langkah untuk mencapai tujuan. Strategi *greedy* mengasumsikan bahwa sekumpulan keputusan lokal terbaik dapat mengarah pada optimalisasi global.

Prinsip *greedy*: "**take what you can get now!**". Algoritma *greedy* membentuk solusi langkah per langkah (step by step). Pada setiap langkah, terdapat banyak pilihan yang perlu dievaluasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Tidak bisa mundur lagi (kembali) ke langkah sebelumnya. Jadi pada setiap langkah, kami memilih optimum lokal (local optimum) dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (global optimum).

Algoritma *greedy* memecahkan persoalan secara langkah per langkah (step by step) sedemikian sehingga,

pada setiap langkah:

1. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "take what you can get now!")
2. Hal kedua, "berharap" bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Elemen-elemen algoritma *greedy*:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb.)
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi

4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif : memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

Program yang akan dibuat merupakan implementasi algoritma *greedy* dalam program bot pada starter pack yang disediakan.

Untuk memulai permainan, kami mengunduh dan mengekstrak arsip starter pack yang dapat diakses pada link https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2. Isi dari arsip starter pack adalah beberapa folder dan script sebagai berikut:

- engine-publish
- logger-publish
- reference-bot-publish
- runner-publish
- starter-bots
- visualiser
- building-a-bot.md
- README.md
- run.sh

Ada beberapa komponen yang perlu diketahui untuk memahami cara kerja *game* ini, mereka adalah:

1. Engine

   Engine merupakan komponen yang berperan dalam mengimplementasikan logic dan rules *game*.
2. Runner

   Runner merupakan komponen yang berperan dalam menggelar sebuah match serta menghubungkan bot dengan engine.

3. Logger

   Logger merupakan komponen yang berperan untuk mencatat log permainan sehingga kami dapat mengetahui hasil permainan. Log juga akan digunakan sebagai input dari visualizer

Garis besar cara kerja program *game* Galaxio adalah sebagai berikut:

1. Runner –saat dijalankan– akan meng-host sebuah match pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.
4. Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar match dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut BotCount yang dimiliki file JSON ”appsettings.json”. File tersebut terdapat di dalam folder “runner-publish” dan “engine-publish”.
5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah Recieve*Game*State karena memberikan status *game*.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi match.

Cara Menjalankan *Game*

Berdasarkan gambaran cara kerja program *game* yang telah disebutkan sebelumnya, berikut merupakan cara menjalankan *game* secara lokal di Windows:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON ”appsettings.json” dalam folder “runner-publish” dan “engine-publish”
2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah “dotnet *Game*Runner.dll”
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah “dotnet Engine.dll”

6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah “dotnet Logger.dll”
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON “*Game*StateLog_{Timestamp}” dalam folder “logger-publish”. Kedua file tersebut diantaranya *Game*Complete (hasil akhir dari permainan) dan proses dalam permainan tersebut.

Greedy

Algoritma yang digunakan pada program bot adalah algoritma *greedy* pada bagian file BotService dalam folder *service.* Secara garis besar, algoritma diaplikasikan dengan jarak dan size dari setiap *game* objek yang ada dan untuk jarak diurutkan berdasarkan yang terdekat dengan bot kami. Selanjutnya, bot akan makan food berdasarakan jarak terdekat. Pada setiap tick, bot kami akan menyeleksi pilihan-pilihan yang ada agar optimal.

BAB 3
APLIKASI STRATEGI *GREEDY*

I. Mapping Persoalan Galaxio

1. Himpunan Kandidat

   Himpunan kandidat yang terdapat dalam Galaxio adalah semua *command* yang tersedia dan kemungkinan semua objek yang dipilih. *Command* utama dalam *game* ini yaitu:

   a. *FORWARD*
   b. STOP
   c. STARTAFTERBURNER
   d. STOPAFTERBURNER
   e. FIRETORPEDOES
   f. FIRESUPERNOVA
   g. DETONATESUPERNOVA
   h. FIRETELEPORTER
   i. TELEPORT
   j. ACTIVATESHIELD

   Selain *command* utama di atas, terdapat tambahan untuk menjalankan *command FORWARD* yaitu getHeadingBetween yaitu menentukan arah dari bot kami dalam derajat. Kandidat lain dalam *game* ini adalah list dari *player* untuk kandidat heading kea rah *player* lawan dan list makanan (*food* dan *superfood*) untuk menentukan heading kearah makanan. Kami juga menggunakan list-list lain untuk membantu pemilihan tindakan yaitu list torpedo, asteroid, awan gas, *wormhole*, dan supernova.

2. Himpunan Solusi

   Himpunan solusi adalah himpunan seluruh *command* yang dapat memenangkan pertandingan.

3. Fungsi Solusi

   Fungsi solusi adalah fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.

4. Fungsi Seleksi

Fungsi seleksi dari algoritma *greedy* kami adalah penentuan objek yang valid dari list kandidat.

5. Fungsi Kelayakan
   Fungsi ini merupakan fungsi yang memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.

6. Fungsi Objektif
   Fungsi ini memaksimumkan atau meminimumkan dari himpunan yang dipilih.

II. Eksplorasi Alternatif Solusi *Greedy*

Alternatif-alternatif solusi algoritma *greedy* yang mungkin dipilih dalam persoalan Galaxio yaitu:

1. *Greedy* by Distance on Food
2. *Greedy* by Distance on *Player*
3. *Greedy* by Size on *Player*
4. Greedy by Distance on Food with Validation Optimization

   Sebelumnya sudah dilakukan cara untuk mendapatkan makanan dengan jarak terdekat, tetapi masih ditemukan masalah penentuannya. beberapa masalah yang ditemukan tersebut adalah makanan yang berada di sekamir gas cloud atau asteroid atau bahkan pemain lain. Oleh karena itu, dapat dilakukan optimasi dengan memvalidasi makanan-makanan tersebut dengan melakukan :

   a. Peroleh list makanan, list gas cloud, list asteroid, list *player* lawan dan diurutkan berdasarkan jarak terdekat
   b. Seleksi indeks ke-0 list makanan dengan pengecekan apakah makanan tersebut terletak di sekamir gas cloud atau asteroid maka mengembalikan false yang berarti food tidak valid dan jika food di sekamir *player* lawan (heading ke makanan di sekamir heading ke lawan)
   c. Jika fungsi seleksi valid bernilai false maka indeks food yang akan dituju akan ditambah satu

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi *command* FORWARD dan heading ke makanan yang telah divalidasi

Himpunan Solusi	Permutasi *command* FORWARD dan heading ke makanan yang valid
Fungsi Solusi	Memeriksa apakah makanan yang dituju adalah makanan yang valid dan *command* FORWARD dijalankan
Fungsi Seleksi	Memvalidasi semua makanan dari jarak terdekat hingga menemukan makanan yang valid (tidak berada pada *obstacle* lain atau di dekat *border* dengan menjalankan *command* FORWARD
Fungsi Kelayakan	Memastikan makanan yang dipilih adalah makanan yang valid
Fungsi Objektif	Bot bergerak ke arah makanan yang valid dengan jarak minimum

5. *Greedy* by Distance on Food and *Player* + Size

Dari yang kami ketahui sebelumnya, permasalahan dengan hanya berfokus kepada makanan saja atau *player* saja dapat menyulitkan. Oleh karena itu, kami perlu mengkombinasikan keduanya. Cara yang dapat dilakukan adalah :

a. Peroleh list makanan dan diurutkan berdasarkan jarak terdekat
b. Peroleh list *player* dan diurutkan berdasarkan jarak terdekat
c. Ketika bot memiliki size lebih kecil dari bot terdekat, bot akan mencari makanan
d. Ketika bot sudah lebih besar dari bot terdekat, maka ia akan langsung menyerang

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi *command* FORWARD dan heading menuju makanan atau bot lawan
Himpunan Solusi	Permutasi *command* FORWARD dengan heading yang mengarah ke makanan terdekat atau bot lawan terdekat yang memiliki size lebih kecil dari bot itu sendiri
Fungsi Solusi	Memeriksa apakah bot mengarah ke makanan terdekat jika bot lawan terdekat memiliki ukuran yang lebih besar atau mengarah ke bot lawan terdekat jika ukuran sudah lebih besar dari bot lawan tersebut
Fungsi Seleksi	Memilih heading ke makanan terdekat saat ukuran lebih kecil dari ukuran bot lawan atau memilih heading ke bot lawan terdekat jika ukuran sudah lebih besar bot lawan tersebut
Fungsi Kelayakan	Memeriksa apakah bot benar mengarah ke makanan jika terlalu kecil dan mengarah ke lawan jika ukurannya sudah lebih besar

Fungsi Objektif	Bot dapat menentukan untuk menuju makanan terdekat ketika ukuran terlalu kecil dan menuju lawan jika ukuran sudah cukup besar sehingga meminimisasi peluang termakan oleh bot lain dan memaksimisasi penyerangan terhadap bot lain

6. *Greedy* by Distance with Attacking Feature

Selain dapat memakan *player* lain, bot juga dapat melakukan serangan dalam bentuk menembak. Bot dapat menembak torpedo, teleporter, dan supernova. Dengan alat-alat ini, Bot dapat menyerang dari jarak jauh dan mengurangi kesempatan bot termakan oleh *player* lain. Untuk saat ini, akan difokuskan ke penggunaan torpedo. Untuk melakukan ini, perlu dilakukan:

a. Peroleh list makanan dan *player*, kemudian diurutkan berdasarkan jarak dan disimpan pada list terpisah
b. Pada awal permainan, bot akan mencari makan hingga mencapai size tertentu
c. Setelah mencapai size tersebut, bot akan mengarah dan menembakkan torpedo ke arah lawan terdekat

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi *command* FORWARD dan FIRETORPEDOES serta heading menuju makanan dan bot lawan
Himpunan Solusi	Permutasi *command* FORWARD dan FIRETORPEDOES serta heading menuju makanan dan bot lawan terdekat sehingga dapat menembak bot lawan dan melakukan penyerangan jarak jauh
Fungsi Solusi	Memeriksa apakah *command* FIRETORPEDOES digunakan untuk menyerang dan mengarah ke bot lawan dengan jarak terdekat
Fungsi Seleksi	Memilih *command* berdasarkan *game state* sehingga bot dapat membesarkan diri dengan mengambil makanan dan menyerang menggunakan torpedo pada bot lawan terdekat
Fungsi Kelayakan	Memeriksa apakah penggunaan FIRETORPEDOES diarahkan kepada bot lawan dengan jarak terdekat
Fungsi Objektif	Bot dapat menuju ke makanan pada jarak terdekat dan meningkatkan ukuran dan dapat menembakkan torpedo ke bot lawan dengan jarak terdekat

7. *Greedy* by Distance with Attacking optimization

Beberapa optimisasi dapat dilakukan pada strategi penyerangan yang dilakukan pada bagian 6. Terdapat beberapa aksi lain yang dapat digunakan untuk mendekati *player* lain selain menggunakan aksi *FORWARD*. Beberapa aksi tersebut adalah penggunaan AfterBurner dan juga teleporter. Strategi ini dapat dilakukan dengan cara :

a. Peroleh data makanan dan *player* dan simpan pada sebuah list terurut meningkat berdasarkan jarak dari bot
b. Jika ukuran bot belum lebih besar dari suatu size, bot akan pergi mencari makanan
c. Setelah size dicapai, bot mengevaluasi jarak antara dirinya dan bot terdekat
    a) Jika terlalu jauh dan tidak terdapat teleporter, maka bot akan mendekatkan diri menggunakan afterburner
    b) Jika terdapat teleporter, maka bot akan menembakkan teleporter ke arah *player* dan mendetonasi pada jarak tertentu relatif terhadap size target (agar jika size target sangat besar tidak perlu terlalu dekat)
d. Setelah berada pada jarak yang relatif dekat dengan player, bot akan menembakkan torpedonya

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi *command* FORWARD, FIRETORPEDOES, FIRETELEPORTER, TELEPORT, STARTAFTERBURNER, dan STOPAFTERBURNER serta heading yang mengarah ke makanan valid atau bot lawan
Himpunan Solusi	Permutasi *command* yang memaksimisasi penyerangan terhadap player terdekat
Fungsi Solusi	Memeriksa apakah permutasi *command* berhasil membuat bot untuk menyerang bot lawan secara maksimal
Fungsi Seleksi	Memilih permutasi *command* dan heading yang membuat bot dapat memiliki ukuran lebih dari bot lawan terdekat dan melakukan penyerangan menggunakan *command* FIRETORPEDOES, FIRETELEPORTER dan TELEPORT, STARTAFTERBURNER dan STOPAFTERBURNER utuk meningkatkan kecepatan penyerangan terhadap bot lawan (mempercepat mendekati lawan)
Fungsi Kelayakan	Memastikan permutasi *command-command* sesuai dan heading yang dituju merupakan makanan atau bot lawan terdekat

Fungsi Objektif	Memaksimalkan penyerangan terhadap bot lain dengan meningkatkan efektivitas dari cara penyerangan (dengan mendekat dan menggunakan fitur lainnya)

8. *Greedy* by Distance with Better Attacking Optimization

Sebelumnya telah diterapkan cara menyerang dengan mendekati *player* lain terlebih dahulu untuk memaksimalkan penyerangan. Sistem tersebut dapat dioptimisasi lebih lanjut dengan peningkatan penggunaan teleporter. Cara yang dapat dilakukan adalah :

a. Peroleh semua data makanan (yang valid) dan *player* di map dan urutkan berdasarkan jarak terkecil, juga buat list *player* terurut berdasarkan ukurannya dari terkecil hingga terbesar.
b. Bandingkan ukuran bot dengan *player* terdekat dan jika bot size lawan lebih besar, maka :
    a) Ketika jarak terlalu dekat dan tidak memiliki torpedo, bot akan menghindari target
    b) Ketika jarak terlalu dekat dan memiliki torpedo, bot akan menembakkan torpedo ke arah lawan
    c) Ketika jarak sudah jauh, bot akan mencari makan
c. Ketika ukuran bot sudah lebih besar dari lawan terdekat maka :
    a) Jika bot memiliki teleporter, bot akan menembakkannya ke lawan
    b) Jika jarak terlalu jauh dan tidak memiliki teleporter,
        1) Jika bot tidak memiliki torpedo, bot akan mendekat
        2) Jika bot memiliki torpedo, bot akan menembak sambil mendekat
        3) Jika jarak bot terlalu jauh, bot akan mencari makanan
d. Ketika ada teleporter yang berada dekat dengan *player* terdekat dari bot, maka bot akan mengaktivasi teleport dan memakan *player* tersebut.

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi *command* FORWARD, FIRETORPEDOES, FIRETELEPORTER, TELEPORT, dan heading menuju makanan dan bot lawan
Himpunan Solusi	Permutasi *command-command* dan heading yang menuju makanan atau bot lawan dengan jarak terdekat dan melakukan penyerangan menggunakan torpedo

Fungsi Solusi	Memeriksa apakah makanan atau bot lawan yang dituju merupakan yang terdekat dari bot dan penggunaan torpedo dan teleporter juga diarahkan kepada bot lawan yang sama dan berhasil digunakan untuk menyerang
Fungsi Seleksi	Memilih heading ke makanan terdekat jika ukuran belum seuai dan memilih heading menuju bot lawan terdekat dan penggunaan afterburner saat jarak dari bot ke bot lawan terlalu jauh dan menggunakan torpedo saat sudah dekat
Fungsi Kelayakan	Memastikan *command-command* yang dilakukan diarahkan ke makanan atau bot lawan dengan jarak terdekat dan penggunaan teleporter atau afterburner dilakukan saat jarak terlalu jauh
Fungsi Objektif	Bot dapat memaksimisasi penyerangan terhadap bot lawan dengan menggunakan afterburner atau teleporter untuk mendekati bot lawan dengan lebih cepat dan penggunaan torpedo dapat maksimal (mengenai target dengan lebih mudah)

III. Analisis Efisiensi dan Efektivitas

1. Greedy by Distance on *food*

Dengan melakukan Greedy by Distance on Food, bot akan mengarah ke makanan terdekat dari dirinya sehingga *greedy* by distance on food bisa dilakukan. Akan tetapi, algoritma ini menjadi problematik pada beberapa kondisi.

Bot tidak mengetahui apakah makanan yang diincar tersebut baik untuk diambil atau tidak. Contohnya jika makanan berada di dekat musuh yang lebih besar ataupun makanan berada di sekamir gas cloud. Pada kondisi-kondisi tersebut, memungkinkan bot untuk berkurang size bahkan hingga mati karena *player* lain yang berada dekat makanan tersebut dan lebih besar.

2. Greedy by Distance on *Player*

Dengan melakukan Greedy by Distance on *Player*, bot akan selalu mencari bot lain yang terdekat dan pergi ke arahnya. Tidak jauh berbeda dengan mencari makanan terdekat, mencari *player* terdekat saja juga terdapat kekurangannya.

Bot hanya akan mengarah ke bot lain tanpa memperdulikan aspek lain seperti penghalang gas cloud ataupun size dari bot yang dikejarnya. Bot yang dikejar tersebut bisa saja memiliki ukuran yang lebih besar daripada dirinya sehingga dengan pergi ke arah bot yang dikejar saja akan berakhir dengan dirinya dimakan oleh bot tersebut. Bisa juga bot mengejar bot lain yang berada di daerah sekamir gas cloud. Walaupun bot memiliki size lebih

besar, dengan berjalan melalui gas cloud, bot akan berkurang ukuran dan bisa saja karena hal tersebut, bot lain menjadi lebih besar dan memakan bot yang mengejar.

3. **Greedy by Size on *Player***

   Ada beberapa masalah yang terdapat pada cara tersebut. Permasalahannya adalah ketika *player* dengan size terkecil jauh dari bot kami. Bot bisa saja terhalangi oleh objek-objek lain seperti gas cloud atau asteroid atau bisa saja teleporter bahkan *player*. Bisa saja ada *player* yang lebih besar di jalur ke arah bot yang kami tuju sehingga termakan di tengah perjalanan.

4. **Greedy by Distance on Food with Validation Optimization**

   Dengan menerapkan Langkah-langkah tersebut, bot kami dapat menentukan makanan yang tidak berpotensi mengurangi kesempatan untuk menang. Tapi walaupun seperti itu, cara ini hanya menugaskan bot untuk mencari makanan saja dan perlu optimisasi lain untuk memastikan kemenangan.

5. **Greedy by Distance on Food and *Player* + Size**

   Cara ini sudah lumayan membantu untuk mencegah persoalan yang kami peroleh dari cara-cara sebelumnya. Bot sudah dapat mengevaluasi ketika ia tidak bisa memakan *player* yang ditujunya. Akan tetapi, masih terdapat kekurangan dari strategi tersebut. Bot bisa saja mengarah ke makanan yang berada di dekat bot lain yang lebih besar daripada bot kami sehingga kami dapat termakan. Bisa juga bot yang dikejar berjarak jauh atau terdapat penghalang lain yang membuat bot kami mengecil atau ada 2 bot yang berada di satu garis lurus dan ketika bot kami akan memakan, bot lain yang mau memakan telah menjadi lebih besar sehingga bot kami ikut termakan. Masih banyak lagi kejadian lainnya yang dapat berubah secara tiba-tiba.

6. **Greedy by Distance with Attacking Feature**

   Dengan strategi tersebut, bot akan selalu menyerang hingga ia berada di bawah batas size yang ditentukan. Akan tetapi, cara menembak ini tidak mempertimbangkan apakah torpedo yang ditembak pasti mengenai target. *Player* terdekat yang diarahkan oleh bot bisa saja berada di jarak yang sangat jauh sehingga dengan menembak dari jarak tersebut, torpedo bisa saja tidak mengenai *player* yang dituju. Bisa juga *player* yang diserang memiliki size yang jauh lebih besar sehingga di jarak dekat bot akan termakan. Hal lain yang dapat terjadi adalah bot terus menyerang tanpa mengubah arah berjalannya sehingga bisa saja bot selalu menembak ke arah musuh tetapi bot itu sendiri berjalan ke arah luar map.

7. Greedy by Distance with Attacking Optimization

Dengan cara ini, kami telah mengeliminasi kemungkinan bahwa bot akan menembak terus selagi berjalan ke arah luar map dan juga mengeliminasi kemungkinan torpedo yang ditembakkan bot tidak mengenai player yang dituju.

8. Greedy by Distance with Attacking Optimization

Dengan cara ini, bot akan memanfaatkan informasi *player* terdekat dan melakukan perhitungan berdasarkan pertimbangannya menurut size, jarak dan kepemilikan alat menembak. Penggunaan teleporter juga dioptimisasi sehingga teleporter digunakan ke *player* terdekat ketika size bot dapat memakan *player* lawan tersebut sehingga teleporter digunakan untuk mempercepat penyerangan ke *player* lain.

IV. Strategi *Greedy* yang Dipilih

Setelah meninjau dan menganalisis dari bab sebelumnya, kami memilih untuk menggabungkan beberapa algoritma greedy di atas dengan tambahan fungsi solusi dengan prinsip heuristic atau berdasarkan pengamatan dan pengujian yang kami lakukan, untuk menjadikan pilihan algoritma-algoritma *greedy* yang kami pilih menjadi beberapa kondisi.

Konsep dari algoritma *greedy* yang kami pilih adalah penggabungan dari seluruh algoritma *greedy* yang sudah dipaparkan di atas dengan beberapa perubahan berdasarkan pendekatan *heuristic* kelompok kami. Secara sederhana, algoritma ini memaksimalkan penyerangan pada setiap kesempatan tergantung batasan-batasan jarak dan ukuran yang telah kami tentukan dan membuat fungsi-fungsi tambahan untuk menyeleksi target food dan player. Kami membagi dua kondisi dalam algoritma ini yaitu secara aktif melakukan penyerangan dan pasif mencari makanan.

Nama Elemen Greedy	**Definisi Elemen**
Himpunan Kandidat	Permutasi dari *command-command* yang digunakan oleh bot kami yaitu: - *FORWARD* - *STOP* - *FIRETORPEDOES* - *FIRESUPERNOVA* - *DETONATESUPERNOVA* - *FIRETELEPORT*

	- *TELEPORT* *command* aksi, terdapat heading
Himpunan Solusi	Kemungkinan dari semua *command* dan *heading* yang membuat bot kami bertahan sampai akhir, memiliki size yang optimum untuk bertahan sampai akhir, meluncurkan torpedo ke lawan dengan benar,
Fungsi Solusi	Pengecekan terhadap semua permutasi dari *command* dan *heading* yang membuat bot menjalankan aksi *command* dan heading dengan benar agar menjadi pemenang.
Fungsi Seleksi	Fungsi seleksi yaitu memilih aksi *command* dan *heading* berdasarkan keadaan game state saat tersebut serta fungsi heuristik dari tingkat prioritas *command* dan *heading* yang harus diikuti. Fungsi heuristik tersebut dilakukan dengan pengecekan apakah ada supernova pada bot dan jika terdapat supernova maka langsung diluncurkan ke arah player terdekat. Selanjutnya dikelompokkan berdasarkan kondisi game state saat itu apakah bot bersifat pasif dengan makan food atau aktif dengan meluncurkan torpedo, teleporter, atau mengejar musuh. Implementasi dari fungsi heuristik dilakukan dengan pembagian kondisi (*if – else condition*).
Fungsi Kelayakan	Fungsi kelayakan dilakukan dengan memeriksa apakah *command bot* dapat dilakukan atau tidak. Fungsi lain yaitu memeriksa apakah game objek *empty* atau tidak agar tidak terjadi *error*.
Fungsi Objektif	Mencari jarak antara bot ke objek lain yang paling minimum yaitu *player, food, superfood, supernova, gas cloud,* dan *teleporter*. Objektif kedua adalah mencari *command* dan *heading* yang membuat penyerangan terhadap lawan secara maksimal mengenai lawan baik dari torpedo dan teleporter.

Alasan kami memilih algoritma *greedy* ini adalah bentuk optimalisasi dari semua algoritma *greedy* yang telah dipaparkan sebelumnya. Berdasarkan dari pengujian beberapa algoritma yang sudah dijelaskan di atas, kami berusaha untuk membuat algoritma *greedy* dengan pendekatan *heuristic* yang menurut kami sudah optimal. Kami tidak menggunakan *after burner* karena sulit untuk dikendalikan dengan konsep greedy. Kami juga tidak menggunakan *shield* karena kami lebih aktif dalam penyerangan. *Shield* dapat digunakan jika kami menganalisis apakah torpedo tertentu sedang mengarah ke kami dan bukan torpedo kami sendiri.

Pertimbangan kami memilih algoritma ini berdasarkan pengujian alternatif solusi algoritma *greedy* pada poin III. Pada poin tersbut kami memulai menguji dari *greedy* yang paling sederhana yaitu *greedy* terhadap *food* dan *player* berdasarkan size dan jarak. Kemudian, kami lanjutkan dengan alternatif *greedy* yang lainnya. Pada algoritma final ini, kami melakukan optimalisasi dengan fungsi seleksi *food* dan *player* yang terdiri dari pengecekan apakah objek tersebut ada di dalam gas cloud dan apakah target ada di sekamir border. Selain itu, terdapat optimisasi dari mode aktif dan pasif tergantung kondisi jarak, ukuran, dan ketersediaan sesuatu.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

I. Implementasi Algoritma *Greedy*

1. Penjelasan dengan Bahasa
    a. List semua kebutuhan objek berdasarkan jenisnya kemudian diurutkan berdasarkan jarak terdekat dengan bot. Untuk smallestFood dan smallest*Player* diberikan fungsi seleksi tambahan yaitu validasi apakah food atau player tersebut berada dalam gas cloud dan apakah berada di dekat border dari permainan.
    b. Inisiasi awal dari batasan-batasan seperti jarak dan ukuran.
    c. Membagi 3 kondisi utama sebagai dasar utama sebagai fungsi solusi yang akan dipilih untuk aksi berikutnya yaitu:
        1) Firing supernova jika memilikinya
        2) Passive mode : bot bertindak lebih utama untuk mencari makanan
        3) Active mode : bot lebih aktif untuk menyerang ke lawan
    d. **Supernova** adalah langkah pertimbangan pertama yang dilakukan oleh bot. Jika bot memiliki supernova, ia akan menembakkannya ke arah *player* dengan jarak terdekat daripadanya.
    e. **Passive mode** merupakan kondisi saat ukuran dari bot lebih besar dari *player* lawan terdekat. Passive mode mempertimbangkan jarak yang terbagi menjadi :
        1) Jarak bot dengan lawan terlalu dekat, kondisi ini terbagi menjadi 2 kegiatan lainnya yaitu:
            a) Jika bot memiliki torpedo, bot akan menembakkan torpedo ke arah lawan sambil menjauh
            b) Jika tidak memiliki torpedo, bot hanya akan menjauh saja
        2) Jarak bot dengan lawan tidak terlalu dekat, bot akan mencari makanan
    f. **Active mode** merupakan kondisi saat ukuran dari bot lebih besar dari *player* lawan terdekat. Active mode mempertimbangkan banyak alat menyerang yang dimiliki oleh bot, yaitu:
        1) Jika bot memiliki peluru teleporter dan ukuran bot setelah menembakkan teleporter tidak lebih kecil dari *player* yang dituju, bot akan menembakkan teleporter ke arah lawan tersebut.
        2) Jika bot tidak memiliki teleporter tetapi memiliki torpedo dan jaraknya sudah sesuai, bot akan menembakkan torpedonya ke arah lawan
        3) Jika tidak memliki keduanya, maka:
            a) Jika jarak sudah dekat, maka bot akan mengejar lawan
            b) Jika jarak terlalu jauh, maka bot akan mencari makanan

g. Setelah melakukan evaluasi utama tersebut, bot akan mengevaluasi detonasi teleporter. Jika pemain terdekat memiliki jarak yang dekat dengan sebuah teleporter, maka bot akan melakukan detonasi pada teleporter (walaupun teleport yang dekat bukan bukan milik bot).

h. Bot kemudian mengevaluasi detonasi supernova. Jika supernova yang berada di map memiliki jarak yang dekat dengan *player* terdekat dari bot, bot akan meledakkan supernova.

i. Diluar semua kondisi tersebut, ketika objek lain selain *player* sudah tidak ada, bot akan mengarah ke tengah map.

2. Pseudocode Program

   a. Fungsi isNextTargetValid

```
function isNextTargetValid (input object : GameObject, input gasCloudList: 
List<GameObject>, input asteroidFieldList : List<GameObject>, input playerList  : 
List<GameObject>) -> boolean

{Fungsi ini merupakan fungsi seleksi dari objek yang sudah diurtukan berupa food 
atau player. Fungsi ini memberikan true jika objek tersebut valid dan false jika 
sebaliknya}


Deklarasi

distNear, degNear, distError : integer


Algoritma

For (elemen di dalam list gasCloud) do

   if(jarak dari object ke gas cloud + distError <=  size gas cloud)

      → false

   if( (dist<=distNear dan (heading object <= heading dari playerList[1] + 
    degNear)) dan  (heading object >= heading dari playerList[1] - degNear) )

       → false

   if (inBorder(object))

       → false

   → true
```

b. Fungsi inBorderValid

```
function (input object : GameObject)
{ Fungsi ini mengembalikan true jika sebuah objek ada di antara radius terluar  }

Deklarasi
inBorder : boolean
distToBorderMin, rad : double

Algoritma
if ((rad - (jarak object ke center map + size object)) <= distToBorderMin)
   inBorder ← true
else
   inBorder  ← false
→ inBorder
```

c. Prosedur computeNext*Player*Action

```
procedure (input playerAction  : PlayerAction)

{ Prosedur tempat algoritma greedy untuk memilih aksi dan heading dari player di
tick saat prosedur ini dipanggil }


Deklarasi

playerList, playerListBySize, mostOftenTarget, wormholeList, gasCloudList,
supernovaBombList, teleporterList : variabel untuk list sekumpulan GameObject

safeDistancePlayer, teleporterAttackThresholdSize, torpedoThresholdSize,
passiveThresholdSize : integer

torpedoAttackThresholdDistance, supernovaDetonateDistance,
teleporterDistanceThreshold : double


Algoritma

{ inisiasi list GameObject }

playerList ← mengurutkan player berdasarkan jarak terhadap bot kami

mostOftenTarget ← mengurutkan food, superfood, dan supernove berdasarkan jarak
terhadap bot kami

wormholeList ← mengurutkan wormhole berdasarkan jarak terhadap bot kami

gasCloudList ← mengurutkan gas cloud berdasarkan jarak terhadap bot kami

supernovaBombList ← mengurutkan supernova pada map berdasarkan jarak terhadap
bot kami

teleporterList ← mengurutkan teleporter pada map berdasarkan jarak terhadap bot
kami


safeDistancePlayer ← 100 + nearestPlayer.getSize();

teleporterAttackThresholdSize ← nearestPlayer.getSize() + 30;

torpedoThresholdSize ← 25;

torpedoAttackThresholdDistance ←  300 + nearestPlayer.getSize();

supernovaDetonateDistance ←  nearestPlayer.getSize() + 20;

teleporterDistanceThreshold ← bot.size * 0.7;

passiveThresholdSize ←  nearestPlayer.getSize();
```

```
if (game object pada peta masih ada)


    {Target dari bot adalah FOOD, SUPERFOOD, SUPERNOVAPICKUP, dan player musuh}

    if (target pada peta masih ada)


        {List index}

        collectibleIndex <- 0 {collectibleIndex adalah index untuk list
mostOftenTarget}

        nearestPlayerIndex <- 1 {index nearestPlayer di set menjadi 1 karena index
0 adalah bot sendiri}


        {Target yang mungkin}

        foodTarget <- mostOftenTarget[collectibleIndex]

        nearestPlayer <- playerList[nearestPlayerIndex]


        {Validasi target makanan}

        while (not isNextTargetValid(foodTarget, gasCloudList, playerList)) do

            if (collectibleIndex < ukuran list mostOftenTarget)

                collectibleIndex <- collectibleIndex + 1

            else

                foodTarget = mostOftenTarget[0]

                collectibleIndex = 0

                break


        {Validasi target musuh terdekat}

        while (not isNextTargetValid(nearestPlayer, gasCloudList, playerList)) do

            if (nearestPlayerIndex < ukuran list playerList)

                nearestPlayerIndex <- nearestPlayerIndex + 1

            else

                nearestPlayer = mostOftenTarget[0]

                nearestPlayerIndex = 0

                break


        nearestPlayer <- playerList[nearestPlayerIndex]
```

```
        {Thresholds}
        safeDistancePlayer <- 200 + nearestPlayer.getSize()
        teleporterAttackThresholdSize <- nearestPlayer.getSize() + 30

        torpedoThresholdSize <- 25
        torpedoAttackThresholdDistance <- 300 + nearestPlayer.getSize()

        supernovaDetonateDistance <- nearestPlayer.getSize() + 20
        teleporterDistanceThreshold <- bot.size * 0.7

        passiveThresholdSize <- nearestPlayer.getSize() + 10

        distNearestPlayer <- jarak antara bot dan nearestPlayer

        {jarak dari ujung bot sendiri ke ujung bot musuh}
        distNearestPlayerFixed <- distNearestPlayer - nearestPlayer.size -
bot.size

        {Decision making}
        if (bot.supernovaAvailable > 0) {Jika bot memiliki supernova, bot akan
langsung menembakkan ke player terdekat}
            heading bot <- heading ke arah musuh terdekat
            action bot <- FIRESUPERNOVA
        else if (inBorderValid(bot)) {Jika bot berada di dekat border, bot
bergerak ke pusat map}
            action bot <- FORWARD
            heading bot <- heading dari bot ke pusat map

        else if (bot.size < passiveThresholdSize) {Passive mode}

            if (distNearestPlayerFixed <= safeDistancePlayer) {Mekanisme
counter/kabur}

                heading bot <- heading dari bot ke musuh terdekat
```

```
                {Jika bot memiliki 2 atau lebih torpedo dan size bot lebih dari
threshold, tembak torpedo}
                if (bot.torpedoCount >= 2 and bot.size > torpedoThresholdSize)
                    heading bot <- heading dari bot ke musuh terdekat
                    action bot <- FIRETORPEDOES
                else {}
                    heading bot <- (heading dari bot ke musuh terdekat - 180) mod
360
                    action bot <- FORWARD


            else {cari makan}
                heading bot <- heading dari bot ke foodTarget
                action bot <- FORWARD


        else {active mode}


            action bot <- FORWARD
            heading bot <- heading dari bot ke musuh terdekat
            {Jika bot memiliki teleporter dan size bot melebihi threshold, tembak
teleporter ke arah musuh}
            if (bot.teleCount > 0 and bot.size >= teleporterAttackThresholdSize)
                heading bot <- heading dari bot ke musuh terdekat
                action bot <- FIRETELEPORT


            {Jika bot tidak memiliki teleporter, memiiki torpedo, dan jarak ke
musuh terdekat memenuhi threshold, maka tembak musuh}
            else if (bot.torpedoCount >= 2 and distNearestPlayerFixed <=
torpedoAttackThresholdDistance)
                action bot <- FIRETORPEDOES
            else {Jika tidak memiliki torpedo sama teleporter atau tidak memenuhi
threshold}


                if (distNearestPlayerFixed <= torpedoAttackThresholdDistance)
{Kejar musuh jika memenuhi threshold}
                    action bot <- FORWARD
                else {Jika musuh terlalu jauh, cari makanan}
                    action bot <- FORWARD
                    heading bot <- heading dari bot ke foodTarget
```

```
            {Jika pada map terdapat teleporter yang sudah ditembakkan}


            if (teleporterList tidak kosong)


                idx <- nearestPlayerIndex

                teleListByTargetDist <- mengurutkan list teleporter berdasarkan
jarak ke bot musuh terdekat


                {cek apakah list kosong untuk menghindari error out of bounds}

                if (teleListByTargetDist tidak kosong)

                    {tembak teleporter jika jarak teleporter ke bot musuh
terdekat memenuhi threshold}


                    if (jarak teleporter ke nearestPlayer - nearestPlayer.size <
teleporterDistanceThreshold
                        && bot.size > nearestPlayer.size + 5)

                        action bot <- TELEPORT


        {Jika ada bomb supernova pada map yang sudah ditembakkan}

        if (supernovaBombList tidak kosong)

            {Detonate supernova ketika jarak supernova dengan player musuh
terdekat memenuhi threshold}

            if (jarak antara bomb supernova dengan bot musuh terdekat <
supernovaDetonateDistance)

                action bot <- DETONATESUPERNOVA


    else {Jika target tidak ada, bergerak ke pusat map}

        action bot <- FORWARD

        heading bot <- heading dari bot ke pusat map
```

II. Penjelasan Struktur Data

1. Struktur Data yang Digunakan dalam Program Bot Galaxio

   Struktur data yang digunakan dalam permainan ini berbentuk *class. Class* tersebut dapat dibagai menjadi 6 kelas utama yang terdapat dalam *package* Models. Terdapat 3 *package* utama dalam program ini yaitu:

   a. *Package* Models berisikan kelas-kelas:

      1) *Game*Object, berisi semua objek yang terdapat di dalam permainan. Kumpulan object types ini terletak pada bagian Enums di dalam public enum ObjectTypes.

Atribut	**Deskripsi**
UUID id	Menunjukkan id dari *Game*Object bertipe UUID
Integer size	Menunjukkan size dari *Game*Object yaitu radius objek
Integer currentHeading	Menunjukkan heading saat ini yang dituju oleh objek dalam satuan derajat
Position position	Menunjukkan posisi dari objek dari kelas position dengan x dan y pada sumbu kartesian
ObjectTypes *game*ObjectType	Menunjukkan tipe objek yaitu *player*, food, wormhole, gas cloud, asteroid field, torpedo salvo, superfood, superova pickup, supernova bomb, teleporter, dan shield
Integer effects	Menunjukkan efek yang diterima oleh objek dengan kumulatif bit flag: 1. 0 = No effect 2. 1 = Afterburner active 3. 2 = Asteriod Field 4. 4 = Gas cloud
Integer torpedoCount	Menunjukkan jumlah torpedo yang dimiliki oleh objek
Integer supernovaAvailable	Menunjukkan apakah objek tersebut memiliki supernova atau tidak
Integer teleCount	Menunjukkan jumlah teleporter yang dimiliki oleh objek
Integer shieldCount	Menunjukkan jumlah shield yang dimiliki oleh objek

2) *Game*State

Atribut	Deskripsi
World world	Objek world dari kelas World
List<*Game*Object> *game*Objects	List objek-objek dalam *game* dari kelas *game* objek
List<*Game*Object> *player*Game*Objects	List objek *player* dari kelas *game* objek

3) *Game*StateDto

Atribut	Deskripsi
World world	Objek world dari kelas World
Map<String, List<Integer>> *game*Objects	Tuple objek-objek dalam *game* berisi string dan list integer dari kelas Map
Map<String, List<Integer>> *player*Objects	Tuple objek-objek *player* dalam *game* berisi string dan list integer dari kelas Map

4) *Player*Action

Atribut	Deskripsi
UUID *player*Id	ID dari *Player* bertipe UUID
*Player*Actions action	Aksi dari *player* dari Enum *Player*Actions
int heading	Arah dari *player* bertipe integer dalam derajat

5) Position

Atribut	Deskripsi
int x	Nilai koordinat x dari objek dalam sumbu horizontal diagram kartesian
int y	Nilai koordinat y dari objek dalam sumbu vertikal diagram kartesian

6) World

Atribut	Deskripsi

Position centerPoint	Posisi dari pusat "*world*" dalam *game* dari kelas Position yang berisi x dan y
Integer radius	Besar radius dari "*world*"
Integer currentTick	Menandakan *tick* saat ini

b. *Package* Enums berisikan public Enum ObjectTypes dan public Enum *Player*Actions
Package ini berisikan enumerasi dari aksi yang dapat dilakukan objek *player* dan juga objek-objek yang terdapat di map. Kedua file tersebut yang menjadi sumber informasi untuk memperoleh seluruh objek dan juga digunakan dalam mengubah aksi dari bot.

1) ObjectTypes.java

Enums	Value
PLAYER	1
FOOD	2
WORMHOLE	3
GASCLOUD	4
ASTEROIDFIELD	5
TORPEDOSALVO	6
SUPERFOOD	7
SUPERNOVAPICKUP	8
SUPERNOVABOMB	9
TELEPORTER	10
SHIELD	11

Atribut	Deskripsi
Public integer value	Menandakan jenis objek

Methods	Deskripsi
ObjectTypes(integer value)	Menginisiasi jenis dari objek berdasarkan value
Public static ObjectTypes valueOf(integer value)	Mengembalikan jenis dari suatu objek

2) *Player*Actions.java

Enums	Value
FORWARD	1
STOP	2
STARTAFTERBURNER	3
STOPAFTERBURNER	4
FIRETORPEDOES	5
FIRESUPERNOVA	6
DETONATESUPERNOVA	7
FIRETELEPORT	8
TELEPORT	9
ACTIVATESHIELD	10

Atribut	Deskripsi
public int value	Menandakan aksi yang dilakukan *player* dari value yang diberikan

Methods	Deskripsi
private *Player*Action(integer value)	Mengubah aksi yang dilakukan *player* berdasarkan value yang diberikan

c. *Package* Services yang berisikan BotService yakni tempat implementasi algoritma *greedy*.

Atribut	Deskripsi
private GameObject bot	Merepresentasikan bot
private PlayerAction playerAction	Merupakan aksi dari bot
private GameState gameState	Merepresentasikan kondisi game saat itu

Method	Deskripsi
public BotService()	Menciptakan objek botservice
public GameObject getBot()	Mengembalikan bot yang sedang digunakan

public void setBot(GameObject bot)	Mengganti bot pada botservice dengan bot yang diberikan
public PlayerAction getPlayerAction()	Mengembalikan aksi dari bot
public void setPlayerAction(PlayerAction playerAction)	Mengubah aksi dari bot pada botservice
private boolean isNextTargetValid(GameObject object, list<GameObject> gasCloudList, List<GameObject> playerList)	Menentukan apakah sebuah objek berada di sekamir gas cloud atau dekat player. Jika ya maka akan mengembalikan false jika tidak akan mengembalikan true
private boolean inBorderValid(GameObject object)	Mengembalikan true jika objek berada di sekamir border dan false jika tidak
Public void computeNextPlayerAction(PlayerAction playerAction)	Fungsi utama yang menentukan aksi apa yang akan dilakukan oleh bot pada tiap tick perubahan *game state*
public GameState getGameState()	Mengembalikan *game state* dari botservice saat ini
public void setGameState(GameState gameState)	Merubah *game state* pada botservice
public void updateSelfState()	Memperbaharui state dari bot saat ini
private double getDistanceBetween(GameObject object1, GameObject object2)	Mengembalikan jarak dari objek 1 dan objek 2
Private double getDistancePosition(GameObject object1, int x, int y)	Mengembalikan jarak objek ke titik (x,y)
private int getHeadingPosition(GameObject object, int x, int y)	mengembalikan arah dari objek ke titik (x,y)
private int getHeadingBetween(GameObject otherObject)	Mengembalikan arah dari bot ke objek yang ditentukan
private int toDegrees(double v)	Mengembalikan derajat dari value v yang diberikan

III. Analisis Desain Solusi Algoritma *Greedy*

1. Analisis Kondisi Optimal dari Algoritma

Algoritma greedy yang telah didesain memiliki performa yang optimal ketika bot berhasil memiliki ukuran yang cukup besar pada awal permainan dengan cara mencari makanan tanpa intervensi dari bot lain. Dengan kondisi awal demikian, bot akan leluasa dalam pemakaian teleporter untuk menyerang musuh terdekat dan langsung memakannya. Kondisi tersebut berlaku lebih optimal ketika bot musuh terdekat memiliki ukuran yang jauh lebih kecil dibandingkan dengan bot kami, karena dengan kondisi tersebut, kegagalan akibat perubahan ukuran bot musuh terdekat menjadi lebih besar dibandingkan bot kami memiliki kemungkinan yang sangat kecil. Selain perbedaan ukuran bot yang besar, jarak yang tidak terlalu jauh antara bot kami dengan bot musuh terdekat juga membuat algoritma yang telah didesain bekerja secara optimal karena kemungkinan teleporter tepat sasaran dengan target cukup besar.

Algoritma greedy yang telah dibuat juga cukup optimal dalam menghadapi bot lawan yang memiliki ukuran yang mirip dengan bot kami. Hal tersebut ditangani dengan menembakkan torpedo diikuti dengan penembakan teleporter. Penembakan torpedo akan membuat bot musuh menjadi lebih kecil dari sebelumnya dan membuat bot kami menjadi lebih besar dari sebelumnya. Setelah itu, teleporter akan diaktivasi oleh bot kami untuk langsung memakan bot musuh tersebut.

Algoritma kami juga menambahkan *threshold* jarak dalam memakai torpedo, sehingga ketika bot melakukan aksi penyerangan, torpedo dapat digunakan secara efektif dan efisien. Hal tersebut menyebabkan *size* bot kami dapat meningkat secara drastis karena torpedo yang mengenai musuh secara tepat. Kondisi tersebut dapat dilanjutkan dengan penembakan teleporter ke arah musuh, kemudian teleporter akan langsung diaktivasi setelah jarak teleporter cukup dekat dengan musuh.

Bot kami juga dapat mengimplementasikan bagian validasi target dengan baik, sehingga bot dapat menambah ukuran dengan memakan makanan tanpa mengorbankan *size* yang hilang akibat dari gas cloud ataupun karena *border* peta. Hal tersebut menyebabkan bot dapat menambah *size* dengan cepat dan optimal sehingga dapat segera meluncurkan penyerangan menggunakan teleporter dan torpedo.

Jumlah teleporter yang dimiliki oleh bot juga sangat mempengaruhi keoptimalan dari algoritma kami. Algoritma akan bekerja secara optimal jika bot memiliki tepat satu teleporter ketika akan menyerang lawan, sehingga tidak ada pemborosan penggunaan *size* bot dalam melakukan penyerangan.

2. Analisis Kondisi Kurang Optimal dari Algoritma

Kegagalan yang paling besar dalam algoritma kami adalah ketika bot kami menyimpan lebih dari satu teleporter. Pada kasus tersebut, terkadang bot akan langsung menembakkan lebih dari satu teleporter ke arah yang sama. Hal tersebut menyebabkan penggunaan *size* bot yang terlalu banyak yang seharusnya tidak perlu, sehingga dapat menyebabkan penyerangan tidak terjadi dilakukan, atau bahkan dapat memutarbalikkan kondisi menjadi sangat merugikan.

Kondisi lain yang tidak dapat diatasi dengan baik dengan algoritma yang telah ada yaitu ketika bot musuh yang ukurannya jauh lebih besar daripada bot kami menembakkan teleporter ke arah bot kami. Kondisi tersebut tidak di-*handle* oleh algoritma kami karena tidak ada cara untuk membedakan penembak suatu teleporter tertentu tanpa memanfaatkan *state* sebelumnya. Sehingga, bot kami tidak akan menghindar dari teleporter yang ditembakkan oleh musuh. Karena ukuran musuh jauh lebih besar dibandingkan dengan bot kami, bot kami akan langsung mengalami kekalahan karena akan langsung dimakan oleh bot musuh ketika bot musuh mengaktivasi teleporter.

Tidak adanya algoritma untuk membedakan teleporter musuh dan teleporter kami juga dapat mengacaukan aksi penyerangan. Ketika teleporter milik musuh mendekat bot musuh yang menjadi target kami, algoritma akan menganggap bahwa teleporter tersebut adalah teleporter yang sudah ditembakkan oleh bot kami. Hal tersebut menyebabkan bot kami mengaktivasi teleporter pada waktu yang salah, sehingga bot kami malah *teleport* ke lokasi yang tidak seharusnya.

Aksi penyerangan dengan teleporter juga dapat gagal ketika jarak antara bot kami dengan bot musuh terdekat cukup jauh, karena bot musuh target kemungkinan besar sudah keluar dari lintasan dari teleporter yang telah ditembakkan. Hal tersebut membuat penggunaan *size* bot menjadi tidak optimal.

Keadaan lain yang dapat menyebabkan penyerangan tidak optimal adalah ketika jarak bot kami dengan bot musuh terdekat mirip dengan jarak bot kami ke bot lainnya. Hal tersebut dapat menyebabkan perubahan objek *nearestPlayer* selama teleporter tersebut masih berjalan, sehingga teleporter yang telah ditembakkan tidak bisa diaktivasi oleh player.

Bot kami juga mengalami kegagalan dalam menghadapi bot musuh dari beberapa arah yang berbeda sekaligus. Algoritma kami didesain supaya bot kami bisa kabur dari musuh dengan arah yang berlawanan/sesuai dengan arah bot yang mengincar kami. Ketika bot kami diincar lebih dari satu arah, bot kami akan bergerak ke arah yang juga merupakan arah yang tidak aman.

Selain itu, ketidakoptimalan pada bot kami adalah ketika terdapat dua atau lebih *food* dengan jarak yang sama ke bot kami. Kondisi tersebut membuat bot kami melakukan perubahan *heading* diantara kedua *food* tersebut sehingga bot hanya akan berhenti berosilasi ketika salah

satu dari *food* tersebut hancur, termakan oleh player lain, atau didekati oleh musuh yang dapat mengubah kondisi pasif/aktif dari bot kami.

BAB 5
KESIMPULAN DAN SARAN

I. Kesimpulan

Kelompok kami telah mengimplementasikan algoritma *greedy* untuk membuat bot yang dapat memaksimisasi penyerangan untuk memperoleh kemenangan. Dikarenakan permainan Galaxio berfokus kepada sistem *last-man-standing wins*, maka kami rasa algoritma *greedy* dalam penyerangan merupakan algoritma yang cukup optimal untuk memenangkan permainan.

Penentuan kebiasaan pada bot dapat ditentukan dengan pendekatan secara heuristik agar lebih mudah dimodifikasi pada saat pengembangan seusai dengan hasil yang diperoleh dari percobaan sebelumnya. Pendekatan ini membantu dalam penentuan aksi yang optimal yang dilakukan oleh bot sehingga dapat memperoleh kemenangannya.

II. Saran

Pada tugas besar 1 Strategi Algoritma terkait penerapan algoritma *greedy*, terdapat beberapa saran yang bisa kami ajukan untuk kedepannya:

1. Sebaiknya pada awal pemberian tugas, terdapat pemahaman bersama terkait tugas besar ini agar tidak terjadi kesalahpahaman untuk selanjutnya. Setelah itu, harus ada pembagain tugas terlebih dahulu dengan jelas
2. Poin penting berikutnya adalah, laporan dapat dikerjakan berbarengan dengan *progress* dari kode program dan alternatif algoritma *greedy* karena dalam penerapannya, kami harus menganalisis dan memahami kelebihan dan kekurangan dari masing-masing algoritma
3. Pengujian antar bot seharusnya tidak random posisi dari objek-objeknya agar permainan antar bot dapat terlaksana dengan adil

DAFTAR PUSTAKA

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-*Greedy*-(2021)-Bag1.pdf

2. https://www.geeksforgeeks.org/*greedy*-algorithms/

3. https://www.includehelp.com/icp/*greedy*-strategy-to-solve-major-algorithm-problems

Link *Repository* github

https://github.com/NerbFox/Tubes1_PlayerNumberOne

Link Video YouTube

https://youtu.be/8BNMmrSmX44