LAPORAN TUGAS BESAR 1 IF2211 STRATEGI ALGORITMA
PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN “GALAXIO”

Oleh:

Kelompok “muggle”

Muhammad Raihan Iqbal	13518134
Varraz Hazandra Abrar	13521020
Kartini Copa	13521026

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

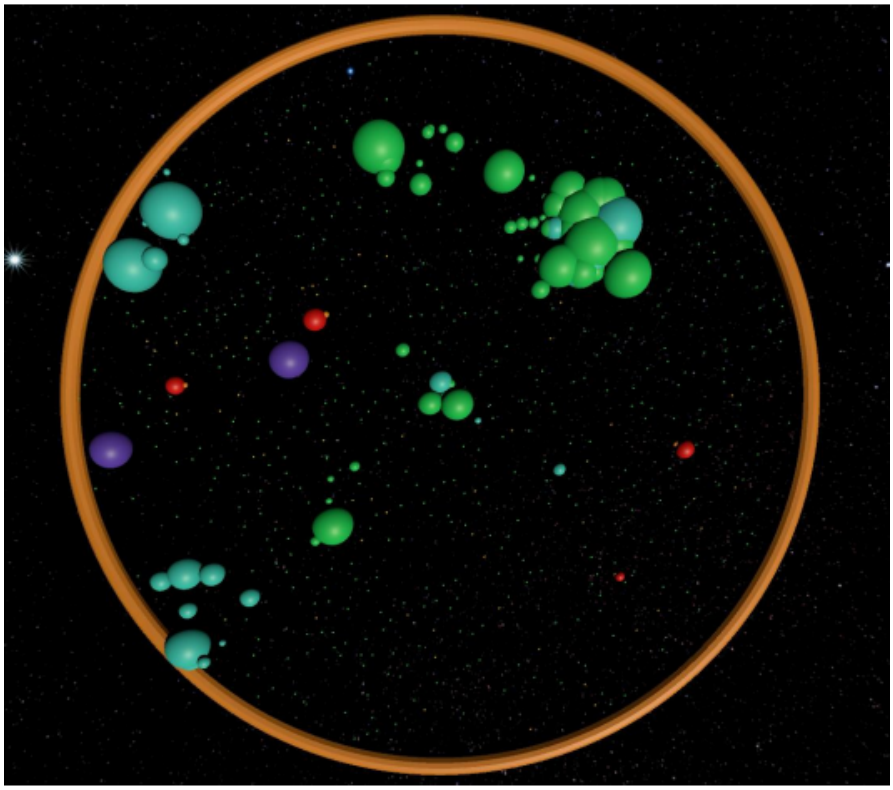
2023

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

1.1 Deskripsi Masalah

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



Gambar 1. Ilustrasi permainan *Galaxio*

BAB II
LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi merupakan persoalan mencari solusi optimal. Persoalan optimasi bisa berupa maksimasi atau minimasi. Contoh persoalan dengan maksimasi adalah persoalan *knapsack*. Diketahui sebuah tas yang memiliki kapasitas terbatas dan item-item yang masing-masing memiliki bobot dan nilai tersendiri. Persoalan ini berfokus kepada bagaimana memilih item-item tersebut sedemikian rupa sehingga nilai total dari item-item yang dipilih berupa nilai maksimal dan total bobotnya tidak melebihi kapasitas tas.

Contoh persoalan optimasi berupa minimasi adalah persoalan mencari tur dengan bobot terpendek dalam persoalan *Travelling Salesman Problem* (TSP). Diketahui seorang *salesman* harus mengunjungi tepat sekali tiap kota dari kota A dan kembali lagi ke A. Kota disimbolkan sebagai *node* dan rute yang menghubungkan antara dua kota disimbolkan sebagai *edge*. Dari A, akan dicari *edge* dengan bobot paling minimum ke *node* selanjutnya, dari *node* minimum yang terhubung dengan *node* A akan dicari lagi *node* minimum, seterusnya sampai bertemu *edge* minimum yang menghubungkan kembali ke kota A setelah mengunjungi tiap *node*.

Arti kata *greedy* itu sendiri adalah serakah, rakus, atau tamak. Algoritma ini menginginkan hasil terbaik dengan usaha seefisien mungkin. Algoritma *greedy* mengevaluasi setiap langkah dalam setiap solusi yang ada untuk memecahkan suatu masalah. Dalam evaluasi tiap langkah, akan diperoleh langkah terbaik yang disebut optimum lokal. Optimum lokal akan dikumpulkan dari tiap langkah dan harapannya akan membentuk optimum global.

Optimum global sendiri belum tentu merupakan solusi terbaik. Optimum global sendiri bisa jadi hanya merupakan solusi sub-optimum atau solusi pseudo-optimum. Hal ini terjadi karena algoritma greedy tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada (sebagaimana pada metode exhaustive search). Selain itu, terdapat beberapa fungsi seleksi yang berbeda sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma *greedy* menghasilkan solusi optimal. Contoh kasus optimum global dari algoritma *greedy* bukan merupakan solusi optimal adalah kasus penukaran uang. Diketahui terdapat koin senilai 1,3,4,

dan 5. Uang senilai 7 akan ditukarkan dengan koin-koin tersebut. Jika menggunakan algoritma *greedy* yang dalam kasus ini akan mencari koin dengan nilai terbesar terlebih dahulu, hasilnya adalah 5+1+1. Itu berarti butuh tiga koin. Namun, solusi optimalnya adalah 4+3 yang berarti hanya butuh 2 koin. Akan tetapi, jika solusi optimal mutlak tidak terlalu diperlukan, algoritma *greedy* pun dapat digunakan untuk mencari solusi hampiran (*approximation*). Hal ini karena jika ingin mencari solusi optimal yang eksak dengan algoritma ini, kebutuhan waktunya eksponensial yang artinya dibutuhkan banyak waktu. Misalnya, mencari tur dengan bobot minimal pada persoalan TSP. Untuk jumlah simpul (n) yang banyak dengan algoritma *brute force*, banyak waktu komputasi yang dibutuhkan untuk menemukannya. Dengan algoritma *greedy*, meskipun tur dengan berbobot minimal tidak dapat ditemukan, solusi dengan algoritma *greedy* dianggap sebagai hampiran solusi optimal.

Jika pun algoritma greedy dapat menghasilkan solusi optimal, keoptimalan solusi tersebut harus dapat dibuktikan secara matematis. Membuktikan optimalitas algoritma greedy secara matematis memiliki kesulitan tersendiri. Akan lebih mudah jika memperlihatkan bahwa algoritma *greedy* tidak selalu optimal dengan menunjukkan *counterexample* (contoh kasus yang menunjukkan solusi yang diperoleh tidak optimal) seperti persoalan penukaran uang yang telah disebutkan di atas di mana terdapat solusi dari algoritma *greedy* yang ternyata bukan solusi optimal. Algoritma *greedy* memiliki elemen. Elemen-elemen pada algoritma *greedy* adalah sebagai berikut.

1. Himpunan kandidat (C)

   Himpunan kandidat merupakan semua tindakan yang akan dipilih dalam setiap langkah.
2. Himpunan solusi (S)

   Himpunan solusi adalah kumpulan elemen-elemen dari himpunan kandidat yang sudah dipilih.
3. Fungsi solusi

   Fungsi solusi merupakan penentu apakah elemen-elemen dalam himpunan kandidat memberikan solusi atau tidak.
4. Fungsi seleksi

   Fungsi seleksi akan memilih elemen-elemen dalam himpunan kandidat dengan pendekatan *greedy*. Pendekatan ini dilakukan secara heuristik. Pendekatan secara heuristik akan lebih memungkinkan penemuan solusi yang optimal.

5. Fungsi kelayakan

   Fungsi kelayakan adalah fungsi yang memeriksa apakah kandidat yang dipilih layak dimasukkan ke himpunan solusi atau tidak.

6. Fungsi objektif

   Fungsi objektif adalah fungsi yang menentukan solusi maksimum atau minimum.

Algoritma *greedy* memiliki skema sebagai berikut.

```
function greedy(c : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan masukan berupa himpunan
kandidat dengan menggunakan algoritma greedy  }
Deklarasi
x : kandidat
s : himpunan_solusi
Algoritma:
s ← {} { inisialisasi s dengan kosong }
{selama solusi belum ditemukan dan himpunan kandidat masih tidak null}
while (not solusi(s) and c != {} ) do
      x ← seleksi(C) { pilih sebuah kandidat dari C}
      c ← c − {x} { x dibuang dari himpunan kandidat karena sudah
      dipilih}
      if layak(s,x} then {x layak dimasukkan ke dalam himpunan solusi}
            s ← s ∪ {x} {x masuk 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
```

2.2 Cara Kerja Program

2.2.1 Cara bot berjalan dalam permainan

Pengaturan gerakan pada Bot dilakukan pada file BotService.java. Pada file ini, terdapat method computeNextPlayerAction yang berisi gerakan apa yang akan dilakukan oleh Bot. Gerakan yang dilakukan oleh Bot terdiri dari dua bagian, yaitu aksi yang dilakukan dan arah aksi. Aksi yang dilakukan bisa berupa maju, menembakkan torpedo, hingga mengaktifkan perisai.

2.2.2 Implementasi Greedy ke bot

Implementasi greedy ke dalam Bot dapat dilakukan dengan menggunakan pengkondisian (if else) berdasarkan prioritas gerakan yang terpilih dalam algoritma greedy yang dipilih.

2.2.3 Menjalankan *Game Engine*

Untuk menjalankan permainan, pengguna terlebih dahulu menentukan jumlah Bot yang akan bertarung dalam permainan pada file appsettings.json yang terdapat pada folder runner-publish dan engine-publish. Lalu, pengguna menjalankan program GameRunner.dll pada folder runner-publish. Selanjutnya, pengguna menjalankan program Engine.dll dan Logger.dll masing-masing pada folder engine-publish dan logger-publish. Setelah itu, pengguna dapat memilih untuk menjalankan bot default di file ReferenceBot.dll pada folder reference-bot-publish menjalankan bot hasil buatan sendiri dengan perintah ‘java -jar lokasiFileJar’. lokasiFileJar merupakan direktori tempat file .jar hasil kompilasi berada. Jumlah bot yang dijalankan harus sesuai dengan jumlah bot yang diinisiasi agar permainan dapat dimulai. Hasil dari permainan akan tersimpan pada folder logger-publish.

BAB III
APLIKASI STRATEGI GREEDY

3.1 Mapping

Berikut *mapping* dalam persoalan memenangkan permainan Galaxio menjadi elemen-elemen algoritma Greedy (himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan, fungsi objektif).

Elemen algoritma Greedy	Pemetaan pada permainan Galaxio
Himpunan kandidat	{Forward, Stop, StartAfterBurner, StopAfterBurner, FireTorpedoes, FireSupernova, DetonateSupernova, FireTeleport, Teleport, ActivativeShield}
Himpunan solusi	*Command* yang valid dan bot kapal berjalan sesuai perintah dan aturan untuk memenangkan permainan.
Fungsi solusi	Memeriksa apakah aksi yang dilakukan kapal tidak membahayakan kapal dari risiko mati sebelum permainan berakhir.
Fungsi seleksi	Pilihlah aksi yang meningkatkan ukuran kapal serta menghindarkan kapal dari risiko penyusutan ukuran.
Fungsi kelayakan	Memastikan bahwa serangan yang dipilih oleh pemain benar-benar efektif dan menghasilkan hasil yang diinginkan.

Fungsi objektif	Memenangkan permainan dengan cara mempertahankan kapal pemain paling terakhir untuk hidup.

3.2 Alternatif Solusi Greedy

3.2.1 *Greedy by food*

Strategi *greedy by food* mengutamakan menambah ukuran kapal dengan memakan *food* atau *superfood*. Khusus *superfood*, kapal akan mengonsumsi makanan dengan nilainya menjadi dua kali lipat ukuran makanan biasa selama 5 ticks. Strategi *greedy by food* bisa dibilang merupakan strategi dasar dalam permainan ini di mana kapal memang perlu makanan untuk meningkatkan ukuran kapal sehingga peluang kapal bertahan sampai akhir lebih besar. Namun, karena strategi ini hanya berfokus pada makanan, mungkin saja kapal akan mengabaikan situasi di mana posisi makanan lebih dekat dengan objek yang memberikan damage dan bergerak seperti supernova dan torpedo salvo. Pada akhirnya, mengandalkan satu strategi ini saja kurang efisien mengingat risiko terkena *damage* diabaikan.

3.2.2 *Greedy by defending*

Strategi *greedy by defending* mengutamakan bagaimana ukuran kapal tidak mengecil untuk mengurangi risiko mati sebelum permainan selesai. Kapal akan memilih untuk menghindari serangan kapal musuh, supernova dari lawan, fire torpedo, *gas cloud,* dan radius. Kapal juga akan mempertimbangkan ukuran kapal sendiri jika ingin mengeluarkan supernova dan menggunakan *command* AfterBurner. Hal ini dilakukan untuk mengurangi risiko kalah karena ukuran kapal yang mengecil. Keuntungan dari strategi ini adalah kapal bisa bertahan sampai akhir dengan menghindari aksi yang bisa membuat kapal berisiko mati sebelum permainan usai. Di sisi lain, strategi *greedy by defending* bisa membuat kapal sama sekali tidak melakukan serangan. Karena terlalu fokus dalam pertahanan, kapal juga memilih bertahan ketimbang bergerak mencari makanan untuk mengurangi risiko terkena *damage*. Ukuran kapal pun akan mentok begitu saja sehingga sulit untuk menang menghadapi lawan yang lebih agresif lagi. Kesimpulannya, menggunakan hanya strategi ini kurang efisien.

3.2.3 *Greedy by speed*

Greedy by speed mengedepankan kecepatan kapal dalam permainan memanfaatkan *command* AfterBurner. Kapal akan lebih cepat menjangkau food, bahkan superfood dan mampu mengelak lebih cepat dari serangan yang membahayakan kapal. Namun, hal ini berisiko mengurangi ukuran kapal yang bisa membuat kapal mati sebelum permainan selesai. Strategi ini pun pada akhirnya tidak bisa selalu digunakan mengingat adanya risiko penyusutan ukuran kapal yang membuat kapall berpeluang mati lebih cepat.

3.2.4 *Greedy by weapon*

Greedy by weapon memfokuskan pada pemilihan senjata yang tepat seperti supernova dan torpedo salvo. Kapal akan memilih senjata dengan *damage* tertinggi untuk ditembakkan ke kapal musuh. Strategi *greedy by weapon* juga membutuhkan keakuratan saat menyerang. Namun, strategi ini tak bisa selalu diterapkan. Kapal harus menunggu tiap 10 *tick game* untuk mendapatkan torpedo salvo. Sedangkan, untuk mendapatkan supernova, kapal harus mencarinya terlebih dahulu.

3.3 Strategi Greedy Terpilih

Strategi *greedy* yang kami terapkan dalam pembuatan bot kali ini merupakan kombinasi dari beberapa alternatif solusi *greedy* yang telah dijelaskan pada bagian sebelumnya. Alternatif solusi *greedy* yang dipakai yaitu *greedy by food, greedy by defending,* dan *greedy by weapon*. *Greedy by speed* tidak digunakan karena akan mengorbankan *size* kapal. Strategi tersebut kontras dengan strategi *greedy by defending*. Strategi *greedy by speed* membuat kapal memiliki risiko kalah lebih besar. Urutan strategi *greedy* yang kami pilih yaitu:

1. Melakukan pengecekan posisi bot terlebih dahulu, apakah bot masih berada di dalam area permainan atau sudah keluar area. Jika bot sudah berada di luar, maka gerakan yang dilakukan oleh bot yaitu berjalan balik ke dalam area permainan.
2. Melakukan penyerangan ke bot lain menggunakan torpedo ataupun supernova. Penyerangan dilakukan apabila torpedo atau supernova dimiliki oleh bot.
3. Mencari makanan untuk menambah ukuran bot. Makanan yang dikejar terlebih dahulu merupakan makanan terdekat dari bot.
4. Jika ada serangan dari pemain lain, maka bot akan mengaktifkan perisai jika dimiliki ataupun teleport.

5. Jika terdapat objek berbahaya seperti *gas cloud* atau *asteroid field*, maka bot akan berbalik 180 derajat dan lanjut berjalan lurus.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy (Pseudocode)

```
{mengecek apakah kapal di dalam atau di luar arena}
if (posisi kapal di luar radius) then
      bergerak ke dalam arena di dalam radius
else then {kapal berada di dalam arena}
      {mengecek apakah kapal memiliki torpedo salvo}
      if (kapal punya torpedo salvo) then
            if (ukuran torpedo >= ukuran kapal musuh) then
                        tembakkan ke kapal musuh
            else then
                        {mengecek apakah kapal memiliki supernova}
                        if (kapal punya supernova) then
                              tembakkan ke area yang terdapat musuh
                        else then
                        kapal bergerak mencari makanan
       else then {kondisi kapal tidak punya torpedo salvo}
             {mengecek apakah kapal memiliki supernova}
                        if (kapal punya supernova) then
                              tembakkan ke area yang terdapat musuh
                        else then
                        kapal bergerak menuju makanan terdekat

{kondisi mencari food / superfood}
{mengecek kondisi apakah ada objek yang akan memberikan damage ke kapal}
if ((ada supernova) or (ada torpedo salvo)) then
    if (kapal memiliki shield) then
          aktifkan shield
    else then
          lakukan teleport
else then
       Kapal bergerak menuju makanan terdekat
```

```
{kondisi kapal diserang musuh}
if (ada serangan dari kapal musuh) then
      aktifkan shield

{kondisi terdapat objek yang mengurangi size kapal}
if (ada asteroid field or ada gas cloud) then
      kapal berbalik 180 derajat
      kapal berjalan lurus
```

Method	Fungsi
stayInsideTheRing()	Menjaga agar bot tidak keluar dari batas lingkaran pada permainan
attackStrategy()	Menghitung jarak bot ke makanan terdekat yang ada pada permainan
getFoods()	Membuat *list* makanan yang ada pada permainan
getClosestFood()	Mendapatkan makanan terdekat dari bot pada permainan
eatStrategy()	Menentukan strategi bot saat ingin memakan makanan pada permainan
computeNextPlayerAction(PlayerAction playerAction)	Mengatur aksi-aksi yang akan dilakukan oleh bot pada permainan
attackStrategy()	Menentukan strategi bot saat ingin menyerang musuh pada permainan
protectBot()	Melindungi bot dari serangan musuh pada permainan

avoidDanger()	Menghindari objek berbahaya seperti GASCLOUD dan ASTEROIDFIELD
getSupernova()	Mendapatkan koordinat supernova pada permainan
isGetSupernova()	Mendeteksi apakah bot telah mendapatkan supernova pada permainan
isSupernovaBomb()	Mendeteksi adanya ledakan supernova pada permainan

4.2 Struktur Data

Struktur data yang kami gunakan terbagi menjadi 3 bagian besar yaitu command, entities, enum yang disertakan dengan main.java.

1. Enums : berisikan file-file yang berisi apa saja objek dan aksi yang terdapat dalam permainan.
    a. ObjectType : berisi objek-objek yang terdapat dalam permainan
    b. PlayerActions : berisi aksi yang bisa dilakukan kapal dalam permainan
2. Model : berisikan file-file yang menginisiasi data-data dalam permainan
    a. GameObject : berisi inisiasi objek dalam permainan dimana objek-objek tersebut diberi *value*.
    b. GameState : berisi informasi berupa keadaan-keadaan saat permainan berlangsung
    c. GameStateDto : berfungsi mengirimkan data-data kondisi dalam permainan dalam bentuk *serializable object.*
    d. PlayerAction : menginisiasi aksi dan gerakan kapal dalam permainan
    e. Position : menginisiasi posisi objek-objek dalam permainan
    f. World : menginisiasi objek-objek yang menjadi elemen pembatas dalam permainan
3. Services : berisi file yang menampung strategi *greedy* yang diimplementasikan ke bot kapal
    a. BotService : berisi strategi-strategi yang dituangkan ke bot kapal.
4. Main : berisi eksekutor state-state untuk menjalankan permainan.

4.3. Analisis dan Pengujian

Dalam pengujian yang telah dilakukan, bot yang diimplementasikan memperoleh beberapa kemenangan dan kekalahan. Kemenangan yang diperoleh sebagian besar berasal dari serangan torpedo yang berhasil mengenai bot lawan. Kekalahan yang diterima berasal dari adanya perbedaan pada ukuran bot sehingga bot dimakan oleh bot lawan. Selain itu, terdapat beberapa serangan yang diterima oleh bot sehingga ukuran bot mengecil dan menjadi incaran dari bot lawan. Kekalahan lain juga disebabkan karena penentuan prioritas kondisi yang harus dieksekusi terlebih dahulu sehingga bot cenderung bingung menentukan pilihan dan akhirnya bergerak di tempat.

4.3.1 Keefektifan algoritma Greedy dalam memperoleh nilai optimal

Menyelesaikan masalah pemetaan permainan Galaxio berdasarkan jarak terdekat. Setiap objek pada *world* Galaxio dapat dieksekusi berdasarkan prioritas yang dapat ditentukan secara kondisional.

4.3.2 Kondisi-kondisi yang mempengaruhi keefektifan strategi greedy

Strategi greedy sangat cocok untuk masalah yang memiliki sifat greedy, yaitu masalah yang solusinya dapat dicari dengan cara memilih opsi terbaik pada setiap tahapan. Jika masalah memiliki sifat yang berbeda, seperti dinamis atau kompleks, maka strategi greedy mungkin tidak efektif. Strategi greedy cenderung bekerja dengan baik untuk masalah dengan ukuran kecil atau sedang. Akan tetapi, jika masalah terlalu besar, strategi greedy dapat menghasilkan solusi yang kurang optimal atau bahkan tidak dapat menyelesaikan masalah. Strategi greedy sangat dipengaruhi oleh keadaan awal masalah. Jika keadaan awal memiliki banyak kemungkinan solusi yang baik, strategi greedy dapat berhasil dalam menemukan solusi optimal. Namun, jika keadaan awal memiliki sedikit opsi solusi yang baik, strategi greedy dapat menghasilkan solusi yang kurang optimal. Strategi greedy biasanya cepat dalam menyelesaikan masalah, tetapi jika batasan waktu yang diberikan sangat ketat, strategi greedy mungkin tidak dapat menemukan solusi optimal.

4.3.3 Kelemahan algoritma Greedy dalam memperoleh nilai optimal

Algoritma greedy tidak selalu memperoleh nilai optimal global karena adanya pemilihan opsi terbaik pada suatu waktu untuk tiap tahapan yang ada. Nilai optimal global tidak selalu dapat diakses dari nilai optimal lokal, sehingga algoritma greedy terkadang tidak dapat mencapai hasil terbaik yang mungkin diraih.

4.3.4 Kompleksitas kinerja algoritma Greedy

Kompleksitas kinerja algoritma greedy pada permainan Galaxio bergantung pada strategi greedy yang digunakan dan ukuran peta permainan. Kompleksitas waktu pada permainan Galaxio dapat dinyatakan sebagai $O(n^2)$, di mana n adalah jumlah objek pada peta permainan. Hal ini disebabkan algoritma greedy untuk pengurutan objek-objek pada peta berdasarkan jarak dan memilih objek sebagai target baru. Proses pengurutan ini memiliki kompleksitas waktu sebesar O(n log n) karena menggunakan algoritma pengurutan seperti *quicksort*. Namun, kompleksitas waktu untuk mencari objek-objek terdekat akan menjadi O(n) karena algoritma harus menelusuri setiapobjek pada peta permainan. Karena algoritma greedy harus melakukan pencarian planet terdekat pada setiap iterasi, maka kompleksitas waktu algoritma menjadi $O(n^2)$.

Akan tetapi, kompleksitas waktu algoritma dapat bervariasi tergantung pada strategi greedy yang digunakan dan implementasi algoritma. Selain itu, faktor-faktor seperti kecepatan komputasi, memori, dan ukuran peta permainan juga dapat mempengaruhi kompleksitas kinerja algoritma.

BAB V
KESIMPULAN

5.1 Kesimpulan

Strategi *greedy* digunakan untuk menentukan solusi optimal untuk suatu masalah. Strategi *greedy* bisa diterapkan untuk memenangkan permainan Galaxio. Pemenang dalam permainan ini adalah kapal yang bertahan paling lama. Dalam hal ini, algoritma *greedy* bakal mengevaluasi tiap strategi/solusi untuk memenangkan permainan. Strategi *greedy* yang bisa diterapkan dalam permainan Galaxio adalah *greedy by size, greedy by defending, greedy by speed* dan *greedy by weapon*. Pada akhirnya, kami menggabungkan *greedy by size, greedy by defending,* dan *greedy by weapon* untuk diimplementasikan ke bot kapal.

5.2 Saran

Strategi *greedy* yang diterapkan pada permainan Galaxio mungkin saja masih belum lengkap mengingat cakupan strategi ini cukup luas. Selain itu, strategi yang kami terapkan bisa jadi belum optimal mengingat masih adanya kekalahan yang berarti fungsi objektif pada strategi *greedy* belum tercapai. Diperlukan lebih banyak ide dan referensi untuk mendapatkan lebih banyak strategi. Jadi, strategi yang didapat bisa lebih optimal lagi.

5.3 Komentar dan Refleksi

Tugas besar ini menuntut kita untuk memikirkan strategi greedy terbaik yang dapat diimplementasikan ke dalam bot. Diperlukan koordinasi terhadap ide-ide yang dimiliki oleh tiap anggota tim sehingga dapat dihasilkan strategi greedy yang terbaik. Dalam implementasinya, terdapat beberapa kendala dalam menyusun strategi dikarenakan adanya kesulitan dalam menentukan urutan prioritas gerakan bot.

DAFTAR PUSTAKA

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

[3] https://github.com/EntelectChallenge/2021-Galaxio

LAMPIRAN

Link youtube: https://youtu.be/RlBD-hqDJ6o

Link github: https://github.com/kartinicopa/Tubes1_muggle.git