Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2022/2023

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "*Galaxio*"

Kelompok 13 - EnteGolKiperPalingBaikDiDunia

Anggota Kelompok:

Saddam Annais Shaquille	13521121
M. Dimas Sakti Widyatmaja	13521160
Mohammad Rifqi Farhansyah	13521166

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Daftar Tabel dan Gambar

BAB I
Deskripsi Masalah

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

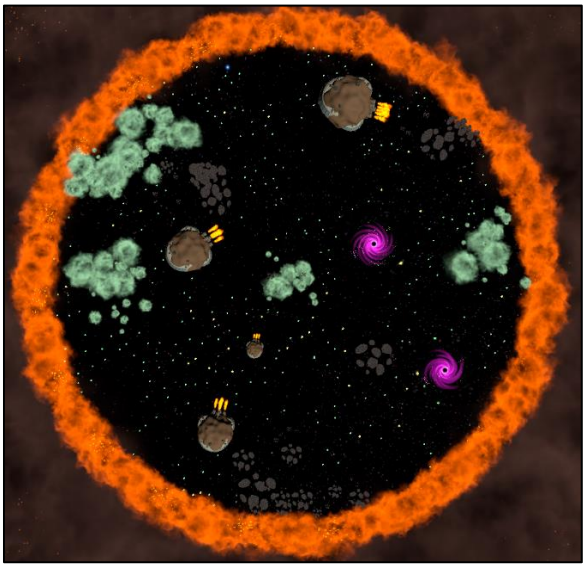
Gambar 1.1. *Entelect Challenge 2021 : Galaxio*
Sumber: https://forum.entelect.co.za/uploads/db1882/optimized/2X/6/6cea15c561d0e2a2e45378a523d96b0aa69a0fe6_2_1024x576.jpeg

Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini. Bahasa pemrograman yang digunakan pada tugas besar ini adalah Java. Bahasa Java tersebut digunkan untuk membuat algoritma pada *bot*. IDE yang digunkan untuk membantu projek ini adalah Intelllij IDEA. Intellij IDEA emrupakan IDE yang kompatibel dengan bahasa Java, karena beberapa tools yang diperlukan dalam tugas besar ini, seperti Maven sudah built in tanpa perlu manambahkan extension. Untuk menjalankan permainan, digunakan sebuah game-engine yang diciptakan oleh Entellect Challenge yang terdapat pada repository github berikut ini. Game-engine yang dibuat oleh Entellect Challenge menggunakan bahasa C# sebagai bahasa pemrograman utama dalam pembuatnnya. 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.



Gambar 1.2. *Gambaran Permainan Galaxio*
Sumber: Dokumen Spesifikasi Tugas Besar 1 IF2211 Strategi Algoritma 2023

4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari player untuk memasukinya dan keluar di pasangan satu lagi. Wormhole akan bertambah besar setiap tick game hingga ukuran maximum. Ketika Wormhole dilewati, maka wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat wormhole lebih besar dari kapal player.
5. Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.
6. Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo akan mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.
7. Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud.
8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran

maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

10. Terdapat beberapa command yang dapat dilakukan oleh player. Setiap tick, player hanya dapat memberikan satu command. Untuk daftar commands yang tersedia, bisa merujuk ke tautan panduan di spesifikasi tugas
11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal player lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi.

Adapun daftar peraturan yang lebih lengkap dari permainan Galaxio, dapat dilihat pada laman berikut ini.

BAB II
Teori Singkat

2.1. Gambaran 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/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 menngunakan 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 dioptimalisasi 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*
Sumber: Dokumen Ajar Algoritma Greedy oleh Rinaldi Munir

Pada akhir tiap iterasi, solusi yang terbentuk adalah optimum lokal, dan pada 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 ditemukan untuk masalah yang hanya membutuhkan solusi hampiran dan tidak memerlukan solusi terbaik mutlak. Solusi ini terkadang lebih baik daripada algoritma yang menghasilkan solusi eksak denga 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 diawali dengan proses pengambilan seluruh data yang tersedia pada *files Galaxio*. Data-data yang terdapat pada *files Galaxio* adalah sebagai berikut:

- *objects* : Seluruh komponen dalam permainan yang direpresentasikan dengan bentuk lingkaran dan memiliki titik pusat dengan X dan Y koordinat serta radius yang mendefinisikan ukuran dan bentuknya.
    - *food* : Makanan yang akan meningkatkan ukuran kapal pemain. Makanan ini nantinya akan tersebar dalam *map*, tidak bergerak, serta akan dihapus jika keluar dari *map*.
    - *superFood* : Makanan yang dapat meningkatkan ukuran kapal pemain dua kali lipat dibanding dengan makanan biasa.
    - *wormholes* : Lubang yang dapat membuat pemain bisa berpindah dari satu posisi ke posisi lainnya.

    - *gasClouds* : Objek yang membuat size kapal berkurang satu tiap *tick* dalam permainan jika suatu kapal berada di dalamnya.
    - *asteroidFields* : Objek yang membuat kecepatan kapal berkurang dengan faktor dua apabila sebuah kapal melewatinya.
    - *torpedoSalvo* : Senjata yang dapat ditembakan ke musuh lainnya dengan efek tertentu.
    - *supernova* : Senjata terkuat yang hanya akan muncul di waktu tertentu antara kuarter pertama dan kuarter terakhir permainan.
    - *teleport* : Objek yang dapat digunakan pemain untuk berpindah posisi dengan cara menembakannya.
    - *id* : Kode unik tertentu yang tersemat pada tiap *game object*.
- *player* : Pemain akan berupa sebuah kapal berbentuk bulat yang akan bertumbuh seiring dengan proses memakan *food* dalam *gameObjects*.
    - *speed* : Aspek yang menentukan seberapa jauhnya sebuah kapal pemain akan bergerak dalam satu *tick*.
    - *size* : Ukuran/radius dari suatu objek.
    - *heading* : Ukuran derajat radian suatu muka objek tertentu.
    - *afterBurner* : Item yang akan meningkatkan kecepatan pemain dengan faktor dua.
    - *darkMatterTorpedos* : Senjata yang dapat digunakan kapal pemain untuk merekayasa area di sekitarnya.
    - *shield* : Item pertahanan yang dapat digunakan untuk menepis serangan pemain lain.
    - *X position* : Posisi pada grafik kartesius permainan berdasarkan sumbu-X.
    - *Y position* : Posisi pada grafik kartesius permainan berdasarkan sumbu-Y.
- *world :* Kumpulan dari beberapa komponen tertentu yang terkait dengan perubahan faktor-faktor esensial pada permainan.
    - *centerPoint* : Titik tengah suatu objek pada *map*.
    - *tick* : Ukuran waktu yang berlaku pada permainan *Galaxio*.
- *command* : Pada tiap *tick* setiap pemain dapat melakukan satu buah perintah/*command* untuk kapal mereka.
    - *FORWARD* : Perintah ini akan memerintahkan kapal bergerak sesuai arahnya pada saat tersebut.
    - *STOP* : Perintah ini akan memerintahkan kapal berhenti hingga ada perintah lain yang dikeluarkan.
    - *START_AFTERBURNER* : Perintah untuk menyalakan *afterBurner*.
    - *STOP_AFTERBURNER* : Perintah untuk mematikan *afterBurner*.
    - *FIRE_TORPEDOS* : Perintah untuk menembakkan *torpedo*.
    - *FIRE_SUPERNOVA* : Perintah untuk menembakkan *supernova*.
    - *DETONATE_SUPERNOVA* : Perintah untuk meledakkan *supernova*.
    - *FIRE_TELEPORTER* : Perintah untuk menembakkan *teleporter*.
    - *TELEPORT* : Perintah untuk berpindah pada posisi *teleporter* saat itu.
    - *USE_SHIELD* : Perintah untuk mengaktifkan *shield*.

Setelah *bot* mengambil data-data di atas, maka *bot* dapat melakukan analisis menggunakan algoritma yang telah dituliskan sebelumnya untuk mencari *command* paling tepat untuk dieksekusi pada tick selanjutnya. Algoritma yang dituliskannya tersebut berbentuk fungsi objektif dari algoritma *greedy*. Perlu ditekankan pula, untuk setiap keadaan pada data, *bot* harus mengirimkan *command* yang bersesuaian.

2.3. Implementasi Algoritma *Greedy* ke dalam *Bot* Permainan *Galaxio*

Terdapat beberapa algoritma yang dapat digunakan sebagai algoritma *bot*. Beberapa algoritma yang dapat digunakan adalah diantaranya algoritma *brute force*, algoritma *greedy*, algoritma *tree traversal*, dan lain-lain. Tetapi, semua algoritma tersebut memiliki kelebihan dan kekurangannya masing-masing. Pada algoritma *brute force*, dibutuhkan waktu dan *resource* yang banyak untuk mengetes dan menilai setiap kemungkinan *command* yang dapat terjadi, meskipun hasil paling optimum yang ditemukan merupakan optimum global pada keadaan tersebut. Kemudian, pada algoritma *greedy*, tidak diperlukan waktu atau *resource* yang banyak. Tetapi, dibutuhkan teknik heuristik yang didefinisikan oleh logika dan pola pikir masing-masing pemrogram. Teknik heuristik tersebut mungkin belum dapat menghasilkan solusi optimum global dan walaupun solusi tersebut merupakan solusi optimum global, solusi tersebut susah dibuktikan kebenarannya secara matematis. Algoritma berikutnya yang dapat digunakan adalah algoritma *tree traversal*. Algoritma ini umum digunakan pada *bot game*, dikarenakan pada algoritma ini terdapat pengecekkan beberapa kasus tertentu untuk menentukan langkah selanjutnya yang diambil. Tetapi, algoritma ini tidak seintuitif algoritma *brute force* maupun algoritma *greedy* bagi 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.

2.4. Garis Besar *GameEngine* Permaian *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 seperti *rendering engine for command prompt*, *collisions engine*, dan *scripting language* saja yang terdapat pada *repository*. Untuk beberapa tools tambahan, seperti 2D *rendering engine* dapat diunduh dari *third party software*.

Pada permainan *Galaxio* ini, *game engine* yang digunakan dibuat menggunakan bahasa C# dan sudah tersedia sebelumnya pada *repository* yang diunggah sebelumnya oleh *Entelect* (*Entelect* merupakan penyelenggara dari *challenge* ini). *Game engine* yang diberikan pada *starter-pack* sudah berbentuk file .jar, jadi kita tidak perlu melakukan *compile* dan *build* ulang projek secara manual. *Prerequisites* untuk menjalankan *game engine* adalah memiliki JDK (*Java Development Kit*) pada mesin eksekusi untuk menjalankan *game engine*. Hal ini ditujukan untuk memroses file .jar, melakukan kompilasi file dengan bahasa Java, serta mengeksekusinya. Jika *prerequisites* telah dipenuhi, untuk menjalankan *game engine* pengguna dapat menjalankan file run.bat untuk pengguna dengan OS Windows atau menjalankan perintah shell script pada file Makefile dengan mengetikkan perintah "*make run*" pada *command prompt* untuk pengguna dengan OS Linux atau macOS.

Selain file *game engine*, terdapat beberapa *file* dan *folder* yang perlu ditambahkan pada *starter-pack* oleh pemrogram. Beberapa file dan folder yang wajib ada agar *game engine* dapat dijalankan adalah diantaranya:

- *game-engine.jar* dan *game-engine-jar-with-dependencies.jar*: Berisi *source code* serta *dependecies* dari *game engine*.
- *game-config.json*: Berisi *rules* yang diikuti oleh *game engine* ketika mengeksekusi permainan.
- *game-runner-config.json*: Berisi informasi mengenai *directory* dari *file bot* serta parameter-parameter lainnya yang dibutuhkan oleh *game engine*.
- *Run.sh* dan *run.bat*: Berisi *file* dengan *scripting language* yang digunakan untuk menjalankan *game engine* pada OS. Run.sh digunakan untuk OS Linux dan macOS serta run.bat digunakan untuk OS Windows.

Setelah algoritma *greedy* para *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 III
Aplikasi Strategi *Greedy*

3.1 Pemetaan Elemen/Komponen Algoritma *Greedy* pada *Bot* Permianan *Galaxio*

3.1.1. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Attack Phase*

Salah satu subpermasalahan dalam permainan *Galaxio* adalah adanya beberapa objek yang dapat digunakan oleh kapal pemain untuk saling mengurangi *size* satu sama lain. Kapal pemain dapat menggunakan *torpedo, supernova*, serta *teleport* untuk mengurangi *size* musuh dan bahkan langsung memakannya.

Tabel 3.1.1.1 Komponen Algoritma Greedy pada Permasalahan Attack Phase

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Himpunan *bot* musuh yang berada pada lapangan.
Himpunan solusi	*Command* pada *bot* sendiri untuk menyerang musuh yang terpilih untuk diserang.
Fungsi solusi	Memeriksa strategi menyerang terbaik yang dapat digunakan pada *tick* tertentu seperti menembakkan *torpedo*, *supernova*, atau melakukan *teleport* dan menentukan musuh teroptimal untuk diserang.
Fungsi seleksi	Memilih musuh yang terkecil dan jarak dari terdekat.
Fungsi kelayakan	Memeriksa apakah tidak ada objek-objek yang berbahaya berdasarkan fungsi radar yang mendekat dan musuh yang dipilih lebih kecil ukurannya dibanding bot sendiri.
Fungsi objektif	Mencari strategi paling tepat berdasarkan musuh yang telah dipilih

3.1.2. Pemetaan Elemen/Komponen Algoritma Greedy pada Permasalahan *Defense Phase*

Subpermasalahan mendasar lainnya dalam permainan *Galaxio* adalah melakukan pertahanan dari upaya serangan oleh objek pemain lainnya. Kapal pemain dapat menggunakan *after burner*, *shields*, *wormhole,* hingga *asteroid* untuk berlindung dari serangan kapal pemain lain.

Tabel 3.1.2.1 Komponen Algoritma Greedy pada Permasalahan Defense Phase

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Objek-objek membahayakan sepeti kapal pemain lain, teleporter pemain lain, batas dunia, torpedo, dan *gas cloud*.
Himpunan solusi	*Command* pada *bot* untuk melakukan penangan berdasarkan ancaman yang sedang ada dalam *tick* tertentu.
Fungsi solusi	Memeriksa strategi bertahan terbaik yang dapat digunakan pada *tick* tertentu seperti menghidupkan *afterburner*, menyalakan *shield,* atau melakukan *teleport* untuk lari dan menentukan arah teroptimal untuk melarikan diri dari ancaman.
Fungsi seleksi	Memilih objek paling mengancam pada *tick tertentu.*
Fungsi kelayakan	Memeriksa apakah tidak ada objek-objek yang mendekat berdasarkan fungsi radar yang mendekat dan menentukan apakah objek tersebut dikategorikan membahayakan.
Fungsi objektif	Mencari strategi paling tepat berdasarkan objek ancaman dan efek-efek yang dapat *bot* sendiri gunakaan.

3.1.3. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Permasalahan *Default Phase*

Subpermasalahan mendasar yang terakhir adalah mekanisme *default* apabila tidak ada kondisi menyerang atau bertahan yang terpenuhi. Pada awal permainan, tentunya setiap kapal masih belum memenuhi syarat untuk melakukan mekanisme penyerangan sehingga mereka otomatis tidak memerlukan pula mekanisme pertahanan terhadap serangan kapal pemain lain. Oleh karena itu, diperlukan sebuah mekanisme *default* yang akan dijalankan ketika sebuah kapal tidak memenuhi kedua fase itu. Dalam permainan *Galaxio*, tujuan setiap *bot* pemain berusaha untuk menjadi yang terakhir bertahan pada tiap ronde atau menjadi kapal dengan *size* terbesar saat permainan berakhir. Terdapat banyak cara untuk meraih hal tersebut, seperti berusaha mencari objek *food* sebanyak-banyaknya, melakukan penyerangan pada kapal musuh untuk mengurangi *size*-nya, atau bahkan hanya berdiam pada *asteroid* hingga seluruh kapal lain saling berkompetisi satu sama lain.

Tabel 3.1.3.1 Komponen Algoritma Greedy pada Permasalahan Default Phase

Nama Elemen/Komponen	Definisi Elemen/Komponen
Himpunan kandidat	Himpunan *food* atau *superfood* yang berada pada permainan.
Himpunan solusi	*Food* atau *superfood* yang terpilih.
Fungsi solusi	Memeriksa apakah *food* atau superfood yang dipilih sudah merupakan yang terdekat.
Fungsi seleksi	Memilih *food* atau *superfood* terdekat dengan cara melakukan *sorting* pada objek *food* atau *superfood* yang berada pada permainan.
Fungsi kelayakan	Meemeriksa apakah himpunan *food* atau *superfood* tidak kosong.
Fungsi objektif	*Food* atau *superfood* yang dipilih terdekat.

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. Sebagai contoh, apabila suatu objek kapal mencoba untuk meningkatkan *size*-nya maka secara otomatis *speed* dari kapal tersebut akan berkurang. Oleh karena itu, strategi algoritma *greedy* yang disusun oleh penulis selalu memiliki interaksi dengan strategi algoritma *greedy* pada bidang yang berbeda. Berikut ini merupakan beberapa alternatif strategi algoritma *greedy* yang dapat diimplementasikan pada *bot* pemain:

3.2.1 Strategi *Attack Phase*

Dalam fase menyerang, hal pertama yang harus dilakukan adalah mencari kapal musuh yang tepat untuk diserang. Dalam memilih kapal musuh yang tepat untuk diserang, terdapat beberapa variabel yang dapat dijadikan untuk pertimbangan antara lain seperti ukuran kapal musuh, kecepatan musuh, jarak ke kapal musuh, serta keadaan di lapangan. Setelah memilih kapal musuh untuk diserang, kami perlu memikirkan metode penyerangan ke kapal musuh. Terdapat tiga opsi untuk menyerang kapal musuh, yaitu mendekati musuh, menembak dengan torpedo, atau melakukan teleportasi ke koordinat dekat kapal musuh. Ada dua varian dalam mendekati musuh, yaitu dengan atau tanpa menggunakan afterburner, yang masing-masing memiliki peran sendiri-sendiri. Menggunakan afterburner dapat meningkatkan kecepatan kapal hingga dua kali lipat, memudahkan pengejaran musuh, namun harus dilakukan dengan hati-hati karena mengurangi ukuran kapal. Tanpa afterburner, kecepatan kapal hanya

sejalan dengan ukurannya sendiri. Selain itu, torpedo juga dapat digunakan untuk menyerang musuh. Setiap salvo charge dari torpedo dapat menembakkan sepuluh torpedo dan menambah ukuran kita sebesar 10. Terakhir, opsi penyerangan adalah dengan menggunakan teleporter ketika ukuran kapal cukup besar dibandingkan dengan musuh yang ditargetkan. Tujuan utama penggunaan teleporter adalah untuk langsung menyerang musuh dengan efektif, namun memerlukan presisi dan ketepatan dalam penggunaan teleporter.

3.2.2 Strategi *Defense Phase*

Pada strategi *Defense Phase*, tim kami perlu memantau semua objek yang hadir di lapangan. Kemudian, setiap objek akan diklasifikasikan menjadi dua kategori, yaitu objek yang berbahaya yang harus dihindari atau objek yang tidak berbahaya. Untuk mencegah objek berbahaya yang mendekati kapal, kami menggunakan checkRadar. Setelah mendeteksi objek berbahaya, ada dua cara untuk menghindarinya, yaitu dengan melarikan diri menggunakan teleporter atau tanpa teleporter. Jika kami tidak memiliki teleporter, kami dapat memilih untuk lari menggunakan afterburner jika kapal kami besar atau tanpa afterburner jika ukuran kapal kecil. Arah lari juga menjadi pertimbangan yang penting dalam menghindari musuh, oleh karena itu kami memilih *heading* sebesar 45 atau 315. Selain itu, kami juga dapat menembaki musuh dengan torpedo sambil menghindari serangan musuh. Metode ini efektif untuk mengurangi ukuran musuh. Jika ingin melarikan diri menggunakan teleporter, kapal kami harus memiliki teleporter dan ukuran kapal yang cukup besar untuk menghindari pengurangan ukuran kapal. Meski begitu, karena keterbatasan waktu, kami hanya menggunakan teleporter untuk menyerang dan lari dengan afterburner. Kami juga memiliki mekanisme pertahanan dengan mengaktifkan shield ketika diserang dengan torpedo, namun ini hanya berlaku jika ukuran kapal kami cukup besar dan shield dapat diaktifkan. Shield akan sangat efektif melindungi kapal ketika diserang dengan torpedo.

3.2.3 Strategi *Default Phase*

Strategi yang terakhir yaitu mengenai *Default Phase*. Strategi ini membahas tindakan yang harus dilakukan kapal kami dalam situasi menyerang dan bertahan yang tidak memadai. Strategi ini menjadi sangat penting pada awal permainan ketika masih banyak makanan di sekitar dan pada akhir permainan ketika jumlah objek di lapangan berkurang. Tujuan strategi ini adalah untuk membuat kapal mendekati makanan atau superfood terdekat tanpa memandang jenisnya. Kapal kami akan menambah ukurannya sebesar makanan yang dimakan jika memakan food, sedangkan jika kapal kami memakan superfood, selain menambah ukuran sebesar makanan yang dimakan, kapal kami akan mendapatkan efek tambahan yaitu makanan berikutnya akan memberikan tambahan ukuran dua kali lipat dari ukuran makanan normal.

3.3 Analisis Efisiensi dari Kumpulan Solusi Algoritma *Greedy*

Permainan Galaxio ini memiliki banyak variabel yang dapat mempengaruhi jalannya permainan, Setiap kapal dapat mengetahui *game state* sertaa *game object* pada *tick* tertentu. Hal ini memungkinkan tiap kapal untuk mengetahui variabel-variabel penting seperti ukuran kapal musuh, kecepatan musuh, benda-benda yang terdapat dalam lapangan, dan lain sebagainya. Tiap-tiap strategi algoritma Greedy yang kami implementasikan memiliki kompleksitas waktunya masing-masing. Setiap *tick*, kami mengumpulkan daftar kapal musuh yang dapat diserang dalam variabel preyList. Dalam pengumupulannya, dilakukan *sorting* sebanyak jumlah musuh yang ukurannya lebih kecil dari kapal kami. *Sorting* tersebut memerlukan kompleksitas sebesar O(n log n).

Pada strategi *Attack Phase*, terdapat perintah untuk melakukan pelancaran torpedo. Perintah ini hanya memerlukan kompleksitas waktu sebesar O(1). Selain menembakkan torpedo, kapal kami juga dapat

menyalakan *afterburner* untuk mengejar musuh, perintah ini juga hanya memerlukan kompleksitas waktu sebesar O(1).

Untuk strategi *Defense Phase,* kami memanfaatkan fungsi checkRadar untuk melacak objek-objek yang berpotensi untuk membahayakan kapal kami seperti *bot* musuh, torpedo, teleporter, dan lain sebagainya. Kompleksitas waktu dari fungsi checkRadar tergantung pada kompleksitas waktu dari fungsi getCloseGameObjects dan fungsi checkObjectsNextVersion. Kompleksitas waktu dari fungsi getCloseGameObjects adalah O(n), di mana n adalah jumlah objek game dalam status game, karena ia melakukan iterasi melalui setiap objek di status game. Kompleksitas waktu dari fungsi checkObjectsNextVersion juga O(n), di mana n adalah jumlah objek dalam daftar closeObjects, karena ia melakukan iterasi melalui setiap objek dalam daftar closeObjects. Fungsi checkRadar memanggil fungsi getCloseGameObjects empat kali dan fungsi checkObjectsNextVersion empat kali, yang berarti bahwa kompleksitas waktu checkRadar adalah O(4n) = O(n), di mana n adalah jumlah total objek game dalam status game. Oleh karena itu, kompleksitas waktu dari fungsi checkRadar adalah O(n).

Strategi yang terakhir adalah strategi *Default Phase*. Strategi ini baru dijalankan ketika tidak ada objek-objek yang membahayakan serta tidak terdapat kapal musuh yang bisa diserang. Strategi ini bertumpu pada fungsi catchFood. Fungsi ini mengumpulkan seluruh objek *food* dan *superfood* yang terdapat pada lapangan dan mengurutkannya berdasarkan jarak terpendek dengan *bot* kami. Karena fungsi ini memerlukan proses pengurutan, maka fungsi ini memiliki kompleksitas sebesar O(n log n).

3.4 Analisis Efektivitas dari Kumpulan Solusi Algoritma *Greedy*

Pada program *bot* yang telah kami buat, terdapat pilihan-pilihan yang harus kami pilih agar mendapatkan hasil yang optimal dan efektif dalam rangka memenangkan permainan. Dalam strategi 3 fase yang telah kami rancang,variabel-variabel yang dapat kami pertimbangkan dalam memilih aksi pada tiap *tick* antara lain kapal musuh yang diserang, objek yang berada di sekitar, dan efek yang diberikan aksi yang akan kami pilih.

Pada strategi *Attack Phase,* kami memiliki 3 pilihan, mendekati musuh, menembak dengan torpedo, atau melakukan *teleport* ke koordinat dekat kapal musuh. Mendekati musuh memiliki dua variasi, yaitu dengan atau tanpa menggunakan *afterburner*. Dua pilihan ini masing-masing memiliki perannya masing-masing. Mengejar dengan *afterburner* meningkatkan kecepatan kapal menjadi dua kali lipat, sehingga memudahkan pengejaran kapal musuh, namun hal ini juga mengonsumsi ukuran kapal kami, sehingga penggunaannya perlu berhati-hati. Di lain sisi, mengejar tanpa *afterburner* dapat dilakukan tanpa mengonsumsi ukuran kapal, namun kecepatannya sesuai dengan kecepatan *default* kapal sesuai ukurannya. Cara lain untuk menyerang yaitu dengan menembakkan torpedo, Cara ini memerlukan adanya torpedo salvo charge dalam penggunaannya. Satu torpedo salvo charge dapat digunakan untuk menembakkan sepuluh torpedo. Satu torpedo dapat mengurangi ukuran musuh sebanyak 10 dan menambah ukuran kita sebanyak 10 juga. Torpedo juga memiliki kecepatan sebesar 60 sehingga torpedo merupakan salah satu serangan yang cukup efektif dalam mengalahkan kapal musuh. Akan tetapi, torpedo dapat berkurang ukuran dan dampaknya jika bertabrakan dengan objek lain dan menggunakan salvo charge akan mengurangi ukuran kapal sebanyak 20. Metode penyerangan yang terakhir adalah menyerang menggunakan teleporter. Metode ini akan kami jalankan ketika kami memiliki teleporter dan ukuran kapal kami cukup lebih besar dari musuh yang kami incar. Jadi tujuan utama kami menggunakan teleporter adalah untuk melahap kapal musuh secara langsung sehingga kapal musuh langsung lenyap. Cara menyerang ini sangat efektif namun memerlukan presisi dan ketepatan dalam menembakkan dan mengaktifkan teleporter agar dapat mengenai musuh secara langsung.

Pada strategi *Defense Phase*, kami dapat menghindari supernova, *teleporter*, torpedo, *gas cloud*, batas dunia, dan kapal pemain lain. Untuk mengantisipasi objek-objek berbahaya yang mendekati kapal kami, kami memanfaatkan fungsi checkRadar. Setelah melacak benda-benda berbahaya yang mendekat ke kapal, kami dapat memilih dua cara untuk lari, yaitu lari dengan *afterburner* dan lari tanpa *afterburner*. Lari dengan memanfaatkan *afterburner* akan dilaksanakan bila kecepatan *default* kapal kami kecil dan ukuran kapal besar, atau tanpa afterburner jika ukuran kapal kami cukup kecil. Selain lari, kami juga dapat menembaki musuh dengan torpedo jika memungkinkan sambil menghindari musuh. Strategi tersebut akan cukup efektif digunakaan untuk menghindari musuh sekaligus mengurangi ukuran kapal musuh. Selain melarikan diri biasa, kami juga memiliki mekanisme pertahanan dengan mengaktifkan *shield* ketika diserang dengan torpedo. *Shield* hanya akan aktif bila ukuran kapal cukup besar dan *shield* dapat diaktifkan. *Shield* akan menjadi sangat efektif ketika terdapat musuh yang menyerang dengan torpedo dan ukuran kapal kami sudah cukup besar. Walaupun mengurangi ukuran kapal sebesar 20, hal ini akan jauh lebih baik dibanding risiko ditembaki dengan 10 torpedo berukuran 10.

Strategi yang terakhir yaitu mengenai *Default Phase*. Strategi ini tidak memerlukan banyak konsiderasi mengingat strategi ini hanya akan dijalankan ketika kondisi menyerang dan bertahan tidak dipenuhi. Strategi ini akan membuat kapal untuk mendekat ke *food* atau *superfood* terdekat tanpa membedakannya. Jika kapal memakan *food* maka akan menambah ukuran kapal sebanyak ukuran *food* tersebut, sedangkan apabila kapal memakan *superfood* maka akan mendapat efek yang sama seperti memakan *food* namun akan mendapat tambahan efek berupa makanan yang akan dimakan selanjutnya akan memiliki efek penambahan ukuran sebesar 2 kali lipat dari keadaan biasa. Jadi strategi ini akan cukup efektif untuk meningkatkan ukuran kapal ketika tidak ada objek yang berbahaya dan tidak ada musuh yang dapat diserang, terutama saat permainan baru dimulai.

3.5 Strategi *Greedy* yang Digunakan pada Program *Bot*

Strategi heuristik yang penulis ambil sebagai algoritma *greedy* utama dari program *bot* permainan *Galaxio* ini adalah penggabungan dari seluruh strategi yang telah dipaparkan pada subbab 3.2. Penulis mengambil seluruh strategi untuk setiap mekanisme pada permainan *Galaxio* agar program *bot* dapat menangangi seluruh kemungkinan kasus dari *game state* secara efektif dan tepat sasaran. 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. Urutan prioritas tersebut memiliki sifat heuristik pula, dimana pemrogram harus menentukan bagaimana urutannya agar menghasilkan algoritma *greedy* yang paling optimum. Pemrogram dapat menggunakan logika atau mengecek urutan strategi untuk memformulasikan urutan mana saja yang dapat diimplementasikan sebagai algoritma *greedy*.

BAB IV
Implementasi dan Pengujian

4.1 Implementasi Algoritma Greedy pada Bot Permainan Galaxio

Implementasi algoritma greedy pada program kami terdapat pada tiga file, yaitu pada BotService.java, GreedyCommand.java, dan Radar.java. Berikut merupakan implementasi kode untuk algoritma greedy kami pada masing-masing file.

- BotService.java

```
package Services;

import Enums.*;
import Models.*;

import java.util.*;
import java.util.stream.*;

public class BotService {
    private GameObject bot;
    public PlayerAction playerAction;
    private GameState gameState;
    public Integer tick = null;
    public GameObject launchedTeleport = null;
    public boolean justLaunchedTeleportStatus = false;
    public boolean switchDirection = false;

    public BotService() {
        this.playerAction = new PlayerAction();
        this.gameState = new GameState();
    }

    public GameObject getBot() {
        return this.bot;
    }

    public void setBot(GameObject bot) {
        this.bot = bot;
    }

    public PlayerAction getPlayerAction() {
        return this.playerAction;
    }

    public void setPlayerAction(PlayerAction playerAction) {
        this.playerAction = playerAction;
    }
```

```
    public void teleportSerang(PlayerAction playerAction, GameObject enemy) {
        // Punya return value yang menandakan heading teleporter dan bahwa telah
        // menembakkan teleporter
        playerAction.action = PlayerActions.FIRETELEPORT;
        // Kalau belum pakai radar
        playerAction.heading = getHeadingBetween(bot, enemy);
        System.out.println("Attack with launch teleport " + "  bot size : " +
bot.getSize()
                + "    enemy size: " + enemy.getSize() + "  distance : "
                + (getDistanceBetween(bot, enemy) - enemy.getSize() -
bot.getSize()));
        // return GameObject;
    }

    public GameObject checkTeleportSerang(PlayerAction p, GameObject bot,
GameObject launchedTeleport,
            List<GameObject> enemyList, List<GameObject> teleportList) {

        boolean teleportStatus = false;
        if (launchedTeleport != null) {
            System.out.println("test 1");
            for (int i = 0; i < teleportList.size(); i++) {
                System.out.println(teleportList.get(i).getId() + "   " +
launchedTeleport.getId());
                if (teleportList.get(i).getId().equals(launchedTeleport.getId()))
{
                    teleportStatus = true;
                    launchedTeleport = teleportList.get(i);
                }
            }
            GameObject updateLaunceTeleport = launchedTeleport;
            System.out.println(updateLaunceTeleport);
            // System.out.println(updateLaunceTeleport);
            if (teleportStatus) {
                System.out.println("test 2");
                List<GameObject> closeEnemy = enemyList.stream()
                        .filter(item -> getDistanceBetween(item,
updateLaunceTeleport) - bot.getSize()
                                - item.getSize() <= 5
                                && item.getId() != bot.getId() && item.getSize()
< bot.getSize())
                        .sorted(Comparator.comparing(
                                item -> getDistanceBetween(item,
updateLaunceTeleport) - bot.getSize()
                                        - item.getSize()))
```

```
                        .collect(Collectors.toList());

                if (closeEnemy.size() > 0) {
                    System.out.println("detonate attack teleport  size " +
bot.getSize() + "   enemy size :"
                            + closeEnemy.get(0).getSize() + "    distance: "
                            + (getDistanceBetween(updateLaunceTeleport,
closeEnemy.get(0)) - bot.getSize()
                                    - closeEnemy.get(0).getSize()));
                    p.action = PlayerActions.TELEPORT;
                    p.heading = getHeadingBetween(bot, closeEnemy.get(0));
                    return null;
                } else {
                    System.out.println("can't detonate, too far away" );
                    return updateLaunceTeleport;
                }
            }
        }
        return null;
    }

    public void computeNextPlayerAction(PlayerAction playerAction) {
        var objectState = gameState.getGameObjects();
        var playerState = gameState.getPlayerGameObjects().stream().filter(item -
> !item.getShieldStatus())
                .sorted(Comparator.comparing(item -> getDistanceBetween(bot,
item)))
                .collect(Collectors.toList());

        var teleportState = gameState.getGameObjects().stream()
                .filter(item -> item.getGameObjectType() ==
ObjectTypes.TELEPORTER)
                .sorted(Comparator.comparing(item -> getDistanceBetween(bot,
item))).collect(Collectors.toList());

        List<GameObject> preyList = playerState.stream()
                .filter(item -> item.getSize() < bot.getSize())
                .sorted(Comparator.comparing(item -> getDistanceBetween(bot,
item)))
                .collect(Collectors.toList());

        GameObject torpedoTarget = null;
        if (playerState.size() > 1) {
            torpedoTarget = playerState.get(1);
        }
```

```
        GameObject closestEnemy = null;
        if (preyList.size() > 0) {
            closestEnemy = preyList.get(0);
        }

        var objectRadar = new Radar();

        if (!objectState.isEmpty() && !playerState.isEmpty()) {
            if (gameState.getWorld().getCurrentTick() != tick) {
                System.out.println("tick: " +
gameState.getWorld().getCurrentTick());
                System.out.println(launchedTeleport);
                var warning = objectRadar.checkRadar(gameState, bot);
                // System.out.println(warning);
                if (warning != null) {
                    if (warning.gameObjectType == ObjectTypes.SUPERNOVA_BOMB) {
                        // Jika muncul bahaya supernova bomb
                        playerAction = GreedyCommand.run(warning, playerAction,
bot, switchDirection,
                                -90);

                    } else if (warning.gameObjectType == ObjectTypes.TELEPORTER)
{
                        // Jika muncul bahaya teleporter
                        playerAction = GreedyCommand.run(warning, playerAction,
bot, switchDirection,
                                -90);

                    } else if (warning.gameObjectType ==
ObjectTypes.TORPEDO_SALVO){
                        // Jika muncul bahaya torpedo
                        if (!bot.getShieldStatus()) {
                            if (bot.getShieldCount() > 0 && bot.getSize() > 40
                                    && getDistanceBetween(bot, warning) -
bot.getSize() < 45) {
                                playerAction.action =
PlayerActions.ACTIVATESHIELD;
                            } else {
                                playerAction = GreedyCommand.run(warning,
playerAction, bot,
                                        switchDirection, -90);;
                            }
                        }

                    } else if (warning.gameObjectType == ObjectTypes.PLAYER) {
```

```
                        if (launchTorpedo(bot, warning, playerAction, false,
objectState)) {
                        } else {
                            playerAction = GreedyCommand.run(warning,
playerAction, bot, switchDirection,
                                    -60);
                        }

                    } else if (closestEnemy != null
                            ? getDistanceBetween(closestEnemy, bot) -
bot.getSize() - closestEnemy.getSize() < 20
                            : false) {
                        GreedyCommand.activateAfterburner(playerAction, bot,
closestEnemy, true, 0);
                    }

                } else if (!justLaunchedTeleportStatus && bot.getTeleCount() > 0
&& bot.getSize() > 45
                        && (closestEnemy != null
                                ? bot.getSize() - closestEnemy.getSize() > 30
                                : false)
                        && launchedTeleport == null
                        && getDistanceBetween(bot, closestEnemy) - bot.getSize()
- closestEnemy.getSize() > 50) {
                    teleportSerang(playerAction, closestEnemy);
                    justLaunchedTeleportStatus = true;

                } else if (launchTorpedo(bot, torpedoTarget, playerAction, true,
objectState)) {

                }

                // playerAction.action = PlayerActions.STOP;
                else {
                    System.out.println("aman");

                    if (bot.getAfterburnerStatus()) {
                        playerAction.action = PlayerActions.STOPAFTERBURNER;
                    } else {
                        GreedyCommand.catchFood(gameState.getGameObjects(), bot,
playerAction);
                    }
                }
```

```
                if (GreedyCommand.evadeObject(bot, objectState,
gameState.getWorld(), playerAction)) {
                    // objectTracker = GreedyCommand.evadeObject(bot,
objectState, gameState.getWorld(), playerAction,
                    //         objectTracker);
                    switchDirection = !switchDirection;
                }

                if (justLaunchedTeleportStatus && teleportState.size() > 0
                        && playerAction.action != PlayerActions.FIRETELEPORT) {
                    launchedTeleport = teleportState.get(0);
                    justLaunchedTeleportStatus = false;
                }
                launchedTeleport = checkTeleportSerang(playerAction, bot,
launchedTeleport, preyList, teleportState);
                System.out.println("launched teleport:  " + launchedTeleport +
"     just launched teleport: "
                        + justLaunchedTeleportStatus);
                // if (headingTele != -1) {
                // headingTele = checkTeleportSerang(playerAction, headingTele,
preyTracker,
                // playerState);
                // }

            }

            if (gameState.getWorld().getCurrentTick() != tick) {
                // prevState = gameState;
                tick = gameState.getWorld().getCurrentTick();
                System.out.println();
            }
            this.playerAction = playerAction;
        }
    }

    public GameState getGameState() {
        return this.gameState;
    }

    public void setGameState(GameState gameState) {
        this.gameState = gameState;
        updateSelfState();
    }

    private void updateSelfState() {
```

```
        Optional<GameObject> optionalBot =
gameState.getPlayerGameObjects().stream()
                .filter(gameObject -> gameObject.id.equals(bot.id)).findAny();
        optionalBot.ifPresent(bot -> this.bot = bot);
    }

    public static double getDistanceBetween(GameObject object1, GameObject
object2) {
        var triangleX = Math.abs(object1.getPosition().x -
object2.getPosition().x);
        var triangleY = Math.abs(object1.getPosition().y -
object2.getPosition().y);
        return Math.sqrt(triangleX * triangleX + triangleY * triangleY);
    }

    public static int getHeadingBetween(GameObject bot, GameObject otherObject) {
        var direction = toDegrees(Math.atan2(otherObject.getPosition().y -
bot.getPosition().y,
                otherObject.getPosition().x - bot.getPosition().x));
        return (direction + 360) % 360;
    }

    public static int getHeadingBetween(GameObject bot, int x, int y) {
        var direction = toDegrees(Math.atan2(y - bot.getPosition().y, x -
bot.getPosition().x));
        return (direction + 360) % 360;
    }

    public static int toDegrees(double v) {
        return (int) (v * (180 / Math.PI));
    }

    public boolean launchTorpedo(GameObject bot, GameObject enemy, PlayerAction
p,
            boolean attack, List<GameObject> ObjList) {
        if (enemy == null) {
            return false;
        }
        boolean condition1 = bot.getSize() > (attack ? 20 : 15);
        boolean condition2 = false;
        if (attack) {
            if (enemy.getSize() > 50) {
                condition2 = getDistanceBetween(bot, enemy) - bot.getSize() -
enemy.getSize() < 600;
            } else if (enemy.getSize() > 80) {
```

```
                condition2 = true;
            } else {
                condition2 = getDistanceBetween(bot, enemy) - bot.getSize() -
enemy.getSize() < 400
                        && enemy.getSize() > 20;
            }

        } else {
            condition2 = getDistanceBetween(bot, enemy) - bot.getSize() -
enemy.getSize() > 150;
        }

        boolean condition3 = bot.getTorpedoCount() > 0;

        int heading = getHeadingBetween(bot, enemy);
        List<GameObject> obstructObjList = ObjList.stream().filter(item ->
getHeadingBetween(bot, item)
                + toDegrees(Math.atan2(item.getSize(), getDistanceBetween(bot,
item))) > heading
                && getHeadingBetween(bot, item)
                        - toDegrees(Math.atan2(item.getSize(),
getDistanceBetween(bot, item))) < heading
                && getDistanceBetween(bot, item) < getDistanceBetween(bot, enemy)
&&
                (item.getGameObjectType() == ObjectTypes.ASTEROID_FIELD
                        || item.getGameObjectType() == ObjectTypes.GAS_CLOUD
                                && item.getGameObjectType() ==
ObjectTypes.WORMHOLE))
                .collect(Collectors.toList());
        boolean condition4 = obstructObjList.size() < 2;

        boolean condition = condition1 && condition2 && condition3 && condition4;

        System.out.println("heading clear  (condition4): " + condition4 +
"  obstruction count: " + obstructObjList.size());

        if (condition) {
            System.out
                    .println((attack ? "Attack with torpedo" : "Defend with
torpedo") + "   bot size : " + bot.getSize()
                            + "    enemy size: "
                            + enemy.getSize() + "  distance : "
                            + (getDistanceBetween(bot, enemy) - bot.getSize() -
enemy.getSize()) + "   heading:  "
                            + getHeadingBetween(bot, enemy));
```

```
            p.action = PlayerActions.FIRETORPEDOES;
            p.heading = getHeadingBetween(bot, enemy);
        }

        return condition;
    }
}
```

- GreedyCommand.java

```
package Services;

import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

import Enums.ObjectTypes;
import Enums.PlayerActions;
import Models.GameObject;
import Models.PlayerAction;
import Models.World;

public class GreedyCommand extends BotService {
  public static PlayerAction run(GameObject enemy, PlayerAction p, GameObject
bot,
      boolean switchDirection, int direction) {

    System.out.println("speed: " + bot.getSpeed() + "  size:  " + bot.getSize() +
"  status: "
        + bot.getAfterburnerStatus() + " effect:  " + bot.getEffects() +
"  distance: "
        + (BotService.getDistanceBetween(enemy, bot) - enemy.getSize() -
bot.getSize()) + "   enemy size  :"
        + enemy.getSize() + "  objectType:  " + enemy.getGameObjectType());

    if (enemy.getGameObjectType() != ObjectTypes.TORPEDO_SALVO) {
      activateAfterburner(p, bot, enemy, false, direction);
    }

    // if (objectTracker == null || !(objectTracker == null ? UUID.randomUUID() :
objectTracker).equals(enemy.getId())) {
    if (switchDirection) {
      p.heading = (getHeadingBetween(enemy, bot) + 45) % 360;

      // (enemy.currentHeading + direction) % 360;
    } else {
```

```
      p.heading = (getHeadingBetween(enemy, bot) - 45) % 360;
    }
    System.out.println("bot heading :" + bot.getCurrentHeading() + "   danger
heading: " + enemy.getCurrentHeading());
    // }
    return p;
  }

  public static void activateAfterburner(PlayerAction p, GameObject bot,
GameObject enemy, boolean attack, int heading) {
    if (attack) {
      if (bot.getAfterburnerStatus()) {
        if (bot.getSize() - enemy.getSize() < 20 || bot.getSize() < 30) {
          p.action = PlayerActions.STOPAFTERBURNER;
          p.heading = BotService.getHeadingBetween(bot, enemy);
        } else {
          p.action = PlayerActions.FORWARD;
          p.heading = BotService.getHeadingBetween(bot, enemy);
        }
      } else {
        if (bot.getSpeed() == 0) {
          p.action = PlayerActions.FORWARD;
          p.heading = BotService.getHeadingBetween(bot, enemy);
        } else if (bot.getSize() - enemy.getSize() > 20 && bot.getSize() > 40) {
          p.action = PlayerActions.STARTAFTERBURNER;
          p.heading = BotService.getHeadingBetween(bot, enemy);
        }
      }

    } else {
      // System.out.println(bot.getAfterburnerStatus());
      if (!bot.getAfterburnerStatus()) {
        if (bot.getSpeed() == 0) {
          // System.out.println("speed 0");
          p.action = PlayerActions.FORWARD;
        } else if (bot.getSize() > 30) {
          if (enemy.getGameObjectType() == ObjectTypes.PLAYER) {
            if (BotService.getDistanceBetween(bot, enemy) - bot.getSize() -
enemy.getSize() < 100
                && enemy.getSize() - bot.getSize() < 20) {
              p.action = PlayerActions.STARTAFTERBURNER;
            } else {
              p.action = PlayerActions.FORWARD;
            }
          } else {
```

```
            p.action = PlayerActions.STARTAFTERBURNER;
          }

        } else if (bot.getSize() < 20) {
          // System.out.println("kurang dari 20");
          p.action = PlayerActions.FORWARD;
        }
      } else {
        if (bot.getSize() < 20 && getDistanceBetween(bot, enemy) > 50) {
          p.action = PlayerActions.STOPAFTERBURNER;
        } else if (bot.getTorpedoCount() > 0 && bot.getSize() > 35 &&
enemy.getGameObjectType() == ObjectTypes.PLAYER) {
          System.out.println("fire torpedos while running");
          p.action = PlayerActions.FIRETORPEDOES;
          p.heading = getHeadingBetween(bot, enemy);
        } else {
          p.action = PlayerActions.FORWARD;
          p.heading = (getHeadingBetween(bot, enemy) + heading) % 360;
        }
      }
    }
  }

  public static boolean checkBorder(GameObject bot, World w, PlayerAction p) {
    var condition = Math.sqrt(Math.pow(bot.getPosition().getX(), 2) +
Math.pow(bot.getPosition().getY(), 2))
        + bot.getSize() + 40 > w.getRadius();
    if (condition) {
      System.out.println("Evade border   size: " + bot.getSize());
      p.heading = BotService.getHeadingBetween(bot, 0, 0);
      p.action = PlayerActions.FORWARD;
      return true;
    }
    return false;
  }

  public static boolean checkHarmfulObject(GameObject bot, List<GameObject>
gameObj, PlayerAction p,
      ObjectTypes objectType, int treshold, String text) {
    var ObjList = gameObj.stream()
        .filter(
            item -> item.getGameObjectType() == objectType)
        .sorted(Comparator.comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList());
    if (ObjList.size() > 0) {
```

```
      var condition = BotService.getDistanceBetween(bot, ObjList.get(0)) -
bot.getSize()
          - ObjList.get(0).getSize() < treshold;
      if (condition) {
        System.out.println(text + "  size: " + bot.getSize());
        p.heading = (BotService.getHeadingBetween(bot, ObjList.get(0)) + 180) %
360;
        p.action = PlayerActions.FORWARD;
        return true;
      }
    }
    return false;
  }

  public static boolean evadeObject(GameObject bot, List<GameObject> gameObj,
World w, PlayerAction p) {
    if (p.action == PlayerActions.FIRETORPEDOES || p.action ==
PlayerActions.FIRETELEPORT
        || p.action == PlayerActions.FIRESUPERNOVA) {
      return false;
    }
    // objectTracker = checkHarmfulObject(bot, gameObj, p,
    // ObjectTypes.ASTEROID_FIELD, 10, "evade asteroid field",
    // objectTracker);
    boolean status = false;
    status = checkHarmfulObject(bot, gameObj, p, ObjectTypes.GAS_CLOUD,40, "evade
gas cloud");
    // objectTracker = checkHarmfulObject(bot, gameObj, p, ObjectTypes.WORMHOLE,
20,
    // "evade wormhole", objectTracker);
    if (status) {
      return status;
    }
    status = checkBorder(bot, w, p);
    // switchDirection = !switchDirection;
    return status;
  }

  public static void catchFood(List<GameObject> allState, GameObject bot,
PlayerAction p) {
    System.out.println("Cathing food     Size : " + bot.getSize());
    var foodList = allState.stream()
        .filter(
            item -> ((bot.getCurrentHeading() - item.getCurrentHeading()) % 360 <
165 || (bot.getCurrentHeading() - item.getCurrentHeading()) % 360 > 195) &&
```

```
item.getGameObjectType() == ObjectTypes.FOOD || item.getGameObjectType() ==
ObjectTypes.SUPER_FOOD)
        .sorted(Comparator.comparing(item -> getDistanceBetween(bot, item)))
        .collect(Collectors.toList());

    if (foodList.size() > 0) {
      System.out.print("food distance1 : " +
BotService.getDistanceBetween(foodList.get(0), bot));
      if (foodList.size() > 1) {
        System.out.println("   distance2 : " +
BotService.getDistanceBetween(foodList.get(1), bot));
      }
      p.heading = getHeadingBetween(bot, foodList.get(0));
      p.action = PlayerActions.FORWARD;
    } else {
      p.action = PlayerActions.FORWARD;
      p.heading = getHeadingBetween(bot, 0, 0);
    }

  }
}
```

- Radar.java

```
package Services;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

import Enums.ObjectTypes;
import Models.GameObject;
import Models.GameState;

public class Radar {
  public boolean supernovaDefend;
  public boolean teleportDefend;
  public boolean torpedoDefend;
  public boolean playerDefend;

  public static final int ALL_CLEAR = 0;
  public static final int TELEPORTDEFENCE = 1;
  public static final int SUPERNOVADEFENCE = 2;
  public static final int TORPEDODEFENCE = 3;
```

```
  public Radar() {
    supernovaDefend = false;
    teleportDefend = false;
    torpedoDefend = false;
    playerDefend = false;
  }

  public List<GameObject> getCloseGameObjects(GameState gameState, GameObject
bot, ObjectTypes objectType,
      int distance) {
    // Mendapatkan semua gameObject dalam radius distance
    if (objectType == ObjectTypes.PLAYER) {
      var allPlayer = gameState.getPlayerGameObjects()
          .stream()
          .filter(item -> item.getGameObjectType() == objectType
              && (BotService.getDistanceBetween(bot, item) - bot.getSize() -
item.getSize() < distance))
          .sorted(Comparator.comparing(item -> BotService.getDistanceBetween(bot,
item)))
          .collect(Collectors.toList());
      if (allPlayer.size() > 1) {
        return allPlayer.subList(1, allPlayer.size());
      } else {
        return null;
      }

    } else {
      return gameState.getGameObjects()
          .stream()
          .filter(item -> item.getGameObjectType() == objectType
              && (BotService.getDistanceBetween(bot, item) < (distance +
bot.getSize() + item.getSize())))
          .collect(Collectors.toList());
    }

  }

  public GameObject checkObjectsNextVersion(List<GameObject> closeObjects,
GameObject bot, int vs) {
    List<GameObject> objectList = new ArrayList<>();
    if (closeObjects != null) {
      for (int i = 0; i < closeObjects.size(); i++) {
        double objDistance = BotService.getDistanceBetween(closeObjects.get(i),
bot);
```

```
        double warningTreshold = BotService.toDegrees(Math.atan2(bot.getSize() +
10, objDistance));
        double objDegree = BotService.getHeadingBetween(closeObjects.get(i),
bot);
        boolean warningStatus = (closeObjects.get(i).getCurrentHeading() >
(objDegree - warningTreshold) &&
            closeObjects.get(i).getCurrentHeading() < (objDegree +
warningTreshold)) || (BotService.getDistanceBetween(closeObjects.get(i), bot) -
closeObjects.get(i).getSize() - bot.getSize() < 60 &&
closeObjects.get(i).getGameObjectType() != ObjectTypes.TORPEDO_SALVO);

        if (closeObjects.get(i).getGameObjectType() == ObjectTypes.PLAYER) {
          if (closeObjects.get(i).getSize() > bot.getSize() && warningStatus) {
            objectList.add(closeObjects.get(i));
          }
        } else {
          if (closeObjects.get(i).gameObjectType == ObjectTypes.SUPERNOVA_BOMB) {
            warningTreshold = Math.atan2(bot.getSize() + 480, objDistance);
            warningStatus = closeObjects.get(i).getCurrentHeading() > (objDegree
- warningTreshold)
                && closeObjects.get(i).getCurrentHeading() < (objDegree +
warningTreshold);
          }
          if (warningStatus) {
            // if (closeObjects.get(i).getGameObjectType() ==
ObjectTypes.TORPEDO_SALVO) {
            //   System.out.println("heading: ");
            // }
            // System.out.println("vm: " + vm + " > vs: " + vs*Math.cos(radian));
            objectList.add(closeObjects.get(i));
          }

        }
      }
    } else {
      return null;
    }

    // System.out.println("test");

    if (objectList.size() > 0) {
      var newObjList = objectList.stream()
          .sorted(Comparator.comparing(item -> BotService.getDistanceBetween(bot,
item)))
          .collect(Collectors.toList());
```

```
      return newObjList.get(0);
    } else {
      return null;
    }
  }

  public GameObject checkRadar(GameState gameState, GameObject bot) {
    if (gameState != null) {
      var currSupernova = getCloseGameObjects(gameState, bot,
ObjectTypes.SUPERNOVA_BOMB, 1000);
      var currTele = getCloseGameObjects(gameState, bot, ObjectTypes.TELEPORTER,
250);
      var currTorpedo = getCloseGameObjects(gameState, bot,
ObjectTypes.TORPEDO_SALVO, 500);
      var currPlayer = getCloseGameObjects(gameState, bot, ObjectTypes.PLAYER,
230);

      var danger = checkObjectsNextVersion(currSupernova, bot, 40);
      if (danger != null) {
        supernovaDefend = true;
        return danger;
      }
      danger = checkObjectsNextVersion(currTele, bot, 20);
      if (danger != null) {
        teleportDefend = true;
        return danger;
      }
      danger = checkObjectsNextVersion(currTorpedo, bot, 20);
      if (danger != null) {
        torpedoDefend = true;
        return danger;
      }
      danger = checkObjectsNextVersion(currPlayer, bot, 20);
      if (danger != null) {
        playerDefend = true;
        return danger;
      }
    }
    return null;
  }
}
```

4.2 Penjelasan Struktur Data pada Bot Permainan Galaxio

Struktur data pada permainan *Galaxio* ini berbentuk class. *Class* tersebut dapat dibagi menjadi 3 kategori utama, yaitu: *Enums* yang berisi efek yang dapat dikeluarkan, tipe objek, serta *player actions* yang dapat dilakukan pada permainan *Galaxio* ; *Models* berisi komponen-komponen yang terkait dengan game objek serta suatu game state tertentu; *Services* berisi perintah-perintah yang akan dieksekusi dalam program *bot Galaxio*. Selain itu, terdapat pula *Main.java* sebagai *Class* tambahan. File "*Main.java*" merupakan sebuah kelas utama (*main class*) pada sebuah program Java yang menggunakan beberapa *library* eksternal, seperti "*com.microsoft.signalr*" dan "*org.slf4j*". Pada program ini terdapat implementasi koneksi menggunakan protokol *SignalR* untuk mengirim pesan ke *server*, serta implementasi objek-objek dari model "*GameObject*", "*Position*", dan "*GameState*". Selain itu, kelas "*BotService*" digunakan untuk mengatur aksi pemain pada permainan. Program ini akan mengirim pesan "*Register*" ke *server* menggunakan koneksi *SignalR* saat program dijalankan, dan selanjutnya akan mengirimkan aksi yang akan dilakukan oleh *bot* yang dikendalikan oleh program tersebut.

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

A. Kategori Enums

Kelas-kelas pada kategori ini merupakan kumpulan dari efek, tipe objek, serta aksi pemain yang dapat digunakan pada permainan *Galaxio*. Berikut ini penjabaran lebih lanjut dari tiap *subclass*:

i. Effects.java

Kelas "*Effects*" adalah sebuah enumerasi (*enum*) pada program *Java* yang mendefinisikan konstanta-konstanta dengan nama-nama efek pada permainan. Setiap konstanta memiliki nilai bilangan bulat (*Integer*) yang merepresentasikan kode unik untuk efek tersebut. Konstanta-konstanta efek yang didefinisikan pada kelas ini adalah:

- *AFTERBURNER*: Efek penggunaan *afterburner* pada sebuah objek dalam permainan.
- *ASTEROIDFIELD*: Efek berada di wilayah lapangan asteroid dalam permainan.
- *GASCLOUD*: Efek berada di wilayah awan gas dalam permainan.
- *SUPERFOOD*: Efek pengambilan makanan super pada permainan.
- *SHIELD*: Efek penggunaan pelindung (*shield*) pada sebuah objek dalam permainan.

ii. ObjectTypes.java

Kelas "*ObjectTypes*" adalah sebuah enumerasi (*enum*) pada program *Java* yang mendefinisikan konstanta-konstanta dengan nama-nama tipe objek pada permainan. Setiap konstanta memiliki nilai bilangan bulat (*Integer*) yang merepresentasikan kode unik untuk tipe objek tersebut. Konstanta-konstanta tipe objek yang didefinisikan pada kelas ini adalah:

- *PLAYER*: Tipe objek untuk pemain pada permainan.
- *FOOD*: Tipe objek untuk makanan pada permainan.
- *WORMHOLE*: Tipe objek untuk lubang cacing pada permainan.
- *GAS_*CLOUD: Tipe objek untuk awan gas pada permainan.
- *ASTEROID_FIELD*: Tipe objek untuk daerah asteroid pada permainan.
- *TORPEDO_SALVO*: Tipe objek untuk tembakan torpedo pada permainan.
- *SUPER_FOOD*: Tipe objek untuk makanan super pada permainan.
- *SUPERNOVA_PICKUP*: Tipe objek untuk pengambilan item *supernova* pada permainan.

- *SUPERNOVA_BOMB*: Tipe objek untuk bom *supernova* pada permainan.
- *TELEPORTER*: Tipe objek untuk *teleporter* pada permainan.
- *SHIELD*: Tipe objek untuk pelindung (*shield*) pada permainan.

iii. PlayerActions.java

Kelas "*PlayerActions*" adalah sebuah enumerasi (*enum*) pada program *Java* yang mendefinisikan konstanta-konstanta dengan nama-nama aksi pemain pada permainan. Setiap konstanta memiliki nilai bilangan bulat (*Integer*) yang merepresentasikan kode unik untuk tipe objek tersebut. Setiap aksi diberikan nilai Integer unik, dan ada sepuluh aksi yang berbeda, yaitu: *FORWARD*, *STOP*, *STARTAFTERBURNER*, *STOPAFTERBURNER*, *FIRETORPEDOES*, *FIRESUPERNOVA*, *DETONATESUPERNOVA*, *FIRETELEPORT*, *TELEPORT*, dan *ACTIVATESHIELD*. Aksi-aksi ini kemudian dapat dipilih oleh pemain saat bermain game dan digunakan untuk mengontrol objek dalam permainan.

B. Kategori Models

Kelas-kelas pada kategori ini merupakan kumpulan penjabaran dari fungsi yang terkait dengan *game objek*, *game state*, serta implementasi dari *player actions*:

i. GameObject.java

- *Attributes*

Tabel 4.2.2.1.1 Attributes pada GameObject.java

Attributes	Description
public UUID id	Atribut UUID (Universally Unique Identifier) yang dapat digunkan untuk mengidentifikasi objek dalam game.
public Integer size	Atribut bertipe integer untuk menunjukkan ukuran dari suatu objek.
public Integer speed	Atribut bertipe integer untuk menunjukkan kecepatan dari suatu objek.
public Integer currentHeading	Atribut bertipe integer untuk menunjukkan arah objek saat tertentu.
public Position position	Atribut berupa objek dari kelas *Position* yang merepresentasikan posisi objek dalam game.
public ObjectTypes gameObjectType	Atribut berupa objek dari kelas *ObjectTypes* yang merepresentasikan tipe objek dalam game.
public Integer effects	Atribut bertipe integer untuk menunjukkan efek yang sedang dimiliki oleh suatu objek.
public Integer torpedoCount	Atribut bertipe integer untuk menunjukkan jumlah torpedo yang dimiliki oleh objek.
public Integer supernovaAvailable	Atribut bertipe integer untuk menunjukkan jumlah supernova yang masih tersedia bagi objek.
public Integer teleCount	Atribut bertipe integer untuk menunjukkan jumlah perisai yang masih tersedia bagi objek.
public Integer shieldCount	Atribut bertipe integer untuk menunjukkan jumlah perisai yang masih tersedia bagi objek.

public boolean afterburner	Atribut bertipe boolean untuk menunjukkan apakah objek sedang menggunakan afterburner atau tidak.
public boolean asteroidField	Atribut bertipe boolean untuk menunjukkan apakah objek sedang berada.
public boolean gasCloud	Atribut bertipe boolean untuk menunjukkan apakah objek sedang berada di dalam *gas cloud* atau tidak.
public boolean superFood	Atribut bertipe boolean untuk menunjukkan apakah objek sedang mendapatkan *super food* atau tidak.
public boolean shield	Atribut bertipe boolean untuk menunjukkan apakah objek sedang menggunakan perisai atau tidak.

- *Methods*

Tabel 4.2.2.1.2 Methods pada GameObject.java

Methods	Description
public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, Integer effects, Integer torpedoCount, Integer supernovaAvailable, Integer teleCount, Integer shieldCount)	Method berupa constructor untuk membuat objek dengan parameter id, *size*, *speed*, *currentHeading*, *position*, *gameObjectType*, *effects*, *torpedoCount*, *supernovaAvailable*, *teleCount*, dan *shieldCount*.
public void decodeEffect(Integer effect)	Method untuk mendekode *effects* menjadi atribut boolean *asteroidFields*, *gasCloud*, *superFood*, *afterburner*, dan *shield.*
public UUID getId()	Method untuk mendapatkan id objek.
public void setId(UUID id)	Method untuk mengubah id objek.
public int getSize()	Method untuk mendapatkan ukuran objek.
public void setSize(int size)	Method untuk mengubah ukuran objek.
public int getSpeed()	Method untuk mendapatkan kecepatan objek.
public void setSpeed(int speed)	Method untuk mengubah kecepatan objek.
public Position getPosition()	Method untuk mendapatkan posisi objek.
public void setPosition(Position position)	Method untuk mengubah posisi objek.
public ObjectTypes getGameObjectType()	Method untuk mendapatkan tipe objek.
public void setGameObjectType(ObjectTypes gameObjectType)	Method untuk mengubah tipe objek.
public Integer getCurrentHeading()	Method untuk mendapatkan arah objek pada saat tertentu.
public void setCurrentHeading(Integer currentHeading)	Method untuk mengubah arah objek pada saat tertentu.
public Integer getEffects()	Method untuk mendapatkan efek yang dimiliki oleh objek.

public void setEffects(Integer effects)	Method untuk mengubah efek yang dimiliki oleh objek.
public Integer getTorpedoCount()	Method untuk mendapatkan jumlah torpedo yang dimiliki oleh objek.
public void setTorpedoCount(Integer torpedoCount)	Method untuk mengubah jumlah torpedo yang dimiliki oleh objek.
public Integer getSupernovaAvailable()	Method untuk mendapatkan jumlah supernova yang masih tersedia bagi objek.
public void setSupernovaAvailable(Integer supernovaAvailable)	Method untuk mengubah jumlah supernova yang masih tersedia bagi objek.
public Integer getTeleCount()	Method untuk mendapatkan jumlah teleportasi yang masih tersedia bagi objek.
public void setTeleCount(Integer teleCount)	Method untuk mengubah jumlah teleportasi yang masih tersedia bagi objek.
public Integer getShieldCount()	Method untuk mendapatkan jumlah perisasi yang masih tersedia bagi objek.
public void setShieldCount(Integer shieldCount)	Method untuk mengubah jumlah perisasi yang masih tersedia bagi objek.
public static GameObject FromStateList(UUID id, List<Integer> stateList)	Method yang digunakan untuk mengkonversi list of integers menjadi sebuah object 'GameObject'.

ii. GameState.java

- *Attributes*

Tabel 4.2.2.2.1 Attributes pada GameState.java

Attributes	Description
public World world	Atribut bertipe *World* yang merepresentasikan dunia pada game.
public List<GameObject> gameObjects	Atribut bertipe list yang berisi *GameObject* pada game yang termasuk kategori selain *player*.
public List<GameObject> playerGameObjects	Atribut bertipe list yang berisi *GameObject* pada *game* yang termasuk kategori *player*.

- *Methods*

Tabel 4.2.2.2.2 Methods pada GameState.java

Methods	Description
public GameState()	Method berupa constructor tanpa parameter yang digunakan untuk membuat *instance GameState* baru dengan atribut *World, gameObjects*, dan *playerGameObjects* yang kosong.
public GameState(World world , List<GameObject> gameObjects, List<GameObject> playerGameObjects)	Method berupa constuctor dengan parameter yang digunakan untuk membuat *instance GameState* baru

	dengan atribut sesuai dengan parameter yang diberikan.
public World getWorld()	Method yang digunakan untuk mengambil nilai atribut *world* pada *instance GameState*.
public void setWorld(World world)	Method yang digunakan untuk mengatur nilai atribut *world* pada *instance GameState*.
public List<GameObject> getGameObjects()	Method yang digunakan untk mengambil List dari *gameObjects* pada *instance GameState*.
public void setGameObjects(List<GameObject> gameObjects)	Method yang digunakan untuk mengatur list dari *gameObjects* pada *instance GameState*.
public List<GameObject> getPlayerGameObjects()	Method yang digunakan untuk mengambil list dari *playerGameObjects* pada *instance GameState*.
public void setPlayerGameObjects(List<GameObject> playerGameObjects)	Method yang digunakan untuk mengatur list dari *playerGameObjects* pada *instance GameState*.

iii. GameStateDto.java

- *Attributes*

Tabel 4.2.2.3.1 Attributes pada GameStateDto.java

Attributes	Description
private World world	Atribut bertipe *'Models.World'* yang merepresentasikan dunia atau lingkungan permainan dalam *game*.
private Map<String, List<Integer>> gameObjects	Atribut bertipe *'Map<String, List<Integer>>'* yang merepresentasikan objek-objek dalam permainan. Setiap objek diberi *key* yang merupakan nama dari objek tersebut. Setiap *key* memiliki *value* bertipe *'List<integer>'* yang merepresentasikan state dari objek pada suatu waktu tertentu.
private Map<String, List<Integer>> playerObjects	Atribut bertipe *'Map<String, List<Integer>>'* yang merepresentasikan objek-objek yang dimiliki oleh pemain dalam permainan. Setiap objek diberi *key* yang merupakan nama dari objek tersebut. Setiap *key* memiliki *value* bertipe *'List<Integer>'* yang merepresentasikan state dari objek pada suatu waktu tertentu.

- *Methods*

Tabel 4.2.2.3.2 Methodspada GameStateDto.java

Methods	Description

public Models.World getWorld()	Method *getter* untuk atribut *world*. Method ini mengembalikan objek dari tipe *World*.
public void setWorld(Models.World world)	Method *setter* untuk atribut *world*. Method ini menerima sebuah objek bertipe *World* dan mengeset atribut *world* dengan objek tersebut.
public Map<String, List<Integer>> getGameObjects()	Method *getter* untuk atribut *gameObjects*. Method ini mengembalikan sebuah *Map* yang memiliki *key* berupa nama objek dan *value* berupa *state* dari objek pada suatu waktu tertentu.
public void setGameObjects(Map<String, List<Integer>> gameObjects)	Method *setter* untuk atribut *gameObjects*. Method ini menerima sebuah *Map* yang memiliki *key* berupa nama objek dan *value* berupa *state* dari objek pada suatu waktu tertentu, dan mengeset atribut *gameObjects* dengan *Map* tersebut.
public Map<String, List<Integer>> getPlayerObjects()	Method *getter* untuk atribut *playerObjects*. Method ini mengembalikan sebuah *Map* yang memiliki *key* berupa nama objek yang dimiliki oleh pemain dan *value* berupa *state* dari objek tersebut pada suatu waktu tertentu.
public void setPlayerObjects(Map<String, List<Integer>> playerObjects)	Method *setter* untuk atribut *playerObjects*. Method ini menerima sebuah *Map* yang memiliki *key* berupa nama objek yang dimiliki oleh pemain dan *value* berupa *state* dari objek tersebut pada suatu waktu tertentu, dan mengeset atribut *playerObjects* dengan *Map* tersebut.

iv. PlayerAction.java

- *Attributes*

Tabel 4.2.2.4.1 Attributes pada PlayerAction.java

Attributes	Description
public UUID playerId	Atribut bertipe *UUID* dan merepresentasikan *ID* unik dari pemain yang mengambil aksi tersebut.
public PlayerActions action	Atribut bertipe *PlayerActions* dan merepresentasikan aksi yang diambil oleh pemain, diambil dari *enum PlayerActions*
public int heading	Atribut bertipe *integer* dan merepresentasikan arah yang diambil oleh pemain.

- *Methods*

Tabel 4.2.2.4.2 Methods pada PlayerAction.java

Methods	Description
public UUID getPlayerId()	Method *getter* untuk atribut *playerId*.
public void setPlayerId(UUID playerId)	Method *setter* untuk atribut *playerId*.
public PlayerActions getAction()	Method *getter* untuk atribut *action*.
public void setAction(PlayerActions action)	Method *setter* untuk atribut *action*.
public int getHeading()	Method *getter* untuk atribut *heading*.
public void setHeading(int heading)	Method *setter* untuk atribut *heading*.

v. Position.java

- *Attributes*

Tabel 4.2.2.5.1 Attributes pada Position.java

Attributes	Description
public int x	Atribut bertipe data *integer* yang merepresentasikan koordinat x dari posisi.
public int y	Atribut bertipe data *integer* yang merepresentasikan koordinat y dari posisi.

- *Methods*

Tabel 4.2.2.5.2 Methods pada Position.java

Methods	Description
public int getX()	Method untuk mengembalikan nilai koordinat x dari posisi.
public void setX(int x)	Method yang digunakan untuk mengubah nilai koordinat x dari posisi dengan nilai yang diberikan pada parameter.
public int getY()	Method yang mengembalikan nilai koordinat y dari posisi.
public void setY(int y)	Method yang digunakan untuk mengubah nilai koordinat y dari posisi dengan nilai yang diberikan pada parameter.

vi. World.java

- *Attributes*

Tabel 4.2.2.6.1 Attributes pada World.java

Attributes	Description
public Position centerPoint	Atribut bertipe *Position* yang merepresentasikan titik tengah dari dunia *game*.
public Integer radius	Atribut bertipe *Integer* yang merepresentasikan jari-jari lingkaran yang mengelilingi dunia *game*.
public Integer currentTick	Atribut bertipe *Integer* yang merepresentasikan waktu saat ini pada dunia *game*.

- *Methods*

Tabel 4.2.2.6.2 Methods pada World.java

Methods	Description
public Position getCenterPoint()	Method yang mengembalikan objek *Position* yang merepresentasikan titik tengah dari dunia *game*.
public void setCenterPoint(Position centerPoint)	Method untuk mengatur nilai atribut *centerPoint* dengan *objek Position* yang diinputkan.
public Integer getRadius()	Method yang mengembalikan nilai *Integer* yang merepresentasikan jari-jari lingkaran yang mengelilingi dunia *game*.
public void setRadius(Integer radius)	Method untuk mengatur nilai atribut radius dengan nilai *Integer* yang diinputkan.
public Integer getCurrentTick()	Method yang mengembalikan nilai *Integer* yang merepresentasikan waktu saat ini pada dunia *game*.
public void setCurrentTick(Integer currentTick)	Method untuk mengatur nilai atribut *currentTick* dengan nilai *Integer* yang diinputkan.

C. Kategori Services

Kelas-kelas pada kategori ini merupakan kumpulan penjabaran dari methods dan attributes pada bot yang digunakan oleh kelompok kami:

i. BotService.java

- *Attributes*

Tabel 4.2.3.1.1 Attributes pada BotService.java

Attributes	Description
private GameObject bot	Atribut bertipe *GameObject* yang merepresentasikan *bot*.
private PlayerAction playerAction	Atribut bertipe *PlayerAction* yang merepresentasikan aksi *bot* sekarang.
private GameState gameState	Atribut bertipe *GameState* yang merepresentasikan *currentState* dari permainan.

- *Methods*

Tabel 4.2.3.1.2 Methods pada BotService.java

Methods	Description
public GameObject getBot()	Method getter untuk *GameObject bot*.
public void setBot(GameObject bot)	Method setter untuk *GameObject bot*.
public PlayerAction getPlayerAction()	Method dengan *return type PlayerAction* yang akan mengembalikan nilai dari atribut *playerAction*.
public void setPlayerAction(PlayerAction playerAction)	Method yang digunakan untuk mengubah nilai atribut *playerAction*.

	Method ini tidak mengembalikan nilai (*void*), dan menerima parameter *playerAction* yang merupakan objek *PlayerAction* yang akan digunakan untuk mengganti nilai *playerAction* pada objek saat ini.
public int teleportSerang(PlayerAction playerAction, GameObject enemy)	Method yang digunakan untuk menentukan aksi yang akan dilakukan *bot* pada serangan *teleport*. Method ini tidak mengembalikan nilai (*void*), dan menerima beberapa parameter yaitu *playerAction*, *enemy* (*GameObject*). Method ini mengeset *playerAction.action* menjadi *PlayerActions.FIRETELEPORT*, menghitung heading menuju *enemy*, dan menampilkan pesan "*Attack with launch teleport bot size*: x *enemy size*: y *distance*: z" dengan x dan y adalah ukuran *bot* dan *enemy*, dan z adalah jarak antara *bot* dan *enemy*.
public int checkTeleportSerang(PlayerAction playerAction, int headingTele, UUID preyTracker, List<GameObject> enemyList)	Method yang digunakan untuk memeriksa apakah serangan *teleport* yang dilakukan sebelumnya berhasil atau tidak. Method ini mengembalikan nilai *GameObject* atau *null*. Method ini menerima beberapa parameter yaitu p (*PlayerAction*), *bot* (*GameObject*), *launchedTeleport* (*GameObject*), *enemyList* (*List<GameObject>*), dan *teleportList* (*List<GameObject>*). Method ini akan mengembalikan *null* jika serangan *teleport* berhasil. Jika serangan *teleport* tidak berhasil, method ini akan mengembalikan objek *launchedTeleport*.
public void computeNextPlayerAction(PlayerAction playerAction)	Method yang digunakan untuk menghitung aksi selanjutnya yang akan dilakukan *bot*. Method ini tidak mengembalikan nilai (*void*), dan menerima parameter *playerAction* yang merupakan objek *PlayerAction* yang akan diberikan aksi pada *bot*. Method ini mengumpulkan informasi tentang keadaan *game*, *playerState*, *teleportState*, *preyList*, *torpedoTarget*, *closestEnemy*, dan *objectRadar*. Kemudian method ini melakukan pengecekan terhadap bahaya yang muncul dan memilih aksi yang akan

	dilakukan *bot* berdasarkan hasil pengecekan tersebut.
public void setGameState(GameState gameState)	Method *setter* yang diigunakan untuk memilih *GameState* secara spesifik, kemudian memanggil method *updateSelfState* untuk memperbarui *bot state* terhadap *game state* baru.
private void updateSelfState()	Method untuk memperbarui keadaan *bot* berdasarkan keadaan permainan saat ini, khususnya dengan mencari objek *bot* dalam daftar objek permainan pemain dan menetapkannya sebagai *bot* dalam kelas.
public static double getDistanceBetween(GameObject object1, GameObject object2)	Method untuk menghitung jarak antara dua objek *GameObject* menggunakan teorema *Pythagoras*.
public static int getHeadingBetween(GameObject bot, GameObject otherObject)	Method untuk menghitung arah dalam derajat antara *bot* dan objek lain. Ini dilakukan dengan menggunakan metode *atan2*() untuk menemukan sudut antara kedua objek, dan mengonversi sudut menjadi derajat.
public static int getHeadingBetween(GameObject bot, int x, int y)	Method untuk menghitung arah dalam derajat antara *bot* dan set koordinat yang ditentukan oleh x dan y. Ini mirip dengan metode sebelumnya, tetapi menggunakan koordinat daripada objek *GameObject*.
public static int toDegrees(double v)	Method untuk mengonversi sudut dalam radian menjadi derajat.
public boolean launchTorpedo(GameObject bot, GameObject enemy, int distanceTreshold, PlayerAction p, boolean attack)	Method untuk memeriksa apakah bot dapat meluncurkan *torpedo* pada musuh yang ditentukan, dan jika dapat, mengatur objek *PlayerAction* ke nilai yang sesuai untuk melakukannya. Metode ini mengambil objek *bot*, objek musuh, objek *PlayerAction* untuk dimodifikasi, *flag boolean attack* yang menunjukkan apakah *bot* menyerang atau bertahan, dan daftar objek permainan untuk diperiksa adanya penghalang. Metode ini menghitung beberapa kondisi untuk menembakkan *torpedo*, termasuk ukuran *bot*, ukuran musuh, jarak antara *bot* dan musuh, jumlah *torpedo* yang tersedia, dan apakah ada penghalang di antara keduanya. Jika semua kondisi terpenuhi, objek *PlayerAction* diatur untuk menembakkan *torpedo* ke arah

	yang sesuai. Metode ini mengembalikan nilai boolean yang menunjukkan apakah *torpedo* telah diluncurkan atau tidak.

ii. GreedyCommand.java

- *Methods*

Tabel 4.2.3.2.1 Methods pada GreedyCommand.java

Methods	Description
public static PlayerAction run(GameObject enemy, PlayerAction p, UUID objectTracker, GameObject bot, boolean switchDirection, int direction)	Method yang digunakan untuk mengambil sebuah objek *GameObject* yang mewakili musuh, sebuah objek *PlayerAction* yang mewakili aksi saat ini dari *bot*, sebuah objek *UUID* yang mewakili objek saat ini yang sedang dilacak oleh *bot*, sebuah objek *GameObject* yang mewakili *bot* saat ini, sebuah *boolean* yang menunjukkan apakah *bot* harus mengganti arah, dan sebuah *integer* yang mewakili arah. Metode ini mencetak beberapa informasi mengenai kecepatan, ukuran, status, efek, jarak dari musuh, ukuran musuh, dan tipe objek. Kemudian, metode ini memeriksa tipe objek musuh dan mengaktifkan *afterburner bot* jika musuh bukan *torpedo salvo*. Akhirnya, metode ini memperbarui *heading* dari *bot* dan mengembalikan objek *PlayerAction* yang diperbarui.
public static void activateAfterburner(PlayerAction p, GameObject bot, GameObject enemy, boolean attack)	Method yang digunakan untuk mengambil sebuah objek *PlayerAction* yang mewakili aksi saat ini dari *bot*, sebuah objek *GameObject* yang mewakili *bot* saat ini, sebuah objek *GameObject* yang mewakili musuh, dan sebuah *boolean* yang menunjukkan apakah *bot* sedang menyerang musuh. Metode ini menentukan apakah *bot* harus memulai, menghentikan, atau terus menggunakan *afterburner* berdasarkan kondisi-kondisi tertentu, dan memperbarui objek *PlayerAction* sesuai dengan kondisi tersebut.
public static UUID checkBorder(GameObject bot, World w, PlayerAction p, UUID objectTracker)	Method yang digunakan untuk mengambil sebuah objek *GameObject* yang mewakili *bot* saat ini, sebuah *World* yang mewakili dunia permainan, sebuah objek *PlayerAction* yang mewakili aksi saat ini dari *bot*, dan sebuah objek *UUID*

	yang mewakili objek saat ini yang sedang dilacak oleh *bot*. Metode ini memeriksa apakah *bot* berada dekat dengan batas dunia permainan, dan memperbarui objek *PlayerAction* untuk menghindari batas tersebut jika diperlukan.
public static UUID checkHarmfulObject(GameObject bot, List<GameObject> gameObj, PlayerAction p, ObjectTypes objectType, int treshold, String text, UUID objectTracker)	Method yang digunakan untuk menerima objek *GameObject* yang mewakili *bot* saat ini, daftar objek *GameObject* yang mewakili objek lain di dunia permainan, objek *PlayerAction* yang mewakili tindakan saat ini dari *bot*, objek *ObjectTypes* yang mewakili jenis objek berbahaya yang akan diperiksa, sebuah bilangan bulat yang mewakili jarak ambang batas ke objek berbahaya, sebuah string yang mewakili teks untuk dicetak jika bot perlu menghindari objek berbahaya, dan objek *UUID* yang mewakili objek saat ini yang sedang dilacak oleh *bot*. Metode ini memeriksa apakah ada objek berbahaya dalam jarak ambang batas dari *bot* dan memperbarui objek *PlayerAction* untuk menghindari objek berbahaya jika perlu.
public static UUID evadeObject(GameObject bot, List<GameObject> gameObj, World w, PlayerAction p, UUID objectTracker)	Method yang digunakan untuk menerima objek *GameObject* yang mewakili *bot* saat ini, daftar objek *GameObject* yang mewakili objek lain di dunia *game*, objek *World* yang mewakili dunia *game*, objek *PlayerAction* yang mewakili tindakan saat ini dari *bot*, dan objek *UUID* yang mewakili objek saat ini yang sedang dilacak oleh *bot*. Metode memeriksa apakah *bot* dekat dengan *torpedo*, dan memperbarui objek *PlayerAction* untuk menghindari objek jika perlu. Jika *bot* dekat dengan *torpedo*, metode juga memperbarui *objectTracker* untuk melacak *torpedo*.
public static void catchFood(List<GameObject> allState, GameObject bot, PlayerAction p)	Method yang digunakan untuk menangkap makanan oleh *bot* dalam *game*. Method ini mengambil tiga parameter, yaitu sebuah *List<GameObject> allState* yang merepresentasikan seluruh objek dalam game, sebuah *GameObject bot* yang merepresentasikan *bot* yang menangkap makanan, dan sebuah *PlayerAction* p yang merepresentasikan tindakan yang

	diambil oleh *bot*. Dalam method ini, terdapat beberapa langkah yang dilakukan untuk menentukan makanan mana yang akan diambil oleh *bot*, di antaranya adalah melakukan *filtering* pada *list allState* untuk mencari objek makanan, melakukan sorting berdasarkan jarak terdekat, dan menentukan arah dan tindakan yang harus dilakukan oleh *bot* untuk menangkap makanan tersebut. Jika tidak ada makanan yang ditemukan dalam *game*, *bot* akan bergerak maju ke arah yang ditentukan oleh method *getHeadingBetween*(). Sedangkan jika ada makanan yang ditemukan, *bot* akan bergerak maju ke arah makanan terdekat menggunakan tindakan *PlayerActions.FORWARD*.

iii. Radar.java

- *Attributes*

Tabel 4.2.3.3.1 Attributes pada Radar.java

Attributes	Description
public boolean supernovaDefend	Atribut bertipe *boolean* yang digunakan untuk menyimpan status apakah objek *super nova* dalam radius jangkauan atau tidak.
public boolean teleportDefend	Atribut bertipe *boolean* yang digunakan untuk menyimpan status apakah objek *teleporter* dalam radius jangkauan atau tidak.
public boolean torpedoDefend	Atribut bertipe *boolean* yang digunakan untuk menyimpan status apakah objek *torpedo salvo* dalam radius jangkauan atau tidak.
public boolean playerDefend	Atribut bertipe *boolean* yang digunakan untuk menyimpan status apakah ada objek *player* dalam radius jangkauan atau tidak.
public static final int ALL_CLEAR = 0	Atribut bertipe *boolean* yang digunakan untuk menandakan tidak ada ancaman apa pun.
public static final int TELEPORTDEFENCE = 1	Atribut bertipe *boolean* yang digunakan untuk menandakan ancaman *teleporter*.
public static final int SUPERNOVADEFENCE = 2	Atribut bertipe *integer* yang digunakan untuk menandakan ancaman *super nova*.
public static final int TORPEDODEFENCE = 3	Atribut bertipe *integer* yang digunakan untuk menandakan ancaman *torpedo salvo*.

- *Methods*

Tabel 4.2.3.3.2 Methods pada Radar.java

Methods	Description
public Radar()	Method berupa konstruktor kelas Radar, digunakan untuk menginisialisasi semua atribut dalam kelas ini dengan *false*.
public List<GameObject> getCloseGameObjects(GameState gameState, GameObject bot, ObjectTypes objectType, int distance)	Method yang digunakan untuk mendapatkan semua objek *game* dalam radius jarak tertentu dari objek pemain *bot*. Objek yang dicari tergantung pada parameter *objectType* yang diberikan. Method ini mengembalikan daftar objek dalam radius jarak yang ditentukan.
public GameObject checkObjectsNextVersion(List<GameObject> closeObjects, GameObject bot, int vs)	Method yang digunakan untuk mengecek apakah ada objek *game* yang berbahaya dalam radius jarak tertentu dari objek pemain *bot*. Method ini memeriksa apakah objek berada dalam radius jangkauan pemain dan apakah arah objek berpotensi mengancam pemain. Method ini mengembalikan objek game yang paling dekat dengan pemain jika ditemukan.
public GameObject checkRadar(GameState gameState, GameObject bot)	Method utama dalam kelas ini yang digunakan untuk mengecek apakah ada ancaman dalam radius jangkauan pemain. Method ini memeriksa daftar objek game dalam radius jangkauan dan memanggil method *checkObjectsNextVersion* untuk mengecek apakah objek *game* tersebut berbahaya. Jika objek *game* berbahaya, method ini mengembalikan objek *game* yang paling dekat dengan pemain dan mengatur atribut yang sesuai dengan jenis ancaman. Jika tidak ada ancaman, method ini mengembalikan *null*.

iv. TeleSuperNova.java

- *Attributes*

Tabel 4.2.3.4.1 Attributes pada TeleSuperNova.java

Attributes	Description
private GameObject bot	Atribut bertipe *GameObject* dan digunakan untuk menyimpan informasi tentang objek *bot* pada permainan.
private PlayerAction playerAction	Atribut bertipe *PlayerAction* dan digunakan untuk menyimpan informasi tindakan yang akan dilakukan oleh objek pemain pada permainan.

private GameState gameState	Atribut bertipe *GameState* dan digunakan untuk menyimpan informasi tentang kondisi *game* pada suatu waktu tertentu.
private int headingTele	Atribut bertipe *int* dan digunakan untuk menyimpan informasi arah *teleporter* pada permainan.
private int headingSupernova	Atribut bertipe *int* dan digunakan untuk menyimpan informasi arah *supernova* pada permainan.

- *Methods*

Tabel 4.2.3.4.2 Methods pada TeleSuperNova.java

Methods	Description
public TeleSuperNova()	Method berupa *constructor* untuk kelas *TeleSuperNova*. Method ini akan membuat instance dari kelas *PlayerAction* dan *GameState* yang akan digunakan dalam permainan.
public GameObject getBot()	Method yang digunakan untuk mengambil nilai atribut *bot*.
public void setBot(GameObject bot)	Method yang digunakan untuk mengubah nilai atribut *bot*.
public PlayerAction getPlayerAction()	Method yang digunakan untuk mengambil nilai atribut *playerAction*.
public void setPlayerAction(PlayerAction playerAction)	Method yang digunakan untuk mengubah nilai atribut *playerAction*.
public void tembakTorpedo(PlayerAction playerAction)	Method yang digunakan untuk menembakkan *torpedo* ke musuh yang terdekat pada permainan.
public int teleportSerang(PlayerAction playerAction, int firedTeleport)	Method yang digunakan untuk melakukan teleportasi ke arah musuh dan menembakkan *torpedo* ke musuh yang terdekat pada permainan.
public int checkTeleportSerang (PlayerAction playerAction, int headingTele)	Method yang digunakan untuk melakukan pengecekan apakah *teleporter* sudah siap digunakan dan melakukan teleportasi pada permainan.
public int teleportBertahan(PlayerAction playerAction)	Method yang digunakan untuk melakukan teleportasi ke suatu area yang aman pada permainan.
public int checkTeleportBertahan (PlayerAction playerAction, int headingTele)	Method yang digunakan untuk melakukan pengecekan apakah *teleporter* masih berada dalam permainan dan melakukan teleportasi ke arah yang aman pada permainan.
public int tembakSuperNova(PlayerAction playerAction)	Method ini bertujuan untuk menembakkan *supernova*. Method ini memiliki satu parameter yaitu *playerAction* yang merupakan objek *PlayerAction*. Method ini mengubah nilai dari

	playerAction.action menjadi *PlayerActions.FIRETELEPORT*. Selain itu, method ini juga melakukan pengecekan pada objek *gameState* apakah terdapat objek *GameObject* atau tidak. Jika tidak ada objek *GameObject*, maka method ini tidak akan melakukan apa-apa. Jika terdapat objek *GameObject*, maka method akan mengambil objek *GameObject* dengan jenis *ObjectTypes.PLAYER* dan menyimpannya dalam sebuah *list*. *List* ini kemudian diurutkan berdasarkan jarak antara *bot* dan objek *GameObject* dengan menggunakan fungsi *getDistanceBetween*(*bot*, *item*). Selanjutnya, method ini akan mengubah nilai dari *playerAction.heading* dengan memanggil fungsi *getHeadingBetween*(*playerList.get*(1)) pada elemen kedua dari *list playerList* yang diurutkan sebelumnya. Terakhir, method akan mengembalikan nilai dari *playerAction.heading*.
public int checkDetonateSuperNova (PlayerAction playerAction, int headingSupernova)	Method ini bertujuan untuk mengecek apakah bot memiliki *supernova* dan apakah objek *GameObject* dengan jenis *ObjectTypes.SUPERNOVA_BOMB* masih ada dalam permainan. Method ini memiliki dua parameter yaitu *playerAction* yang merupakan objek *PlayerAction*, dan *headingSupernova* yang merupakan *heading* dari *supernova* yang ditembakkan sebelumnya. Method ini mengembalikan nilai *integer* yang menunjukkan heading *supernova* atau -1 jika tidak ada *supernova* yang tersedia atau jarak antara musuh dan *supernova* lebih jauh dari radius tambah 50. Method ini pertama-tama mengambil objek *GameObject* dengan jenis *ObjectTypes.SUPERNOVA_BOMB* dan *getCurrentHeading*() sama dengan *headingSupernova*. Objek-objek ini kemudian diurutkan berdasarkan jarak antara *bot* dan objek *GameObject* dengan menggunakan fungsi *getDistanceBetween*(*bot*, *item*). Setelah itu, method ini mengambil objek *GameObject* dengan jenis *ObjectTypes.PLAYER* dan mengurutkannya berdasarkan jarak antara supernova dan objek *GameObject* dengan

	menggunakan fungsi *getDistanceBetween*(*supernovaList*.*get*(0), item). Jika *supernovaList* kosong, method ini akan mengembalikan -1. Jika tidak, method ini akan membandingkan jarak antara musuh terdekat dan *supernova* dengan menggunakan variabel *supernovaDistance*. Jika *supernovaDistance* kurang dari atau sama dengan radius bot ditambah 50, maka method ini akan mengubah nilai dari *playerAction*.*action* menjadi *PlayerActions*.*DETONATESUPERNOVA* dan mengembalikan nilai -1. Jika tidak, method akan mengembalikan nilai *headingSupernova* untuk menunggu hingga *tick* selanjutnya.
public void computeNextPlayerAction(PlayerAction playerAction)	Method ini digunakan untuk menghitung aksi selanjutnya yang akan dilakukan oleh *player*. Pada bagian awal method, *playerAction.action* di-set menjadi *PlayerActions*.*FORWARD*, dan *playerAction*.*heading* di-*generate* secara acak dalam rentang 0 hingga 359. Selanjutnya, jika *game state* tidak kosong, method akan mencari objek makanan (*ObjectTypes*.*FOOD*) dalam *game state* dan mengurutkannya berdasarkan jarak terdekat dari posisi *bot*. Kemudian, *playerAction.heading* akan di-*set* sesuai dengan arah yang menuju objek makanan terdekat.
public GameState getGameState()	Method ini digunakan untuk mendapatkan *game state*.
public void setGameState(GameState gameState)	Method ini digunakan untuk mengubah *game state* dan memperbarui *state* objek *bot*.
private void updateSelfState()	Method ini digunakan untuk memperbarui *state* objek *bot* berdasarkan *game state* yang ada.
private double getDistanceBetween(GameObject object1, GameObject object2)	Method ini digunakan untuk menghitung jarak antara dua objek.
private int getHeadingBetween(GameObject otherObject)	Method ini digunakan untuk menghitung arah yang harus diambil untuk menuju objek lain (dalam hal ini, objek makanan terdekat).
private int toDegrees(double v)	Method untuk mengonversi sudut dalam radian menjadi derajat.

4.3 Pengujian Bot serta Analisis Performansi Permainan Galaxio

Dalam pengujiannya, kami mempertandingkan bot kami, BestGolKiper, dengan dua bot milik tim lain, YasinBot dan eresElMejor, serta dengan ReferenceBot. Karena permainan ini juga mengandalkan faktor keberuntungan kami perlu melakukan beberapa pertandingan agar hasil yang kami dapatkan akurat.

Berdasarkan tujuh pertandingan percobaan yang telah kami lakukan, kami mendapatkan hasil berupa 4 kemenangan dan 3 kekalahan. Berikut merupakan GameStateLog GameComplete dari tiap pertandingan yang kami ujikan.

- Pertandingan 1:

logger-publish > {} GameStateLog_2023-02-17_04-30-31_GameComplete.json > ...


```
{"TotalTicks":345,"Players":[
  {"Placement":1,"Seed":24421,"Score":67,"Id":"a4c5a5a0-b013-40a6-bfe7-fbae6aeb0873",
  "Nickname":"BestGolKiper","MatchPoints":8},
  {"Placement":3,"Seed":21180,"Score":59,"Id":"65d8bdc1-b958-45bc-b5cf-7c0f0def4317",
  "Nickname":"eresElMejor","MatchPoints":4},
  {"Placement":2,"Seed":24775,"Score":53,"Id":"0862fdd8-25ff-4415-a941-0aacd67a6d5a",
  "Nickname":"Yasin_Bot","MatchPoints":6},
  {"Placement":4,"Seed":7666,"Score":7,"Id":"3e926311-cff3-42a1-bb04-bd5c4256cd58",
  "Nickname":"ReferenceBot","MatchPoints":2}],
  "WorldSeeds":[34681],"WinningBot":{
    "Id":"a4c5a5a0-b013-40a6-bfe7-fbae6aeb0873","Size":54,"Speed":4,
    "GameObjectType":1,"CurrentHeading":112,"Position":{"X":-134,"Y":103}
  }}
```


Gambar 4.1. *Pertandingan 1 Memperoleh Peringkat 1*
Sumber: Dokumen Penulis

- Pertandingan 2:

logger-publish > {} GameStateLog_2023-02-17_04-40-49_GameComplete.json > ...


```
{"TotalTicks":514,"Players":[
  {"Placement":1,"Seed":26200,"Score":83,"Id":"77611031-7015-4a62-bf2f-bb62f81f88f0",
  "Nickname":"Yasin_Bot","MatchPoints":8},
  {"Placement":2,"Seed":13684,"Score":63,"Id":"66e4797c-b553-49a9-a762-1a13abe0b54f",
  "Nickname":"BestGolKiper","MatchPoints":6},
  {"Placement":3,"Seed":11596,"Score":15,"Id":"aceef580-c0b9-4c51-8df0-6bbd05c6a940",
  "Nickname":"eresElMejor","MatchPoints":4},
  {"Placement":4,"Seed":13785,"Score":12,"Id":"98f9b94b-fa13-4275-adc1-8fbbb6717ad1",
  "Nickname":"ReferenceBot","MatchPoints":2}],"WorldSeeds":[36581],"WinningBot":{
    "Id":"77611031-7015-4a62-bf2f-bb62f81f88f0","Size":143,"Speed":2,"GameObjectType":1,
    "CurrentHeading":40,"Position":{"X":-31,"Y":-315}
  }
}
```


Gambar 4.2. *Pertandingan 2 Memperoleh Peringkat 2*
Sumber: Dokumen Penulis

- Pertandingan 3:

logger-publish > {} GameStateLog_2023-02-17_04-45-06_GameComplete.json > ...


```
{"TotalTicks":423,"Players":[
  {"Placement":1,"Seed":499,"Score":35,"Id":"61c19572-adc4-427e-b468-2e05d2c95f47"
  ,"Nickname":"BestGolKiper","MatchPoints":8},
  {"Placement":4,"Seed":17460,"Score":2,"Id":"4781e06b-ad73-4932-ad3f-781978a57a4a"
  ,"Nickname":"Yasin_Bot","MatchPoints":2},
  {"Placement":2,"Seed":15080,"Score":52,"Id":"58fba7ef-d77b-468d-9155-93c6e5519c2f"
  ,"Nickname":"eresElMejor","MatchPoints":6},
  {"Placement":3,"Seed":3538,"Score":33,"Id":"514228fb-b578-4a34-b9f5-558237ea9c80"
  ,"Nickname":"ReferenceBot","MatchPoints":4}],"WorldSeeds":[24927],"WinningBot":{
    "Id":"61c19572-adc4-427e-b468-2e05d2c95f47","Size":114,"Speed":2,"GameObjectType":1,
    "CurrentHeading":268,"Position":{"X":219,"Y":111}
  }
}
```


Gambar 4.3. *Pertandingan 3 Memperoleh Peringkat 1*
Sumber: Dokumen Penulis

- Pertandingan 4:


```
{"TotalTicks":655,"Players":[
  {"Placement":1,"Seed":28226,"Score":73,"Id":"9abc9d38-266d-4d29-9ce5-94f00d941d48"
  ,"Nickname":"BestGolKiper","MatchPoints":8},
  {"Placement":3,"Seed":1210,"Score":71,"Id":"ad82745b-73a1-46ff-a579-e800bc9c57ac"
  ,"Nickname":"Yasin_Bot","MatchPoints":4},
  {"Placement":4,"Seed":28937,"Score":23,"Id":"2d695ad8-acba-4ae2-841d-507fa6522be4"
  ,"Nickname":"ReferenceBot","MatchPoints":2},
  {"Placement":2,"Seed":9303,"Score":60,"Id":"24c7bf13-7ddf-49c8-9bb0-74230a75478c"
  ,"Nickname":"eresElMejor","MatchPoints":6}],"WorldSeeds":[43260],"WinningBot":{
    "Id":"9abc9d38-266d-4d29-9ce5-94f00d941d48","Size":46,"Speed":5,"GameObjectType":1,
    "CurrentHeading":285,"Position":{"X":194,"Y":46}
  }
}
```


Gambar 4.4. *Pertandingan 4 Memperoleh Peringkat 1*
Sumber: Dokumen Penulis

- Pertandingan 5:


```
{"TotalTicks":476,"Players":[
  {"Placement":1,"Seed":19575,"Score":81,"Id":"97a485a2-0f4b-4228-950c-1bc0aa8ef419"
  ,"Nickname":"BestGolKiper","MatchPoints":8},
  {"Placement":2,"Seed":14386,"Score":76,"Id":"eeef4bf7-91aa-4c02-bb79-25cdcc826742"
  ,"Nickname":"Yasin_Bot","MatchPoints":6},
  {"Placement":4,"Seed":26483,"Score":11,"Id":"93cab0a5-be64-43f8-8be0-98473c763656"
  ,"Nickname":"ReferenceBot","MatchPoints":2},
  {"Placement":3,"Seed":6697,"Score":26,"Id":"284f9c05-f11f-4b28-a408-cb19178a75c6"
  ,"Nickname":"eresElMejor","MatchPoints":4}],"WorldSeeds":[20829],"WinningBot":{
    "Id":"97a485a2-0f4b-4228-950c-1bc0aa8ef419","Size":19,"Speed":11,"GameObjectType":1,
    "CurrentHeading":51,"Position":{"X":-140,"Y":417}
  }
}
```


Gambar 4.5. *Pertandingan 5 Memperoleh Peringkat 1*
Sumber: Dokumen Penulis

- Pertandingan 6:


```
{"TotalTicks":436,"Players":[
  {"Placement":1,"Seed":27649,"Score":46,"Id":"f7295ca3-009d-4995-a5bf-8c2b75743f34"
  ,"Nickname":"Yasin_Bot","MatchPoints":8},
  {"Placement":2,"Seed":20943,"Score":28,"Id":"007455b6-1958-492f-916c-f8f38ab44ba2"
  ,"Nickname":"BestGolKiper","MatchPoints":6},
  {"Placement":3,"Seed":18186,"Score":35,"Id":"a9cf88db-cff4-488e-a633-bd9663d1ca2b"
  ,"Nickname":"eresElMejor","MatchPoints":4},
  {"Placement":4,"Seed":21902,"Score":17,"Id":"2acb2ea9-75e5-44a3-95e2-d26f27944230"
  ,"Nickname":"ReferenceBot","MatchPoints":2}],"WorldSeeds":[38513],"WinningBot":{
    "Id":"f7295ca3-009d-4995-a5bf-8c2b75743f34","Size":35,"Speed":6,"GameObjectType":1,
    "CurrentHeading":210,"Position":{"X":189,"Y":260}
  }
}
```

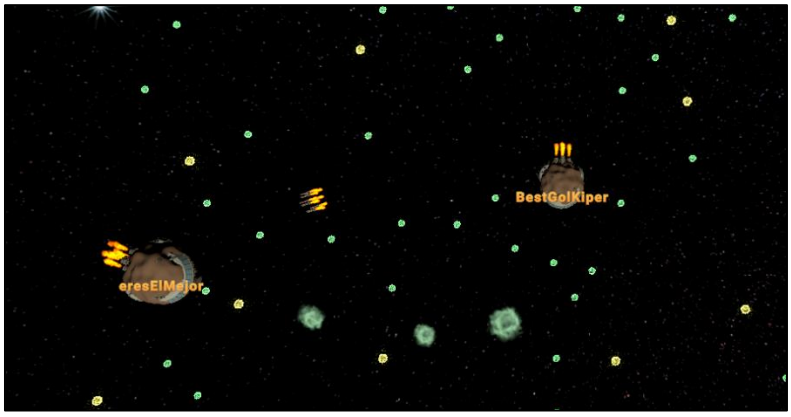

Gambar 4.6. *Pertandingan 6 Memperoleh Peringkat 2*
Sumber: Dokumen Penulis

- Pertandingan 7:

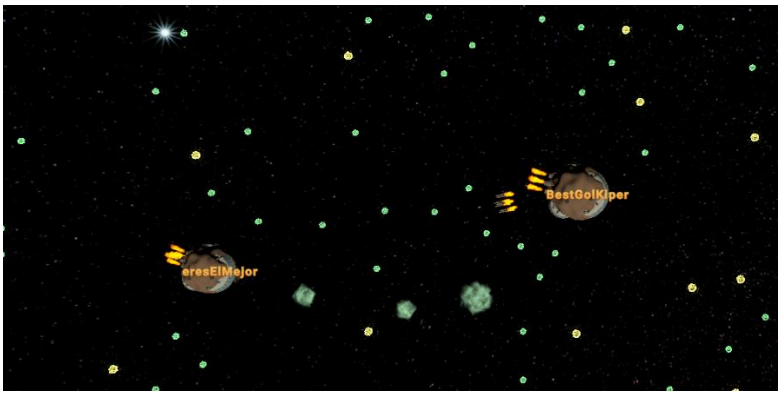
Gambar 4.7. *Pertandingan 7 Memperoleh Peringkat 2*
Sumber: Dokumen Penulis

Berdasarkan percobaan pertandingan yang telah kami lakukan, kami berhasil memenangkan 4 pertandingan dari 7 pertandingan. Dengan persentase kemenangan 57,14 persen, dapat disimpulkan bahwa algoritma greedy yang kami buat dapat berjalan dengan efektif sesuai dengan strategi greedy yang kami buat. Namun masih banyak hal yang dapat kami kembangkan lebih lanjut mengenai algoritma greedy kami. Setelah memvisualisasikan ketujuh pertandingan tersebut, kami dapat mengamati bahwa fungsi-fungsi yang kami buat berdasarkan strategi kami berhasil dijalankan dan diterapkan dalam permainan. Contoh-contohnya adalah sebagai berikut.

Berdasarkan strategi *Attack Phase*, kami dapat melakukan penyerangan dengan torpedo dan teleporter untuk melahap bot musuh. Pada pertandingan 1, kami sukses menggunakan teleporter untuk memangsa musuh serta menggunakan torpedo untuk menembaki musuh dan meningkatkan ukuran. Hal tersebut sesuai dengan gambar hasil pertandingan di bawah ini.

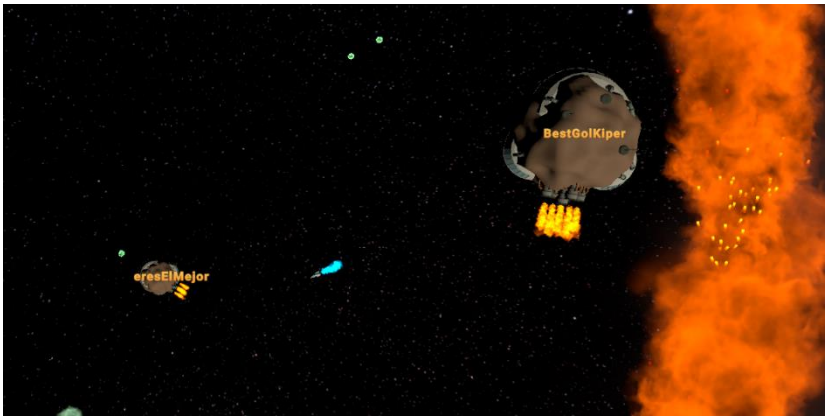


Gambar 4.8. *Bot Melepaskan Torpedo pada Attack Phase*
Sumber: Dokumen Penulis

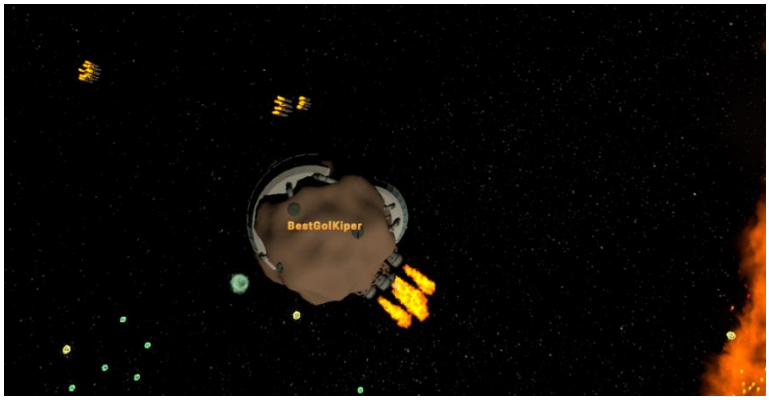


Gambar 4.9. *Torpedo Bot Mengenai Musuh pada Attack Phase*
Sumber: Dokumen Penulis

Strategi penyerangan pertama yang dilakukan pada *Attack Phase* adalah dengan menggunakan torpedo. Sesuai dengan *Method tembakTorpedo* pada *Modul BotService.java* kami akan menembakkan torpedo terhadap objek *Player* terdekat setelah melakukan mekanisme *sorting* berdasarkan jarak pada seluruh objek *Player* yang ada. *Bot* kemudian akan merubah heading-nya terlebih dahulu berdasarkan fungsi *getHeadingBetween* agar bisa mengarahkan pada musuh yang menjadi incaran serangan. Selanjutnya, *Bot* akan mempertimbangkan dengan berbagai parameter yang ada untuk menembak torpedo ke musuh dengan susunan yang paling efektif (beruntun atau berselang) tergantung pada perpindahan posisi musuh, *stock torpedo* yang dimiliki oleh *bot*, dan berbagai pertimbangan lainnya. Untuk lebih lengkapnya, terdapat *Method launchTorpedo* pada *Modul BotService.java* yang merupakan implementasi keseluruhan secara mendetail tentang mekanisme penyerangan menggunakan *torpedo* miliki *bot* kami.



Gambar 4.10. *Bot Melepaskan Teleporter pada Attack Phase*
Sumber: Dokumen Penulis



Gambar 4.11. *Bot Berhasil Memakan Musuh dengan Teleporter pada Attack Phase*
Sumber: Dokumen Penulis

Strategi penyerangan kedua adalah dengan menggunakan *teleporter*. Secara umum, mekanisme penyerangan yang dilakukan oleh *bot* menggunakan *teleporter* sama dengan penyerangan menggunakan torpedo. Bot akan melakukan mekanisme sorting berdasarkan jarak pada seluruh objek *Player* yang ada sebagai tahapan awal menentukan musuh yang menjadi incaran. Selanjutnya, *bot* akan merubah arah *heading*-nya menuju musuh tersebut. *Bot* tidak akan langsung berpindah menuju posisi *teleporter* pada beberapa *tick* kemudian, melainkan akan mempertimbangkan kondisi berdasarkan *Method checkTeleportSerang* pada *Modul BotService.java*, yaitu: jika *teleporter* kita memiliki jarak kurang dari sama dengan ukuran musuh ditambah 5 maka bot akan berpindah ke posisi tersebut, tetapi jika tidak terpenuhi maka *teleporter* akan dibiarkan saja. Namun, pelepasan *teleporter* juga terlebih dahulu akan mempetimbangkan perbandingan ukuran *bot* kita dan *bot* musuh.

Berdasarkan strategi *Defense Phase*, kami sudah berhasil mengaktifkan *shield* saat diserang dengan torpedo oleh musuh, fungsi ini dapat dilihat pada pertandingan 2. Selain mengaktifkan *shield*, kami juga dapat memanfaatkan *afterburner* untuk melarikan diri dari musuh dan torpedo. Hal tersebut sesuai dengan gambar hasil pertandingan di bawah ini.

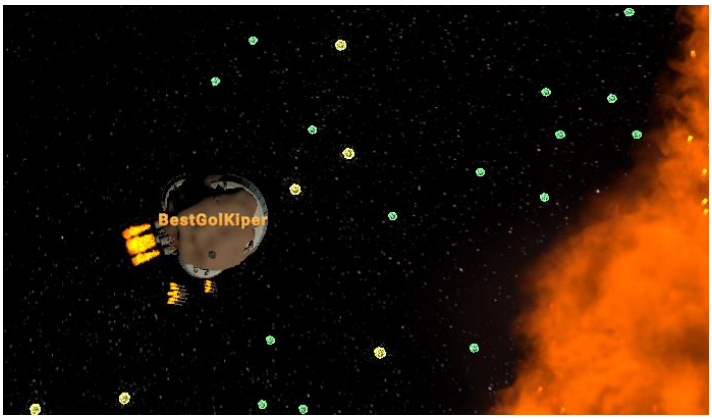
Gambar 4.12. *Bot Menggunakan Shield pada Defence Phase*
Sumber: Dokumen Penulis

Pada strategi bertahan ini, kami memanfaatkan beberapa mekanisme, antara lain: mekanisme run, *activateAfterBurner*, serta penggunaan *shield*. Mekanisme *run* terdapat pada *Method Run* yang ada dalam *Modul GreedyCommand.java*. *Bot* akan mempertimbangkan *heading* terbaik untuk mencari tempat aman dari musuh berdasarkan *heading* dari musuh serta parameter bertipe *boolean* bernama *switchDirection*. Selain itu, terdapat pula mekanisme bertahan menggunakan *AfterBurner* apabila bahaya yang mengancam

dirasa tidak biasa. Namun, mekanisme ini akan terlebih dahulu mempertimbangkan ketersediaan *afterBurner*, jika memungkinkan maka *bot* secara otomatis akan menghidupkan *AfterBurner* untuk mencari posisi terbaik dalam menghindari musuh. Apabila tidak memungkinkan maka *bot* akan kembali menggunakan mekanisme *run*. Kedua mekanisme tersebut juga akan dimanfaatkan untuk menghindari *boundary* dan *gasCloud*.

Mekanisme lainnya adalah penggunaan *shield* untuk menghindari serangan *torpedo* milik pemain musuh. Dalam mekanisme ini, kami akan mendeteksi *torpedo* yang mengarah pada kami dan akan langsung mengaktifkan *shield* sebagai bentuk pertahanan. Selanjutnya, *bot* akan menembakkan *torpedo* ke arah musuh yang menembakkan *torpedo*-nya. Dengan demikian, *bot* musuh tidak dapat dengan mudah melakukan penyerangan secara beruntun karena hal itu justru akan merugikan-nya.

Untuk strategi *Default Phase*, strategi *greedy* yang kami pilih dapat dilihat pada tiap pertandingan keefektifannya. Pada tiap pertandingan, kapal kami akan mendekati makanan terdekat apabila tidak ada objek yang mengancam serta tidak ada musuh yang dapat diserang.



Gambar 4.13. *Bot Mencari Makanan Terdekat pada Default Phase*
Sumber: Dokumen Penulis

BAB V
Kesimpulan Saran dan Refleksi

5.1 Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma *greedy* untuk membuat *bot* permainan *Galaxio* yang bisa mencapai tujuan objektif yaitu menjadi kapal terakhir yang dapat bertahan atau menjadi kapal terbesar yang masih bertahan hingga *tick* terakhir. Dari hasil yang didapatkan, dapat dilihat bahwa penggunakan strategi *greedy* cukup optimal dalam kasus ini, dikarenakan pada permainan *Galaxio* pemilihan paling tamak pada tiap langkahnya kemungkinan besar merupakan kemungkinan terbaik untuk permainan secara umum. Untuk mengilustrasikan, misal pada suatu langkah kita melakukan *greedy* dengan mencoba menjadi kapal sebesar mungkin pada saat itu, langkah yang kita ambil ini sejalan dengan permainan untuk menjadi kapal terbesar sehingga merupakan salah satu pilihan terbaik yang bisa bot lakukan saat itu.

Namun, tentu strategi *greedy* ini juga perlu didampingi dengan teknik heuristik, agar strategi *greedy* yang digunakan lebih optimal dan cocok untuk berbagai kondisi. Secara umum, *bot* yang kami buat berhasil membuktikan bahwa strategi *greedy* cukup baik digunakan dalam pembuatan *bot*.

5.2 Saran

Berikut ini beberapa saran yang bisa kita ajukan untuk selanjutnya:

- Pembagian tugas sebaiknya dilakukan lebih cepat agar *workload* masing-masing anggota lebih jelas sejak awal sehingga pengerjaannya lebih terstruktur.

Referensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
https://github.com/EntelectChallenge/2021-Galaxio
https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2
https://github.com/EntelectChallenge/2021-Galaxio/tree/develop/game-engine
https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md

Pranala Terkait

Link Repository:
https://github.com/rifqifarhansyah/Tubes1_EnteGoalKiperPalingBaikDiDunia
Link Video :
https://youtu.be/4KGqGQTKT2A