LAPORAN TUGAS BESAR
IF2211 - STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Galaxio"

Kelas Mahasiswa K03

Dosen Pengampu: Ir. Rila Mandala, M.Eng., Ph.D.

Disusun Oleh:

Kelompok 19 - Artificial Ilham

Angger Ilham Amanullah	**13521001**
Ditra Rizqa Amadia	**13521019**
Muhammad Haidar Akita Tresnadi	**13521025**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Tugas pertama dalam Strategi Algoritma ini meminta mahasiswa untuk mengembangkan bot kapal dalam game *Galaxio* yang berfungsi untuk bertanding dengan bot kapal dari pemain lain. Tujuan utama permainan adalah untuk menjaga agar bot kapal mahasiswa tetap bertahan hidup hingga akhir permainan. *Game engine* dapat diperoleh pada laman berikut :

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

Untuk dapat memenangkan permainan, mahasiswa diminta untuk mengimplementasikan strategi *greedy* pada bot kapal yang dikembangkan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* yang ada di dalam *starter-pack* pada laman berikut ini :

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

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

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

10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.

3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan oleh kapal player. Apabila player mengkonsumsi Food, maka Player akan bertambah ukuran yang sama dengan Food. Food memiliki peluang untuk berubah menjadi Super Food. Apabila Super Food dikonsumsi maka setiap makan Food, efeknya akan 2 kali dari Food yang dikonsumsi. Efek dari Super Food bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player

tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.

9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Berikut jenis-jenis dari command yang ada dalam permainan:
    a. FORWARD
    b. STOP
    c. START_AFTERBURNER
    d. STOP_AFTERBURNER
    e. FIRE_TORPEDOES
    f. FIRE_SUPERNOVA
    g. DETONATE_SUPERNOVA
    h. FIRE_TELEPORTER
    i. TELEPORTUSE_SHIELD

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

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma *greedy* adalah salah satu strategi algoritma yang bertujuan memilih tindakan paling optimal pada setiap langkahnya dengan tujuan mencapai solusi optimal. Secara umum, algoritma ini bekerja dengan melakukan pemilihan terbaik pada setiap langkahnya tanpa mempertimbangkan konsekuensinya di masa depan.

Ide yang terdapat pada algoritma *greedy* adalah mencapai solusi optimal dengan cara mengambil keputusan terbaik pada setiap langkahnya dengan harapan keputusan - keputusan yang diambil akan mengarah kepada solusi paling optimal. Pada setiap langkah yang diambilnya, algoritma ini akan menimbang keuntungan atau nilai fungsional terbaik yang diberikan dan mengambilnya.

Meskipun begitu, terkadang keputusan optimal pada setiap langkah tidak mengarah kepada solusi paling optimal. Oleh karena itu, dibutuhkannya analisis matematis dan pengujian untuk memastikan bahwa algoritma ini memberikan solusi yang optimal.

Terdapat beberapa elemen atau kompenen yang perlu didefinisikan di dalam algoritma greedy. Beberapa elemen atau komponen algoritma greedy tersebut adalah:

1. Himpunan Kandidat : Merupakan kumpulan opsi atau kandidat yang dapat dipilih pada setiap langkah.
2. Himpunan Solusi : Kumpulan opsi atau kandidat yang telah dipilih sebagai solusi.
3. Fungsi solusi: Menentukan apakah kumpulan dari solusi yang telah dikumpulkan sebelumnya dapat memberikan solusi atau tidak.
4. Fungsi seleksi: Memilih solusi berdasarkan suatu strategi greedy tertentu. Fungsi ini bersifat heuristik, yang berarti dirancang untuk mencari solusi yang paling optimum secara praktis, meskipun mungkin tidak secara matematis.
5. Fungsi kelayakan: Mengevaluasi apakah opsi yang dipilih oleh fungsi seleksi dapat dimasukkan ke dalam kumpulan solusi.
6. Fungsi objektif: Maksimalkan atau minimalisasi suatu parameter pada masalah yang diberikan.

Berdasarkan elemen dan komponen di atas, algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C, yang memenuhi kriteria tertentu. Dalam hal ini, S harus memenuhi dua kriteria:

(1) S harus merupakan suatu solusi, artinya S harus memenuhi persyaratan atau batasan yang telah ditentukan dalam persoalan yang sedang diselesaikan;

(2) S harus dioptimalkan oleh fungsi objektif, yang berarti S harus menghasilkan nilai terbaik dari semua solusi yang mungkin. Proses seleksi opsi pada algoritma greedy didasarkan pada strategi

heuristik tertentu, dan setiap opsi yang dipilih harus memenuhi kriteria kelayakan. Algoritma greedy dapat diimplementasikan untuk berbagai jenis persoalan yang membutuhkan pencarian solusi secara optimal dan efisien.

2.2 Garis Besar Cara Kerja Program

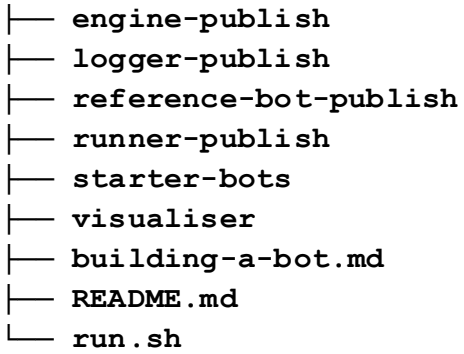
Galaxio merupakan permainan dengan jenis *turn-based strategy* berupa simulasi. Permainan ini berjalan dengan program berisi logika yang telah dibuat oleh para pemain dengan tujuan untuk tetap bertahan hingga akhir. Setiap pemain mengirimkan perintah mereka berupa program melalui *Application Programming Interface* (API). Secara umum, program pada Galaxio bekerja dengan cara berikut :

1. Program menerima program yang berisi tindakan yang diinginkan oleh pemain melalui API, seperti bertahan maupun menyerang lawan.
2. Setelah mendapat perintah, program akan memperbarui semua status permainan sesuai dengan program tersebut.
3. Program lalu mengambil keputusan yang tepat berdasarkan kondisi *bot* pemain dan juga perintah yang sudah dibuat.
4. Setelah mengambil keputusan, program akan mengirimkan respons ke pemain melalui API. Respons yang dikirim berupa tindakan yang telah dipilih oleh program. Setelah mengirim respons ini, program akan menunggu input dari pemain lain hingga permainan selesai.

Sebelum menjalankan program, diperlukan requirement atau prasayarat yang perlu diinstal terlebih dahulu yaitu

- JDK
- Apache Maven
- Net Core versi 3.1 dan 5.0

Berikut ini merupakan gambaran struktur dari folder yang dibutuhkan untuk menjalankan permainan galaxio



Untuk memahami cara kerja game galaxio, ada beberapa komponen penting yang perlu dipahami. Komponen-komponen tersebut meliputi engine, runner, dan logger.

- Engine berfungsi untuk menerapkan logika dan aturan-aturan game.
- Runner bertugas untuk mengatur jalannya pertandingan dan menghubungkan bot dengan engine.
- Logger berguna untuk mencatat log permainan, yang akan menjadi input bagi visualizer dan memberikan informasi mengenai hasil permainan.

Program game Galaxio bekerja dengan beberapa tahapan yang perlu dijalankan untuk memulai pertandingan.

Pertama, Runner akan meng-host sebuah match pada sebuah hostname tertentu dan menghubungkan ke Engine. Runner juga akan melakukan koneksi dengan Logger.

Kedua, Engine akan menunggu sampai semua bot pemain terkoneksi ke Runner untuk memulai permainan. Jumlah bot dalam satu pertandingan diatur pada atribut BotCount dalam file JSON "appsettings.json". File tersebut terdapat di dalam folder "runner-publish" dan "engine-publish". Setelah bot terkoneksi, permainan akan dimulai sesuai dengan konfigurasi.

Ketiga, bot akan mendengarkan event dari Runner, terutama event RecieveGameState karena memberikan informasi status permainan. Bot juga mengirim event kepada Runner yang berisi aksi bot. Permainan akan berlangsung sampai selesai. Setelah selesai, dua file JSON akan dibuat yang berisi kronologi pertandingan. Hal ini dapat membantu analisis dan evaluasi hasil permainan. Selanjutnya untuk menjalankan game Galaxio secara lokal di Windows, dapat dilakukan dengan mengikuti langkah-langkah berikut:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON "appsettings.json" dalam folder "runner-publish" dan "engine-publish". Sesuaikan dengan jumlah bot yang ingin Anda mainkan.
2. Buka terminal baru pada folder "runner-publish".Lalu jalankan runner menggunakan perintah "dotnet GameRunner.dll". Pastikan untuk menunggu hingga runner terkoneksi dengan engine dan logger sebelum melanjutkan.
3. Kemudian Buka terminal baru pada folder "engine-publish" dan jalankan engine menggunakan perintah "dotnet Engine.dll". Pastikan engine terkoneksi dengan runner sebelum melanjutkan.
4. Buka terminal baru pada folder "logger-publish". Jalankan logger menggunakan perintah "dotnet Logger.dll". Pastikan logger terkoneksi dengan runner sebelum melanjutkan.
5. Jalankan seluruh bot yang ingin dimainkan pada terminal terpisah. Pastikan bot terkoneksi dengan runner sebelum melanjutkan.
6. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON "GameStateLog_{Timestamp}" dalam folder "logger-publish". Kedua file tersebut berisi informasi mengenai hasil akhir permainan dan proses yang terjadi selama permainan berlangsung.

Dengan mengikuti langkah-langkah tersebut, Anda dapat menjalankan game Galaxio secara lokal di Windows dan menikmati permainan dengan jumlah bot yang telah dikonfigurasi sebelumnya.

2.3 Garis Besar Penerapan Algoritma Greedy Pada Bot Galaxio

Algoritma *greedy* digunakan dalam proses pembuatan program perintah yang akan digunakan dalam permainan ini karena dengan menggunakan Algoritma ini pemain bisa menentukan tujuan dan strategi yang ingin dicapai oleh *bot*. Tujuan dan strategi mungkin seperti menyerang dan bertahan pada situasi tertentu. Pemain dapat membuat program mengevaluasi kondisi setiap *tick* -nya. Manfaatkan itu untuk menentukan aksi terbaik pada setiap *tick* -nya. Dalam hal ini, kemungkinan besar setelah satu percobaan hasil yang didapatkan kurang memuaskan. Oleh karena itu, lakukanlah *trial and error* hingga mendapat hasil yang paling baik.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Alternatif Solusi Greedy

3.1.1 Greedy with Attacking

Greedy with attacking adalah pendekatan algoritma *greedy* yang memprioritaskan penyerangan kepada bot lawan sebagai pendekatannya. Serangan yang berhasil dieksekusikan ke lawan akan membuat bot kita mendapatkan tambahan *size* sehingga memperbesar kemungkinan kemenangan. Beberapa penyerangan yang dapat dilakukan yaitu memakan bot lawan, menembakinya menggunakan torpedo, menggunakan *teleport* untuk memakan bot lawan, dan lain lain.

a. Mapping Elemen Greedy
   i. Himpunan kandidat : Semua *command* yang berhubungan dengan *Attacking.*
   ii. Himpunan solusi : *Command* yang dipilih.
   iii. Fungsi solusi : Memeriksa apakah command yang dipilih memenuhi kondisi yang telah ditentukan
   iv. Fungsi seleksi : Pilih *command* terurut berdasarkan prioritas yang telah dibuat.
   v. Fungsi kelayakan : Memeriksa apakah *command* yang dipilih dapat dijalankan
   vi. Fungsi objektif : Menyerang lawan dan mendapatkan *size* lebih banyak

b. Analisis Efisiensi dan Efektivitas Solusi

   Terdapat tiga pilihan utama yang digunakan dalam *greedy* ini, yaitu penggunaan *teleport,* penggunaan *torpedo*, dan juga mengejar musuh yang memiliki *size* lebih kecil sehingga kita bisa memakannya.

   Greedy with attacking ini bisa menjadi salah satu strategi bagus karena jika berhasil akan menambah *size* bot yang memperbesar kemungkinan kemenangan. Namun, bisa jadi solusi ini bukanlah yang paling efektif.

   Strategi ini efektif apabila musuh yang ditarget dengan serangan torpedo sedang berada dalam *cooldown* aktivasi *shield* dan lokasi *spawn* player terdapat banyak *food* maupun *superfood* sehingga bisa melakukan penyerangan berupa *teleport* terlebih dahulu. Strategi ini tidak efektif ketika setelah penembakan *teleport* bot musuh melakukan satu dan lain hal yang membuatnya memiliki *size* lebih besar dibandingkan bot kita sehingga membatalkan aktivasi *teleport* dan membuat kita kehilangan *size* sia - sia.

3.1.2 Greedy with Defending

Greedy with defending adalah pendekatan algoritma *greedy* yang memprioritaskan bertahan dari serangan bot lawan sebagai pendekatannya. Bertahan dari serangan lawan dapat membuat bot tidak kehilangan *size* miliknya bahkan bisa membantu bertahan hingga akhir game. Beberapa aksi bertahan yang dapat dilakukan yaitu menjauh dari bot lawan dengan *size* yang lebih besar, aktivasi shield, dan lain lain.

a. Mapping Elemen Greedy
    i. Himpunan kandidat : Semua *command* yang berhubungan dengan *defending*.
    ii. Himpunan solusi : *Command* yang dipilih.
    iii. Fungsi solusi : Memeriksa apakah command yang dipilih memenuhi kondisi yang telah ditentukan.
    iv. Fungsi seleksi : Pilih *command* terurut berdasarkan prioritas yang telah dibuat.
    v. Fungsi kelayakan : Memeriksa apakah *command* yang dipilih dapat dijalankan
    vi. Fungsi objektif : Bertahan dari ancaman bot lawan

b. Analisis Efisiensi dan Efektivitas Solusi

    Terdapat dua pilihan utama yang digunakan dalam *greedy* ini, yaitu aktivasi *shield* dan menjauh dari ancaman bot musuh dengan size yang lebih besar.

    Greedy with defending ini merupakan salah satu strategi yang dipilih hanya pada saat kondisi tertentu saja seperti saat ada serangan torpedo dari lawan maupun ketika terdapat bot musuh yang memiliki *size* lebih besar dan berada dekat pada bot kita.

    Strategi ini cukup efektif untuk bertahan hidup karena sekalinya dimakan oleh bot lawan maka kita akan langsung kalah. Selain itu juga aktivasi shield selain membelokkan arah torpedo lawan, jika torpedo tersebut mengenai lawan maka kita yang akan mendapatkan *size* tambahan. Strategi ini kurang efektif ketika bot lawan menggunakan torpedo yang hanya memakan 10 *size* untuk memancing aktivasi shield kita yang memakan hingga 20 *size* sehingga bot akan kekurangan banyak size. Strategi ini bisa dibilang akan jarang terpakai dibanding strategi *greedy* lainnya.

3.1.3 Greedy with Surviving

Greedy with surviving adalah pendekatan algoritma *greedy* yang memprioritaskan untuk bertahan hidup selama mungkin sebagai pendekatannya. Bertahan hidup dapat membuat bot memenangkan permainan ini. Beberapa aksi bertahan hidup yang dapat dilakukan yaitu memutari jalan jika ada *gas cloud*, tidak keluar dari *world edge*, dan lain - lain.

a. Mapping Elemen Greedy
    i. Himpunan kandidat : Semua *command* yang berhubungan dengan *surviving.*
    ii. Himpunan solusi : *Command yang dipilih.*
    iii. Fungsi solusi : Memeriksa apakah command yang dipilih memenuhi kondisi yang telah ditentukan.
    iv. Fungsi seleksi : Pilih *command* terurut berdasarkan prioritas yang telah dibuat.

v. Fungsi kelayakan : Memeriksa apakah *command* yang dipilih dapat dijalankan
vi. Fungsi objektif : Bertahan hidup selama mungkin

b. Analisis Efisiensi dan Efektivitas Solusi

Terdapat beberapa pilihan utama yang digunakan dalam algoritma ini, diantaranya yaitu menghindar ketika ada *gas cloud,* tidak keluar dari *world edge,* dan lain - lain.

Greedy with Surviving ini merupakan salah satu strategi yang akan paling sering dipakai selama tidak sedang proses *attacking* maupun *defending* karena tujuan utama dari game ini ialah bertahan selama mungkin.

Strategi ini efektif karena memenuhi tujuan utama dalam game ini. Walaupun begitu, terdapat kelemahan jika kita terlalu sering menjalankan strategi ini maka ada saat - saat tertentu kita akan terpojok dan berakhir dengan kekalahan dimakan bot lain.

3.1.4 Greedy with Scoring

Greedy with scoring adalah pendekatan algoritma greedy yang melakukan pembobotan berdasarkan kondisi saat itu sehingga nantinya akan diambil keputusan dengan total skor atau pembobotan yang terbesar. Kondisi yang akan dihitung pembobotannya dilakukan dengan cara menghitung seluruh keadaan di sekitar bot seperti food, superfood, jarak musuh, jarak gas cloud, jarak asteroid field, jarak world edge dan size musuh terdekat.

a. Mapping Elemen Greedy
   i. Himpunan kandidat : Semua *command* yang tersedia.
   ii. Himpunan solusi : *Command yang dipilih.*
   iii. Fungsi solusi : Memeriksa apakah command yang dipilih memenuhi kondisi yang telah ditentukan.
   iv. Fungsi seleksi : Pilih *command* terurut berdasarkan prioritas yang telah dibuat.
   v. Fungsi kelayakan : Memeriksa apakah *command* yang dipilih dapat dijalankan
   vi. Fungsi objektif : Mencari *command* atau keputusan terbaik saat itu berdasarkan pembobotan

b. Analisis efisiensi dan Efektivitas solusi

Semua command dapat terpilih jika menerapkan algoritma greedy berdasarkan skoring, sehingga strategi ini cukup efektif dan komprehensif dalam mengambil keputusan. Namun dikarenakan tingkat ketelitian dalam implementasi algoritma ini sangat tinggi agar tidak terjadi salah pengambilan keputusan, maka algoritma ini cukup sulit untuk diaplikasikan untuk semua command atau printah.

3.2 Pendekatan Lain

- Prioritaskan *superfood* dibanding *food*

  Ketika jarak *superfood* terdekat berada kurang dari 75 ditambah jarak *food* terdekat, maka bot akan mengincar *superfood* terlebih dahulu. Hal ini dilakukan karena *superfood* memiliki *buff* ketika makan food maka akan dikali 2 *size* yang didapatkan. Ketika bot bergerak menuju *superfood* pun seringkali ada *food* disepanjang jalannya sehingga algoritma ini sangat berfungsi.

- Pembobotan setiap object

  Pada algoritma penghindaran yang dipakai pada Greedy by surviving, dilakukan pembobotan object - object seperti bot musuh, *gas cloud, wormhole,* dan lain - lain. Pembobotan ini dilakukan untuk menentukan arah gerakan terbaik yang akan diambil oleh bot.

- Menghindari center world

  Pada saat *early* hingga *mid game,* sebisa mungkin menghindari *center world,* hal ini dilakukan untuk mengurangi ancaman yang akan didapatkan karena berdasarkan hasil analisis kami saat *early* hingga *mid game*, *center world* menjadi tempat yang paling bahaya disebabkan bisa datangnya ancaman dari segala arah.

3.3 Solusi Greedy yang Diterapkan

Seperti yang telah dijelaskan pada poin - poin sebelumnya, terdapat tiga pendekatan *greedy* yang dapat diimplementasikan sesuai dengan situasi dan kondisinya. Sehingga, diputuskan untuk setiap *tick*, bot akan mempertimbangkan pilihan *command* yang akan dijalankan. Pada program bot kami, alur cara kerja bot yang kami pakai adalah sebagai berikut :

1. Pada awal spawn bot akan memakan *food* dan *superfood* di sekitarnya.
2. Jika memungkinkan, bot akan menembakkan teleport ke arah lawan terdekat sehingga bisa melakukan penyerangan.
3. Menghindari bot yang lebih besar, *world edge, gas cloud,* dan pusat *world* dengan skoring untuk menambah peluang hidup lebih lama.
4. Jika terdapat *torpedo* disekitar bot, maka akan mengaktivasi *shield*.
5. Jika sudah tidak ada *object* lagi, maka akan bermain dipusat *world* untuk menghindar dari *world edge*.

Poin - poin diatas merupakan gabungan dari beberapa alternatif solusi *greedy* yang sudah disebutkan sebelumnya. Poin 1,3, dan 5 merupakan implementasi *greedy with surviving* dengan skoring.. Poin 2 merupakan implementasi *greedy with attacking.* Sedangkan untuk poin 4 merupakan implementasi *greedy with defending.*

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy pada Bot Permainan Galaxio

Implementasi algoritma greedy terdapat pada file program BotService.java, pada method ComputeNextPlayerAction. Untuk mempermudah pembacaan kode program dan proses pengerjaan maka terdapat method lain yang kami buat. Method tersebut berada pada file yang sama, dan akan dijabarkan setelah pseudocode method ComputeNextPlayerAction.

4.2 Implementasi Dalam Pseudocode

```
{Teleport}

if (currentTeleporter != null) then

    if (distanceBetween(currentTeleporter,
nearestEnemy2Teleporter) < botSize / 2 and
nearestEnemy2Teleporter.size < botSize - 6) then

        playerAction <- TELEPORT

        setTeleporterHeading(-999)


{Stay in World Zone}

if (worldRadius - DistanceBetweenWorldCenter(bot) - (botSize /
2) < threatDistanceWorldBorder) then

    if (botFacingKuadran = 1) then

        if (bestScore(0, 2, 3, 4) = 2) then playerHeading <-
headingBetweenWorldCenter(-45)

        else if (bestScore(0, 2, 3, 4) = 3) then playerHeading
<- headingBetweenWorldCenter(0)

        else if (bestScore(0, 2, 3, 4) = 4) then playerHeading
<- headingBetweenWorldCenter(45)

        else playerHeading <- headingBetweenWorldCenter(0)


    else if (botFacingKuadran = 2) then

        if (bestScore(1, 0, 3, 4) = 1) then playerHeading <-
headingBetweenWorldCenter(45)
```

```
        else if (bestScore(1, 0, 3, 4) = 3) then playerHeading
<- headingBetweenWorldCenter(-45)

        else if (bestScore(1, 0, 3, 4) = 4) then playerHeading
<- headingBetweenWorldCenter(0)

        else playerHeading <- headingBetweenWorldCenter(0)


    else if (botFacingKuadran = 3) then

        if (bestScore(1, 2, 0, 4) = 1) then playerHeading <-
headingBetweenWorldCenter(0)

        else if (bestScore(1, 2, 0, 4) = 2) then playerHeading
<- headingBetweenWorldCenter(45)

        else if (bestScore(1, 2, 0, 4) = 4) then playerHeading
<- headingBetweenWorldCenter(-45)

        else playerHeading <- headingBetweenWorldCenter(0)


    else if (botFacingKuadran = 4) then

        if (bestScore(1, 2, 3, 0) = 1) then playerHeading <-
headingBetweenWorldCenter(-45)

        else if (bestScore(1, 2, 3, 0) = 2) then playerHeading
<- headingBetweenWorldCenter(0)

        else if (bestScore(1, 2, 3, 0) = 3) then playerHeading
<- headingBetweenWorldCenter(45)

        else playerHeading <- headingBetweenWorldCenter(0)


{Avoid Teleporter}


else if (teleporterList != null and currentTeleporter = null and
currentTeleporterHeading = -999 and
distanceBetween(nearestTeleporter, bot) <
threatDistanceTeleporter) then


    score1 <- scoringKuadran1(gameState)
```

```
    score2 <- scoringKuadran2(gameState)

    score3 <- scoringKuadran3(gameState)

    score4 <- scoringKuadran4(gameState)


    if (getHeadingKuadran(nearestTeleporter) = 1) then

        if (bestScore(0, score2, 0, score4) = score2) then
playerHeading = getSpecifiedHeadingBetween(nearestTeleporter, -
90)

        else playerHeading <-
getSpecifiedHeadingBetween(nearestTeleporter, 90)


    else if (getHeadingKuadran(nearestTeleporter) = 2) then

        if (bestScore(score1, 0, score3, 0) = score1) then
playerHeading <- getSpecifiedHeadingBetween(nearestTeleporter, -
90)

        else playerHeading =
getSpecifiedHeadingBetween(nearestTeleporter, 90)


    else if (getHeadingKuadran(nearestTeleporter) = 3) then

        if (bestScore(0, score2, 0, score4) = score2) then
playerHeading <- getSpecifiedHeadingBetween(nearestTeleporter, -
90)

        else playerHeading =
getSpecifiedHeadingBetween(nearestTeleporter, 90)


    else if (getHeadingKuadran(nearestTeleporter) = 4) then

        if (bestScore(score1, 0, score3, 0) = score1) then
playerHeading <- getSpecifiedHeadingBetween(nearestTeleporter, -
90)

        else playerHeading =
getSpecifiedHeadingBetween(nearestTeleporter, 90)
```

```
{Shoot Teleporter}

else if (currentTeleporterHeading = -999 and currentTeleporter =
null and distanceBetween(bot, nearestEnemy) < 500 and
distanceBetween(bot, nearestEnemy) > 75 and nearestEnemySize <
botSize - 27 and botSize > 40) then

    playerHeading <- headingBetween(nearestEnemy)

    setTeleporterHeading(headingBetween(nearestEnemy))

    playerAction <- FIRETELEPORT


{Activate Shield}

else if (torpedoList != null and distanceBetween(bot,
nearestTorpedo) <= threatDistanceTorpedo and botSize ><- 30)
then

    playerAction <- ACTIVATESHIELD


{Chasing a Way Smaller Enemy}

else if (enemyList != null and nearestEnemySize < 0.5 * botSize
and distanceBetween(nearestEnemy, bot) <= 2 * botSize) then

    playerHeading <- HeadingBetween(nearestEnemy)


{Shoot Torpedo}

else if (enemyList != null and botSize >= 20 and
((nearestEnemySize >= botSize and distanceBetween(bot,
nearestEnemy) <= threatDistanceEnemy) or (nearestEnemySize <
botSize and distanceBetween(bot, nearestEnemy) < 70)) and
bot.torpedoSalvoCount != 0) then

    playerHeading <- HeadingBetween(nearestEnemy)

    playerAction <- FIRETORPEDOES


{Avoid Gas Cloud}

else if (gasCloudList != null and
distanceBetween(nearestGasCloud, bot) < threatDistanceGasCloud)
then
```

```
if (botFacingKuadran = 1) then

    if (bestScore(0, 2, 3, 4) = 2) then playerHeading <-
getSpecifiedHeadingBetween(nearestGasCloud,100)

    else if (bestScore(0, 2, 3, 4) = 3) then playerHeading <-
getSpecifiedHeadingBetween(nearestGasCloud,180)

    else if (bestScore(0, 2, 3, 4) = 4) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,260)

    else playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)


else if (botFacingKuadran = 2) then

    if (bestScore(1, 0, 3, 4) = 1) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,260)

    else if (bestScore(1, 0, 3, 4) = 3) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,100)

    else if (bestScore(1, 0, 3, 4) = 4) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)

    else playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)


else if (botFacingKuadran = 3) then

    if (bestScore(1, 2, 0, 4) = 1) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)

    else if (bestScore(1, 2, 0, 4) = 2) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,260)

    else if (bestScore(1, 2, 0, 4) = 4) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,100)

    else playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)


else if (botFacingKuadran = 4) then
```

```
    if (bestScore(1, 2, 3, 0) = 1) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,100)

    else if (bestScore(1, 2, 3, 0) = 2) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)

    else if (bestScore(1, 2, 3, 0) = 3) then playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,260)

    else playerHeading =
getSpecifiedHeadingBetween(nearestGasCloud,180)


{Target A Smaller Enemy}

else if (nearestEnemySize + 50 < botSize and
distanceBetween(nearestEnemy, bot) <= 2 * botSize) then

    playerHeading <- HeadingBetween(nearestEnemy)


{Running Away from An Enemy}

else if (enemyList != null and nearestEnemySize >= botSize and
distanceBetween(nearestEnemy, bot) <= threatDistanceEnemy) then

    playerHeading <- SpecifiedHeadingBetween(enemyList.(0), 180)


{Going For Food}

else if (foodList != null and superFoodList != null) then

    {Prioritizing Super Food}

    if (distanceBetween(nearestSuperFood, bot) <
distanceBetween(nearestFood, bot) + 75) then

        playerHeading <- HeadingBetween(nearestSuperFood)

    {Going for Food}

    else playerHeading <- HeadingBetween(nearestFood)


{Going To Center}

else  playerHeading <- headingBetweenWorldCenter(0)
```

4.3 Struktur Data

Pada pembuatan bot Galaxio menggunakan bahasa Java, dalam pembuatannya program dibuat menggunakan pendekatan pemrograman berbasis objek yang mana data - datanya dibuat dalam sekumpulan kelas. Program ini secara umum terbagi menjadi 4 bagian, yaitu Enums, Models, Services, dan Main.

a. Enums

Pada bagian ini, terdapat beberapa kelas - kelas yang merupakan bagian dari enumerasi data yang merupakan nilai konstan. Kelas - kelas tersebut diantranya :

- ObjectTypes

  Kelas ObjectTypes berisi enumerasi tipe - tipe objek yang berada pada game ini, yaitu PLAYER, FOOD, WORMHOLE, GAS_CLOUD, ASTEROID_FIELD, TORPEDO_SALVO, SUPERFOOD, SUPERNOVA_PICKUP, SUPERNOVA_BOMB, dan TELEPORTER, SHIELD.

- PlayerActions

  Kelas PlayerActions berisi enumerasi aksi yang dapat dilakukan oleh bot pada game ini, yaitu FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRESUPERNOVA, DETONATESUPERNOVA, FIRETELEPORT, TELEPORT, dan ACTIVATESHIELD.

b. Models

Models adalah bagian program dimana memiliki fungsi dalam mempresentasikan dan memanipulasi data kelas ataupun objek dalam permainan Galaxio ini. Beberapa program yang terdapat pada Galaxio ini antara lain adalah sebagai berikut :

- GameObject

  Kelas GameObject menyimpan data atribut pada sebuah objek. Atribut - atribut ini yaitu id, size, speed, currentHeading, position, gameObjectType, effects, torpedoSalvoCount, supernovaAvailable, teleporterCount, dan shieldCount.

- GameState

  Kelas GameState menyimpan data atribut tentang *world* pada permainan Galaxio. Data tentang *world* yang disimpan yaitu world, gameObjects, dan playerGameObjects.

- GameStateDto

  Kelas GameStateDto mengirim data dari gameState atau sebagai representasi sederhana dari gameState. Atribut yang disimpan mirip dengan yang ada pada GameState yaitu world, gameObjects, dan playerObjects.

- PlayerAction

Kelas PlayerAction berisi fungsi dan prosedur yang berfungsi untuk mengatur arah dan aksi yang akan bot lakukan. Atribut yang disimpan yaitu playerId, action, dan heading.

- Position

  Kelas Position menyimpan dan mereturn posisi suatu objek. Atribut yang disimpan yaitu x dan y yang merupakan koordinat suatu objek.

- World

  Kelas World menyimpan segala sesuatu yang berkaitan dengan world. Atribut yang disimpan yaitu centerPoint, radius, dan currentTick.

c. Services

Service adalah bagian yang menyediakan fungsionalitas pada bot. Pada *package* bot permainan Galaxio berbasis bahasa Java kali ini, Services hanya memiliki satu file yaitu BotServices yang berisi algoritma - algoritma yang akan dipakai dalam pemilihan aksi yang akan dilakukan oleh bot. Pemain pun hanya perlu untuk mengubah isi BotServices sesuai dengan kebutuhan masing - masing untuk mencapai tujuan dari permainan ini yaitu bertahan hidup sehingga dapat memenangkan permainan.

d. Main

Bagian Main merupakan bagian kode yang dijalankan pertama kali saat dipanggil dari *executable*-nya.

4.4 Analisis dan Pengujian

Dalam algoritma yang kami buat, kami mendefinisikan beberapa kasus pada setiap *tick*. Yang pertama adalah kami menentukan perintah yang akan dieksekusi menggunakan sistem *scoring* yaitu pembobotan ketika bot berada pada kondisi yang kurang baik seperti dekat dengan *world edge,* terdapat musuh dengan *size* yang lebih besar berada dekat dengan bot, terdapat beberapa obstacle seperti *asteroid field* maupun *gas cloud.* Ketika bot berada pada salah satu kondisi ataupun beberapa kondisi diatas sekaligus, maka kami akan melakukan *scoring* kearah mana bot harus bergerak. Setiap kondisi dihitung *score*-nya sesuai dengan bobot yang sudah kami tentukan. Semakin besar *score* pada suatu arah berarti semakin aman jika bot bergerak kearah sana. Oleh karena itu, bot akan diperintahkan untuk bergerak menuju arah yang memiliki bobot terbesar.

Jika bot sedang tidak berada pada kondisi - kondisi diatas maka bot akan melakukan aksi lain seperti mengumpulkan *food* dan melakukan penyerangan kepada bot lawan dengan *size* yang lebih kecil sehingga dapat memakan mereka dan mendapatkan *size* tambahan.

BAB V

KESIMPULAN

5.1 Kesimpulan

Dari tugas besar ini, kami belajar membuat algoritma greedy untuk bot permainan Galaxio. Program akhir ini memanfaatkan tidak hanya satu algoritma greedy melainkan gabungan dari beberapa algoritma greedy.

5.2 Saran

Saran untuk kelompok kami di antaranya :

1. Setelah perilisan tubes, setidaknya membaca modul dan mencobanya, jangan menunggu hingga h-7 deadline.
2. Algoritma dan program ini masih bisa dikembangkan baik dari algoritma, efektivitasnya, hingga ke segi format kode.

DAFTAR PUSTAKA

Levitin, Anany, Introduction to The Design and Analysis of Algorithms, 3rd ed, USA: Addison-Wesley, 2012, pp. 97-98

LINK PENTING :

LINK YOUTUBE : https://youtu.be/DgUJKA0QUcA
LINK GITHUB : https://github.com/ilhamanullah/Tubes1_Artificial-Ilham