PEMANFAATAN ALGORITMA ***GREEDY*** DALAM MENJALANKAN BOT PADA PERMAINAN ***GALAXIO*** DENGAN BAHASA PEMROGRAMAN **JAVA**

Disusun untuk memenuhi laporan tugas besar mata kuliah IF2211 Strategi Algoritma semester 4 di Institut Teknologi Bandung.

Disusun oleh kelompok subsetRPL10:

Kevin John Wesley Hutabarat (13521042)

Ryan Samuel Chandra (13521140)

Haziq Abiyyu Mahdy (13521170)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

PRAKATA

Manusia tidak pernah lepas dari rasa syukur kepada Tuhan Yang Maha Esa, atas berkat dan rahmat-Nya yang tidak pernah lepas dari kehidupan kita. Demikian pula kami bersyukur karena dapat menyelesaikan tugas besar ini tepat waktu, setelah dua minggu penuh melakukan kerja sama pemrograman secara rutin. Tidak lupa, kami mengucapkan banyak terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., selaku dosen pengajar mata kuliah IF2211 Strategi Algoritma, beserta jajaran asisten yang mendampingi, atas kesempatan yang diberikan kepada kami untuk belajar melalui tugas ini.

Dokumen ini berisi laporan lengkap dengan judul "Pemanfaatan Algoritma *Greedy* dalam Menjalankan Bot pada Permainan *Galaxio* dengan Bahasa Pemrograman Java". Laporan ini selain dibuat agar bermanfaat bagi masyarakat, juga secara khusus disusun untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma di semester 4 Teknik Informatika Institut Teknologi Bandung.

Dengan berbagai usaha pengumpulan informasi, serta uji coba program, kami memberikan performa maksimal untuk memenuhi tujuan pembuatan yang telah ditetapkan, meskipun berada di tengah tekanan waktu dan hambatan lingkungan.

Kami berharap percobaan kami ini dapat menjadi sesuatu yang memuaskan bagi semua pembaca. Tetapi, kami pun menyadari bahwa masih banyak kekurangan yang bisa diperbaiki. Maka dari itu, kami mohon kritik dan saran yang membangun untuk perkembangan percobaan kami. Terima kasih, selamat membaca.

Bandung, 17 Februari 2023

Penyusun Laporan

DAFTAR ISI

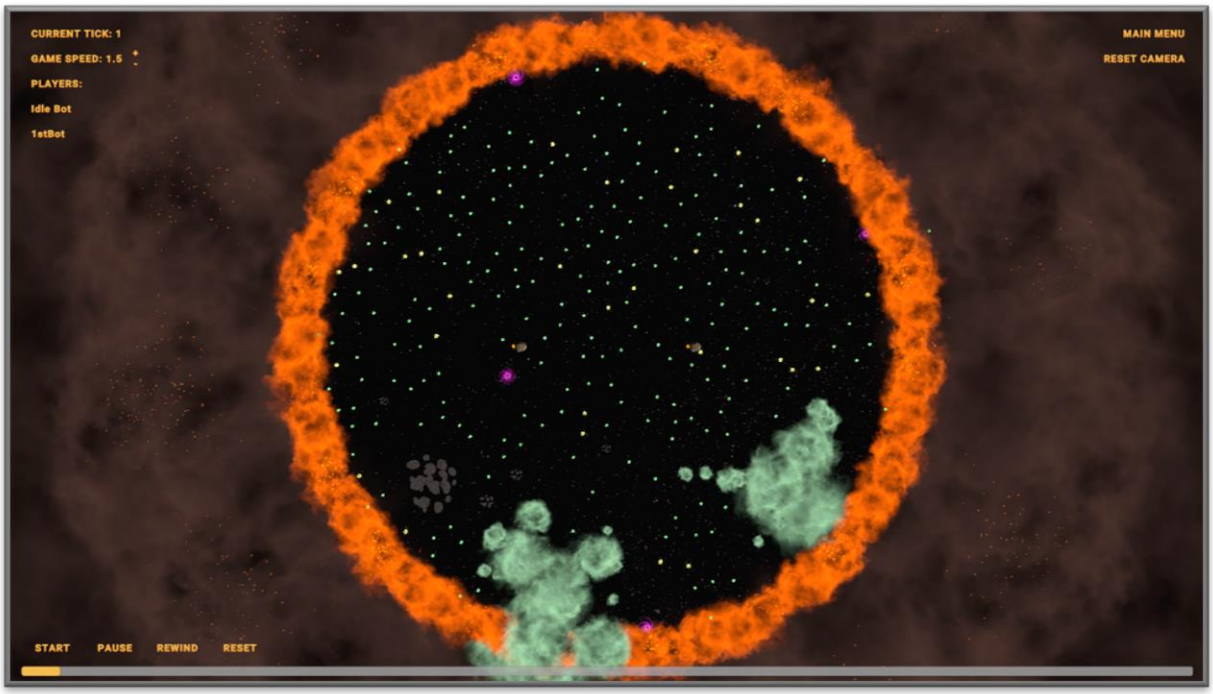
BAB 1
DESKRIPSI TUGAS

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

Gambar 1.1 Tampilan Awal Permainan *Galaxio*

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 maksimum pada game sama dengan ukuran radius. Pada peta, akan terdapat 5 objek, yaitu “Players”, “Food”, “Wormholes”, “Gas Clouds”, dan “Asteroid Fields”. Ukuran peta akan mengecil seiring batasan peta mengecil.



Gambar 1.2 Tampilan Peta Permainan *Galaxio*

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 dikonsumsi oleh kapal *player*. Apabila *player* mengkonsumsi Food, maka ia akan bertambah ukuran sebesaran ukuran Food yang dikonsumsi. 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 maksimum. Ketika Wormhole dilewati, maka Wormhole akan mengecil sebanyak setengah dari ukuran kapal yang melewatinya dengan syarat Wormhole lebih besar dari kapal *player*.
5. "Gas Cloud" 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 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 maksimum adalah 10.
9. Ketika kapal *player* bertabrakan dengan kapal lain, maka kapal yang lebih kecil akan dikonsumsi oleh kapal yang lebih besar sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maksimum 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. USESHIELD

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.

(Dikutip dari spesifikasi tugas besar IF2211 tahun 2023)

BAB 2
LANDASAN TEORI

2.1 Definisi Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang memecahkan persoalan berdasarkan langkah demi langkah, yang pada setiap langkahnya berusaha untuk selalu mengambil pilihan terbaik yang dapat diperoleh dari semua kemungkinan pada langkah itu tanpa memikirkan langkah selanjutnya. Algoritma ini hanya dapat memperoleh optimum lokal, dan dari memilih optimum lokal tersebut diharapkan akan berakhir dengan optimum global. Algoritma *greedy* adalah salah satu algoritma yang paling banyak digunakan karena tergolong sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi adalah persoalan untuk mencari solusi yang paling optimal, dapat berupa maksimasi dan minimasi.

Algoritma *greedy* dapat digunakan untuk persoalan yang solusi terbaik mutlaknya tidak terlalu diperlukan. Algoritma *greedy* dapat menghasilkan solusi hampiran yang waktu prosesnya lebih singkat daripada menggunakan algoritma dengan kebutuhan waktu eksponensial. Agar menghasilkan solusi yang optimal, kita harus memilih fungsi seleksi yang tepat.

Algoritma *greedy* memiliki beberapa elemen, yaitu:

1. Himpunan Kandidat: berisi kandidat yang akan dipilih pada setiap langkah
2. Himpunan Solusi: berisi kandidat yang sudah dipilih
3. Fungsi Solusi: Menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi Seleksi: Memilih kandidat berdasarkan strategi *greedy* tertentu, bersifat *heuristic* (spekulatif, berdasarkan pengalaman)
5. Fungsi Kelayakan: Memeriksa apakah kandidat yang dipilih layak untuk masuk solusi
6. Fungsi Obyektif: Fungsi yang memaksimumkan atau meminimumkan

2.2 Cara Kerja Program

Program ini dapat dijalankan dengan mengunduh *file zip* yang terdapat pada tautan yang telah disediakan. Pada folder starter-pack tersebut, terdapat beberapa folder yang mendukung keberjalanan program, yaitu engine-publish, logger-publish, reference-bot-publish, runner-publish, starter-bots, dan visualiser. Pada folder engine-publish berisi komponen yang berperan dalam mengimplementasikan logika dan aturan-aturan dalam game. Pada folder logger-publish berisi komponen yang berperan dalam sebuah match, juga berperan dalam menghubungkan bot dengan *engine*. Folder logger-publish berisi komponen yang berperan untuk mencatat log permainan sehingga pemain dapat mengetahui hasil permainan. Hasil dari log ini akan diinput ke *visualizer* nantinya. Starter-bots berisi bot-bot beserta konfigurasinya, yang akan dikembangkan sedemikian rupa berdasarkan strategi *greedy*.

Berikut ini adalah alur cara kerja game Galaxio:

1. Runner akan meng-*host* sebuah *match* pada sebuah *hostname* tertentu.
2. Engine dijalankan untuk berkoneksi dengan runner. Engine akan menunggu sampai semua bot pemain terkoneksi dengan server.
3. Logger melakukan koneksi dengan runner.
4. Mengkoneksikan bot dengan runner agar *match* dapat dimulai. Jumlah bot yang akan digunakan dapat diatur pada *file* JSON "appsettings.json" pada folder "runner-publish" dan "engine-publish"
5. Permainan dimulai sesaat setelah semua bot telah terkoneksi.
6. Bot yang terkoneksi akan mendengarkan *event-event* dari runner. Salah satu *event* yang penting adalah RecieveGameState, karena *event* ini memberikan status game pada saat itu.
7. Bot mengirimkan *event* kepada runner yang berisi aksi bot
8. Permainan akan berlangsung sampai selesai dan akan dibuat dua *file* JSON berisi kronologi *match*.

Untuk menjalankan game secara lokal dapat dilakukan dengan tahapan-tahapan sebagai berikut:

1. Konfigurasikan jumlah bot yang akan digunakan pada file "appsettings.json" dalam folder "runner-publish" dan "engine-publish"
2. Buka terminal baru pada folder runner-publish dan ketik "dotnet GameRunner.dll" pada terminal.
3. Buka terminal baru pada folder engine-publish dan ketik "dotnet Engine.dll" pada terminal.
4. Buka terminal baru pada folder logger-publish dan ketik "dotnet Logger.dll" pada terminal.
5. Jalankan semua bot yang ingin dimainkan, masing-masing bot dijalankan pada terminal yang terpisah
6. Setelah permainan selesai, kronologi *match* akan tersimpan pada 2 file JSON, yang kemudian bisa dilihat secara visualnya melalui *visualizer* dalam folder starter-pack

Cara lain ntuk menjalankan Game Runner, Engine, dan Logger adalah dengan memanfaatkan "run.bat" yang tersedia. Sebelum itu, bot-bot yang akan digunakan harus dibuat *file* .jar nya terlebih dahulu. Membuat *file* jar dapat dilakukan dengan maven ataupun inteliJ. Jika menggunakan maven, pembuatan *file* jar dapat dilakukan dengan mengetikkan "mvn clean package" pada terminal folder bot tersebut.

BAB 3
APLIKASI STRATEGI *GREEDY*

3.1 Pemetaan Persoalan Galaxio

Himpunan kandidat	:	himpunan *action* (aksi) yang dapat dilakukan oleh bot beserta *heading* (arah bot melakukan aksinya)
Himpunan solusi	:	aksi dan *heading* yang terpilih
Fungsi solusi	:	memeriksa apakah aksi dan *heading* yang terpilih adalah aksi dan *heading* yang paling menguntungkan berdasarkan poin
Fungsi seleksi	:	memilih aksi dan *heading* dengan profit yang tertinggi
Fungsi kelayakan	:	-
Fungsi obyektif	:	memaksimalkan keuntungan sehingga dapat bertahan sampai akhir permainan

3.2 Alternatif Solusi

3.2.1 Strategi *Greedy* Berdasarkan Mengejar Objek yang Menguntungkan

Salah satu strategi dasar yang dapat dikerjakan dengan algoritma *greedy*, yaitu mengambil sebanyak-banyaknya objek yang dianggap paling "menguntungkan", misalnya makanan, atau sesuatu yang dapat menambah poin dan ukuran bot. Langkah-langkah yang diimplementasikan sebagai berikut:

a. Membuat sebuah *list* yang berisi bot-bot lawan terurut berdasarkan jarak antara bot tersebut dengan bot sendiri.
b. Membuat sebuah *list* yang berisi objek-objek lain yang terdapat dalam permainan, terurut berdasarkan nilai profit.
c. Membuat sebuah *list* yang berisi bot-bot lawan yang berada pada jangkauan yang dekat. Jika *list* tersebut memiliki isi, maka akan diperiksa apakah ada bot yang berukuran lebih besar. Jika ada, maka bot tersebut akan ditandai sebagai dangerousPlayer. Jika tidak ada, maka akan diambil bot yang terbesar dan ditandai sebagai largestEdiblePlayer.
d. Jika semua bot lain berukuran lebih kecil, heading akan diatur sehingga menghadap ke largestEdiblePlayer dan aksi yang dilakukan adalah maju.
e. Jika ada yang lebih besar, maka akan diperiksa ukuran bot. Apabila lebih besar dari 50 dan bot memiliki torpedoSalvo, maka bot akan menembakkan torpedosalvo ke arah lawan. Jika tidak, maka bot akan menyalakan afterburner untuk melarikan diri. Jika ukuran bot lebih kecil daripada 50, maka bot akan menghindar dari bot lawan tanpa afterburner.
f. Jika afterburner aktif dan tidak ada bot yang perlu dihindari, maka aksi yang akan dilakukan bot adalah mematikan afterburner.

g. Jika bot sudah berada dekat dengan tepi *map*, maka bot akan diatur supaya berjalan mengarah ke titik tengah *map*.
h. Jika ukuran bot lebih besar dari 50 dan memiliki 5 torpedo salvo, maka bot akan menembak bot lain yang paling besar dengan torpedosalvo.
i. Jika *list* objek memiliki isi, maka akan diperiksa nilai profit. Jika profit bernilai negatif, maka objek tersebut akan ditandai sebagai benda berbahaya (dangerousObj). Jika positif, maka akan dicari objek dengan nilai profit yang paling tinggi.
j. Jika tidak ada objek dengan profit negatif yang terdeteksi, maka bot akan berjalan mengarah kepada mostProfitableObject. Jika ada, maka akan diperiksa objek berbahaya tersebut bertipe apa. Jika objek bertipe SUPERNOVABOMB, maka arah bot akan dibelokkan 90 derajat dan tetap berjalan. Jika objek tersebut bertipe torpedosalvo, maka bot akan dibelokkan, dan jika bot lebih besar daripada 50 dan memiliki shield, maka shield akan diaktifkan. Jika tidak, maka bot akan berjalan saja.
k. Jika objek tersebut bertipe *gas cloud* dan bot sudah terperangkap dalam gascloud, maka bot akan berjalan mengarah ke titik tengah *map* (ini hanyalah pendekatan heuristik, karena biasanya *gas cloud* jarang terletak di tengah *map*). Jika tidak, maka bot akan berbelok 120 derajat. Jika objek tersebut bertipe lain, maka bot akan dibelokkan 120 derajat.
l. Jika dari kasus-kasus sebelumnya tidak ada yang terpenuhi, maka bot akan berjalan saja sesuai dengan arah sebelumnya.

3.2.2 Strategi *Greedy* Berdasarkan Menghindari Objek-Objek Berbahaya

Strategi *greedy* ini berfokus pada pelarian dari objek-objek berbahaya yang dapat mengganggu keberlangsungan bot. Langkah-langkah strategi ini adalah:

a. Membuat sebuah *list* yang berisi bot-bot lawan yang lebih besar dari ukuran bot sendiri yang jaraknya cukup dekat (radius 0,4 kali ukuran *map*), terurut berdasarkan jarak dari yang terdekat.
b. Membuat sebuah *list* yang berisi objek-objek berbahaya lain yang jaraknya cukup dekat, yaitu ASTEROIDFIELD, GASCLOUD, TORPEDOSALVO, dan SUPERNOVABOMB. Daftar terurut berdasarkan jarak dari yang terdekat.
c. Memeriksa kedua *list* tersebut ada isinya atau tidak. Jika keduanya ada, maka akan ditentukan apa yang lebih dekat. Jika yang lebih dekat adalah bot, program akan menangani bot, begitu pun sebaliknya.
d. Jika ada bot yang berbahaya, bot akan menghindari bot lawan. Jika bot memiliki TORPEDOSALVO, maka bot dapat menembak bot lawan. Jika tidak, maka bot akan menghindar. Jika ukuran bot cukup untuk menghidupkan AFTERBURNER, maka bot akan menghindar menggunakan AFTERBURNER.

e. Jika ada objek yang berbahaya, bot akan menghindari objek tersebut dengan arah 90 atau 180 derajat dari objek tersebut, tergantung jenis objeknya.
f. Jika kedua *list* kosong, maka bot hanya akan berjalan lurus tanpa mengubah *heading*.

3.2.3 Strategi *Greedy* Berdasarkan Perbandingan Bobot terhadap Jarak Bot dengan Objek

Berbeda dengan *greedy by profit*, strategi *greedy* ini mempertimbangkan jarak antara bot dengan objek, di samping memperhitungkan keuntungan yang bisa didapat dengan menargetkan objek tersebut. Berikut adalah elemen strategi *greedy* yang dipakai:

1. Himpunan kandidat: himpunan yang berisi objek-objek (*player* dan *nonplayer*) yang terdapat pada peta, diurutkan secara menaik berdasarkan jaraknya.
2. Himpunan solusi: salah satu dari beberapa kandidat terawal.
3. Fungsi solusi: memeriksa kandidat mana yang memiliki nilai terbesar dari perbandingan keuntungan dengan jaraknya pada bot.
4. Fungsi seleksi: memilih salah satu dari beberapa kandidat terawal yang memiliki nilai terbesar dari perbandingan keuntungan dengan jaraknya pada bot.
5. Fungsi kelayakan: tidak ada
6. Fungsi obyektif: memaksimalkan pendapatan keuntungan (bobot) per jarak dengan harapan bot dapat bertahan sampai akhir permainan.

Berikut langkah yang diterapkan dalam kode program untuk mengimplementasikan strategi "*greedy by density*" tersebut:

a. Program membuat dua buah larik, yang pertama berisi seluruh objek nonpemain yang ada pada peta permainan, diurutkan berdasarkan jarak terdekat. Satu lagi berisi semua objek pemain, diurutkan berdasarkan jarak terdekat.
b. Program ditambah penanganan beberapa kasus khusus seperti SUPERNOVA yang telah ditembakkan, AFTERBURNER yang menyala, serta kapal yang mendekati ujung peta.
c. Secara umum (jika tidak ada kasus khusus), program akan mengambil masing-masing beberapa objek (*default*: 5) dari kedua larik, kemudian menghitung bobot objek tersebut dibagi jaraknya dengan bot. Program akan menyimpan indeks objek yang memiliki nilai luaran-fungsi paling besar.
d. Program kemudian memanggil fungsi `action` dengan parameter objek terpilih. Fungsi ini akan mengembalikan aksi dan arah yang harus dituju pada *tick game* selanjutnya.
e. Benda-benda berbahaya akan dihindari, makanan akan diincar, bot lain akan ditembak dengan TORPEDOSALVO dan didekati jika ukurannya lebih kecil, namun ditinggalkan jika ukurannya lebih besar. Sedangkan, beberapa objek dianggap sebagai benda netral dan tidak ditangani terlalu dalam.
f. Pada kesempatan tertentu, AFTERBURNER diaktifkan untuk mempercepat gerak dan dimatikan sesaat kemudian. Tentu saja ketersediaan ukuran (nyawa) bot selalu

diperiksa. Selain itu, bot yang lebih besar akan ditembak dengan SUPERNOVA (jika ada) yang diledakkan beberapa saat setelah ditembakkan.

3.3 Efisiensi dan Efektivitas Solusi yang Dipilih

Dari ketiga alternatif tersebut, metode yang digunakan berbeda-beda. Ada yang menggunakan algoritma *greedy* berdasarkan profitnya, menghindari objek berbahaya, dan berdasarkan perbandingan antara profit dengan jarak terhadap bot. Dengan strategi yang berbeda ini, hasil yang diperoleh tentunya akan berbeda pula.

Pada alternatif yang pertama, bot berfokus kepada mengambil objek-objek yang menguntungkan, seperti FOOD, SUPERFOOD, bahkan bot lawan yang lebih kecil. Tetapi alternatif solusi ini juga mempertimbangkan apabila profit tertinggi yang terdeteksi bernilai negatif, bot akan menandainya sebagai objek yang berbahaya. Jika ada bot lain yang dekat, bot akan menembakkan torpedosalvo untuk mencuri ukuran lawan. Jika hanya ada objek yang berbahaya, bot akan menghindari dan melindungi diri dari objek tersebut. Kelemahan dari bot ini adalah bot acap kali hanya bolak-balik karena berada di antara objek yang jaraknya sama.

Pada alternatif yang kedua, bot berfokus kepada menghindari objek-objek yang berbahaya, baik bot lawan yang lebih besar maupun objek lain yang mengurangi ukuran bot. Jika ukuran bot cukup untuk melawan, maka bot akan menembak torpedosalvo ataupun melindungi diri dengan shield. Bot juga akan menghindari tepian *map* dengan berjalan kearah tengah *map*. Jika tidak ada objek yang harus dihindari, bot akan mengejar objek lain yang menguntungkan. Kelemahan dari algoritma ini adalah bot acap kali hanya bolak-balik karena berada di antara objek dengan jarak yang sama, juga tidak bisa keluar apabila terperangkap di dalam GASCLOUD ataupun ASTEROIDFIELD.

Pada alternatif yang ketiga, bot berfokus kepada mengejar objek dengan perbandingan antara profit dengan jarak yang tertinggi. Semakin tinggi perbandingannya, maka akan semakin menguntungkan bot. Kelemahan dari alternatif ini adalah terkadang perbandingan profit yang tertinggi bernilai *negative*, namun bot tetap mengejar objek tersebut, yang mana hal ini dapat merugikan bot.

Dari semua alternatif yang ada, strategi *greedy* yang dirasa paling sesuai dengan keseluruhan tujuan adalah "*greedy by profit*", alias mengejar objek yang menguntungkan. Alasannya adalah dengan mengejar objek yang paling menguntungkan, dan dengan penambahan fitur menghindari objek berbahaya, maka bot akan lebih cepat untuk memperbesar ukurannya, dan dapat mengkonsumsi bot lain yang lebih kecil dari ukurannya. Dengan demikian, bot akan lebih mudah untuk menang dibandingkan dengan penggunaan strategi *greedy* yang lainnya.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy* pada Program Bot dengan notasi *pseudocode*

```
procedure computeNextPlayerAction (input/output playerAction: PlayerAction, input bot: GameObject, nearbyObjectList: List of GameObject, playerList: List of GameObject, boolean afterburnerStatus)
    {I.S. playerAction tidak terdefinisi atau berisi aksi pemain sebelumnya. nearbyPlayerList terdefinisi, berisi kumpulan pemain lain yang berada di dekat bot yang terurut membesar berdasarkan jarak bot dengan pemain lain tersebut. nearbyObjectList terdefinisi, berisi semua objek yang terurut membesar berdasarkan jarak bot dengan objek tersebut.}
    {F.S. playerAction yang telah ditentukan berdasarkan kondisi bot}

KAMUS LOKAL
       allSmaller, isSafe: boolean
       largestEdiblePlayer: GameObject
       maxPlayerSize: integer
       targetBot: GameObject
       mostProfitableObj: GameObject
       dangerousObj: GameObject
       maxProfit: integer

ALGORITMA
       nearbyPlayerList ← filter(playerList)
       {ambil pemain yang ada dalam radius (0.15 * radius peta) di sekitar bot}
       if (not isEmpty(nearbyPlayerList)) then
       {Jika terdeteksi objek yang dekat dengan bot}
              allSmaller ← true
              maxPlayerSize ← -1

              for each player in nearbyPlayerList
                     if (getSize(player) > getSize(bot)) then:
                            dangerousPlayer ← player
                            allSmaller ← false
                            break

                     else if (getSize(player) > maxPlayerSize) then
                            maxPlayerSize ← getSize(player)
                            largestEdiblePlayer ← player

              if (allSmaller) then
              {Jika semua pemain yang ada di sekitar bot ukurannya lebih kecil dibandingkan bot, maka bot maju dengan arah menuju pemain terbesar}
                     playerAction.action ←  FORWARD
                     playerAction.heading ← getHeadingBetween(largestEdiblePlayer)

              else {Ada pemain terdekat yang lebih besar dibandingkan bot}
                     if (getSize(bot) > 50) then
                            if (torpedoSalvoCount > 0) then
```

```
                        {Bot menembak torpedosalvo dengan arah menuju
                        dangerousBot}
                                playerAction.action <- FIRETORPEDOES
                                playerAction.heading <-
                                getHeadingBetween(dangerousBot)
                else
                {Bot berbalik dari pemain tersebut dan menggunakan
                afterburner}
                        playerAction.action ← STARTAFTERBURNER
                        playerAction.heading ← (getHeadingBetween(dangerousBot)
                                                + 180) mod 360
                        afterburnerStatus ← true
        return

if (afterburnerStatus) then
        playerAction.action ← STOPAFTERBURNER
        return

if (tooFarFromCenter(bot)) then
{Jika bot terlalu dekat dengan tepi map, bot bergerak menuju titik tengah
map}
        playerAction.action ← FORWARD
        playerAction.heading ← getHeadingToCenter()
        return

if (size(bot) >= 50) then
        if (torpedoSalvoCount = 5) then
                targetBot ← getMaxSize(playerList)
                playerAction.action ← FIRETORPEDOES
                playerAction.heading ← getHeadingBetween(targetBot)
                return

if (not isEmpty(nearbyObjectList)) then
        isSafe ← true
        maxProfit ← -1

        for each obj in nearbyObjectList
                if (getProfit(obj) < 0) then
                        isSafe ← false
                        dangerousObj ← obj
                        break
                else if (getProfit(obj) > maxProfit) then
                        maxProfit ← getProfit(obj)
                        mostProfitableObj ← obj

        if (isSafe)
        {Jika tidak ada objek berbahaya di sekitar bot, maka bot maju ke arah
        objek yang paling menguntungkan}
                playerAction.heading ← getHeadingBetween(mostProfitableObj)
                playerActoin.action ← FORWARD
        else
                if (objectType(dangerousObj) = SUPERNOVABOMB) then
```

```
                {jika ada supernova bomb yang dekat, maka bot menghadap tegak
                lurus terhadap bomb tersebut, karena jika bot menghadap 180
                derajat dari bomb, maka bot akan terus dikejar oleh bomb}
                        playerAction.heading ← (getHeadingBetween(dangerousObj)
                                                + 90) mod 360
                        playerAction.action ← FORWARD
                else if (objectType(dangerousObj) = TORPEDOSALVO) then
                        if (botSize > 50 and shieldCount > 0) then
                        {jika ada torpedo salvo yang mendekat,bot memiliki
                        shield dan botSize mencukupi, maka bot akan mengaktifkan
                        shield}
                                playerAction.action ← USESHIELD
                        else
                                playerAction.action ← FORWARD
                else if (objectType(dangerousObj)=GASCLOUD) then
                        if (getEffectiveDistanceTo(dangerousObj) <= 0) then
                        {jika bot terperangkap dalam GASCLOUD, maka bot
                        diarahkan ke titik tengah map}
                                playerAction.heading ← getHeadingToCenter()
                                playerAction.action ← FORWARD
                        else
                                playerAction.heading ← (getHeadingBetween(
                                dangerousObj) + 120) mod 360
                                playerAction.action ← FORWARD

                else
                        playerAction.heading ← (getHeadingBetween
                        (dangerousObj) + 120) mod 360
                        playerAction.action ← FORWARD
                return
        playerAction.action ← FORWARD
        return
```

4.2 Struktur Data yang Digunakan dalam Program Bot *Galaxio*

Terdapat tiga *package* yang disediakan dalam program bot *Galaxio*, yaitu “Enums”, “Models”, dan “Services”, serta satu program utama (Main.java).

4.2.1 *Package* Enums

Package bernama “Enums” berisi semua *enum* yang dibutuhkan sebagai pilihan aksi pemain serta tipe objek yang ada pada *game*. *Enum* adalah sekumpulan konstanta yang sudah ditentukan (*predefined*) di mana jumlah anggota *enum* konstan dan tidak dapat diubah. Enum juga dapat memiliki atribut seperti kelas, dan tiap anggota enum akan dihubungkan dengan nilai dari atributnya sendiri. Terdapat dua *source file* pada *package* ini, yaitu ObjectTypes.java dan PlayerActions.java.

A. ObjectTypes.java

Berikut adalah seluruh elemen *enum* ObjectTypes. Catatan: Elemen TORPEDOSALVO hingga SHIELD sebelumnya belum ada di *starter bot* karena belum terupdate.

```
  PLAYER(1, 0),
  FOOD(2, 2),
  WORMHOLE(3, 0),
  GASCLOUD(4, -2),
  ASTEROIDFIELD(5, -1),
  TORPEDOSALVO(6, -3),
  SUPERFOOD(7, 4),
  SUPERNOVAPICKUP(8, 1),
  SUPERNOVABOMB(9, -4),
  TELEPORTER(10, 0),
  SHIELD(11, 3);
```

Berikut adalah atribut *enum* ObjectTypes.

Nama atribut	Keterangan
`public final Integer value;`	Merupakan nilai yang berguna untuk mengidentifikasi tiap elemen ObjectTypes. Nilai ini diperlukan karena setiap objek yang ada dalam *game* dikirimkan melalui *list of integer*.
`private final Integer profit;`	Merupakan bobot/keuntungan yang dimiliki tiap objek. Atribut ini merupakan atribut yang baru ditambahkan pada pengerjaan tugas besar ini (belum disediakan sebelumnya) karena berkaitan dengan algoritma *greedy* yang digunakan

Berikut adalah metode *enum* ObjectTypes.

Nama metode	Keterangan
`ObjectTypes(Integer value, Integer profit)`	Merupakan konstruktor untuk *enum* ObjectTypes
`public static ObjectTypes valueOf(Integer value)`	Merupakan metode untuk memetakan suatu nilai *integer* dengan elemen dari ObjectTypes
`public Integer getProfit()`	Merupakan metode untuk mendapatkan profit dari suatu elemen ObjectTypes

B. PlayerActions.java

Berikut adalah seluruh elemen enum PlayerActions. Catatan: elemen FIRETORPEDOES hingga USESHIELD sebelumnya belum ada di *starter bot* karena belum ter-*update*.

```
  FORWARD(1),
  STOP(2),
  STARTAFTERBURNER(3),
  STOPAFTERBURNER(4),
  FIRETORPEDOES(5),
  FIRESUPERNOVA(6),
  DETONATESUPERNOVA(7),
  FIRETELEPORTER(8),
  TELEPORT(9),
  USESHIELD(10);
```

Berikut adalah atribut *enum* PlayerActions.

Nama atribut	Keterangan
`public final Integer value;`	Merupakan nilai yang berguna untuk mengidentifikasi tiap elemen PlayerActions. Nilai ini diperlukan karena setiap objek yang ada dalam game dikirimkan melalui *list of integer*.

Berikut adalah metode *enum* PlayerActions.

Nama metode	Keterangan
`private PlayerActions(Integer value)`	Merupakan konstruktor untuk *enum* PlayerActions

4.2.2 *Package* Models

Package Models berisi seluruh *class* yang diperlukan dalam pemodelan *game*, seperti objek game (termasuk bot), keadaan *game*, aksi pemain, posisi (dalam koordinat kartesian), serta dunia (*world*).

A. GameObject.java

Berikut adalah atribut *class* GameObject.

Nama atribut	Keterangan
`public UUID id;`	ID objek
`public Integer size;`	Ukuran objek
`public Integer speed;`	Kecepatan objek
`public Integer currentHeading;`	Arah objek menghadap (dalam satuan derajat)
`public Position position;`	Posisi objek
`public ObjectTypes gameObjectType;`	Tipe objek (dari *enum* ObjectTypes)
`public int effectsHash;`	Kode *hash* dari efek yang dimiliki objek

`public int torpedoSalvoCount;`	Jumlah torpedo salvo yang dimiliki objek yang hanya dimiliki oleh objek berupa player. Objek selain player memiliki nilai 0 pada atribut ini serta pada tiga atribut di bawah ini
`public int supernovaAvailable;`	Jumlah supernova yang dimiliki *player*
`public int teleporterCount;`	Jumlah teleporter yang dimiliki *player*
`public int shieldCount;`	Jumlah shield yang dimiliki *player*

Berikut adalah metode *class* GameObject.

Nama metode	Keterangan
`public GameObject(UUID id, Integer size, Integer speed, Integer currentHeading, Position position, ObjectTypes gameObjectType, int effectsHash, int torpedoSalvoCount, int supernovaAvailable, int teleporterCount, int shieldCount)`	Merupakan konstruktor untuk objek dari *class* GameObject
`public UUID getId()`	*Getter* atribut ID
`public void setId(UUID id)`	*Setter* atribut ID
`public int getSize()`	*Getter* atribut size
`public void setSize(int size)`	*Setter* atribut size
`public int getSpeed()`	*Getter* atribut speed
`public void setSpeed(int speed)`	*Setter* atribut speed
`public Position getPosition()`	*Getter* atribut position
`public void setPosition(Position position)`	*Setter* atribut positoin
`public ObjectTypes getGameObjectType()`	*Getter* atribut gameObjectType
`public void setGameObjectType(ObjectTypes gameObjectType)`	*Setter* atribut setGameObjectType
`public static GameObject FromStateList(UUID id, List<Integer> stateList)`	Mendapatkan *state list* dari server, kemudian mengembalikan GameObject lengkap dengan atributnya berdasarkan elemen-elemen *state list* tersebut

B. GameState.java

Berikut adalah atribut *class* GameState.

Nama atribut	Keterangan
`public World world;`	Merupakan komponen ‘dunia’ dari *game*. Atribut ini merupakan objek dari kelas World yang akan dijelaskan lebih lanjut pada bagian World.java
`public List<GameObject> gameObjects;`	Merupakan *list* yang berisi seluruh objek yang ada pada *game* (selain pemain) pada waktu tertentu. *List* ini akan diperbarui setiap *tick*
`public List<GameObject> playerGameObjects;`	Merupakan list yang berisi seluruh pemain yang ada pada *game* (tidak termasuk pemain yang sudah kalah). *List* ini akan diperbarui setiap *tick*

Berikut adalah metode *class* GameState.

Nama metode	Keterangan
`public GameState()`	Merupakan konstruktor *default* untuk objek dari kelas GameState.
`public GameState(World world , List<GameObject> gameObjects, List<GameObject> playerGameObjects)`	Merupakan konstruktor berparameter untuk objek dari kelas GameState
`public World getWorld()`	*Getter* atribut world
`public void setWorld(World world)`	*Setter* atribut world
`public List<GameObject> getGameObjects()`	*Getter* atribut gameObjects
`public void setGameObjects(List<GameObject> gameObjects)`	*Setter* atribut gameObjects
`public List<GameObject> getPlayerGameObjects()`	*Getter* atribut playerGameObjects
`public void setPlayerGameObjects(List<GameObject> playerGameObjects)`	*Setter* atribut playerGameObjects

C. GameStateDto.java

Berikut adalah atribut *class* GameStateDto

Nama atribut	Keterangan
`private World world;`	Merupakan komponen ‘dunia’ dari *game*
`private Map<String, List<Integer>> gameObjects;`	Merupakan *map* (pasangan *key-value*) dengan *key* berupa *string* dan *value* berupa *list of integer*. Atribut ini berguna untuk proses entri objek pada saat *game*

	dimulai dan akan digunakan pada program utama (Main.java)
`private Map<String, List<Integer>> playerObjects;`	Merupakan *map* (pasangan *key-value*) dengan *key* berupa *string* dan *value* berupa *list of integer*. Atribut ini berguna untuk proses entri pemain pada saat *game* dimulai dan akan digunakan pada program utama (Main.java)

Berikut adalah metode *class* GameStateDto

Nama metode	Keterangan
`public Models.World getWorld()`	*Getter* atribut world
`public void setWorld(Models.World world)`	*Setter* atribut world
`public Map<String, List<Integer>> getGameObjects()`	*Getter* atribut gameObjects
`public void setGameObjects(Map<String, List<Integer>> gameObjects)`	*Setter* atribut gameObjects
`public Map<String, List<Integer>> getPlayerObjects()`	*Getter* atribut playerObjects
`public void setPlayerObjects(Map<String, List<Integer>> playerObjects)`	*Setter* atribut playerObjects

D. PlayerAction.java

Berikut ini adalah atribut yang terdapat dalam *class* PlayerAction

Nama atribut	Keterangan
`public UUID playerId;`	Merupakan ID dari pemain
`public PlayerActions action;`	Merupakan aksi yang akan dilakukan pemain
`public int heading;`	Merupakan arah pemain dalam melakukan aksi

Berikut ini adalah metode yang terdapat dalam class PlayerAction

Nama metode	Keterangan
`public UUID getPlayerId()`	*Getter* atribut playerId
`public void setPlayerId(UUID playerId)`	*Setter* atribut playerId
`public PlayerActions getAction()`	*Getter* atribut action

`public void setAction(PlayerActions action)`	*Setter* atribut action
`public int getHeading()`	*Getter* atribut heading
`public void setHeading(int heading)`	*Setter* atribut heading

E. Position.java

Berikut ini adalah atribut yang terdapat dalam *class* Position

Nama atribut	Keterangan
`public int x;`	Komponen absis dari koordinat objek
`public int y;`	Komponen ordinat dari koordinat objek

Berikut ini adalah metode yang terdapat dalam *class* Position

Nama metode	Keterangan
`public Position()`	Konstruktor default untuk objek dari kelas Position
`public Position(int x, int y)`	Konstruktor objek kelas Position dengan parameter x dan y
`public int getX()`	Fungsi untuk mendapatkan nilai x (absis) pada objek dari kelas Position
`public void setX(int x)`	Fungsi untuk meng-*assign* atribut x pada objek kelas Position menjadi integer x pada parameter
`public int getY(`	Fungsi untuk mendapatkan nilai y (ordinat) pada objek dari kelas Position
`public void setY(int y)`	Fungsi untuk meng-*assign* atribut y pada objek kelas Position menjadi integer y pada parameter

F. World.java

Berikut ini adalah atribut yang terdapat dalam *class* World

Nama Atribut	Keterangan
`public Position centerPoint;`	Merupakan titik tengah dari peta/dunia yang secara *default* bernilai (0,0)
`public Integer radius;`	Merupakan radius dari peta/dunia. Nilai radius akan selalu berkurang setiap *tick*
`public Integer currentTick;`	Menunjukkan waktu yang dihitung dari mulainya permainan dalam satuan *tick*

Berikut ini adalah metode yang terdapat dalam *class* World

Nama metode	Keterangan
`public Position getCenterPoint()`	*Getter* atribut centerPoint
`public void setCenterPoint(Position centerPoint)`	*Setter* atribut centerPoint
`public Integer getRadius()`	*Getter* atribut radius
`public void setRadius(Integer radius)`	*Setter* atribut radius
`public Integer getCurrentTick()`	*Getter* atribut currentTick
`public void setCurrentTick(Integer currentTick)`	*Setter* atribut currentTick

4.2.3 *Package* Services

Package "Services" berisi layanan yang disediakan untuk membuat algoritma bot.

A. BotService.java

Berikut adalah atribut *class* BotService

Nama atribut	Keterangan
`private GameObject bot;`	Merupakan komponen bot yang merupakan objek dari kelas GameObject
`private PlayerAction playerAction;`	Merupakan komponen berupa aksi player yang merupakan objek dari kelas PlayerAction
`private GameState gameState;`	Berisi keadaan *game* yang selalu diperbarui tiap *tick* dan dapat diakses oleh bot. Bot dapat mengakses *list* yang berisi seluruh pemain yang ada di *game* melalui metode `getPlayerGameObjects()` serta seluruh objek yang ada di *game* melalui metode `getGameObjects()`
`private boolean afterburnerStatus;`	Menggambarkan keadaan *afterburner* milik bot. Atribut ini akan bernilai *true* jika bot baru saja menyalakan *afterburner* dengan perintah STARTAFTERBURNER. Atribut ini akan bernilai *false* jika bot baru saja mematikan *afterburner* dengan perintah STOPAFTERBURNER

Berikut adalah metode *class* BotService

Nama metode	Keterangan
`public BotService()`	Merupakan konstruktor untuk objek dari kelas BotService
`public GameObject getBot()`	*Getter* atribut bot
`public void setBot(GameObject bot)`	*Setter* atribut bot
`public PlayerAction getPlayerAction()`	*Getter* atribut playerAction
`public void setPlayerAction(PlayerAction playerAction)`	*Setter* atribut playerAction
`public void computeNextPlayerAction(PlayerAction playerAction)`	Merupakan metode yang digunakan untuk menentukan aksi pemain selanjutnya. Algoritma *greedy* terletak pada metode ini
`public GameState getGameState()`	*Getter* atribut gameState
`public void setGameState(GameState gameState)`	*Setter* atribut gameState
`private void updateSelfState()`	Merupakan metode yang digunakan untuk memperbarui keadaan bot
`private double getDistanceBetween(GameObject object1, GameObject object2)`	Menemukan jarak Euclidean antara dua objek
`private int getHeadingBetween(GameObject otherObject)`	Menentukan *heading* yang diperlukan untuk menuju suatu objek
`private int toDegrees(double v)`	Mengubah sudut radian ke derajat
`private double getBotRadius()`	Mendapatkan posisi bot terhadap titik tengah peta/dunia (radius bot). Radius diperoleh dengan jarak Euclidean. Metode ini merupakan metode tambahan yang berguna untuk mengecek apakah posisi bot < World radius. Hal ini penting karena apabila bot melewati batas dunia/*map*, ukuran bot akan mengecil hingga mati
`private int getHeadingToCenter()`	Mendapatkan *heading* yang diperlukan untuk pergi ke tengah dunia. Metode ini merupakan metode tambahan biasa dipanggil apabila radius bot sudah hampir sama dengan World radius agar bot dapat kembali ke posisi yang aman

`private int getWorldRadius()`	Mendapatkan radius World pada saat metode ini dipanggil. Metode ini ditambahkan karena radius dunia pada *game* selalu berkurang setiap *tick* nya, sehingga perlu dilakukan pengecekan
`private double getEffectiveDistanceTo(GameObject other)`	Mendapatkan jarak efektif antara bot dengan objek lainnya, yaitu jarak Euclidean antara bot dengan objek lainnya dikurangi dengan ukuran bot dan objek lain tersebut. Metode ini penting supaya apabila ada bot lain yang berukuran besar, kita tidak menganggap posisi bot tersebut jauh dari bot kita hanya karena ukuran bot tersebut besar.

4.2.4 Program Utama (Main.java)

Program utama berisi pemanggilan method yang diperlukan untuk menghubungkan bot ke server, inisialisasi objek-objek yang diperlukan, serta mendapatkan aksi yang akan dilakukan bot setiap satuan tick untuk kemudian dikirim ke server. Program utama tidak memiliki atribut dan hanya memiliki satu metode, yaitu "main".

4.3 Analisis Desain Algoritma

4.3.1 Analisis Kompleksitas Algoritma *Greedy*

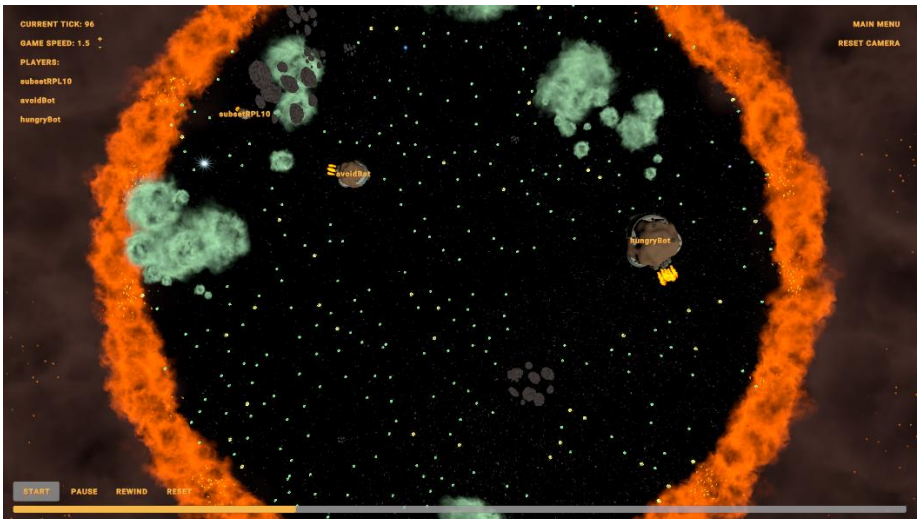
Jika proses *sorting* dan *filtering list* tidak diperhitungkan, maka tahapan yang dilakukan pada algoritma ini beserta kompleksitas waktunya dalam notasi *Big-Oh* adalah sebagai berikut.

Tahapan	Kompleksitas waktu
Perhitungan jarak Euclidean antarobjek	*O*(1)
Perhitungan *heading* yang diperlukan	*O*(1)
Traversal list pemain	*O*(n)
Traversal list objek	*O*(n)
Assignment heading dan aksi yang dilakukan	*O*(1)

Sehingga, kompleksitas keseluruhan algoritma *greedy* bot ini adalah *O*(n)

4.3.2 Analisis Keoptimalan Algoritma *Greedy*

4.3.2.1 Contoh kasus yang menyebabkan algoritma tidak optimal

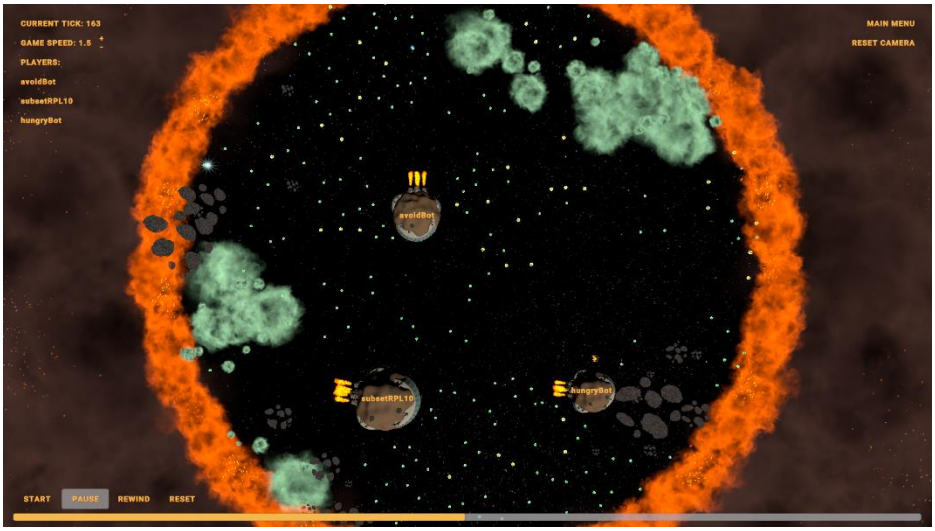


Gambar 4.1 Contoh kasus tidak optimal

Terdapat *confusion* pada bot ketika terdapat dua buah objek yang harus dihindari yang berdekatan dan berjarak sama dari bot, sehingga bot hanya membolak-balikkan arah heading nya. Contohnya adalah bot subsetRPL10 pada gambar di atas yang terjebak di antara gas cloud dan tepi *map*. Idealnya, bot bisa mencari jalan keluar, tetapi algoritma ini belum dapat menyelesaikan permasalahan ini dengan optimal.

4.3.2.2 Contoh kasus yang menyebabkan algoritma optimal

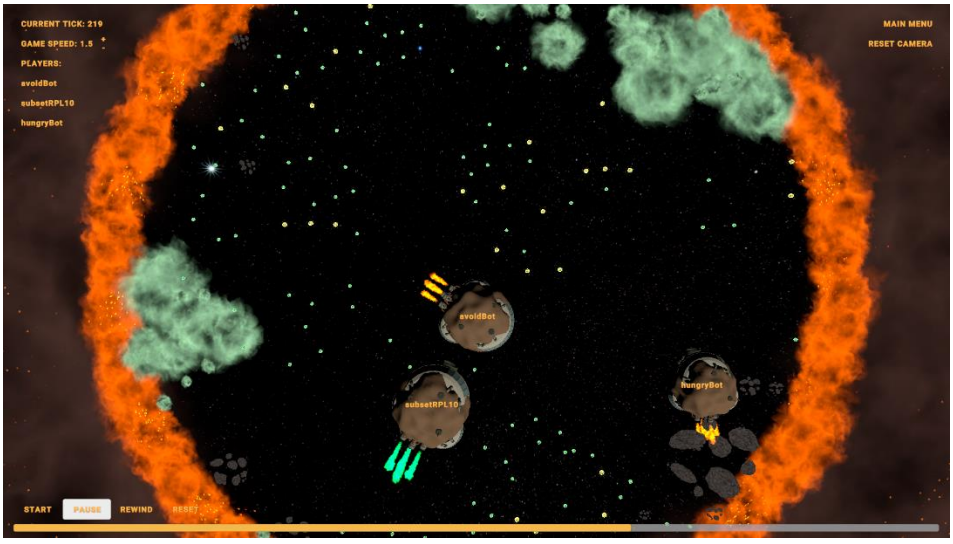
Gambar 4.2 Contoh kasus optimal (1)



Gambar 4.3 Contoh kasus optimal (2)

Contoh kasus yang menyebabkan algoritma optimal adalah ketika tembakan berhasil mengenai lawan yang berukuran besar dalam jarak jauh. Pada algoritma ini, bot memang didesain untuk menembak lawan dengan ukuran terbesar agar bot dapat menambah ukuran dan mengurangi ancaman dari bot yang berukuran besar. Tembakan jarak jauh memiliki peluang yang cukup rendah untuk mengenai lawan, namun apabila tembakan tersebut tepat sasaran, akan memberikan manfaat yang besar bagi bot sehingga merupakan kasus optimal. Seperti pada gambar di atas, tembakan bot subsetRPL10 berhasil mengurangi ukuran hungryBot.

4.3.2.3 Contoh kasus unik



Gambar 4.2 Contoh kasus unik

Salah satu kasus unik yang terjadi adalah *afterburner* yang menyala ketika bot sedang mengejar bot lain. Hal ini berbeda dengan penggunaan *afterburner* yang didesain pada algoritma. Seharusnya, *afterburner* hanya digunakan untuk menghindari bot berbahaya, tetapi pada kasus ini, *afterburner* malah aktif saat mengejar pemain lain. Hal ini menjadi keuntungan tak terduga yang terjadi pada bot.

BAB 5
PENUTUP

5.1 Kesimpulan

Galaxio adalah sebuah permainan *battle royale* yang mempertandingkan sebuah bot kapal dengan beberapa bot lainnya.Tujuannya adalah membuat bot yang dibuat sendiri bertahan sampai tersisa sendiri di akhir permainan. Ada banyak peraturan dan objek yang perlu diperhatikan dalam *Galaxio*. Bot kapal memiliki kemampuan untuk bergerak, mengubah arah, mengambil makanan yang tersebar pada peta, hingga memakan bot lain.

Bot untuk permainan *Galaxio* bisa dibuat menggunakan algoritma *greedy* dalam bahasa pemrograman Java. Algoritma *greedy* memecahkan sebuah persoalan dengan berusaha selalu mengambil pilihan terbaik pada setiap langkah tanpa memikirkan langkah selanjutnya. Algoritma ini diharapkan dapat berakhir dengan optimum global dari optimum lokal.

Pada akhirnya, terbukti bahwa bot yang dimaksud berhasil diimplementasikan. Kode program secara lengkap dapat dilihat pada bagian **Lampiran**, sedangkan fungsi-fungsi dan strategi *greedy* yang digunakan bervariasi, dapat dilihat pada bagian **BAB 3** dan **BAB 4**.

5.2 Saran

1. Dalam mengerjakan pemrograman, sebaiknya para pemrogram memiliki catatan kecil tentang fungsi, prosedur, dan juga *class* yang sudah ada, maupun yang dibuat sendiri. Hal ini bertujuan untuk mempermudah kerja sama tim serta diri sendiri apabila ingin mencari subrutin atau *atribut* untuk proses tertentu. Selain itu, catatan juga berguna untuk menjamin tidak adanya duplikasi.
2. Membaca spesifikasi dan aturan permainan dengan teliti sebelum melakukan pemrograman sebaiknya dilakukan bersama-sama dengan anggota kelompok yang lain. Dengan demikian, setiap orang memiliki pemahaman yang sama dan sanggup untuk menyelesaikan tugas masing-masing sebaik-baiknya.
3. Pemrograman sebaiknya dilakukan secara bertahap dengan keadaan pikiran yang segar. Bertahap, maksudnya dilakukan sambil mencoba bagian per bagian. Ketika ada bagian yang tidak sesuai harapan, harus diperbaiki dahulu sebelum menumpuk di akhir. Apabila dalam pengerjaan merasa penat atau lelah, sebaiknya beristirahat sejenak.

5.3 Refleksi

Tugas besar ini memberikan banyak sekali pengalaman dan kebersamaan bagi kelompok kami. Kami mendapat ilmu baru tentang pemrograman berorientasi dalam bahasa Java, khususnya untuk permainan seperti *Galaxio*. Di sini pula kami belajar, bahwa pengaturan waktu dan komunikasi yang baik sangat diperlukan dalam membentuk sebuah tim yang tuntas mengerjakan tanggung jawabnya. Apabila satu orang saja tidak menjalankan tugas dengan benar, maka tidak ada hasil seindah ini.

Dengan keterbatasan pengetahuan dan keahlian, kami pun jadi harus berusaha sangat keras mengeksplor lebih dan lebih dalam lagi mengenai tugas ini, baik dari internet, hingga teman dan senior. Akhirnya, usaha kami tidak sia-sia. Tugas besar pertama kami di semester 4 terselesaikan dengan baik dan tepat waktu.

DAFTAR PUSTAKA

1. Munir, Rinaldi. 2021. *Algoritma Greedy (Bagian 1)*. Bahan Kuliah IF2211 Strategi Algoritma. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-*Greedy*-(2021)-Bag1.pdf pada tanggal 17 Februari 2023.
2. Munir, Rinaldi. 2021. *Algoritma Greedy (Bagian 2)*. Bahan Kuliah IF2211 Strategi Algoritma. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-*Greedy*-(2021)-Bag2.pdf pada tanggal 17 Februari 2023.
3. Munir, Rinaldi. 2021. *Algoritma Greedy (Bagian 3)*. Bahan Kuliah IF2211 Strategi Algoritma. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-*Greedy*-(2022)-Bag3.pdf pada tanggal 17 Februari 2023.
4. ThePheonixGuy. 30 Juni 2021. *The Game*. Terakhir diakses pada tanggal 17 Februari 2023 dari https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md#torpedo-salvo di https://github.com/EntelectChallenge/2021-Galaxio .

LAMPIRAN

Link Repository GitHub:

https://github.com/haziqam/Tubes1_subsetRPL10

Link video Youtube:

https://youtu.be/Nn9tJUn4Uj8