LAPORAN TUGAS BESAR

IF2211 Strategi Algoritma

Disusun Oleh :

Muhammad Zaydan Athallah (13521104)
Irsyad Nurwidianto Basuki (13521072)
Aulia Mey Diva Annandya (13521103)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung 2023

DAFTAR ISI

BAB 1: 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.

Bahasa pemrograman yang digunakan pada tugas besar ini adalah Java. Bahasa Java tersebut digunakan untuk membuat algoritma pada bot. IDE yang digunakan untuk membantu membuat projek ini adalah Visual Studio Code. Visual Studio Code merupakan IDE yang cukup kompatibel dengan bahasa Java. Namun, beberapa tools untuk java seperti Maven tidak built in .Untuk menjalankan permainan, digunakan sebuah game engine yang diciptakan oleh Entellect Challenge yang terdapat pada repository githubnya.

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 SuperFood 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. TorpedoSalvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Berikut jenis-jenis dari command yang ada dalam permainan:
    a. FORWARD
    b. STOP
    c. START_AFTERBURNER
    d. STOP_AFTERBURNER
    e. FIRE_TORPEDOES
    f. DETONATE_SUPERNOVA
    g. FIRE_TELEPORTER
    h. 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. Gambaran Algoritma Greedy secara Umum

Algoritma greedy adalah algoritma yang menerapkan konsep "greedy" untuk mendefinisikan solusinya. Algoritma greedy sering digunakan untuk mencari solusi masalah optimasi untuk memaksimalkan atau meminimalkan suatu parameter. Algoritma greedy dibentuk dengan menyelesaikan tugas dan membentuk solusi langkah demi langkah (step by step). Banyak langkah yang dapat dievaluasi dalam setiap tahapan ini. Dalam algoritma greedy, programmer harus menentukan keputusan terbaik pada setiap langkah. Namun, untuk algoritma greedy, pelacakan mundur tidak diperbolehkan (Anda tidak dapat melihat kembali ke solusi sebelumnya untuk menentukan solusi langkah saat ini). Oleh karena itu, pemrogram diharapkan memilih solusi optimal lokal pada setiap langkah. Tujuannya agar langkah optimasi lokal 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 sub-persoalan tersebut
- 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.

Terdapat beberapa elemen/komponen 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).

Dengan menggunakan elemen/komponen di atas, algoritma greedy dapat didefinisikan sebagai berikut: “Algoritma greedy merupakan pencarian sebuah himpunan bagian *S* dari himpunan kandidat *C*, dimana *S* memenuhi kriteria kelayakan sebagai solusi paling optimum, yaitu *S* merupakan suatu himpunan solusi dan *S* di-optimisasi oleh fungsi objektif.”

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 2.1. *Pseudocode* algoritma *greedy*

Pada akhir tiap iterasi, solusi yang terbentuk adalah optimum lokal, dan paka akhir loop while-do akan ditemukan optimum global, namun optimum global yang ditemukan ini belum tentu merupakan solusi yang terbaik karena algoritma greedy tidak melakukan operasi secara menyeluruh kepada semua kemungkinan yang ada.

Kesimpulannya, 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. Garis Besar Cara Kerja Bot Permainan Galaxio

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

1. *World* : Kondisi World/Map berupa bidang kartesius

   Setiap object pada *World* akan dijelaskan sebagai berikut:

   a. *Position* : Berisi posisi centerpoint (0,0) pada map
   b. (Integer) *Radius* : Ukuran lingkaran void suatu map
   c. (Integer) *currentTick* : Seperti waktu permainan berjalan, contoh : bot bisa melakukan 1 action setiap satu tick.

2. List<gameObject> : List berisi gameObject

   Setiap object pada *gameObject* akan dijelaskan sebagai berikut :

   a. *id* : Berisi id
   b. *(*Integer) *size* : size object
   c. (Integer) *speed* : kecepatan sebuah object
   d. (Integer) currentHeading : tujuan/arah object
   e. (Position) *position* : Posisi (x,y) dari object
   f. ObjectTypes : Tipe/Jenis dari object yaitu,
      i. PLAYER
      ii. FOOD
      iii. WORMHOLE
      iv. GASCLOUD
      v. ASTEROIDFIELD
      vi. TORPEDOSALVADO
      vii. SUPERFOOD
      viii. SUPERNOVAPICKUP
      ix. SUPERNOVABOMB
      x. TELEPORTER
      xi. SHIELD

Cara kerja bot dimulai dengan menjalankan computeNextPlayerAction dimana prosedur tersebut akan memanggil prosedur resolveNewTarget berulang kali. computeNextPlayerAction akan mendeteksi target. Selain itu, prosedur tersebut juga dapat mengecek apakah posisi bot berada di batas arena maupun posisi bot yang terjepit dua objek yang harus dihindari. Algoritma greedy pada bot diimplementasikan dalam resolveNewTarget. Program akan mengambil semua objek terdekat berdasarkan tipe dari objek tersebut. Terdapat var fixfood untuk mengambil food ataupun superfood yang paling dekat.

Algoritma greedy dimulai dengan membandingkan jarak fixfood dan bot dengan jarak gascloud dan bot di dalam prosedur resolveNewTarget. Jika fixfood lebih dekat, maka akan di cek jarak bot dengan player terdekat. Bila bot sangat dekat dengan player lain dan bot memiliki size yang lebih besar empat kali dari player lain, maka bot akan mengejar player tersebut. Sebaliknya, bila bot memiliki size lebih kecil, bot akan pergi berlawanan arah dari player terdekat. Bila player lain tidak lebih kecil dari 4 kali bot size dan tidak lebih besar dari bot, maka bot akan pergi menuju fixfood.

Jarak gascloud juga merupakan salah satu penerapan algoritma greedy by obstacle. Bila jarak gascloud terdekat dan bot lebih dekat daripada jarak fixfood terdekat, maka bot akan pergi berlawanan arah dari gascloud. Bila terdapat gascloud dan player yang ukurannya lebih besar dari bot, program akan mensetting bot untuk tetap lari dari player walaupun terdapat gascloud. Terdapat juga torpedo untuk menembak player lain dengan ukuran lebih kecil maupun lebih besar selama berada dalam radius penembakan, teleport untuk berpindah tempat menuju bot lawan atau menghindari void, shield untuk berlindung dari tembakan torpedo player lain, serta supernova untuk menembakan supernova bomb kepada player lain guna mengurangi ukuran player lain.

2.3 Game Engine Galaxio

Dalam permainan Galaxio ini, terdapat beberapa komponen yang diperlukan agar permainan dapat berjalan:

1. *Engine* : *Engine* disini bertanggung jawab untuk menegakkan aturan permainan, dengan menerapkan perintah bot ke status permainan jika valid

2. *Runner* : *Runner* bertanggung jawab untuk menjalankan pertandingan antar pemain, memanggil perintah yang sesuai seperti yang diberikan oleh bot dan menyerahkannya ke mesin untuk dieksekusi.
3. *Logger* : *Logger* bertanggung jawab untuk menangkap semua perubahan pada status game, serta pengecualian apa pun, dan menuliskannya ke file log di akhir game.
4. *Reference Bot* : *Reference Bot* adalah bot referensi yang disediakan oleh Entelect (pembuat game Galaxio) yang dapat digunakan sebagai referensi pengerjaan bot serta lawan bot. Tersedia dalam bahasa C#.
5. *Starter bot* : Bot awal yang tersedia dalam beberapa bahasa pemrograman. Starter bot memiliki beberapa fungsi awalan untuk memudahkan peserta dalam membuat bot. Pada tugas besar ini digunakan Starter Bot dengan bahasa Java.

2.4 Struktur Game Engine

Untuk membuat bot dalam permainan Galaxio, diperlukan pengunduhan starter-pack yang telah disediakan oleh Entelect. *Release* yang terakhir yang digunakan terdapat pada tautan berikut : https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2.

Starter-pack berisi beberapa file, yaitu:

1. Engine-publish : berisikan file - file game engine
2. logger-publish : berisikan log - log dari game yang telah dijalankan
3. runner-publish : berisikan file - file game runner
4. reference-bot-publish : berisikan file - file dari reference bot
5. Starter-bots : berisikan berbagai file bot awalan dalam berbagai bahasa pemrograman.
6. visualiser : merupakan aplikasi untuk memvisualkan permainan.

Pada folder Starter-bots terdapat folder src pada path starter-bots/JavaBot/src/main/java yang berisikan beberapa folder dan file berikut :

1. Folder *enums* : folder ini berisikan file aksi yang dapat dilakukan oleh bot serta objek - objek apa saja yang terdapat dalam game.
2. Folder *models* : folder ini berisikan file yang mengatur dunia, state dari game, hingga identitas dari tiap objek yang ada.

3. Folder *services* : folder ini berisikan BotServices.java yang digunakan untuk penerimaan logika tertentu
4. File *Main.java* : File ini berisikan program utama yang akan menjalankan bot dengan menginstansiasi bot dan menerima perintah dari bot untuk dikirimkan ke game engine.

2.5 Implementasi Algoritma Greedy ke dalam Bot Permainan Galaxio

Ada beberapa algoritma yang dapat digunakan sebagai algoritma bot. Beberapa algoritma yang dapat digunakan antara lain algoritma brute force, algoritma *greedy*, algoritma tree traversal, dll. Namun, semua algoritma ini memiliki pro dan kontra. Dalam algoritma brute force, dibutuhkan banyak waktu dan sumber daya untuk menguji dan mengevaluasi setiap instruksi yang mungkin, bahkan jika hasil yang paling optimal ditemukan sebagai optimal global dalam situasi tersebut. Kemudian dalam algoritma *greedy* tidak memakan banyak waktu dan sumber daya. Namun, teknik heuristik yang dibutuhkan ditentukan oleh logika dan pola pikir masing-masing programmer. Teknik heuristik mungkin tidak menghasilkan solusi optimal global, dan bahkan jika itu adalah solusi optimal global, solusi tersebut sulit dibuktikan kebenarannya secara matematis. Algoritma selanjutnya yang dapat digunakan adalah algoritma tree traversal. Algoritma ini sering digunakan pada robot game karena pada algoritma ini kasus-kasus tertentu diuji untuk menentukan langkah selanjutnya yang harus diambil. Namun, algoritma ini tidak se-intuitif algoritma brute force atau algoritma *greedy* untuk pemrogram.

Oleh karena itu, penulis menggunakan algoritma *greedy* sebagai algoritma pembentukan bot dikarenakan cukup intuitif serta tidak memerlukan waktu atau resource yang terlalu banyak. Selain itu, terdapat cukup banyak kemungkinan solusi *greedy* yang dapat dieksplorasi oleh penulis. Meskipun terdapat peluang bahwa penulis tidak dapat menciptakan algoritma *greedy* yang menghasilkan solusi optimum global, tetapi setidaknya penulis dapat selalu mencapai solusi optimum lokal yang nilainya mendekati solusi optimum global.

2.6 Garis Besar Game Engine Permainan Galaxio

Game engine adalah suatu framework perangkat lunak yang didesain sebagai kumpulan tools dan features yang digunakan untuk membantu development dari suatu game. Umumnya di dalam suatu game engine terdapat beberapa core development tools, diantaranya adalah rendering engine (untuk melakukan render pada grafik 2D atau 3D), physics engine atau collision engine (untuk mengatur bagaimana aturan fisika pada game), sound engine, scripting language, animation engine, artificial intelligence, networking engine, streaming engine, memory management, threading support, localization support, scene graph, serta video support. Pada permainan ini tidak semua tools disediakan oleh pemrogram, hanya beberapa tools esensial.

Pada permainan Galaxio ini, game engine yang digunakan sudah tersedia pada repository yang diunggah sebelumnya oleh Entelect (Entelect merupakan penyelenggara dari challenge ini).

Pada awalnya, starter bot yang didefinisikan oleh *Entellect* belumlah lengkap, masih terdapat file Java yang mengandung fitur-fitur penting belum ditambahkan ke dalam folder src dari starter bot. Oleh karena itu, penulis harus terlebih dahulu mendefinisikan beberapa file Java dan melakukan import seluruh file tersebut ke dalam file bot.java.

Setelah algoritma greedy pada starter bot telah dibuat, maka harus dilakukan kompilasi agar perubahan tersebut dapat diaplikasikan pada file jar dari starter bot. Pada program ini, telah disediakan build tools dari Apache Maven Project. Intellij IDEA telah menyediakan build tools Maven sehingga penulis tidak perlu menambahkan extension apapun di dalam IDEnya. Terdapat beberapa command dari build lifecycle yang dapat dijalankan oleh Maven, diantaranya adalah berikut ini:

- *Validate* : Melakukan validasi apakah seluruh data dan informasi yang dibutuhkan untuk melakukan build project sudah terpenuhi.
- *Compile* : Melakukan kompilasi terhadap source code yang telah dibuat oleh pemrogram.
- test: Melakukan testing menggunakan source code hasil kompilasi terhadap suatu framework yang sudah ditentukan sebelumnya.

- *Package* : Melakukan packaging dari hasil kompilasi source code menjadi suatu file dengan format jar.
- *Verify* : Melakukan tes verifikasi integrasi terhadap file jar yang telah dibangun (menentukan apakah project sudah siap untuk di-install).
- *Install* : Melakukan instalasi project ke dalam local repository beserta dependencies project tersebut.
- *Deploy* : Menyudahi build environment yang telah dibuat sebelumnya dan melakukan copy project ke dalam remote repositories.

Pada program Galaxio ini, untuk melakukan kompilasi perubahan source code dan kemudian mengubahnya ke dalam bentuk file jar, pemrogram hanya perlu menjalankan perintah compile dan install saja. Pemrogram tidak perlu melakukan semua perintah pada build lifecycle.

BAB 3 : Aplikasi Strategi *Greedy*

3.1. Pemetaan Elemen/Komponen Algoritma Greedy pada Bot Permainan Galaxio

3.1.1 Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan Food

Dalam permainan Galaxio, tujuan setiap bot pemain berusaha untuk bertahan hidup hingga saat terakhir. Terdapat banyak cara untuk meraih hal tersebut, salah satunya adalah memilih prioritas makanan yang ingin diambil. Terdapat var nearestFood dan nearestSuperFood yang merupakan variabel yang mengumpulkan jarak food dan superfood dengan bot dalam list secara ascending. Setelah mengambil jarak terdekat, terdapat var fixfood yang membandingkan jarak food dan superfood. Ketika jarak food dengan bot lebih kecil dari 0.8 kali jarak superfood dengan bot, maka var fixfood akan mengambil food terdekat. Sedangkan jika jarak food dengan bot lebih besar dari 0.8 kali jarak superfood dengan bot, maka var fixfood akan mengambil superfood terdekat.

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Jarak semua jenis makanan dari bot.
Himpunan Solusi	Makanan yang memiliki nilai lebih besar dan dekat.
Fungsi Solusi	Memilih makanan yang memiliki nilai lebih besar dan dekat.
Fungsi Seleksi	Memilih berdasarkan data keadaan *game state* saat ini dan fungsi heuristik tingkat prioritas *command* yang harus diikuti. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa apakah jarak makanan yang

	memiliki point lebih besar memiliki jarak yang tidak berbeda jauh dengan makanan yang memiliki point lebih kecil.
Fungsi Objektif	Memaksimalkan ukuran bot dengan cara menambah ukuran bot secepat mungkin.

3.1.2 Pemetaan Elemen/Komponen Algoritma Greedy pada permasalahan *Bot Size*

Greedy by Bot Size adalah pendekatan algoritma greedy yang memprioritaskan perbandingan ukuran bot kami dengan ukuran target. Terdapat variabel dangerzone dimana variabel tersebut menyatakan daerah berbahaya untuk dihindari. Terdapat tiga kondisi dangerzone. Jika ukuran target lebih dari 150, maka dangerzone yang dihindari bot kami sebesar dua kali dari size target. Jika ukuran target diantara 75 dan 150, maka dangerzone yang dihindari bot kami sebesar empat kali dari size target. Jika ukuran target di antara 0 dan 75, maka dangerzone yang dihindari bot kami sebesar enam kali dari size target.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Jarak bot dengan semua player
Himpunan Solusi	Jarak terdekat dengan player lain
Fungsi Solusi	Menghitung jarak player lain dengan bot dan membandingkan size bot dengan bot lawan serta menghindar bila player lain lebih besar dan player lain dalam radius dangerzone
Fungsi Seleksi	Memilih command berdasarkan data

	keadaan game state saat ini dan fungsi heuristik tingkat prioritas *command* yang harus diikuti. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik). Bila bot dalam radius dangerzone, maka bot tidak akan mengejar makanan melainkan mengambil arah berlawanan dengan player terdekat. Bila bot tidak dalam dangerzone, maka bot akan mengambil makanan terdekat.
Fungsi Kelayakan	Jarak bot dengan player terdekat dapat dibandingkan dengan jarak radius dangerzone. Fungsi berjalan.
Fungsi Objektif	Meminimalkan jarak bot dengan player dengan ukuran lebih besar dan memaksimalkan pengejaran bila player terdekat lebih kecil.

3.1.3 Pemetaan Elemen/Komponen Algoritma Greedy Batas Arena/Void

Greedy by Void adalah pendekatan algoritma greedy yang memprioritaskan jarak bot dengan batas arena atau void. Algoritma greedy ini diimplementasikan dengan mengecek jarak bot dari world center atau titik pusat arena. Jarak tersebut ditambahkan dengan 1.5 kali ukuran bot. Jika penjumlahan tersebut lebih besar dari radius arena, maka bot akan menuju arah yang berlawanan dari batas arena. Hal tersebut dimplementasikan dengan menggunakan fungsi getHeadingBetween(worldCenter).

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Memilih heading antara makanan terdekat dengan teleport kearah worldCenter menjauhi void.
Himpunan Solusi	Menjauhi void.
Fungsi Solusi	Memilih antara teleport menjauhi void atau ke makanan terdekat.
Fungsi Seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti. Bentuk fungsi heuristik tersebut menggunakan counter untuk setiap mendekati void apabila counter melewati batas tertentu (sudah terlalu lama dekat void) maka bot akan memanggil command teleport. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa apakah counter sudah melewati batas yang ditentukan.
Fungsi Objektif	Mencari cara untuk menjauhi void seefektif mungkin.

3.1.4 Pemetaan Elemen/Komponen Algoritma Greedy Obstacle

Greedy by obstacle adalah pendekatan algoritma greedy yang memprioritaskan jarak bot dengan obstacle-obstacle pada game seperti gasCloud. Implementasi algoritma greedy ini dimulai dengan mengambil semua jarak objek yang bertipe gasCloud dengan bot dan dimasukkan ke dalam list secara *ascending*. Kemudian, program akan mengecek jarak gascloud terdekat dengan bot kami. Jarak tersebut akan dibandingkan dengan 3 kali ukuran bot kami. Bila jarak tersebut lebih kecil dari 3 kali ukuran bot, maka bot akan bergerak dengan arah yang berlawanan dengan gasCloud. Jika tidak, maka bot akan lanjut mengejar bot ataupun mencari makanan terdekat.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Jarak obstacle dengan bot.
Himpunan Solusi	Jarak terdekat bot dengan obstacle.
Fungsi Solusi	Melakukan pengecekan objek di sekitar bot
Fungsi Seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti. Bentuk fungsi heuristik tersebut mengambil arah sebaliknya dari obstacle. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa keadaan sekitar berdasarkan data keadaan state saat itu.
Fungsi Objektif	Mencari cara yang paling efektif menghindari obstacle.

3.1.5 Pemetaan Elemen/Komponen Algoritma Greedy *FIRETORPEDOES*

Greedy of fire torpedoes adalah pendekatan algoritma greedy yang memprioritaskan menembak player ketika berada dalam radius penembakan baik dalam keadaaan dikejar player lain maupun mengejar player lain. Untuk memulai penembakan, terdapat var targetIsPlayer. Bila targetIsPlayer bernilai true, bot dapat menembakkan fire torpedoes. Bila targetIsPlayer bernilai false, bot tidak dapat menembakkan fire torpedoes. Kondisi tersebut dapat dicapai bila bot lebih kecil dari player dan bot dalam dangerzone. Selain itu, kondisi tersebut juga dapat dicapai bila bot lebih besar dari player dan bot dalam radius yang dekat dengan player tersebut.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Offensif torpedoes atau defensif torpedoes.
Himpunan Solusi	Menembak ke arah bot lawan.
Fungsi Solusi	Mengecek jarak dan ukuran bot lawan dengan bot.
Fungsi Seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti. Bentuk fungsi heuristik tersebut menembakan torpedoes ke lawan atau tidak. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa keadaan sekitar berdasarkan

	data keadaan state saat itu.
Fungsi Objektif	Mencari cara agar bot menang atau bertahan hidup dari bot lawan.

3.1.6 Pemetaan Elemen/Komponen Algoritma Greedy *FIRETELEPORT dan TELEPORT*

Greedy by fireteleport dan teleport adalah pendekatan algoritma greedy yang memprioritaskan berpindah tempat. Terdapat dua kondisi yang menyebabkan bot melakukan teleport. Kondisi pertama disaat bot berada di tepi arena atau void. Di situasi ini, bot akan menembakkan fireteleport ke arah worldCenter guna menjauhi batas arena. Kondisi kedua disaat player terdekat dengan bot memiliki ukuran lebih kecil dari bot. Dalam situasi tersebut, bot akan menembakkan fireteleport ke arah player terdekat guna memakan player tersebut.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Ofensif teleport atau teleport menjauhi void.
Himpunan Solusi	Menembak ke arah bot lawan atau ke worldCenter menjauhi void
Fungsi Solusi	Mengecek jarak dan ukuran bot lawan dengan bot. Mengecek jarak bot dengan void.
Fungsi Seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti. Bentuk fungsi heuristik tersebut memilih menembakan teleport

	atau tidak. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa keadaan sekitar berdasarkan data keadaan state saat itu.
Fungsi Objektif	Mencari cara agar bot memakan lawan atau menjauhi void

3.1.7 Pemetaan Elemen/Komponen Algoritma Greedy *SHIELD*

Greedy by shield adalah pendekatan algoritma greedy yang memprioritaskan berlindung dari firetorpedoes musuh. Shield akan muncul dalam kondisi jika terdapat torpedo di sekitar bot. Shield digunakan untuk melindungi bot dari torpedo agar ukuran bot tidak berkurang.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Berlindung dari torpedoes
Himpunan Solusi	Menyalakan *Shield*
Fungsi Solusi	Mengecek lokasi torpedoes. Menyalakan shield.
Fungsi Seleksi	Memilih command berdasarkan data keadaan game state saat tersebut serta fungsi heuristik tingkat prioritas command yang harus diikuti. Bentuk fungsi heuristik tersebut memilih menyalakan shield atau tidak. Tingkat prioritas tersebut bersifat

	statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa keadaan sekitar berdasarkan data keadaan state saat itu.
Fungsi Objektif	Menyalakan shield untuk memantulkan torpedoes.

3.1.8 Pemetaan Elemen/Komponen Algoritma Greedy *SUPERNOVA*

Greedy by supernova adalah pendekatan algoritma greedy yang memprioritaskan mengambil supernova dan menembakkannya ke player lain bila memiliki supernova. Supernova bersifat *rare* karena hanya terdapat satu dalam satu permainan yang muncul di awal permainan atau di akhir permainan. Untuk mengejar supernova, program akan mengecek ukuran bot. Bila ukuran bot lebih dari 150, maka zuperzone bernilai sebesar 2.5 kali dari ukuran bot. Bila ukuran bot diantara 75 dan 150, maka zuperzone bernilai sebesar 5 kali dari ukuran bot. Bila ukuran bot diantara 0 dan 75, maka zuperzone bernilai sebesar 7 kali dari ukuran bot.

Nama Elemen/Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Cek objek bertipe supernova.
Himpunan Solusi	Menuju supernova.
Fungsi Solusi	Apabila supernova berada dalam radius zuperzone, bot akan menuju supernova.
Fungsi Seleksi	Mencari semua objek dalam data game object dimana object tersebut bertipe

	supernova. Bila terdapat supernova, program akan mengecek apakah supernova berada di dalam zuperzone. Terdapat fungsi heuristik tingkat prioritas command harus diikuti. Tingkat prioritas tersebut bersifat statik selama permainan berlangsung (tidak ada perubahan kondisi pada fungsi heuristik).
Fungsi Kelayakan	Memeriksa keadaan sekitar berdasarkan data keadaan state saat itu.
Fungsi Objektif	Mengambil supernova agar dapat menembakkan supernova kepada player lain.

3.2. Eksplorasi Alternatif Solusi Algoritma Greedy pada Bot Permainan Galaxio

Terdapat banyak sekali alternatif solusi algoritma *greedy* yang dapat diimplementasikan pada bot permainan Galaxio. Hal ini dikarenakan terdapat banyak elemen dari game engine yang dapat diakses oleh bot dan kemudian diubah state pada elemen tersebut. Selain itu, elemen-elemen tersebut saling berinteraksi dengan elemen lainnya sehingga jika terdapat perubahan pada suatu state elemen, terdapat peluang bahwa state lainnya berubah pula. Berikut ini merupakan beberapa alternatif strategi algoritma greedy yang dapat diimplementasikan pada bot pemain :

3.2.1 Strategi Heuristik *Food* dan *SuperFood*

Strategi pertama yang diimplementasikan adalah melakukan pencarian makanan terdekat dan membandingkan jarak dan jenis kedua makanan tersebut. Strategi ini akan memilih hasil yang optimal seperti, memilih superFood apabila jarak food dengan bot dan superfood dengan bot tidak berbeda jauh. Strategi ini merupakan strategi yang paling

ringkas dan mudah diimplementasi, Hal ini karena bot hanya fokus mengejar dan memilih makanan yang optimal untuk menambah ukuran bot.

3.2.2 Strategi Heuristik *Bot Size*

Strategi berikutnya yang diimplementasikan adalah melakukan pencarian musuh terdekat dan membandingkan ukuran musuh dengan bot kita. Apabila ukuran musuh lebih kecil maka bot akan mengubah arah ke musuh tersebut. Begitupun sebaliknya, apabila ukuran musuh lebih besar, maka bot akan mengarah ke arah yang berlawanan dari posisi musuh. Strategi ini digunakan untuk mengakuisisi musuh atau juga bertahan hidup agar tidak termakan oleh musuh.

3.2.3 Strategi Heuristik Batas Arena/*Void*

Strategi berikutnya yang diimplementasikan adalah melakukan perubahan arah apabila bot dekat dengan batas arena atau *void.* Bot akan berubah arah menuju titik tengah dunia dan juga ke arah tegak lurus dengan arah titik tengah dunia. Apabila bot ‘terjepit’ - memantul berulang kali ke batas arena atau *void*, maka bot akan menembakkan teleporter ke arah titik tengah dunia dan melakukan teleport agar menjauh dari batas arena atau *void*.

3.2.4 Strategi Heuristik Obstacle

Strategi berikutnya yang diimplementasikan adalah mengubah arah bot agar berlawanan dengan obstacle. Bot akan berubah arah agar berlawanan atau memutari obstacle yang ada.

3.2.5 Strategi Heuristik *Fire Torpedoes*

Strategi berikutnya yang diimplementasikan adalah menembakkan *fire torpedoes* ke musuh. Bot akan menembakkan *fire torpedoes* ke arah musuh pada jarak dekat tertentu dan juga apa bila ukuran bot lima kali lebih besar daripada musuh. Hal ini dilakukan agar penembakkan *fire torpedoes* tidak meleset dan tidak menyebabkan bot mati karena menembakkan *fire torpedoes*.

3.2.6 Strategi Heuristik *Fire Teleport* dan *Teleport*

Strategi berikutnya yang diimplementasikan adalah menembakkan *teleport* serta melakukan *teleport.* Bot akan menembakkan teleport pada dua kondisi. Yang pertama, apabila bot terlalu lama dekat dengan *void* atau batas arena. Jika bot terlalu lama dekat dengan *void*, bot akan menembakkan *teleport* ke arah titik tengah dunia untuk menjauh dari *void*. Yang kedua, bot akan teleport ke arah musuh. Bot akan menembakkan *teleport* ke arah musuh terdekat apabila ukuran musuh lebih kecil daripada bot serta ukuran bot tidak terlalu kecil. Hal ini dilakukan agar apabila bot tidak mati saat menembak *teleport*. Bot akan melakukan *teleport* ketika *teleport* sudah berada di dekat musuh dan ukuran bot lebih besar dari pada musuh.

3.2.7 Strategi Heuristik *Shield*

Strategi berikutnya yang diimplementasikan adalah menyalakan *shield*. *Shield* akan menyala apabila terdapat *Fire Torpedoes* di dekat bot. *Shield* ini akan menyala untuk melindungi bot dari dan memantulkan *Fire Torpedoes*.

3.2.8 Strategi Heuristik *Supernova*

Strategi terakhir yang diimplementasikan adalah mengambil dan menembakkan *Supernova*. Bot akan mencari *Supernova* terdekat apabila terdapat *Supernova* pada map. Setelah bot mendapatkan *supernova*, bot akan langsung menembakkan *supernova* ke arah musuh dan setelah tick tertentu *supernova* akan meledak.

3.3 Analisis Efisiensi dari Kumpulan Solusi Algoritma Greedy

3.3.1 Efisiensi Strategi Heuristik Food dan Superfood

Strategi ini sangat efisien karena bot mengambil makan yang terdekat dengan dirinya dan juga mengambil superfood terdekat yang memiliki keuntungan yang lebih besar. Hal ini membuat bot bertumbuh tanpa perlu waktu yang lama.

3.3.2 Efisiensi Strategi Heuristik Bot Size

Strategi ini mempertimbangkan ukuran bot dengan ukuran player lain. Disaat ukuran player lain lebih besar, bot akan pergi berlawanan arah untuk menghindari player agar tidak termakan oleh kapal lain. Begitu juga disaat ukuran bot lebih besar dari ukuran player, bot akan memakan kapal tersebut.

3.3.3 Efisiensi Strategi Heuristik Batas Arena / Void

Strategi ini mempertimbangkan jarak bot dengan batas arena atau void. Disaat jarak bot sangat dekat dengan batas arena, bot akan pergi ke arah berlawanan dari batas arena. Hal ini mencegah berkurangnya ukuran bot.

3.3.4 Efisiensi Strategi Heuristik Obstacle

Strategi ini mempertimbangkan jarak obstacle berupa gas cloud dengan bot. Disaat jarak bot sangat dekat dengan gas cloud, bot akan pergi ke arah berlawanan dari gas cloud. Hal ini mencegah berkurangnya ukuran bot.

3.3.5 Efisiensi Strategi Heuristik *FIRETORPEDOES*

Strategi ini mempertimbangkan penembakan player lain agar ukuran player lain berkurang. Strategi ini efisien karena lebih mudah untuk mengalahkan player lain.

3.3.6 Efisiensi Strategi Heuristik *TELEPORT* dan *FIRETELEPORT*

Strategi ini mempertimbangkan perpindahan tempat secara cepat. Misalnya pada kasus menembakkan fireteleport kepada player dengan ukuran lebih kecil untuk memakan kapal player lain untuk menambah ukuran bot itu sendiri.

3.3.7 Efisiensi Strategi Heuristik *SHIELD*

Strategi ini mempertimbangkan untuk melindungi diri dari tembakan torpedo player lain. Hal ini upaya untuk mempertahankan ukuran bot agar tidak berkurang secara drastis.

3.3.8 Efisiensi Strategi Heuristik *SUPERNOVA*

Strategi ini mempertimbangkan untuk mengambil supernova agar dapat menyerang musuh dengan supernova bomb. Strategi ini efisien guna untuk mengalahkan player lain.

3.4. Analisis Efektivitas dari Kumpulan Solusi Algoritma Greedy

3.4.1. Efektivitas Strategi Heuristik Food dan Superfood

Strategi ini belum tentu selalu berdampak baik terhadap bot, Hal ini karena meskipun food menambahkan ukuran bot, bot tersebut bisa saja mendekati bot lawan yang memiliki ukuran lebih besar.

3.4.2. Efektivitas Strategi Heuristik Bot *Size*

Menurut penulis, strategi ini efektif untuk dapat bertahan hidup karena dapat menghindar dari musuh dan dapat memakan musuh yang lebih kecil.

3.4.3 Efektivitas Strategi Heuristik Batas Arena / *Void*

Pada bagian ini, efektivitas menghindari void sangatlah bagus, Akan tetapi apabila menjauhi void dengan melakukan teleport, ada kemungkinan bot akan teleport mendekati player lawan dan berisiko mati.

3.4.4 Efektivitas Strategi Heuristik Obstacle

Pada bagian ini, efektivitas menghindari obstacle cukup baik,Namun jika ada bot lawan yang dekat dengan player, bot akan lebih memilih menghindari obstacle dibandingkan bot lawan

3.4.5 Efektivitas Strategi *FIRETORPEDOES*

Efektivitas strategi ini sangatlah tinggi, bot bisa menembakan torpedoes secara defensif dan ofensif. Bot hanya akan menembak torpedoes apabila jarak dengan bot lawan dekat. Bot lawan tidak akan menyangka bahwa bot yang lebih kecil dan dikejar akan balik melawan dan menembak torpedoes secara beruntun.

3.4.6 Efektivitas Strategi FIRETELEPORT dan TELEPORT

Efektivitas strategi ini cukup baik, bot akan melakukan teleport saat terlalu lama berada di dekat void, dan bot juga bisa melakukan teleport ke bot lawan yang jauh lebih kecil.

3.4.7 Efektivitas Strategi *SHIELD*

Efektivitas strategi ini sangatlah baik, bot akan menyalakan shield ketika ada torpedo di dekatnya, bot juga akan mengantisipasi serangan balik lawan dengan menyalakan *shield* ketika menyerang.

3.4.8 Efektivitas Strategi Supernova

Efektivitas strategi Supernova cukup, bot akan mengambil supernova apabila berada di dekat bot dan akan langsung menembakkan ke bot lawan terdekat.

3.5. Strategi Greedy yang Digunakan pada Program Bot

Pada point sebelumnya, telah dijabarkan beberapa pendekatan greedy yang digunakan pada tugas besar kali ini bergantung pada posisi, keuntungan, dan kerugian dari bot kami. Strategi heuristik yang penulis ambil sebagai algoritma *greedy* utama dari program *bot* permainan Galaxio ini adalah penggabungan dari seluruh strategi yang telah dipaparkan sebelumnya.

Agar seluruh strategi heuristik dapat digabungkan menjadi suatu algoritma *greedy*, penulis harus mendefinisikan urutan dari strategi mana yang harus dieksekusikan terlebih dahulu. Salah satu cara mendefinisikan urutan tersebut adalah dilihat dari seberapa besar prioritas suatu strategi jika dibandingkan dengan strategi lainnya.

Setelah dipertimbangkan dan dicoba pada *game engine*,kami telah berhasil memformulasikan urutan pengeksekusian strategi heuristik yang menghasilkan algoritma *greedy* paling optimum.Algoritma *bot* lawan yang digunakan sebagai tolak ukur adalah *reference bot* bawaan yang dibuat secara *random*, Urutan prioritas dari pengeksekusian strategi heuristik yang telah diformulasikan oleh penulis adalah sebagai berikut :

- Strategi memakan food atau Superfood terdekat dari bot
- Strategi ketika ada Player lain di sekitar bot
- Strategi ketika ada Gas Cloud di dekat bot
- Strategi ketika bot berada di dekat Batas Map/*Void*
- Strategi situasi penggunaan Teleporter
- Strategi situasi menggunakan Torpedoes

BAB 4: Implementasi dan Pengujian

4.1. Repository Github

Bot yang kami buat dapat diakses melalui repository Github pada tautan berikut : https://github.com/zaydanA/Tubes1_Restu-Ibu

Bonus :

https://youtu.be/izqWlyy1sL4

4.2. Implementasi Algoritma Greedy pada Bot Permainan Galaxio

botService.java

```
KAMUS LOKAL

bot : GameObject

playerAction : PlayerAction

gameState : GameState

target : GameObject

targetIsPlayer : boolean

shieldactive : boolean

worldCenter : GameObject

udah : boolean

kejepitctr : int

count : int

temptick : int

teletarget : boolean

teletarget1 : boolean

telekita : GameObject

telectr : int

issupernova : boolean

tempticksupernova : int
```

```
ALGORITMA

targetIsPlayer <- false

shieldactive <- false

udah <- false

kejepitctr <- 0

count <- 0

teletarget1 <- false

telectr <- 0

issupernova <- false

tempticksupernova <- 0
```

```
procedure computeNextPlayerAction()

KAMUS LOKAL

worldC : GameObject

nearestPlayer : List<GameObject>

teleporter : List<GameObject>

nearesttorpedo : List<GameObject>


ALGORITMA

//Inisiasi data yang diperlukan


// Mencari target/musuh terdekat

nearestPlayer <- getPlayerGameObjects(target.id !=

bot.id).sorted(getDistanceBetween(bot,target))


// Mencari teleporter terdekat

teleporter <- getGameObjects(tele.getGameObjectType =

TELEPORTER).sorted(getDistanceBetween(bot,tele))
```

```
// Mencari torpedo terdekat
nearesttorpedo <- getGameObjects(torpedo.getGameObjectType =
TORPEDOSALVADO).sorted(getDistanceBetween(bot, torpedo))



// Mendapatkan titik tengah dunia
worldC <- GameObject(null, null, null, null, world.getCenterPoint(), null)
//mendapatkan titik tengah
distanceFromWorldCenter <- getDistanceBetWeen(bot, worldC)



if(target = null or target = worldCenter) then
      heading <- resolveNewTarget()
else
      if(targetWithNewValues = null or !targetWithNewValues.isEmpty)
            heading <- resolveNewTarget()
      else
            target <- targetWithNewValues[0]
            if(1.25 * target.size < bot.getSize())
                  heading <- getHeadingBetween(target)
                  teletarget <- false
                  if(bot.getSize() > 50 and telectr > 80)
                        playerAction <- PlayerActions.TELEPORT
                        telectr <- 0
            else
                  heading <- resolveNewTarget()
```

```
if(!teleporter.IsEmpty) then
      telekita <- teleporter[0]
      teletarget <- true

if(distanceFromWorldCenter + (1.5 * bot.getSize()) > world.getRadius()) then
//mengecek apakah bot berada di dekat batas dunia
      worldCenter <- new GameObject(null, null, null, null, new Position(0,0),
null)
      heading <- getHeadingBetween(worldC)
      target <- worldCenter
      count++
      kejepitctr++

if(targetIsPlayer) then
      heading <- getHeadingBetween(nearestPlayer[0])
      playerAction <- FIRETORPEDOES

if(kejepitctr > 10) then
      heading <- getHeadingBetween(worldC) + 90

if(kejepitctr > 20) then
      heading <- getHeadingBetWeen(worldC)
      temptick <- world.GetCurrentTick() + 10
      playerAction <- FIRETELEPORT
      nembaktele <- true
      kejepitctr <- 0

if(teletarget1) then
```

```
      if(getDistanceBetween(telekita, nearestPlayer[0] <= 75 + bot.getSize*()
/ 2 and teletarget1)) then
            playerAction <- TELEPORT
            teletarget1 <- false
            teletarget <- false

if(teletarget && bot.getSize() > 50)
      heading <- getHeadingBetween(nearestPlayer[0])
      playerAction <- FIRETELEPORT
      teletarget <- false
      teletarget1 <- true
if(!nearesttorpedo.isEmpty()) then
       if (getDistanceBetween(nearesttorpedo[0], bot) < 2 * bot.getSize() &&
bot.getSize() > 60)then
       shieldactive <- true

if(shieldactive) then // Aktivasi shield
       playerAction.action <- PlayerActions.ACTIVATESHIELD;
       shieldactive <- false

if(bot.supernovaAvailable > 0) then
       playerAction.action <- PlayerActions.FIRESUPERNOVA
       playerAction.heading <- getHeadingBetween(nearestPlayer[0])
       issupernova <- true
       tempticksupernova <- gameState.world.getCurrentTick() + 20

if(issupernova)then
       supernovactr++
```

```
if(tempticksupernova = gameState.world.getCurrentTick()) then
       playerAction.action <- PlayerActions.DETONATESUPVERNOVA
       issupernova <- false

playerAction.heading <- heading
playerAction.action <- playerAction
```

```
function resolveNewTarget() -> int
KAMUS LOKAL
heading : int
fixfood : GameObject
worldC2 : GameObject
nearestFood : List<GameObject>
nearestSuperFood : List<GameObject>
nearestPlayer : List<GameObject>
nearestGasCloud : List<GameObject>
zupernova : List<GameObject>
direction2NearestPlayer : int
dangerzone <- double
zuperzone <- double


ALGORITMA
// Inisiasi data yang diperlukan
// Mendapatkan target terdekat
nearestPlayer <- getPlayerGameObjects(target.id !=
bot.id).sorted(getDistanceBetween(bot,target))
```

```
// Mendapatkan makanan terdekat
nearestFood <- getGameObjects(item.getGameObjectType() =
FOOD).sorted(getDistanceBetween(bot,item))

// Mendapatkan makanan super terdekat
nearestSuperFood <- getGameObjects(superfood.getGameObjectType() =
SUPERFOOD).sorted(getDistanceBetween(bot,superfood))

// Mendapatkan Gas Cloud terdekat
nearestGasCloud <- getGameObjects(gascloud.getGameObjectType() =
GASCLOUD).sorted(getDistanceBetween(bot,gascloud))

// Mendapatkan Supernova terdekat
nearestSupernova <- getGameObjects(supernova.getGameObjectType() =
SUPERNOVA).sorted(getDistanceBetween(bot,supernova))

if(!nearestPlayer.isEmpty()) then
      "You Win!"

direction2NearestPlayer <- getHeading(nearestPlayer[0])

// Mencari lebih dekat ke food atau superfood
if(getDistanceBetween(nearestFood[0, bot] < 0.8 *
getDistanceBetween(nearestSuperFood[0], bot))) then
      fixfood <- nearestFood[0]
else
      fixfood = nearestSuperFood[0]
```

```
if(getDistanceBetween(fixfood, bot) < getDistanceBetween(nearestGasCloud[0],
bot) or (getDistanceBetween(nearestPlayer[0], bot) <
getDistanceBetween(nearestGasCloud[0], bot))) then

      //Bot lebih kecil daripada musuh
      if(nearestPlayer[0].getSize() > bot.getSize()) then
            dangerzone <- 0
            zuperzone <- 0

            if (nearestPlayer[0].getSize() > 150) then
                  dangerzone <- 1.5 * nearestPlayer[0].getSize()
            else if (nearestPlayer[0].getSize() > 75) then
                  dangerzone <- 4 * nearestPlayer[0].getSize()
            else if (nearestPlayer[0].getSize() > 0)
                  dangerzone <- 6 * nearestPlayer[0].getSize()

            if(bot.getSize() > 150) then
                  zuperzone = 2.5 * bot.getSize()
            else if(bot.getSize() > 75) then
                  zuperzone = 5 * bot.getSize()
            else if(bot.getSize() > 0) then
                  zuperzone = 7 * bot.getSize()

            if(getDistanceBetween(nearestPlayer[0], bot) < dangerzone) then
                  udah <- false
                  heading <- GetOppositeDirection(nearestPlayer[0], bot])
                  target <- fixfood
```

```
                  if(nearestPlayer[0].getSize() < 5 * bot.getSize()) then
                        targetIsPlayer <- true
                  else
                        targetIsPlayer <- false
            else if (!nearestSupernova.isEmpty() and
getDistanceBetween(nearestSupernova[0], bot) < zuperzone)
                  heading <- getHeading(nearestSupernova[0])
                  target <- fixfood
                  targetIsPlayer <- false
            else
                  heading <- getHeadingBetween(fixfood)
                  target <- fixfood
                  targetIsplayaer <- false

      // Bot lebih besar daripada musuh
      else if (nearestPlayer[0].getSize() < 1.25 * bot.getSize()) then
            udah <- false
            if(getDistanceBetween(nearestGasCloud[0], bot) < 3 *
bot.getSize()) then // Menghindar dari Gas Cloud
                  heading <- GetOppositeDirection2(nearestGasCloud[0], bot)
            else
                  if(getDistanceBetween(nearestPlayer[0], bot) < (4 *
bot.getSize())) then // Bot mengejar musuh
                        heading <- direction2NearestPlayer
                        target <- nearestPlayer[0]
                        if(getDistanceBetween(nearestPlayer[0], bot) < 5 *
bot.getSize()) then
```

```
                              targetIsPlayer <- true
                  else // Bot mencari makan
                        heading <- getHeadingBetween(fixfood)
                        target <- fixfood
                        targetIsPlayer <- false

      else if (fixfood != null) then
            heading <- getHeadingBetween(fixfood)
            target <- fixfood
            targetIsPlayer <- false
      else
            target <- nearestPlayer[0]
            worldC2 <- new new GameObject(null, null, null, null,
world.getCenterPoint(), null)
            heading <- getHeadingBetween(worldC2)
            targetIsPlayer <- false
else // Menghindar dari Gas Cloud
      dangerzone1 <- 0
      dangerzone2 <- 0
      if (nearestPlayer[0].getSize() > 150) then
            dangerzone1 <- 2 * bot.getSize()
        dangerzone2 <- 0.5 * bot.getSize()
      else if (nearestPlayer[0].getSize() > 75) then
            dangerzone1 <- 4 * bot.getSize()
            dangerzone2 <- 1.5 * bot.getSize()
      else if (nearestPlayer[0].getSize() > 0) then
            dangerzone1 <- 8 * bot.getSize()
            dangerzone2 <- 3 * bot.getSize()
```

```
       if (getDistanceBetween(nearestPlayer[0], bot) < (dangerzone1)) then
            heading <- GetOppositeDirection2(nearestPlayer[0], bot)
            target <-  fixfood
      else if (getDistanceBetween(nearestGasCloud[0], bot) < dangerzone2) then
            heading <- GetOppositeDirection2(nearestGasCloud[0], bot)
            target <- fixfood
      else
            heading <- GetOppositeDirection2(nearestGasCloud[0], bot)
            target <- nearestPlayer[0]
      count++
if(target<-worldCenter) then
      heading <- getHeadingBetween(fixfood)

-> heading
```

```
function getDistanceBetween(GameObject object1, GameObject object2) ->
double
        var triangleX <- Math.abs(object1.getPosition().x -
object2.getPosition().x)
        var triangleY <- Math.abs(object1.getPosition().y -
object2.getPosition().y)
        -> Math.sqrt(triangleX * triangleX + triangleY * triangleY);
    }
```

```
function getHeadingBetween(GameObject otherObject) -> int
        var direction <- toDegrees(Math.atan2(otherObject.getPosition().y -
bot.getPosition().y,
                otherObject.getPosition().x - bot.getPosition().x))
```

```
        -> (direction + 360) % 360
```

```
function toDegrees(double v) -> int{
        -> (int) (v * (180 / Math.PI))
```

```
function toDegrees2(double v, boolean p) -> int
        count = 0;
        if(p) then
            return (int) (v * (180 / Math.PI) - 90)
        else
            -> toDegrees(v)
```

```
function GetOppositeDirection(GameObject gameObject1, GameObject
gameObject2) -> int
        -> toDegrees(Math.atan2(gameObject2.getPosition().y -
gameObject1.getPosition().y, gameObject2.getPosition().x -
gameObject1.getPosition().x))
```

```
function GetOppositeDirection2(GameObject gameObject1, GameObject
gameObject2) -> int
        if(count > 1) then
        udah <- true
        -> toDegrees2(Math.atan2(gameObject2.getPosition().y -
gameObject1.getPosition().y, gameObject2.getPosition().x -
gameObject1.getPosition().x), udah)
```

4.3. Penjelasan Struktur Data pada Bot Permainan Galaxio

Pada bot *Galaxio* yang berbasis bahasa Java, pendekatan yang digunakan adalah dengan pemrograman berbasis objek, dimana data akan dienkapsulasi dalam kelas-kelas. Dalam proses pembuatannya, kami juga menambahkan beberapa kelas dan struktur data dalam keseluruhan program untuk melengkapi implementasi bot. Program yang kami kembangkan secara umum terbagi menjadi lima bagian besar, yaitu kelas-kelas Models, kelas-kelas Services, kelas-kelas Enums, dan Main.

Berikut pemaparan lebih mendalam mengenai beberapa class yang ada pada bot Galaxio

A. Kategori *Enums*

Kelas – kelas pada kategori ini merupakan aksi/ability yang bisa dilakukan oleh bot, semua class disini merupakan implementasi dari PlayerActions.java.

i. PlayerActions.java

Kelas PlayerActions berisi enumerasi dari aksi yang bisa dilakukan oleh bot, yaitu FORWARD,STOP,START,START_AFTERBURNER),STOP_AFTERBURNER,FIRETORPEDOES,FIRE_SUPERNOVA,DETONATE_SUPVERNOVA,FIRETELEPORT,TELEPORT,ACTIVATE_SHIELD.

ii. ObjectTypes.java

Kelas ObjectTypes berisi enumerasi dari semua objek yang ada pada game *Galaxio*, yaitu PLAYER,FOOD,WORMHOLE,GASCLOUD, ASTEROIDFIELD, TORPEDOSALVADO,SUPERFOOD,SUPERNOVAPICKUP,SUPERNOVABOMB,TELEPORTER,SHIELD.

B. Kategori *Models*

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

i. GameObject.java

Kelas GameObject berisi atribut-atribut yang dapat dimiliki oleh game object, seperti (UUID) id,(Integer) size,(Integer) speed,(Integer) currentHeading,(Position) position,dan (objectTypes) gameObjectType.

ii. GameState.java

Kelas GameState berisi atribut-atribut yang menggambarkan kondisi state game pada saat tick ini, seperti (World) world,(List<gameObject>) gameObject,dan (List<gameObject>) playerGameObjects,

iii. GameStateDto.java

Kelas GameStateDto berisi atribut-atribut yang menjadi pengirim informasi dan penerima informasi antara gamerunner dengan gamestate,seperti (World) world, (Map<String, List<Integer>>) gameObjects,dan (Map<String, List<Integer>>) playerObjects.

iv. PlayerAction.java

Kelas PlayerAction berisi aksi yang sedang dijalankan bot, kelas ini memiliki atribut (UUID) playerId,(PlayerActions) action, dan (int) heading.

v. Position.java

Kelas Position berisi data atribut posisi dari bot, posisi berbentuk dalam kartesius ((int) x, (int) y).

vi. World.java

Kelas World berisi data atribut-atribut pada world, seperti (Position) centerPoint, (Integer) Radius, (Integer) currentTick.

C. Kategori *Services*

i. BotService.java

Kelas ini berisi implementasi dari algoritma *greedy* pada bot. Pada kelas ini, terdapat beberapa atribut dan fungsi yang berguna untuk menjalankan bot tersebut.

- Informasi Bot lawan
  Berisi informasi lawan terdekat, meliputi id,size dll.
- Informasi GameState
  Informasi ini berisi tentang state pada tick saat ini
- Informasi GameObject
  Informasi ini berisi gameObject yang diambil infonya oleh bot.
- Informasi int
  - kejepitctr : Counter untuk mengetahui apakah bot terjepit saat berada di dekat void.
  - count : Counter untuk mengatur perubahan arah pada fungsi GetOppositeDirection().
  - temptick : Counter untuk melakukan timer saat teleport menjauhi void (menuju worldcenter).
  - telectr : Counter ntuk mengatur banyaknya teleporter sesuai dengan hitungan tick game.
  - supernovactr : Counter untuk mengatur jarak supernova sesuai dengan hitungan tick game.
  - tempticksupernova : Counter untuk melakukan pencocokan timer dengan tick game ketika supernova sudah ditembakkan.
- Informasi boolean
  - targetIsPlayer : Boolean untuk menentukan apakah akan menembakkan *fire torpedoes* ke target atau tidak.
  - shieldactive : Boolean untuk menentukan apakah akan mengaktifkan shield atau tidak.

- udah : boolean untuk menentukan pemanggilan dua fungsi, yaitu antara toDegrees atau toDegrees2.
- nembaktele : Boolean untuk menentukan apakah bot telah menembakkan teleport atau belum.
- teletarget : Boolean untuk menetukan apakah akan menembakkan teleport ke target atau tidak.
- teletarget1 : Boolean untuk menunjukan bahwa bot telah menembakkan teleport.
- issupernova : Boolean untuk menentukan apakah bot memiliki supernova atau tidak.

D. Main

Pada bagian Main, terdapat *entrypoint* dari bot. *Entrypoint* adalah bagian kode pertama yang dijalankan saat bot pertama kali dipanggil dari *executable*-nya. Isi dari kelas main hanyalah instansialisasi hal-hal teknis yang berkaitan dengan penerimaan game state dan penerjemahan command dari bot menuju game engine.

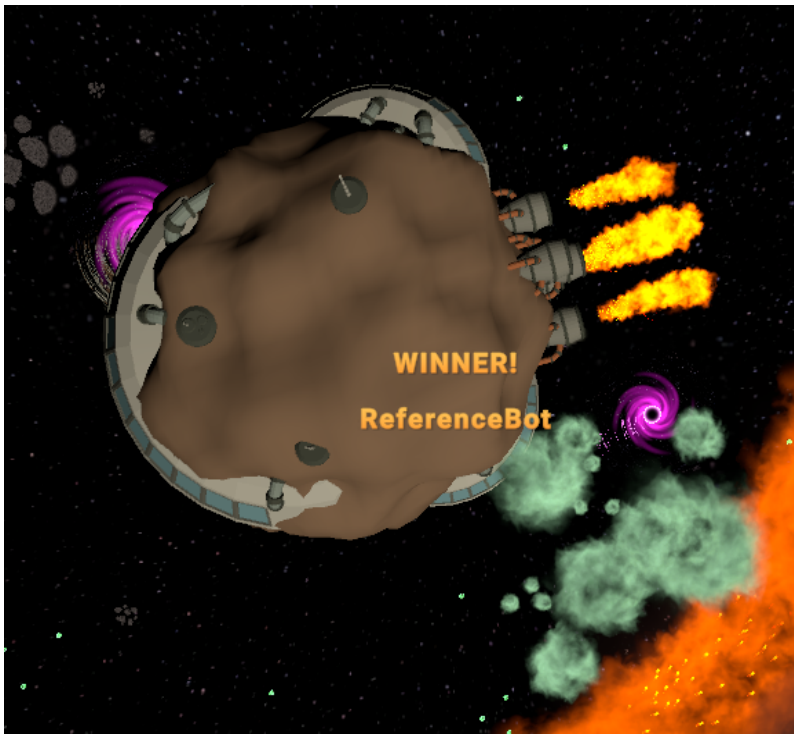
4.4. Pengujian Bot serta Analisis Performansi Bot Permainan Galaxio

Pengujian bot dilakukan dengan melawankan bot kami dengan ReferenceBot dan Bot kami sendiri. Namun, logika dari reference bot ini masih sangatlah buruk, bahkan beberapa action tidak digunakan sama sekali, sehingga kesulitan untuk mendapatkan parameter yang valid untuk pengetesan bot kami. Oleh karena itu, kami menambahkan bot kami sendiri untuk membantu mendapatkan parameter untuk pengetesan bot.

Permainan *Galaxio* ini memiliki sangat banyak faktor dan cukup bergantung pada lawan serta keberuntungan, perlu banyak pengujian untuk mendapatkan hasil yang sesuai. Contoh faktor keberuntungan adalah posisi spawn object pada setiap runner dijalankan yang bersifat random. Oleh karena itu, dilakukan pengujian sebanyak 5 kali, dan digunakan visualizer untuk melihat hasil yang didapat. Berikut hasil akhir 5 match tersebut :

Match 1 :

Match 2 :



Match 3 :

Match 4 :

Match 5 :

Setelah dilakukan lima kali pertandingan melawan tujuh *reference bot*, terlihat bot kalah sekali dengan reference bot yang disebabkan oleh faktor objek - objek yang terdapat pada map. Hal ini juga dapat disebabkan oleh algoritma reference bot yang terbilang cukup ambigu yang mengakibatkan reference bot tidak menggunakan algoritma greedy. Contoh keambiguan reference bot adalah apabila reference bot lebih besar dari pada bot kami, *reference* bot akan langsung mengejar bot kami walaupun jaraknya sangatlah jauh sehingga saat kami melakukan pengetesan 7 *reference* bot melawan 1 bot player, bot player kewalahan menghadapi *reference* bot yang begitu banyak mengejar bot player.

Menurut analisis yang telah dilakukan, bot kami dapat kalah karena kami mengutamakan **greedy by food.** Hal ini mengakibatkan bot kami tidak mengincar suatu pemain secara langsung yang membuat reference bot memakan reference bot lainnya sehingga suatu reference bot tertentu menjadi sangat besar dan tidak ada ruang yang cukup untuk bot kami untuk mencari makanan.

Analisis lain yang telah dilakukan adalah ketika bot kami menang. Hal ini dikarenakan bot kami melakukan **greedy by player**. Hal ini membuat bot kami menyerang bot musuh apabila ukuran bot kami lebih besar daripada bot musuh. Selain itu reference bot juga belum mempertimbangkan objek - objek yang dapat membahayakan reference bot tersebut, sehingga reference bot dapat mati bukan karena dimakan. Hal ini dimanfaatkan oleh bot kami dengan

menggunakan **greedy by obstacle** yang membuat bot kami menghindar objek - objek atau daerah yang dapat membahayakan bot.

BAB 5: Kesimpulan dan Saran

5.1 Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma algoritma *greedy* untuk membuat bot permainan *Galaxio* yang setidaknya bisa menjalankan action-action yang tersedia pada Enums PlayerAction. Dari hasil yang didapatkan, dapat dilihat bahwa algoritma *greedy* pada bot cukup optimal.

5.2. Saran

Ada beberapa perbaikan yang bisa kita lakukan untuk selanjutnya :

- Sebaiknya melakukan pembahasan spesifikasi secepatnya dan segera memahami cara kerja program keseluruhan
- Pembuatan laporan dapat dilakukan lebih cepat dan tidak mendekati deadline
- Melakukan pembagian lebih cepat agar pekerjaan lebih terstruktur

DAFTAR PUSTAKA

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