Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II Tahun 2022/2023
Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Galaxio"

Kelompok 31 - GreedBot:

Ghazi Akmal Fauzan	13521058
Ilham Akbar	13521068
Ahmad Ghulam Ilham	13521118

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Tahun 2022/2023

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.

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Galaxio*. Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan *strategi greedy* untuk memenangkan permainan. 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 m8eledakannya 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.

BAB II
Landasan Teori

2.1 Algoritma Greedy Secara Umum

Algoritma greedy merupakan sebuah algoritma yang mengimplementasikan konsep "greedy" dalam pendefinisian solusinya. Algoritma greedy biasanya digunakan untuk mencari solusi dari persoalan optimisasi (optimization problem) untuk memaksimumkan atau meminimumkan suatu parameter. Algoritma greedy dibentuk dengan memecah permasalahan dan membentuk solusi secara langkah per langkah (step by step). Pada setiap langkah tersebut, terdapat banyak langkah yang dapat dievaluasi. Pada algoritma greedy, pemrogram harus menentukan keputusan terbaik pada setiap langkahnya. Tetapi, di dalam algoritma greedy tidak diperbolehkan adanya backtracking (tidak dapat melihat mundur ke solusi sebelumnya untuk menentukan solusi langkah sekarang). Oleh karena itu, diharapkan pemrogram memilih solusi yang merupakan optimum lokal (local optimum) pada setiap langkahnya. Hal ini bertujuan bahwa langkah-langkah optimum lokal tersebut mengarah pada solusi dengan optimum global (global optimum).

Suatu persoalan dapat diselesaikan dengan algoritma greedy apabila persoalan tersebut memiliki dua sifat berikut:

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

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

- Himpunan kandidat (C): Berisi kandidat yang mungkin dipilih pada setiap langkahnya.
- Himpunan solusi (S): Berisi kandidat yang sudah terpilih sebagai solusi.
- Fungsi solusi (solution function): Menentukan apakah himpunan solusi yang dikumpulkan sudah memberikan solusi. (Domain: himpunan objek, Range: boolean).
- Fungsi seleksi (selection function): Memilih kandidat berdasarkan strategi greedy tertentu. Fungsi ini memiliki sifat heuristik (fungsi dirancang untuk mencari solusi optimum dengan mengabaikan apakah fungsi tersebut terbukti paling optimum secara matematis). (Domain: himpunan objek, Range: objek).
- Fungsi kelayakan (feasibility function): Memeriksa apakah kandidat yang terpilih oleh fungsi seleksi dapat dimasukkan ke dalam himpunan solusi. (Domain: himpunan objek, Range: boolean).
- Fungsi objektif (objective function): Memaksimumkan atau meminimumkan suatu parameter pada suatu persoalan. (Domain: himpunan objek, Range: himpunan objek).

Skema umum algoritma greedy menggunakan pseudocode dengan pendefinisian elemen/komponennya adalah sebagai berikut:

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
   x : kandidat
   S : himpunan_solusi

Algoritma:
   S ← {}      { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
        x ← SELEKSI(C)     { pilih sebuah kandidat dari C}
        C ← C – {x}        { buang x dari C karena sudah dipilih }
        if LAYAK(S ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
            S ← S ∪ {x}     { masukkan x ke dalam himpunan solusi }
        endif
   endwhile
   {SOLUSI(S) or C = {} }

   if SOLUSI(S) then    { solusi sudah lengkap }
      return S
   else
     write('tidak ada solusi')
   endif
```

Gambar 1. Implementasi Algoritma Greedy

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. Untuk contoh dari hal ini kita dapat melihat Traveling Salesman Problem, dimana penggunaan algoritma greedy akan jauh lebih cepat dibandingkan dengan penggunaan brute force, walaupun solusi yang ditemukan biasanya hanya hampiran dari solusi optimal.

2.2 Cara Kerja Bot Permainan Galaxio

Setiap tick, pemain akan memasukkan satu command untuk bot. Semua command akan divalidasi terlebih dahulu sebelum dieksekusi oleh *game engine*. Jika command tidal valid, bot akan berada pada state yang sama. Command seluruh bot dilakukan pada waktu bersamaan.

Setiap bot dapat mengirimkan beberapa command dalam satu tick, tetapi hanya satu yang akan dieksekusi oleh *game engine*. Command diproses secara FIFO (First In, First Out). Pemain tidak perlu mengirim command setiap tick dan dapat menunggu kapan ingin mengirimkan command kepada bot miliknya. Berikut beberapa command yang dapat dilakukan oleh bot:

1. Command Forward
   Menggerakkan bot maju dengan arah sesuai heading yang ditentukan bot.
2. Command Stop
   Menghentikan aksi bot sebelumnya dan menunggu sampai command lain dikirim oleh pemain.
3. Command StartAfterburner
   Menghidupkan afterburner bot untuk menambah kecepatan bot. Efek afterburner dijelaskan lebih lengkap pada bagian Deskripsi Tugas.

4. Command StopAfterburner
   Mematikan afterburner bot.
5. Command FireTorpedoes
   Menggunakan 1 salvo charge untuk menembakkan torpedo dengan arah sesuai heading bot.
6. Command FireSupernova
   Menggunakan supernova yang sudah diambil dan menembakkannya kea rah heading bot.
7. Command DetonateSupernova
   Meledakkan supernova yang sudah ditembakkan.
8. Command FireTeleporter
   Menggunakan 1 teleport charge dan 20 size bot untuk menembakkan teleporter kea rah heading bot.
9. Command Teleport
   Melakukan teleport ke tempat teleport yang sudah ditembakkan.
10. Command UseShield
    Mengaktifkan shield untuk melindungi bot dari torpedo.

Pengaturan gerakan pada Bot dilakukan pada file BotService.java. Terdapat method computeNextPlayerAction yang berisi gerakan apa yang akan dilakukan oleh Bot. Gerakan yang dilakukan oleh Bot terdiri dari dua bagian, yaitu aksi yang dilakukan dan arah aksi. Aksi yang dilakukan bisa berupa maju, menembakkan torpedo, hingga mengaktifkan shield.

2.3 Implementasi Algoritma Greedy ke dalam Bot Permainan

Pada algoritma greedy, pemrograman tidak membutuhkan waktu dan resource yang banyak. Tetapi, dibutuhkan logika dan pola pikir masing-masing pemrogram untuk menentukan himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Himpunan dan fungsi tersebut mungkin tidak mencapai solusi optimum global dan hanya mencapai solusi optimum local.

Penulis menggunakan algoritma greedy sebagai algoritma pembentukan bot karena tidak memerlukan waktu dan resource yang banyak. Selain itu, cukup banyak kemungkinan solusi greedy yang dapat dieksplorasi. Terdapat peluang algoritma greedy tidak menghasilkan solusi optimum global, tetapi setidaknya penulis dapat mencapai solusi optimum lokal yang nilainya mendekati solusi optimum global.

2.4 Cara Menjalankan *Game Engine*

Requirement dan Instalasi

1. Java version 11 (https://www.oracle.com/java/technologies/downloads/#java8)
2. IntelIiJ IDEA (https://www.jetbrains.com/idea/)
   atau Apache Maven (https://maven.apache.org/download.cgi)
3. .Net Core 3.1 (https://dotnet.microsoft.com/en-us/download/dotnet/3.1)
4. Vs Code (https://code.visualstudio.com/Download)

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.

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

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON ”appsettings.json” dalam folder “runner-publish” dan “engine-publish”
2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah “dotnet 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 (Buka folder target dan jalankan perintah "Java -jar GreedBot.jar" untuk menjalankan bot ini)
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.

Cara menonton visualisasi hasil adalah sebagai berikut permainan

1. Ekstrak file zip Galaxio dalam folder “visualiser” sesuai dengan OS
2. Jalankan aplikasi Galaxio lalu Buka menu “Options”
3. Salin path folder “logger-publish” pada “Log Files Location”, lalu “Save”
4. Buka menu “Load”
5. Pilih file JSON yang ingin diload pada “Game Log”, lalu “Start”
6. Pilih start, pause, rewind, dan reset
7. Bermain!!

BAB III
Aplikasi Strategi Greedy

3.1 Mapping Persoalan Galaxio Menjadi Elemen-Elemen Algoritma Greedy

3.1.1 Pemetaan Elemen/Komponen Umum Algoritma Greedy

Pemenang pada permainan Galaxio ditentukan dari siapa yang bertahan paling lama atau mampu bertahan hingga akhir permainan. Tiap pemain memiliki satu bot kapal. Diperlukan strategi *greedy* untuk menentukan solusi paling optimal untuk memenangkan permainan. Berikut algoritma Greedy (himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan, fungsi objektif) :

Himpunan kandidat pada game Galaxio :

- Forward
- Stop
- StartAfterBurner
- StopAfterBurner
- FireTorpedoes
- FireSupernova
- DetonateSupernova
- FireTeleport
- Teleport
- ActivativeShield

Untuk mengisi himpunan solusi, diperlukan fungsi solusi yang menentukan apakah tindakan yang diambil oleh kapal dalam permainan Galaxio merupakan tindakan paling optimal untuk memenangkan permainan.

Fungsi seleksi adalah fungsi yang menentukan tindakan terbaik dalam setiap kondisi dalam permainan Galaxio. Dalam game ini, pilihlah aksi yang membuat kapal bertahan lebih lama terlebih dahulu ketimbang menyerang. Dalam game ini, fungsi kelayakannya adalah *command* valid seperti

Fungsi objektif adalah fungsi yang menentukan apa tujuan akhir dari permainan Galaxio. Fungsi objektif dalam permainan Galaxio adalah bertahan paling lama dalam permainan untuk memenagkan permainan.

Berikut Pemetaan Strategi *Greedy* yang digunaka :

- Himpunan solusi :

  Command yang valid dan bot kapal berjalan sesuai perintah dan aturan untuk memenangkan permainan.
- Fungsi solusi :

  Memeriksa apakah aksi yang dilakukan kapal tidak membahayakan kapal dari risiko mati sebelum permainan berakhir.
- Fungsi seleksi :

  Pilihlah aksi yang meningkatkan ukuran kapal serta menghindarkan kapal dari penyusutan ukuran.

- Fungsi kelayakan :

  Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid. Daftar command yang valid telah dipaparkan pada bab sebelumnya.
- Fungsi objektif :

  Memenangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup.

3.1.2 Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan Greed by Food

Salah satu subpermasalahan dalam permainan Galaxio adalah adanya Greed by Food. Bot akan fokus untuk mencari makan tanpa memedulikan sekitar sehingga memungkinkan untuk ditembak oleh bot lain.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari command Forward
Himpunan solusi	Kemungkinan permutasi dari command yang membuat bot bergerak maju mencari makan terdekat
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot maju mencari makan terdekat
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid.
Fungsi objektif	Mencari permutasi dari command yang membuat bot bergerak maju mencari makan terdekat

Tabel 1. Algoritma Greedy pada Permasalahan Greed by Food

3.1.3 Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan Greed by Defending

Salah satu subpermasalahan dalam permainan Galaxio adalah adanya Greed by Defending. Bot akan fokus untuk mengaktifkan shield secara terus menerus tanpa memedulikan ditembak atau tidak sehingga menyebabkan bot mati.

Nama Elemen/Komponen	Definisi Elemen/Komponen

Himpunan kandidat	Permutasi dari commanfd Activate Shield
Himpunan solusi	Kemungkinan permutasi dari command yang membuat bot mangaktifkan shield terus menerus
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot mangaktifkan shield terus menerus
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid.
Fungsi objektif	Mencari permutasi dari command yang membuat bot bergerak mangaktifkan shield terus menerus

Tabel 2. Algoritma Greedy pada Permasalahan Greed by Defending

3.1.4 Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan Greed by Speed

Salah satu subpermasalahan dalam permainan Galaxio adalah adanya Greed by Speed. Bot akan terus menerus mengaktifkan afterburner yang menambah kecepatan sehingga kemungkinan mati karena kehabisan nyawa

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari command Forward , StartAfterBurner
Himpunan solusi	Kemungkinan permutasi dari command yang membuat bot mangaktifkan afterburner terus menerus
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot mangaktifkan afterburner terus menerus
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid.
Fungsi objektif	Mencari permutasi dari command yang membuat bot bergerak mangaktifkan afterburner terus menerus

Tabel 3. Algoritma Greedy pada Permasalahan Greed by Speed

3.1.5 Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan Greed by Weapon

Salah satu subpermasalahan dalam permainan Galaxio adalah adanya Greed by Speed. Bot akan terus menerus menembak meskipun tidak ada musuh yang menembak sehingga akan menguragi nyawa bot dan akan mati

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Permutasi dari command FireTorpedoes, FireSupernova, FireTeleport
Himpunan solusi	Kemungkinan permutasi dari command yang membuat bot mangaktifkan FireTorpedoes, FireSupernova, FireTeleport terus menerus
Fungsi solusi	Melakukan pengecekkan apakah permutasi dari command tersebut membuat bot mangaktifkan FireTorpedoes, FireSupernova, FireTeleport terus menerus
Fungsi seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti.
Fungsi kelayakan	Memeriksa apakah command yang dituliskan oleh bot merupakan command yang valid.
Fungsi objektif	Mencari permutasi dari command yang membuat bot bergerak mangaktifkan FireTorpedoes, FireSupernova, FireTeleport terus menerus

Tabel 4. Algoritma Greedy pada Permasalahan Greed by Weapon

3.2 Explorasi Altenatif Solusi Greedy

1. Greedy by Food

Strategi *greedy by food* mengutamakan menambah ukuran kapal dengan memakan *food*, *superfood*, atau kapal lawan yang lebih besar. Khusus *superfood*, kapal akan mengonsumsi makanan dengan nilainya menjadi dua kali lipat ukuran makanan biasa selama 5 ticks.

- Berdasarkan efeknya, Superfood memiliki prioritas lebih tinggi dibandingkan dengan Food.
- Jika terdapat Superfood atau Food di sekitar bot, Gerakan akan mendekati Superfood dan Food tersebut.

2. Greedy by Defending

Strategi *greedy by defending* mengutamakan bagaimana ukuran kapal tidak mengecil untuk mengurangi risiko mati sebelum permainan selesai. Kapal akan memilih untuk menghindari kapal musuh yang berpotensi memakan kapal sendiri, supernova dari lawan, torpedo dari lawan, *gas cloud*, dan radius. Hal ini dilakukan untuk mengurangi risiko kalah karena ukuran kapal yang mengecil.

- Bot perlu kemampuan untuk menghindari dari Player yang berukuran lebih besar
- Bot perlu menghindar dari supernova dan torpedo milik lawan
- Bot perlu menghindar dari jangkauan Gas Cloud
- Bot perlu tetap berada dalam radius World agar tidak terlempar keluar
- Bot dapat mengatifkan shield untuk melindungi dari Torpedo Player terdekat

3. Greedy by Speed

Greedy by speed mengedepankan kecepatan kapal dalam permainan memanfaatkan *command* AfterBurner. Keunggulannya, kapal akan lebih cepat menjangkau food, bahkan superfood dan mampu mengelak lebih cepat dari serangan yang membahayakan kapal. Namun, hal ini berisiko mengurangi ukuran kapal yang bisa membuat kapal mati sebelum permainan selesai.

- Bot dapat mengaktifkan Afterburner untuk mencapai target lebih cepat
- Bot dapat mengejar Player terdekat yang memiliki ukuran lebih kecil disbanding bot
- Bot dapat memanfaatkan Afterburner untuk menjauh dari Player dengan ukuran yang lebih besar dari bot.
- Bot dapat menentukan kapan Afterburner efektif untuk dinyalakan dan kapan perlu dimatikan.
- Bot dapat menghindari Asteroid agar tidak mengurangi kecepatan bot.

4. Greedy by Weapon

Greedy by weapon memfokuskan pada pemilihan senjata yang tepat seperti supernova juga dengan mempertimbangkan kondisi-kondisi. Kondisi yang dimaksud di antaranya adalah size kapal sendiri. Risiko rugi akan lebih jika mengeluarkan senjata namun size kapal sendiri tidak terlalu besar.

- Bot dapat menembakkan Torpedo ke arah Player terdekat
- Bot dapat mencari Supernova jika tersedia pada Map
- Bot dapat menembakkan dan meledakkan Supernova pada kondisi terdekat
- Bot dapat menembakkan dan menggunakan Teleporter dalam kondisi-kondisi tertentu

3.3 Analisis Efisiensi dan Efektivitas Kumpulan Alternatif Solusi Greedy

Pada permainan Galaxio, penulis mengimplementasikan penggunaan sorted list untuk menyimpan berbagai kondisi GameObject pada GameState tertentu. Dengan begitu, bot akan menyimpan informasi nearestPlayer, nearestFood, nearestSuperFood, nearestOpponent, dan nearestGasCloud. Dengan begitu, dengan asumsi proses sorting menggunakan quicksort, kompleksitas waktu pemilihan struktur data adalah O(log n).

Kemudian, dari seluruh sorted list tersebut akan dipilih elemen pertama, yaitu elemen terdekat sehingga kompleksitas waktu pemilihan target adalah O(1). Seluruh strategi pemilihan target dilakukan secara sekuensial sehingga dari sisi efektivitas belum tentu menghasilkan strategi yang optimal.

Berdasarkan keempat strategi *greedy* yang dieksplorasi, terdapat kemungkinan bahwa pemilihan target akan menghasilkan aksi yang kurang tepat, misalnya bot dikepung oleh Gas Cloud dan Player lain. Pada keadaan tersebut, terdapat kemungkinan bot akan mengalami kebingungan harus bergerak kea rah mana sehingga menghasilkan strategi yang kurang efektif.

Meskipun begitu, penulis memilih beberapa strategi *greedy*yang dinilai sudah cukup optimal untuk dapat mencapai solusi optimum local karena pemilihan target sudah diurutkan

berdasarkan prioritas terlebih dahulu. Dengan begitu, target dengan prioritas lebih tinggi akan diutamakan untuk dipenuhi terlebih dahulu.

3.4 Strategi Greedy yang Dipilih

Berdasarkan berbagai macam dan metode percobaan, penulis menentukan bahwa beberapa strategi *greed* memiliki prioritas dan urgensi yang lebih tinggi dibanding beberapa strategi *greedy* lainnya. Salah satu contohnya adalah penulis tidak mengimplementasikan strategi mengenai Asteroid dan Wormhole.

Penulis menentukan bahwa Asteroid tidak terlalu memengaruhi efektivitas dan efisiensi strategi karena tidak berpengaruh terhadap ukuran kapal sehingga tidak terlalu penting untuk diimplementasikan . Sementara itu, Wormhole tidak diimplementasikan karena penulis menentukan bahwa efek Wormhole terlalu random untuk dipertimbangkan sehingga tidak efektif dan efisien untuk dimanfaatkan.

Namun, selain kedua GameObject tersebut, penulis menentukan bahwa strategi *greedy* perlu diterapkan untuk mencapai solusi optimum local. Berikut adalah urutan prioritas yang penulis terapkan dalam menerapkan strategi *greedy* pada permainan Galaxio.

1. Mengumpulkan seluruh sorted list untuk GameObject Food, Superfood, GasCloud, Player.
2. Target dan aksi pertama adalah yang berkaitan dengan Player. Apabila Player terdekat memiliki ukuran lebih besar, bot akan berusaha menghindar dari Player.
3. Target dan aksi selanjutnya adalah Gas Cloud karena dapat mengurangi ukuran bot. Bot akan berusaha bergerak memutari gas cloud untuk mencari target baru tanpa harus melewati Gas Cloud.
4. Target dan aksi selanjutnya adalah Player terdekat yang memiliki ukuran lebih kecil dibanding bot. Bot akan berusaha mengejar Player tersebut untuk memakannya.
5. Target dan aksi selanjutnya adalah berkaitan dengan Supernova. Jika tersedia Supernova pada Map, bot akan berusaha untuk mengambil Supernova. Hal tersebut dilakukan dengan tujuan utama agar Player lain tidak mendapat Supernova.
6. Setelah keempat prioritas target dan aksi di atas terpenuhi, bot akan mulai mengecek keadaan Superfood dan Food terdekat. Bot akan bergerak menuju Superfood atau Food terdekat dengan syarat tidak menyalahi keempat prioritas utama di atas.

Dalam kondisi tertentu, bot akan melakukan beberapa aksi khusus seperti di bawah ini.

1. Jika tidak terdapat Superfood dan Food di sekitar bot, bot akan otomatis mencoba untuk bergerak menuju pusat Map sebagai cara menghindari Radius yang terus mengecil.
2. Jika bot sedang dalam kodnisi mengejar Player, bot akan menyalakan Afterburner untuk mencapai Player lebih cepat. Jika penggunaan Afterburner dirasa sudah cukup, bot akan mematikan Afterburner.
3. Jika bot berhasil mengambil Supernova, bot akan menembakkan Supernnova pada Player terdekat dan meledakkannya jika jarak Supernova dengan Player sudah cukup dekat.
4. Jika bot ingin mencapai tempat tertentu secara cepat, bot dapat menembakkan Teleporter untuk berpindah secara langsung.

5. Jika bot sedang menghindar atau sedang mengejar Player terdekat dan dirasa tembakan Torpedo dapat mengenai Player, bot akan menembakkan Torpedo kea rah Player.

BAB IV
Implementasi dan Pengujian

4.1 Implementasi Algoritma Greedy

Implementasi algoritma greedy pada program terdapat pada file BotService.java

4.1.1 Prosedur computeNextPlayerAction

```
procedure computeNextPlayerAction(input/output PlayerAction : playerAction)
{menghitung player action yang akan diambil}


KAMUS LOKAL

heading : integer                       { heading yang akan digunakan untuk menghitung next heading }
defaultTarget : GameObject                { target yang akan digunakan untuk menghitung next heading }
targetWithNewValues : GameObject
playerGameObjects : List<GameObject>          { daftar semua game object yang dimiliki oleh player }


ALGORITMA
playerAction.heading <- resolveNewTarget()      {menghitung heading yang akan digunakan untuk menghitung next
                                                heading dan menghitung player action yang akan diambil}
playerAction.action <- resolveNewAction()

print("Player action:" + playerAction.action + ":" + playerAction.heading)

this.playerAction <- playerAction               {menyimpan player action yang baru dihitung}
```

Gambar 2. Prosedur computeNextPlayerAction

4.1.2 Fungsi resolveNewAction

```
function resolveNewAction() -> PlayerActions
{menghitung player action yang akan diambil}

KAMUS LOKAL

afterburnerCondition : boolean
teleporterCondition : boolean
supernovaCondition : boolean


ALGORITMA

if (!afterburnerCondition and targetIsPlayer and (getDistanceBetween(target, bot) < 2*bot.size) and ((bot.size - target.size) >= 20)) then
    print("Activating afterburner")           { jika afterburner belum aktif dan target adalah player, dan jarak antara bot dan
                                              target kurang dari 2 kali ukuran bot,
                                              dan ukuran bot lebih besar dari target, maka afterburner akan diaktifkan }
    afterburnerCondition = true
    -> PlayerActions.StartAfterburner

else if (afterburnerCondition and ((targetIsPlayer and (getDistanceBetween(target, bot) > 2*bot.size)) or (!targetIsPlayer) or ((bot.size - target.size) < 20))) then
    print("Deactivating afterburner")
    afterburnerCondition = false
    -> PlayerActions.StopAfterburner              { jika afterburner sudah aktif dan target bukan player atau jarak antara bot dan
                                                  target lebih dari 2 kali ukuran bot,
                                                  atau ukuran bot lebih kecil dari target, maka afterburner akan dinonaktifkan }

else if (!supernovaCondition and targetIsPlayer and (bot.supernovaAvailable > 0)) then
    println("Firing Supernova")
    supernovaCondition = true
    -> PlayerActions.FireSupernova

else if (supernovaCondition and targetIsPlayer and supernovaBomb != null and (getDistanceBetween(target, supernovaBomb) < 200)) then
    print("Detonating Supernova")
    supernovaCondition = false
    -> PlayerActions.DetonateSupernova

else if (!teleporterCondition and targetIsPlayer and bot.teleporterCount > 0 and ((bot.size - target.size) > 40) and (getDistanceBetween(target, bot) < 5*bot.size)) then
    print("Firing Teleporter")
    teleporterCondition = true
    -> PlayerActions.FireTeleport                 { jika teleporter belum aktif dan target adalah player, dan teleporter masih tersedia,
                                                  dan ukuran bot lebih besar dari target,
                                                  dan jarak antara bot dan target kurang dari 5 kali ukuran bot, maka teleporter akan diaktifkan }

else if (teleporterCondition and targetIsPlayer and teleporter != null and (getDistanceBetween(target, teleporter) < 2*nearestOpponent.size)) then
    print("Teleporting")
    teleporterCondition = false
    -> PlayerActions.Teleport                       { jika teleporter sudah aktif dan target adalah player, dan teleporter tidak null, dan jarak antara target
                                                    dan teleporter kurang dari 2 kali ukuran target, maka teleporter akan dinonaktifkan }
```

```
else if (targetIsPlayer and bot.size > 20 and bot.torpedoSalvoCount > 0) then
    print("Firing Torpedoes at target")
    -> PlayerActions.FireTorpedoes                  { jika target adalah player, dan ukuran bot lebih besar dari 20, dan torpedo masih tersedia,
                                                    maka torpedo akan diaktifkan }

else if (avoidingPlayer and (bot.shieldCount > 0) and (getDistanceBetween(nearestOpponent, bot) < 3*nearestOpponent.size) and (bot.size > 50) and (nearestOpponent.torpedoS
    print("Activating shield")
    -> PlayerActions.ActivateShield                   { jika bot sedang menghindari player, dan shield masih tersedia, dan jarak antara bot dan
                                                      player kurang dari 3 kali ukuran player, dan ukuran bot lebih besar dari 50,
                                                      dan torpedo player masih tersedia, maka shield akan diaktifkan }


else
    -> PlayerActions.Forward                      { jika tidak ada kondisi yang terpenuhi, maka bot akan bergerak maju }
```

Gambar 3. Fungsi resolveNewAction

4.1.3 Fungsi resolveNewTarget

```
function resolveNewTarget() -> integer
{ menghitung heading baru }


KAMUS LOKAL
heading, nearestSuperFood, nearestGasCloud, nearestPlayer : var
distanceToSuperFood, firedTeleporter, distanceToFood : var
distanceFromWorldCenter, availableSupernova, bombSupernova : var
avoidingPlayer, targetIsPlayer : boolean
centerPosition : Position
world : World
radius : integer

ALGORITMA
heading <- new Random().nextInt(360)                                        { heading akan diisi dengan angka random dari 0-359 }
if (!gameState.getGameObjects().isEmpty()) then
    nearestFood <- gameState.getGameObjects()
        .stream().filter(item -> item.getGameObjectType() = ObjectTypes.Food)          { mengambil game object yang memiliki tipe food }
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList())
    nearestSuperFood <- gameState.getGameObjects()
        .stream().filter(item -> item.getGameObjectType() = ObjectTypes.Superfood)         { mengambil game object yang memiliki tipe superfood }
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList())
    nearestGasCloud <- gameState.getGameObjects()
        .stream().filter(item -> item.getGameObjectType() = ObjectTypes.GasCloud)              { mengambil game object yang memiliki tipe gas cloud }
        .sorted(Comparator
            .comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList())
    nearestPlayer <- gameState.getPlayerGameObjects()                               { mengambil game object yang memiliki tipe player }
        .stream().filter(player -> player.getId() != bot.getId())
        .sorted(Comparator
            .comparing(player -> getDistanceBetween(bot, player)))
        .collect(Collectors.toList())
    nearestOpponent <- nearestPlayer.get(0)                                    { mengambil player yang paling dekat dengan bot }

    if (nearestPlayer.size() > 0) then
        nearestOpponent <- nearestPlayer.get(0)

    if (teleporterCondition) then
        firedTeleporter <- gameState.getGameObjects()
            .stream().filter(item -> item.getGameObjectType() == ObjectTypes.Teleporter)           { mengambil game object yang memiliki tipe teleporter }
            .sorted(Comparator
                .comparing(item -> getDistanceBetween(nearestOpponent, item)))
            .collect(Collectors.toList())
```

```
      if (firedTeleporter.size() > 0) then
          teleporter <- firedTeleporter.get(0)                                      { mengambil teleporter yang paling dekat dengan player }
  availableSupernova <- gameState.getGameObjects()
      .stream().filter(item -> item.getGameObjectType() = ObjectTypes.SupernovaPickup)
      .sorted(Comparator
          .comparing(item -> getDistanceBetween(bot, item)))
      .collect(Collectors.toList())                                             { mengambil teleporter yang paling dekat dengan player }

  if (availableSupernova.size() > 0) then
      supernovaPickup <- availableSupernova.get(0)

  bombSupernova <- gameState.getGameObjects()
      .stream().filter(item -> item.getGameObjectType() = ObjectTypes.SupernovaBomb)
      .sorted(Comparator
          .comparing(item -> getDistanceBetween(bot, item)))
      .collect(Collectors.toList())

  if (bombSupernova.size() > 0) then
      supernovaBomb <- availableSupernova.get(0);

  if (bot.size > 50 and bot.torpedoSalvoCount > 0) then
      heading <- getHeadingBetween(nearestPlayer.get(0))              { jika ukuran bot lebih besar dari 20 dan torpedo masih tersedia,
                                                                        maka heading akan diisi dengan heading antara bot dan player }
      target <- nearestPlayer.get(0)
      avoidingPlayer <- false
      targetIsPlayer <- true

  else if ((nearestPlayer.get(0)).size > bot.size and (getDistanceBetween(nearestPlayer.get(0), bot) < 200)) then
      heading <- getAttackerResolution(nearestPlayer.get(0), nearestSuperFood.get(0), nearestFood.get(0))
      avoidingPlayer <- true
      targetIsPlayer <- false
      print("Avoiding Opponent")

  else if ((getDistanceBetween(nearestGasCloud.get(0), bot)) < 3*bot.size) then
      heading <- getHeadingBetween(nearestGasCloud.get(0)) + 90
      avoidingPlayer <- false
      targetIsPlayer <- false
      print("Avoiding Gas Cloud")

  else if ((nearestPlayer.get(0)).size < bot.size and (getDistanceBetween(nearestPlayer.get(0), bot) < 200)) then
      heading <- getHeadingBetween(nearestPlayer.get(0))
      target <- nearestPlayer.get(0)
      avoidingPlayer <- false
      targetIsPlayer <- true
      print("Targeting Opponent")
```

```
else if ((supernovaPickup != null) and (getDistanceBetween(supernovaPickup, bot) < 250)) then
    heading <- getHeadingBetween(supernovaPickup)
    target <- supernovaPickup
    avoidingPlayer <- false
    targetIsPlayer <- false
    print("Heading for Supernova")

else if (nearestSuperFood != null) then
    if (nearestFood != null) then
        distanceToSuperFood <- getDistanceBetween(bot, nearestSuperFood.get(0))        { jika superfood tidak kosong dan food tidak kosong,
                                                                                         maka akan dihitung jarak antara bot dan superfood dan jarak antara bot
                                                                                         dan food }
        distanceToFood <- getDistanceBetween(bot, nearestFood.get(0))

        if (distanceToSuperFood > distanceToFood) then
            heading <- getHeadingBetween(nearestFood.get(0))                    { jika jarak antara bot dan superfood lebih besar dari jarak antara bot
                                                                                  dan food, maka heading akan diisi dengan heading antara bot dan food }
            target <- nearestFood.get(0)
            avoidingPlayer <- false
            targetIsPlayer <- false
            print("Targeting Food")

        else
            heading <- getHeadingBetween(nearestSuperFood.get(0))               { jika jarak antara bot dan superfood lebih kecil dari jarak antara bot
                                                                                  dan food, maka heading akan diisi dengan heading antara bot dan superfood }
            target <- nearestSuperFood.get(0)
            avoidingPlayer <- false
            targetIsPlayer <- false
            print("Targeting Superfood")

    else
        heading <- getHeadingBetween(nearestSuperFood.get(0))                 { jika superfood tidak kosong dan food kosong, maka heading akan diisi
                                                                                dengan heading antara bot dan superfood }
        target <- nearestSuperFood.get(0)
        avoidingPlayer <- false
        targetIsPlayer <- false
        print("Targeting Superfood")

else if (nearestFood != null) then
    heading <- getHeadingBetween(nearestFood.get(0))                      { jika superfood kosong dan food tidak kosong, maka heading akan diisi dengan
                                                                            heading antara bot dan food }
    target <- nearestFood.get(0)
    avoidingPlayer <- false
    targetIsPlayer <- false
    print("Targeting Food")

centerPosition <- new Position()
distanceFromWorldCenter <- getDistanceBetween(bot,new GameObject(null, null, null, null, centerPosition, null, null, null, null, null, null))
                                                                {menghitung jarak antara bot dan world center}
```

```
   world <- gameState.getWorld()
   radius <- world.getRadius()

   if (radius = null) then
       if (distanceFromWorldCenter + (1.5 * bot.size) > 1000) then
           worldCenter <- new GameObject(null, null, null, null, centerPosition, null, null, null, null, null, null)
                                                                    { jika radius kosong dan jarak antara bot dan world center ditambah 1.5 kali
                                                                    ukuran bot lebih besar dari 1000, maka heading akan diisi dengan heading antara bot
                                                                    dan world center }
           heading <- getHeadingBetween(worldCenter)
           target <- worldCenter
           targetIsPlayer <- false
           print("Near the edge, going to the center")

   else
       if (distanceFromWorldCenter + (1.5 * bot.size) > radius) then
           worldCenter <- new GameObject(null, null, null, null, centerPosition, null, null, null, null, null, null)
                                                                    { jika radius tidak kosong dan jarak antara bot dan world center ditambah 1.5 kali
                                                                    ukuran bot lebih besar dari radius, maka heading akan diisi dengan heading antara bot
                                                                    dan world center }
           heading <- getHeadingBetween(worldCenter)
           target <- worldCenter
           targetIsPlayer <- false
           print("Near the edge, going to the center")

-> heading
```

Gambar 4. Fungsi resolveNewTarget

4.1.4 Fungsi getAttackerResolution

```
function getAttackerResolution(attacker : GameObject, closestSuperFood : GameObject, closestFood : GameObject ) -> integer
{ fungsi untuk menghitung heading antara bot dan player }

KAMUS LOKAL
distanceToAttacker, distanceToSuperFood, distanceToFood : var
distanceBetweenAttackerAndSuperFood, distanceBetweenAttackerAndFood : var

ALGORITMA
if (closestFood = null and closestSuperFood = null) then
    -> getOppositeHeading(attacker)        // jika food dan superfood kosong, maka heading akan diisi dengan heading antara bot dan player yang berlawanan arah

distanceToAttacker <- getDistanceBetween(bot, attacker)               { jika food dan superfood tidak kosong, maka akan dihitung jarak antara bot dan player,
                                                                        jarak antara bot dan superfood, jarak antara bot dan food, jarak antara player dan superfood,
                                                                        dan jarak antara player dan food}

distanceToSuperFood <- getDistanceBetween(bot, closestSuperFood)      { jika jarak antara bot dan player lebih besar dari speed player dan jarak antara player dan
                                                                        superfood lebih besar dari jarak antara bot dan player, maka akan dihitung jarak antara bot
                                                                        dan superfood dan jarak antara bot dan food}

distanceToFood <- getDistanceBetween(bot, closestFood)                { jika jarak antara bot dan superfood lebih besar dari jarak antara bot dan food,
                                                                        maka heading akan diisi dengan heading antara bot dan food }
distanceBetweenAttackerAndSuperFood <- getDistanceBetween(attacker, closestSuperFood)   { jika jarak antara bot dan superfood lebih kecil dari jarak antara bot
                                                                                          dan food, maka heading akan diisi dengan heading antara bot dan superfood }
distanceBetweenAttackerAndFood <- getDistanceBetween(attacker, closestFood)

if ((distanceToAttacker > attacker.speed) and ((distanceBetweenAttackerAndSuperFood - distanceToAttacker) > 2*attacker.size)) then
    if (distanceToFood > distanceToSuperFood) then
        -> getHeadingBetween(closestSuperFood)            { jika jarak antara bot dan player lebih besar dari speed player dan jarak antara player dan
                                                            superfood lebih kecil dari jarak antara bot dan player, maka akan dihitung jarak antara bot dan
                                                            superfood dan jarak antara bot dan food }
    else
        -> getHeadingBetween(closestFood)                 { jika jarak antara bot dan superfood lebih besar dari jarak antara bot dan food,
                                                            maka heading akan diisi dengan heading antara bot dan food }

else if ((distanceToAttacker > attacker.speed) and (distanceBetweenAttackerAndFood - distanceToAttacker) > 2*attacker.size)) then
    if (distanceToSuperFood > distanceToFood) then
        -> getHeadingBetween(closestFood)                   { jika jarak antara bot dan superfood lebih kecil dari jarak antara bot dan food,
                                                              maka heading akan diisi dengan heading antara bot dan superfood}
    else
        -> getHeadingBetween(closestSuperFood)              { jika jarak antara bot dan superfood lebih besar dari jarak antara bot dan food,
                                                              maka heading akan diisi dengan heading antara bot dan food}

else
    -> getOppositeHeading(attacker)                         { jika jarak antara bot dan player lebih kecil dari speed player dan jarak antara player dan
                                                              superfood lebih kecil dari jarak antara bot dan player, maka heading akan diisi dengan
                                                              heading antara bot dan player yang berlawanan arah}
```

Gambar 5. Fungsi getAttackerResolution

4.2 Struktur Data

Struktur data yang kami gunakan terbagi menjadi 3 bagian besar yaitu command, entities, dan enums yang disertakan dengan bot.java dan main.java.

1. Enums : berisikan file-file yang berisi apa saja objek dan aksi yang terdapat dalam permainan.
   - ObjectType : berisi objek-objek yang terdapat dalam permainan
   - PlayerActions : berisi aksi yang bisa dilakukan kapal dalam permainan
2. Model : berisikan file-file yang menginisiasi data-data dalam permainan
   - GameObject : berisi inisiasi objek dalam permainan di mana objek-obek tersebut diberi value.
   - GameState : berisi informasi dari keadaan-keadaan yang diisi saat permainan berlangsung
   - GameStateDto : berfungsi mengirimkan data-data kondisi dalam permainan dalam bentuk seriazable object.
   - PlayerAction : menginisiasi aksi dan gerakan kapal dalam permainan
   - Position : menginisiasi posisi objek-objek dalam permainan
   - World : menginisiasi objek-objek yang menjadi elemen pembatas dalam permainan
3. Services
   - BotService : berisi strategi-strategi yang dituangkan ke bot kapal.
4. Main : berisi eksekutor state-state untuk menjalankan permainan.

Berikut penjelasan struktur data tambahan yang digunakan dalam implementasi algoritma *greedy.*

Nama Method	Tipe	Fungsi / Deskripsi
resolveNewTarget	int	Menentukan target baru untuk aksi bot selnajutnya
getAttackerResolution	int	Menentukan arah dan aksi bot jika Player terdekat lebih besar dari bot

Tabel 2. Struktur Data Method

Nama Atribut	Tipe	Fungsi / Deskripsi
Private target	GameObject	Hasil penentuan objek yang ingin didekati.
Private worldCenter	GameObject	Titik pusat map untuk menentukan radius
Private targetIsPlayer	boolean	Menentukan apakah target adalah Player atau bukan
actionID	PlayerActions	Menentukan aksi apa yang akan dilakukan bot
heading	int	Menentukan arah aksi bot

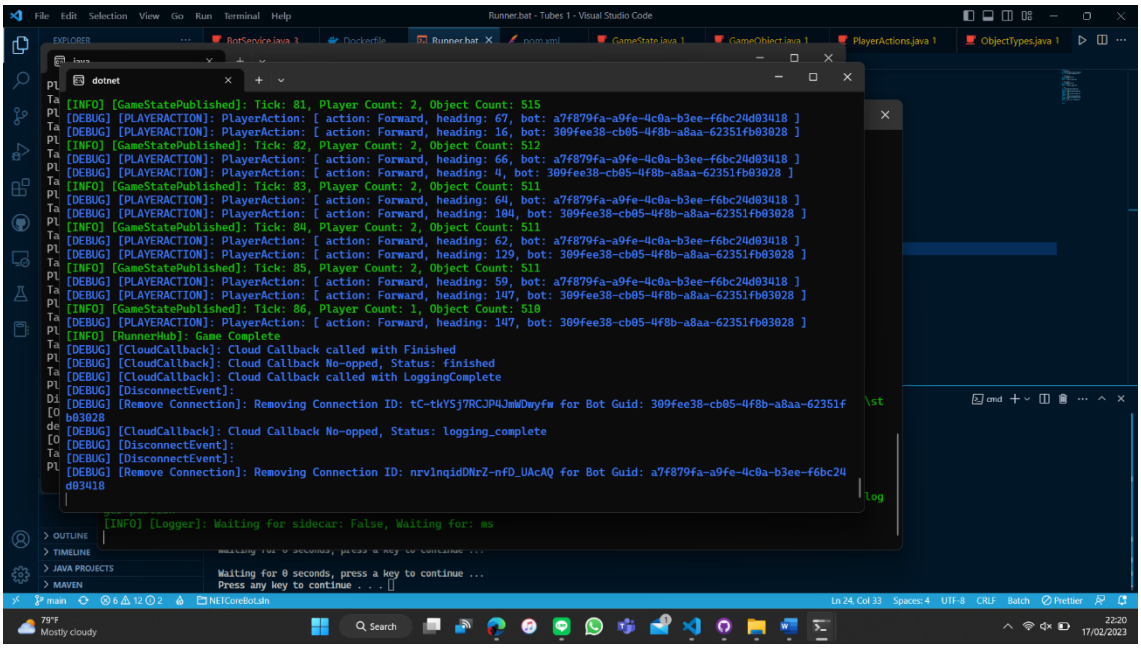
playerGameObjects	List<GameObject>	Menampung semua Player yang ada dalam permainan
defaultTarget	GameObject	Target default adalah target yang sebelumnya (jika tidak perlu mengganti target)
targetWithNewValues	GameObject	Menentukan target baru
centerPosition	Position	Titik pusat map untuk menentukan radius
distanceFromWorldCenter	double	Jarak antara bot dengan pusat map
nearestFood	List<GameObject>	Menampung semua Food terurut dari yang paling dekat
nearestSuperFood	List<GameObject>	Menampung semua Superfood terurut dari yang paling dekat
nearestPlayer	List<GameObject>	Menampung semua Player terurut dari yang paling dekat
distanceToSuperFood	double	Jarak antara bot dengan Superfood terdekat
distanceToFood	double	Jarak antara bot dengan Food terdekat
distanceToAttacker	double	Jarak antara bot dengan Player terdekat
distanceBetweenAttacker AndSuperFood	double	Jarak antara Player terdekat dengan Superfood terdekat
distanceBetweenAttacker AndFood	double	Jarak antara Player terdekat dengan Food terdekat

Tabel 3. Struktur Data Atribut

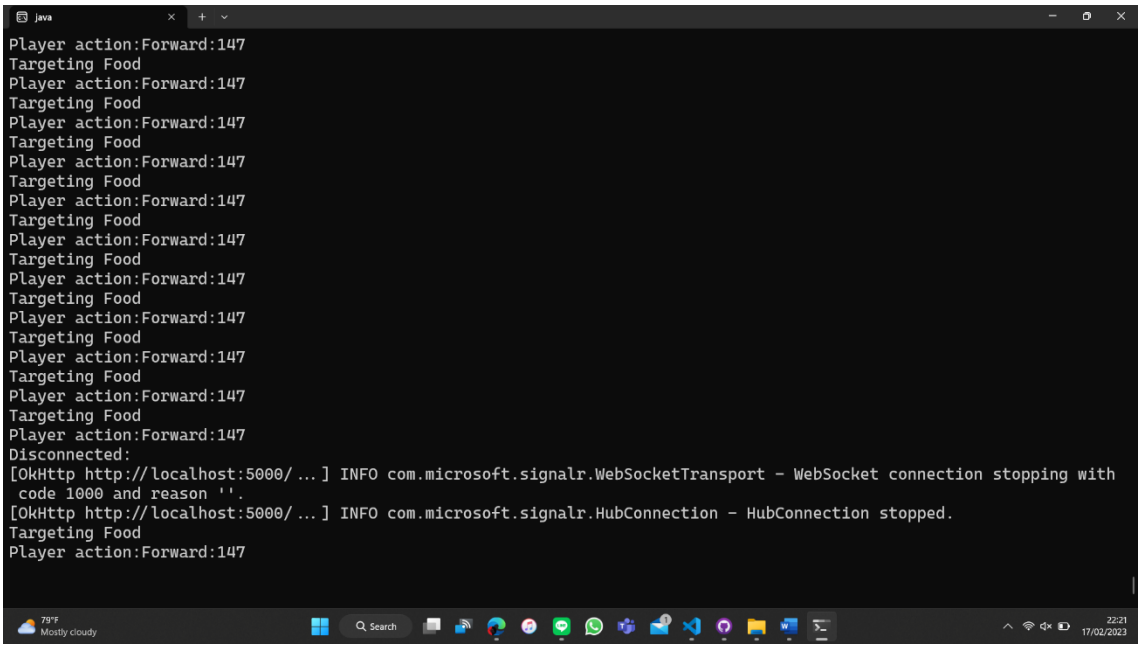
4.3 Analisis Desain Solusi

Berikut adalah analisis desain solusi berdasarkan percobaan yang telah kami jalankan. Percobaan ini dilakukan dengan jumlah 4 bot (1 bot yang kami buat dan 3 reference bot). Percobaan dilakukan sebanyak 6 kali.

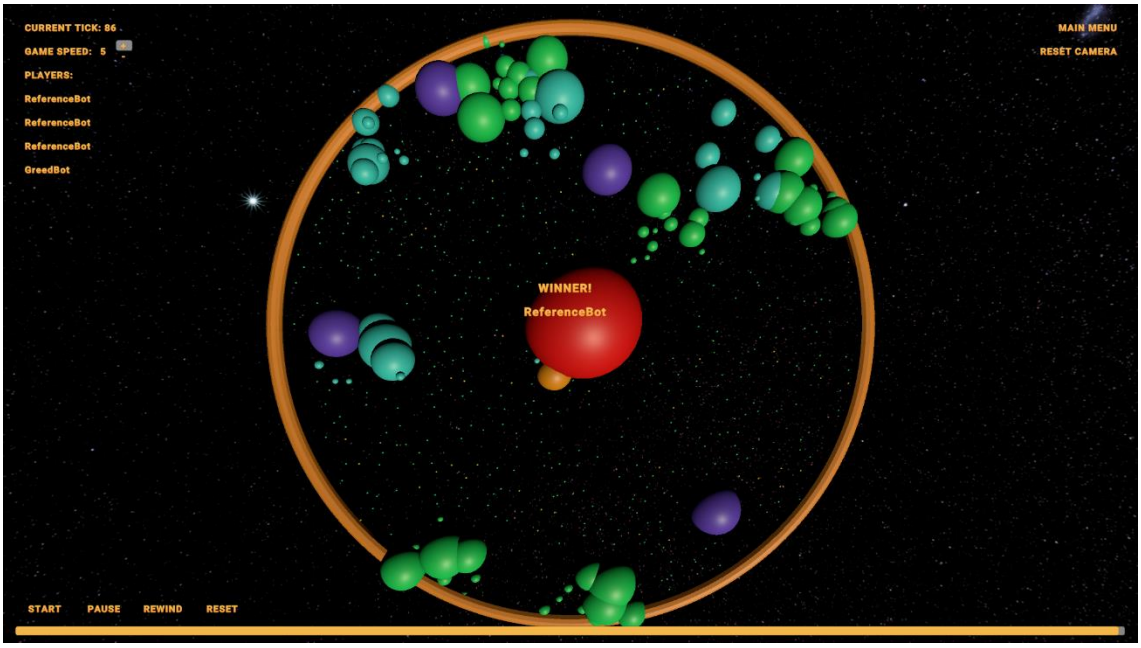
- Percobaan 1 (Kalah)



Gambar 6 Percobaan 1 dotnet logger



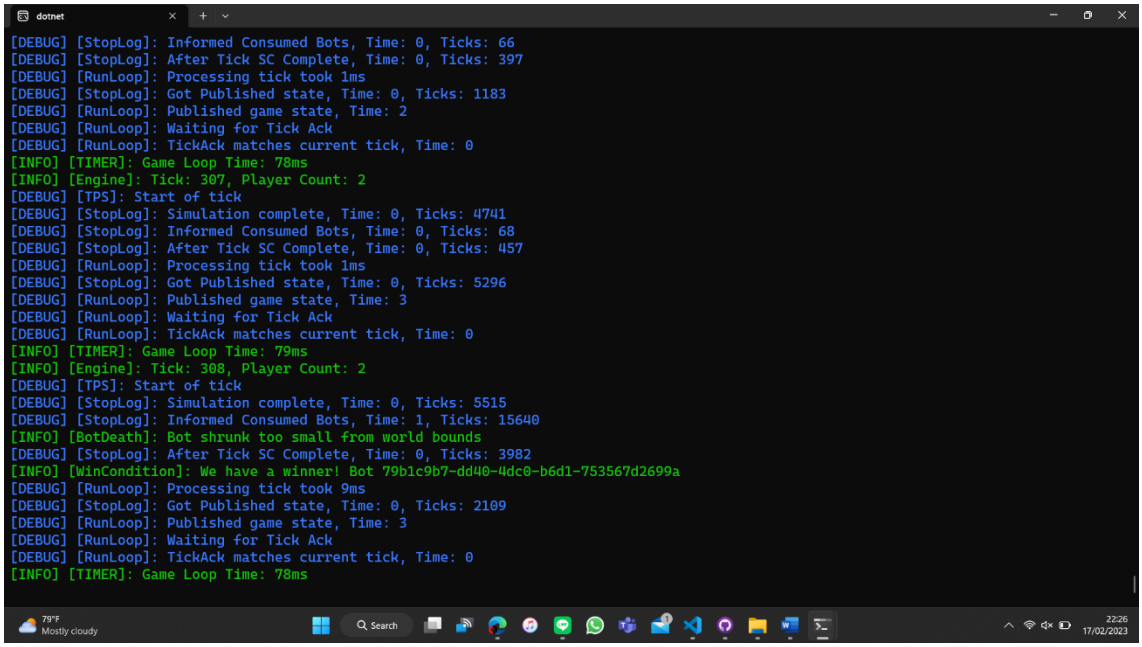
Gambar 7 Percobaa 1 bot logger



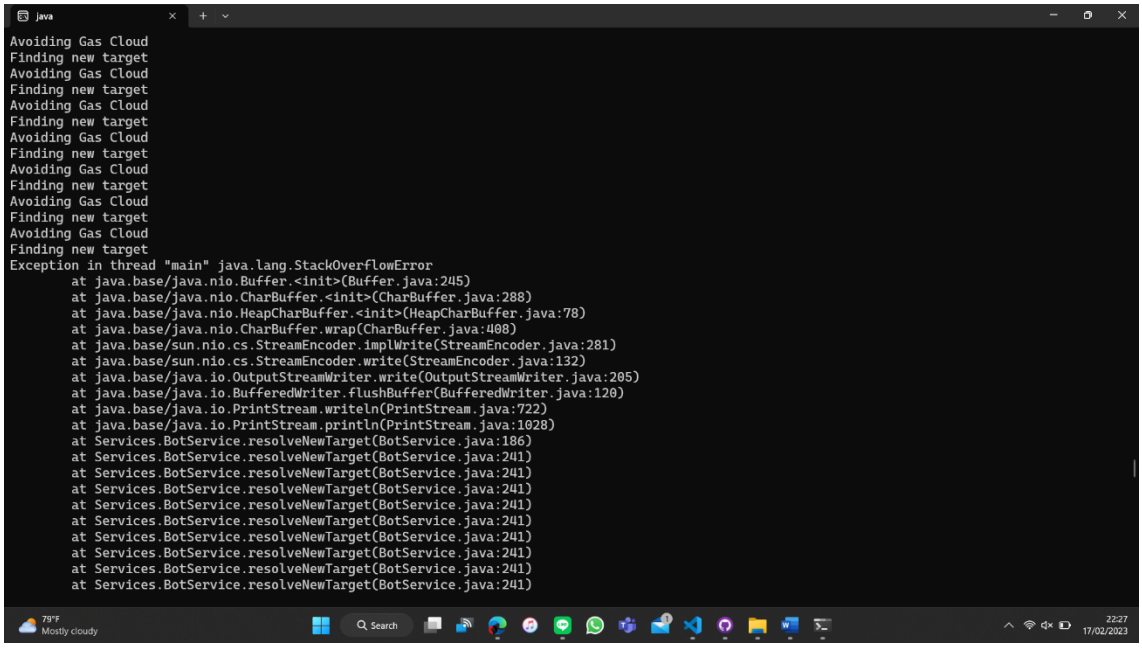
Gambar 8. Percobaan 1 Visualiser

Pada percobaan 1 ini, GreedBot mengalami kekalahan karena dimakan oleh pemain lawan. Alasan terjadinya hal ini karena GreedBot sedang melakukan algoritma greedy untuk mengambil makanan. Pada saat mengambil makanan, bot lawan berhasil memakan GreedBot.

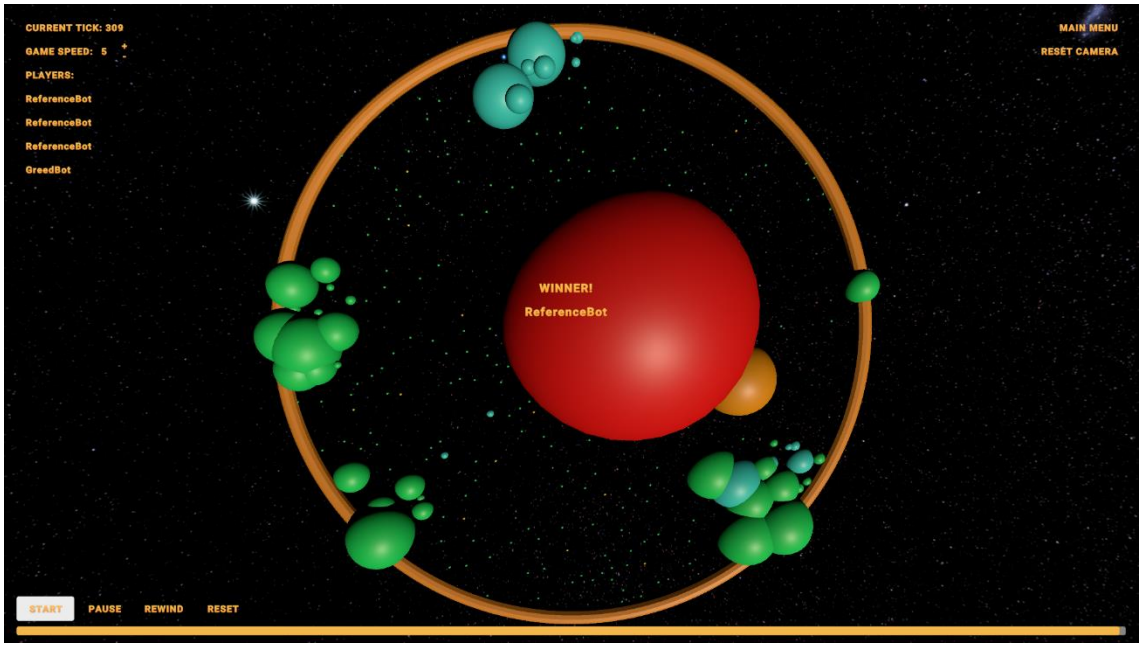
- Percobaan 2 (Error)



Gambar 9 Percobaa 1 dotnet logger



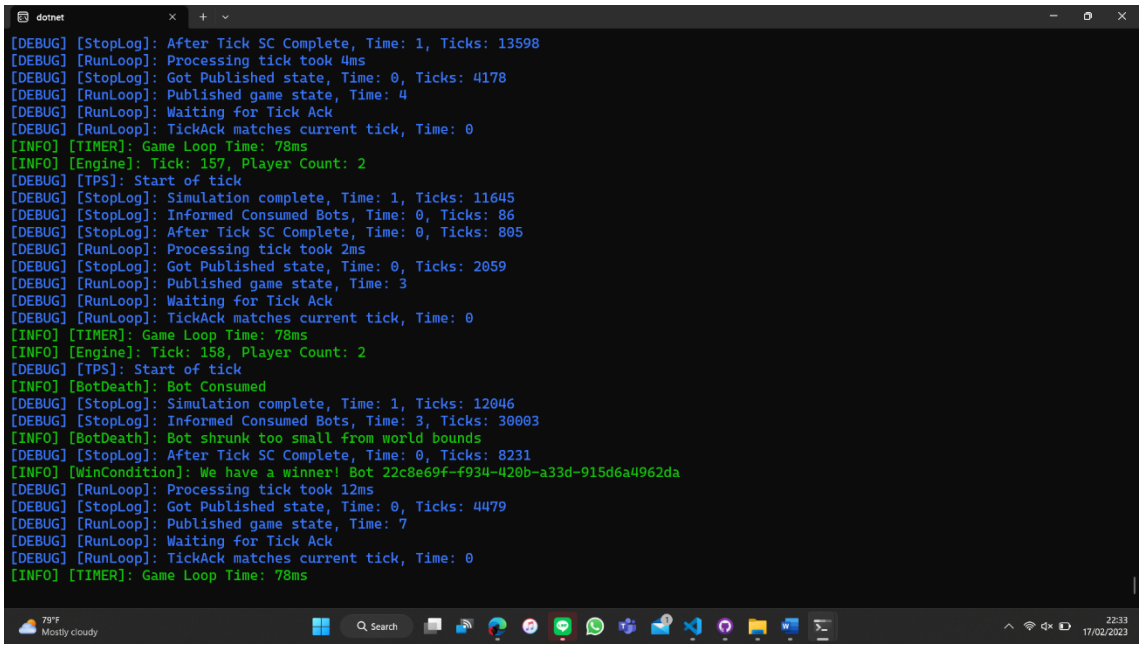
Gambar 10 Percobaan 2 bot logger



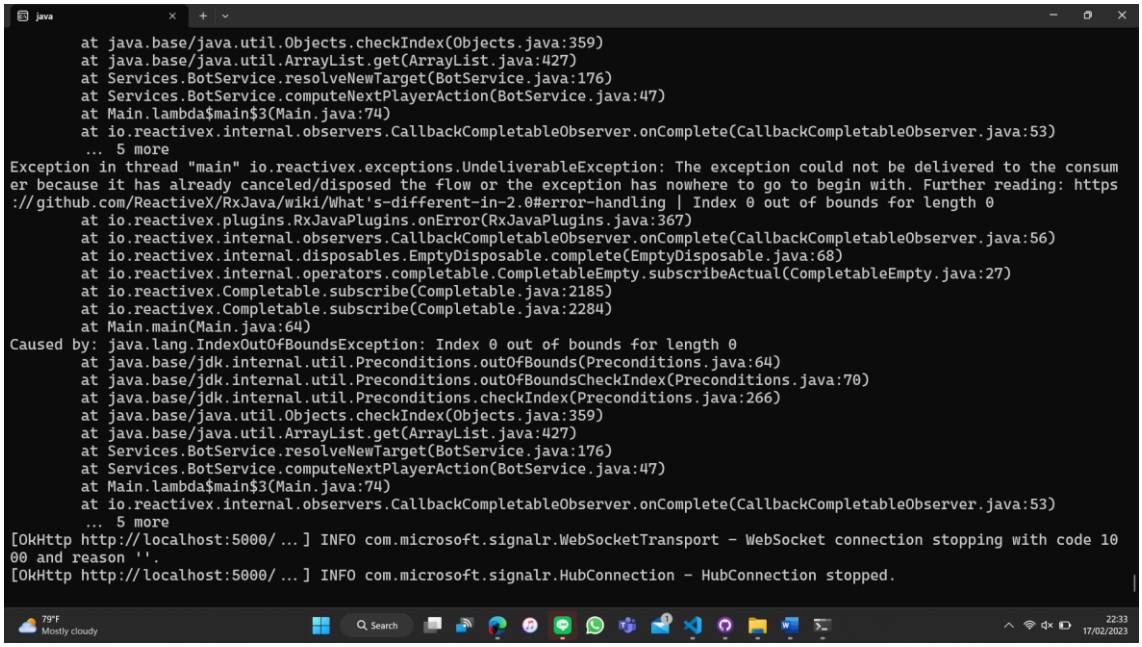
Gambar 11. Percobaan 2 Visualiser

Pada percobaan 2 ini, terdapat error pada GreedBot. Hal ini terjadi karena bot sedang berusaha menghindari gas cloud, namun pada saat itu juga bot berusaha menghindari radius, sehingga terjadi error tersebut.

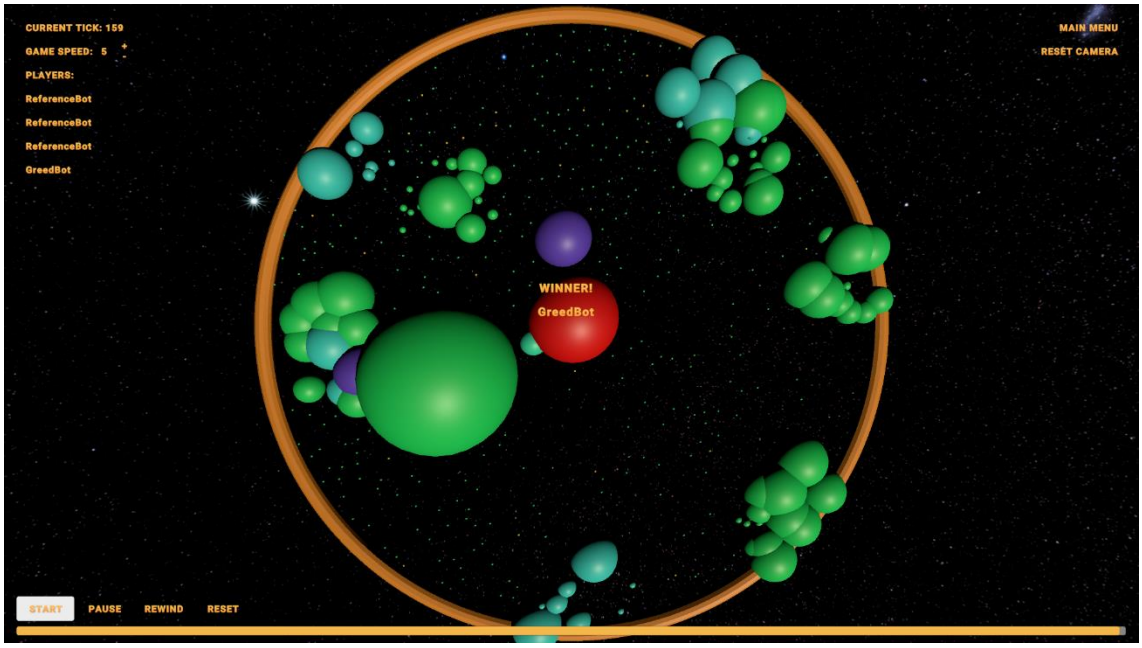
- Percobaan 3 (Menang)



Gambar 12 Percobaan 3 dotnet logger



Gambar 13 Percobaan 3 bot logger



Gambar 14 Percobaan 3 visualiser

Pada percobaan 3 ini, GreedBot mengalami kemenangan karena GreedBot berhasil memakan seluruh lawan. Pada percobaan ini, semua implementasi algoritma greedy berhasil berjalan dengan baik.

BAB V
Kesimpulan dan Saran

5.1 Kesimpulan

Strategi *greedy* digunakan untuk menentukan solusi optimal untuk suatu masalah. Dalam permainan Galaxio, pemenangnya adalah kapal yang bertahan paling lama. Untuk memenangi permainan, strategi *greedy* diterapkan untuk menentukan langkah-langkah optimal. Strategi *greedy* yang diterapkan adalah *greedy by size*, *greedy by food*, *greedy by speed.*

Berikut adalah beberapa hal yang dapat dilakukan bot berdasarkan strategi *greedy* yang sudah diterapkan:

1. Menentukan apakah bot perlu mengejar atau menjauh dari player lain
2. Menembakkan Torpedo ke arah musuh terdekat
3. Mencari Superfood atau Food terdekat
4. Mencari Supernova jika tersedia
5. Meledakkan Supernova setelah menembakkan Supernova
6. Menembakkan Teleporter menuju arah tertentu
7. Berpindah tempat menggunakan Teleporter yang sudah ditembakkan
8. Mengaktifkan Shield dalam kondisi tertentu

5.2 Saran

Pada pengerjaan tugas besar ini, terdapat kendala waktu karena diperlukan waktu untuk mempelajari dan memahami cara kerja *game* engine sebelum membuat implementasi bot. Agar waktu mengerjakan tugas cukup berikut adalah hal yang perlu dilakukan:

1. Segera pelajari fungsi-fungsi yang belum dipahami sehingga dapat diidentifikasi masalah seperti apa yang akan ditemui dan seberapa lama waktu yang dibutuhkan untuk menyelesaikan permasalahan tersebut.
2. Segera membagi tugas agar pekerjaan dapat segera diselesaikan sesuai dengan kesepakatan pembagian tugas.
3. Tidak menunda-nunda pengerjaan tugas.

5.3 Komentar dan Refleksi

Pada tugas besar kali ini sangat menuntut untuk memikirkan strategi greedy terbaik yang dapat diimplementasikan ke dalam bot. Pada tugas kali ini juga diperlukan koordinasi antar anggota penyusun agar dapat menyelesaikan setiap masalah yang terjadi. Meskipun dapat menyelesaikan tugas dengan tepat waktu, mungkin masih terdaat beberapa kesalahan.

DAFTAR PUSTAKA

Referensi

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf
- https://www.geeksforgeeks.org/greedy-algorithms/

Repository
https://github.com/ghaziakmalf/Tubes1_GreedBot

Link Youtube
https://youtu.be/x4JAkaX0Ijw