LAPORAN TUGAS BESAR

Penerapan Algoritma Greedy dalam Aplikasi Permainan “Galaxio”

IF2211 Strategi Algoritma

Kelompok Yasin_Bot

Dosen : Dr. Ir. Rinaldi, M.T.

Anggota Kelompok :

Muhammad Hanan	**(13521041)**
Alex Sander	**(13521061)**
Made Debby Almadea Putri	**(13521153)**

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung

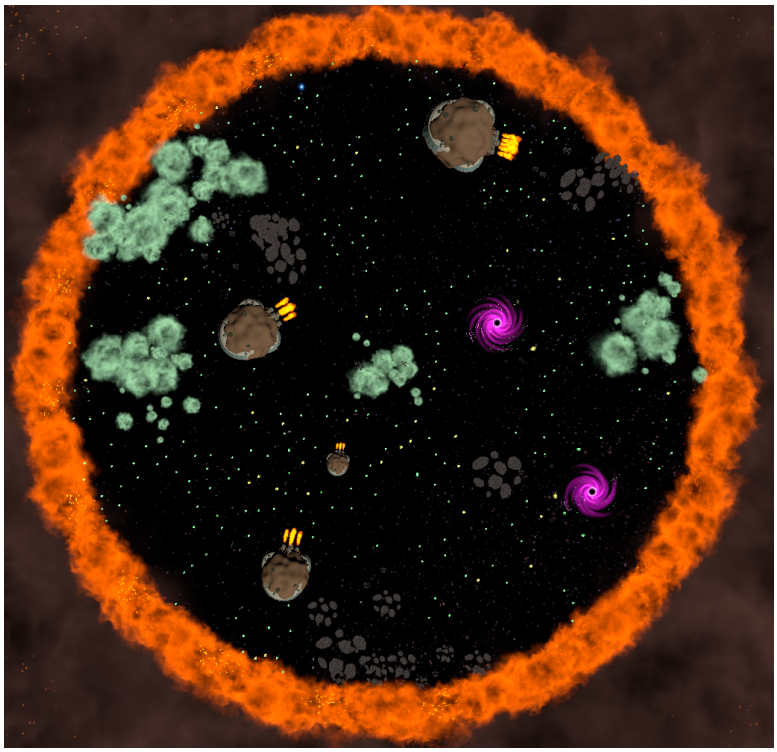
2023

Daftar Isi

BAB I

DESKRIPSI TUGAS

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



Gambar 1. Ilustrasi Permainan *Galaxio*

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Galaxio*. Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack.* Spesifikasi permainan yang digunakan pada

tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Galaxio* pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan berbentuk kartesius yang memiliki arah positif dan negatif. Peta hanya menangani angka bulat. Kapal hanya bisa berada di *integer* x,y yang ada di peta. Pusat peta adalah 0,0 dan ujung dari peta merupakan radius. Jumlah ronde maximum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu *Players, Food, Wormholes, Gas Clouds, Asteroid Fields*. Ukuran peta akan mengecil seiring batasan peta mengecil.
2. Kecepatan kapal dilambangkan dengan x. Kecepatan kapal akan dimulai dengan kecepatan 20 dan berkurang setiap ukuran kapal bertambah. Ukuran (radius) kapal akan dimulai dengan ukuran 10. *Heading* dari kapal dapat bergerak antar 0 hingga 359 derajat. Efek *afterburner* akan meningkatkan kecepatan kapal dengan faktor 2, tetapi mengecilkan ukuran kapal sebanyak 1 setiap tick. Kemudian kapal akan menerima 1 salvo charge setiap 10 tick. Setiap kapal hanya dapat menampung 5 salvo charge. Penembakan slavo torpedo (ukuran 10) mengurangkan ukuran kapal sebanyak 5.
3. Setiap objek pada lintasan punya koordinat x,y dan radius yang mendefinisikan ukuran dan bentuknya. *Food* akan disebarkan pada peta dengan ukuran 3 dan dapat dikonsumsikan oleh kapal *player*. Apabila *player* mengkonsumsi *Food,* maka *Player* akan bertambah ukuran yang sama dengan *Food. Food* memiliki peluang untuk berubah menjadi *Super Food.* Apabila *Super Food* dikonsumsi maka setiap makan *Food,* efeknya akan 2 kali dari *Food* yang dikonsumsi. Efek dari *Super Food* bertahan selama 5 tick.
4. Wormhole ada secara berpasangan dan memperbolehkan kapal dari *player* untuk memasukinya dan keluar di pasangan satu lagi. *Wormhole akan* bertambah besar setiap tick game hingga ukuran maximum. Ketika *Wormhole* dilewati, maka *wormhole* akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat *wormhole* lebih besar dari kapal *player.*
5. *Gas Clouds akan tersebar pada peta. Kapal dapat melewati gas cloud. Setiap kapal bertabrakan dengan gas cloud, ukuran dari kapal akan mengecil 1 setiap tick game. Saat kapal tidak lagi bertabrakan dengan gas cloud, maka efek pengurangan akan hilang.*
6. *Torpedo Salvo akan muncul pada peta yang berasal dari kapal lain. Torpedo Salvo berjalan dalam lintasan lurus dan dapat menghancurkan semua objek yang berada pada lintasannya. Torpedo Salvo dapat mengurangi ukuran kapal yang ditabraknya. Torpedo Salvo* akan

mengecil apabila bertabrakan dengan objek lain sebanyak ukuran yang dimiliki dari objek yang ditabraknya.

7. *Supernova* merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. *Player* yang menembakannya dapat meledakannya dan memberi *damage* ke *player* yang berada dalam zona. Area ledakan akan berubah menjadi *gas cloud.*
8. *Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.*
9. Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan dikonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.
10. Terdapat beberapa *command* yang dapat dilakukan oleh *player.* Setiap tick, *player* hanya dapat memberikan satu *command.* Berikut jenis-jenis dari *command* yang ada dalam permainan:
    a. FORWARD
    b. STOP
    c. STARTAFTERBURNER
    d. STOPAFTERBURNER
    e. FIRETORPEDOES
    f. FIRESUPERNOVA
    g. DETONATESUPERNOVA
    h. FIRETELEPORTER
    i. TELEPORT
    j. ACTIVATESHIELD
11. Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus *tie breaking* (semua kapal mati). Jika mengonsumsi kapal *player* lain, maka score bertambah 10, jika mengonsumsi *food* atau melewati wormhole, maka score

bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila *tie breaker* maka pemenang adalah kapal dengan score tertinggi.

BAB II

LANDASAN TEORI

2.1. Gambaran Algoritma *Greedy* Secara Umum

Algoritma greedy adalah salah satu algoritma pemecahan masalah atau persoalan langkah per langkah dimana algoritma akan mengambil pilihan terbaik yang dapat diperoleh pada momen tersebut dan berharap bahwa solusi yang didapatkan pada akhir proses merupakan solusi yang optimum. Algoritma ini tidak pasti akan menghasilkan solusi paling optimum namun lebih cepat jika dibandingkan dengan *exhaustive search*. Oleh karena itu, algoritma ini sering digunakan apabila solusi yang diinginkan tidak harus terbaik.

Algoritma *greedy* meliputi beberapa elemen yang perlu didefinisikan dalam membentuk algoritma *greedy*, yaitu:

1. Himpunan kandidat (C) berisi kandidat yang akan dipilih pada setiap langkah. Misalnya simpul/sisi di dalam graf, job, task, koin, benda, karakter, dan sebagainya
2. Himpunan solusi (S) berisi kandidat yang sudah dipilih
3. Fungsi solusi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi (*selection function*) yaitu memilih kandidat berdasarkan strategi *greedy* tertentu dan bersifat heuristik
5. Fungsi kelayakan (*feasible*) yaitu memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi
6. Fungsi obyektif yaitu obyektif dari algoritma *greedy*, memaksimumkan atau meminimumkan suatu hal dalam permasalahan

Oleh karena itu, dengan elemen-elemen di atas, maka dapat disimpulkan algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, *S*, dari himpunan kandidat, *C*, yang dalam hal ini, *S* harus memenuhi beberapa kriteria yang ditentukan, yaitu *S* menyatakan suatu solusi dan *S* di optimisasi oleh fungsi obyektif.

Skema umum algoritma *greedy* dapat terlihat dari potongan *pseudocode* berikut ini

```
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
```

Dari *pseudocode* di atas terlihat di setiap akhir iterasi terbentuk solusi berupa optimum lokal dan di akhir while-do diperoleh optimum global jika ada.

Algoritma greedy yang digunakan dapat disusun sedemikian rupa sehingga solusi yang dihasilkan lebih optimum. Salah satu caranya adalah dengan metode *knapsack*, dimana algoritma greedy dilakukan pada sasaran berdasarkan kategori-kategori tertentu. Kemudian dibandingkan solusi hasil algoritma greedy berdasarkan kategori dengan kategori lainnya dan menghasilkan solusi paling optimal diantara solusi yang diberikan. Namun, karena algoritma tetap tidak beroperasi secara menyeluruh pada semua solusi yang ada, solusi yang diberikan tidak pasti paling optimum.

Oleh karena itu, algoritma *greedy* belum sepenuhnya dapat menghasilkan solusi yang paling optimum. Akan tetapi, algoritma *greedy* selalu menghasilkan solusi *sub-optimal* atau menghampiri solusi yang optimal.

2.2. Cara Kerja Bot Permainan Galaxio

Bot permainan Galaxio secara garis besar bekerja dengan mengirimkan aksi dan *heading* (arah) yang akan dilakukan bot pada setiap tick permainan. Aksi yang dapat dilakukan oleh bot permainan seperti yang disebutkan pada Deskripsi Tugas dari laporan ini, sedangkan heading merupakan arah dari bot. Heading merupakan sebuah integer dari 0 hingga 360 yang menunjukkan arah berlawanan jarum jam. Beberapa aksi tidak memerlukan heading. Pada setiap

tick permainan, bot akan mendapatkan state baru dari game engine yang disimpan di dalam objek game state. Adapun data yang diperoleh dari game state tersebut antara lain

1. World data
    - *currentTick* : tick permainan saat ini
    - *radius* : radius dari map/world pada *currentTick*
    - *centerPoint* : posisi titik tengah map/world pada bidang kartesian
2. Game objects dan Player objects
    - *Id* : id dari objek/player
    - *size* : radius dari objek/player
    - *speed* : kecepatan dari objek/player
    - *currentHeading* : arah dari objek/player
    - *position* : posisi dari objek/player pada bidang kartesian
    - *gameObjectType* : tipe dari objek/player tersebut. Tipe yang ada dalam permainan ini antara lain
        a. PLAYER
        b. FOOD
        c. WORMHOLE
        d. GASCLOUD
        e. ASTEROIDFIELD
        f. TORPEDOSALVO
        g. SUPERFOOD
        h. SUPERNOVAPICKUP
        i. SUPERNOVABOMB
        j. TELEPORTER
        k. SHIELD
    - *effects* : kode hash dari efek yang sedang mempengaruhi suatu objek/pemain. Kode berupa jumlah dari setiap kode efek yang berpengaruh. Efek yang ada beserta kodenya adalah sebagai berikut

Kode	1	2	4	8	16
Efek	Afterburner	Asteroidfield	Gascloud	Superfood	Shield

- *torpedoSalvoCount* : jumlah torpedo salvo yang dimiliki oleh player
- *teleporterCount* : jumlah teleporter yang dimiliki oleh player
- *supernovaAvailable* : apakah player memiliki supernova bomb atau tidak
- *shieldCount* : jumlah shield yang dimiliki oleh player

Seluruh data di atas dapat digunakan oleh bot dalam melakukan analisis untuk menentukan aksi pada tick selanjutnya. Aksi dan heading yang dikirim oleh bot harus sesuai dengan ketentuan. Jika tidak, maka bot akan terus mempertahankan state sebelumnya sehingga menyebabkan hal yang di luar ekspektasi. Sebuah bot dikatakan sebagai pemenang dari permainan Galaxio jika bot tersebut menjadi kapal terakhir untuk hidup.

2.3. Implementasi Algoritma *Greedy* ke Dalam Bot Galaxio

Salah satu algoritma yang dapat diterapkan dalam membangun bot galaxio adalah algoritma *greedy* dengan menggunakan teknik heuristik dalam menentukan langkah optimal selanjutnya. Algoritma *greedy* dapat diterapkan karena pada dasarnya permainan galaxio merupakan permasalahan optimasi dengan fungsi objektif menjadi player terakhir yang bertahan hidup. Strategi *greedy* yang digunakan dalam bot galaxio tidak hanya terpaku dengan satu strategi tetapi *multi-strategi* yang ditentukan berdasarkan data state dari game yang diterima dari setiap tick. Dalam implementasinya, sesuai dengan definisi dari algoritma *greedy*, algoritma akan mengkalkulasi solusi (aksi dan heading) optimal secara lokal dan berharap solusi tersebut menjadi solusi optimal secara global.

2.4. Cara Kerja *Game Engine* Permainan Galaxio

Game Engine merupakan *software framework* yang didesain untuk mengembangkan *video games* dan secara umum meliputi berbagai *library* atau program yang mendukung permainan. *Game Engine* permainan galaxio dibuat dengan menggunakan bahasa C# dan dapat diakses melalui *public repository* github yang di-*publish* oleh Entellect. Untuk dapat menjalankan *game engine* dengan baik, maka terdapat beberapa *prerequisite* yang perlu dipenuhi antara lain: .NET v3.1 dan v5.0 serta JDK versi 11 jika menggunakan bahasa java. Dalam repository tersebut juga disediakan *starter-pack* yang dapat digunakan sebagai awalan dalam membangun bot galaxio. Starter pack terdiri dari beberapa komponen yang digunakan untuk menjalankan game ini:

1. Engine, komponen yang mengimplementasikan *logic* serta *rules* dari game
2. Runner, komponen yang menghubungkan setiap bot dengan engine sehingga dapat melaksanakan *match*
3. Logger, komponen yang mencatat *log* dari permainan sehingga dapat mengetahui hasil dari permainan serta menjadi input dari *visualizer*

Untuk menjalankan permainan secara lokal, terdapat beberapa langkah yang perlu dilakukan:

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan dalam *match* pada *file* "appsettings.json" yang ada pada *folder* "runner-publish" dan "engine-publish"
2. Buka terminal baru pada *folder* "runner-publish" lalu jalankan *command* "dotnet GameRunner.dll"
3. Buka terminal baru pada *folder* "engine-publish" lalu jalankan *command* "dotnet Engine.dll"
4. Buka terminal baru pada *folder* "logger-publish" lalu jalankan *command* "dotnet Logger.dll"
5. Jalankan seluruh bot yang ingin dimainkan
6. Setelah permainan selesai, *log* permainan akan tersimpan dalam *file* "GameStateLog_{TimeStamp} pada folder "logger-publish". Kedua file terdiri dari file hasil akhir permainan dan seluruh log dari permainan.

Starter pack juga telah menyediakan *script* untuk LINUX dan MAC OS yang dapat menjalankan seluruh perintah dari nomor 2 hingga 6. *Script* tersebut dapat dijalankan dengan command "sh run.sh" tanpa tanda kutip.

BAB III

APLIKASI STRATEGI GREEDY

3.1. *Mapping* Persoalan *Galaxio* Menjadi Elemen-elemen Algoritma *Greedy*

3.1.1. *Mapping* Persoalan *Galaxio* pada Persoalan Secara Umum

Dalam permainan *Galaxio*, setiap bot memiliki tujuan untuk menjadi bot terakhir yang bertahan hidup. Terdapat banyak cara untuk mencapai tujuan tersebut yaitu seperti menjauhi pemain lain yang lebih besar atau dengan mengambil makanan terdekat sebanyak mungkin dan lain sebagainya.

Nama Elemen / Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Seluruh kemungkinan aksi atau *command* dan pasangan. Aksi terdiri dari FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER, FIRETORPEDOES, FIRESUPERNOVA, DETONATESUPERNOVA, FIRETELEPORT, TELEPORT, dan ACTIVATESHIELD. Sedangkan *heading* merupakan himpunan bilangan bulat non-negatif dari 0 hingga 360.
Himpunan Solusi	Seluruh kemungkinan pasangan dari himpunan kandidat aksi dan himpunan kandidat *heading* yang membuat bot bertahan hingga akhir permainan.
Fungsi Solusi	Melakukan pengecekan apakah pasangan aksi dan *heading* dapat membawa bot bertahan hidup hingga akhir permainan.
Fungsi Seleksi	Memilih aksi dan heading berdasarkan *game state* yang diberikan pada tick saat itu. Pemilihan aksi dan heading menggunakan teknik heuristik dengan memperhatikan urutan prioritas dari aksi yang harus dilakukan.
Fungsi Kelayakan	Memeriksa apakah pasangan aksi dan heading merupakan kombinasi yang valid. Aksi serta heading yang valid adalah aksi dan

	heading yang ada di himpunan kandidat serta aksi yang hanya dapat dilakukan jika memenuhi syarat seperti ACTIVATESHIELD hanya bisa dilakukan jika memiliki shield.
Fungsi Obyektif	Mencari pasangan aksi dan heading yang dapat memaksimumkan waktu bot bertahan hidup.

3.1.2. *Mapping* Persoalan *Galaxio* pada Persoalan Teleport

Teleporter yang sudah ditembakkan tidak otomatis membuat bot teleport ke tempat yang diinginkan sehingga bot perlu melakukan teleport di tick berikutnya.

Nama Elemen / Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Aksi TELEPORT atau tidak melakukan aksi tersebut. Heading pada sub-persoalan ini tidak berpengaruh.
Himpunan Solusi	Aksi TELEPORT atau tidak melakukan aksi tersebut.
Fungsi Solusi	Melakukan pengecekan apakah teleporter yang kita tembak dekat dengan player lain yang memiliki *size* lebih kecil dari bot kita dan tidak berada di dekat objek berbahaya atau player lain dengan *size* lebih besar dari bot kita.
Fungsi Seleksi	Memilih aksi berdasarkan *game state* yang diberikan pada tick saat itu dengan pendekatan heuristik yaitu memilih aksi TELEPORT jika memenuhi fungsi solusi dan tidak melakukan aksi tersebut jika tidak memenuhi.
Fungsi Kelayakan	Memeriksa apakah aksi tersebut sesuai dengan aturan permainan yaitu aksi TELEPORT hanya dilakukan jika kita sudah menembakkan teleporter dan teleporter tersebut masih ada di dalam

	map.
Fungsi Obyektif	Melakukan teleportasi untuk memakan/menyerang player lain sehingga menghasilkan *damage* maksimum.

3.1.3. *Mapping* Persoalan *Galaxio* pada Persoalan Menembak Teleporter

Teleporter merupakan salah satu tools yang dapat membuat bot menjadi lebih kuat jika bot teleport ke tempat lawan yang memiliki *size* lebih kecil. Selain bisa menambah size tanpa harus mengejar lawan dengan manual, bot juga dapat langsung mengeliminasi lawan sehingga lawan dari bot kita berkurang.

Nama Elemen / Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Pasangan aksi FIRETELEPORTER dengan heading dari 0 sampai 360.
Himpunan Solusi	Pasangan aksi FIRETELEPORTER dengan heading dari 0 sampai 360 yang menunjukkan heading ke player lain.
Fungsi Solusi	Melakukan pengecekan apakah heading tersebut menuju ke player lain dengan *size* yang lebih kecil dari kita jika *size* kita berkurang sebanyak 40, yaitu *cost* dari menembakkan teleport ditambah suatu angka pengaman.
Fungsi Seleksi	Memilih heading ke player lain dengan jarak terdekat serta player lain tidak berada di dekat objek yang berbahaya seperti gascloud dan player lain yang lebih besar dari bot kita.
Fungsi Kelayakan	Memeriksa apakah heading merupakan heading yang valid yaitu 0 sampai 360 dan aksi yang valid yaitu hanya FIRETELEPORT saja.
Fungsi Obyektif	Mencari heading menuju player lain yang memiliki *size* lebih kecil dari kita sehingga memberikan *damage* maksimum saat kita

	berhasil teleport.

3.1.4. *Mapping* Persoalan *Galaxio* pada Persoalan Meledakkan Supernova Bomb

Sama seperti teleporter, supernova bomb yang telah ditembak harus diledakkan agar dapat memberikan *damage* ke player di sekitarnya

Nama Elemen / Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Aksi DENOTATESUPERNOVA atau tidak melakukan aksi tersebut. Heading pada sub-persoalan ini tidak berpengaruh.
Himpunan Solusi	Aksi DENOTATESUPERNOVA atau tidak melakukan aksi tersebut.
Fungsi Solusi	Melakukan pengecekan apakah supernova bomb yang kita tembak dekat dengan player lain.
Fungsi Seleksi	Memilih aksi berdasarkan *game state* yang diberikan pada tick saat itu dengan pendekatan heuristik yaitu memilih aksi DENOTATESUPERNOVA jika memenuhi fungsi solusi dan tidak melakukan aksi tersebut jika tidak memenuhi.
Fungsi Kelayakan	Memeriksa apakah aksi tersebut sesuai dengan aturan permainan yaitu aksi DENOTATESUPERNOVA hanya dilakukan jika kita sudah menembakkan supernova bomb dan supernova bomb tersebut masih ada di dalam map.
Fungsi Obyektif	Meledakkan supernova bomb untuk menyerang player lain sehingga menghasilkan *damage* maksimum.

3.1.5. *Mapping* Persoalan *Galaxio* pada Persoalan Menembak Supernova Bomb

Supernova Bomb merupakan *tools* yang dapat menghasilkan *damage* yang besar bagi pemain yang ada di sekitarnya jika diledakkan. Sama seperti teleport, supernova

bomb sebelum diledakkan harus ditembak terlebih dahulu ke suatu arah dan akan berjalan lurus ke arah yang sama hingga diledakkan atau keluar dari map

Nama Elemen / Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Pasangan aksi FIRESUPERNOVA dengan heading dari 0 sampai 360.
Himpunan Solusi	Pasangan aksi FIRESUPERNOVA dengan heading dari 0 sampai 360 yang menunjukkan heading ke player lain.
Fungsi Solusi	Melakukan pengecekan apakah heading tersebut menuju ke player lain.
Fungsi Seleksi	Memilih heading ke player lain yang dapat menghasilkan *damage* terbesar jika supernova diledakkan.
Fungsi Kelayakan	Memeriksa apakah heading merupakan heading yang valid yaitu 0 sampai 360 dan kita memiliki supernova pickup sehingga bisa menggunakan aksi FIRESUPERNOVA.
Fungsi Obyektif	Mencari heading menuju player lain untuk menyerang player tersebut sehingga menghasilkan *damage* maksimum.

3.1.6. *Mapping* Persoalan *Galaxio* pada Persoalan Bertahan dari Player Lain

Sub-persoalan bertahan dari player lain diperlukan karena player lain dapat menargetkan kita sebagai *bot* yang akan dieliminasi dengan dikejar. Tujuan dari sub-persoalan ini adalah untuk bertahan dari player lain dan dapat dicapai dengan bergerak menjauhi player lain, menembakkan torpedo ke player lain, atau dengan bertahan menggunakan shield

Nama Elemen / Komponen	Definisi Elemen/Komponen

Himpunan Kandidat	Pasangan aksi dan heading dengan aksi yang berlaku adalah FORWARD dan FIRETORPEDOES. Heading yang berlaku adalah bilangan bulat dari 0 sampai dengan 360.
Himpunan Solusi	Pasangan aksi dan heading dengan aksi yang berlaku adalah FORWARD dan FIRETORPEDOES. Heading adalah arah ke player lain dalam derajat yang berpotensi berbahaya yaitu memiliki jarak yang dekat dengan kita yang merupakan bilangan bulat dari 0 sampai dengan 360 serta memiliki *size* yang lebih besar dari kita.
Fungsi Solusi	Melakukan pengecekan apakah pasangan aksi dan heading tersebut dapat membuat *bot* kita menghindari bahaya dari *bot* lawan.
Fungsi Seleksi	Memilih aksi dan heading berdasarkan *game state* yang diberikan pada *tick* saat ini. Pemilihan aksi dan heading bersifat heuristik dengan memperhatikan urutan prioritas.
Fungsi Kelayakan	Memeriksa apakah aksi merupakan aksi yang valid yaitu FORWARD dan FIRETORPEDOES dengan FIRETORPEDOES hanya dapat dilakukan jika *bot* memiliki torpedo salvo. Memeriksa apakah heading merupakan heading yang valid yaitu bilangan bulat dari 0 sampai dengan 360.
Fungsi Obyektif	Mencari pasangan aksi dan heading yang dapat membuat *bot* kita menghindari bahaya *bot* lawan yang lebih besar dalam jangka waktu seminimum mungkin.

3.1.7. *Mapping* Persoalan *Galaxio* pada Persoalan Menghindari Objek Berbahaya dan Batas dunia

Tujuan dari sub-persoalan ini adalah menghindari objek berbahaya yang dapat mengurangi *size* dari bot dan harus dihindari yaitu *gas cloud*, *torpedo salvo*, serta *teleporter* dalam jangka waktu seminimum mungkin.

Nama Elemen / Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Pasangan aksi dan heading dengan aksi berupa FORWARD, FIRETORPEDOES, atau ACTIVATESHIELD dan heading berupa bilangan bulat dari 0 sampai 360.
Himpunan Solusi	Pasangan aksi dan heading dengan aksi berupa FORWARD, FIRETORPEDOES, atau ACTIVATESHIELD. Jika aksi berupa FORWARD maka *heading* menunjukkan arah yang aman sehingga dapat menjauh dari objek yang berbahaya. Jika aksi berupa FIRETORPEDOES, maka *heading* menunjukkan arah ke objek berbahaya tersebut.
Fungsi Solusi	Memeriksa apakah *heading* menjauhi objek yang berbahaya jika aksi berupa FORWARD dan apakah *heading* menuju objek yang berbahaya jika aksi berupa FIRETORPEDOES.
Fungsi Seleksi	Memilih aksi dan *heading* yang dapat membuat bot kita keluar dari bahaya objek lain atau batas dunia dan tidak menyebabkan *side effect* yang dapat merugikan seperti *size* menjadi kecil karena menembak torpedo salvo atau mengaktifkan *shield*
Fungsi Kelayakan	Memeriksa apakah aksi dan *heading* valid. Aksi yang valid dalam persoalan ini adalah FORWARD, FIRETORPEDOES, dan ACTIVATESHIELD. Jika aksi FIRETORPEDOES, maka bot harus memiliki torpedo salvo. Jika aksi ACTIVATESHIELD, maka bot harus memiliki shield. *Heading* yang valid adalah heading menjauhi objek yang berbahaya jika aksi berupa FORWARD dan *heading* menuju objek yang berbahaya jika aksi berupa FIRETORPEDOES.
Fungsi	Terbebas dari objek yang berbahaya dalam jangka waktu sesingkat

Obyektif	mungkin.

3.1.8. *Mapping* Persoalan *Galaxio* pada Persoalan Menembak Torpedo Salvo untuk Menyerang Pemain Lawan

Sub persoalan selanjutnya adalah persoalan menembak torpedo salvo untuk memberikan *damage* ke pemain yang lain. Torpedo salvo hanya bisa ditembak jika kita memiliki torpedo salvo.

Nama Elemen / Komponen	**Definisi Elemen/Komponen**
Himpunan Kandidat	Pasangan aksi FIRETORPEDOES dengan *heading* berupa bilangan bulat dari 0 sampai 360.
Himpunan Solusi	Pasangan aksi FIRETORPEDOES dengan *heading* dari 0 sampai 360 yang terpilih.
Fungsi Solusi	Memeriksa apakah *heading* menuju pemain lain dan *heading* tidak terhalang oleh objek selain pemain.
Fungsi Seleksi	Memilih *heading* menuju pemain lain yang terdekat.
Fungsi Kelayakan	Memeriksa apakah aksi valid yaitu bot kita memiliki torpedo salvo sehingga bisa melakukan FIRETORPEDOES serta *heading* bilangan bulat dari 0 sampai 360.
Fungsi Obyektif	Menembak torpedo salvo ke pemain lain sehingga menghasilkan *damage* yang maksimum.

3.1.9. *Mapping* Persoalan *Galaxio* pada Persoalan Mengejar Makanan *Food, Super Food*, dan Pemain Lain

Sub persoalan lain dari permainan *Galaxio* adalah pencarian makanan dengan tujuan untuk menambah size dari bot secepat mungkin. Terdapat berbagai cara untuk

mencapai tujuan tersebut antara lain dengan memakan *food* biasa, memakan *super food*, atau memakan player lain

Nama Elemen / Komponen	Definisi Elemen/Komponen
Himpunan Kandidat	Pasangan aksi dan heading dengan aksi FORWARD dan heading berupa bilangan bulat dari 0 sampai 360.
Himpunan Solusi	Pasangan aksi dan heading dengan aksi berupa FORWARD. Heading merupakan elemen himpunan bilangan bulat dari 0 sampai dengan 360 yang menunjukkan arah menuju objek *food*, *super food*, atau *player* lain.
Fungsi Solusi	Melakukan pengecekan apakah pasangan aksi dan heading membuat *bot* bertambah menjadi lebih besar dan tidak membuat *bot* berada di state yang lebih berbahaya seperti bergerak menuju objek yang berbahaya atau batas map.
Fungsi Seleksi	Memilih aksi dan heading berdasarkan *game state* yang diberikan pada tick saat itu secara heuristik yang memperhatikan urutan prioritas dari aksi yang harus dilakukan serta memilih heading ke makanan dengan jarak terpendek.
Fungsi Kelayakan	Memeriksa apakah aksi merupakan aksi yang valid yaitu FORWARD dan heading menuju *food*, *super food*, dan pemain lain.
Fungsi Obyektif	Menambah *size* dari *bot* kita semaksimum mungkin dalam selang waktu seminimum mungkin.

3.2. Eksplorasi Alternatif Solusi Algoritma *Greedy* pada Bot Permainan Galaxio

Setiap solusi algoritma *greedy* untuk menyelesaikan persoalan yang ada pada poin 3.1 dapat mempengaruhi *game state* di tick berikutnya. Seperti contoh penggunaan teleporter dapat

mengurangi *size* dari *bot* kita sebanyak 20. Oleh karena itu, solusi algoritma *greedy* untuk permainan ini disusun dengan pertimbangan akan mempengaruhi *state* untuk tahap selanjutnya.

3.2.1. Mekanisme Melakukan Teleport

Bot mendapatkan teleporter di setiap 100 tick permainan. Jika bot memiliki teleporter maka bot dapat menembakkan teleporter tersebut ke arah dari 0 hingga 360 derajat. Akan tetapi, menembakkan teleporter tidak sama dengan melakukan teleport. Setelah menembakkan teleporter, teleporter akan terus berjalan lurus ke arah yang telah ditentukan hingga bot melakukan teleport atau teleporter keluar batas dunia. Oleh karena itu, arah yang semula terdapat pemain lawan diperlukan strategi penentuan waktu dalam melakukan teleport.

Strategi pertama adalah *greedy by distance*. Dalam strategi ini algoritma akan mencari semua object pemain (*bots*) yang berada di dekat teleporter yang sudah ditembakkan. Dekat disini memiliki arti jarak antara pemain dan teleporter, dihitung dari kulit terluar, kurang dari sama dengan 60 ditambah setengah dari *size* kita untuk menjamin terjadi tabrakan antara kapal kita dan pemain lawan. Untuk memastikan bahwa tabrakan membuat kita mendapatkan *size* lawan, maka dilakukan pengecekan kedua yaitu apakah pemain lawan memiliki *size* yang lebih kecil dari kapal kita.

Strategi kedua adalah *greedy by size*. Dalam strategi ini algoritma akan mencari semua object pemain (*bots*) yang berada di dekat garis lurus yang ditarik dari teleporter ke batas dunia dengan arah yang sama dengan teleporter. Pemain dengan size terbesar akan dijadikan target teleporter pada *tick* tersebut dan teleporter dilakukan jika jarak antara pemain tersebut dan teleporter kurang dari sama dengan 60 ditambah setengah dari *size* kita.

3.2.2. Mekanisme Menembak Teleporter

Teleporter merupakan salah satu *tools* yang dapat ditembakkan oleh pemain. Pada awal permainan, pemain mendapatkan 1 teleporter dan akan mendapatkan 1 teleporter setiap 100 tick. Untuk melakukan teleport, pemain harus menembakkan teleporter terlebih dahulu. Penembakan teleporter memakan 20 *size* pemain. Setelah ditembakkan, teleporter akan berjalan lurus ke arah yang telah ditentukan. Oleh karena itu, perlu strategi untuk menentukan arah teleporter ditembakkan.

Strategi pertama adalah *greedy by distance*. Dalam strategi ini algoritma akan mencari semua object pemain yang memiliki *size* lebih kecil dari kapal kita dikurang 20 (*cost* dari teleporter) dan berada di daerah yang aman (tidak ada ancaman seperti *gas cloud*). Heading dari teleporter merupakan heading antara kapal kita dengan pemain terdekat.

Strategi kedua adalah *greedy by size.* Dalam strategi ini mirip seperti *greedy by distance* hanya saja pemilihan heading teleporter merupakan heading antara kapal kita dengan pemain yang memiliki size terbesar setelah dilakukan *filtering*.

3.2.3. Mekanisme Meledakkan Supernova Bomb

Supernova bomb merupakan *tools* yang dapat ditembak oleh pemain. Seperti pada teleporter, agar supernova bomb dapat memberikan *damage* kepada pemain disekitarnya, maka pemain yang menembakkan supernova bomb harus meledakkan-nya juga di saat tertentu sebelum ia melewati batas dunia. Oleh karena itu, diperlukan strategi untuk menentukan kapan saatnya meledakkan supernova bomb yang sudah ditembak. Strategi tersebut bersifat heuristik dengan penjelasan sebagai berikut.

Strategi dari mekanisme ini adalah *greedy by distance*. Pada strategi ini, asumsi kita telah menembakkan supernova bomb, selama kita masih belum tereliminasi dan supernova bomb masih ada di dalam map, maka setiap mendapatkan *game state* baru dilakukan pengecekan apakah terdapat pemain lawan yang memiliki jarak kurang dari sama dengan 100 dan jarak antara kapal kita dengan supernova bomb lebih dari sama dengan 250. Jika ada maka supernova bomb diledakkan pada saat itu. Jika tidak ada maka kapal kita menunggu *game state* baru sebelum melakukan pengecekan kembali. Jika pada suatu saat supernova bomb tidak menemukan pemain lawan dengan jarak yang memenuhi, tetapi sudah mendekati batas dunia, maka supernova bomb akan diledakkan pada saat itu juga.

3.2.4. Mekanisme Menembak Supernova Bomb

Supernova bomb dapat memberikan *damage* seperti *gas cloud* pada pemain di sekitarnya setelah diledakkan. Sama seperti teleporter, supernova bomb perlu ditembak terlebih dahulu ke suatu arah dan setelah ditembak, supernova bomb akan terus berjalan lurus hingga diledakkan atau keluar batas dunia. Oleh karena itu, diperlukan strategi untuk menentukan arah supernova bomb ditembakkan.

Strategi pertama adalah *greedy by distance*. Pada strategi ini, heading atau arah dari supernova bomb merupakan *heading* antara kapal kita dengan pemain lawan yang memiliki jarak terdekat dengan kapal kita.

Strategi kedua adalah *greedy by size*. Pada strategi ini, heading atau arah dari supernova bomb merupakan *heading* antara kapal kita dengan pemain lawan yang memiliki *size* terbesar.

3.2.5. Mekanisme Mengambil Supernova Pickup

Supernova pickup merupakan *tools* untuk mengambil supernova yang dapat ditembakkan. Supernova pickup hanya *spawn* sekali dalam satu permainan. Strategi dalam pengambilan supernova pickup bersifat heuristik. Dalam strategi ini, selama pemain belum tereliminasi dan selama supernova pickup masih ada di dalam map, maka dilakukan pengecekan apakah dalam radius (kecepatan kapal *player* dikali 4) dihitung dari kulit terluar kapal kita terdapat objek supernova pickup. Jika ada, maka kapal akan mengambil supernova pickup tersebut dengan aksi SUPERNOVAPICKUP. Jika tidak, maka pengecekan akan dilakukan kembali saat mendapatkan *game state* yang baru.

3.2.6. Mekanisme Bertahan dari Pemain Lain

Pada permainan *Galaxio*, pemain dapat menyerang lawan dengan melakukan tabrakan ke lawan yang lebih kecil sehingga pemain yang lebih besar mendapatkan *size* tambahan sebesar *size* lawan yang lebih kecil. Oleh karena itu, diperlukan strategi untuk bertahan dari serangan player lain. Strategi bertahan ini bersifat heuristik dengan memperhatikan urutan prioritas sebagai berikut:

a. Jika pada permainan tersisa 2 pemain (kita dan 1 pemain lawan) serta jarak pemain lawan dan kita berjarak kurang dari sama dengan 80 satuan dan lebih besar dari kita, maka terdapat 2 strategi yang dapat dipilih berdasarkan prioritas
    - Jika kita dapat menyerang pemain lawan dengan torpedo salvo dengan mekanisme yang sama seperti yang dijelaskan pada bagian 3.2.9 poin a, maka kita serang pemain lawan dengan torpedo salvo
    - Jika strategi di atas tidak bisa dilakukan, maka strategi selanjutnya sama dengan strategi yang dijelaskan pada poin c

b. Jika pada permainan terdapat lebih dari 2 pemain serta jarak pemain lawan dan kita berjarak kurang dari sama dengan 160 dan lebih besar dari kita dan kita dapat menyerang pemain lawan dengan torpedo salvo dengan mekanisme yang sama seperti yang dijelaskan pada bagian 3.2.9, maka kita serang pemain lawan dengan torpedo salvo
c. Jika pada permainan terdapat lebih dari 2 pemain serta jarak pemain lawan dan kita berjarak kurang dari sama dengan 160 dan lebih besar dari kita dan kita tidak dapat menyerang pemain lawan dengan torpedo salvo maka terdapat 2 strategi berdasarkan *game state* pada tick tersebut
    - Jika jarak antara kapal kita dengan batas dunia kurang dari sama dengan 20 satuan maka kita menghindar dari pemain lawan dengan melakukan aksi FORWARD dengan heading ke arah yang tidak menuju keluar dunia dan tidak menuju pemain lain yang menuju kita
    - Jika jarak antara kapal kita dengan batas dunia lebih dari 20 satuan maka kita menghindar dari pemain lawan dengan melakukan aksi FORWARD dengan heading ke arah yang berlawanan dari arah pemain lawan

3.2.7. Mekanisme Menghindar dari Objek Berbahaya dan Batas Dunia

Pada permainan *Galaxio*, kapal kita terkena *damage* jika mengenai objek berbahaya seperti *gas cloud* dan *torpedo salvo*. Selain objek yang langsung membuat kita terkena *damage* setelah bertabrakan, terdapat objek *teleporter* yang juga berbahaya karena jika mengarah ke kapal kita kemungkinan player lain dapat teleport dan bertabrakan dengan kita. Terdapat pula objek *supernova bomb* yang berbahaya karena dapat meledak menjadi *gas cloud* di waktu yang tidak dapat ditebak. Selain objek yang secara eksplisit diberikan *state*-nya ke kita, kita juga perlu menghindari batas dunia karena dapat memberikan *damage* jika kita melewati batas dunia. Strategi menghindar dari objek berbahaya dan batas dunia bersifat heuristik dengan prioritas urutan aksi seperti berikut:

1. Jika jarak kita dengan batas dunia kurang dari sama dengan 20 maka kita bergerak dengan heading ke arah titik tengah dunia
2. Jika jarak kita dengan teleporter terdekat kurang dari 400 dan teleporter mengarah ke arah kita ± 60 derajat maka kita bergerak dengan heading tegak lurus dari *heading* object ke kita

3. Jika jarak kita dengan gas cloud terdekat kurang dari sama dengan 80 atau kita terkena efek gas cloud maka kita bergerak dengan *heading* berlawanan arah dari *heading* antara kita dengan *gas cloud*
4. Jika jarak kita dengan torpedo salvo terdekat kurang dari sama dengan 150 dan torpedo tersebut mengarah ke kita maka terdapat 2 strategi yang dapat dilakukan yaitu sebagai berikut
    a. Jika kita memiliki shield dan *size* kita memenuhi *size* minimal untuk menggunakan shield, maka kita gunakan shield dengan aksi ACTIVATESHIELD
    b. Jika kita memiliki torpedo salvo dan *size* kita memenuhi *size* minimal untuk menembak torpedo, maka kita tembak torpedo ke arah torpedo salvo yang menuju kita
    c. Jika kita tidak memiliki torpedo salvo atau *size* kita tidak memenuhi *size* minimal untuk menembak torpedo, maka kita bergerak dengan *heading* tegak lurus dengan arah torpedo salvo tersebut

3.2.8. Mekanisme Menyerang Pemain Lawan dengan Torpedo Salvo

Selain melakukan tabrakan, pemain juga dapat menyerang lawan dengan menembakkan torpedo salvo ke pemain yang lain. Jika torpedo mengenai pemain yang lain, maka *size* pemain yang terkena torpedo salvo akan berkurang sebanyak *size* dari torpedo, sedangkan pemain yang menembakkan torpedo salvo akan bertambah sebanyak *size* dari torpedo. Menyerang pemain lawan memiliki *benefit* yang besar sehingga perlu strategi yang baik. Strategi dalam menyerang pemain lawan bersifat heuristik sebagai berikut:

Jika jarak pemain lawan dengan kita kurang dari sama dengan radius dengan radius berupa 200 satuan jika hanya terdapat 2 pemain (kita dan 1 lawan lain), 400 satuan jika terdapat > 2 pemain dan besar kapal kita memenuhi besar minimum untuk menembakkan torpedo salvo dan pada arah kapal kita dan lawan lain tidak terdapat objek berbahaya seperti *wormhole, asteroid field,* dan *gas cloud* yang dapat membuat torpedo kita terbuang sia-sia. Pemain lawan juga harus memiliki *size* lebih dari 10 karena pemain yang memiliki size kurang dari 10 lebih cepat daripada torpedo salvo, maka terdapat 2 strategi dalam memilih heading dari torpedo salvo

Strategi pertama adalah *greedy by distance*. Pada strategi ini, heading dari torpedo salvo adalah heading antara kapal kita dengan pemain lawan dengan jarak terpendek dari kapal kita

Strategi kedua adalah *greedy by size*. Pada strategi ini, heading dari torpedo adalah heading antara kapal kita dengan pemain lawan terbesar yang memenuhi syarat

3.2.9. Mekanisme Mengejar Makanan *Food*, *Super Food,* dan Pemain Lain

Cara lain untuk menambah *size* dari kapal kita adalah dengan mengejar objek *food* dan *super food* serta mengejar pemain lain lalu bertabrakan. Jika kapal kita memakan objek *food* maka kapal kita bertambah besar sebanyak 3 satuan, sedangkan jika kita memakan *super food* maka kapal kita akan mendapatkan *power up* yang menambah *absorption rate* dari makanan ataupun player yang kita makan. *Absorption rate* menjadi 2 kali lipat, dengan kata lain jika kita memakan makanan yang memiliki *size* 3, maka *size* kita akan bertambah sebanyak 6 satuan. Jika kita menabrak pemain lain yang lebih kecil dari kita, maka *size* kita akan bertambah sebesar *size* dari pemain lain tersebut. Strategi yang digunakan dalam mengejar *food*, *super food*, dan pemain lain bersifat heuristik dengan urutan prioritas sebagai berikut:

1. Jika jarak pemain lawan dengan kita kurang dari sama dengan 8 kali besar kapal kita dan pemain lawan lebih kecil dari kita, maka kapal kita bergerak dengan *heading* ke arah pemain lawan tersebut
2. Jika dalam radius 50 satuan terdapat *super food*, maka kita bergerak menuju *super food*
3. Jika tidak terdapat *super food*, maka kita bergerak menuju *food*

Dalam memilih *super food*, *food*, dan pemain lain yang akan dimakan menggunakan strategi *greedy by distance* yaitu memilih *super food* atau *food* yang memiliki jarak terpendek dari kapal kita.

3.3. Analisis Efisiensi dan Efektivitas Dari Kumpulan Alternatif Solusi

3.3.1. Mekanisme Melakukan Teleport

Pada bagian ini terdapat iterasi list pemain yang ada pada *world* untuk mendapatkan semua pemain yang memenuhi kriteria. Proses ini memiliki kompleksitas O(N).

3.3.2. Mekanisme Menembak Teleporter

Pada bagian ini terdapat iterasi list pemain yang ada pada *world* untuk mendapatkan semua pemain yang memenuhi kriteria untuk ditembak oleh teleporter. Proses ini memiliki

kompleksitas O(N). Setelah mendapatkan semua pemain yang memenuhi kriteria, list kembali dilakukan iterasi untuk mencari pemain dengan jarak terpendek dalam strategi *greedy by distance* dan *size* terbesar dalam strategi *greedy by size*. Proses ini memiliki kompleksitas O(N). Sehingga mekanisme ini memiliki kompleksitas O(N).

3.3.3. Mekanisme Meledakkan Supernova Bomb

Pada bagian ini terdapat iterasi list pemain yang ada pada *world* untuk mendapatkan semua pemain yang memenuhi kriteria. Proses ini memiliki kompleksitas O(N).

3.3.4. Mekanisme Menembak Supernova Bomb

Pada bagian ini terdapat iterasi list pemain yang ada pada *world* untuk mendapatkan semua pemain yang memenuhi kriteria untuk ditembak oleh supernova bomb. Setelah itu list dilakukan iterasi kembali untuk mendapatkan pemain dengan jarak terpendek dalam strategi *greedy by distance* dan *size* terbesar dalam strategi *greedy by size*. Sehingga mekanisme ini memiliki kompleksitas O(N).

3.3.5. Mekanisme Mengambil Supernova Pickup

Pada bagian ini tidak melakukan iterasi list apapun karena supernova pickup hanya *spawn* sekali dalam satu game sehingga hanya mengecek apakah supernova pickup ada di dekat kapal kita. Mekanisme ini memiliki kompleksitas O(1).

3.3.6. Mekanisme Bertahan dari Pemain Lain

Pada bagian ini tidak melakukan iterasi list apapun karena hanya mengecek apakah pemain lain memiliki jarak yang dianggap bahaya dengan kita dan apakah pemain tersebut lebih besar dari kita, sehingga mekanisme ini memiliki kompleksitas O(1).

3.3.7. Mekanisme Menghindar dari Objek Berbahaya dan Batas Dunia

Pada bagian ini juga tidak melakukan iterasi list apapun karena hanya mengecek apakah kita berada di dekat objek yang berbahaya sehingga mekanisme ini memiliki kompleksitas O(1).

3.3.8. Mekanisme Menyerang Pemain Lawan dengan Torpedo Salvo

Pada bagian ini terdapat iterasi list pemain untuk mencari pemain-pemain yang memenuhi kriteria. List tersebut kemudian dilakukan iterasi kembali untuk mencari pemain terdekat dalam strategi *greedy by distance* dan pemain terbesar dalam strategi *greedy by size.* Mekanisme ini memiliki kompleksitas O(N).

3.3.9. Mekanisme Mengejar Makanan *Food*, *Super Food*, dan Pemain Lain

Pada bagian ini terdapat iterasi list objek *food*, *super food*, dan pemain untuk mencari *food*, *super food*, dan pemain yang memenuhi kriteria. List tersebut kemudian dilakukan iterasi kembali untuk mencari objek atau pemain terdekat, sehingga mekanisme ini memiliki kompleksitas O(N).

3.4. Strategi *Greedy* yang Dipilih

Strategi *greedy* yang dipilih adalah gabungan dari strategi-strategi yang dipaparkan pada bagian 3.3. Hal ini karena permainan *Galaxio* cukup kompleks sehingga tidak dapat diselesaikan hanya dengan satu strategi *greedy* saja. Penggabungan strategi-strategi tersebut membentuk sebuah algoritma *greedy* yang bersifat heuristik. Setelah melakukan *trial* dan *error*, penulis mendapatkan urutan prioritas yang akan diterapkan pada algoritma utama dari bot. Urutan prioritas tersebut adalah sebagai berikut:

1. Jika kita sudah menembak supernova bomb dan supernova bomb dekat dengan player lain maka kita ledakan supernova bomb dengan aksi DETONATESUPERNOVA. (Bagian 3.2.4)
2. Jika kita sudah menembak teleporter dan teleporter masih ada di map dan teleporter berada di dekat pemain yang lebih kecil, maka kita teleport dengan aksi TELEPORT. (Bagian 3.2.2)
3. Jika supernova pickup ada di map dan jarak antara kita dan supernova pickup dekat, maka kita mengambil supernova pickup. (Bagian 3.2.6)
4. Jika kita memiliki supernova dan terdapat pemain lain yang memenuhi syarat untuk ditembakkan supernova, maka kita tembak supernova dengan aksi FIRESUPERNOVA dengan *heading* ke pemain tersebut. (Bagian 3.2.5)

5. Jika kita memiliki teleporter dan terdapat pemain lain yang memenuhi syarat untuk dapat dimakan oleh kapal kita setelah dikurangi oleh *cost* dari menembak teleporter maka kita tembak teleporter ke arah pemain tersebut dengan aksi FIRETELEPORT. (Bagian 3.2.3)
6. Jika terdapat pemain lain yang berjarak cukup dekat dengan kita dan pemain tersebut lebih besar dari kita maka kita lakukan mekanisme bertahan dari pemain lain. (Bagian 3.2.7)
7. Jika kapal kita dekat dengan batas dunia atau objek berbahaya seperti teleporter pemain lain, *gas cloud*, *torpedo salvo*, dan *supernova bomb* yang ditembak oleh pemain lain, maka lakukan mekanisme menghindar. (Bagian 3.2.8)
8. Jika kita dapat menembak torpedo salvo dan kapal kita dekat dengan pemain lain dalam radius minimum tertentu, maka kita tembak torpedo salvo ke arah pemain lain tersebut dengan aksi FIRETORPEDOES. (Bagian 3.2.9)
9. Jika kapal kita dekat dengan objek *food, super food*, dan pemain lain maka kita mengejar objek/pemain tersebut. (Bagian 3.2.10)
10. Jika semua hal di atas tidak terpenuhi maka aksi yang dilakukan adalah STOP artinya tidak ada aksi yang dapat dilakukan

Pada bagian 3.3 terdapat beberapa mekanisme yang memiliki lebih dari satu strategi *greedy* sehingga ada beberapa catatan untuk strategi apa yang dipilih untuk beberapa mekanisme, seperti:

1. *Greedy by Distance* dalam Mekanisme Melakukan Teleport

   Strategi ini dipilih karena strategi kedua, *greedy by size* tidak dapat menjamin apakah pemain dengan *size* terbesar masih ada di jalur teleporter berjalan di tick berikutnya. Dengan menerapkan *greedy by distance* maka kapal kita akan melakukan teleport sesaat setelah ditemukan pemain yang berada di dekat teleporter yang sedang berjalan
2. *Greedy by Distance* dalam Mekanisme Menembak Teleporter

   Strategi ini dipilih karena semakin dekat pemain lain dengan kita maka teleporter akan sampai semakin cepat sehingga pemain lawan tereliminasi lebih cepat dan kita bisa bertambah besar lebih cepat, meningkatkan kemungkinan kita untuk menjadi pemain terakhir yang bertahan hidup.
3. *Greedy by Size* dalam Mekanisme Menembak Supernova Bomb

Strategi ini dipilih karena semakin besar pemain maka kecepatan pemain semakin lambat. Pemain yang kecil dapat dengan cepat menghindar dari supernova bomb saat diledakkan sehingga supernova bomb kita akan terbuang sia-sia.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

4.1.a *Pseudocode*

Implementasi algoritma greedy pada program terdapat pada file BotService.java, tersusun dalam method `computeNextPlayerAction`. Berikut *pseudocode* algoritma fungsi dan prosedur yang digunakan untuk menentukan perilaku *bot*.

```
function computeNextPlayerAction(PlayerAction playerAction)

   if playerAction.action is null then
      set playerAction.action to PlayerActions.FORWARD
      set playerAction.heading to random integer between 0 and 259

  if gameState.GameObjects is not empty then
     get bot distance from boundary
     get supernova bomb location
     get activated supernova bomb location
     get list of dangerous objects near player
     get effects on player
    if player has fired teleporter then
      call getFiredTeleporter()
    if player has fired supernova bomb and the bomb is near other
    player then
      set player action to detonate supernova
      return true
    if player has fired teleporter and the teleporter is still
    available then
      set player action to teleport
      return true
    if supernova bomb is near then
      set action to go to supernova bomb location
      return true
    if player has supernova in inventory then
      if player is able to fire superbomb then
```

```
      fire supernova bomb to available target
      return true
   if offense action possible then
    set player action to offense action
    return true
   if there is a bigger player around us then
    if player can fire torpedo salvo at bigger player then
     fire torpedo salvo at bigger player
    else run from bigger player
    return true
   if player is too close to boundary then
    set player action to head to the center of the map
    return true
   if there are enemy teleporter near us then
    set player action to avoid teleporter
    return true
   if there are gas clouds in player's direction then
     set player heading to opposite direction of the gas cloud
     return true
   if there are torpedoes heading to player then
    set player action to avoid torpedoes
    return true
   if there is supernova bomb heading to player then
    set player action to avoid supernova bomb
    return true
   if player is able to shoot torpedoes then
    set player action to shoot torpedoes
    return true
   if there are any vulnerable player nearby then
    set player action to chase the vulnerable player
    return true

   else
    goToFood() //search for food
clear current effect
return true
```

```
procedure moveToCenter()

  get world center position
  create world center object with world center position
  get direction to world center
  while current direction to world center near dangerous object
   increment current direction by 1
  set player action to move to current direction
```

```
function getNearestTeleporter()

   scan objects nearby player for teleporter
   if found teleporter then
     return nearest teleporter
   return null
```

```
procedure getFiredTeleporter()

    scan nearby object for teleporter shot by player
    if found then
      get nearest teleporter shot by player
```

```
function isTeleporterStillAvailable()

    scan nearby teleporter that is shot by us
    if not found then
      set firedTeleporter to null
      set isTeleporter to false
      return false
    else
      get teleporter shot by player
      return true
```

```
function isTeleporterNearSmallerEnemy()

    if player has not shot teleporter then
      return false
    else
      Scan for smaller players
      if found and teleporter currently is near smaller player then
        return true
      return false
```

```
function fireTeleporter()

    if player has fired teleporter then
      return false
    scan for other players that is not near dangerous objects
    if found then
      set nearest enemy that is smaller than player as target
      set player action to shoot teleporter to target's direction
      return true
    return false
```

```
function fireSuperbomb()

     scan for players that are adequately far
     if not found then
       return false
     set player action to shoot supernova bomb to largest found player
     return true
```

```
function isSupernovaNearOtherPlayer()

     scan for shot supernova on the map
     if not found then
```

```
        return false
    scan for players near found supernova bomb
    if not found then
        if supernova bomb is near boundary then
          return true
        return false
    return true
```

```
function fireTorpedoSalvo()

    if player does not have torpedo salvo charge then
        return false
    if only 2 players left in game then
        set minimal size to shoot to TORPEDO_COST * 4
        set area to scan to 200 units
    else
        set minimal size to shoot to TORPEDO_COST * 6
        set area to scan to 400 units
    if player size is smaller than minimal size then
        return false
    scan for players within set area
    if not found then
        return false
    scan for non-player objects that can be shot within the set area
    check for any target that can be directly shot without objects
    that can be shot blocking it
    if not found then
        return false
    set player action to fire torpedoes at the nearest target
    return true
```

```
function isObjectNearDangerousObject(GameObject obj)

    if only 2 players left in game then
        return false
```

```
    scan for objects near obj that is considered dangerous
    if not found and obj distance to boundary is small then
        return true
    return false
```

```
function isHeadingNearDangerousObject(int heading)

    scan for objects that are considered dangerous that is close that
    the player is heading to
    scan for enemies that is bigger than player that the player is
    heading to
    if found any dangerous object or enemies then
        return true
    return false
```

```
function getVulnerableNearPlayer()

    scan for enemies around the player within certain radius
    if not found then
        return null
    if nearest enemy is adequately small then
        return nearest enemy
    return null
```

```
procedure goToFood()

    if there aren't any superfood then
        find for food target
        if not found then
          set player action to stop
        else set player action to move to nearest food target
```

```
function computeFoodTarget_alternate()
```

```
    scan for food object near the player that is not near dangerous
    object

    if not found then
        return false
    if there is more than one food target that is the nearest then
        set player action to move towards food target with lower id
    set player action to move towards nearest food target
```

```
function getSuperfood(List<GameObject> object)

    scan for superfood object near player that is not near dangerous
    object

    if found then
        set player action to move towards nearest superfood
        return true
    return false
```

```
function getGasCloudWithin(List<GameObject> objList)

     scan for gas cloud object
     return list from scan
```

```
function getAsteroidWithin(List<GameObject> objList)

     scan for asteroid object
     return list from scan
```

```
function getObjectsWithin(int radius)

     scan for objects within set radius
```

```
    return list from scan
```

```
function getObjectsWithin(int radius, ObjectTypes type)

    scan for certain type objects within set radius
    return list from scan
```

```
function getPlayersWithin(int radius)

    scan for enemies within set radius
    return list from scan
```

```
function getSupernova()

    scan for supernova bomb pickups on the map
    if not found then
        return null
    return supernova bomb from scan
```

```
function getSupernovaBomb()

    scan for shot supernova bomb on the map
    if not found then
        return null
    return shot supernova bomb from scan
```

```
procedure updateSelfState()

    self-update the player
```

```
function getDistanceBetween(GameObject object1, GameObject object2)

     get x difference between object1 and object2
     get y difference between object1 and object2
     return cartesian distance between object1 and object2
```

```
function getOuterDistanceBetween(GameObject object1, GameObject
object2)

     get distance between object1 and object2
     return distance between - object1.size - object2.size
```

```
function getHeadingBetween(GameObject otherObject)

     compute
     toDegrees(atan2(otherObject.position.y-bot.position.y,otherObject
     .position.x-bot.position.x)) and assign to result

     return (result + 360) % 360
```

```
function getDistanceBoundary(GameObject obj)

     get obj distance from origin point and assign to dfo

     return gameState.world.radius - dfo - obj.size
```

```
function toDegrees(double v)

     return (int) (v*(180/PI))
```

```
function getDistanceBetween(GameObject go)
```

```
    return getDistanceBetween(bot,go)
```

```
function getOppositeDirection(GameObject obj)

    return (180 + getHeadingBetween(obj)) % 360
```

```
function getDangerousNearPlayer()

    scan for enemies within certain radius
    if not found then
        return null
    if largest found enemy is larger than player then
        return largest found enemy
    return null
```

```
procedure runFromAtt(GameObject atkr)

    set player action to move forward
    get player distance from boundary
    if player is near boundary then
        set player action heading to circle inside the boundary
    else set player action heading to getOppositeDirection(atkr)
```

```
function dodgeObj(GameObject obj,int radius)

    set player action to move with heading 90 +
    getOppositeDirection(obj)

    return true
```

```
function dodgeTorpedos(GameObject obj)
```

```
    get torpedo current heading
    if torpedo heading to player from the right and not straight at
    player then

        if player size is adequate and shield is available then
          set player action to activate shield
        else set player action to move to the left
        return true
    else if torpedo heading to player from the left and not straight
    at player then

        if player size is adequate and shield is available then
          set player action to activate shield
        else set player action to move to the right
        return true
    else if torpedo heading straight to player then
        if player size is adequate then
          set player action to shoot torpedo at incoming torpedo
        else set player action to run away
    else return false //torpedo is considered not heading to us
```

```
function isObjHeadingUs(GameObject obj)

    get obj current heading
    if obj heading is heading to us within certain angle range then
        return true
    return false
```

```
function getGasCloudInPath(GameObject obj)

    scan for gas clouds that is within a certain radius
    if found then
        return nearest gas cloud
    return null
```

```
procedure getEffects(int hashCode)

     if hashCode = 0 then do nothing
     else
         if hashCode is odd then
           add AFTERBURNER effect to current effect
           getEffects(hashCode -=1)
         if hashCode >= 16 then
           add SHIELD effect to current effect
           getEffects(hashCode -=16)
         if hashCode >= 8 then
           add SUPERFOOD effect to current effect
           getEffects(hashCode -=8)
         if hashCode >= 4 then
           add GASCLOUD effect to current effect
           getEffects(hashCode -=4)
         if hashCode >=2 then
           add ASTEROIDFIELD effect to current effect
           getEffects(hashCode -=2)
```

```
function isObjInBetween_Alternate(List<GameObject>, GameObject Player)

     i iterate for LG.size times
         get distance between bot and LG[i] and set to a
         get distance between bot and Player and set to b
         get distance between LG[i] and Player and set to c
         if c > b then
           continue
         else
           compute a*sin(arccos((a*a + b*b + c*c)/(2*a*b))) and set to
           d
           if d>LG[i].size then
              return false
     return true
```

4.1.b. Struktur Data

- **Atribut**

Berikut adalah atribut yang digunakan dalam program beserta penjelasan mengenai fungsi dari atribut tersebut.

Nama	Tipe	Deskripsi
bot	private GameObject	Merepresentasikan objek player pada permainan
playerAction	private PlayerAction	Merepresentasikan aksi yang akan dilakukan player
gameState	private GameState	Merepresentasikan *game state* secara keseluruhan
superbomb	private GameObject	Merepresentasikan objek supernova bomb pickups pada permainan
firedTeleporter	private GameObject	Merepresentasikan objek teleporter yang telah ditembak player
isTeleporterFired	private boolean	Menunjukkan status apabila teleporter sudah ditembak
firedSuperbomb	private boolean	Menunjukkan status apabila supernova bomb sudah ditembak
currentEffect	private HashSet<Effects>	Merepresentasikan status effect yang dimiliki oleh player
SAFETY_NUM	final int	Merepresentasikan nilai size tambalan aman saat player ingin melakukan teleport pada teleporter
TELEPORT_COST	final int	Merepresentasikan nilai size yang

		digunakan untuk menembakkan teleporter
SUPERFOOD_RADIUS	final int	Merepresentasikan nilai radius yang akan digunakan untuk mencari objek superfood
TORPEDO_RADIUS	final int	Merepresentasikan nilai radius yang akan digunakan untuk mencari *target* penembakan torpedo
AVOID_TORPEDO_RADIUS	final int	Merepresentasikan nilai radius yang akan digunakan untuk mencari objek torpedo yang mengarah ke player
GASCLOUD_RADIUS	final int	Merepresentasikan nilai jarak minimal untuk menghindari *gas cloud*
SUPERNOVABOMB_RADIUS	final int	Merepresentasikan nilai radius yang akan digunakan untuk mencari *target* penembakkan supernova bomb
TELEPORT_SPEED	final int	Merepresentasikan nilai kecepatan teleporter
TORPEDO_COST	final int	Merepresentasikan nilai size yang digunakan untuk menembakkan torpedo

- **Method**

Berikut tabel yang berisikan fungsi dan prosedur yang digunakan dalam program beserta penjelasan mengenai method tersebut. Perhatikan bahwa semua fungsi dan prosedur yang tertera bersifat *private*.

Nama fungsi/prosedur	Tipe	Deskripsi
getBot()	GameObject	Mengembalikan objek player
setPlayerAction()	void	Mengubah aksi player pada *tick* selanjutnya
getPlayerAction()	PlayerAction	Mengembalikan aksi player yang sedang aktif
computeNextPlayerAction()	boolean	Menentukan aksi player pada *tick* selanjutnya berdasarkan situasi
getGameState()	GameState	Mengembalikan *game state*
setGameState (GameState gameState)	void	Mengubah *game state*
moveToCenter()	void	Mengubah aksi player untuk bergerak ke arah pusat *map*
getNearestTeleporter()	GameObject	Mengembalikan object *teleporter* terdekat
getFiredTeleporter()	void	Mendapatkan *teleporter* yang baru saja ditembak dan disimpan pada firedTeleporter
isTeleporterStillAvailable()	boolean	Mengembalikan boolean berdasarkan adanya *teleporter* yang ditembak oleh *player* dalam *map*
isTeleporterNearSmallerEnemy()	boolean	Mengembalikan boolean berdasarkan adanya pemain musuh di sekitar *teleporter* yang ditembak *player*

fireTeleporter()	boolean	Menembakkan *teleporter* pada musuh
fireSuperbomb()	boolean	Menembakkan *supernova bomb* pada musuh
isSupernovaNearOtherPlayer()	boolean	Mengembalikan boolean berdasarkan adanya pemain musuh di sekitar *supernova bomb* yang ditembak *player*
fireTorpedoSalvo()	boolean	Menembakkan *torpedo salvo* pada musuh
isObjectNearDangerousObject (GameObject *obj*)	boolean	Mengembalikan boolean berdasarkan adanya objek yang dianggap berbahaya di sekitar *obj*
isHeadingNearDangerousObject (int *heading*)	boolean	Mengembalikan boolean berdasarkan adanya objek yang dianggap berbahaya di arah *heading*
getVulnerableNearPlayer()	GameObject	Mengembalikan *player* musuh yang lebih kecil dan terdekat terhadap *player*
goToFood()	void	Method untuk mencari makanan
computeFoodTarget()	boolean	Method untuk mencari makanan berdasarkan jarak dan keamanan lintasan
getSuperfood (List<GameObject>	boolean	Method untuk mencari *superfood* di sekitar *player*

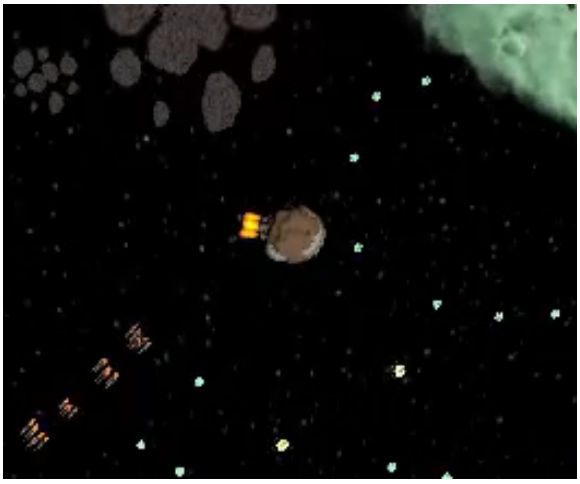
object)		
getGasCloudWithin (List<GameObject> *objList*)	List<GameObject>	Method untuk mencari objek *gas cloud* di sekitar *player*
getAsteroidWithin (List<GameObject> *objList*)	List<GameObject>	Method untuk mencari objek *asteroid field* di sekitar *player*
getObjectsWithin (int *radius*)	List<GameObject>	Mengembalikan list berisi objek di sekitar *player*
getObjectsWithin (int *radius*, ObjectTypes *type*)	List<GameObject>	Mengembalikan list berisi objek di sekitar *player* dengan tipe objek tertentu
getPlayersWithin (int *radius*)	List<GameObject>	Mengembalikan list berisi objek pemain musuh
getSupernova()	GameObject	Mengembalikan objek *supernova pickups* yang berada di *map*
getSupernovaBomb()	List<GameObject>	Mengembalikan *supernova bomb* yang sudah ditembakkan di *map*
getOuterDistanceBetween (GameObject *object1*, GameObject *object2*)	double	Mengembalikan nilai jarak terdekat antara sisi *object1* dengan sisi *objek2*
getHeadingBetween (GameObject *otherObject*)	int	Mengembalikan nilai arah player menuju objek *otherObject*
getDistanceBoundary	int	Mengembalikan nilai jarak antara *obj*

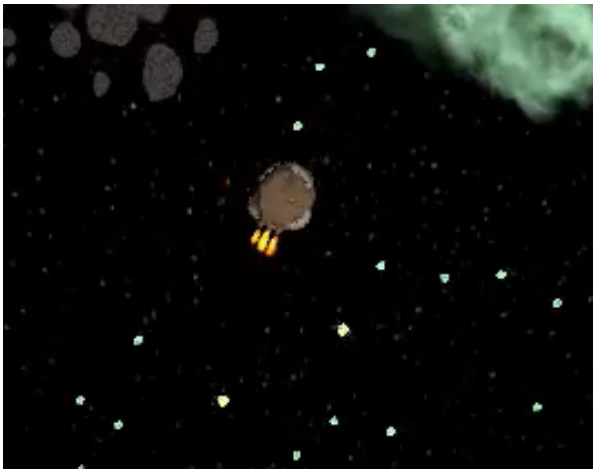
(GameObject *obj*)		dengan *boundary*
toDegrees(double v)	int	Mengkonversi suatu nilai menjadi satuan derajat
getDistanceBetween (GameObject *go*)	double	Mengembalikan jarak antara *player* dengan objek *go*
getOppositeDirection (GameObject *obj*)	int	Mengembalikan arah berlawanan dengan arah *player* menuju objek *obj*
getDangerousNearPlayer( )	GameObject	Mengembalikan pemain lain yang ukurannya lebih besar dari player dan jarak dengan *player* dekat
runFromAtt (GameObject atkr)	void	Method untuk menentukan arah kabur dari pemain musuh
dodgeObj (GameObject obj, int *radius*)	boolean	Method untuk menghindari objek yang mengarah kepada *player*
dodgeTorpedos (GameObject *obj*)	boolean	Method untuk menghindari object *torpedo* yang mengarah kepada *player*
isObjHeadingUs (GameObject *obj*)	boolean	Mengembalikan boolean berdasarkan jika arah *obj* menuju *player*
getGasCloudInPath()	GameObject	Mengembalikan objek *gas cloud* terdekat dengan *player*
getEffects(int *hashCode*)	void	Mengambil efek yang dimiliki oleh *player*
isObjectInBetween(List<	boolean	Mengembalikan boolean berdasarkan

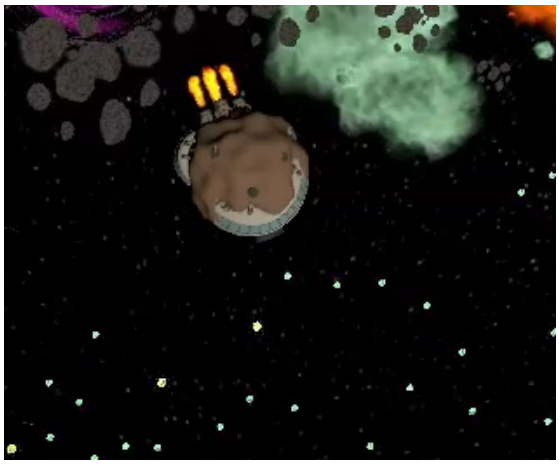
GameObject> LG, GameObject *Player*)		adanya objek di antara *player* dengan objek *Player*

4.2. Pengujian

4.2.1. Mekanisme Pengambilan *Food*

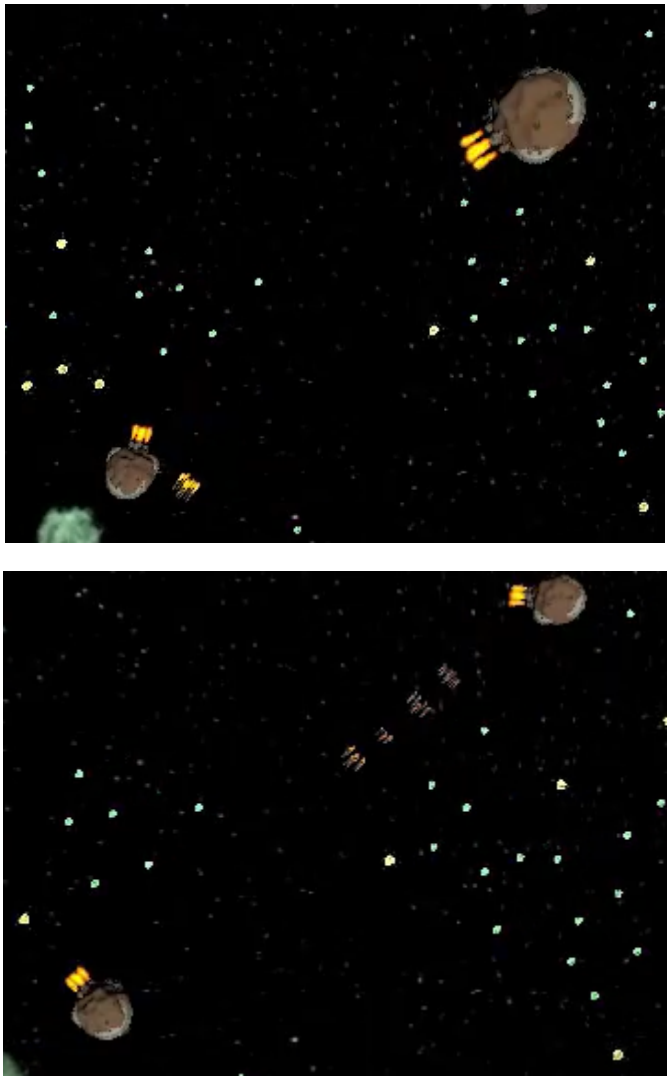






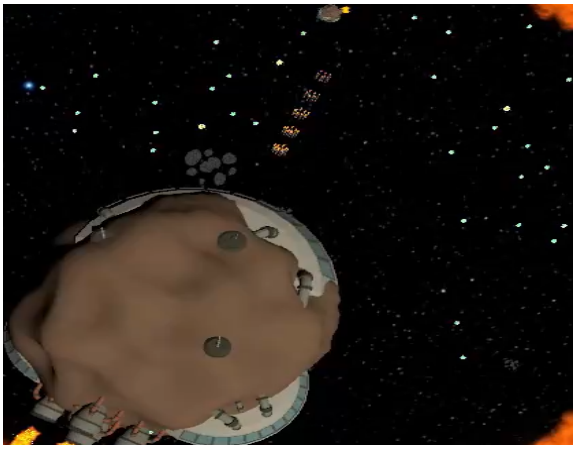
Dapat diamati bahwa dalam mekanisme mencari makanan, program berhasil untuk mencari dan bergerak menuju makanan terdekat dengan *player*.

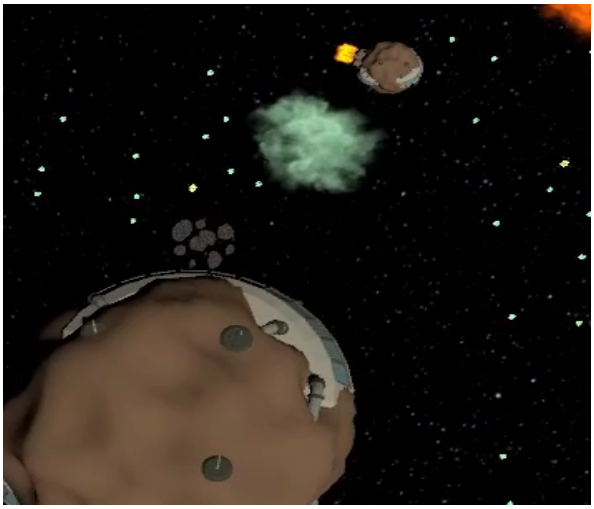
4.2.2 Mekanisme *Torpedo Salvo*



Dapat diamati bahwa mekanisme *torpedo* berhasil bekerja, dengan adanya penembakan *torpedo salvo* kepada musuh yang dekat dengan *player*.

4.2.3 Mekanisme *Supernova Bomb*





Dapat diamati bahwa mekanisme *supernova bomb* berhasil dengan adanya pengambilan *supernova bomb pickup* dan ditembakkan ke arah pemain terbesar selain *player*.

4.2.4 Mekanisme *Teleporter*

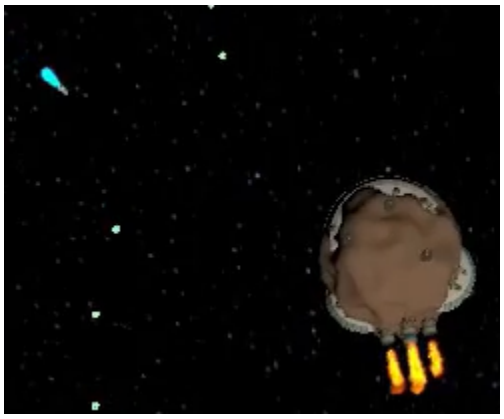
Dapat diamati bahwa mekanisme *teleporter* berhasil dengan adanya penembakan *teleporter* ke arah musuh dengan ukuran yang lebih kecil dan melakukan *teleport* saat *teleporter* dekat dengan musuh dengan ukuran yang lebih kecil dari *player*.

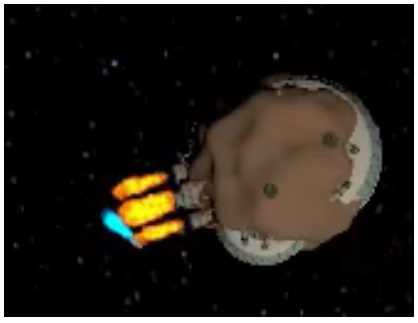
4.2.5 Mekanisme Penghindaran *Dangerous Objects*

- *Gas Cloud*

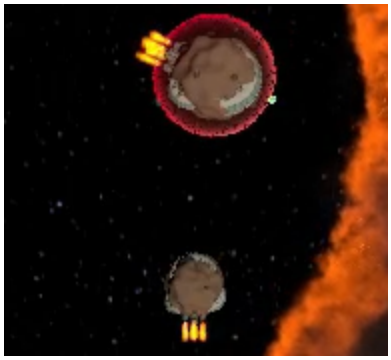
- *Torpedo*

- *Teleporter*





- *Larger Enemy*



Dari keempat kasus diatas, dapat diamati bahwa mekanisme penghindaran *dangerous object* berhasil dengan adanya pergerakan ke arah yang bertujuan untuk menghindari objek berbahaya yang terdeteksi.

4.2.6 Mekanisme *Shield*

Dapat diamati bahwa mekanisme *shield* berhasil bekerja dengan adanya aktivasi *shield* saat ada torpedo yang mendekat dengan *player*.

4.2.7 Pengujian Berdasarkan Hasil

- **Match 1**

{"Placement":1,"Seed":634,"Score":107,"Id":"e3d12e62-3862-454e-84d9-591bdc1bab59","Nickname":"Yasin_Bot","MatchPoints":8},

Placement : 1

Skor akhir : 107

- **Match 2**

{"Placement":1,"Seed":16643,"Score":86,"Id":"f9bfde5c-fa1c-4457-a09f-0817b0f5fc1c","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 86

- **Match 3**

{"Placement":3,"Seed":9133,"Score":27,"Id":"16d6cdf5-4047-4ceb-80fc-9584fedcf856","Nickname":"Yasin_Bot","MatchPoints":4}

Placement : 3

Skor akhir : 27

- **Match 4**

{"Placement":1,"Seed":732,"Score":70,"Id":"325f3a75-8d11-4f7c-8bf9-16ab2ccc1d4f","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 70

- **Match 5**

{"Placement":2,"Seed":28614,"Score":13,"Id":"7f765d92-b1c5-4e1e-8f2f-b71427e80a4c","Nickname":"Yasin_Bot","MatchPoints":6}

Placement : 2

Skor akhir : 13

- **Match 6**

{"Placement":4,"Seed":19995,"Score":37,"Id":"cc5ae7ae-bcc4-4bb2-9071-422e7ae732d1","Nickname":"Yasin_Bot","MatchPoints":2}

Placement : 4

Skor akhir : 37

- **Match 7**

{"Placement":1,"Seed":4892,"Score":77,"Id":"a55fbc1f-70a0-4b24-8838-5e8a6a79e4f1","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 77

- **Match 8**

{"Placement":1,"Seed":3918,"Score":93,"Id":"23f66ec4-a5d1-465f-b9b4-a827138611e5","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 93

- **Match 9**

{"Placement":1,"Seed":1103,"Score":55,"Id":"61669f7a-b2b7-4e26-9b5a-99a34d4b6f6b","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 55

- **Match 10**

{"Placement":1,"Seed":17006,"Score":56,"Id":"4af6c25d-04a5-453e-ad6b-d826ed09a818","Nickname":"Yasin_Bot","MatchPoints":8}

Placement : 1

Skor akhir : 56

Setelah melakukan simulasi pertandingan sebanyak 10 kali dengan 3 *bot* lain (bukan *reference bot*), didapatkan 8 pertandingan dengan posisi pertama dan 2 pertandingan dengan hasil yang kurang memuaskan. Setelah analisis, ada beberapa faktor yang dapat mempengaruhi hasil yang kami dapatkan, yaitu efisiensi algoritma, *map seed*, dan efisiensi algoritma *bot* lain. Dari hasil simulasi ini didapatkan persentase kemenangan sebesar 80%, dan dapat disimpulkan bahwa *bot* yang telah kami buat dengan algoritma *Greedy* cukup kuat dan kami yakin dapat mengalahkan *bot-bot* lain.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma *greedy* dalam pembuatan *bot* permainan "Galaxio" yang memiliki tujuan objektif berupa mengakumulasi skor tertinggi dari semua *bot* yang bertanding dalam permainan. Dari hasil yang didapatkan, dapat diamati bahwa aplikasi strategi *greedy* sangat berguna dalam permainan "Galaxio", dikarenakan adanya kebutuhan ukuran pemain yang besar dan penyerangan kepada musuh agar ukurannya mengecil untuk menjamin kemenangan. Strategi *greedy* digunakan pada saat pengambilan makanan agar dapat mendapat makanan terbanyak dalam waktu yang singkat, saat pemilihan target untuk diserang, dan lain sebagainya.

Namun, strategi *greedy* yang telah diimplementasikan juga perlu didampingi dengan teknik heuristik agar algoritma *bot* secara keseluruhan menjadi lebih optimal dan dapat beraksi sesuai situasi. Secara umum, *bot* yang telah kami buat membuktikan bahwa strategi *greedy* cukup efisien untuk diaplikasikan ke dalam permainan "Galaxio", dengan persentase kemenangan yang cukup besar.

5.2 Saran

Untuk saran kedepannya, kami berharap tugas besar dapat disertakan dengan asistensi untuk mempermudah mahasiswa dalam mengerjakan dan memastikan apakah hal yang sudah dikerjakan berjalan dengan baik atau tidak. Selain itu, kami juga berharap referensi untuk file panduan tugas besar ditambahkan agar memperjelas proses pengerjaan.

Lampiran

Link Repository Github:

https://github.com/debbyalmadea/Tubes1_Yasin_bot

Link Video Youtube:

bit.ly/VideoDemoYasinBot