

Tugas Besar 1 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

**Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan  
“Galaxio”**

Disusun Oleh:

Kelompok 37 Enliven Squad

Muhammad Bangkit Dwi Cahyono 13521055

Farhan Nabil Suryono 13521114

Johanes Lee 13521148

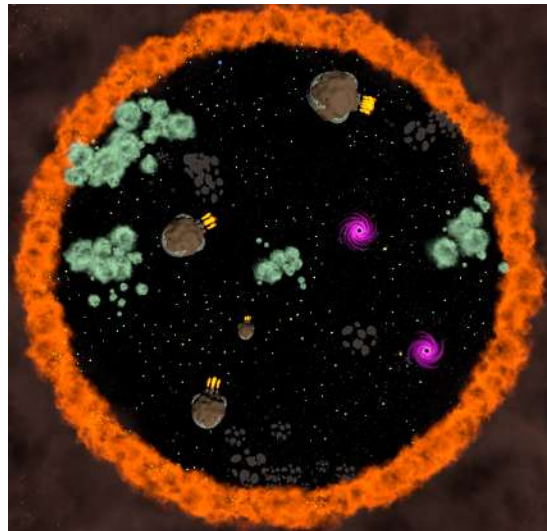


**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2023**

## 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. Agar dapat memenangkan pertandingan, setiap *bot* harus mengimplementasikan strategi tertentu untuk dapat memenangkan permainan.

Bahasa pemrograman yang digunakan pada tugas besar ini adalah Java. Bahasa Java tersebut digunakan untuk membuat algoritma pada *bot*. IDE yang digunakan untuk membantu membuat proyek ini adalah IntelliJ IDEA. IntelliJ IDEA merupakan IDE yang kompatibel dengan bahasa Java, disebabkan IDE tersebut dilengkapi dengan *tools*, seperti Maven yang sudah built in tanpa perlu menambahkan extension. Untuk menjalankan permainan, digunakan sebuah *game engine* yang diciptakan oleh *Entellect Challenge* yang terdapat pada repository githubnya. *Game engine* yang dibuat oleh *Entellect Challenge* menggunakan bahasa C# sebagai bahasa pemrograman utama dalam pembuatannya.



Spesifikasi permainan yang digunakan di dalam tugas besar ini disesuaikan dengan spesifikasi permainan “Galaxio” yang terdapat pada repository *Entellect Challenge*. Beberapa peraturan umum atau *game rules* yang terdapat pada permainan ini adalah diantaranya:

1. Peta

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. Setiap *tick* permainan, peta akan mengecil dan objek yang berada di luar peta akan dihancurkan, kecuali pemain (objek) yang akan berkurang ukurannya 1 setiap *tick*.

## 2. Objek

Semua objek pada permainan direpresentasi sebagai objek dengan bentuk lingkaran dan memiliki titik pusat (x,y) dan radius yang mendefinisikan ukurannya. Pada permainan terdapat beberapa objek, yakni sebagai berikut:

### a. Players (Kapal)

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 salvo torpedo (ukuran 10) mengurangi ukuran kapal sebanyak 5.

### b. Food dan Superfood

Setiap objek pada lintasan punya koordinat (x,y) dan radius yang mendefinisikan ukuran dan bentuknya. Food akan disebar 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.

### c. Wormholes

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.

d. Gas Clouds

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.

e. Asteroid Fields

Asteroid fields akan tersebar di sekitar peta dalam kelompok awan yang lebih kecil. Kapal dapat melintasi asteroid fields, namun begitu kapal bertabrakan dengannya, kecepatannya akan berkurang 2 kali lipat. Setelah kapal tidak lagi bertabrakan dengan asteroid fields, efeknya akan hilang.

f. Torpedo Salvo

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.

g. Supernova

Supernova merupakan senjata yang hanya muncul satu kali pada permainan di antara quarter pertama dan quarter terakhir. Senjata ini tidak akan bertabrakan dengan objek lain pada lintasannya. Player yang menembakannya dapat meledakannya dan memberi damage ke player yang berada dalam zona. Area ledakan akan berubah menjadi gas cloud. 8. Player dapat meluncurkan teleporter pada suatu arah di peta. Teleporter tersebut bergerak dalam direksi dengan kecepatan 20 dan tidak bertabrakan dengan objek apapun. Player tersebut dapat berpindah ke tempat teleporter tersebut. Harga setiap peluncuran teleporter adalah 20. Setiap 100 tick player akan mendapatkan 1 teleporter dengan jumlah maximum adalah 10.

h. Teleport

Seorang pemain dapat meluncurkan teleporter ke suatu arah di peta. Teleporter bergerak ke arah itu dengan kecepatan 20 dan tidak bertabrakan dengan apapun. Pemain kemudian dapat menggunakan perintah lain untuk berteleportasi

ke lokasi teleporter mereka. Itu akan hancur jika mencapai akhir peta. Dibutuhkan ukuran pemain sebanyak 20 untuk meluncurkan teleporter terlepas dari apakah mereka berteleportasi ke sana atau tidak. Seorang pemain mulai dengan satu teleporter dan mendapatkan yang lain setiap 100 *ticks* dan dapat memiliki maksimal 10 pada titik mana pun.

### 3. Collisions (Tabrakan)

Ketika kapal player bertabrakan dengan kapal lain, maka kapal yang lebih besar akan mengonsumsi kapal yang lebih kecil sebanyak 50% dari ukuran kapal yang lebih kecil. Hasil dari tabrakan akan mengarahkan kedua kapal tersebut berlawanan arah.

### 4. Commands

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. START\_AFTERBURNER
- d. STOP\_AFTERBURNER
- e. FIRE\_TORPEDOES
- f. FIRE\_SUPERNOVA
- g. DETONATE\_SUPERNOVA
- h. FIRE\_TELEPORTER
- i. TELEPORT
- j. USE\_SHIELD

### 5. Akhir Permainan dan Scoring

Setiap player akan memiliki score yang hanya dapat dilihat jika permainan berakhir. Score ini digunakan saat kasus tie breaking (semua kapal mati). Jika mengonsumsi kapal player lain, maka score bertambah 10, jika mengonsumsi food atau melewati wormhole, maka score bertambah 1. Pemenang permainan adalah kapal yang bertahan paling terakhir dan apabila tie breaker maka pemenang adalah kapal dengan score tertinggi.

## Bab 2: Landasan Teori

### 2.1. Dasar Teori Algoritma *Greedy* secara Umum

Algoritma *Greedy* adalah salah satu strategi algoritma yaitu dengan memecahkan permasalahan secara langkah per langkah dimana dalam setiap langkahnya algoritma mengambil pilihan yang terbaik tanpa melihat kemungkinan lain dengan harapan mendapat optimum global melalui optimum lokal.

Suatu persoalan dapat diselesaikan dengan algoritma *greedy* jika persoalan tersebut memiliki dua sifat, yakni:

1. Solusi optimal dari persoalan dapat ditentukan dari solusi optimal sub persoalan tersebut.
2. Pada setiap persoalan, terdapat langkah yang dapat dilakukan, langkah tersebut menghasilkan solusi optimal pada sub persoalan. Langkah ini juga dapat disebut sebagai greedy choice.

Algoritma *Greedy* memiliki 6 elemen yaitu:

1. Himpunan Kandidat : Berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan Solusi : Berisi kandidat yang telah terpilih
3. Fungsi Solusi : Menentukan apakah himpunan kandidat yang dipilih sudah merupakan suatu solusi.
4. Fungsi Seleksi : Memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi ini memiliki sifat heuristik.
5. Fungsi Kelayakan : Memeriksa apakah kandidat yang telah dipilih layak dimasukkan himpunan solusi.
6. Fungsi Obyektif : Memaksimumkan atau meminimumkan

Secara garis besar, algoritma *greedy* dapat digunakan untuk masalah yang membutuhkan solusi hampiran (tidak *exact*) atau membutuhkan solusi yang tidak selalu terbaik, namun cukup mendekati. Solusi ini terkadang lebih baik daripada algoritma yang menghasilkan solusi *exact* dengan kebutuhan waktu yang eksponensial.

## 2.2. Garis Besar Cara Kerja *Bot* pada Permainan *Galaxio*

*Bot* pada permainan *Galaxio* memiliki beberapa *attribute* yaitu:

1. Size : Ukuran *bot*
2. Speed : Kecepatan gerak *bot*
3. Heading : Arah gerak *bot*
4. Posisi : Posisi *bot* saat ini
5. Effects : Efek yang sedang dimiliki *bot* seperti *afterburner* dan *shield*
6. Salvo Count : Suatu *resources* yang akan digunakan untuk menembak torpedo
7. Supernova Available : Menandakan apakah *bot* memiliki supernova
8. Teleporter Count : Jumlah teleporter yang dimiliki *bot*
9. Shield Count : Jumlah *shield* yang dimiliki *bot*

Selain itu, *Bot* pada permainan *Galaxio* memiliki 10 jenis aksi yang dapat dilakukan pada setiap *tick*-nya (satuan waktu dalam *game Galaxio*). 10 jenis aksi itu adalah sebagai berikut.

### 1. FORWARD

Menerima masukan berupa *heading* lalu mengubah arah gerak *bot* sesuai *heading* yang telah dimasukkan.

### 2. STOP

*Bot* akan berhenti bergerak.

### 3. START\_AFTERBURNER

*Bot* akan bergerak lebih cepat dengan mengaktifkan *afterburner*, namun *size bot* akan berkurang selama *afterburner* aktif.

### 4. STOP\_AFTERBURNER

Mematikan *afterburner* sehingga kecepatan *bot* kembali normal.

### 5. FIRE\_TORPEDOES

Menerima masukan berupa *heading* lalu menembak torpedo ke arah *heading* yang telah ditentukan dan menghabiskan *salvo count* serta mengurangi *size bot*. *Damage* torpedo dihitung berdasarkan *size* dari torpedo.

### 6. FIRE\_SUPERNOVA

Bila *bot* telah mengambil *pickup* dari *supernova*, *bot* akan menerima masukan berupa *heading* dan menembakkan bom *supernova* ke arah *heading* yang telah ditentukan.

#### 7. DETONATE\_SUPERNOVA

Bila *bot* telah menembakkan bom *supernova*, *bot* akan meledakkan *supernova* yang akan membuat semua *bot* di radiusnya menerima *damage* dan meninggalkan *Gas Cloud* di bekas radius ledakan.

#### 8. FIRE\_TELEPORTER

Bila *bot* memiliki *teleport charge*, *bot* akan menerima masukan berupa *heading* dan menembakkan teleporter ke arah *heading* yang telah ditentukan.

#### 9. TELEPORT

Bila *bot* telah menembakkan teleporter, *bot* akan berpindah posisi ke teleporter yang telah ditembakkan.

#### 10. USE\_SHIELD

Bila *bot* memiliki *shield*, *bot* akan mengaktifkan *shield* yang akan men-deflect torpedo yang mengenai *bot*.

### 2.3. Implementasi Algoritma *Greedy* pada *Bot* Permainan *Galaxio*

Strategi *Greedy* diimplementasikan kepada *Bot* melalui kelas *BotService* yang telah tersedia pada starter-pack yang telah diberikan *Galaxio*. Dengan mengubah isi dari prosedur *computeNextPlayerAction*, *bot* akan melakukan aksi sesuai dengan algoritma yang telah didesain dalam prosedur tersebut. Aksi diprioritaskan berdasarkan teknik *greedy* nya.

Terdapat cukup banyak kemungkinan solusi *greedy* yang dapat dieksplorasi oleh penulis. Meskipun algoritma *greedy* tidak menghasilkan solusi optimum global, tetapi setidaknya dapat selalu mencapai solusi optimum lokal yang nilainya mendekati solusi optimum global.

### 2.4. Garis Besar *Game Engine* pada Permainan *Galaxio*

*Game Engine* dari *Game Galaxio* memiliki peran untuk men-generate dunia dimana *game Galaxio* berjalan. *Engine* memunculkan *food* dan objek-objek lainnya di berbagai tempat dan akan memunculkan *supernova pickup* di tengah berjalannya



permainan. Selain sebagai *generator* dunia, *Game Engine* akan mengisi salvo count dan *resource* lainnya kepada semua *bot* pemain.

Untuk cara kerja *Game Engine Galaxio*, *Engine* akan membuat suatu *connection* lokal di dalam komputer dan *bot* akan melakukan koneksi dengan *Engine*. Bila *bot* yang masuk telah sesuai dengan jumlah player yang diterima *Engine*, *Engine* akan memulai game *Galaxio*. Selama berjalannya permainan, *Engine* akan mengkalkulasi segala kejadian dalam *Game* dan terus menerus mengirim GameState ke semua bot agar bot dapat mengetahui kondisi saat ini untuk menjalankan strategi yang telah diimplementasikan dalam bot.

### **Bab 3: Aplikasi Strategi *Greedy***

#### **3.1. Pemetaan Elemen/Komponen Algoritma *Greedy* pada *Bot* Permainan *Galaxio***

##### **3.1.1. Himpunan Kandidat :**

Himpunan Kandidat dalam pengaplikasian Strategi *Greedy* untuk *Bot* permainan *Galaxio* adalah kumpulan aksi-aksi yang dapat dilakukan bot.

##### **3.1.2. Himpunan Solusi :**

Pilihan aksi-aksi yang memungkinkan untuk dilakukan di suatu waktu.

##### **3.1.3. Fungsi Solusi :**

Memeriksa apakah aksi yang telah dipilih merupakan sesuatu yang harus diprioritaskan untuk dilakukan.

##### **3.1.4. Fungsi Seleksi :**

Memilih aksi yang menghasilkan keuntungan terbesar agar meningkatkan kemungkinan kemenangan.

##### **3.1.5. Fungsi Kelayakan :**

Apakah suatu aksi memiliki manfaat dan tidak membuang-buang waktu atau merugikan *bot*.

##### **3.1.6. Fungsi Obyektif :**

Keuntungan yang didapat dari melakukan aksi yang telah terpilih adalah keuntungan terbesar yang mungkin didapat.

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

Terdapat beberapa solusi alternatif yang dapat diimplementasi pada permainan *Galaxio*. Hal ini disebabkan banyaknya objek dari game dengan informasi yang dapat diakses oleh *bot* serta banyaknya aksi yang dapat dilakukan berdasarkan informasi-informasi seluruh objek di dalam permainan. Dengan demikian, kemungkinan persoalan dapat menjadi sangat beragam serta setiap objek dapat memiliki suatu keterhubungan dan memengaruhi objek lain yang ada pada game. Berdasarkan hal ini, dirancang teknik atau strategi *greedy* yang diimplementasikan dalam algoritma *bot* ini yang sebagian besar dikelompokkan berdasarkan objek yang memengaruhi aksi tersebut.

### 3.2.1. *Greedy by Using Supernova*

*Bot* akan memprioritaskan mengejar *supernova pickup* apabila sedang tidak dalam kondisi berbahaya dan *bot* ini merupakan *bot* terdekat dari *supernova pickup* tersebut. Setelah memiliki *supernova pickup* juga akan meledakkan *supernova* apabila terdapat banyak lawan di sekitar *supernova* atau *supernova bomb* sudah berada di pinggir. Lawan yang menjadi target penembakan *supernova* ini adalah lawan dengan ukuran terbesar. Selain itu, *supernova* tidak akan ditembakkan jika masih berada dekat dengan *bot* pemain agar tidak melukai diri sendiri.

### 3.2.2. *Greedy by Teleport*

*Bot* akan menembak *teleporter* untuk memenuhi salah satu dari 2 tujuan, yaitu penyerangan dan pertahanan. Dalam mode menyerang, *bot* akan menembakkan *teleporter* ke arah *bot* lain yang terdekat dan berukuran lebih kecil dibandingkan ukuran *bot* setelah meluncurkan *teleporter* (yaitu ukuran *bot* saat ini dikurangi *cost* ukuran yang digunakan untuk menembak *teleporter* tersebut). Dalam mode bertahan, *teleporter* akan diluncurkan jika sudah tidak ada makanan di sekitar *bot*. Arah *teleporter* diluncurkan ke arah rata-rata makanan di peta permainan jika masih ada setidaknya 20 makanan di dalam permainan yang sedang berlangsung.

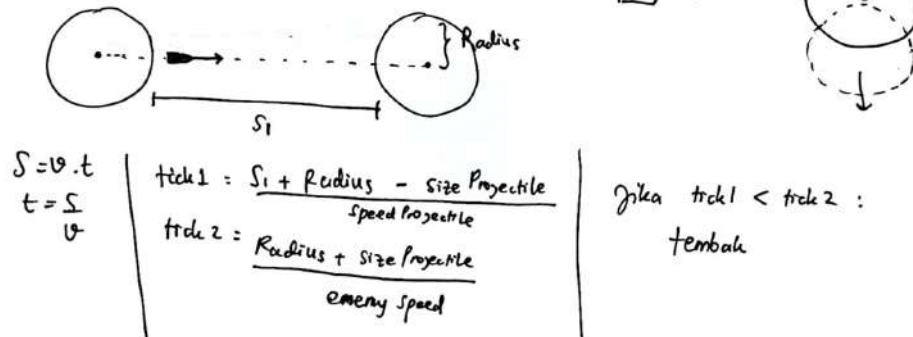
Aksi *teleport* ditunda atau bahkan dibatalkan jika terdapat bahaya di sekitar *teleporter* yang sudah ditembak, seperti adanya *gas cloud*, torpedo, ataupun *supernova bomb*. Dalam kedua mode, *bot* akan melancarkan aksi *teleport* jika aksi tersebut tidak akan menyebabkan pengurangan ukuran *bot*. Perhitungan ini melibatkan penjumlahan ukuran *bot* apabila mendapatkan makanan atau mengonsumsi musuh setelah aksi *teleport* serta pengurangan ukuran *bot* apabila mengenai torpedo di sekitar *teleporter*. Selain itu, *bot* juga akan otomatis melancarkan aksi *teleport* jika terdapat *supernova pickup* di sekitar *teleporter* yang ditembakkan sebelumnya.

### 3.2.3. *Greedy by Firing Torpedoes*

Apabila *bot* memiliki jumlah *salvo* yang cukup, *bot* akan menembakkan torpedo ke *bot* terdekat. *Bot* akan mengkalkulasi arah penembakan agar meningkatkan

kemungkinan lawan tertembak. Berikut adalah kalkulasi arah penembakan yang digunakan agar penembakan yang dilakukan memiliki *miss rate* yang kecil.

### Teknik Menembak



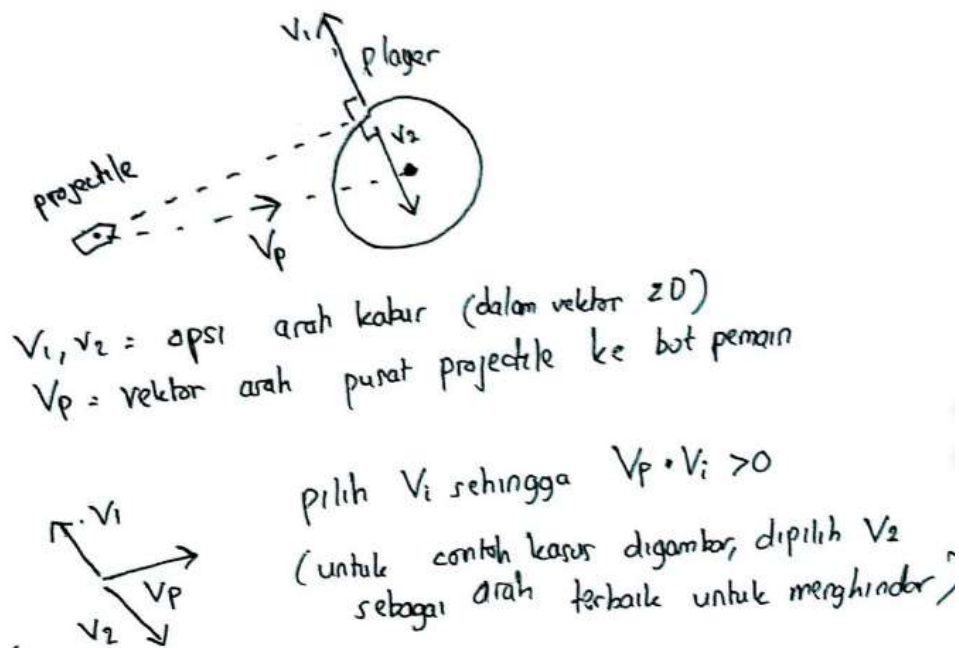
Teknik yang diimplementasikan adalah dengan menghitung waktu yang dibutuhkan torpedo untuk sampai ke musuh (*tick1*) dengan waktu yang dibutuhkan musuh untuk kabur tegak lurus dari arah torpedo (*tick2*). Untuk mendapatkan nilai *tick1*, dihitung jarak dari permukaan *bot* (diasumsikan sebagai titik awal munculnya torpedo yang akan ditembakkan) ke titik pusat musuh (dengan kasus terburuknya adalah musuh kabur tegak lurus sehingga diperlukan jarak tambahan sebesar radius musuh untuk mengenai pinggir musuh tersebut). Hasil perhitungan jarak tersebut kemudian dikurangkan dengan ukuran (radius) torpedo karena torpedo hanya perlu mengenai musuh pada bagian depan torpedo tersebut. Total jarak ini kemudian lalu dibagi dengan speed torpedo sehingga diperlukan jumlah tik yang diperlukan untuk mengenaui musuh.

Nilai *tick2* dapat diperoleh dengan menghitung waktu yang dibutuhkan musuh untuk menghindari dengan kasus terburuk bagi penembak adalah musuh menghindari dengan arah tegak lurus dari arah lintasan torpedo. Berdasarkan kasus terburuk tersebut, *tick2* diperoleh dengan cara menjumlahkan radius musuh dengan ukuran torpedo (musuh juga perlu menghindari dari sisi torpedo) yang kemudian dibagi dengan kecepatan musuh. Jika *tick1* bernilai lebih kecil daripada *tick2*, waktu yang dibutuhkan torpedo untuk mengenai musuh lebih kecil dari waktu musuh untuk kabur. Apabila kondisi tersebut terpenuhi, *bot* akan menembakkan torpedo sehingga *miss rate* torpedo tersebut menjadi lebih kecil. Perhitungan ini pun dapat

diimplementasikan ke *projectile* yang lain (seperti penembakan *supernova* atau *teleporter*) untuk *offensive strategy*.

### 3.2.4. Greedy by Escaping

Bot akan mengkalkulasi arah gerak menjauhi *bot* besar yang berjarak tertentu serta torpedo, bom *supernova*, dan *teleporter* yang mengarah ke bot pemain. Untuk setiap objek berupa musuh besar, bot pemain mengambil rute terbaik dengan arah berlawanan dengan arah bot ini ke musuh besar tersebut. Namun, untuk objek *projectiles* (torpedo, bom *supernova*, dan *teleporter* musuh), pemain menghitung rute terbaik dengan arah tegak lurus arah pergerakan *projectiles* tersebut. Berikut perhitungan yang digunakan dalam memilih dua kemungkinan arah tegak lurus yang diperoleh untuk menghindari setiap *projectiles*.

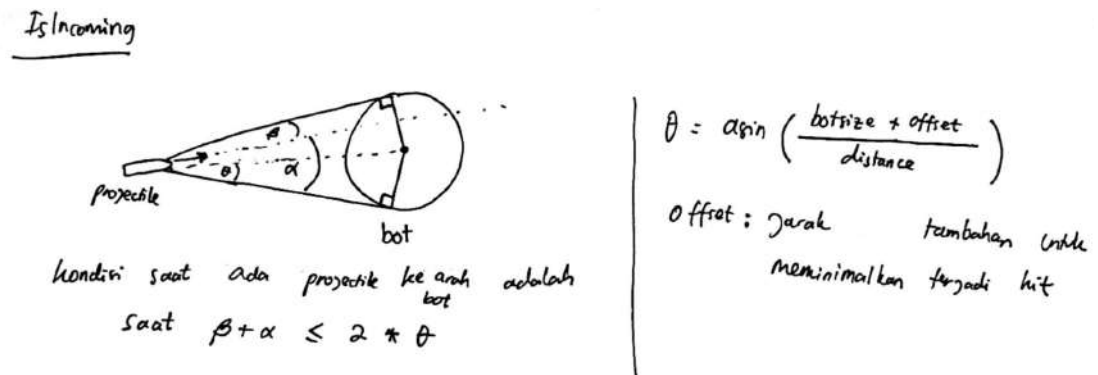


Bot akan menjumlahkan bobot arah kabur dari setiap bahaya (yaitu vektor satuan arah kabur di kali satu per jarak *projectile* ke bot) sebagai pendekatan *greedy* agar meminimalisasi kemungkinan terkena bahaya dari setiap objek. Untuk kasus khusus menghindari dari bot yang hanya memiliki dua rute menghindari serta penjumlahan arah tersebut menghasilkan vektor nol, diambil jalur yang tegak lurus dari garis yang

menghubungkan titik pusat kedua bot musuh yang ingin dihindari. Sekali lagi, diambil salah satu dari dua rute tegak lurus yang mungkin dan dalam hal ini diambil rute yang mendekati pusat peta permainan.

### 3.2.5. Greedy by Defense

*Bot* memiliki strategi defensif dari torpedo yang datang ke arah *bot* dengan cara menembak torpedo lawan bila memiliki cukup salvo. Hal ini dilakukan karena menembak torpedo lawan juga dapat meningkatkan size bot. Aksi bertahan juga dapat dilakukan dengan menggunakan shield. Kondisi *bot* yang sedang dalam bahaya dibagi menjadi 2 kasus, yaitu 1) kondisi saat ada *projectile* ke arah *bot* dapat dihindari serta 2) kondisi saat *projectile* berada dalam *danger zone* (*projectile* yang mengarah ke *bot* sulit atau tidak mungkin dihindari).



*Botsize* adalah radius dari *bot* dan *offset* adalah jarak tambahan untuk lebih meminimalkan terjadinya *calculation error*. Selain itu, perhitungan *danger zone* dilakukan seperti berikut.

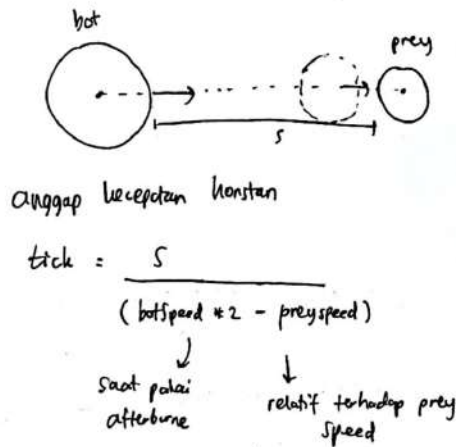
Danger

kondisi sama dengan incoming  
 dengan zona 80% dari zona incoming  
 dan  $distance \leq constant * bot\ size$

Jika suatu *projectile*, sudah memasuki *danger zone*, *bot* akan menembakkan *defensive torpedoes*. Hal ini disebabkan penembakan torpedo kepada torpedo lawan akan memberikan size sebanyak maksimal 5 kepada bot pemain (ukuran pemain berkurang 5 saat menembak torpedo tetapi mendapatkan kembali ukuran sebesar maksimal 10 jika mengenai torpedo lain ataupun musuh). Dengan demikian, strategi ini dapat menguntungkan *bot* sebanyak maksimal 5 ukuran untuk setiap *projectile* yang datang.

### 3.2.6. Greedy by Hunting

Dalam kondisi aman dari bahaya dan terdapat bot kecil di sekitar bot, bot akan mengejar bot-bot kecil tersebut, baik dengan pergerakan biasa, afterburner, maupun teleport. Untuk penggunaan *afterburner*, teknik perhitungan yang digunakan adalah sebagai berikut.

After burner

Jika, pengurangan size lebih pakai after burner  
 $(bot\ size - 2 * tick) > prey\ size + offset :$   
 berani after burner

⊕ Namun, jika tick selanjutnya tidak memenuhi, langsung matikan

Teknik yang digunakan yaitu dengan membandingkan *bot size* dengan *prey size* saat menggunakan afterburner dengan kecepatan *bot* relatif terhadap *prey*. Jika hal ini dipenuhi maka *bot* berani untuk memakan musuh. Implementasi ini menggunakan  $2 * tick$  agar mengurangi potensi error dan juga menggunakan  $+ offset$ . Pada kenyataannya, *bot* berkurang 1 per *tick* jika menggunakan *afterburner*.

### 3.2.7. Greedy by World Border

Apabila *bot* berada cukup pinggir pada peta permainan (dengan *offset* tertentu), *bot* akan bergerak dengan arah mendekati *center point* agar mencegah pengurangan size karena berada di luar *world*.

### 3.2.8. Greedy by Obstacle

*Bot* akan menghindari dari *obstacles* seperti *asteroid field* dan *gas cloud*. Namun, *bot* memprioritaskan (bobot yang lebih tinggi) untuk menjauh dari *gas cloud* dibandingkan menjauh dari *asteroid* karena risiko yang lebih tinggi. *Obstacle* akan dihindari jika berada di dalam radar deteksi tertentu (besar radar diperoleh dari beberapa eksperimen).

Untung masing-masing *obstacle*, *bot* menghitung rute kabur dengan arah menjauh dari pusat *obstacle* tersebut. Nilai bobot kontribusi setiap arah berbanding terbalik dengan jarak yang dibutuhkan *bot* untuk keluar dari *obstacle* (jika *bot* sudah berada di



dalam *obstacle* ataupun jarak sisi *obstacle* dengan sisi *bot* (jika *bot* belum memasuki *obstacle*).

### 3.2.9. *Greedy by Food*

Dalam kondisi aman dari bahaya, bot akan mengejar food atau superfood untuk memperbesar size *bot*. Makanan hanya dideteksi ketika makanan tersebut berjarak minimal sebesar radar *bot* (ukuran radar didapatkan melalui beberapa percobaan). *Bot* akan mengejar makanan terdekat yang terdeteksi radar tersebut. Apabila *bot* memiliki efek *super food*, *bot* akan menaikkan prioritas pencarian *food* sehingga berkontribusi dalam perhitungan arah total *bot* (termasuk ketika ada dalam bahaya yang perlu dihindari). Selain itu, ketika tidak ada makanan yang terdeteksi, ada dua pilihan bagi bot: menembakkan *teleporter* ke tempat yang memiliki banyak makanan atau *stop* (jika tidak ada aksi dengan prioritas lebih tinggi yang perlu dilakukan).

## 3.3. Analisis Efisiensi dan Efektivitas dari Kumpulan Solusi Algoritma *Greedy*

### 3.3.1. Analisis Strategi 1 : *Supernova*

Tujuan strategi ini adalah memanfaatkan supernova sebagai senjata terkuat dan terlangka dalam game *Galaxio* karena hanya muncul sekali dalam satu permainan. Strategi ini dapat memberikan keuntungan besar bagi *bot* pemain. Kelebihan dari strategi ini adalah *bot* dapat membuat lawan yang jauh lebih kuat menjadi mati atau mengecil karena *damage* yang besar serta mengacaukan pergerakan musuh karena *gas cloud* yang dihasilkan dari sisa ledakan. Karena strategi *greedy* yang digunakan adalah mengincar *bot* terbesar, *miss rate* tembakan *supernova* akan menjadi lebih kecil akibat *bot* yang lebih besar memiliki kecepatan yang kecil. Dengan demikian, *bot* yang besar akan lebih sulit menghindari tembakan *supernova*. Strategi ini juga dapat mengurangi tingkat bahaya akibat bot musuh yang terlalu besar.

Kelemahan dari strategi ini adalah risikonya yang cukup tinggi. Karena *supernova pickup* berada di tengah *World*, bot yang akan mengambil harus mendatangi bagian tengah *World* sehingga dapat menjadi target bagi bot lain. Selain itu, *supernova* ditembakkan ke arah pusat musuh tetapi tidak memperhitungkan peluang musuh menghindar sehingga *supernova* yang ditembakkan dapat menjadi sia-sia.

### 3.3.2. Analisis Strategi 2 : *Teleport*

Strategi *teleport* yang digunakan dalam penyerangan dapat menjadi mematikan bagi *bot* musuh yang memiliki ukuran lebih kecil dari *bot* pemain dan berpapasan dengan *teleporter* yang sudah ditembakkan. Mengeliminasi pemain lain akan sangat menguntungkan *bot* karena mengurangi bahaya terbesar permainan. Selain itu, *bot* pemain juga dapat mendapatkan *supernova pickup* jika *teleporter* berpapasan dengan objek tersebut.

Strategi *teleporter* yang digunakan untuk bertahan juga dapat menguntungkan pemain ketika pemain sedang berada di daerah kosong makanan. Dengan demikian, pemain dapat berpindah ke tempat yang lebih banyak makanan sekaligus memanfaatkan momen tersebut jika sedang dikejar bahaya.

Kekurangan strategi ini adalah kurangnya perhitungan peluang *bot* lain menghindari *teleporter* yang *bot* ini tembak untuk menyerang. Apabila *bot* berhasil menghindar, besar kemungkinan *bot* pemain tidak akan meluncurkan aksi *teleport* untuk menghindari kemungkinan terburuk. Dengan demikian, *bot* ini akan mengalami kerugian ukuran yang digunakan untuk menembakkan *teleporter*.

### 3.3.3. Analisis Strategi 3 : *Firing Torpedoes*

Strategi menembak torpedo yang digunakan cukup akurat karena dilakukan perhitungan peluang musuh menghindar. Strategi ini juga dapat menjadi salah satu cara pemain menambah ukuran diri sendiri apabila torpedo yang ditembakkan berhasil mengenai musuh ataupun torpedo musuh. Selain itu, strategi ini dapat menjadi mematikan bagi musuh yang memiliki ukuran yang kecil dan tidak menggunakan *shield*.

Kerugian strategi yang digunakan adalah torpedo masih dapat tidak mengenai musuh atau torpedo lain karena pemberian *offset* tambahan yang memperlonggar syarat penembakan. Pelonggaran syarat ini dilakukan agar jumlah tembakan *bot* tidak terlalu sedikit serta adanya unsur ketidakpastian dalam pergerakan *bot* lawan. Selain itu, torpedo yang ditembakkan dapat dimanfaatkan musuh dengan strategi yang sama

untuk menambah ukurannya (dengan menghancurkan torpedo yang ditembakkan bot ini).

#### 3.3.4. Analisis Strategi 4 : *Escaping*

Strategi ini sangat menguntungkan *bot* untuk menghindari berbagai jenis bahaya yang mendatangnya. Selain itu, pendekatan arah rata-rata sebagai strategi *greedy* lebih baik dibandingkan hanya melihat satu bahaya. Kalkulasi yang digunakan dalam menentukan arah kabur sangat meningkatkan akurasi aksi ini. Namun, karena strategi ini menjadi salah satu strategi prioritas tertinggi dalam implementasi bot ini, *bot* lebih memprioritaskan menghindar dari bahaya walaupun ada kesempatan lain yang menguntungkan bot (seperti mengabaikan *supernova pickup* ataupun makanan yang tidak berada di rute kabur).

#### 3.3.5. Analisis Strategi 5 : *Defense*

Strategi menembak kembali torpedo yang mengarah ke kita dapat menjadi alternatif penambahan ukuran bot. Selain itu, penggunaan shield dapat mencegah penerimaan kerusakan yang sangat besar apabila *bot* tidak memiliki *salvo count* untuk menembak torpedo. Strategi ini sangat jarang merugikan *bot* pemain karena kondisi yang terdefinisi dengan baik.

#### 3.3.6. Analisis Strategi 6 : *Hunting*

Strategi ini sangatlah efisien dalam mendapatkan keuntungan dengan cepat karena *bot* akan mendapatkan tambahan *size* yang cukup besar dengan memakan *bot* lainnya. Selain itu, strategi ini dapat menyudutkan pemain lain yang dikejar sehingga *bot* musuh terpaksa pergi ke tempat yang tidak menguntungkan. Namun, strategi ini bisa menjadi kerugian karena lawan dapat membalikkan keadaan dengan memakan makanan untuk menambah *size* atau menembakkan torpedo sehingga *bot* pemain mengecil.

#### 3.3.7. Analisis Strategi 7 : *World Border*

Strategi ini menghindarkan pemain dari bahaya yang cukup tinggi, yaitu pengurangan ukuran yang signifikan akibat keluar dari batas peta permainan. Karena radius peta juga ikut mengecil, besar kecepatan relatif *bot* akan menjadi lebih kecil dibandingkan kecepatan asli *bot* sehingga *bot* akan lebih sulit memasuki kembali daerah aman. Namun, bobot strategi ini diatur lebih kecil dibandingkan strategi menghindar dari beberapa bahaya lain yang lebih mematikan sehingga ada kemungkinan *bot* mengabaikan (karena bobot yang relatif lebih kecil) strategi ini.

#### 3.3.8. Analisis Strategi 8 : *Obstacle*

Strategi ini menghindarkan *bot* dari *gas clouds* dan *asteroid fields* yang merugikan *bot*, terutama *gas clouds* yang dapat menjadi mematikan apabila ukuran *bot* kecil. Namun, karena prioritas yang sama (walaupun bobot yang lebih kecil) dengan beberapa strategi lain, strategi ini dapat mengubah hasil perhitungan total dan memengaruhi hasil penentuan aksi strategi lain.

#### 3.3.9. Analisis Strategi 9 : *Food*

Strategi ini merupakan strategi paling *basic* dan pasti dimanfaatkan karena *bot* hampir tidak mungkin menang apabila tidak makan cukup banyak makanan. Strategi ini umum dipilih pada saat permainan baru dimulai dan sangat menguntungkan pemain untuk menambah ukuran dan memungkinkan dilancarkan aksi lainnya. Strategi ini juga menjadi pilihan *default* (selain *stop* jika tidak ada makanan) apabila tidak ada aksi lain dengan prioritas yang lebih tinggi. Karena diletakkan pada prioritas terendah, strategi ini tidak akan mengganggu hasil perhitungan strategi lainnya.

### 3.4. Strategi *Greedy* yang Digunakan pada Program *Bot*

Dari seluruh solusi *greedy* yang telah dijelaskan dan dianalisis, kelompok kami memutuskan untuk menggabungkan seluruh strategi dan membuat skala prioritas penentuan penggunaan strategi. Secara umum, skala prioritas yang telah kami rancang adalah sebagai berikut.

1. Meledakkan *supernova*
2. Melakukan *teleport*

3. Menembak *teleport* ke arah *bot* kecil
4. Memberhentikan *afterburner* apabila *size* terlalu kecil
5. Menembakkan torpedo ke *bot* terdekat
6. Melarikan diri dari *bot* besar
7. Melakukan *offensive afterburner* (untuk mengejar lawan)
8. Memberhentikan *afterburner* apabila tidak ada *bot* kecil di sekitar atau bila *afterburner* membahayakan *bot*
9. Mengejar lawan dengan mempertahankan *afterburner*
10. Melakukan *defense* dengan *shield* atau menembakkan torpedo ke arah torpedo lawan
11. Menjauh dari torpedo
12. Mengejar *bot* yang lebih kecil
13. Menjauh dari *world border*
14. Menjauh dari bom supernova
15. Menjauh dari *gas cloud*
16. Menjauh dari asteroid
17. Fokus mencari makan apabila *superfood* aktif
18. Mengejar pickup supernova apabila memungkinkan
19. Menembakkan supernova
20. Mengejar makanan terdekat atau menembak *teleporter* ke tempat dengan banyak makanan

Urutan di atas tidak hanya berarti bahwa urutan lebih rendah hanya dilakukan apabila urutan di atasnya tidak dilakukan. Terdapat beberapa aksi dengan prioritas sama sehingga salah satu pendekatan yang diambil adalah mengambil keputusan tengah dari seluruh keputusan tersebut (salah satunya adalah menggunakan rata-rata dalam perhitungan arah pergerakan bot).

Selain itu, untuk aksi-aksi dengan skala prioritas 18 ke bawah, kami membuatnya hanya dilakukan apabila memang tidak ada bahaya yang harus dihindari atau sasaran yang harus dikejar.

## Bab 4: Implementasi dan Pengujian

### 4.1. Implementasi Algoritma *Greedy* pada *Bot Permainan Galaxio*

```
function computeNextPlayerAction() → Aksi
```

```
{ Fungsi menentukan aksi player }
```

#### KAMUS LOKAL

```
tickTimer, maxEnemySize : integer
```

```
effectList                : list of Boolean
```

```
directionVectors          : list of EscapeInfo
```

```
t                          : EscapeInfo
```

```
temp                      : WorldVector
```

```
superNovaBombs, playersList, incomingTeleporters, biggerPlayer, preys :  
list of GameObject
```

```
teleporter                : GameObject
```

#### ALGORITMA FUNGSI

```
if (tickTimer > 0) then
```

```
    tickTimer ← tickTimer - 1
```

```
// Game belum dimulai
```

```
if (gameState = null or gameState.world = null or gameState.world.radius =  
null or gameState.world.centerPoint = null) then
```

```
    → Tidak melakukan apa apa
```

```
Effects.getEffectList(bot.effectsCode)
```

```
RadarService.updateAttributes(gameState, bot)
```

```
superNovaBombs <- SupernovaService.getSupernovaBombs(gameState)
```

```
if (superNovaBombsExist and superNovaAlmostOutside or superNovaNearPlayer  
and not isDetonated and isSuperNovaFired and botNotInSuperNovaRange) then  
/* jika kita sudah nembak supernova dan supernova bombnya dekat musuh atau  
akan keluar map (menghindari error) */
```

```
    isDetonated ← true
```

```
    isSuperNovaFired ← false
```

```
→ DETONATESUPERNOVA

if (isTeleporterFired) then
    teleporter ← getFiredTeleport
    if (teleporter != null) then
        if (isTeleportTargetSafe) then
            → TELEPORT

if (not isTeleporterFired and isTeleportAvailable) then
    if (isNoAttack) then
        headingAksi ← toDegree(getAttackDirection)
        → FIRETELEPORT

playersList ← getOtherPlayerList()
maxEnemySize ← getBiggestEnemySize(playersList)
incomingTeleports ← getIncomingTeleporter()

if (not isEmptyPlayerList and isTorpedoAvailable and isPriorHit) then
    /* menembak */

    headingAksi ← getHeadingBetween(bot, nearestPlayer)
    → FIRETORPEDOES

// KASUS PINDAH 1

biggerPlayer ← getBiggerPlayerInRange(playerDangerRange)
if ((not isEmptyBiggerPlayer or not isEmptyIncomingTeleports) and
    (sizeDifference < offset)) then
    directionVectors.add(weightedEscapeFromPlayerVector)

/* AFTERBURNER */
/* OFFENSIVE */

List<GameObject> preys = getSmallerPlayer()
/* Kalau belum nyala, kalau ada preys, dan tidak ada player lebih besar di
sekitar */
```

```

if (not isEmptyPreys and isEmptyDirectionVectors and not isAfterburner)
then
    if (isNearestPreyReachable) then
        headingAksi ← getHeadingBetween(bot, nearestPrey)
        → STARTAFTERBURNER

/* Kalau sedang nyala, kalau ada player lebih besar di sekitar, dan sedang
afterburner atau
gaada preys di sekitar dan sedang afterburner */
if (isNotSafe or isEmptyPreys and isAfterburner) then
    → STOPAFTERBURNER

/* Kalau ga ada player lebih besar di sekitar dan lagi ngejar preys dan
sedang afterburner */
if (isSafe and not isEmptyPreys and isAfterburner) then
    /* Kalau ternyata malah bahaya bisa mati */
    if (notSafe) then
        → STOPAFTERBURNER
    else
        /* Kalau aman */
        headingAksi ← getHeadingBetween(bot, nearestPrey)
        → FORWARD

incomingTorpedo <- getIncomingTorpedo()

if (isThereIncomingTorpedo and torpedoInDangerZone) then
/* Jika ada torpedo yang mengarah ke kita dan jika torpedo (terdekat) di
dalam danger zone kita */
    if (shieldAvailable and bot.size > 60 and isNearTorpedo) then
        → ACTIVATESHIELD
    if (torpedoAvailable)) {
        /* Defend with shooting */
        headingAksi ← getHeadingBetween(bot, incomingTorpedo)
        → FIRETORPEDOES
    } else {
        /* jika ada >= 2 torpedo datang && jaraknya sudah lumayan dekat

```



```
        saat di state ini prioritas lebih rendah dari defend with shooting
    */

    if (incomingTorpedo.size() >= 2 and shieldAvailable and size > 40
and DistanceBetween(bot, torpedo) < 60) then
        → ACTIVATESHIELD

// KASUS PINDAH 2
if (isThereIncomingTorpedo and not torpedoInDangerZone) then
/* jika torpedo terdetekt mengarah ke kita tetapi bukan dalam danger zone
*/
    directionVectors.add(weightedEscapeFromTorpedoVector)

// KASUS PINDAH 3
if (not isEmptyPreys or (isThereIncomingTorpedo and sizeDifference >
offset)) then
    directionVectors.add(weightedChasePreysVector)

// KASUS PINDAH 4
// jika keluar map
if (bot.isOutsideMap) then
    directionVectors.add(weightedToCenterVector)

// KASUS PINDAH 5
incomingSupernova ← getIncomingSupernova()
if (isThereIncomingSupernova) then
/* ada supernova bomb mengarah ke kita */
    directionVectors.add(weightedEscapeFromSupernovaBombVector)

// KASUS PINDAH 6
// jika masuk cloud
if (isNearCloud) then
    directionVectors.add(weightedEscapeFromCloudVector)

// KASUS PINDAH 7
// jika masuk asteroid
if (isNearAsteroid) then
    directionVectors.add(weightedEscapeFromAsteroidVector)
```

```
// KASUS PINDAH 8
// jika punya superfood
if (superFoodEffectActive and foodAvailable) then
    directionVectors.add(weightedChaseFoodVector)

// PERHITUNGAN PERPINDAHAN BERDASARKAN TIAP WEIGHT
if (isNotEmptyDirectionVectors) then
    res ← calculateResult(directionVectors)
    if (not isZero(res)) then
        headingAksi ← toDegree(res)
        → FORWARD

// KASUS SELANJUTNYA ADALAH KASUS TIDAK ADA YANG PERLU DIKEJAR ATAU
// DIHINDARI
if (supernovaPickupExist and botNearestfromPickup) then
    tickTimer ← 10;
    headingAksi ← getHeadingBetween(bot, superNovaPickup)
    → FORWARD

if (bot.supernovaAvailable and isSupernovaFired and tickTimer <= 0) then
/* Punya supernova pickup */
    // players sudah terurut dari terdekat
    headingAksi ← getHeadingBetween(bot, biggestOtherBot)
    → FIRESUPERNOVA
}

if (not isTeleporterFired and bot.teleporterAvailable) then
    foods ← getAllFoodsNearPlayer
else
    foods ← getAllFoodsInWorld

if (isFoodExist) then
    headingAksi ← getHeadingBetween(bot, food)
    → FORWARD
else if (not isTeleporterFired and isTeleportAvailable(bot)) then
    → TELEPORT
```

```
if (not foodsEmpty) then
    headingAksi ← getHeadingBetween(bot, food)
else
    headingAksi ← getHeadingBetween(bot, centerPoint)
isTeleporterAttacking ← false
else
    → STOP
```

#### 4.2. Penjelasan Struktur Data pada *Bot* Permainan *Galaxio*

Implementasi algoritma keputusan *bot* yang dibuat banyak melibatkan struktur data list (class *ArrayList* dalam bahasa java) serta *hash map*. List banyak digunakan dalam menyimpan berbagai objek dengan kelas *GameObject* serta *EscapeInfo*. *Hash map* digunakan untuk memetakan nilai tipe objek (*ObjectType*) dengan list yang menyimpan objek-objek dengan tipe yang sama. Selain tipe data yang telah disebutkan, berikut kelas-kelas yang digunakan dalam implementasi algoritma *bot* ini.

##### 4.2.1. *GameObject*

*GameObject* adalah kelas yang merepresentasikan semua objek di dalam game *Galaxio* ini. Kelas ini memiliki atribut *gameObjectType* yang menyimpan nilai tipe-tipe objek di dalam game ini: *Player*, *Food*, *Wormhole*, *Gas Cloud*, *Asteroid Field*, *Torpedo*, *Superfood*, *Supernova Pickup*, *Supernova Bomb*, *Teleporter*, dan *Shield*. Setiap objek juga menyimpan atribut *id* dengan kelas *UUID*, *size* dengan kelas *integer*, *speed* dengan kelas *integer*, *heading* dengan kelas *integer*, serta *position* dengan kelas *position*. Untuk objek bertipe *player*, data yang disimpan dalam objek, terdapat tambahan atribut berupa *effectsCode* dengan kelas *integer*, *torpedoSalvoCount* dengan kelas *integer*, *supernovaAvailable* dengan tipe *integer*, *teleporterCount* dengan tipe *integer*, dan *shieldCount* dengan tipe *integer*.

##### 4.2.2. *Position*

*Position* adalah kelas yang merepresentasikan posisi objek di dalam peta game *Galaxio*. Atribut yang disimpan dalam kelas ini adalah *integer* *x* dan *y* yang merepresentasikan koordinat kartesian pada peta permainan.

#### 4.2.3. *PlayerAction*

*PlayerAction* adalah kelas yang merepresentasikan aksi yang dilakukan suatu pemain. *PlayerAction* inilah yang akan dikirimkan sebagai suatu *payload* kepada *Engine*. Kelas ini memiliki beberapa nilai aksi: *forward*, *stop*, *startAfterBurner*, *stopAfterBurner*, *fireTorpedoes*, *fireSupernova*, *detonateSupernova*, *fireTeleport*, *teleport*, dan *activateShield*.

Selain atribut *action* (bertipe *PlayerActions*), kelas *PlayerAction* menyimpan atribut *playerId* dengan kelas *UUID*, dan *heading* dengan kelas *integer*. Atribut *heading* menentukan sudut arah aksi dilakukan dalam peta permainan. Namun, hanya beberapa jenis aksi yang melibatkan data *heading*: *forward* untuk arah gerak player, *fireTorpedoes* untuk arah tembakan torpedo, *fireSupernova* untuk arah tembakan supernova, dan *fireTeleport* untuk arah tembakan teleporter.

#### 4.2.4. *World*

*World* adalah kelas yang merepresentasikan dunia permainan dalam game *Galaxio*. *World* memiliki 3 atribut, yaitu *centerPoint* (titik pusat peta) dengan kelas *Position*, *radius* (jari-jari peta) dengan kelas *integer*, dan *currentTick* (*tick* permainan) dengan kelas *integer*.

#### 4.2.5. *WorldVector*

*WorldVector* adalah kelas yang merepresentasikan suatu vektor dua dimensi dalam game *Galaxio*. Atribut yang ada di dalam *WorldVector* berupa *integer* *x* dan *y* yang merepresentasikan nilai komponen vektor dua dimensi. *WorldVector* diimplementasikan untuk menghitung arah gerak player yang tidak mungkin dilakukan dalam bentuk besar sudut.

#### 4.2.6. *EscapeInfo*

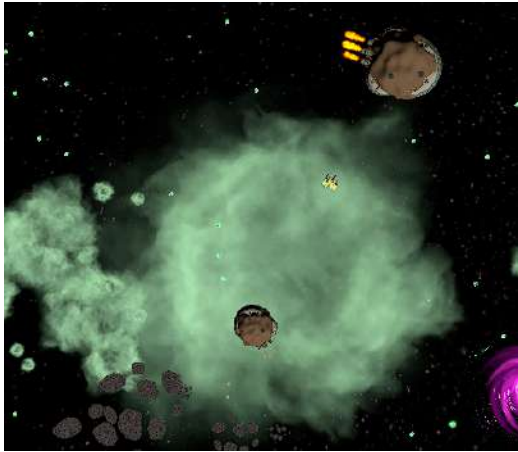

*EscapeInfo* adalah kelas tambahan yang diimplementasikan untuk menentukan arah gerak bot dalam game *Galaxio*. *EscapeInfo* menyimpan 2 atribut, yaitu *escapeDirection* dengan kelas *WorldVector* serta *weight* dengan kelas *integer*. Atribut


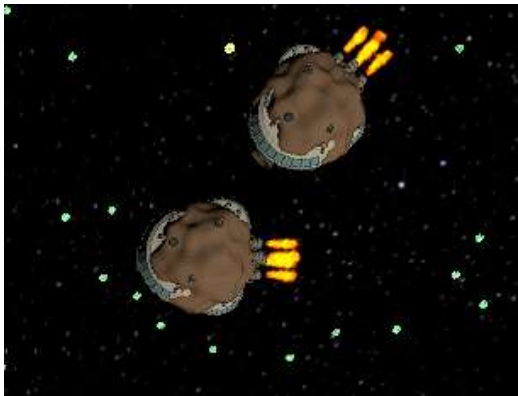
*weight* menyimpan nilai bobot kontribusi arah pergerakan atribut *escapeDirection*. Dalam pembuatan objek dengan kelas *EscapeInfo*, nilai vektor *escapeDirection* akan dinormalisasi terlebih dahulu untuk memudahkan perhitungan.

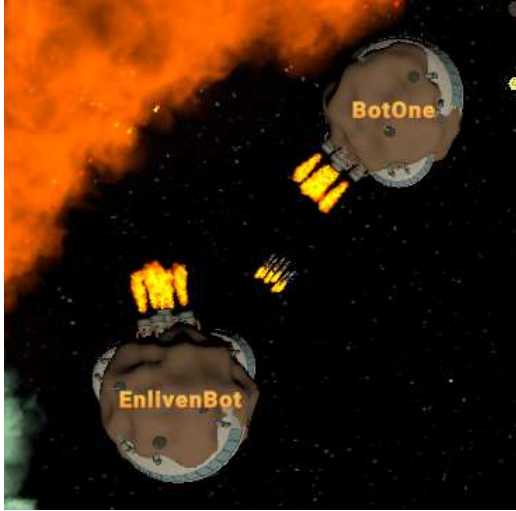


#### 4.2.7. *GameState*



*GameState* adalah kelas yang merpresentasikan keadaan game *Galaxio* pada suatu waktu. *Engine* akan mengirimkan objek dengan kelas *GameState* ke *bot service* agar bot dapat mengakses data yang diperlukan untuk kalkulasi pergerakan selanjutnya. *GameState* memiliki atribut *world* dengan kelas *World*, *gameObjects* dengan tipe *list of GameObject*, dan *playerGameObjects* dengan tipe *list of GameObject*.

### 4.3. Analisis Desain Solusi Algoritma *Greedy* pada Pengujian *Bot*

No	Lampiran	Penjelasan
1	 <pre> ===== Ticks : 193 Action : DETONATESUPERNOVA Heading : 342 Description : Detonate supernova to kill player ===== </pre>	Detonate supernova saat supernova ada di sekitar musuh
2		Teleport ke bot musuh jika size kita lebih besar dari bot musuh sehingga dimakan

	<pre>===== Ticks : 615 Action : TELEPORT Heading : 311 Description : Teleport to target area =====</pre>	
3	 <pre>===== Ticks : 600 Action : FIRETELEPORT Heading : 2 Description : Fire teleport to enemy bot =====</pre>	Menembakkan teleport ke bot yang lebih kecil untuk dimakan
4	 <pre>===== Ticks : 70 Action : STOPAFTERBURNER Heading : 278 Description : Incoming danger or no bot to chase =====</pre>	Stop afterburner jika <i>no bot to chase</i> , (context: bot musuh pada gambar ini makan banyak food)

5	 <pre> ===== Ticks : 266 Action : FIRETORPEDOES Heading : 42 Description : Shoot torpedos to closest bot ===== </pre>	<p>EnlivenBot menembak torpedo ke arah BotOne sebagai <i>bot</i> terdekat</p>
6	 <pre> ===== Ticks : 418 Action : FORWARD Heading : 9 Description : Avoid [bigger bot]   Chasing ===== </pre>	<p>EnlivenBot (yang kanan) kabur dari bot yaang lebih besar dari dirinya</p>
7		<p>Mengecek kondisi lalu active afterburner jika bot musuh dekat dan memenuhi kondisi</p>

	<pre> ===== Ticks : 119 Action : STARTAFTERBURNER Heading : 322 Description : Active afterburner to chase player ===== </pre>	
8	 <pre> ===== Ticks : 134 Action : STOPAFTERBURNER Heading : 267 Description : Stop afterburner (size limit exceeded) ===== </pre>	Stop afterburner karena tidak memenuhi kondisi
9	 <pre> ===== Ticks : 127 Action : FORWARD Heading : 280 Description : Chasing player using afterburner ===== </pre>	EnlivenBot mengejar Dotjar memakai afterburner

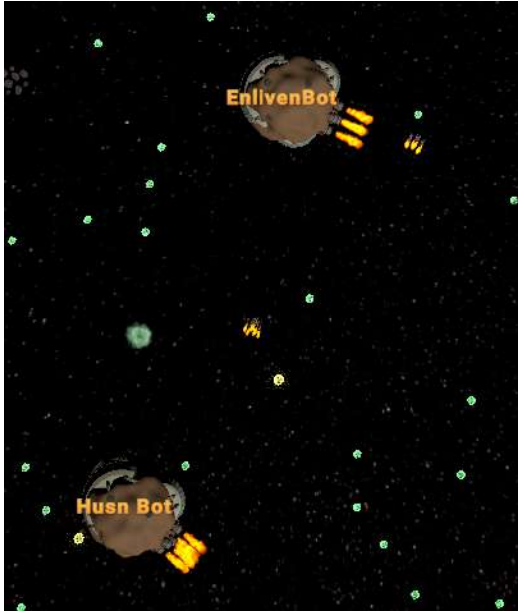




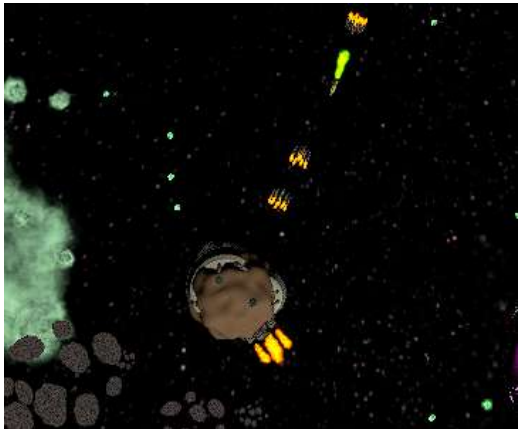
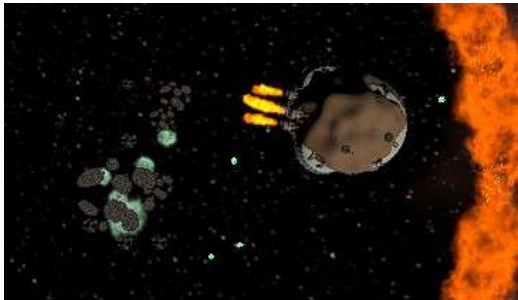
10


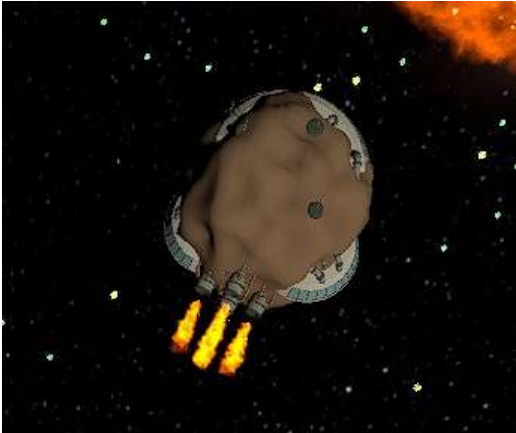


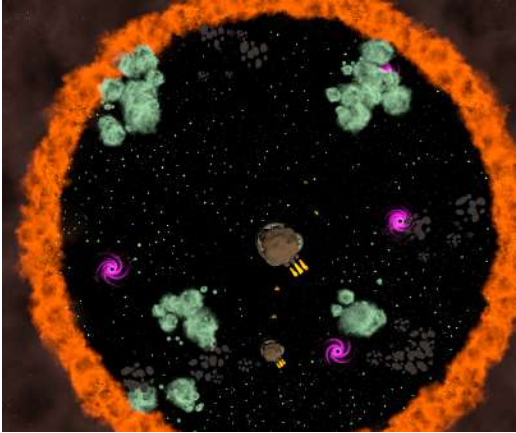
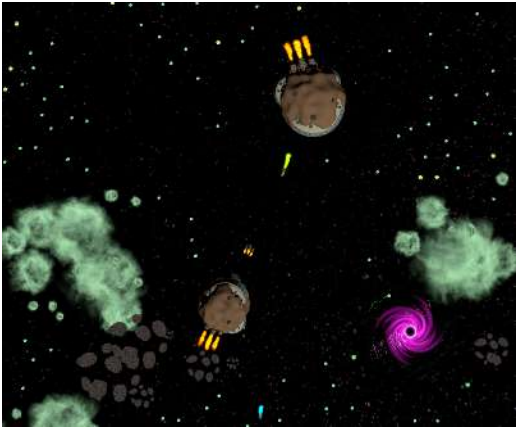
```
=====
Ticks : 167
Action : FIRETORPEDOES
Heading : 299
Description : Shooting incoming torpedos
=====
```

EnlivenBot menembakkan *defensive* torpedo ke arah torpedo lawan

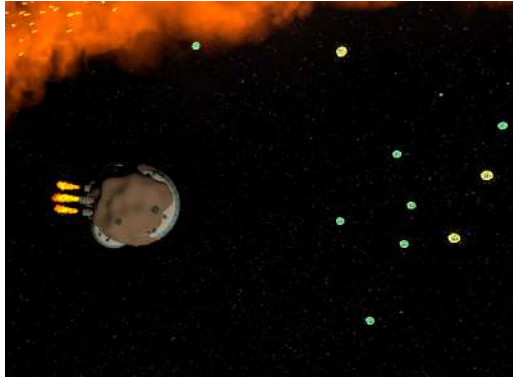
11	 <pre data-bbox="418 840 938 987">===== Ticks : 184 Action : FORWARD Heading : 212 Description : Avoid [incoming torpedos]   Chasing =====</pre>	EnlivenBot berhasil menghindari dari torpedo yang ditembakkan Husn Bot
12	 <pre data-bbox="418 1470 938 1617">===== Ticks : 408 Action : FORWARD Heading : 321 Description : Avoid [incoming torpedos]   Chasing =====</pre>	EnlivenBot mengejar bot lebih kecil

13	 <pre>===== Ticks : 182 Action : FORWARD Heading : 298 Description : Avoid [map border]   Chasing =====</pre>	EnlivenBot menjauh dari <i>World Border</i>
14	 <pre>===== Ticks : 665 Action : FORWARD Heading : 19 Description : Avoid [incoming torpedos] [gas clouds] [asteroid fields]   Chasing[] =====</pre>	Menghindar supernova (pada gambar ini karena ada gas clouds, asteroid dan torpedo jadinya tidak terlalu terlihat, selain itu aksi susah didapat karena jarang ditembak supernova)
15		EnlivenBot menjauh dari <i>gas clouds</i>

	<pre> ===== Ticks : 664 Action : FORWARD Heading : 25 Description : Avoid [incoming torpedos] [gas clouds] [asteroid fields]   Chasing[] ===== </pre>	
16	 <pre> ===== Ticks : 191 Action : FORWARD Heading : 177 Description : Avoid [incoming torpedos] [asteroid fields] ===== </pre>	EnlivenBot menjauh dari <i>asteroid field</i>
17	 <pre> ===== Ticks : 378 Action : FORWARD Heading : 301 Description : Getting nearest food ===== Ticks : 379 Action : FORWARD Heading : 325 Description : Avoid[]   Chasing[foods] ===== </pre>	Mencari makanan saat mendapat <i>super food</i>

18	 <pre>===== Ticks : 103 Action : FORWARD Heading : 113 Description : Getting supernova pickup =====</pre>	Menuju ke (0, 0) untuk mengambil supernova pickup
19	 <pre>===== Ticks : 183 Action : FIRESUPERNOVA Heading : 256 Description : Shooting supernova =====</pre>	

20



```
=====  
Ticks : 229  
Action : FORWARD  
Heading : 358  
Description : Getting nearest food  
=====
```

sebelum teleport



```
=====  
Ticks : 408  
Action : FORWARD  
Heading : 164  
Description : Avoid [map border] | Chasing  
=====
```

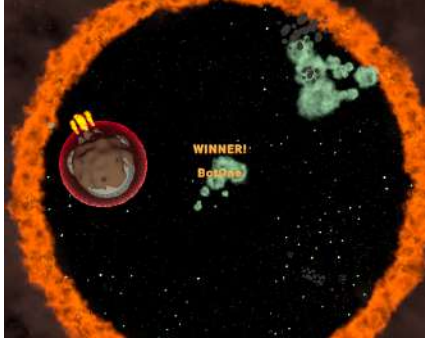
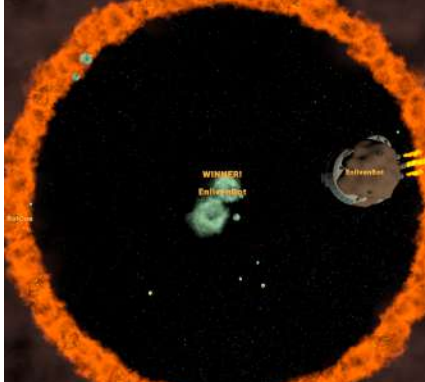
```
=====  
Ticks : 409  
Action : FIRETELEPORT  
Heading : 157  
Description : Teleporting to foods  
=====
```

setelah teleport


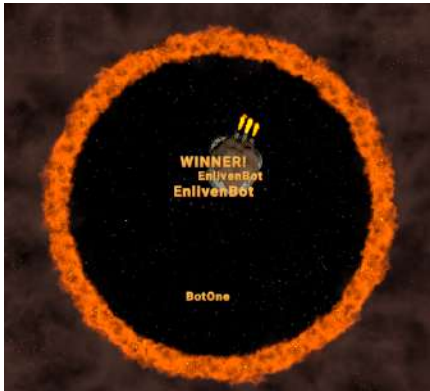
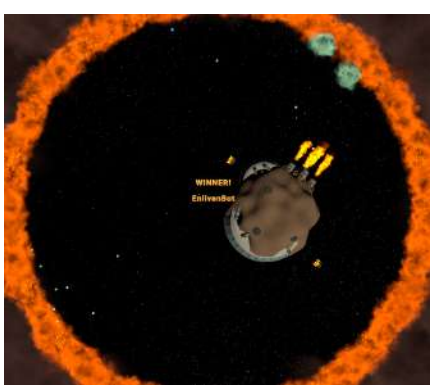
Mengejar *food* terdekat, jika tidak ada food disekitar maka akan teleport ke food dengan jumlah food yang banyak

	 <pre> ===== Ticks : 436 Action : TELEPORT Heading : 18 Description : Teleport to target area ===== </pre>	
--	---	--

Hasil BO5 dari pertandingan EnlivenBot, Husn Bot, YasinBot, BotOne

No Pertandingan	Lampiran Hasil Pertandingan	Urutan
1		<ol style="list-style-type: none"> <li>1. BotOne</li> <li>2. EnlivenBot</li> <li>3. YasinBot</li> <li>4. Husn Bot</li> </ol>
2		<ol style="list-style-type: none"> <li>1. EnlivenBot</li> <li>2. BotOne</li> <li>3. YasinBot</li> <li>4. Husn Bot</li> </ol>



3		<ol style="list-style-type: none"> <li>1. EnlivenBot</li> <li>2. YasinBot</li> <li>3. BotOne</li> <li>4. Husn Bot</li> </ol>
4		<ol style="list-style-type: none"> <li>1. EnlivenBot</li> <li>2. BotOne</li> <li>3. YasinBot</li> <li>4. Husn Bot</li> </ol>
5		<ol style="list-style-type: none"> <li>1. EnlivenBot</li> <li>2. BotOne</li> <li>3. YasinBot</li> <li>4. Husn Bot</li> </ol>

Dapat dilihat dari hasil pertandingan Best Of 5 kami dengan BotOne, YasinBot, dan Husn Bot bahwa *bot* kami mendapat *winrate* 80%. *Win rate* ini tergolong tinggi sehingga dapat disimpulkan bahwa strategi *greedy* yang kami implementasi sudah cukup bagus. Namun, hal ini belum tentu dapat menjadi acuan karena *bot* lawan kami masih sangat memungkinkan untuk diperbagus lagi oleh pembuatnya atau ada *bot* lain yang tidak kami lawan yang lebih bagus.



## Bab 5: Kesimpulan, Saran, Komentar, dan Refleksi

### 5.1. Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma *greedy* untuk membuat bot permainan Galaxio yang dapat mencapai tujuan objektif, yaitu bertahan hidup hingga akhir permainan. Namun, tentunya algoritma yang kita buat jauh dari kata sempurna, masih banyak kekurangan-kekurangan yang ada dikarenakan algoritma ini *greedy*, fokus program, yakni mengambil keuntungan yang sebanyak-banyaknya.

Secara umum, bot yang kami buat berhasil membuktikan bahwa strategi *greedy* cukup baik digunakan dalam pembuatan bot, didukung dengan keberhasilan pengujian melawan bot – bot lain yang ada. Walaupun demikian, tetap terdapat faktor *luck* dalam map, pemetaan objek di game, dan lain-lain. Hal itulah yang membuat hasil algoritma tidak konsisten dan tidak bisa dikatakan paling baik dibandingkan yang lain.

### 5.2. Saran

Terkait dengan tugas ini, berikut beberapa saran yang dapat dilakukan untuk kedepannya:

- Pembagian tugas dilakukan lebih awal agar *workload* masing-masing anggota jelas dan pengerjaan lebih terstruktur.
- Pembuatan laporan dapat dicicil agar dapat selesai lebih cepat dan tidak terbebani saat mepet dengan *deadline*.
- Pengujian dapat dilakukan dengan melawan bot lain yang lebih adil lagi. Salah satunya dengan membuat *map* yang statis sehingga dapat mengurangi faktor *luck* yang berpengaruh pada permainan.

### 5.3. Komentar

Tugas yang diberikan menarik, kreatif, dan mendorong kita untuk lebih *explore* terkait tugas ini. Permainan yang diberikan sangat menarik dan seru. Namun, pada permainan tetap terdapat faktor *luck* dalam segi *map* permainan yang berubah-ubah dan dapat merugikan salah satu pemain. Selain itu, permainan ini juga terdapat *bug* yang belum diselesaikan hingga sekarang dan *bug* tersebut sangat mengganggu dalam proses pengembangan algoritma.

#### 5.4. Refleksi

Refleksi dari tugas kali ini, yakni kurangnya persiapan yang matang, terlalu fokus pada satu hal sehingga pekerjaan yang lain keteteran. Hal ini menyebabkan pengerjaan tugas terlalu mepet *deadline*. Selain itu juga, terlalu ingin menang setiap permainan padahal fokus utama pada tugas kali ini adalah implementasinya pada Algoritma Greedy.

## Daftar Pustaka

<https://gist.github.com/gunvirranu/6816d65c0231981787ebefd3bdb61f98>

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

[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)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf)

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

### **Link Penting**

Link Github : [https://github.com/Altair1618/Tubes1\\_EnlivenSquad](https://github.com/Altair1618/Tubes1_EnlivenSquad)

Link Video : <https://www.youtube.com/watch?v=90fbda1WG1o>