

LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA
Pemanfaatan Algoritma Greedy dalam Permainan “Galaxio”



Disusun oleh:

13521086 Ariel Jovananda
13521158 Muhammad Dhiwaul Akbar
13521167 Irgiansyah Mondo

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2021/2022

DAFTAR ISI

DAFTAR ISI	1
BAB 1 DESKRIPSI TUGAS	2
BAB 2 LANDASAN TEORI	5
A. Algoritma Greedy	5
B. Cara Kerja Program	6
BAB 3 APLIKASI STRATEGI GREEDY	10
A. Mapping Persoalan Permainan “Galaxio” Menjadi Elemen-Elemen Algoritma Greedy	10
B. Eksplorasi Alternatif Solusi Greedy	11
C. Analisis Efisiensi dan Efektivitas Solusi Greedy	13
D. Strategi Greedy Yang Dipilih	14
BAB 4 IMPLEMENTASI DAN PENGUJIAN	17
A. Implementasi Algoritma Greedy	17
B. Struktur Data Yang Digunakan	19
C. Analisis Solusi Algoritma Greedy	21
BAB 5 KESIMPULAN DAN SARAN	26
A. Kesimpulan	26
B. Saran	26
DAFTAR PUSTAKA	27
LAMPIRAN	28

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.

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Galaxio*. *Game engine* dapat diperoleh pada laman berikut:

<https://github.com/EntelectChallenge/2021-Galaxio>

Tugas mahasiswa adalah mengimplementasikan bot kapal dalam permainan *Galaxio* dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini:

<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2>

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 disebarluaskan pada peta dengan ukuran 3 dan dapat dikonsumsi 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 menembakkannya dapat meledakannya dan memberi *damage* ke *player* yang berada dalam zona. Area ledakan akan berubah menjadi *gas cloud*.
8. *Player* dapat meluncurkan *teleporter* pada suatu arah di peta. *Teleporter* tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat *teleporter* tersebut. Harga setiap peluncuran *teleporter* adalah 20. Setiap 100 tick player akan mendapatkan 1 *teleporter* dengan jumlah maximum adalah 10.
9. Ketika kapal *player* bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi oleh kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih besar hingga ukuran maximum dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua dari kapal tersebut lawan arah.

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

Adapun peraturan yang lebih lengkap dari permainan *Galaxio*, dapat dilihat pada laman :

<https://github.com/EntelectChallenge/2021-Galaxio/blob/develop/game-engine/game-rules.md>

BAB 2

LANDASAN TEORI

A. Algoritma *Greedy*

Metode desain populer yang dapat digunakan untuk mengoptimalkan solusi masalah adalah algoritma *greedy*. Algoritma ini mencoba menyelesaikan masalah dalam serangkaian fase, yang masing-masing fase mengeksplorasi solusi potensial dari solusi parsial yang telah ditemukan. Fitur penting algoritma ini adalah bahwa pilihan yang dibuat pada setiap tahap harus layak, optimal secara lokal, dan tidak dapat dibatalkan. Suatu pilihan harus layak dalam arti harus mengikuti batasan tantangan yang diberikan. Secara lokal optimal berarti bahwa setiap pilihan adalah yang terbaik yang saat ini dapat diakses secara lokal dari semua pilihan yang dapat dibuat pada saat itu.

Sesuai dengan namanya, *greedy*, yang berarti rakus, algoritma memilih solusi yang paling “rakus” untuk setiap jalur dihadapi dengan harapan akan menghasilkan solusi yang ideal secara global untuk masalah yang dihadapi. Dengan kata lain, karena algoritma hanya mempertimbangkan pengoptimalan untuk setiap masalah unik, tidak menyeluruh, algoritma *greedy* tidak selalu memberikan solusi optimal, hanya solusi suboptimal. Ini karena keputusan yang dibuat hanya untuk menjadi yang terbaik pada saat yang bersangkutan.

Terdapat beberapa elemen yang membantu menentukan suatu algoritma greedy:

1. Himpunan kandidat, C: Mengidentifikasi kandidat yang akan dipilih untuk setiap posisi.
2. Himpunan solusi, S: Berisi kandidat yang sudah terpilih.
3. Fungsi solusi: Untuk menentukan apakah kandidat yang terpilih sudah memberikan solusi.
4. Fungi seleksi/selection function: memilih kandidat menggunakan strategi *greedy* saat ini.

5. Fungsi kelayakan/feasibility function: Mempertimbangkan apakah kandidat yang terpilih dapat masuk ke himpunan solusi, dan apakah kandidat tertentu memenuhi syarat atau tidak.

6. Fungsi objektif: memaksimumkan atau meminimumkan.

Berikut adalah skema umum-nya algoritma *greedy*:

```

function greedy( $C : \text{himpunan\_kandidat}$ )  $\rightarrow$   $\text{himpunan\_solusi}$ 
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
   $x : \text{kandidat}$ 
   $S : \text{himpunan\_solusi}$ 

Algoritma:
   $S \leftarrow \{\}$  {inisialisasi  $S$  dengan kosong}
  while (not SOLUSI( $S$ )) and ( $C \neq \{\}$ ) do
     $x \leftarrow \text{SELEKSI}(C)$  {pilih sebuah kandidat dari  $C$ }
     $C \leftarrow C - \{x\}$  {buang  $x$  dari  $C$  karena sudah dipilih}
    if LAYAK( $S \cup \{x\}$ ) then { $x$  memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi}
       $S \leftarrow S \cup \{x\}$  {masukkan  $x$  ke dalam himpunan solusi}
    endif
  endwhile
  {SOLUSI( $S$ ) or  $C = \{\}$ }

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

```

Gambar 2.1 Skema umum algoritma *greedy*

B. Cara Kerja Program

Permainan “*Galaxio*” memiliki beberapa komponen. Komponen-komponen tersebut berbentuk seperti ini:

```
└── engine-publish
└── logger-publish
└── reference-bot-publish
└── runner-publish
└── starter-bots
└── visualiser
└── building-a-bot.md
└── README.md
└── run.sh
```

Gambar 2.2 Komponen-komponen permainan “*Galaxio*”

Ada beberapa komponen yang perlu diketahui untuk memahami cara kerja game ini, mereka adalah:

- Engine: Engine merupakan komponen yang berperan dalam mengimplementasikan logic dan rules game.
- Runner: Runner merupakan komponen yang berperan dalam menggelar sebuah match serta menghubungkan bot dengan engine.
- Logger: Logger merupakan komponen yang berperan untuk mencatat log permainan sehingga kita dapat mengetahui hasil permainan. Log juga akan digunakan sebagai input dari visualizer

Garis besar cara kerja program game “*Galaxio*” adalah sebagai berikut:

1. Runner –saat dijalankan– akan meng-host sebuah match pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:5000.
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner. Setelah terkoneksi, Engine akan menunggu sampai bot-bot pemain terkoneksi ke runner.
3. Logger juga melakukan hal yang sama, yaitu melakukan koneksi dengan runner.

4. Pada titik ini, dibutuhkan beberapa bot untuk melakukan koneksi dengan runner agar match dapat dimulai. Jumlah bot dalam satu pertandingan didefinisikan pada atribut BotCount yang dimiliki file JSON "appsettings.json". File tersebut terdapat di dalam folder "runner-publish" dan "engine-publish".
5. Permainan akan dimulai saat jumlah bot yang terkoneksi sudah sesuai dengan konfigurasi.
6. Bot yang terkoneksi akan mendengarkan event-event dari runner. Salah satu event yang paling penting adalah RecieveGameState karena memberikan status game.
7. Bot juga mengirim event kepada runner yang berisi aksi bot.
8. Permainan akan berlangsung sampai selesai. Setelah selesai, akan terbuat dua file json yang berisi kronologi match.

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

1. Lakukan konfigurasi jumlah bot yang ingin dimainkan pada file JSON "appsettings.json" dalam folder "runner-publish" dan "engine-publish"
2. Buka terminal baru pada folder runner-publish.
3. Jalankan runner menggunakan perintah "dotnet GameRunner.dll"
4. Buka terminal baru pada folder engine-publish
5. Jalankan engine menggunakan perintah "dotnet Engine.dll"
6. Buka terminal baru pada folder logger-publish
7. Jalankan engine menggunakan perintah "dotnet Logger.dll"

8. Jalankan seluruh bot yang ingin dimainkan
9. Setelah permainan selesai, riwayat permainan akan tersimpan pada 2 file JSON “*GameStateLog_{Timestamp}*” dalam folder “logger-publish”. Kedua file tersebut diantaranya *GameComplete* (hasil akhir dari permainan) dan proses dalam permainan tersebut.

Bot yang berada di dalam permainan “*Galaxio*” bekerja dalam *tick*. *Tick* merupakan suatu unit untuk mengukur waktu dalam permainan ini. Jadi setiap *tick* yang berjalan dalam permainan ini, *bot* akan melakukan kalkulasi dan melakukan pilihan dari sejumlah pilihan yang ada. Setiap pilihan, *bot* memiliki kondisi yang harus dipenuhi agar pilihan tersebut bisa dipilih agar *bot* bisa tahu untuk memilih pilihan apa di situasi yang berbeda-beda.. Dalam kondisi-kondisi ini dimana kita bisa mengimplementasi algoritma *greedy* kami.

Di dalam komponen-komponen permainan ini terdapat map yang bernama *Services* di dalam map ini terdapat program yang bernama *botService.java*, di dalam program ini ada fungsi yang bernama *computeNextPlayerAction*. Implementasi untuk algoritma *greedy* dapat dilakukan dalam fungsi tersebut menggunakan fungsi-fungsi dan objek-objek yang sudah ada dari program permainan “*Galaxio*”.

BAB 3

APLIKASI STRATEGI GREEDY

A. *Mapping* Persoalan Permainan “Galaxio” Menjadi Elemen-Elemen Algoritma *Greedy*

Konsep permainan ini adalah *last man standing*, seperti judulnya, untuk memenangkan permainan ini *bot* kami harus menjadi *bot* terakhir dalam permainan tersebut. *Bot* bisa memakan/menghindar/membela/menyerang *bot* lain. Maka dari itu, strategi kami sangat fokus dalam kondisi *bot* kami, dan *bot* lain yang berada dalam jarak dekat dengan *bot* kami.

Dalam *mapping* persoalan permainan “Galaxio”, diuraikan elemen-elemen algoritma *greedy* sebagai berikut:

1. Himpunan Kandidat:

{FORWARD, STOP, STARTAFTERBURNER, STOPAFTERBURNER,
FIRETORPEDOES, FIRESUPERNOVA, DETONATESUPERNOVA,
FIRETELEPORT, TELEPORT, ACTIVATESHIELD}

2. Himpunan Solusi:

Pilihan yang valid untuk setiap *tick* yang akan dijalankan oleh *bot*.

3. Fungsi Solusi:

Memeriksa apakah *command* yang dilakukan oleh *bot* bisa membuat *bot* menjadi *last man standing*.

4. Fungsi Seleksi:

Pilih *command* yang bisa menambahkan ukuran *bot*, menyerang dan mengurangi ukuran *bot* musuh, atau pergi dari *bot* musuh.

5. Fungsi Kelayakan:

Pilihan *command* tersebut adalah *command* yang paling valid dalam situasi itu. Misal, menyerang hanya apabila ukuran *bot* lebih besar dari ukuran *bot* musuh atau jarak *bot*

sudah cukup aman untuk menyerang dari jauh, jadi tidak menyerang jika ukuran *bot* lebih kecil dan jarak belum cukup jauh.

6. Fungsi Objektif:

Memenangkan permainan dengan *bot* menjadi *bot* yang terakhir dalam permainan.

B. Eksplorasi Alternatif Solusi *Greedy*

Berdasarkan *command* serta *state* yang berada dalam permainan “*Galaxio*”, ada beberapa alternatif solusi yang kami pikirkan untuk memenangkan permainan ini.

1. Strategi *memaksimalkan* ukuran *bot*

Sesuai dengan namanya, pada strategi ini, *bot* hanya fokus untuk memaksimalkan ukurannya dengan makan semua makanan yang ada di sekitarnya. Strategi ini bertujuan untuk menghasil ukuran *bot* yang lebih besar dibanding *bot* musuh yang lain, karena dalam permainan ini, apabila *bot* anda hanya bisa memakan *bot* musuh yang lain apabila ukuran *bot* anda lebih besar.

Strategi ini dapat diimplementasikan dengan mencari arah makanan terdekat, dan melakukan *command FORWARD*. Dengan mengulangi implementasi ini di setiap *tick* permainan, *bot* akan terus mengarah kepada makanan terdekat, dan bergerak maju kepada arah tersebut.

2. Strategi *memaksimalkan* ukuran *bot* dan menyerang *bot* lain

Pada strategi ini, kita mengkombinasikan strategi sebelumnya dengan strategi menyerang *bot* musuh yang lain. Strategi ini bertujuan untuk memperbesar ukuran *bot* dan sekaligus menyerang *bot* musuh yang berada di sekitar, karena dalam permainan ini apabila *bot* memakan *bot* musuh, *bot* akan bertambah ukuran dan sekaligus mengurangi jumlah pemain yang berada di permainan, sehingga membawa *bot* makin dekat kepada tujuan akhirnya.

Strategi ini dapat diimplementasikan mirip dengan strategi sebelumnya, namun ada tambahan implementasi seperti memeriksa daerah sekitar untuk *bot* musuh, apabila ada

bot musuh yang lain, dekatkan *bot* tersebut dengan mendapatkan arahnya, dan melakukan *command FORWARD*. Apabila sudah cukup dekat lakukan *command STARTAFTERBURNER* agar bisa menutup sisa jarak dengan cepat, sekaligus lakukan *command FIRETORPEDOES* untuk mengurangi ukuran *bot* musuh agar makin lebih mudah untuk dimakan.

3. Strategi *memaksimalkan* ukuran *bot*, menyerang *bot* lain, dan mekanisme pergi

Pada strategi ini, kita mengkombinasikan strategi-strategi sebelumnya dengan mekanisme pergi. Strategi ini bertujuan untuk memperbesar ukuran *bot*, tahu kapan menyerang *bot* musuh lain, dan kapan untuk pergi menjauh dari *bot* musuh yang berada di sekitar, karena dalam permainan ini, selain mencoba untuk memperbesar ukuran *bot* dan memakan *bot* musuh yang lain, *bot* juga harus mampu mengenal situasi dimana kondisi lokal tidak memungkinkan untuk *bot* menang melawan *bot* musuh. Apabila *bot* berada di dalam situasi tidak menguntungkan tersebut, *bot* akan melakukan mekanisme pergi.

Strategi ini dapat diimplementasikan mirip dengan strategi-strategi sebelumnya, namun ada beberapa kondisi tambahan untuk menyerang, dan penambahan mekanisme pergi. Sebelumnya *bot* hanya makan dan menyerang. Sekarang *bot* mempunyai kondisi untuk selalu memeriksa ukurannya sendiri dan jarak dia kepada musuh terdekat, sampai ukuran tertentu *bot* akan terus makan, apabila sudah melewati ukuran tersebut, *bot* akan mencari musuh yang bisa dimakan. Selain itu, apabila ada musuh yang berada di sekitar *bot*, *bot* akan memeriksa musuh tersebut, apabila musuh ukurannya lebih besar *bot* akan pergi dengan mendapatkan arah yang sebalik dari musuh tersebut dan melakukan *command FORWARD*, apabila sebaliknya *bot* akan coba untuk menyerang.

Ada juga tambahan implementasi menyerang, apabila *bot* sudah pergi cukup jauh, *bot* akan mencari musuh terdekat dan melakukan *command FIRETORPEDOES* sekaligus memeriksa ukuran dan jarak *bot* musuh tersebut yang sedang di tembak oleh *torpedoes*. Apabila sudah tidak dekat, *bot* akan kembali mencari makanan, apabila masih dekat dan ukurannya sudah berkurang, *bot* akan mencoba untuk menyerang kembali, namun apabila masih dekat dan ukuran musuh masih lebih besar, *bot* akan tetap pergi menjauh.

C. Analisis Efisiensi dan Efektivitas Solusi *Greedy*

1. Strategi *memaksimalkan* ukuran *bot*

Pada strategi ini, ukuran *bot* ingin kami maksimalkan dengan memakan semua makanan yang ada di sekitar *bot*. Efektivitas strategi ini cukup sesuai dengan tujuan untuk memenangkan permainan, karena untuk memakan *bot* musuh yang lain, ukuran *bot* kami harus lebih besar dari ukuran mereka. Namun banyak situasi dimana disekitar *bot* tidak ada makanan yang cukup untuk mengalahkan ukuran-ukuran *bot* lain. Ada juga situasi lain dimana apabila kita hanya memakan dan tidak mekanisme pergi dari *bot* lain, *bot* menjadi mangsa yang mudah dimakan. Selain itu, dengan hanya memakan dan tidak menyerang atau pergi, *bot* memiliki kecenderungan untuk keluar dari peta permainan, akhirnya *bot* akan musnah dengan sendirinya. Strategi ini juga tidak efisien, karena *bot* hanya fokus memakan, untuk memenangkan permainan perlu waktu yang cukup banyak agar *bot* bisa menang ukuran dengan *bot* musuh yang lain. Dengan hal-hal tersebut, strategi ini belum cukup efektif untuk memenangkan permainan.

2. Strategi *memaksimalkan* ukuran *bot* dan menyerang *bot* lain

Pada strategi ini, *bot* akan melakukan hal yang sama seperti strategi sebelumnya yaitu, coba untuk memaksimalkan ukuran *bot* dengan memakan makanan yang berada di sekitar *bot*. Tetapi selain makan, *bot* juga bisa menyerang musuh yang sekaligus juga berada di sekitarnya. Mulai dari mendekati *bot* musuh tersebut, apabila sudah cukup dekat, aktifkan *AFTERSURNER* dan tembakkan *TORPEDOES*. Efektivitas strategi *greedy* ini adalah, karena tujuannya adalah menjadi *bot* terakhir dalam permainan, dengan memaksimalkan ukuran dengan makanan, dan mencoba untuk menyerang musuh yang berada dalam sekitar *bot* akan lebih efektif, karena dua pilihan tersebut apabila berhasil tidak hanya menambahkan ukuran *bot*, tetapi juga mengurangi jumlah pemain yang berada dalam permainan. Secara efisien, strategi ini lebih baik dari strategi sebelumnya, karena selain fokus makan, *bot* juga bisa menyerang lawan dengan harapan bisa memakan *bot* musuh tersebut, jadi *bot* bisa mencapai tujuan akhirnya dengan lebih cepat. Namun strategi ini belum cukup efektif, *bot* belum bisa mengenal kondisi dimana *bot* harus pergi dari situasi dimana *bot* tidak bisa menang melawan *bot* lain. Tingkat

keselamatan *bot* masih minim, karena hal tersebut, untuk mencapai tujuan akhir akan lebih sulit untuk *bot* apabila ia tidak bisa menyelamatkan dirinya sendiri.

3. Strategi *memaksimalkan* ukuran *bot*, menyerang *bot* lain, dan mekanisme pergi

Pada strategi ini, *bot* akan melakukan hal-hal yang sama dari strategi-strategi sebelumnya tetapi ditambahkan mekanisme pergi. Tanpa banyak detail, dalam strategi ini, *bot* ditambahkan implementasi untuk mengenal situasi dimana *bot* ada dalam situasi yang tidak menguntungkan, apabila *bot* berada dalam situasi tersebut, *bot* akan pergi menjauh dari situasi tersebut. Misal apabila ada *bot* musuh yang lebih besar, dan menutupi jarak dengan *bot*, maka *bot* akan pergi menjauh dengan arah yang sebalik kepada *bot* musuh tersebut. Strategi ini cukup efektif untuk memenangkan permainan, karena sekarang *bot* bisa mempertahankan keselamatannya sendiri, *bot* mampu mengenal situasi kapan ia harus makan, kapan harus menyerang, dan kapan harus pergi dari situasi. Secara efisien, strategi ini lebih baik dibanding strategi-strategi sebelumnya, karena selain mencoba untuk menyerang musuh, *bot* juga ada mekanisme pergi yang bisa mempertahankan dirinya untuk tahap-tahap akhir permainan, dengan harapan bisa memenangkan tahap-tahap akhir tersebut. Namun kekurangan dari strategi ini adalah, kadang *bot* masih bisa mengenal situasi dengan salah, selain itu, *bot* juga tidak benar-benar memanfaatkan semua *command* yang ada dalam permainan.

D. Strategi *Greedy* Yang Dipilih

Strategi yang akhirnya dipilih adalah “Strategi *memaksimalkan* ukuran *bot*, menyerang *bot* lain, dan mekanisme pergi”, pertimbangan untuk memilih strategi ini adalah, *bot* harus mampu mengenal situasi, kapan ia harus makan, kapan menyerang dan kapan pergi. Strategi-strategi sebelumnya tidak memenuhi kemampuan-kemampuan tersebut. Selain itu, pertimbangan lokal lebih baik daripada strategi-strategi sebelumnya, karena selain mencoba untuk memilih pilihan lokal yang paling efektif untuk menambahkan ukuran *bot*, *bot* juga mampu memilih pilihan lokal yang paling efektif agar ukurannya tidak makin mengecil atau dimakan oleh *bot* musuh lain. Dengan semua yang dikatakan, berikut adalah rincian implementasi algoritma *greedy* yang kami lakukan:

1. Untuk langkah-langkah pertama, *bot* akan mencari makan dulu sampai memenuhi suatu kondisi/konstanta ukuran (apabila ukuran *bot* < 40), atau *bot* akan mencari makan sekitar jika tidak ada *bot* yang berada di sekitar daerahnya. Daerah tersebut diperiksa menggunakan konstanta (jarak *bot* dengan musuh terdekat > radius ukuran *bot* + radius ukuran musuh + 200), jadi apabila tidak ada *bot* sampai suatu konstanta radius *bot* akan tetap mencari makan sampai memenuhi kondisi/konstanta ukuran. *Bot* mencari makan dengan mencari arah makanan terdekat, dan melakukan *command FORWARD* kepada arah tersebut.
2. Langkah selanjutnya adalah, apabila *bot* sudah memenuhi kondisi/konstanta ukuran, *bot* akan mencari musuh yang terdekat dengannya. Apabila *bot* lebih besar ia akan mencoba untuk menyerang musuh tersebut dengan berbagai tahap (tahapan akan dijelaskan lebih di bagian 3), namun apabila ukuran musuh terdekat lebih besar daripada ukuran *bot*, maka akan ada dua kondisi yang diperiksa. Kondisi pertama adalah untuk memeriksa jarak *bot* kepada musuh terdekat, apabila jaraknya cukup jauh (jarak *bot* dengan musuh terdekat > radius ukuran *bot* + radius ukuran musuh + 70), *bot* justru akan menyerang musuh yang lebih besar dengan mendapatkan arah musuhnya dan melakukan *command FIRETORPEDOES*, dengan tujuan untuk mengurangi ukuran musuh, dan *bot* juga sekaligus memeriksa ukuran *bot* musuh per setiap *tick*, apabila ukuran musuh berhasil menjadi lebih kecil, *bot* akan menyerang kembali, apabila tidak *bot* akan tetap menembak atau pergi dari situasi. Tetapi apabila jarak musuh dari awal masih terlalu dekat (jarak *bot* dengan musuh terdekat < radius ukuran *bot* + radius ukuran musuh + 70) *bot* akan pergi menjauh dengan mendapatkan arah yang sebalik dari arah musuh dan melakukan *command FORWARD* pada arah tersebut.
3. Langkah yang akan diambil *bot* apabila musuh terdekat dari *bot* lebih kecil adalah untuk menyerang *bot* musuh tersebut. Ada beberapa tahap yang dilakukan *bot* ketika ingin menyerang, pertama periksa jarak, apabila jarak *bot* dengan musuh terdekat masih jauh (jarak *bot* dengan musuh terdekat > radius ukuran *bot* + radius ukuran musuh + 70), kita ingin menutup jarak tersebut dengan mendapatkan arah *musuh* dan melakukan *command FORWARD* kepada arah tersebut. Apabila jarak sudah cukup dekat (jarak *bot* dengan musuh terdekat < radius ukuran *bot* + radius ukuran musuh + 70) *bot* akan mendapatkan arah musuh dan melakukan *command FIRETORPEDOES*, tujuannya agar untuk semakin mengecilkan ukuran musuh agar lebih mudah di makan. Selanjutnya, apabila jarak *bot*

dengan musuh sudah semakin dekat (jarak *bot* dengan musuh terdekat < radius ukuran *bot* + radius ukuran musuh + 60) *bot* akan melakukan *command STARTAFTERBURNER* kepada arah musuh, dengan tujuan untuk menutup jarak dengan lebih cepat dan bisa memakan musuh tersebut.

4. *Command AFTERBURNER* memakan ukuran *bot*, jadi ada juga kondisi untuk melakukan *command STOPAFTERBURNER*. Kondisinya adalah apabila jarak sudah melebihi kondisi untuk melakukan *command STARTAFTERBURNER* (jarak *bot* dengan musuh terdekat > radius ukuran *bot* + radius ukuran musuh + 60), *bot* akan melakukan *command STOPAFTERBURNER* karena jarak masih terlalu jauh untuk menyalakannya, dan agar ukuran *bot* tidak semakin mengecil dan memposisikan *bot* dalam situasi yang tidak menguntungkan.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma *Greedy*

*Tulisan yang berada di dalam kurung kurawal berwarna merah ({}) adalah catatan untuk pembaca agar mudah untuk memahami program.

```
procedure computeNextPlayerAction(PlayerAction playerAction)
{I.S. playerAction yang bertipe PlayerAction}
{F.S. playerAction}
```

KAMUS LOKAL

foodList : List of gameObjects
playerList: List of gameObjects

ALGORITMA

```
foodList ← gameState.getGameObjects().stream().filter(item -> item.getGameObjectType() ==
ObjectTypes.FOOD).sorted(Comparator.comparing(item -> getDistanceBetween(bot,
item))).collect(Collectors.toList()) {Menginisialisasi list bernama foodList, elemen merupakan food dan diurutkan berdasarkan makanan terdekat}
```

```
if (not gameState.getGameObjects().isEmpty()) then
    var playerList ←
gameState.getPlayerGameObjects().stream().sorted(Comparator.comparing(item ->
getDistanceBetween(bot, item))).collect(Collectors.toList()) {Menginisialisasi list bernama playerList,
elemen merupakan pemain dan diurutkan dengan berdasarkan jarak terdekat}
```

```
    var nearestPlayer ← playerList.get(1) {Mengambil index kedua dari playerList, index kedua merupakan pemain yang terdekat dari bot. Memakai index kedua dan bukan pertama, karena index pertama merupakan bot itu sendiri}
```

```
    var nearestFood ← foodList.get(0) {Mengambil index pertama dari playerList, index kedua merupakan pemain yang terdekat dari bot. Memakai index kedua dan bukan pertama, karena index pertama merupakan bot itu sendiri}
```

```
    var sizeDiffOfNP ← bot.size - nearestPlayer.size
    if (bot.size < 40 and (getDistanceBetween(bot, nearestPlayer) > bot.size + nearestPlayer.size
+ 200)) then {Memeriksa apakah ukuran bot kurang dari 40 dan jarak terdekat bot musuh masih lebih
```

besar dari ukuran *bot* ditambah ukuran musuh terdekat ditambah 200, apabila kedua kondisi terpenuhi cari makanan terdekat}

```

output("kemek ngab")
playerAction.heading ← getHeadingBetween(nearestFood)
playerAction.action ← PlayerActions.FORWARD
else
    playerAction.heading ← getHeadingBetween(nearestPlayer);
    if(bot.size < nearestPlayer.size) then {Memeriksa apakah ukuran bot kurang dari musuh terdekat, jika iya masuk ke dalam kondisi}
        if(getDistanceBetween(bot, nearestPlayer) > bot.size + nearestPlayer.size + 70) then
            {Memeriksa apakah jarak bot dengan musuh sudah lebih besar dari ukuran bot ditambah ukuran musuh terdekat ditambah 70, jika iya lakukan command FIRETORPEDOES}
                output("coba tembak")
                playerAction.heading ← getHeadingBetween(nearestPlayer)
                playerAction.action ← PlayerActions.FIRETORPEDOES
            else {Jika jarak masih lebih kecil dari ukuran bot ditambah ukuran musuh terdekat ditambah 100, maka bot akan mengambil arah yang sebalik dari musuh, dan melakukan command FORWARD}
                output("kabur ngab")
                playerAction.heading ← -getHeadingBetween(nearestPlayer)
                playerAction.action ← PlayerActions.FORWARD

        else {jika bot lebih besar dari musuh terdekat, masuk kedalam kondisi ini}
            if(getDistanceBetween(bot, nearestPlayer) > bot.size + nearestPlayer.size + 70) then
                {Memeriksa jarak terdekat bot musuh apakah masih lebih besar dari ukuran bot ditambah ukuran musuh terdekat ditambah 70, jika iya bot akan mengambil arah musuh terdekat dan melakukan command FORWARD dengan tujuan untuk menutup jarak antara bot dengan musuh}
                output("deketin")
                playerAction.heading ← getHeadingBetween(nearestPlayer)
                playerAction.action ← PlayerActions.FORWARD
            else {Jika jarak sudah kurang dari ukuran bot ditambah ukuran musuh terdekat ditambah 70, maka akan masuk ke dalam kondisi ini }
                if(getDistanceBetween(bot, nearestPlayer) < bot.size + nearestPlayer.size + 70) then
                    {Memeriksa jarak terdekat bot musuh apakah lebih kecil dari ukuran bot ditambah ukuran musuh terdekat ditambah 70, jika iya bot akan mengambil arah musuh terdekat dan melakukan command FIRETORPEDOES dengan tujuan untuk makin mengurangi ukuran musuh agar lebih mudah untuk dimakan}
                    output("tembak ngares")
                    playerAction.heading ← getHeadingBetween(nearestPlayer)
                    playerAction.action ← PlayerActions.FIRETORPEDOES
                    if(getDistanceBetween(bot, nearestPlayer) < bot.size + nearestPlayer.size + 50)
then {Memeriksa jarak terdekat bot musuh apakah lebih kecil dari ukuran bot ditambah ukuran musuh terdekat ditambah 50, jika iya bot akan mengambil arah musuh terdekat dan melakukan command STARTAFTERBURNER dengan tujuan untuk menutup jarak antara bot dengan musuh dengan lebih cepat}

        output("serbu ngares")
        playerAction.heading ← getHeadingBetween(nearestPlayer)
        playerAction.action ← PlayerActions.STARTAFTERBURNER
    else {Kondisi ini berfungsi apabila bot sempat masuk ke dalam kondisi yang melakukan command STARTAFTERBURNER, tujuan dari kondisi ini adalah apabila bot gagal menutup

```

```
jarak dengan menggunakan command STARTAFTERBURNER, dan jarak musuh menjauh dari ukuran  
bot ditambah ukuran musuh terdekat ditambah 50, maka lakukan command STOPAFTERBURNER}  
    output("selaw dulu")  
    playerAction.heading ← getHeadingBetween(nearestPlayer)  
    playerAction.action ← PlayerActions.STOPAFTERBURNER  
}  
}  
}  
}  
}  
}  
this.playerAction ← playerAction  
}  
}
```

B. Struktur Data Yang Digunakan

Struktur data yang digunakan dalam game Galaxio berbasis kelas. Paket Pemula Game Galaxio Entelect menawarkan beberapa kategori. Kelas -kelasnya termasuk GameObject, GameState, PlayerAction, Position, World, Bot, dan BotService.

1. Class GameObject

Kelas ini merupakan kelas yang menyimpan objek-objek yang digunakan dalam permainan. Setiap objek memiliki atributnya masing-masing (id, size, speed, currentHeading, position, gameObjectType, position, gameObjectType, effects, TorpedoSalvoCount, SupernovaAvailable, TeleportCount, dan ShieldCount). Dengan semua yang dikatakan, atribut “gameObjectType” mempunyai fungsi untuk menunjukkan jenis objek.

2. Class GameState

Class GameState mempunyai fungsi untuk menunjukkan keadaan permainan saat ini. Didalam kelas ini terdapat tiga atribut yaitu world, gameObjects, dan playerGameObjects.

3. Class PlayerAction

Kelas ini secara garis besar fungsinya adalah untuk mendeskripsikan aksi dari pemain. Kelas ini mempunyai tiga atribut yaitu playerId, PlayerActions, dan heading, fungsi dari masing-masing atribut tersebut adalah untuk menunjukkan pemain, untuk menggambarkan aksi pemain, dan untuk menunjukkan arah gerak pemain.

4. Position

Kelas ini sesuai namanya adalah untuk menunjukkan posisi pemain dalam koordinat peta. Ada dua atribut dalam kelas ini yaitu x (absis) dan y (ordinat).

5. World

Kelas world dibuat untuk menggambarkan daerah atau area valid yang ada didalam permainan . kelas ini terdiri 3 atribut ,yaitu CenterPoint (koordinat titik tengah area permainan) , radius (radius area permainan dihitung dari titik tengah atau CenterPoint) dan currentTick (menunjukkan waktu atau tick permainan) .

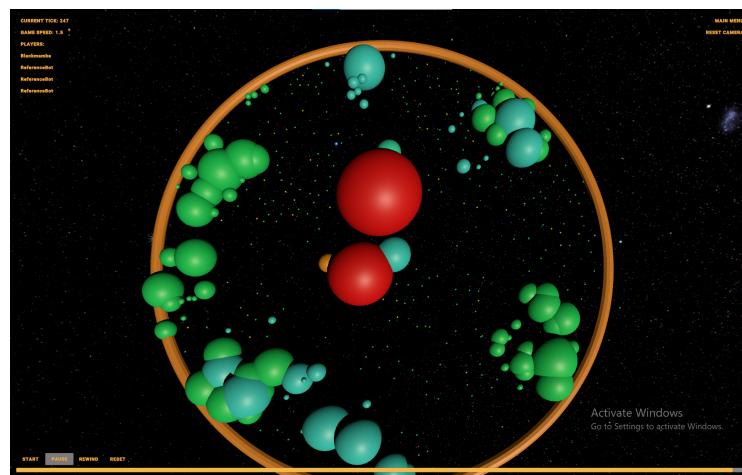
6. BotService

Kelas ini berisikan method-method untuk menjalankan algoritma pada bot. Method-method pada BotService bersifat umum (tidak bersifat spesifik seperti bergerak ,menyerang , maupun bertahan) dalam artian , implementasinya dapat digunakan dalam berbagai aksi . salah satu method yang ada di BotService adalah method computeNextPlayerAction yang digunakan untuk menentukan aksi apa yang akan dilakukan oleh bot selanjutnya (aksi yang dimaksud bisa berupa penyerangan , pertahanan atau pergerakan mencari makanan) .

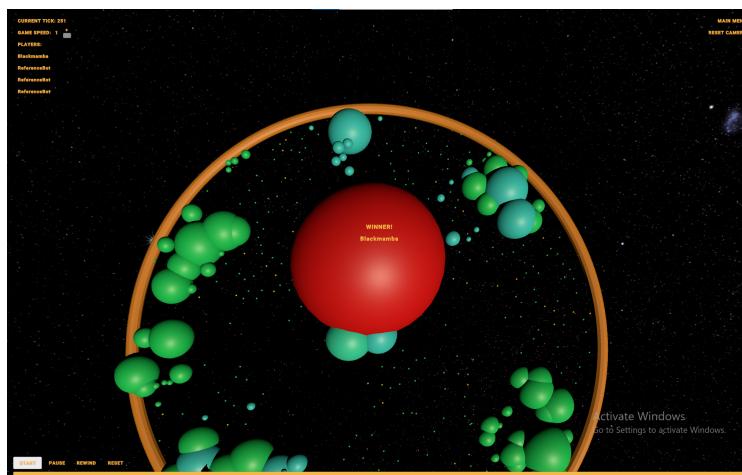
C. Analisis Solusi Algoritma *Greedy*

Untuk menjalankan permainan agar bisa dilakukan pengujian pertama kita harus masuk ke dalam *directory JavaBot*, ketika sudah berada di dalam *directory* tersebut ketik dalam *terminal* “*mvn clean package*” sehabis itu kembali ke *directory* dimana ada *RunGame.bat*. Jika berhasil melakukan tahap-tahap, permainan seharusnya berjalan.

1. Pengujian pertama



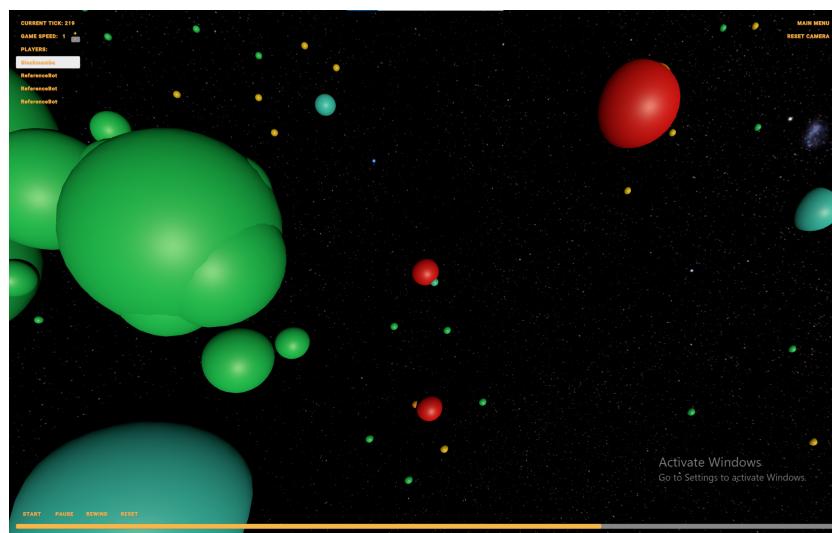
Gambar 3.1 *Bot* menyalakan *AFTERBURNER* untuk memakan musuh



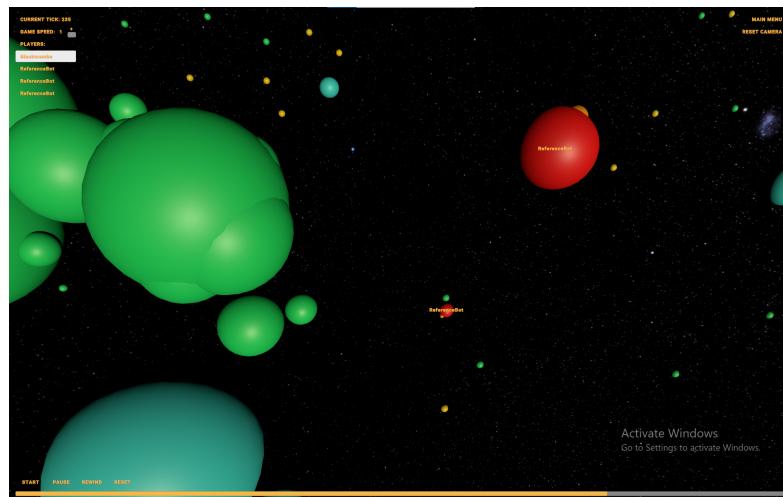
Gambar 3.2 *Bot* memenangkan permainan

Dalam pengujian ini *bot* mampu menang karena mampu membesarkan ukuran, setelah ukuran sudah cukup besar, *bot* mencari musuh terdekat, dan ketika musuh sudah cukup dekat, sekuens untuk nyerang dilakukan.

2. Pengujian kedua



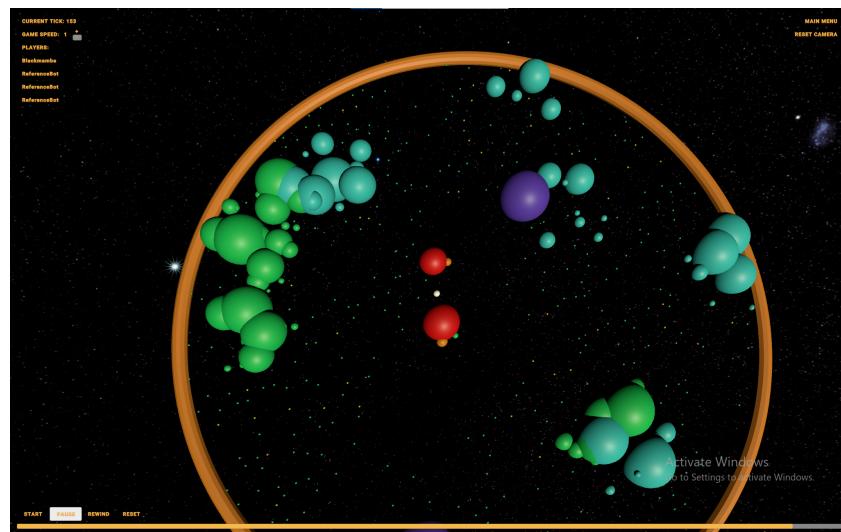
Gambar 3.3 Detik-detik sebelum *bot* menghabiskan ukurannya sendiri



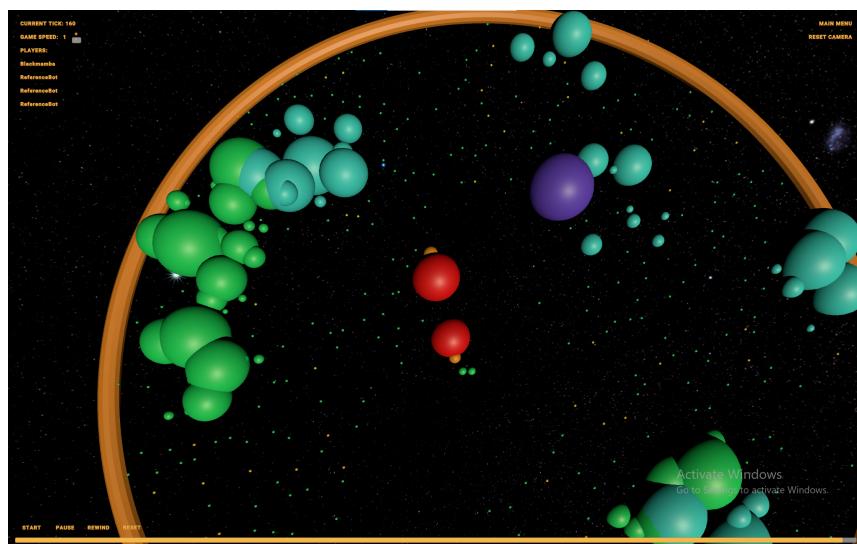
Gambar 3.4 Ukuran *bot* habis dan *bot* hilang

Dalam pengujian ini, terdapat sifat aneh yang berada di *AFTERBURNER*, *bot* memakan ukurannya sendiri sampai ukurannya habis dan *bot* akhirnya hilang dan permainan tidak dimenangkan.

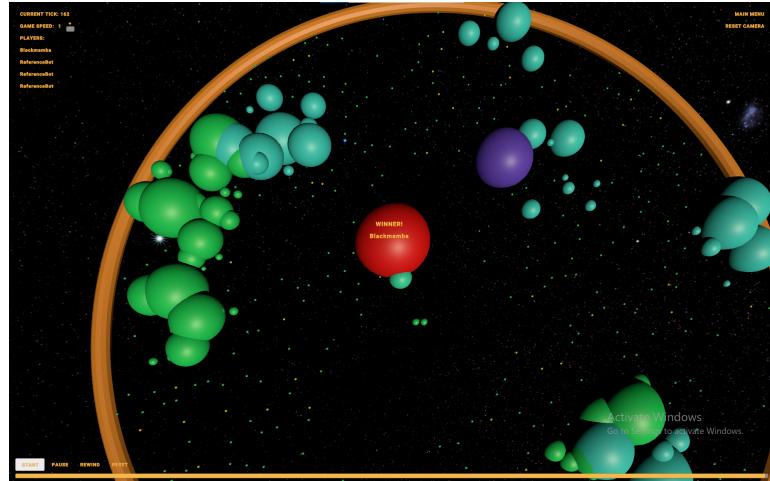
3. Pengujian ketiga



Gambar 3.5 *Bot* menyerang musuh yang lebih besar



Gambar 3.6 *Bot* mengenal situasi dan karena ukuran musuh sudah lebih kecil
Bot menyerang



Gambar 3.7 *Bot* berhasil memakan musuh menggunakan *AFTERBURNER*

Dalam pengujian ini, *bot* mampu menemukan mengenal situasi bahwa ada musuh yang lebih besar dan menyerang musuh tersebut. Selain itu setelah musuh sudah kalah ukuran, *bot* mampu menyerang musuh dan ketika jarak sudah dekat *bot* melakukan *command STARTAFTERBURNER* dan berhasil memakan musuhnya. Hasil akhir *bot* menang permainan.

Dari beberapa pengujian yang tercantum dalam laporan ini, dan pengujian yang tidak tercantum, bisa disimpulkan bahwa *bot* sudah bisa menemukan solusi yang optimal untuk beberapa situasi. *Bot* sudah cukup optimal untuk mencari makanan terdekat, kapan mencari musuh terdekat, kapan untuk pergi dari situasi, dan kapan untuk menyerang. Situasi optimal bisa dicapai apabila *bot* tidak mencari makanan sampai ujung peta, *bot* musuh yang di serang mempunyai selisih ukuran yang cukup besar, *bot* mampu memakan musuh ketika *AFTERBURNER* menyala, *bot* mampu menyerang *bot* yang lebih besar, dan *bot* pergi dari musuh yang tidak bertambah ukuran.

Namun ada beberapa situasi dimana *bot* tidak melakukan hal-hal yang optimal, dan kadang merugikan *bot* itu sendiri. Misal, *bot* bisa mencari makanan sampai keluar peta yang hasil akhirnya *bot* akan hilang dari permainan. *Bot* juga terkadang bisa melakukan osilasi ketika ada dua makanan yang memiliki jarak yang sama, hasilnya *bot* akan mutar bolak-balik, *bot* juga terkadang tidak bisa pergi dengan cepat dari musuh yang sedang mengejarnya namun *bot* musuh juga sekaligus bertambah ukuran.

Bot juga kadang bisa melakukan kesalahan ketika menyalakan *AFTERRBURNER* untuk menyerang, jadi misalkan *tick* sebelumnya musuh kecil dan *bot* menyerang, tetapi *tick* selanjutnya musuh menjadi lebih besar, *bot* tidak cukup cepat untuk mengenal situasi dan akhirnya termakan. Selain itu ada juga situasi dimana seperti pengujian kedua, ketika *bot* gagal untuk memakan musuh saat menyalakan *AFTERRBURNER*, *bot* tidak akan menemukan kondisi dimana ia harus mematikan *AFTERRBURNER* tersebut, hasilnya adalah *bot* akan menggunakan *AFTERRBURNER* sampai ia menghabiskan ukurannya sendiri.

Data yang belum diperiksa adalah bagaimana *bot* melawan *bot-bot* yang lain. Namun *bot* sudah cukup optimal untuk rata-rata situasi, dan mampu menang apabila melawan *reference bot* yang sudah tercantum dalam permainannya.

BAB 5

KESIMPULAN DAN SARAN

A. Kesimpulan

Kelompok Kami berhasil Mengimplementasikan greedy algorithm untuk membuat *bot Galaxio game* yang bisa memanfaatkan greedy agar *bot* mampu mempertimbangkan pilihan yang diprioritaskan. Dari penjelasan dari bab-bab sebelumnya, *bot* mampu memilih pilihan yang optimal untuk situasi nya, misal apabila ada situasi yang bisa menambahkan ukuran, *bot* akan makan terlebih dahulu, apabila ada musuh *bot* akan mencoba untuk menyerang agar jumlah pemain berkurang, dan apabila ada situasi dimana *bot* bisa termakan ia pergi dari situasi tersebut. Semua seutuhnya, semua pilihan *bot* bersifat optimal dalam lokal dengan harapan bisa menentukan hasil global yang optimal.

B. Saran

Dari tugas besar ini tentunya program yang kami buat tidak sempurna dan masih memiliki banyak keterbatasan, Untuk itu, kami mengharapkan kritik dan saran dari pihak-pihak yang akan mengevaluasi tugas ini. Tentunya *bot* kami masih bisa ditingkatkan untuk mendapat hasil yang lebih optimal dengan alternatif algoritma *greedy* yang lebih detail, atau algoritma lain yang lebih kompleks. Tidak hanya itu, kita sebagai kelompok juga perlu lebih banyak komunikasi agar pekerjaan tugas lebih mudah untuk dikerjakan bersama.

DAFTAR PUSTAKA

<https://github.com/EntelectChallenge/2021-Galaxio/releases/tag/2021.3.2> Diakses pada 16 Februari 2023

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)
Diakses pada 16 Februari 2023

LAMPIRAN

Link Repository:

https://github.com/arieljovananda88/Tubes1_Blackmamba.git

Video Kelompok:

https://itb-ac-id.zoom.us/rec/share/jf_2QmYlkpd1t60YUTQKaAy6oEliPjPD_akYsxOD0QV4Iidhl1kufO6v3hSU88Cp.9YMBLRByZBhaEcUt Passcode: ?p9P.da.