

IMPLEMENTASI ALGORITMA *GREEDY* DALAM PERMAINAN GALAXIO

Diajukan sebagai pemenuhan tugas besar 1.



Oleh:

Kelompok 51 (*Greedland*)

1. 13521067 - Yobel Dean Christoper
2. 13521117 - Maggie Zeta Rosida S
3. 13521131 - Jeremy Dharmawan Rahardjo

Dosen Pengampu : Dr. Ir. Rinaldi Munir, MT.

IF2211 - Strategi Algoritma

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2023

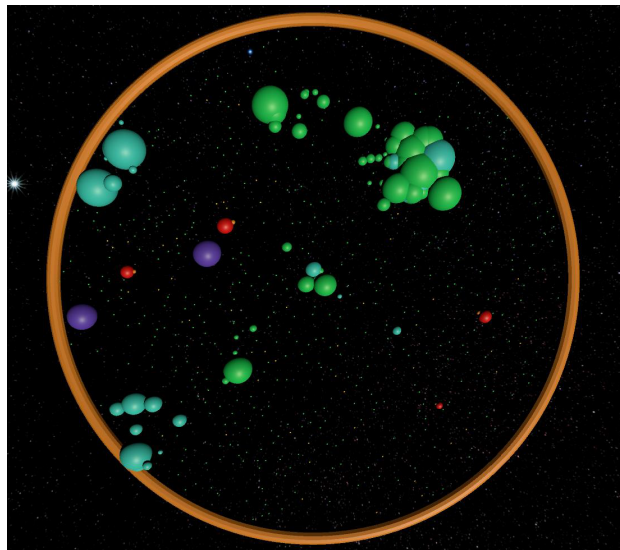
DAFTAR ISI

DAFTAR ISI	1
BAB I	2
BAB II	4
2.1. Algoritma Greedy	4
2.2. Cara Kerja Program	4
BAB III	7
3.1. Dekomposisi dan Pemetaan Masalah dalam Permainan Galaxio Menjadi Upa Elemen Algoritma Greedy	7
3.1.1 Greedy by Growth Rate	7
3.1.2 Defensive Greedy	8
3.1.3 Offensive Greedy	9
3.2. Multialternatif Strategi Greedy	10
3.2.1 Alternatif-Alternatif yang Dirumuskan	9
3.2.2 Analisis Efisiensi dan Efektivitas dari Multistrategi Greedy yang Dirumuskan	9
3.2.3 Pemilihan Strategi Greedy yang Optimal	11
BAB IV	15
4.1 Implementasi Kode Program	15
4.2 Penjelasan Modularitas Program	21
4.3 Dokumentasi Beserta Deskripsi Pengujian Program	23
4.4 Analisis Strategi Algoritma yang Diusulkan	25
BAB V	26
DAFTAR REFERENSI	27
LAMPIRAN	28

BAB I

DESKRIPSI TUGAS

Galaxio adalah sebuah game battle royale yang mempertandingkan bot kapal anda dengan beberapa bot kapal yang lain. Setiap pemain akan memiliki sebuah bot kapal dan tujuan dari permainan adalah agar bot kapal anda yang tetap hidup hingga akhir permainan. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah. Agar dapat memenangkan pertandingan, setiap bot harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.



Gambar 1.1 Ilustrasi Visual Game Galaxio

Adapun permainan dapat diperoleh di tautan berikut:

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

Tujuan tugas besar ini adalah mengimplementasikan bot kapal dalam permainan Galaxio dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, program dapat diperoleh melalui starter-bots yang dapat dimodifikasi di dalam starter-pack pada laman berikut ini:

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

Untuk peraturan lengkap bagaimana permainan Galaxio ini bekerja, dapat dilihat di tautan berikut:

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

BAB II

DASAR TEORI

2.1. Algoritma Greedy

Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga, pada setiap langkah mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu (memilih solusi optimum lokal) tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan konvergen menuju solusi optimum global.

Adapun berikut elemen-elemen yang diperlukan dalam algoritma *greedy* adalah sebagai berikut:

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

2.2. Cara Kerja Program

Sebelum merancang program, beberapa prerequisites yang diperlukan sebagai berikut:

- a. Java (minimal Java 11): <https://www.oracle.com/java/technologies/downloads/#java>
- b. IntelliJ IDEA: <https://www.jetbrains.com/idea/>
- c. NodeJS: <https://nodejs.org/en/download/>
- d. Net Core 3.1: <https://dotnet.microsoft.com/en-us/download/dotnet/3.1>

Untuk menjalankan permainan, game engine yang digunakan dapat diunduh melalui <https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2> dan mengekstrak arsip *starter pack*. 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

```

3 komponen penting dalam memahami cara kerja game ini, yaitu :

1. Engine, komponen yang berperan dalam mengimplementasikan logic dan rules game.
2. Runner, komponen yang berperan dalam menggelar sebuah *match* serta menghubungkan bot dengan engine.
3. Logger, komponen yang berperan untuk mencatat *log* permainan sehingga kita 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 RecieveGameState 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.

BAB III

APLIKASI STRATEGI GREEDY

3.1. Dekomposisi dan Pemetaan Masalah dalam Permainan Galaxio Menjadi Upa Elemen Algoritma Greedy

Permainan Galaxio ini dapat dikomposisi secara umum menjadi tiga buah strategi dalam permainan: bertahan, menyerang, dan mencari makanan. Terdapat beberapa pendekatan heuristik selain tiga buah strategi tersebut, namun pada bot yang kami implementasikan akan menggunakan 3 buah strategi tersebut. Kami melakukan dekomposisi masalah menjadi upa elemen strategi *greedy* sebagai berikut:

3.1.1 Greedy by Growth Rate

Strategi ini merupakan pengembangan lanjutan dari strategi pertama, dengan melakukan seleksi terhadap objek *FOOD* atau *SUPERFOOD* yang memiliki bobot (*size increment*) berbeda dengan jarak (*euclidean distance*) yang berbeda pula. Pendekatan heuristik ini menyeleksi kandidat makanan yang memiliki *growth rate* (laju pertumbuhan) terbesar, dimana *growth rate* didefinisikan dengan membagi bobot dari *FOOD/SUPERFOOD* dengan jarak antara bot dan objek tersebut. Secara matematis, dapat diformulasikan sebagai berikut:

$$\frac{ds}{dt} = \frac{ds}{dx} \frac{dx}{dt} = \frac{ds}{dx} v$$

Dimana ds/dt merupakan laju pertumbuhan, ds/dx merupakan rasio antara perubahan ukuran yang dikarenakan objek tersebut dengan beda jarak, dan dx/dt atau v merupakan laju objek (atribut yang dapat diambil di gim Galaxio; bersifat konstan).

Adapun berikut elemen-elemen yang diperlukan dalam algoritma *Greedy by Growth Rate* adalah sebagai berikut:

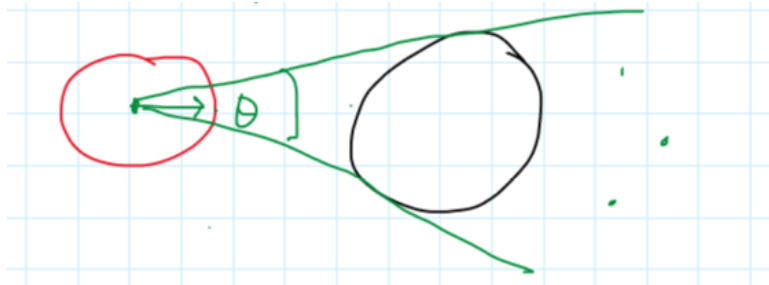
1. Himpunan kandidat : Objek *FOOD* maupun *SUPERFOOD*
2. Himpunan solusi: Himpunan *FOOD/SUPERFOOD* dengan bobot pertumbuhan terkecil
3. Fungsi solusi: Mengembalikan objek dengan nilai bobot pertumbuhan terkecil

4. Fungsi seleksi : Melakukan pemilihan seluruh objek bertipe *FOOD/SUPERFOOD*
5. Fungsi kelayakan : Melakukan pengurutan larik yang berisi objek tersebut berdasarkan bobot pertumbuhan terbesar
6. Fungsi obyektif : Memilih objek tersebut dengan bobot pertumbuhan terbesar.

3.1.2 Defensive Greedy

Strategi ini merupakan strategi bertahan yang dilakukan dengan cara menghindari bot lawan yang memiliki ukuran yang lebih besar dan juga menghindari rintangan dan objek-objek yang merugikan, seperti *gas cloud* yang dapat mengurangi ukuran kapal atau *asteroid field* yang dapat memperlambat kecepatan bot. Selain itu, kami juga menghindari penyusutan peta/map yang terjadi agar ukuran bot kami tidak tereduksi.

Strategi ini merupakan pengembangan dari strategi berdasarkan laju pertumbuhan tercepat. Mekanisme penghindaran dan pencarian makanan yang kami gunakan diilustrasikan sebagai berikut:



Gambar 3.1 Mekanisme Penghindaran(Sumber: Dokumentasi Penulis)

Dengan mengasumsikan bentuk objek adalah **lingkaran** dan tiap ukuran objek merupakan **diameter** lingkaran, sehingga mekanisme pencarian makanan dibatasi dengan batasan jangkauan sudut θ pada ilustrasi di atas.

Strategi ini juga dilengkapi dengan fitur-fitur perlindungan tambahan, diantaranya:

1. Akan mengaktifkan efek *shield* apabila ada objek torpedo di dekatnya

2. Melakukan penembakan torpedo atau *supernova bomb* apabila ada lawan di dekatnya sebagai mekanisme perlindungan
3. Jika lawan cukup dekat, arah dapat diubah dengan berbalik 180 derajat dari arah lawan yang mengejar kita

Adapun berikut elemen-elemen yang diperlukan dalam algoritma *Deffensive Greedy* adalah sebagai berikut:

1. Himpunan kandidat : Objek *FOOD* maupun *SUPERFOOD* atau arah(*heading*)
2. Himpunan solusi: Himpunan *FOOD/SUPERFOOD* dengan bobot pertumbuhan terkecil
3. Fungsi solusi: Mengembalikan arah dan/atau objek makanan yang memenuhi fungsi
4. Fungsi seleksi : Melakukan pemilihan seluruh objek bertipe *FOOD/SUPERFOOD* , *GASCLOUD*, *ASTEROIDFIELD*, *BOT*, dan *TORPEDO*
5. Fungsi kelayakan : Melakukan pemilihan objek yang patut diwaspadai dan dihindarkan berdasarkan batasan jarak yang ditentukan
6. Fungsi obyektif : Memilih sikap defensif yang paling mungkin diambil dengan mencari makanan pada arah yang tidak sama dengan arah objek yang diwaspadai atau mekanisme melarikan diri

3.1.3 *Offensive Greedy*

Strategi ini merupakan strategi di mana *greedy* yang bekerja untuk mengejar dan memakan bot lawan yang memiliki ukuran yang lebih kecil agar ukuran bot dapat bertambah menjadi lebih besar. Terdapat fitur-fitur tambahan yang telah diterapkan untuk mengoptimasi pengejaran bot lawan yang lebih kecil, seperti efek *AFTERBURNER* dan *TELEPORT*.

Adapun berikut elemen-elemen yang diperlukan dalam algoritma *Offensive Greedy* adalah sebagai berikut:

1. Himpunan kandidat : Bot lawan

2. Himpunan solusi: Bot lawan yang lebih kecil dengan jarak yang memenuhi untuk dikejar, efek dan aksi tambahan untuk mengoptimasi Bot
3. Fungsi solusi: Mengembalikan Bot lawan yang lebih kecil dengan jarak yang memenuhi untuk dikejar beserta fitur optimasi tambahan
4. Fungsi seleksi : Melakukan seleksi seluruh seleksi bot yang ada di peta dimana bot tersebut bukan dirinya sendiri
5. Fungsi kelayakan : Melakukan pemilihan bot lawan yang memiliki ukuran lebih kecil dengan jarak tertentu
6. Fungsi obyektif : Memilih bot lawan dengan ukuran terkecil dengan batasan jarak tertentu

3.2. Multialternatif Strategi Greedy

Karena strategi *greedy* diterapkan dengan prinsip heuristik, kami menyadari untuk mengusulkan multikemungkinan mengenai strategi *greedy* lain yang dianggap mangkus.

3.2.1 Alternatif-Alternatif yang Dirumuskan

Berikut merupakan eksplorasi kami terhadap berbagai alternatif strategi *greedy* yang tidak kami implementasikan namun sempat untuk diusulkan:

1. Greedy by Food Distance

Algoritma ini merupakan ide pertama yang kami pikirkan sebagai solusi dari permainan bot galaxio. Pengukur ini didasarkan mencari objek makanan tanpa memperhatikan bobotnya(*FOOD* maupun *SUPERFOOD*) yang terdekat dengan mengukur jarak antara 2 objek(*euclidian distance*).

2. Greedy by Optimum Path

Strategi ini merupakan strategi mencari makanan atau memakan bot lawan dengan mempertimbangkan jalur terpendek. Ketika bot sudah menemukan sasarannya, bot akan mencari jalur terpendek untuk mencapainya dan memakannya dengan memerhatikan *obstacle* yang ada di sekitar bot. Algoritma yang digunakan menggunakan pendekatan Algoritma Dijkstra, dimana jarak antara bot dengan makanan dapat direpresentasikan sebagai graf berbobot.

3. Greedy by Food Density

Strategi ini merupakan strategi di mana bot akan mencari area dengan kepadatan makanan yang tinggi, seperti daerah di mana terdapat banyak makanan yang berdekatan satu dengan lainnya dan akan memakan makanan tersebut sebanyak mungkin. Dengan memakan makanan di daerah dengan kepadatan makanan yang tinggi, bot dapat dengan cepat bertumbuh dan berukuran lebih besar. Caranya ialah dengan menyeleksi klaster-klaster makanan sesuai *heading*(arah) yang tersedia dalam derajat(sebanyak 360 arah) dan mengkalkulasi banyaknya *FOOD/SUPERFOOD* yang ada di klaster tersebut dibagi dengan luas klaster yang dipilih.

3.2.2 Analisis Efisiensi dan Efektivitas dari Multistrategi Greedy yang Dirumuskan

1. Analisis terhadap Greedy by Growth Rate

Strategi ini mengutamakan pertumbuhan dengan menerapkan seleksi terhadap Objek makanan (*FOOD* maupun *SUPERFOOD*) yang memiliki bobot (*size increment*) dan jarak (*euclidean distance*) berbeda. Namun, strategi ini memiliki resiko bot dapat menjadi sasaran bot lawan.

2. Analisis terhadap Defensive Greedy

Strategi tersebut berfungsi untuk membawa pergerakan bot untuk menghindari objek-objek yang merugikan sehingga pergerakannya dapat dilakukan dengan lebih efektif. Strategi ini juga berfungsi membawa pergerakan bot untuk menghindari bot lawan yang berbahaya sehingga bot lawan tidak akan memakan. Namun, jika bot hanya fokus dengan menghindari *obstacle*, bot tidak akan fokus untuk mencari makanan sehingga bot tidak efektif untuk memperbesar ukurannya.

3. Analisis terhadap Offensive Greedy

Strategi ini berfungsi untuk membawa pergerakan bot untuk terus mencari bot lawan yang berukuran lebih kecil yang mudah dimakan untuk tumbuh lebih besar. Namun, jika bot hanya fokus untuk mencari bot-bot lawan untuk dimakan, bot tidak akan fokus untuk menghindari *obstacle* yang ada sehingga berisiko bot diserang oleh bot-bot lawan yang berbahaya.

4. Analisis terhadap Greedy by Food Euclidean Distance

Strategi ini berfungsi untuk membawa pergerakan bot untuk terus bergerak dan mencari objek makanan (*FOOD* maupun *SUPERFOOD*) yang berada paling dekat dan memakan

makanan tersebut secepat mungkin untuk tumbuh lebih besar. Namun, jika bot hanya fokus pada tumbuh besar dengan memakan makanan sebanyak mungkin, bot bisa menjadi sasaran bot lawan yang memiliki ukuran lebih besar. Kompleksitas waktu dari algoritma ini adalah $O(n)$ untuk kasus rata-rata (tanpa mempertimbangkan pengurutan, diasumsikan memilih seluruh objek berjumlah n), di mana n adalah jumlah makanan dalam area permainan. Apabila setelah menyeleksi kandidat dan dilakukan pengurutan, kompleksitas waktunya menjadi $O(n \log(n))$.

Algoritma ini melakukan iterasi untuk setiap makanan untuk mencari makanan terdekat. Jadi, semakin banyak makanan dalam area permainan, semakin lama waktu yang dibutuhkan untuk menemukan makanan terdekat. Namun, kompleksitas waktu akan menjadi lebih baik ketika makanan semakin sedikit dan algoritma ini hanya memerlukan waktu yang konstan untuk memeriksa apakah bot sudah mencapai makanan terdekat atau belum.

5. Analisis terhadap Greedy by Optimum Path

Strategi ini berfungsi untuk menemukan jalur terbaik menuju tujuan dengan mempertimbangkan faktor seperti jarak, ukuran, dan posisi untuk mencapai tujuan. Namun, jika bot terlalu fokus mencari jalur terbaik, bot dapat menjadi abai terhadap bot lawan yang membahayakan sehingga dapat menjadi sasaran serangan bot lawan. Kompleksitas waktu dari algoritma ini tergantung pada kompleksitas dari fungsi untuk mencari tujuan, jarak, dan posisi, serta jumlah objek dalam fungsi yang menyatakan objek-objek dalam permainan. Jika kompleksitas dari fungsi-fungsi tersebut dianggap sebagai $O(n)$, maka kompleksitas dari algoritma ini adalah $O(n^2)$, karena algoritma ini melakukan perulangan sebanyak n kali, di mana n adalah jumlah objek dalam fungsi objek permainan. Hal ini didasarkan pada algoritma Dijkstra (*pathfinding algorithm*).

6. Analisis terhadap Greedy by Food Density

Strategi ini berfungsi untuk bot yang ingin tumbuh dengan cepat dan memperoleh keuntungan dari area dengan kepadatan makanan yang tinggi. Secara konseptual, algoritma ini bekerja dengan memindai seluruh peta dan menghitung jumlah makanan dalam area tertentu, lalu memilih area dengan jumlah makanan terbanyak untuk dijadikan target berikutnya. Bot akan bergerak menuju area tersebut dan memakan makanan di

sekitarnya. Setelah makanan habis, bot akan kembali memindai peta untuk mencari area berikutnya dengan kepadatan makanan tertinggi, dan begitu seterusnya. Kompleksitas waktu algoritma ini tergantung pada ukuran peta, yaitu jumlah sel yang ada dalam peta. Pada setiap langkah, algoritma melakukan pemindaian seluruh peta untuk menghitung jumlah makanan dalam setiap area, sehingga kompleksitas waktu algoritma ini adalah $O(n^2)$, di mana n adalah jumlah sel dalam peta. Namun, daerah dengan kepadatan makanan yang tinggi dapat menjadi tempat di mana banyak bot lawan berkumpul dan jika bot terlalu fokus pada memakan makanan, bot mungkin bisa menjadi sasaran dari bot lawan yang berukuran lebih besar.

3.2.3 Pemilihan Strategi Greedy yang Optimal

Kami memilih untuk mengkombinasikan tiga strategi greedy yang akan diimplementasikan dalam program kami, terurut berdasarkan prioritas implementasi, yaitu *defensive greedy*, *offensive greedy*, dan *grow weight greedy*.

Defensive greedy fokus pada strategi bertahan hidup dengan memprioritaskan menghindari objek-objek yang merugikan dan menghindari benturan langsung dengan bot lawan yang lebih besar serta berusaha untuk tetap bergerak agar tidak menjadi sasaran lawan dan membantu bot bertahan lama.

Offensive greedy fokus pada melakukan serangan dengan memakan bot lawan yang berukuran lebih kecil dan menghindari bot-bot lawan yang berukuran lebih besar. *Grow weight greedy* fokus pada urutan pengambilan makanan yang harus diambil terlebih dahulu berdasarkan beratnya. Sehingga dengan menggunakan kombinasi ini, bot dapat mempertahankan hidupnya, menyerang dengan efektif, dan tumbuh lebih cepat.

Kompleksitas waktu untuk kombinasi strategi tersebut bergantung pada bagaimana strategi-strategi tersebut diimplementasikan. Namun, secara umum(*average case*), kombinasi ini memiliki kompleksitas waktu yang lebih tinggi, yaitu $O(n \log n)$, di mana n adalah jumlah objek yang ada di sekitar pemain. Hal ini karena strategi *defensive greedy* dan *offensive greedy*

membutuhkan perhitungan jarak untuk menentukan apakah suatu objek bisa diserang atau tidak, sementara *grow weight greedy* membutuhkan perhitungan untuk menentukan urutan makanan mana yang harus terlebih dulu diambil berdasarkan beratnya.

Alasan kami memilih solusi *greedy by growth rate* dibandingkan algoritma pencarian rute dengan memodelkan himpunan makanan pada peta karena lebih mangkus dalam waktu komputasi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Kode Program

Kode program yang kami buat mengimplementasikan 3 strategi *greedy* yang diusulkan: *deffensive greedy* , *offensive greedy* , dan *greedy by growth rate*.

Berikut Merupakan Implementasi kode program kami untuk bagian implementasi strategi greedy

```
Program computeNextPlayerAction
// Kode program ada di file BotService.java
//

//Inisiasi Atribut
private GameObject bot;
private GameState

//Setter Getter diasumsikan sudah terdefinisi

procedure computeNextPlayerAction(PlayerAction pa) {
    /*Prioritas:
    1. Defensif
    2. Ofensif
    3. by Size Growth Rate
    */

    // Inisiasi playeraction default

    pa.action =PlayerActions.FORWARD;
    pa.heading = new Random().nextInt(360);

    //Dapatkan state terkini
    greed1.setGameState(this.gameState);
    greed2.setGameState(this.gameState);
    greed3.setGameState(this.gameState);

    greed1.setBot(this.bot);
    greed2.setBot(this.bot);
    greed3.setBot(this.bot);

    //KONDISIONAL UNTUK MEMILIH STRATEGI GREEDY BERDASARKAN
```


PRIORITAS

```
//=====//  
  
    IF (greed1.isDeffensive()) THEN{  
        greed1.updatePlayerAction(pa);  
    }  
    ELIF(greed2.isEnemyNear()) THEN{  
        greed2.updatePlayerAction(pa);  
    }  
    ELSE{  
        greed3.updatePlayerAction(pa);  
    }  
}
```

Program deffensiveGreedy

// Implementasi prosedur pada file greedyByDeff.java

```
public PlayerAction updatePlayerAction(PlayerAction  
playerAction){  
    if(playerAction.action == PlayerActions.STARTAFTERBURNER)  
playerAction.action = PlayerActions.STOPAFTERBURNER; //Saat  
defensif, harus mematikan  
    else playerAction.action = PlayerActions.FORWARD;  
//Default  
  
    var objList = gameState().getGameObjects()  
        .stream().filter(item ->  
item.getGameObjectType() == ObjectTypes.GASCLOUD ||  
item.getGameObjectType() == ObjectTypes.ASTEROIDFIELD)  
        .sorted(Comparator  
            .comparing(item ->  
getEdgeDistanceBetween(getBot(), item)))  
        .collect(Collectors.toList());  
  
    var BotList = gameState().getPlayerGameObjects()  
        .stream().filter(enemy -> enemy.getId()  
!= getBot().getId())  
        .sorted(Comparator  
            .comparing(item ->  
getEdgeDistanceBetween(getBot(), item)))  
        .collect(Collectors.toList());  
  
    var torpedoList = gameState().getGameObjects()  
        .stream().filter(item -> item.getGameObjectType()  
== ObjectTypes.TORPEDOSALVO)  
        .sorted(Comparator  
            .comparing(item ->
```

```

getEdgeDistanceBetween(getBot(), item))
    .collect(Collectors.toList());
    if (!BotList.isEmpty() && !objList.isEmpty()) {;

        /*
            Prioritas kondisi greedy: (Prioritas dari
pertama)
            1. Activate shield kalau ada torpedo,
tanpa mempertimbangkan keadaan sebelum/sesudah
            2. Menghindar dari lawan:
            2a. Tembak torpedo/supernova kalau punya
            2b. Greedy by Food tapi dengan constrain
enemy, default
            2c. Jika sangat dekat, langsung putar
balik 180. Kalau ada teleporter, bagus.
            3. Menghindar dari hazard objects; Greedy
by food
            4. Menjauhi map yang mengecil!

        */
        if(!torpedoList.isEmpty()){
            if(isTorpedoDangerous(torpedoList.get(0)) &&
getBot().getShieldCount() > 0 && !isTorpedoFired){
//isTorpedoFired berguna untuk mendeteksi apakah objek torpedo
yang ditembakkan adalah punya bot kita atau bukan
                playerAction.action =
PlayerActions.ACTIVATESHIELD;
            }
        }
        else if(ifNearEnemy(BotList.get(0))){
            //Dekat dengan lawan, default
            //Implementasi dari 2b. Greedy by Food, tapi
dengan constrain enemy, default

            //=====//
            this.enemyBot = BotList.get(0);
            double enemyCenterDistance =
getDistanceBetween(getBot(), this.enemyBot);

            int enemyCenterHeading =
getHeadingBetween(objList.get(0));
            int enemySize = objList.get(0).getSize();
            playerAction.heading =
getHeadingFoodWithConstraint(enemyCenterDistance,
enemyCenterHeading, enemySize);
            //=====//

            if(this.TorpedoTick -
getGameState().getWorld().getCurrentTick() > 3 &&

```

```

this.isTorpedoFired){
    this.isTorpedoFired = false;
}
else if(getBot().getTorpedoSalvoCount() > 0){
    playerAction.heading =
enemyCenterHeading;
    playerAction.action =
PlayerActions.FIRETORPEDOES;
    this.TorpedoTick =
getGameState().getWorld().getCurrentTick();
    this.isTorpedoFired = true;
}
else if(getBot().getSupernovaAvailable() > 0
&& !isSupernovaFired){
    playerAction.heading =
enemyCenterHeading;
    playerAction.action =
PlayerActions.FIRESUPERNOVA;
    this.SupernovaTick =
getGameState().getWorld().getCurrentTick();
    this.isSupernovaFired = true;
}
else if(this.isSupernovaFired &&
getGameState().getWorld().getCurrentTick() >
this.SupernovaTick + 3){
    playerAction.action =
PlayerActions.DETONATESUPERNOVA;
}
// Apabila dekat sekali, gunakan
enemydistancedanger. Lari ke arah yang berlawanan
else if(ifVeryNearEnemy(objList.get(0))){
    playerAction.action =
PlayerActions.STARTAFTERBURNER;
    playerAction.heading =
(enemyCenterHeading + 180) % 360;

    // Lakukan teleportasi apabila sempat
    if(getBot().TeleporterCount > 0 &&
!this.isTeleport && getBot().getSize() > 50){
        playerAction.action =
PlayerActions.FIRETELEPORT;
        this.TeleportTick =
getGameState().getWorld().getCurrentTick();
        this.isTeleport = true;
    }

    if(this.isTeleport &&
getGameState().getWorld().getCurrentTick()-TeleportTick > 5){
        playerAction.action =
PlayerActions.TELEPORT;
        this.isTeleport = false;
    }
}

```

```

        }

    }

    }
    else if(ifNearObj(objList.get(0))){ //Dekat dengan
hazardous object
        this.hazardousObject = objList.get(0);

        double objCenterDistance =
getDistanceBetween(getBot(), this.hazardousObject);

        int objCenterHeading =
getHeadingBetween(this.hazardousObject);

        int objSize = objList.get(0).getSize();

        playerAction.heading =
getHeadingFoodWithConstraint(objCenterDistance,
objCenterHeading, objSize);

    }
    else if(isNearEdge()){
        double x = getBot().getPosition().getX();
        double y = getBot().getPosition().getY();
        int angles = (toDegrees(Math.atan2(y,x)) +
180) % 180;
        playerAction.heading = angles;
    }

}
return playerAction;
}

```

Program offensiveGreedy

// Implementasi prosedur pada file greedyByOff.java

```

public PlayerAction updatePlayerAction(PlayerAction
playerAction){
    playerAction.action = PlayerActions.FORWARD;
    if (!getGameState().getGameObjects().isEmpty()) {
        var BotList = getGameState().getPlayerGameObjects()
        .stream().filter(enemy -> enemy.getId() !=
getBot().getId())
        .sorted(Comparator
        .comparing(item -> getDistanceBetween(getBot(),
item)))
    }
}

```

```

        .collect(Collectors.toList());

        this.nearestEnemyDistance =
getEdgeDistanceBetween(getBot(), BotList.get(0));
        int predictedSize = getBot().getSize() + (int) 0.5
* (getBot().getSpeed() - (int)
Math.sqrt(getBot().getSpeed()*getBot().getSpeed() +
4*getEdgeDistanceBetween(getBot(), BotList.get(0))));
        int deltaSize = getBot().getSize() -
BotList.get(0).getSize();
        int deltaPredictedSize = predictedSize -
BotList.get(0).getSize();
        Boolean validTeleportSize = getBot().getSize() >
50;

        //Activate brutal chasing mode
        if(this.nearestEnemyDistance > 15 &&
this.nearestEnemyDistance < 25 &&
getBot().getTeleporterCount() > 0 && deltaSize > 30 &&
!this.isTeleportFired && validTeleportSize){
            playerAction.action =
PlayerActions.FIRETELEPORT;
            this.isTeleportFired = true;
            this.tickTeleportFired =
getGameState().getWorld().getCurrentTick();
        }
        else if(this.isTeleportFired &&
getGameState().getWorld().getCurrentTick() -
this.tickTeleportFired > 5){
            this.isTeleportFired = false;
            playerAction.action = PlayerActions.TELEPORT;
        }

        else if(deltaPredictedSize > 3 ){ //Perhitungan
matematis sudah dibuktikan di kertas
            playerAction.action =
PlayerActions.STARTAFTERBURNER;
            this.bruteChaseMode = true;
        }
        else{
            this.bruteChaseMode = false;
        }

        playerAction.heading =
getHeadingBetween(BotList.get(0));
    }
    return playerAction;
}

```

Program GreedyByGrowthRate

```
// Implementasi prosedur pada file greedyByEW.java

public PlayerAction updatePlayerAction(PlayerAction
playerAction){
    if(playerAction.action == PlayerActions.STARTAFTERBURNER)
playerAction.action = PlayerActions.STOPAFTERBURNER; //Saat
defensif, harus mematikan
    else playerAction.action = PlayerActions.FORWARD;
//Default
    if (!getGameState().getGameObjects().isEmpty()) {
        var foodList = getGameState().getGameObjects()
            .stream().filter(item ->
item.getGameObjectType() == ObjectTypes.FOOD ||
item.getGameObjectType() == ObjectTypes.SUPERFOOD)
            .sorted(Comparator
                .comparing(item ->
item.getGameObjectType() == ObjectTypes.FOOD ?
getDistanceBetween(getBot(), item) :
getDistanceBetween(getBot(), item)/2))
            .collect(Collectors.toList());

        playerAction.heading =
getHeadingBetween(foodList.get(0));
        this.nearestFood = getDistanceBetween(getBot(),
foodList.get(0));
        this.typeFood =
foodList.get(0).getGameObjectType();
        this.theID = foodList.get(0).getId();
    }
    return playerAction;
}

public void debug(){
    System.out.println("Jarak Makanan Terdekat: " +
this.nearestFood);
    System.out.println("Jenis Makanan Terdekat: " +
this.typeFood);
    System.out.println("ID Makanan: " + this.theID);
}
}
```

4.2 Penjelasan Modularitas Program

Berikut adalah penjelasan tiap-tiap modul program yang kami usulkan:

No	Nama Kelas	Jenis Kelas	Deskripsi
----	------------	-------------	-----------

1	public class Main	-	Program utama untuk menjalankan permainan, disini dilakukan proses kompilasi program
2	enum Effects	Enum	Berisi berbagai efek yang dapat dialami player di permainan
3	public enum ObjectTypes	Enum	Mewakili jenis-jenis objek yang ada di dalam permainan
4	public enum PlayerActions	Enum	Mewakili aksi yang dapat dilakukan oleh pemain di dalam permainan
5	public class Position	Models	Merepresentasikan posisi atau lokasi suatu objek di dalam permainan
6	public class World	Models	Merepresentasikan dunia atau lingkungan permainan yang terdiri dari posisi serta radius dari dunia/peta pada saat tersebut
7	public class PlayerAction	Models	Merepresentasikan aksi yang dilakukan oleh pemain
8	public class GameObject	Models	Merepresentasikan objek-objek yang ada di dalam permainan
9	public class GameState	Models	Merepresentasikan keadaan atau status permainan saat ini. Kelas ini mengandung larik GameObject nonbot, world, dan larik Bot bertipe GameObject
10	public class GameStateDto	Models	Merupakan implementasi kelas yang mirip dengan GameState namun implementasinya sebagai umpan balik untuk mendapatkan keadaan permainan dari <i>game engine</i>
11	public class greedy	Greedy(Parent)	Mewakili algoritma greedy untuk melakukan

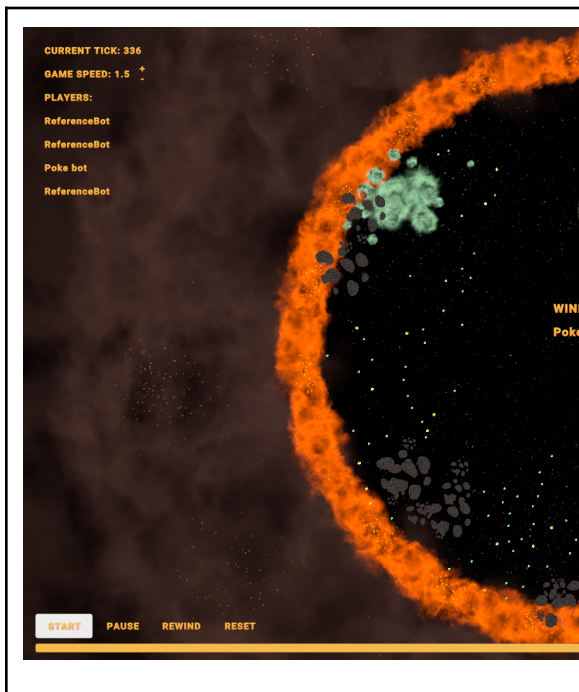
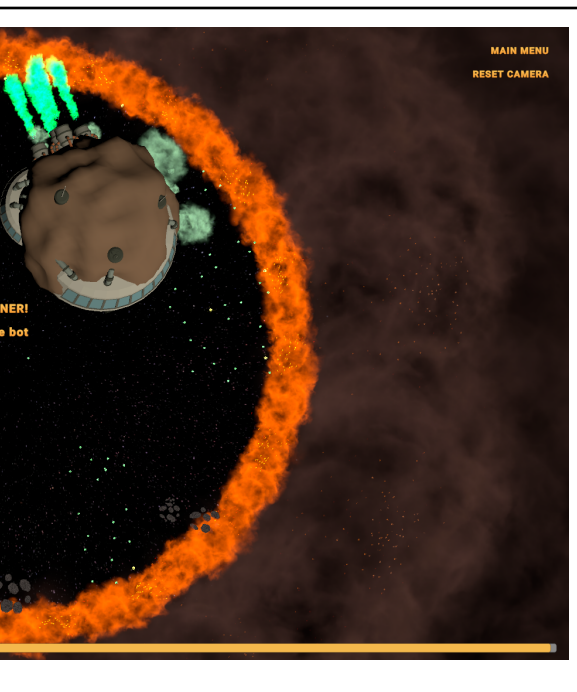
			optimisasi dalam permainan
12	public class greedyByDeff	Greedy	Mewakili algoritma greedy untuk melakukan optimisasi pada pertahanan dalam permainan, merupakan turunan dari kelas greedy.
13	public class greedyByOff	Greedy	Mewakili algoritma greedy untuk melakukan optimisasi pada serangan dalam permainan, merupakan turunan dari kelas <i>greedy</i> .
14	public class greedyByEW	Greedy	Mewakili algoritma greedy untuk melakukan optimisasi pada pencarian makanan berdasarkan <i>growth rate</i> tertinggi.
15	public class botService	Service	Merupakan kelas yang mengandung kondisi bot pada saat itu, kondisi game(gamestate) beserta PlayerAction. Pada versi modifikasi kami, kami menyertakan kelas greedy untuk mengembangkan bot berbasis algoritma greedy.

4.3 Dokumentasi Beserta Deskripsi Pengujian Program

- a) Proses debug dan run program

	
<p>Deskripsi</p>	<p>Melakukan running dan debugging untuk mengetahui kondisi bot berdasarkan gamestate beserta strategi <i>greedy</i> yang diterapkan di saat itu.</p>

b) Visualisasi

	
<p>Deskripsi</p>	<p>Melakukan Visualisasi terhadap bot program</p>

4.4 Analisis Strategi Algoritma yang Diusulkan

No	Nama Strategi	Analisis
1	<i>Defensive Greedy</i>	Defensive greedy optimal dalam menghindari objek-objek yang merugikan sehingga pergerakannya dapat dilakukan dengan lebih efektif. Strategi ini juga berfungsi membawa pergerakan bot untuk menghindari bot lawan yang berbahaya sehingga bot lawan tidak akan memakan. Namun, strategi ini tidak optimal dalam untuk mencari makanan sehingga bot tidak efektif untuk memperbesar ukurannya.
2	<i>Offensive Greedy</i>	Offensive greedy optimal jika digunakan untuk membawa pergerakan bot untuk terus mencari bot lawan yang berukuran lebih kecil yang mudah dimakan untuk tumbuh lebih besar. Namun, strategi ini kurang optimal untuk menghindari <i>obstacle</i> yang ada sehingga berisiko bot diserang oleh bot-bot lawan yang berbahaya.
3	<i>Greedy by Growth Rate</i>	Greedy by Growth optimal untuk membawa pergerakan bot untuk memakan makanan yang memiliki rasio bobot dengan jarak terbesar untuk secepat mungkin sehingga bot dapat tumbuh lebih besar. Keunggulan strategi ini ialah bot lebih fokus untuk mencari makanan terdekat dalam rangka meningkatkan ukurannya, namun rentan untuk diserang musuh.

BAB V

KESIMPULAN, SARAN, DAN REFLEKSI

Dari Tugas Besar I IF2211 Strategi Algoritma Semester II 2022/2023 berjudul *Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”*, kami mendapati bahwa untuk menciptakan bot dalam permainan *Galaxio* ini dapat dilakukan dengan menggunakan pendekatan strategi *greedy*. Alternatif strategi *greedy* juga tidak hanya satu, melainkan terdapat *defensive greedy*, *offensive greedy*, *greedy by growth rate*, *greedy by euclidean distance*, dan *dijkstra food greedy*.

Dari setiap alternatif strategi *greedy* pasti memiliki kelebihan dan kekurangannya masing-masing. Penyelesaian dengan strategi *greedy* dengan mengoptimalkan situasi pada setiap lingkup lokal tentunya dengan harapan pada lingkup global tercapai kondisi yang optimal. Akan tetapi, ekspektasi tersebut tidak selalu berhasil jika menggunakan strategi *greedy*. Ada kalanya strategi mencapai kondisi yang optimal dan tidak menutup kemungkinan juga jika strategi *greedy* tidak mencapai kondisi optimal.

Saran-saran yang dapat kami berikan untuk Tugas Besar I IF2211 Strategi Algoritma Semester II 2022/2023 adalah algoritma *greedy* yang digunakan dalam Tugas Besar ini masih dapat dikembangkan untuk kepentingan efisiensi program karena tentu masih terdapat kekurangan. Selain itu, untuk memudahkan programmer *me-maintenance* program yang ada, ada baiknya kode program dibuat lebih modular dan tersegmentasi dengan baik. Terakhir, program ini dapat dipublikasikan setelah dikembangkan lebih lanjut agar dapat bermanfaat menjadi referensi publik.

Setelah menyelesaikan Tugas Besar I IF2211 Strategi Algoritma Semester II 2022/2023, kami dapat merefleksikan bahwa komunikasi antaranggota kelompok berjalan cukup baik sehingga tidak terjadi miskomunikasi dan kesalahpahaman dalam pengerjaan tugas besar ini. Sebelum dimulainya pengerjaan tugas besar ini juga kami melakukan diskusi untuk membahas pembagian kerja untuk setiap anggota kelompok.

DAFTAR REFERENSI

Munir, Rinaldi. *Homepage Rinaldi Munir*. Diakses dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/>

Jana-Wessels dkk. Diakses dari
<https://github.com/EntelectChallenge/2021-Galaxio>

LAMPIRAN

Tautan repository tugas besar:

https://github.com/jejejery/Tubes1_greedland

Tautan Video:

<https://youtu.be/Zcibxm-xZHs>