# LAPORAN TUGAS BESAR

**IF2211 Strategi Algoritma**

**Kelompok Agario**


**Anggota Kelompok :**

**(13521050) Naufal Syifa Firdaus**

**(13521097) Shidqi Indy Izhari**

**(13521112) Rayhan Hanif Maulana Pradana**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

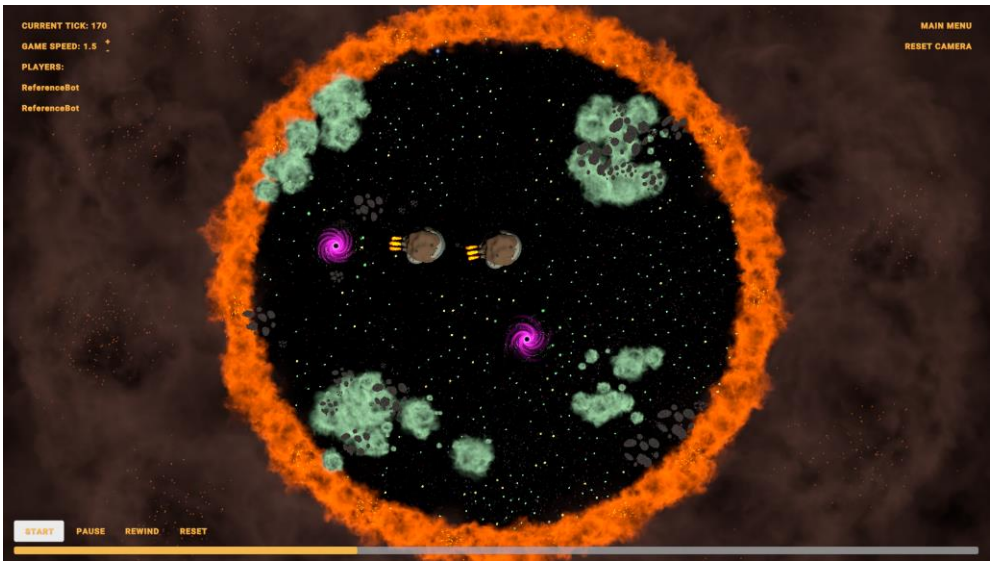
**2022**

# Daftar Isi

# Bab 1:
# Deskripsi Tugas

## 1.1 Deskripsi permainan Galaxio

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




Gambar 1. Ilustrasi permainan Galaxio

## 1.2 Aturan umum permainan Galaxio

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 maksimum 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.

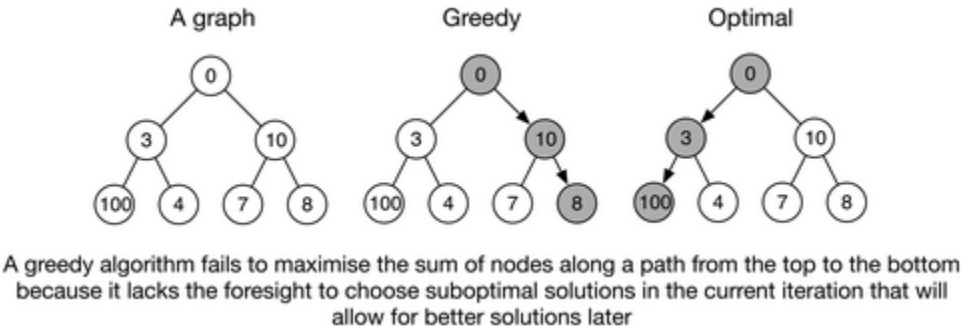
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player 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:
    1) FORWARD
    2) STOP
    3) START_AFTERBURNER
    4) STOP_AFTERBURNER
    5) FIRE_TORPEDOES
    6) FIRE_SUPERNOVA
    7) DETONATE_SUPERNOVA
    8) FIRE_TELEPORTER
    9) 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 2:
# Landasan Teori

## 2.1 Dasar Teori Algoritma Greedy

Jika ditinjau dari segi bahasa, greedy berarti serakah, tamak, atau rakus. Hal ini pun menjadi definisi utama dari algoritma ini. Algoritma dijalankan dengan solusi langkah per langkah dan untuk setiap langkah yang dijalankan, kita ingin mengambil keputusan yang paling optimal, atau dalam kasus ini besar nilainya (lokal maksimum), tanpa memikirkan konsekuensi pada langkah berikutnya. Kekurangannya, algoritma greedy tidak selalu menghasilkan solusi yang paling optimal karena ketidak perhatiannya pada langkah berikutnya. Untuk menggunakan algoritma greedy, terdapat syarat agar suatu persoalan dapat diselesaikan. Persoalan tersebut harus memiliki dua buah sifat berikut:

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




Gambar 2. Contoh penyelesaian algoritma greedy yang tidak menghasilkan solusi optimal
Sumber: https://besjournals.onlinelibrary.wiley.com/doi/10.1111/1365-2656.12963

Algoritma greedy memiliki beberapa komponen/elemen utama sebagai berikut:

1. Himpunan kandidat (C): Merupakan himpunan yang berisi kandidat yang mungkin dipilih pada setiap langkahnya.

2. Himpunan solusi (S): Merupakan himpunan yang berisi kandidat yang paling cocok dan terpilih menjadi solusi.
3. Fungsi solusi (*Solution function*): Merupakan sebuah fungsi yang menentukan apakah himpunan solusi S benar-benar dapat dijadikan solusi dengan domain himpunan objek dan range boolean.
4. Fungsi seleksi (*Selection function*): Merupakan sebuah fungsi yang memilih kandidat berdasarkan strategi dituju yang bersifat heuristik dengan domain himpunan objek dan range objek.
5. Fungsi kelayakan (*Feasibility function*): Merupakan sebuah fungsi untuk menguji kelayakan kandidat yang telah terseleksi oleh fungsi seleksi. Jika dirasa layak, kandidat akan dimasukkan ke dalam himpunan solusi S. Fungsi ini memiliki domain himpunan objek dan range boolean.
6. Fungsi objektif (*Objective function*): Adalah fungsi yang memaksimumkan atau meminimumkan suatu parameter dalam persoalan. Fungsi ini mempunyai domain himpunan objek dan range himpunan objek.

Terdapat berbagai permasalahan yang dapat dipecahkan dengan menggunakan algoritma greedy, berikut adalah daftar beberapa persoalan yang dapat diselesaikan menggunakan algoritma greedy:

1) Coin exchange problem
2) Activity selection problem
3) Minimasi waktu pada sistem
4) Knapsack problem
5) Job scheduling with deadlines
6) Minimum spanning tree
7) Shortest path
8) Huffman code
9) Egyptian fraction

### 2.2 Cara kerja program

Pada dasarnya, bot bekerja dengan meninjau seluruh kondisi yang sedang terjadi ketika permainan berlangsung dan akan menjalankan aksi berdasarkan pertimbangan terhadap kondisi tersebut. Berikut adalah daftar kondisi tersebut:

1. gameState: Detail dari permainan yang bersifat benda/objek dan peta.
    1) world: Arena permainan
        - ➔ centerPoint: Titik tengah pada arena permainan
            - x: Posisi pada sumbu x
            - y: Posisi pada sumbu y
        - ➔ radius: Jarak antara titik tengah hingga arena permainan terjauh
        - ➔ currentTick: Tick ketika permainan berlangsung
    2) gameObjects: List yang berisi seluruh objek dalam permainan selain bot
        - ➔ gameObject: Satu buah objek pada permainan
            - id: Nomor unik objek
            - size: Ukuran objek
            - speed: Kecepatan objek
            - currentHeading: Arah tuju objek
            - position: Posisi objek
            - gameObjectType: Tipe objek
            - effect: Efek pada objek
            - torpedoSalvoCount: Banyaknya torpedo
            - supernovaAvailable: Banyaknya supernova
            - teleportCount: Banyaknya teleport
            - shieldCount: Banyaknya shield
    3) playerGameObjects: List yang berisi seluruh bot yang sedang bermain pada arena
        - ➔ playerGameObject: Satu buah bot pada permainan
            - id: Nomor unik bot
            - size: Ukuran bot
            - speed: Kecepatan bot
            - currentHeading: Arah tuju bot
            - position: Posisi bot

- gameObjectType: Tipe bot
- effect: Efek pada bot
- torpedoSalvoCount: Banyaknya torpedo
- supernovaAvailable: Banyaknya supernova
- teleportCount: Banyaknya teleport
- shieldCount: Banyaknya shield

2. bot: Pemain yang aksinya dapat diubah
   - ➔ id: Nomor unik bot
   - ➔ size: Ukuran bot
   - ➔ speed: Kecepatan bot
   - ➔ currentHeading: Arah tuju bot
   - ➔ position: Posisi bot
   - ➔ gameObjectType: Tipe bot
   - ➔ effect: Efek pada bot
   - ➔ torpedoSalvoCount: Banyaknya torpedo
   - ➔ supernovaAvailable: Banyaknya supernova
   - ➔ teleportCount: Banyaknya teleport
   - ➔ shieldCount: Banyaknya shield
3. playerAction: Command yang akan dilakukan oleh bot
   - ➔ playerId: id bot yang bersangkutan
   - ➔ action: Aksi yang dipilih
     - Forward: Bot akan maju kedepan
     - Stop: Bot berhenti maju
     - StartAfterBurner: Bot menyalakan afterburner
     - StopAfterBurner: Bot mematikan afterburner
     - FireTorpedoes: Bot menembakkan torpedo
     - FireSupernova: Bot menembakkan supernova
     - DetonateSupernova: Bot meledakkan supernova
     - FireTeleport: Bot menembakan partikel teleport
     - Teleport: Bot berpindah pada partikel teleport

- ActivateShield: Bot menyalakan shield

➔ heading: Arah tuju bot ketika aksi dijalankan

Berdasarkan kondisi-kondisi di atas ini lah bot dapat menentukan aksi apa yang terbaik untuk dilakukan pada saat permainan berlangsung sehingga implementasi dari algoritma greedy dapat terwujud.

Untuk menjalankan permainan, dapat melakukan langkah-langkah sebagai berikut:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON ”appsettings.json” dalam folder “runner-publish” dan “engine-publish”
2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah “dotnet GameRunner.dll”
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah “dotnet Engine.dll”
6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah “dotnet Logger.dll”
8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON “GameStateLog_{Timestamp}” dalam folder “logger-publish”. Kedua file tersebut diantaranya GameComplete (hasil akhir dari permainan) dan proses dalam permainan tersebut.

# Bab 3:
# Aplikasi Strategi Greedy

### 3.1 Mapping persoalan algoritma greedy

Seperti yang telah dijelaskan sebelumnya, algoritma greedy memiliki beberapa komponen/elemen. Tujuan dari *mapping*/pemetaan ini adalah agar dapat memudahkan penulis untuk mengimplementasikan kode dan pembaca agar dapat memahami algoritma yang telah ditulis dengan lebih mudah.

Dalam *game* Galaxio, tujuan utama pemain adalah menjadi terakhir yang bertahan pada permainan agar menjadi pemenangnya. Terdapat bermacam-macam cara untuk meraih kemenangan tersebut. Pada pembuatan bot kali ini, penulis membagi strategi kedalam empat buah *state*: *general, grow, offensive,* dan *defensive*. Elemen dalam mapping strategi Greedy dalam masing-masing state adalah kumpulan aksi yang telah kami definisikan yang bisa bot lakukan pada suatu waktu. Berikut adalah penjelasannya:

1. Mapping persoalan algoritma greedy ketika bot dalam *state general*

   Secara normalnya, atau *by default,* bot harus memiliki beberapa standar dasar permainan agar tidak terjadi hal yang tidak diinginkan seperti bot yang keluar dari radius aman arena permainan. Selain itu, dalam state general harus memiliki aksi *default* ketika semua kondisi state lain tidak terpenuhi. Aksi *default* ini adalah mencari makan sehingga bot dapat bertumbuh besar dan mengurangi kemungkinan kematian.

| Nama Elemen/Komponen | Definisi Elemen/Komponen |
|---|---|
| Himpunan kandidat | Setiap langkah terdefinisi yang bisa diambil oleh Bot dalam satu waktu. |
| Himpunan solusi | Langkah yang membuat Bot membesar dan tidak keluar dari World. |
| Fungsi solusi | Pilih langkah yang dapat menjadikan Bot membesar dan tetap dalam World |
| Fungsi seleksi | Pilih langkah yang buat Bot membesar dan tetap dalam World |

| | |
|---|---|
| Fungsi kelayakan | Semua langkah dinilai layak |
| Fungsi objektif | Langkah dieksekusi dengan meminimalkan durasi dan jarak tempuh. |

2. Mapping persoalan algoritma greedy ketika bot dalam *state grow*

Untuk dapat memakan player lain, sesuai dengan peraturan pada permainan, bot harus memiliki ukuran yang lebih besar dari player lain yang akan dimakan. Grow merupakan salah satu strategi penting untuk dapat memenangkan permainan karena pada saat pertandingan baru dimulai, semua pemain akan memiliki ukuran yang sama sehingga perbedaan ukuran sangatlah krusial. State grow mengutamakan pengambilan supernova, superfood dan food.

| Nama Elemen/Komponen | Definisi Elemen/Komponen |
|---|---|
| Himpunan kandidat | Setiap langkah terdefinisi yang bisa diambil oleh Bot dalam satu waktu. |
| Himpunan solusi | Langkah yang membuat Bot membesar pada awal permainan dan menghindari *Gas Cloud*. |
| Fungsi solusi | Pilih langkah yang membuat Bot membesar dan menghindari *Gas Cloud*. |
| Fungsi seleksi | Pilih langkah yang paling menguntungkan dalam konteks grow. |
| Fungsi kelayakan | Langkah dinilai layak jika jarak mencukupi. |
| Fungsi objektif | Langkah yang diambil diberlakukan untuk objek dengan posisi paling dekat dengan Bot Agario. |

3. Mapping persoalan algoritma greedy ketika bot dalam *state offensive*

Algoritma greedy dalam state offensive memiliki tujuan untuk mengeliminasi pemain lain. Bot akan memasuki state offensive jika Bot lawan paling dekat dengan Bot Agario memiliki ukuran lebih besar dan permainan sudah berjalan cukup lama. Dalam offensive ada beberapa pilihan aksi dengan kondisinya sendiri yang jika terpenuhi dapat

dikatakan sebagai langkah penyelesaian yang paling baik sesuai saat ini. Jika lawan yang paling dekat jaraknya dapat ditempuh dengan

| Nama Elemen/Komponen | Definisi Elemen/Komponen |
|---|---|
| Himpunan kandidat | Setiap langkah terdefinisi yang bisa diambil oleh Bot dalam satu waktu. |
| Himpunan solusi | Langkah-langkah yang bertujuan untuk memusnahkan Bot pemain lain. |
| Fungsi solusi | Pilih langkah yang bertujuan untuk memusnahkan Bot pemain lain. |
| Fungsi seleksi | Pilih langkah yang tersedia sesuai kondisi Bot. |
| Fungsi kelayakan | Langkah dinilai layak jika ukuran dan posisi Bot memenuhi syarat. |
| Fungsi objektif | Langkah yang diambil diberlakukan untuk Bot pemain dengan posisi paling dekat dengan Bot Agario. |

4. Mapping persoalan algoritma greedy ketika bot dalam *state defensive*

Algoritma *greedy* memasuki state *defensive* apabila ukuran dari Bot Agario lebih kecil dibandingkan dengan ukuran dari Bot pemain lain yang terdekat, serta Bot pemain lain yang terdekat tersebut berada di dalam radius bahaya yang telah didefinisikan. Algoritma *defensive* utamanya bertujuan untuk menghindari, berlindung, atau melarikan diri dari ancaman Bot lain, serta melawan Bot terdekat jika kondisi memenuhi.

| Nama Elemen/Komponen | Definisi Elemen/Komponen |
|---|---|
| Himpunan kandidat | Setiap langkah terdefinisi yang bisa diambil oleh Bot dalam satu waktu. |
| Himpunan solusi | Langkah yang membuat bot melarikan diri/menghindar atau berlindung dan membela diri dari ancaman/bahaya. |
| Fungsi solusi | Pilih langkah yang bertujuan untuk |

| | |
|---|---|
| | melarikan diri atau berlindung dan membela diri dari bahaya. |
| Fungsi seleksi | Pilih langkah yang tersedia sesuai kondisi Bot. |
| Fungsi kelayakan | Langkah dinilai kayak jika ukuran dan posisi Bot memenuhi syarat. |
| Fungsi objektif | Langkah yang diambil diberlakukan untuk Bot pemain dengan posisi paling dekat dengan Bot Agario. |

### 3.2 Eksplorasi alternatif solusi greedy

Selain beberapa solusi yang telah disebutkan di atas, masih terdapat banyak sekali implementasi solusi greedy untuk memenangkan permainan. Hal ini mungkin terjadi karena beragamnya detail-detail menarik yang tersedia dalam game. Detail tersebut terdiri dari banyaknya command pada bot, arena permainan yang terus berubah, aksi bot/player lain yang tidak terduga, dan juga game state itu sendiri. Berikut merupakan beberapa alternatif solusi greedy yang telah kami rangkum:

1. Greedy by scoring

   Berdasarkan peraturan pada permainan, apabila terjadi kasus imbang di antara pemain, penentuan pemenang akan dihitung berdasarkan score tertinggi. Berikut adalah rincian dari perhitungan score:

   - Memakan pemain lain : 10 poin
   - Memakan makanan (food) : 1 poin
   - Menjelajahi wormhole : 1 poin

   Urutan prioritas dari algoritma ini, sesuai dengan banyakan poin yang didapat, adalah memakan pemain lain agar poin yang didapat lebih tinggi. Memakan pemain lain akan dilakukan hingga jumlah bot lain adalah satu. Setelah itu, prioritas utama dari bot pemain adalah mencari makanan sembari menghindari bahaya luar seperti serangan torpedo, serangan supernova, gas cloud, dan radius luar.

2. Greedy by distance

Strategi algoritma ini dapat terbilang sebagai strategi yang super defensive. Tujuan utama dari greedy by distance adalah menjaga jarak sejauh mungkin dengan pemain lain agar meminimalisir hal yang berbahaya bagi bot pemain seperti serangan supernova, serangan torpedo, dan risiko termakan oleh pemain lain. Sebagai tambahannya, algoritma juga dapat mengimplementasikan agar bot dapat berada pada jarak dekat radius, tetapi tidak keluar dari radius. Hal ini diperlukan karena strategi lain kemungkinan besar strategi pemain lain adalah menjauh dari radius. Maka dari itu, melalui penggunaan strategi ini, pemain seharusnya dapat menjadi bot terakhir yang bertahan pada permainan.

### 3.3 Analisis efisiensi dan efektifitas solusi greedy

1. Analisis efisiensi dan efektifitas solusi greedy ketika bot dalam ***state general***

   Dalam permainan *Galaxio,* posisi setiap makanan dan radius dari arena diberi tahu kepada Bot setiap ticknya. Dengan informasi tersebut efisiensi algoritma general untuk membesarkan Bot dan menjaga Bot agar tetap didalam *World* dapat dinilai sangat efisien. Informasi posisi tiap makanan dapat digunakan untuk mengkalkulasikan jarak tiap makanan dengan Bot untuk mendapatkan makanan yang paling dekat dan hasilnya pasti selalu optimal. Ini berarti Bot dapat memakan makanan yang paling dekat dalam waktu yang paling singkat pula.

   Jika dinilai dari segi efektifitas, solusi greedy ini kurang dapat mencapai tujuan utama dari permainan *Galaxio* yaitu menjadi pemain paling akhir yang hidup. Jika dilihat dari sisi pertumbuhan saja, sekilas algoritma ini masuk akal karena bertujuan untuk membesarkan Bot sebesar mungkin. Namun jika diperhitungkan adanya pemain lain yang akan menyerang Bot, algoritma ini tidak menangani hal tersebut sehingga kemungkinan Bot mati ditangan lawan sangat tinggi. Terlebih lagi dengan adanya *Cloud Gas* algoritma ini tidak memperhitungkan berkurangnya ukuran Bot oleh *Cloud Gas*.

2. Analisis efisiensi dan efektifitas solusi greedy ketika bot dalam ***state grow***

   Algoritma dalam state grow mengutamakan pertumbuhan Bot dan pengambilan item dalam permainan. Seperti pembahasan sebelumnya, setiap posisi item dalam permainan diberitahu kepada Bot sehingga lokasi dari tiga item terpenting dalam permainan juga dapat ditemukan. Dengan begitu Bot dapat secara efisien menentukan *food*

dan *superfood* paling dekat. Begitu juga dengan *supernova*, ketika *supernova* muncul ke permainan, Bot akan langsung mengarah untuk mengambilnya. Ini berakibat aksi Bot yang terhitung efisien dalam bertumbuh dan mengambil item.

Dari sisi efektifitas, algoritma grow sudah memperhitungkan objek-objek yang dapat merugikan Bot dan menghindarinya. Namun, ketika dihadapkan pada Bot pemain lain, grow tidak punya algoritma untuk mengatasi serangan dan kejaran Bot lawan sehingga kemungkinan Bot mati ditangan lawan masih tinggi.

3. Analisis efisiensi dan efektifitas solusi greedy ketika bot dalam ***state offensive***

   Bukan hanya posisi item dalam permainan yang diberi tahu, tapi juga posisi pemain lain dan juga ukurannya. Dengan informasi ini, algoritma offensive dapat menentukan Bot lawan terdekat. Lalu jika lawan tersebut ukurannya lebih kecil dari Bot Agario, Bot tersebut ditentukan sebagai target untuk diserang. Metode penargetan tersebut sudah dinilai cukup efisien untuk menentukan target lawan dengan mempertimbangkan jarak mereka, algoritma dapat mengeksekusi aksi dengan durasi seminimal mungkin. Selain itu, dengan memperhitungkan ukuran dan ketersediaan aksi Bot Agario, dapat mengkalkulasi apa aksi serangan yang paling mungkin dilakukan dengan sumber daya yang ada saat ini.

   Algoritma offensive memiliki solusi aktif untuk meraih hasil optimal yaitu menjadi Bot terakhir yang hidup dengan cara mengeliminasi pemain lain. Namun, jika hanya mengeliminasi pemain lain, tidak cukup untuk memenangkan pertandingan karena ada beberapa faktor yang menentukan keberhasilan eliminasi lawan. Beberapa diantaranya adalah ukuran Bot, kecepatan Bot, dan ketersediaan torpedo dan teleport. Algoritma dalam state offensive tidak mengatasi cara mengembangkan ukuran Bot sama sekali sehingga pada awal permainan dimana ukuran Bot sangat kecil, sudah pasti akan termakan oleh Bot lawan yang menggunakan algoritma greedy berdasarkan makanan. Selain itu, jika Bot lawan yang paling dekat memiliki ukuran lebih besar dan Bot Agario tetap menjalankan strategi offensive, sudah dapat dipastikan akan termakan Bot lawan.

4. Analisis efisiensi dan efektifitas solusi greedy ketika bot dalam ***state defensive***

   Algoritma dalam state defensive bertujuan untuk menghindari serangan dan kejaran Bot lawan. Seperti yang sudah disebutkan sebelumnya, posisi dan ukuran Bot lawan dapat diketahui setiap waktu. Dengan informasi itu, Bot secara efisien dapat memperhitungkan

aksi dan arah yang paling efisien untuk menghindari serangan lawan dalam waktu yang sesingkat mungkin.

Efektifitas algoritma dalam defensive mungkin memiliki kemungkinan terbesar untuk meraih solusi optimal yaitu menjadi pemain terakhir yang bertahan jika diaplikasikan sendiri. Menghindari serangan Bot lain dapat menyebabkan Bot lawan saling memakan sehingga tersisa Bot Agario. Namun, ketika Bot lawan tersisa satu, tanpa strategi grow atau offensive, Bot dapat dipastikan tak berdaya melawan Bot lawan mengingat kemampuannya yang hanya menghindar.

### 3.4 Strategi greedy yang dipilih

Strategi greedy yang dipakai sebagai algoritma utama untuk bot adalah penggabungan dari keempat strategi yang telah dijelaskan pada subbab 3.1. Penggabungan ini bertujuan utama untuk menangani seluruh kemungkinan yang mungkin saja terjadi ketika permainan dijalankan. Tentu, penggabungan ini memiliki skala prioritas pada setiap strateginya agar dapat menghasilkan solusi yang optimal. Prioritas paling tinggi adalah general state yang merupakan state dasar bagi bot yang selalu akan berjalan selama permainan berlangung. Selanjutnya adalah state grow karena pengumpulan komponen untuk perkembangan bot sangatlah penting, khususnya pada awal permainan. Dan yang terakhir adalah offensive dan defensive. Kedua state ini dapat berubah tergantung dengan kondisi yang sedang terjadi pada permainan.

# Bab 4:
# Implementasi dan Pengujian

## 4.1 Implementasi algoritma greedy pada bot

Pengimplementasian algoritma pada bot ditulis dengan bahasa pemrograman Java, yang merupakan salah satu dari beberapa bahasa pemrograman yang tersedia untuk permainan Galaxio. Secara spesifik, algoritma ditulis di dalam file BotService.java. Pada file tersebut, terdapat fungsi-fungsi yang mengatur bagaimana bot akan bekerja selama permainan berlangsung. Source code lengkap terlampir pada bagian lampiran di laporan ini.

## 4.2 Penjelasan struktur data

Kode utama dari permainan Galaxio menggunakan prinsip pemrograman berbasis objek dalam kodenya sehingga struktur data yang digunakan merupakan class dan enum. Terdapat beberapa class dan enum penting dalam kodenya yang mewakili objek-objek penting pada permainan, diantaranya:

1. class GameObject (GameObject.java)

   Sesuai namanya, kelas ini berisi dengan seluruh aspek benda dari permainan Galaxio. Aspek-aspek dari benda tersebut meliputi:

   1) id

      Merupakan nomor unik dari benda yang dimaksud.

   2) size

      Seluruh benda dalam permainan direpresentasikan dalam bentuk lingkaran dan besarnya ditentukan dengan radius.

   3) speed

      Adalah satuan kecepatan benda terkait yang berupa bilangan bulat.

   4) currentHeading

      Merupakan arah benda sedang menuju dalam satuan derajat, dengan jarak 0 hingga 359 derajat.

   5) position

      Posisi dalam permainan ini dimodelkan dalam bidang kartesian dengan titik tengah (0,0), terhubung dengan class Position.

6) gameObjectType

   Adalah tipe dari benda yang dimaksud, berupa kelas enumerasi ObjectTypes.

7) effect

   Adalah angka yang mewakili efek yang sedang terjadi pada benda tersebut.

8) torpedoSalvoCount

   Banyaknya angka torpedo salvo pada benda yang bersangkutan, berupa bilangan bulat.

9) supernovaAvailable

   Banyaknya supernova yang tersedia pada benda yang bersangkutan, berupa bilangan bulat.

10) teleportCount

    Banyaknya teleport yang dapat dipakai pada benda terkait, berupa bilangan bulat.

11) shieldCount

    Banyaknya shield yang dapat digunakan pada benda terkait, berupa bilangan bulat.

2. class GameState (GameState.java)

   Kelas GameState adalah kelas yang memuat seluruh state atau kondisi dari permainan. State tersebut terdiri dari arena permainan, list GameObject objek normal, dan list GameObject pemain yang detailnya sebagai berikut:

   1) world

      Merupakan peta atau arena permainan yang telah dideklarasikan dalam kelas World.

   2) List gameObject

      List ini berisi dengan seluruh gameObject yang ada pada permainan.

   3) List playerGameObject

      List ini berisi dengan seluruh bot pemain pada permainan.

3. class PlayerAction (PlayerAction.java)

   Kelas ini berisi tentang segala hal yang berkaitan dengan aksi yang akan dijalankan oleh bot yang meliputi hal-hal berikut:

   1) playerID

      Merupakan ID unik dari setiap pemain.

   2) action

Adalah hasil enumerasi dari kelas PlayerActions yang berisi tentang beragam aksi.

3) heading

Merupakan arah kemana bot sedang menuju, berukuran 0 hingga 359 derajat.

4. class Position (Position.java)

Kelas ini merupakan dasar dari arena permainan. Dengan menggunakan bidang kartesius, seluruh posisi dari objek dapat ditentukan. Bidang kartesius berpusat pada titik (0,0) dan memiliki dua sumbu, x dan y, yang berarah positif dan negatif. Komponen utama dari kelas ini adalah nilai x dan y yang berupa bilangan bulat.

5. class World (World.java)

World adalah komponen penting dalam permainan yang berisi hal-hal berikut:

1) centerPoint

centerPoint adalah kelas Position yang memuat titik tengah dari peta.

2) radius

Merupakan jarak dari ujung batas peta hingga titik tengah yang membatasi peta permainan. Apabila terdapat objek yang berada di luar radius, akan segera hilang secara otomatis.

3) currentTick

Unsur waktu dalam permainan Galaxio dihitung dengan satuan tick. currentTick adalah nilai tick pada saat state tertentu yang berupa bilangan bulat.

6. enum ObjectTypes (ObjectTypes.java)

Berikut adalah daftar enumerasi dari ObjectTypes:

1) Player : 1
2) Food : 2
3) Wormhole : 3
4) GasCloud : 4
5) AsteroidField : 5
6) TorpedoSalvo : 6
7) SuperFood : 7
8) SupernovaPickup : 8
9) SupernovaBomb : 9
10) Teleporter : 10

11) Shield : 11

7. enum PlayerActions (PlayerActions.java)

   Berikut adalah daftar enumerasi dari PlayerActions:

   1) Forward : 1
   2) Stop : 2
   3) StartAfterBurner : 3
   4) StopAfterBurner : 4
   5) FireTorpedoes : 5
   6) FireSupernova : 6
   7) DetonateSupernova : 7
   8) FireTeleport : 8
   9) Teleport : 9
   10) ActivateShield : 10

### 4.3 Analisis dan desain solusi algoritma greedy yang diimplementasikan

A. Penentuan State

State menentukan algoritma greedy apa yang akan diaplikasikan pada bot dalam satu waktu. Untuk menentukan state apa yang akan diaplikasikan pada bot, dibutuhkan algoritma penentu state berisi kondisional dengan syarat untuk setiap statenya. Kondisional untuk tiap state sebagai berikut:

1. Grow : Jika bot lawan paling dekat memiliki ukuran yang lebih besar dan bot sedang tidak berada pada radius projectile berbahaya.
2. Offensive : Jika bot lawan paling dekat memiliki ukuran yang lebih kecil.
3. Defensive : Jika bot lawan paling dekat memiliki ukuran yang lebih besar dan bot sedang berada pada radius projectile berbahaya.

Code snippet

```
private int setState(double safeRadiusPlayer, GameObject NearestPlayer, double
attackRadius){
```

```
/*  Menghasilkan state dari bot untuk memilih algoritma greedy yang dipakai.
    0. Grow
    1. Offensive = Nearest Player terdekat lebih kecil ukurannya
    2. Defensive = Nearest Player lebih besar dan ukurannya lebih besar
*/
    int state = 0;
    double Distance = getDistanceBetween(bot, NearestPlayer) - bot.getSize() -
NearestPlayer.getSize();

    if((bot.getSize() > NearestPlayer.getSize() && Distance < attackRadius) ||
bot.getSize() > 500){
        state = 1;
    }
    else {
        if(Distance <= safeRadiusPlayer){
            state = 2;
        }
    }

    if(gameState.getWorld().getCurrentTick() != null){
        if(gameState.getWorld().getCurrentTick() < 100){
            state = 0;
        }
    }
    return state;
}
```

B. State General

State ini merupakan sebuah state yang akan dipanggil selama keberjalanan permainan atau dengan kata lain, state ini terdapat pada state-state lainnya. State ini terdiri dari fungsi-fungsi yang akan dijelaskan pada snippet di bawah ini:

Code snippet

```
private int isGasCloudNear(double safeRadiusGasCloud){
    GameObject gas = findNearestObject(ObjectTypes.GasCloud);
    if((getDistanceBetween(bot, gas) - bot.getSize() - gas.getSize())
<= safeRadiusGasCloud){
        return 1;
    }
    else{
        return 0;
    }
```

```
}

//ubah heading jika ketemu dengan gascloud
private void rotateNearGas(int heading){ // bug
    GameObject gas = findNearestObject(ObjectTypes.GasCloud);
    int x = bot.getPosition().getX();
    int y = bot.getPosition().getY();
    int x2 = gas.getPosition().getX();
    int y2 = gas.getPosition().getY();
    double deltaX = x-x2;
    double deltaY = y-y2;
    double newheading = Math.atan2(deltaY, deltaX);
    newheading += Math.PI;
    playerAction.heading = (int) newheading;
}

private void setANewHeadingIfOutOfBound(int heading){
    // mengubah heading jika ketemu dengan bound/radius luar game
    int newheading = heading;
    double distance = Math.sqrt(Math.pow(bot.getPosition().getX(), 2)
+ Math.pow(bot.getPosition().getY(), 2));
    GameObject centerWorld = bot;
    Position zero = bot.getPosition();

    zero.setX(0); zero.setY(0);
    centerWorld.setPosition(zero);

    if(gameState.getWorld().getRadius() != null){
        if((int) distance + bot.getSize() + 10 =
gameState.getWorld().getRadius()){
            newheading = getHeadingBetween(centerWorld);

            System.out.println("PRIMARY: On The Edge, Change Heading
to: ");
            System.out.println(newheading);
        }
    }
```

C. State Grow

Pada state ini, bot akan mencari resources yang dibutuhkan agar bot dapat menjadi lebih kuat seperti food, superfood, dan supernova. Tujuan utama dari state ini adalah agar bot dapat menyeimbangi atau melebihi bot lain terlebih dahulu sebelum menyerang mereka.
Posisi dari ketiga buah komponen yang dicari akan dicari menggunakan fungsi findNearestObject. Kemudian, akan dilakukan pemilihan kondisional berdasarkan jarak antara ketiga objek yang dicari tersebut dengan bot pemain. Jarak yang paling dekat akan diambil sebagai kandidat untuk diambil. Tetapi, hal ini juga perlu melalui proses pengecekan apakah terdapat gas di dekat bot. Apabila aman, bot akan segera menuju objek yang dituju dan jika tidak, bot akan berputar menggunakan fungsi rotateNearGas. Fungsi dari objek yang diambil adalah:

1. Food
   Membuat size dari bot semakin besar.
2. Superfood
   Membuat size dari bot semakin besar, lebih efektif dari food.
3. Supernova
   Dapat dipakai di kemudian kesempatan pada state offensive untuk mengalahkan musuh.

Code snippet

```
GameObject nearestfood = findNearestObject(ObjectTypes.Food);
GameObject nearestsuperfood = findNearestObject(ObjectTypes.SuperFood);
GameObject nearestsupernova = findNearestObject(ObjectTypes.SupernovaPickup);
if(getDistanceBetween(bot, nearestsupernova) <= getDistanceBetween(bot,
nearestsuperfood) && nearestsupernova.getGameObjectType() ==
ObjectTypes.SupernovaPickup){
    if(isGasCloudNear(safeRadiusGasCloud)==0){
        setHeadingToNearest(ObjectTypes.SupernovaPickup);
        messageBot += " Action : Heading for picking a supernova";
    }
    else{
        rotateNearGas(playerAction.heading);
    }
}
else{
    if(getDistanceBetween(bot, nearestsuperfood) < getDistanceBetween(bot,
nearestfood)){
        if(isGasCloudNear(safeRadiusGasCloud)==0){
            setHeadingToNearest(ObjectTypes.SuperFood);
            messageBot += " Action : Heading for picking a superfood";
        }
        else{
            rotateNearGas(playerAction.heading);
        }
    }
    else{
        if(isGasCloudNear(safeRadiusGasCloud)==0){
```

```
            setHeadingToNearest(ObjectTypes.Food);
            messageBot += " Action : Heading for picking a regular food";
        }
        else{
            rotateNearGas(playerAction.heading);
        }
    }
}
```

D. State Offensive

State offensive adalah state di mana bot pemain lain ukurannya lebih kecil jika dibandingkan dengan bot kita. State ini dapat dibilang sebagai strategi utama dari bot untuk memenangkan permainan karena dapat menghilangkan lawan. State memiliki beberapa pilihan aksi yang dapat diambil oleh bot sesuai dengan kondisi saat ini. Kondisi beserta aksi yang dapat diambil adalah sebagai berikut:

1. Mengejar lawan dengan After Burner

   Bot akan mengejar bot lawan ketika jarak dari bot tersebut bisa ditempuh menggunakan afterburner dan ketika bot lawan terkejar, ukuran bot Agario yang tersisa masih lebih besar dari bot lawan.

2. Menembakan Torpedo

   Jika kecepatan lawan lebih kecil dari 15, maka lawan dinilai target yang sesuai untuk torpedo karena akan cukup sulit untuk menghindari torpedo yang kecepatannya 20.

3. Menembakan Teleport

   Teleport digunakan ketika bot lawan ukurannya lebih kecil dibanding bot Agario ketika ukurannya dikurangi 20 (harga menggunakan teleport). Cara berikut dinilai paling efektif dalam mengejar musuh dan memakannya ketika ukuran bot mencukupi.

Code snippet

```
playerAction.heading = getHeadingBetween(NearestPlayer);
int distance = (int) getDistanceBetween(bot, NearestPlayer) - bot.getSize() -
NearestPlayer.getSize();
```

```
if((bot.getSize() - distance / (bot.getSpeed())^2) > NearestPlayer.getSize()){
    // AFTER BURNER
    // akan menyala ketika size bot setelah memakai afterburner
    // dan sampai di bot lawan lebih besar dari bot lawan.
    playerAction.action = PlayerActions.StartAfterBurner;
    messageBot += " Action :Start AfterBurner";
}
if(NearestPlayer.getSpeed() < 15){
    fireTorpedo();
    messageBot += " Action :Fire Torpedoes";
}
if(NearestPlayer.getSize() < bot.getSize()-40){
    fireTeleport();
    messageBot += " Action :Fire Teleport";
}
```

E. State Defensive

State defensive adalah state alternatif dari state offensive dimana player terdekat dalam radius tertentu yang telah didefinisikan memiliki ukuran yang lebih besar dibanding bot kita, dan bot terdapat pada radius projectile yang berbahaya. State ini berfokus kepada *survival* agar bot dapat bertahan hingga bahaya/ancaman menghilang. Beberapa aksi yang diimplementasikan pada state ini antara lain:

1. Melarikan diri dari Bot lawan dengan atau tanpa After Burner

Bot akan berputar dalam arah yang ”menjauhi” posisi koordinat Bot lawan dan mengaktifkan efek After Burner apabila size dari bot lebih dari 5. Apabila tidak, Bot akan melarikan diri dengan aksi Forward. Apabila syarat terpenuhi, Bot juga dapat menembakkan dan mengaktifkan teleport untuk melarikan diri secara instan. Aksi ini dilakukan dengan batas jarak antara Bot kita dengan Bot lawan terdekat yang berbahaya yang telah didiefinisikan.

2. Menembakkan Torpedo kepada Bot lawan

Apabila Bot lawan masih “jauh”, yaitu berada di luar batas jarak yang telah dibahas pada poin pertama dalam pembahasan state defensive ini, Bot kita akan berputar menghadap ke posisi Bot lawan dan menembakkan torpedo salvo apabila syarat untuk pengaktifan aksi penembakan torpedo terpenuhi.

3. Mengaktifkan Shield apabila terdapat tembakan Torpedo dari Bot lawan

Apabila terdeteksi sebuah proyektil torpedo yang berada di dalam radius yang telah didefinisikan dari posisi Bot pemain kita, secara otomatis, apabila syarat terpenuhi, Bot akan mengaktifkan Shield.

4. Melarikan diri dari tembakan Supernova dengan atau tanpa AfterBurner

Bot akan berputar dalam arah yang ”menjauhi” proyektil supernova dan mengaktifkan efek After Burner apabila size dari bot lebih dari 5. Apabila tidak, Bot akan melarikan diri dengan aksi Forward. Apabila syarat terpenuhi, Bot juga dapat menembakkan dan mengaktifkan teleport untuk melarikan diri secara instan. Aksi ini dilakukan apabila terdeteksi proyektil supernova yang berada di dalam radius yang sudah ditentukan

5. Menghindar dari Gas Cloud

Aksi-aksi di atas hanya akan dilakukan apabila tidak terdapat Gas Cloud di dekat Bot kita. Jika terdapat Gas Cloud, Bot akan berputar dan berjalan untuk menghindar dari Gas Cloud tersebut.

+

Code snippet

```
GameObject nearestSupernovaBomb =
findNearestObject(ObjectTypes.SupernovaBomb);
int botHeading = getHeadingBetween(nearestSupernovaBomb) - 180;
//playerAction.heading = NearestPlayer.currentHeading;

shieldActivation(safeEnemyTorpedoRadius);

if(bot.getSize() < NearestPlayer.getSize() && getDistanceBetween(bot,
NearestPlayer) <= safeRadiusPlayer) {
    if(bot.getSize() > 5) {
        // Kabur menggunakan afterburner apabila syarat memenuhi
        playerAction.action = PlayerActions.StartAfterBurner;
        messageBot += " Action: AfterBurner Run from Player";
    }
    // Menembakkan teleport untuk kabur jika syarat memenuhi
    fireTeleport();

    if(bot.getSize() < NearestPlayer.getSize()) {
        // bot menembakkan torpedo ketika target lawan lebih besar
dari bot
        // tujuannya untuk mereduksi ukuran bot lawan agar bisa
dimakan.
        playerAction.heading = getHeadingBetween(NearestPlayer);
        fireTorpedo();
        messageBot += " Action: Fire Torpedo";
    }
```

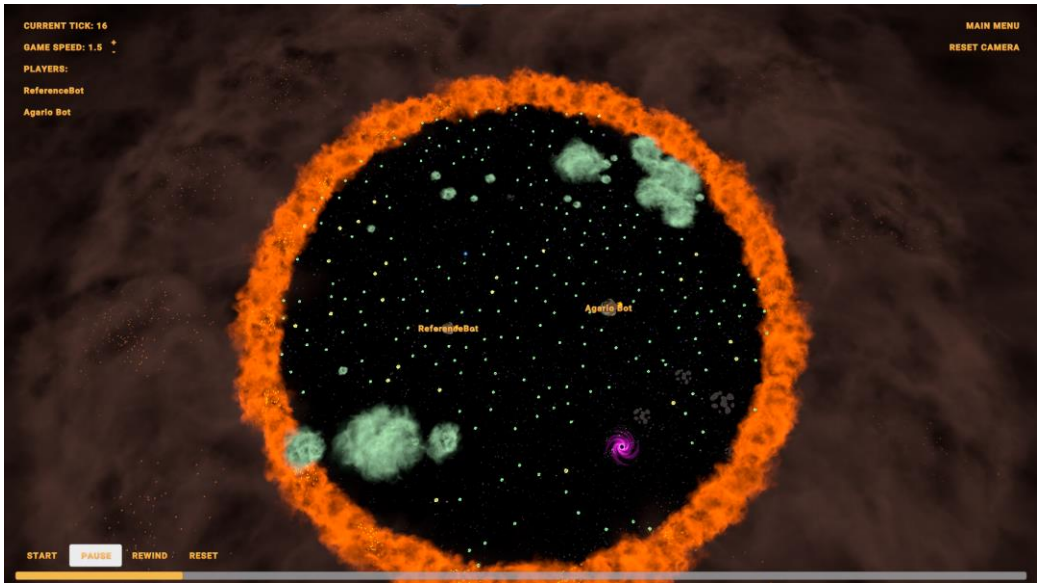
```
        escapeFromPlayerHeading(playerAction.heading);
        // Menembakkan teleport untuk kabur jika syarat memenuhi
        fireTeleport();

        if(bot.getSize() > 5) {
            // Kabur menggunakan afterburner apabila syarat memenuhi
            playerAction.action = PlayerActions.StartAfterBurner;
            messageBot += " Action:AfterBurner Run from Player";
        }
        if(getDistanceBetween(bot, nearestSupernovaBomb) <=
safeRadiusSupernova) {
            playerAction.heading =
nearestSupernovaBomb.currentHeading + 90;
            // Menembakkan teleport untuk kabur jika syarat memenuhi
            fireTeleport();
        if(bot.getSize() > 5) {
            // Kabur menggunakan afterburner apabila syarat memenuhi
            playerAction.action = PlayerActions.StartAfterBurner;
            messageBot += " Action:AfterBurner Run from Supernova";
            }
        }
    }
    else {
        rotateNearGas(playerAction.heading);
    }
escapeTeleport(safeRadiusPlayer);
```

### 4.4 Eksperimen

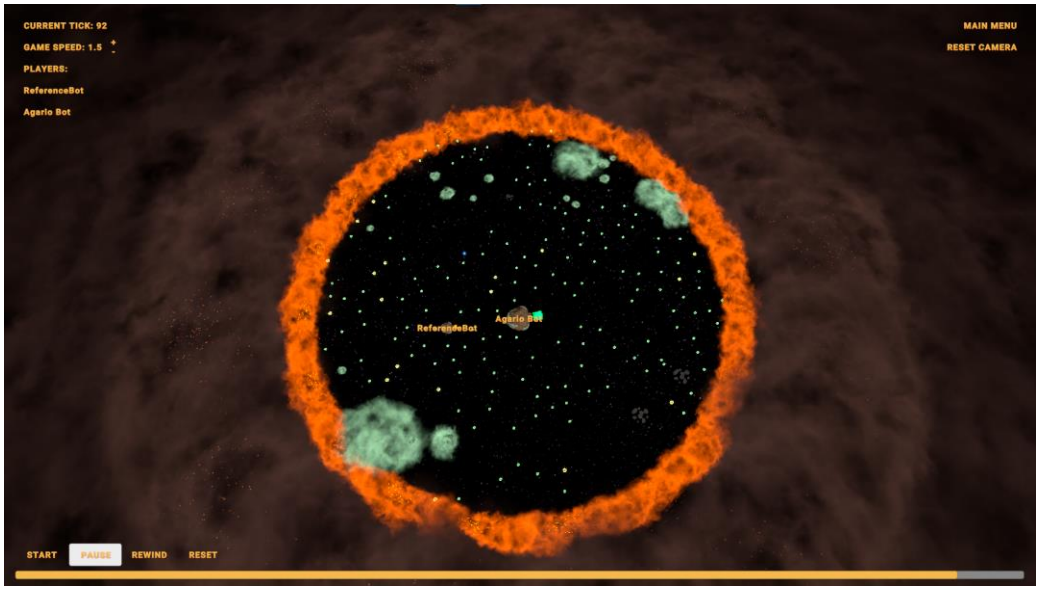
Untuk menguji strategi algoritma greedy dalam bot Agario dibutuhkan pengujian atas algoritmanya secara langsung. Pengujian bertujuan untuk mencari tahu apakah bot Agario dapat mencapai solusi paling optimal dalam permasalahan *Galaxio*. Eksperimen dilakukan sebanyak 3 kali dengan parameter keberhasilannya adalah bot Agario berhasil menjadi bot yang terakhir bertahan.

1. **Eksperimen Pertama, 1 Agario vs 1 Bot Referensi**

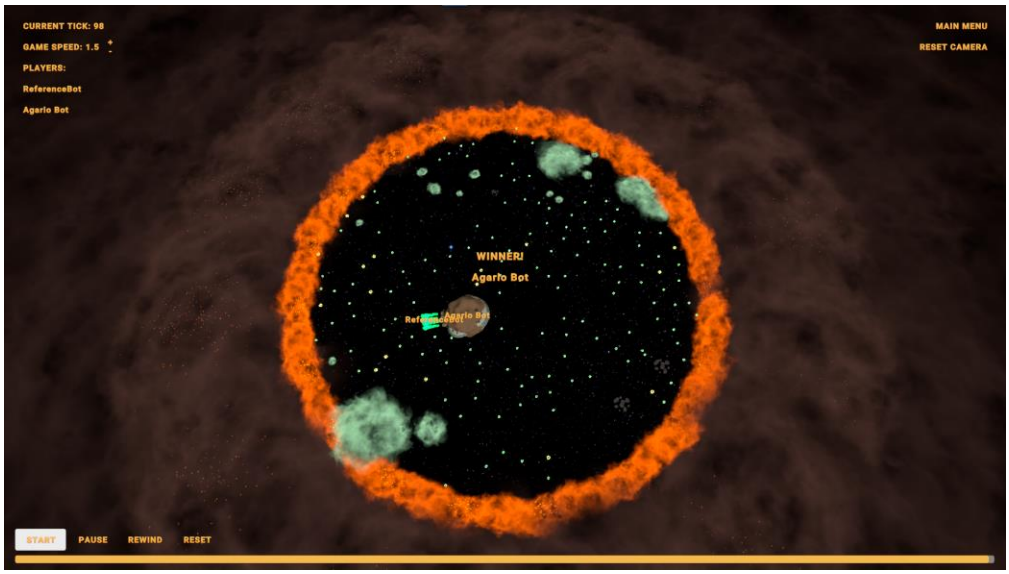
   Eksperimen berikut bertujuan untuk mencari tahu kemampuan bot Agario dalam mengeliminasi lawan secara satu lawan satu.




Dari hasil tangkap layar eksperimen, pertandingan dimulai dengan kedua bot berseberangan dalam arena permainan. Pada saat ini bot Agario masih dalam state grow dan masih mengumpulkan makan dan item lainnya.



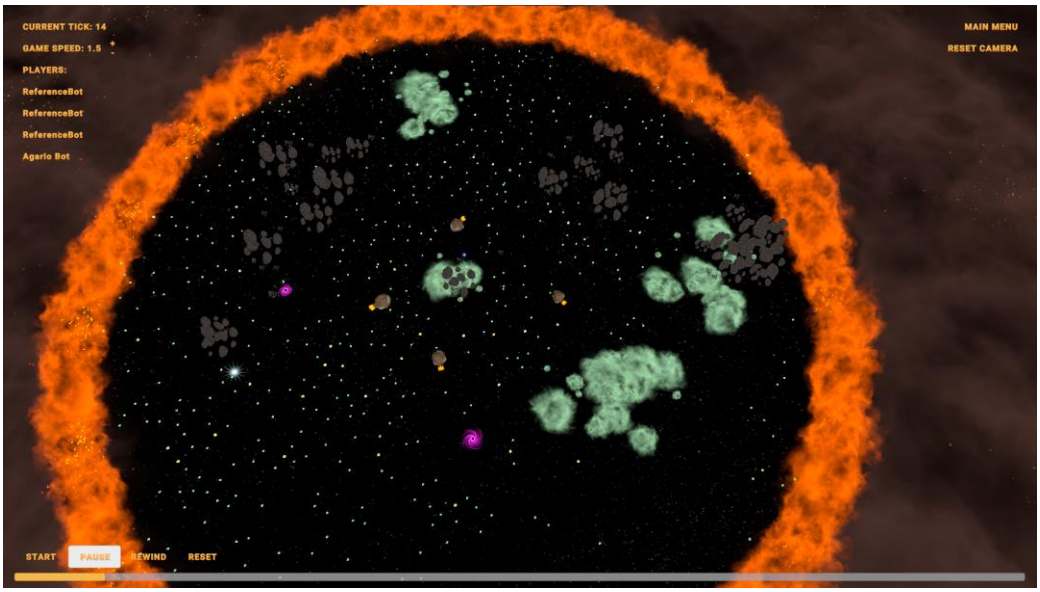

Kondisi diatas menggambarkan bot Agario yang telah memasuki state offensive dan sedang mengejar bot lawan.




Setelah beberapa saat, Agario mampu menangkap bot lawan dan memakannya menjadikan bot Agario pemenang terakhir. Dari eksperimen berikut solusi optimal dari permasalahan tercapai dengan baik.

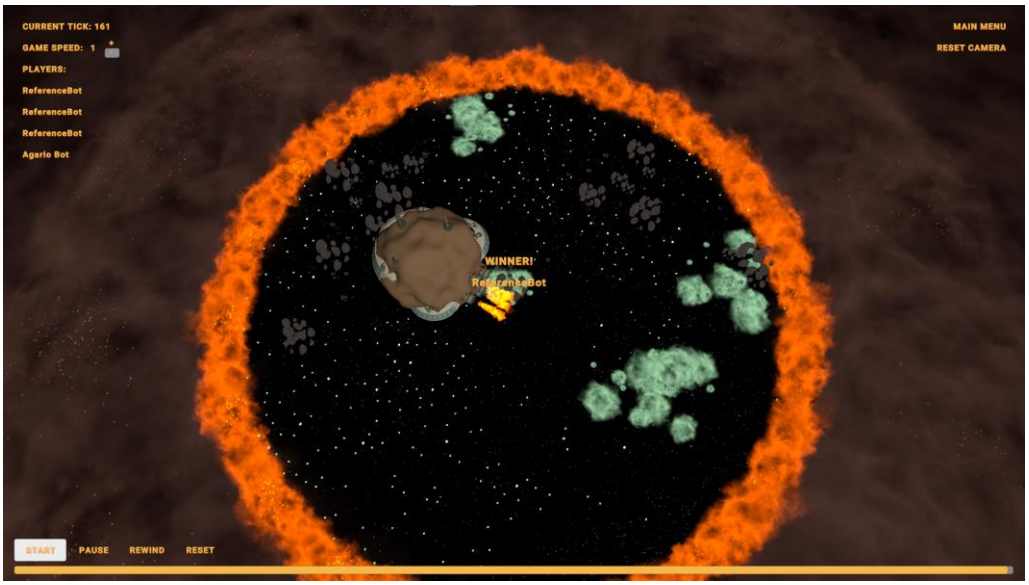
## 2. Eksperimen Kedua, 1 Agario vs 3 Reference Bot

Eksperimen berikut bertujuan untuk mengetahui algoritma greedy yang diterapkan apalah dapat mencapai solusi optimal dalam lingkungan yang seperti pertandingan sebenarnya (lawan beberapa bot sekaligus) walaupun bot yang dipakai masih bot referensi.




Dapat diamati diatas, keempat bot memulai di posisi yang serupa dengan jarak yang sama ke tengah arena.

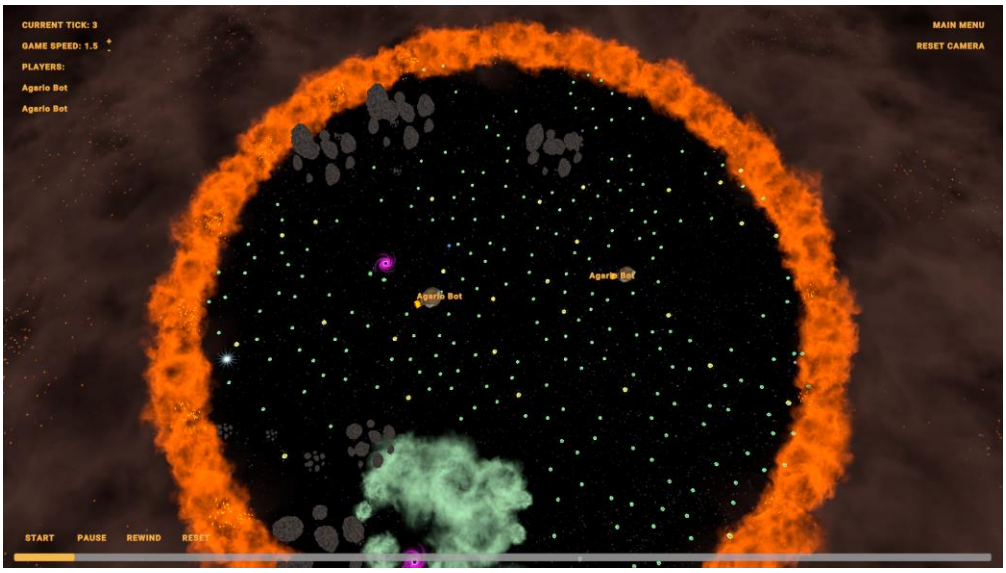

Pada saat ini bot Agario mampu untuk memakan dua bot referensi sekaligus menjadikan bot Agario berukuran besar.




Namun, sayangnya ketika bot Agario menargetkan bot referensi yang lain, bot tersebut mampu untuk mengumpulkan makanan sampai ukurannya lebih besar dari bot Agario. Bot referensi memakan bot Agario mengakibatkan solusi paling optimal dalam permasalahan tidak tercapai dalam eksperimen ini.

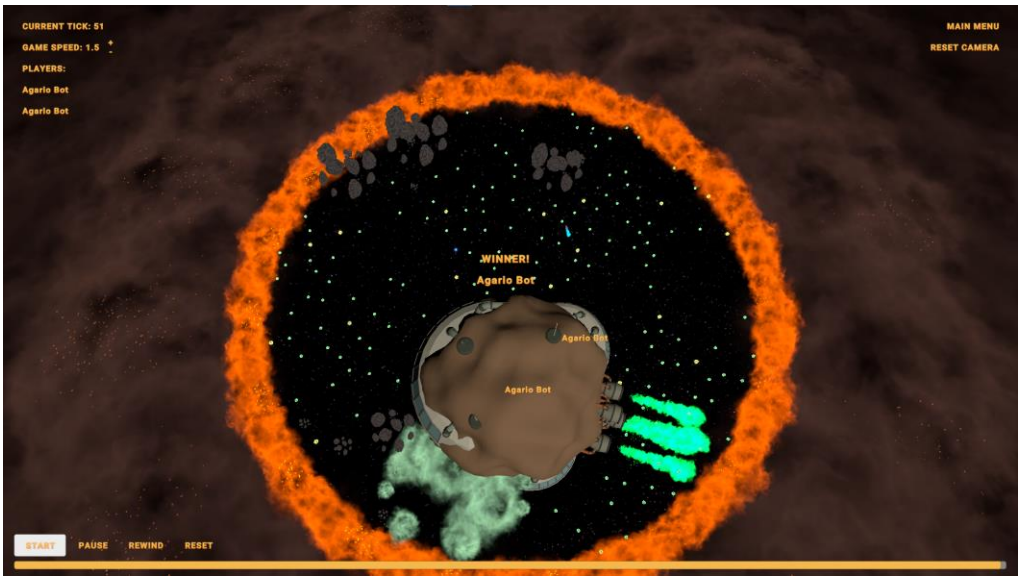
### 3. Eksperimen Ketiga, 2 Agario Melawan Satu Sama Lain

Eksperimen ini bertujuan untuk mencari tahu ketercapaian solusi optimal bagi Agario ketika dihadapkan dengan lawan yang memiliki algoritma yang mirip dengan dirinya sendiri.




Awalnya kedua bot masih dalam state grow sehingga keduanya mengumpulkan sebanyak mungkin makanan.


Salah satu bot Agario menyalakan shieldnya karena ditembaki oleh torpedo lawannya.

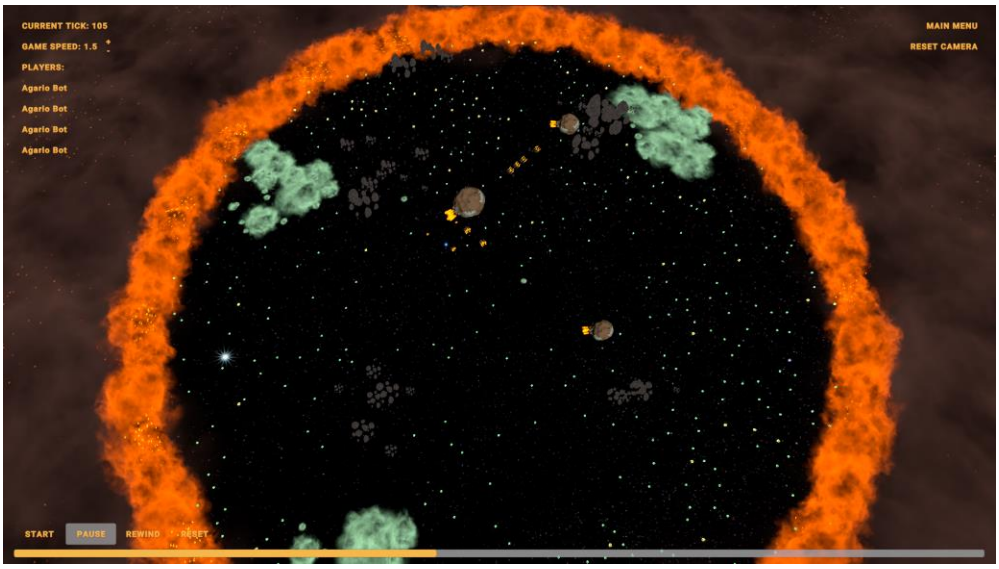



Sesudah shield habis, bot tersebut ternyata memiliki ukuran yang lebih besar daripada bot lainnya sehingga masuk kedalam state offensive, mengejar, dan menangkan bot lawan. Dalam eksperimen ini, bot Agario mencapai solusi optimal sekaligus tidak mencapai solusi optimal.

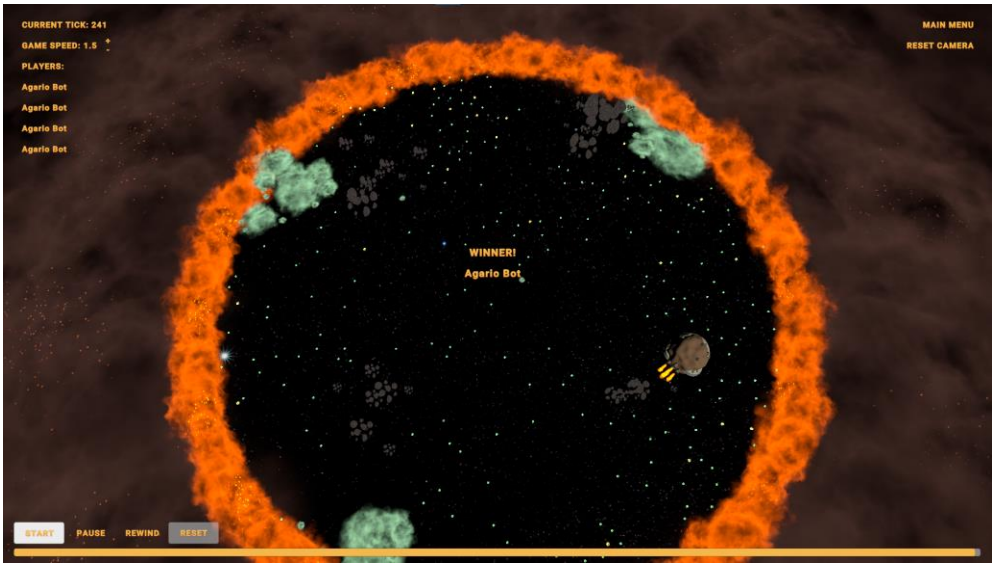
**4. Eksperimen Keempat, 4 Agario Melawan Satu Sama Lain**

Eksperimen ini bertujuan untuk mensimulasikan pertandingan *Galaxio* yang sebenarnya dengan bot-bot yang memiliki algoritma yang lebih komplek dibanding bot referensi.


Semua bot masih dalam state grow dan masih mencari makan untuk mendapatkan ukuran paling besar.




Satu bot Agario tereliminasi karena termakan lawannya. Sementara itu, salah satu bot yang letaknya ditengah ditarget oleh dua bot sekaligus dan ditembaki oleh torpedo. Karena bot tersebut tidak mempunyai shield, maka bot tersebut mati oleh tembakan torpedo.




Pada akhirnya, tersisa satu bot dikarenakan bot lawannya terjebak di antara asteroid dan gas cloud sehingga mati dengan sendirinya. Pada eksperimen inim, keberhasilan bot Agario beragam sesuai dengan urutan eliminasinya.

# Bab 5:
# Penutup

## 5.1 Kesimpulan

Pembuatan bot untuk permainan Galaxio dapat dilakukan menggunakan bahasa pemrograman Java dengan cara mengimplementasikan strategi algoritma greedy yang telah diajarkan dasarnya pada mata kuliah IF2211 - Strategi Algoritma. Bot bertujuan utama agar dapat bertahan paling lama dan menjadi pemenang dalam permainan tersebut. Setelah membuat pemrograman bot ini, dapat disimpulkan penggunaan algoritma greedy pada bot dapat terbilang cukup efektif untuk memenangkan permainan karena pengambilan keputusan terbaik pada setiap tick-nya merupakan metode yang terbaik bagi bot.

## 5.2 Saran

Kami sadar bahwasannya program kami bukanlah yang terbaik jika dibandingkan dengan banyak program lainnya di luar sana. Berikut adalah beberapa saran yang telah kami diskusikan dan kumpulkan untuk pembuatan program serupa kedepannya agar program bot dapat membuahkan hasil yang jauh lebih baik:

1. Membuat variasi greedy yang lebih beragam agar bot dapat leluasa untuk memilih aksi yang terbaik pada setiap state yang tersedia pada permainan.
2. Memperbanyak riset dengan cara menonton pertandingan Galaxio yang telah berlangsung, khususnya di kancah internasional agar mendapatkan ide-ide baru lainnya yang lebih efektif.

## 5.3 Refleksi

Pembuatan tugas besar mata kuliah IF2211 Strategi Algoritma ini merupakan pengalaman yang sangat berharga bagi penulis. Kami belajar bagaimana suatu pengimplementasian strategi algoritma dapat sangat berguna pada kehidupan sehari-hari, khususnya pembuatan bot yang berjalan secara otomatis. Meskipun demikian, penulis merasa masih terdapat beberapa kekurangan pada program bot sehingga penulis sangat berharap untuk pembuatan algoritma serupa kedepannya dapat dieksekusi dengan lebih baik.

# Daftar Pustaka


1. https://tugasanalgo4blog.wordpress.com/2016/12/04/algoritma-greedy/
2. https://besjournals.onlinelibrary.wiley.com/doi/10.1111/1365-2656.12963

# Lampiran

Link Repository GitHub : https://github.com/rayhanp1402/Tubes1_Agario

Link Video YouTube : https://youtu.be/6z3QYSaY1G8