Laporan Tugas Besar I

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Galaxio"

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2022/2023

Kelompok "CSS"

M Zulfiansyah Bayu Pratama	13521028
Jauza Lathifah Annassalafi	13521030
Fahrian Afdholi	13521031

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

DAFTAR ISI

BAB 1

DESKRIPSI TUGAS

1.1 Penjelasan Tugas

Galaxio adalah sebuah *game battle royale* yang mempertandingkan bot kapal yang telah dibuat dengan beberapa bot kapal yang lain. Setiap kelompok akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal tetap hidup hingga akhir permainan. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.

Pada tugas besar pertama Strategi Algoritma ini, digunakan sebuah *game engine* yang mengimplementasikan permainan *Galaxio*. Tugasnya adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan strategi greedy untuk memenangkan permainan. Strategi greedy yang diimplementasikan harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mempertahankan kapal paling terakhir untuk hidup. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menghindari objek yang dapat mengurangi ukuran kapal.

Game engine dapat diperoleh pada laman berikut:

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

Untuk mengimplementasikan bot tersebut, lanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini:

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

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

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

derajat. Efek *afterburner* akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.

3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. *Food* akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan oleh kapal *player*. Apabila *player* mengkonsumsi *Food,* maka *Player* akan bertambah ukuran yang sama dengan *Food. Food* memiliki peluang untuk berubah menjadi *Super Food.* Apabila *Super Food* dikonsumsi maka setiap makan *Food,* efeknya akan 2 kali dari *Food* yang dikonsumsi. Efek dari *Super Food* bertahan selama 5 tick.

4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari *player* untuk memasukinya dan keluar di pasangan satu lagi. *Wormhole akan* bertambah besar setiap tick game hingga ukuran maximum. Ketika *Wormhole* dilewati, maka *wormhole* akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat *wormhole* lebih besar dari kapal *player.*

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

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

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

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

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

2.1 Algoritma Greedy

Algoritma greedy merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Persoalan optimasi (optimization problems) merupakan persoalan mencari solusi optimal. Hanya ada dua macam persoalan optimasi, yakni:

1. Maksimasi (maximization)
2. Minimasi (minimization)

Algoritma ini membentuk solusi langkah per langkah. Pada setiap langkahnya, terdapat banyak pilihan yang perlu dievaluasi dan akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil, dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya. Prinsip utama algoritma greedy adalah “take what you and get now!”. Maksudnya adalah pada setiap langkah dalam algoritma greedy, akan diambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya yang dinamakan dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir. Optimum global belum tentu merupakan solusi optimum (terbaik), tetapi sub-optimum atau pseudo-optimum. Alasannya adalah Algoritma greedy tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada (sebagaimana pada metode exhaustive search dan terdapat beberapa fungsi seleksi yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optiamal. Jadi, pada sebagian masalah algoritma greedy tidak selalu berhasil memberikan solusi yang optimal.

Algoritma greedy memiliki beberapa elemen, yaitu sebagai berikut:

1. Himpunan Kandidat, semua ruang sampel yang dapat dipilih sebagai solusi.
2. Himpunan Solusi, himpunan bagian dari himpunan kandidat yang merupakan solusi dari permasalahan yang ada
3. Fungsi Seleksi, fungsi yang digunakan untuk menyeleksi anggota-anggota dari himpunan kandidat yang akan diperiksa menggunakan fungsi kelayakan agar dapat masuk ke dalam himpunan solusi.
4. Fungsi Layak, fungsi yang digunakan untuk memeriksa apakah solusi yang dipilih merupakan solusi yang layak untuk dimasukkan ke dalam himpunan solusi atau tidak.
5. Fungsi Objektif, solusi optimum yang dihasilkan.

Dengan kata lain, algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S di optimisasi oleh fungsi objektif.

Algoritma greedy dapat digunakan untuk masalah yang hanya membutuhkan solusi hampiran dan tidak memerlukan solusi terbaik mutlak. Solusi ini terkadang lebih baik daripada algoritma yang menghasilkan solusi eksak dengan kebutuhan waktu yang eksponensial.

2.2 Cara Kerja Bot Game Galaxio

Secara garis besar, cara kerja bot dari permainan Galaxio ini adalah pertama-tama bot akan mengambil data yang tersedia pada state files .

- GameObject : list yang berisi objek-objek yang terdapat pada map
  - Id : id objek
  - size : ukuran objek
  - speed : kecepatan objek
  - currentHeading : arah objek
  - position : posisi objek
  - gameObjectType : tipe objek objek, yaitu enemy
- GameState : informasi permainan pada tick saat ini
  - world : objek map
  - gameObject : list objek yang ada di map
  - playerGameObject : list pemain
- PlayerAction : aksi bot yang akan dikirim ke server.
  - playerId : id pemain
  - action : aksi pemain
  - heading : sudut bot ketika melakukan aksi
- Position : posisi objek (x, y)
- World : informasi map
  - centerPoint : titik tengah pada map (x, y)
  - radius : jari-jari lingkaran map saat ini

- currentTick : informasi tick saat ini

Setelah bot mengambil data-data di atas, maka bot dapat melakukan analisis menggunakan algoritma yang telah dijelaskan sebelumnya untuk mencari command paling tepat untuk dieksekusi. Algoritma yang dituliskannya tersebut berbentuk fungsi objektif dari algoritma greedy.

2.3 Implementasi Algoritma Greedy

Pemilihan algoritma greedy sebagai algoritma pembentukan bot dikarenakan cukup intuitif serta tidak memerlukan waktu atau resource yang terlalu banyak. Selain itu, terdapat cukup banyak kemungkinan solusi greedy yang dapat dieksplorasi. Penyusunan sebuah algoritma greedy sangat penting dengan memikirkan semua kemungkinan yang dapat bot lakukan sehingga membuat bot dapat bertahan hingga akhir. Meskipun terdapat peluang tidak dapat menciptakan algoritma greedy yang menghasilkan solusi optimum global, tetapi setidaknya dapat selalu mencapai solusi optimum lokal yang nilainya mendekati solusi optimum global.

2.4 Game Engine Permainan Galaxio

Game engine adalah perangkat lunak yang dapat digunakan untuk membuat game dan pengembangan lebih ekonomis. Game Galaxio dalam tugas ini dijalankan menggunakan game engine yang sudah tersedia dan dapat diunduh pada laman

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

Selain game engine-nya sendiri, untuk menjalankan permainan, dibutuhkan beberapa *requirement* dasar sebagai berikut.

a. Java (minimal Java 11):

   https://www.oracle.com/java/technologies/downloads/#java

b. IntelliJ IDEA: https://www.jetbrains.com/idea/

c. .Net Core 3.1: Link tersedia pada panduan berikut.

File lain yang perlu diunduh adalah file starter-pack.zip pada laman github galaxio. Game engine tersedia dalam bentuk Executable Jar File yang kemudian akan digunakan untuk menjalankan permainan.

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

a. Engine

Engine merupakan komponen yang berperan dalam mengimplementasikan logic dan rules game. Engine akan meneruskan perintah yang dikeluarkan oleh bot ke dalam game untuk diproses, dengan memperhatikan peraturan game yang sudah diatur sebelumnya.

b. Runner

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

c. Logger

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

Dalam menjalankan program ini, terdapat file “run.bat”. Pemain dengan sistem operasi berbasis windows dapat melakukan double-click dalam menjalankan permainannya, sedangkan pemain dengan sistem operasi Linux/Mac dapat menjalankan command “make run”. Pada dasarnya, game-engine dijalankan dengan mode command-line interface (CLI). Untuk mempermudah visualisasi, dapat digunakan visualizer yang terdapat pada folder visualizer dengan nama file “Galaxio”.

BAB 3

APLIKASI ALGORITMA GREEDY

3.1 Pemetaan Persoalan Galaxio

- Himpunan Kandidat : Perintah tindakan yang dilakukan bot pemain dengan representasi

    1. FORWARD
    2. TELEPORT
    3. ACTIVATESHIELD
    4. FIRETORPEDOES
    5. STARTAFTERBURNER
    6. STOPAFTERBURNER
    7. FIRETELEPORT

- Himpunan Solusi : command-command yang terpilih
- Fungsi Solusi : Memeriksa apakah bot pemain dapat bertahan paling akhir.
- Fungsi Seleksi : Memilih command yang mengarahkan bot untuk memakan musuh yang lebih kecil, melakukan aksi agar bot tidak termakan oleh musuh dan tertembak musuh.
- Fungsi Kelayakan : Memeriksa apakah aksi yang dipilih sesuai dengan orientasi dari algoritma yang dipilih
- Fungsi Objektif : Bot dapat bertahan hingga akhir dan menghasilkan kemenangan

3.2 Eksplorasi Alternatif Solusi Greedy

- Greedy by Size

  Algoritma ini dapat menggunakan command FORWARD yang melakukan aksi untuk mengincar FOOD, SUPERFOOD, menghindar dari object berbahaya yang membuat ukuran bot berkurang, dan menghindar dari serangan musuh.

    - ➢ Himpunan Kandidat : semua command yang tersedia
    - ➢ Himpunan Solusi : command yang terpilih
    - ➢ Fungsi Solusi : Memeriksa apakah command yang dipilih membuat ukuran bot menjadi lebih besar.

- ➢ Fungsi Seleksi : Memilih command yang fokus mengambil makanan dan menghindar objek berbahaya
- ➢ Fungsi Kelayakan : Memeriksa apakah command yang dipilih valid dan dapat dijalankan
- ➢ Fungsi Objektif : Memaksimumkan ukuran bot dan bertahan hingga akhir.

- Greedy by Defense

  Algoritma ini dapat mengandalkan command ACTIVATESHIELD yang dapat melindungi ketika terdapat serangan berupa torpedo salvo dari bot lain dan FIRETELEPORT - TELEPORT yang dapat melakukan teleport ke tempat yang aman atau jauh dari musuh.

  - ➢ Himpunan Kandidat : semua command yang tersedia
  - ➢ Himpunan Solusi : command yang terpilih
  - ➢ Fungsi Solusi : Memeriksa apakah command yang dipilih membuat bot dapat bertahan terhadap serangan musuh.
  - ➢ Fungsi Seleksi : Memilih command yang fokus melindungi bot dari serangan dan menghindar dari objek berbahaya
  - ➢ Fungsi Kelayakan : Memeriksa apakah command yang dipilih valid dan dapat dijalankan
  - ➢ Fungsi Objektif : Mempertahankan bot dan bertahan hingga akhir.

- Greedy by attack

  Algoritma ini dapat mengandalkan command FIRETORPEDOES yang melakukan penyerangan terhadap musuh sehingga ukuran musuh mengecil dan mati, STARTAFTERBURNER - STOPAFTERBURNER untuk mendekati musuh yang lebih kecil dengan kecepatan yang lebih tinggi dan memakannya , FIRETELEPORT - TELEPORT untuk melakukan teleport ke musuh yang lebih kecil dan memakannya.

  - ➢ Himpunan Kandidat : semua command yang tersedia
  - ➢ Himpunan Solusi : command yang terpilih
  - ➢ Fungsi Solusi : Memeriksa apakah command yang dipilih membuat bot dapat melakukan penyerangan terhadap musuh.
  - ➢ Fungsi Seleksi : Memilih command yang fokus melakukan penyerangan.
  - ➢ Fungsi Kelayakan : Memeriksa apakah command yang dipilih valid dan dapat dijalankan
  - ➢ Fungsi Objektif : Menyerang bot musuh dan bertahan hingga akhir.

3.3 Analisis Efisiensi dan Efektifitas dari Kumpulan Alternatif Solusi Greedy

Alternatif Solusi Greedy	Efisiensi	Efektifitas
Greedy by Size	Greedy by Size akan memprioritaskan untuk mengincar FOOD dan SUPERFOOD dan melakukan FORWARD, memiliki kompleksitas sebesar O(n).	Strategi ini efektif apabila algoritma dalam menghindar musuh dan objek berbahaya memiliki keakuratan yang baik dan banyak terdapat FOOD dan SUPERFOOD pada map. Namun tidak efektif apabila algoritma tidak dapat mendeteksi musuh dan objek berbahaya dengan baik
Greedy by Defense	Greedy by Defense butuh untuk mengecek kondisi dari musuh dan memiliki kompleksitas algoritma O(n).	Strategi ini efektif apabila algoritma mengaktifkan shield dengan tepat dan melakukan teleport ke tempat yang aman dari musuh. Namun tidak efektif apabila tidak melakukan aksi untuk memperbesar ukurannya sehingga tidak dapat melakukan defense dengan menggunakan shield dan teleport.
Greedy by Attack	Greedy by Attack butuh untuk mengecek posisi musuh dan kondisi musuh, memiliki kompleksitas O(n)	Strategi ini efektif apabila algoritma dapat melakukan penyerangan dengan torpedo salvo tepat kemusuh dan dapat menembakan FIRETELEPORT ke musuh yang lebih kecil dengan baik sehingga dapat teleport dan memakannya. Namun tidak efektif apabila tidak melakukan aksi untuk memperbesar ukurannya sehingga tidak dapat melakukan aksi penyerangan

		terhadap musuh.

3.5 Strategi Greedy yang dipilih

Seperti yang telah dipaparkan sebelumnya, terdapat banyak pendekatan greedy yang dapat diimplementasikan. Pada implementasi program bot kami, alur berpikir yang kami pakai adalah sebagai berikut:

1. Prioritas 1 (Greedy by attack)

   Jika bot dapat teleport, bila bot berada pada posisi teleport maka jarak antara bot dengan musuh sangat dekat dan musuh tersebut mempunyai ukuran yang lebih kecil maka bot akan teleport. Jika tidak terdeteksi musuh maka akan ke prioritas kedua.

2. Prioritas 2 (Greedy by defense and attack)

   Apabila terdeteksi jarak torpedo salvo terhadap bot kurang dari 200 dan torpedo salvo mengarah ke bot kita, kemungkinan yang terjadi sebagai berikut:

   - Jika bot memiliki shield dan ukuran bot sudah bisa menggunakannya, maka bot akan memasang shield.
   - Jika ukuran bot tidak memenuhi untuk dapat menggunakan shield, maka akan mengeluarkan torpedo salvo
   - Jika tidak memenuhi keduanya bot akan berusaha untuk kabur.

3. Prioritas 3 (Greedy by attack)

   Apabila terdapat musuh lebih dari satu, bot dapat mengeluarkan torpedo salvo, musuh berada dalam radius tembak, dan tidak memakai shield, maka bot akan mengeluarkan torpedo salvo.

4. Prioritas 4 (Greedy by attack)

   Jika hanya terdapat satu musuh, dan ukuran bot lebih besar maka bot akan berusaha untuk mengejar musuh. Namun, jika ukuran musuh lebih besar maka bot akan mengeluarkan torpedo salvo ke arah musuh.

5. Prioritas 5 (Greedy by Size)

   Jika bot tidak berada dalam kondisi yang sudah dijelaskan diatas, maka bot akan makan dengan memakan makanan yang tidak dekat dengan objek berbahaya.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Pseudocode

```
Kamus Lokal

GameObject bot

PlayerAction playerAction

GameState gameState

boolean afterburner <- false

int shieldUse <- 0

int torpedoItem <- 0

int teleporterItem <- 100

int teleporterCount <- -1

int teleporterTick <- 0

int tickCount <- 0


Algoritma

    // Jika ada musuh yang berada pada radius tembak, maka bot akan
menembak musuh

    // tersebut

    if (enemy.size() > 1) then

        int idxMin <- 0

        for (int i = 0; i < enemy.size(); i++) then

            // Jika musuh berada dalam radius tembak, maka bot akan
menembak musuh tersebut

            if (isEnemyInRadius(enemy.get[i], bot) and (bot.getSize
>= 40) and (torpedoItem >= 10)) then

                // Deteksi apakah musuh tersebut memakai shield
```

```
atau tidak


                // Cari di dalam list shield, apakah posisinya sama
dengan musuh yang kita
                // targetkan
                boolean isShield <- false
                for (int j = 0; j < shield.size(); j++)
                    if
(shield.get[j].getPosition.equals(enemy.get[i].getPosition)) then
                        // Jika sama maka langsung batal tembak
                        isShield <- true
                        break
                // Jika tidak ada shield, maka bot akan menembak
musuh tersebut
                if (!isShield) then
                    playerAction.action <-
PlayerActions.FIRETORPEDOES
                    playerAction.heading <-
getHeadingBetween(enemy.get[i])
                    torpedoItem <- torpedoItem - 10
                    updateItemBot
                    this.playerAction <- playerAction
                    return
            if (enemy.get[i}.getSize < enemy.get[idxMin].getSize)
then
                idxMin <- i;


        // Jika terdapat musuh yang lebih kecil dari bot, maka bot
menembakkan
        // teleporter
        if ((bot.getSize > (enemy.get[idxMin].getSize) * 2) and
```

```
(bot.getSize >= 200)
                and (teleporterItem >= 100)) then
            playerAction.action <- PlayerActions.FIRETELEPORT
            playerAction.heading <-
getHeadingBetween(enemy.get[idxMin])
            teleporterItem <- teleporterItem - 100
            teleporterCount <- (int) getDistanceBetween(bot,
enemy.get[idxMin]) / bot.getSpeed


            updateItemBot
            this.playerAction <- playerAction
            return


    else if (enemy.size == 1) then
        int i <- 0
        if ((((bot.getSize - 20) > (enemy.get[i].getSize) * 2
                + (int) getDistanceBetween(bot, enemy.get[i]))) and
(bot.getSize > 50)) then
            playerAction.heading <- getHeadingBetween(enemy.get[i])
            if (distancceOfEat(getDistanceBetween(bot,
enemy.get[i]))) then
                playerAction.action <-
PlayerActions.STARTAFTERBURNER
                afterburner <- true
                updateItemBot
                this.playerAction <- playerAction
                return
            else
                if (afterburner) then
                    playerAction.action =
```

```
PlayerActions.STOPAFTERBURNER
                    afterburner <- false
                    updateItemBot
                    this.playerAction <- playerAction
                    return
        else
            if (afterburner) then
                playerAction.action <-
PlayerActions.STOPAFTERBURNER
                afterburner <- false
                updateItemBot
                this.playerAction <- playerAction
                return


            // Jika kita dapat menembak musuh
            if (isEnemyInRadius(enemy.get[i], bot) and (bot.getSize
>= 60) and (torpedoItem >= 10)) then
                // Deteksi apakah musuh tersebut memakai shield
atau tidak


                // Cari di dalam list shield, apakah posisinya sama
dengan musuh yang kita
                // targetkan
                boolean isShield <- false
                for (int j = 0; j < shield.size(); j++)
                    if
(shield.get[j].getPosition.equals(enemy.get[i].getPosition)) then
                        // Jika sama maka langsung batal tembak
                        isShield <- true
```

```
                // Jika tidak ada shield, maka bot akan menembak
musuh tersebut

                if (!isShield) then

                    playerAction.action <-
PlayerActions.FIRETORPEDOES

                    playerAction.heading <-
getHeadingBetween(enemy.get[i])

                    torpedoItem <- torpedoItem - 10

                    updateItemBot

                    this.playerAction <- playerAction

                    return

updateItemBot

this.playerAction <- playerAction
```

```
function int isOtherObjectNearTorpedo(GameObject bot, GameObject
enemy) -> int

        // Buat sebuah list yang isinya adalah objek-objek yang
berbahaya

        // Objek yang berbahaya untuk torpedo adalah gascloud,
asteroid, food, dan

        // superfood

        int torpedoSize <- 10;



        List<GameObject> listDangerousObject <- new ArrayList<>()

        listDangerousObject.addAll(gameState.getGameObjects.stream

                .filter(item -> (item.getGameObjectType ==
ObjectTypes.GAS_CLOUD)).collect(Collectors.toList));

        listDangerousObject.addAll(gameState.getGameObjects.stream

                .filter(item -> (item.getGameObjectType ==
ObjectTypes.ASTEROID_FIELD)).collect(Collectors.toList));
```

```
        listDangerousObject.addAll(gameState.getGameObjects.stream

                .filter(item -> (item.getGameObjectType ==
ObjectTypes.FOOD)).collect(Collectors.toList));

        listDangerousObject.addAll(gameState.getGameObjects.stream

                .filter(item -> (item.getGameObjectType ==
ObjectTypes.SUPERFOOD)).collect(Collectors.toList));



        // Coba hitung persamaan garis lurus yang dibentuk dari bot
ke musuh

        // y = mx + c

        double m <- ((double) enemy.getPosition.y - (double)
bot.getPosition.y)

                / ((double) enemy.getPosition.x - (double)
bot.getPosition.x)

        double c <- (double) bot.getPosition.y - (m * (double)
bot.getPosition.x)



        // Cari seluruh kemungkinan object yang berada di garis
lurus tersebut

        for (int i = 0; i < listDangerousObject.size; i++)

            // Tentukan pusat lingkaran pada objek yang berbahaya

            double a <- (double)
listDangerousObject.get[i].getPosition.x

            double b <- (double)
listDangerousObject.get[i].getPosition.y



            // Tentukan jari-jari lingkaran pada objek yang
berbahaya

            double r <- (double) listDangerousObject.get[i].getSize



            // Tentukan titik potong antara garis lurus dan
lingkaran

            // Rumusnya adalah x = (-m*c + b +/- sqrt((m^2 + 1) *
```

```
(c^2 - r^2 + b^2)) / (m^2

            // + 1)

            double jarak <- Math.abs((m * a - b + c) / Math.sqrt(m
* m + 1))



            // tentukan apakah objek berbahaya berada di garis
lurus

            if (jarak <= r) then

                torpedoSize <- torpedoSize -
listDangerousObject.get[i].getSize

            if (torpedoSize <= 0) then

                -> 0

        -> torpedoSize;
```

```
function isEnemyInRadius(GameObject enemy, GameObject bot) ->
boolean

        int torpedoSize <- isOtherObjectNearTorpedo[bot, enemy]

        double distance <- getDistanceBetween[enemy, bot]

        if (distance > 800) then

            -> false

        // Hitung berapa tick yang diperlukan agar torpedo
menghantam musuh

        // Kecepatan torpedo adalah 60 pixel per tick

        double timeToEnemy <- (distance + ((double) enemy.getSize)
+ ((double) bot.getSize)) / 60.0

        // Hitung posisi musuh setelah beberapa tick

        double enemyX <- (((double) enemy.getSpeed) * timeToEnemy)



        if ((enemyX) < (((double) enemy.getSize) + torpedoSize))
then

            -> true
```

```
        -> false
```

```
function int goToFood(List<GameObject> foodList, List<GameObject>
superFood, GameObject bot,

        // Awalnya masukkan semua isi foodList, superFood, dan
supernovaPickup ke dalam

        // listFood

        List<GameObject> listFood <- new ArrayList<GameObject>()

        listFood.addAll[foodList]

        listFood.addAll[superFood]

        listFood.addAll[supernovaPickup]



        // Urutkan listFood berdasarkan jarak terdekat

        listFood <-
listFood.stream.sorted[Comparator.comparing[item ->
getDistanceBetween[bot, item]]]

                .collect[Collectors.toList]



        // Jika listFood tidak kosong, maka bot akan menuju food
atau superfood terdekat

        if (listFood.size > 0) then

            // Looping semua kemungkinan dalam listFood

            for (int i = 0; i < listFood.size(); i++)



                // Jika isi dari ListFood dekat dengan benda
berbahaya, maka cari food

                // selanjutnya



                if (getDistanceBetween[listFood.get[i],

                        getNearDangerousObject[listDangerousObject,
```

```
listFood.get[i]]] < ((double) bot
                                .getSize - 3) * 2) then
                    continue


                // Jika food berada dekat pada radius batas map,
maka cari food selanjutnya


                // Jika isi dari ListFood tidak dekat dengan benda
berbahaya, maka
                // Cek apakah food tersebut adalah supernovaPickup
dan jarak antara bot dan food
                // tersebut dekat
                if (listFood.get[i].getGameObjectType ==
ObjectTypes.SUPERNOVA_PICKUP
                        and getDistanceBetween[bot,
listFood.get(i)] < (double) bot.getSpeed) then
                    // Jika dekat, maka bot akan menuju food
tersebut
                    supernovaItem <- true


                // Kembalikan heading menuju food tersebut jika
tidak ada benda berbahaya
                -> getHeadingBetween[listFood.get[i]];
        -> 0
```

```
function GameObject getNearDangerousObject(List
listDangerousObject, GameObject obj) -> GameObject
        double distance <- getDistanceBetween[obj,
listDangerousObject.get[0]];
        GameObject ret <- listDangerousObject.get[0];


        for (int i = 0; i < listDangerousObject.size; i++) then
            if (distance > getDistanceBetween[bot,
```

```
listDangerousObject.get[i]]) then
                ret = listDangerousObject.get[i]
        -> ret
```

```
 function boolean isGoToSupernovaPickup(GameObject bot,
List<GameObject> supernovaPickup, List<GameObject> enemy) ->
boolean
        double distance = getDistanceBetween[bot,
supernovaPickup.get[0]];
        double distanceEnemy =
getDistanceBetween[supernovaPickup.get[0], enemy.get[0]];
        if ((distance < distanceEnemy) && (distance < 200)) then
            -> true
        else
            -> false
```

```
function boolean distanceOfShoot(double distance) -> boolean
        if (distance < 600 && distance > 100) then
            -> true
        else
            -> false
```

```
function boolean distancceOfEat(double distance) -> boolean
        if (distance <= 100) then
            -> true
        else
            -> false
```

```
function boolean distanceOfBigBigerShoot(double distance) ->
boolean
        if (distance < 1000) then
            -> true
```

```
        else
            -> false
```

Untuk kode lengkap dapat dilihat pada link berikut:

https://github.com/fchrgrib/tubes1_STIMA

4.2 Struktur Data pada Bot Permainan Galaxio

Struktur data pada permainan galaxio ini berbentuk class. Class dibagi menjadi 3 kategori utama, yaitu: Enums yang berisi berisi konstan – konstan objek yang merupakan bagian dari permainan, Models berisi objek – objek yang bisa di instansiasi pada game galaxio, services merupakan tempat logika bot disimpan. Pembuatan bot dan pengaplikasian algoritma greedy dilakukan pada class ini. Terdapat juga Main.java, 3 kategori diatas merupakan class yang sudah disediakan dari entellect challenge dari awal.

a. Enums

   Pada bagian ini, kelas-kelas yang terdapat di dalamnya melambangkan konstanta yang ada pada permainan galaxio agar dapat dibaca dengan lebih baik oleh bot. Kelas-kelas tersebut antara lain:

   - ObjectTypes memiliki atribut value.
   - PlayerAction memiliki atribut value.

b. Models

   Pada bagian ini, kelas-kelas yang terdapat di dalamnya melambangkan entitas yang berada pada permainan galaxio beserta dengan atribut yang dimilikinya. Kelas-kelas tersebut antara lain:

   - GameObject memiliki atribut id, size, speed, currentHeading, position, gameObject.
   - GameState memiliki atribut world, list gameObject, list playerGameObjects.
   - GameStateDto memiliki atribut world, list gameObject, list playerGameObjects.
   - PlayerAction memiliki atribut playerId, action, heading.
   - Position memiliki atribut x, y.

- World memiliki atribut centerPoint, radius, currentTick.

c. Services

Pada bagian ini, terdapat satu kelas Bot pada file BotService.java yang berisi implementasi inti dari bot. Pada kelas ini, terdapat beberapa atribut dan fungsi yang berguna untuk keberjalanan bot itu sendiri.

d. Main

Pada bagian Main, terdapat entrypoint dari bot, atau dengan kata lain bagian kode pertama yang dijalankan saat bot pertama kali dipanggil dari executable-nya. Isi dari kelas Main tidak terlalu banyak, hanya terdapat inisialisasi hal-hal teknis yang berkaitan dengan penerimaan game state dan penerjemahan command dari bot menuju game engine.

4.3 Analisis Desain Solusi Algoritma Greedy

Solusi algoritma greedy yang diterapkan pada permainan cukup optimal pada beberapa kasus. Algoritma yang sudah dibuat dapat mendeteksi musuh dan melakukan aksi yang harus dilakukan terhadap musuh dengan akurat, seperti melakukan defense dengan teleport, memasang shield, menjauhi musuh, dan mengeluarkan torpedo salvo.

Bot kami memiliki kelebihan pada saat menggunakan *FireTorpedoes* karena pada saat menggunakan skill tersebut kami menggunakan perhitungan *head* musuh sedang menghadap kemana dan musuh akan jalan kemana sehingga bot kami dapat menggunakan skil tersebut secara efektif dan tepat walaupun beberapa *case* pada saat menembak tidak memiliki keakuratan yang kurang dari 100%.

Selain itu, bot kami juga mengeluarkan skill *shield* pada saat timing yang tepat yaitu pada saat lawan mengeluarkan sebuah *torpedo salvo* pada jarak tertentu, maka shield akan dikeluarkan, sebelumnya kami membuat *shield* dengan timing yang kurang tepat karena tidak memperhitungkan *torpedo salvo* yang dikeluarkan dari bot kami sendiri sehingga bot mengeluarkan *skill shield* pada waktu yang kurang tepat, tetapi pada akhirnya kami dapat mengatasi masalah tersebut sehingga mengeluarkan *skill shield* dengan waktu yang tepat.

Di dalam kode kami terdapat sebuah list yang mengumpulkan benda-benda yang berbahaya jika terkena bot di dalam satu list yang sangat memudahkan dalam pengembangan algoritma ini sehingga kode bisa terlihat menjadi lebih bersih.

Selain itu, dalam kode kami sedikit sekali membuat sebuah *nested if* sehingga kode dapat dengan mudah dibaca dan tidak membuat seseorang kebingungan dalam mengembangkan algoritma yang akan dibuat di dalam kode yang ada.

Dalam beberapa kasus bot memiliki beberapa kendala yaitu pada saat musuh mengeluarkan skill *fire torpedo* secara agresif bot kami akan kebingungan karena banyaknya *torpedo salvo* yang dihadapi sehingga biasanya bot akan berjalan tidak karuan. Pada kasus

harus melakukan aksi makan ke suatu food atau superfood terkadang bot tidak dapat mendeteksi objek berbahaya dan menabraknya.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar ini, kami dibuat berpikir untuk menciptakan algoritma greedy terbaik yang bisa diimplementasikan pada Game Galaxio. Tujuan utama dari algoritma greedy kami adalah bertahan hingga akhir dengan melakukan penyerangan terhadap musuh jika kondisinya terpenuhi dan melakukan pertahanan jika terdapat aksi dari bot lain yang membahayakan bot kita. Hasilnya, algoritma kami bisa bertahan hingga akhir ketika melawan bot referensi di setiap match nya. Kita juga melakukan percobaan dengan bot yang bisa melakukan penyerangan dan bot kita memiliki persentase kemenangan 80%.

Algoritma greedy dapat diimplementasikan sebagai strategi untuk bot dalam game galaxio dengan memecahkan persoalan secara langkah per langkah, dimana di setiap langkah akan diambil pilihan terbaik saat itu, tanpa melihat konsekuensi yang ada kedepannya. Tetapi, solusi yang didapatkan dari penggunaan algoritma greedy belum tentu merupakan solusi optimum (terbaik), hal ini dikarenakan algoritma greedy memiliki pilihan fungsi seleksi yang berbeda-beda, sehingga pemilihan fungsi seleksi yang tepat sangat penting dalam penggunaan algoritma greedy.

5.2 Saran

Tugas besar IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 menjadi salah satu proses pembelajaran bagi kelompok dalam menerapkan ilmu-ilmu yang dipelajari pada perkuliahan ataupun dengan melakukan eksplorasi materi secara mandiri. Saran-saran yang dapat kami berikan adalah:

1. Algoritma yang digunakan pada tugas besar ini masih memiliki kekurangan-kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, seperti dengan tidak menggunakan fungsi yang sama berulang-ulang.

2. Penulisan pseudocode tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca pseudocode dengan asumsi program sudah well commented.

3. Cara penulisan kode dan kemampuan menulis komentar juga menjadi hal yang sangat penting dalam mengerjakan kode pemrograman secara berkelompok. Dengan adanya komentar pada kode, anggota lain pada kelompok dapat memahami cara kerja suatu kode dengan lebih cepat.

5.3 Link Repository dan Video Demo

Link Repository : https://github.com/fchrgrib/tubes1_STIMA

Link Video : https://bit.ly/VideoTubesnyaCSS

DAFTAR PUSTAKA

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

https://www.it-jurnal.com/pengertian-algoritma-greedy/

https://www.scribd.com/document/344143304/Pengertian-Metode-Greedy-Dan-Algoritma-Greedy

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

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Greedy-%282018%29.pdf