

LAPORAN TUGAS BESAR
Pemanfaatan Algoritma Greedy dalam Aplikasi
Permainan “Overdrive”

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester I Tahun Akademik 2021/2022



Disusun oleh:

Adiyansa Prasetya Wicaksana (K2)	13520044
Rheza Rizqullah Ecaldy (K3)	13520060
Sarah Azka Arief (K2)	13520083

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021

DAFTAR ISI

BAB I DESKRIPSI TUGAS	1
BAB II LANDASAN TEORI	3
BAB III APLIKASI STRATEGI GREEDY	12
BAB IV IMPLEMENTASI DAN PENGUJIAN	21
BAB V KESIMPULAN DAN SARAN	43
DAFTAR PUSTAKA	45

BAB I

DESKRIPSI TUGAS

Overdrive adalah sebuah game yang mempertandingan 2 bot mobil dalam sebuah ajang balapan. Setiap pemain akan memiliki sebuah bot mobil dan masing-masing bot akan saling bertanding untuk mencapai garis finish dan memenangkan pertandingan. Agar dapat memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu untuk dapat mengalahkan lawannya.

Pada tugas besar ini, setiap kelompok diminta untuk membuat sebuah bot yang akan diadu dengan bot kelompok lain pada permainan Overdrive di kompetisi Tubes 1. Bot tersebut mengimplementasikan algoritma *greedy* yang berperan sebagai strategi bot agar dapat menyelesaikan fungsi objektif dari permainan Overdrive, yaitu memenangkan permainan dengan cara mencapai garis finish lebih awal atau mencapai garis finish bersamaan tetapi dengan kecepatan lebih besar atau memiliki skor terbesar jika kedua komponen tersebut masih bernilai imbang.

Game engine untuk tugas besar ini dapat diperoleh pada laman <https://github.com/EntelectChallenge/2020-Overdrive> dengan implementasi strategi greedy pada bot dilakukan dengan melanjutkan program *starter bot* yang berada di *starter pack* pada tautan berikut <https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>. Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang terdapat pada *game engine* pada tautan di atas. Beberapa aturan umum yang berlaku dapat dilihat pada tautan berikut <https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md>.

Agar dapat menjalankan permainan, beberapa *requirement* dasar yang dibutuhkan adalah:

- a. Java (minimal Java 8): <https://www.oracle.com/java/technologies/downloads/#java8>
- b. IntelliJ IDEA: <https://www.jetbrains.com/idea/>
- c. NodeJS: <https://nodejs.org/en/download/>

Bot harus dibuat dengan menggunakan bahasa Java dan di-build menggunakan IntelliJ sebelum menjalankan permainan kembali. Dilarang menggunakan kode program yang sudah ada untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang. Hasil pertandingan dapat divisualisasikan melalui visualizer pada tautan <https://github.com/Affuta/overdrive-round-runner>.

Beberapa hal lainnya yang perlu dicatat yakni strategi *greedy* harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Selain itu, setiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Terakhir, implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB II

LANDASAN TEORI

1. Algoritma Greedy

Algoritma greedy adalah algoritma yang menyelesaikan persoalan secara langkah per langkah. Algoritma greedy mengikuti prinsip “take what you can get now”, yaitu mengambil pilihan yang terbaik pada langkah tertentu tanpa mempertimbangkan konsekuensinya untuk langkah selanjutnya. Jadi, pada algoritma greedy, dipilih solusi optimum lokal pada setiap langkah dengan harapan bahwa pilihan ini akan mengarah ke solusi optimum global. Namun, algoritma greedy tidak selalu menghasilkan solusi optimum. Hal ini diakibatkan algoritma greedy tidak melakukan pengecekan secara menyeluruh pada seluruh opsi yang ada dan strategi greedy yang digunakan tidak selalu sesuai dengan persoalan yang ada.

Algoritma greedy umumnya memiliki elemen-elemen berikut :

a. Himpunan Kandidat (C)

Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah untuk membentuk solusi (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)

b. Himpunan Solusi (S)

Himpunan yang berisi kandidat yang terpilih sebagai solusi

c. Fungsi Solusi

Fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi

d. Fungsi Seleksi

Fungsi yang memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik

e. Fungsi Kelayakan (Feasible)

Fungsi yang memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)

f. Fungsi Obyektif

Fungsi yang mengoptimalkan solusi

2. Game Engine

Game Engine adalah perangkat lunak yang membantu dalam pembuatan dan pengembangan permainan. Fungsi utama *game engine* adalah membuat proses pembuatan dan pengembangan permainan lebih ekonomis karena cukup dengan satu *game engine* mampu membuat banyak permainan dengan jenis yang berbeda-beda. Salah satu *game engine* yang populer digunakan adalah *Unity*.

3. Game Engine Entelect Overdrive

Dalam *game* Entelect 2020 - Overdrive terdapat beberapa komponen yang diperlukan agar game dapat berjalan:

a. Game Engine

Berfungsi untuk membuat *bot* pemain agar mematuhi segala aturan yang telah didefinisikan oleh Entelect. *Game engine* akan menyalurkan perintah yang dikirimkan oleh *bot* ke dalam *game* untuk diproses lebih lanjut sesuai dengan aturan yang telah didefinisikan.

b. Game Runner

Berfungsi untuk menjalankan pertandingan antar pemain (*bot* yang dibuat oleh pemain). *Game runner* membaca perintah dari *bot* lalu memberikannya kepada *game engine* untuk diolah.

c. Reference bot

Placeholder bot yang memiliki perintah-perintah dan logika dasar agar dapat ditandingkan dengan *starter bot* yang nantinya akan dimodifikasi sesuai dengan keinginan pemain. *Reference bot* tersedia dalam bahasa Java.

d. Starter bot

Bot yang nantinya akan dimodifikasi oleh pemain sesuai dengan keinginan pemain. Pada awalnya, *starter bot* berisi dengan logika dasar, fungsi dasar, dan semua objek yang diperlukan untuk bermain dalam game tersebut. Tersedia dalam berbagai bahasa dan untuk tugas besar kali ini, akan digunakan bahasa Java.

4. Detail Starter Pack dari Entelect 2020 - Overdrive

Entelect 2020 - Overdrive menyediakan versi yang telah di-build sehingga pemain bisa langsung memainkan *game*-nya atau memodifikasi *bot*. Sehingga, hal yang perlu dilakukan pertama kali adalah dengan mengunduh *starter pack* versi terbaru yang berisi *game engine* dan *game runner* yang telah di-built serta berisi serta *bot* dari Overdrive pada laman

berikut (<https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>). Pada tugas besar kali ini, akan dibangun sebuah *bot* dengan bahasa pemrograman Java. *Folder Java di starter bot* terdiri atas dua *folder* src dan target (hasil *build*) serta dua *file* bot.json (identitas *bot*) dan pom.xml (untuk melakukan *build*).

📁 src	05/02/2022 19:30	File folder	
📁 target	06/02/2022 21:04	File folder	
📄 .gitignore	05/02/2022 19:30	GITIGNORE File	2 KB
📄 bot.json	05/02/2022 19:30	JSON File	1 KB
📄 pom.xml	05/02/2022 19:30	XML Document	2 KB

Gambar 2.1. Struktur Folder Java Starter Bot

Pembuatan *bot* itu sendiri akan dilakukan pada folder src yang berisi *source code*. Folder src berisi nested folder sehingga membuat path “src\main\java\za\co\entelect\challenge”. Folder path itu terdiri atas tiga *folder* dan dua *file* yang telah disediakan, yaitu:

a. *Folder command*

Berisi seluruh perintah yang dapat digunakan oleh *bot* pada game Overdrive. Perintah yang disediakan adalah *Accelerate*, *Boost*, *ChangeLane*, *Decelerate*, *DoNothing*, *Emp*, *Fix*, *Lizard*, *Oil*, dan *Tweet*.

b. *Folder entities*

Berisi entitas/objek yang terdapat pada game overdrive. Objek yang tersedia adalah Car, GameState, Lane, dan Position. Objek Car berisi status pada bot yang sedang bermain, status yang tersedia adalah *speed*, *damage*, *state*, *powerups*, *boosting*, dan *boostCounter*. Objek GameState berisi status permainan yang sedang terjadi yaitu *currentRound*, *maxRounds*, *player*, *opponent*, dan *World Map*. Objek Lane berisi *map* tempat permainan sedang dilakukan. Dan terakhir, Objek Position berisi posisi bot dalam bentuk kartesian (x dan y).

c. *Folder enums*

Berisikan konstanta ataupun status yang lebih detail dari entitas/objek game. Folder ini terdiri atas Direction yang berarti gerak *bot*, PowerUps yang berarti *power up* yang tersedia pada permainan, State berarti status yang bisa dialami oleh *bot*, dan Terrain yang berarti objek yang terdapat pada peta permainan.

d. Bot.java

File ini berisikan logika dari *bot* yang dibangun. Pada *file* ini akan diimplementasikan objek-objek yang terdapat pada folder sebelumnya dan penerapan algoritma *greedy*. Semua strategi yang ditulis akan terdapat pada fungsi *run()* pada kelas Bot.

e. Main.java

File ini bertanggung jawab untuk menjalankan strategi yang diterapkan pada Bot.java dengan menginstansiasi *bot* pada setiap ronde dan menerima perintah dari bot tersebut untuk diproses lebih lanjut oleh *game engine*.

📁 command	05/02/2022 19:30	File folder
📁 entities	05/02/2022 19:30	File folder
📁 enums	05/02/2022 19:30	File folder
📄 Bot.java	05/02/2022 19:30	JAVA File 2 KB
📄 Main.java	05/02/2022 19:30	JAVA File 2 KB

Gambar 2.2. Struktur Folder src pada Java Starter Bot

Untuk dapat menjalankan *bot* serta melakukan pertandingan dibutuhkan beberapa aplikasi yang perlu di-*install*, yaitu Java Development Kit (minimal Java 8), IntelliJ IDEA, dan NodeJS. Untuk mendapatkan *file executable* Java (.jar), *file* yang terdapat di dalam folder starter-bots harus di-*build* menggunakan IntelliJ IDEA. Bila *path* yang terdapat pada game-runner-config.json belum tepat, ganti isi dari “player-a” menjadi “./starter-bots/java”. Setelah itu, bisa dijalankan run.bat yang terdapat pada *root* folder starter-pack. Selain itu juga, ada beberapa informasi tambahan mengenai file konfigurasi yang terdapat di dalam *game*:

- a. game-runner-config.json, letaknya pada *root* folder starter-pack, digunakan untuk mengatur lokasi *file bot* pemain A dan B, *file* hasil pertandingan, konfigurasi game, dan game engine.
- b. game-config.json, letaknya juga pada *root* folder starter-pack, digunakan untuk mengatur konfigurasi dasar dari *game* Overdrive seperti panjang dan lebar dari *map*.
- c. bot.json, letaknya pada *folder* java yang ada pada *bot*, berisi identitas dari *bot* yang akan ditandingkan.

Pertandingan antara dua *bot* secara otomatis akan dijalankan pada *command line* sehingga disediakan opsi untuk menggunakan *visualizer*, terdapat dua *visualizer*. Yang pertama <https://github.com/Affuta/overdrive-round-runner>, harus di-*build* terlebih dahulu secara lokal dan agak sulit untuk melihat lebih dari satu permainan. Yang kedua

<https://entelect-replay.raezor.co.za/>, dapat diakses melalui website dan hanya perlu memasukkan match-log dari pertandingan dalam bentuk zip dan bisa memasukkan lebih dari satu match, sehingga lebih mudah untuk melakukan analisis. Kelompok kami, menggunakan *visualizer* kedua yang relatif lebih *simple*.

5. Mengembangkan Starter Bot

Starter Bot yang telah disediakan dari *starter pack* sebenarnya sudah cukup untuk langsung diberikan strategi, namun ada beberapa hal yang perlu dilengkapi agar strategi yang ingin kelompok kami implementasikan dapat terealisasi. Salah satunya adalah pada *folder* enums *file* PowerUps.java (berisi *power up* yang dimiliki oleh *bot* pada permainan) hanya berisi *boost* dan *oil* saja, sehingga perlu ditambahkan *power up* lain yaitu *tweet*, *lizard*, dan *EMP*.

```
package za.co.entelect.challenge.enums;

import com.google.gson.annotations.SerializedName;

public enum PowerUps {
    @SerializedName("BOOST")
    BOOST,
    @SerializedName("OIL")
    OIL,
    @SerializedName("TWEET")
    TWEET,
    @SerializedName("LIZARD")
    LIZARD,
    @SerializedName("EMP")
    EMP;

}
```

Selain itu, pada *file* Lane.java yang terletak pada *folder entities*, kelompok kami menambahkan cyber truck yang merupakan *boolean* yang terhubung pada kondisi isOccupiedByCybertruck pada game state. Penambahan pada Lane.java tersebut dilakukan agar implementasi cybertruck dengan cara mendeteksi apakah suatu block dalam lane tertentu mengandung cybertruck dapat dilakukan.

```
package za.co.entelect.challenge.entities;

import com.google.gson.annotations.SerializedName;
import za.co.entelect.challenge.enums.Terrain;

public class Lane {
    @SerializedName("position")
    public Position position;

    @SerializedName("surfaceObject")
    public Terrain terrain;

    @SerializedName("occupiedByPlayerId")
    public int occupiedByPlayerId;

    @SerializedName("isOccupiedByCyberTruck")
    public boolean cybertruck;

}
```

Setelah mempersiapkan semua kelas dan elemen baru yang diperlukan, langkah selanjutnya adalah memasukan strategi *greedy* yang diinginkan pada file Bot.java. *Starter bot* sudah menyediakan beberapa *method* dan *attribute* awal yang bisa digunakan dalam pengembangan *bot*. *Method* dan *attribute* tersebut bisa dimodifikasi ataupun ditambahkan lebih lanjut sesuai keinginan dari pemain. Semua strategi *greedy* yang ditulis akan diimplementasikan pada *method* run dan *method* run ini harus mengembalikan perintah yang valid, jika perintah yang dikembalikan dari *method* tersebut tidak valid maka *game engine* secara otomatis akan menganggap *bot* tersebut tidak melakukan apapun (*command nothing*). Struktur program Bot.java yang akan dimodifikasi secara *greedy* adalah sebagai berikut.

```
package za.co.entelect.challenge;

import za.co.entelect.challenge.command.*;
import za.co.entelect.challenge.entities.*;
import za.co.entelect.challenge.enums.PowerUps;
```

```
import za.co.entelect.challenge.enums.Terrain;

// import semua module yang dibutuhkan

public class Bot {
    // Inisialisasi konstanta dan command
    public Bot() {
        this.random = new SecureRandom();
        directionList.add(TURN_LEFT);
        directionList.add(TURN_RIGHT);
    }

    public Command run(GameState gameState) {
        // Strategi greedy
    }
}
```

6. Permainan Overdrive

Pada permainan Overdrive, setiap *bot* memiliki aspek *speed*, *position*, *damage*, *powerups*, dan *score*. Pertandingan akan terdiri atas 2 *bot* yang akan diadu satu sama lain dalam sebuah 2D *array* yang terdiri atas 4 *lane* dan 1500 *block*. *Bot* tersebut memiliki *visibility* sebanyak 20 *block* ke depan dan 5 *block* ke belakang. Terdapat 5 tipe block sebagai berikut:

- a. Empty

Block ini tidak berisi apapun

- b. Mud

Block ini berisi lumpur yang akan menambah *damage* sebanyak 1 dan mengurangi skor dari *bot* sebanyak 3. Pada kondisi ini, *speed* akan dikurangi ke *previous speed state*

- c. Oil spill

Block ini berisi *oil* yang akan memperlambat *speed* *bot* menjadi *previous speed state*.

Skor dari *bot* akan berkurang 4 dan *damage* akan bertambah 1

- d. Flimsy wall

Block ini akan mengurangi *speed* dari *bot* menjadi 3 (SPEED_STATE_1), mematikan efek BOOST apabila sedang digunakan oleh *bot*, dan menambah *damage* sebanyak 2.

- e. Finish line

Block ini merupakan garis *finish* dari permainan

Pada awal permainan, *bot* akan memulai dengan kecepatan sebesar 5. Kecepatan dari *bot* didasarkan pada skala kecepatan alias *speed scaling* yang bersifat final. *Speed scaling* yang terdapat pada permainan ini adalah sebagai berikut:

- a. MINIMUM_SPEED = 0
- b. SPEED_STATE_1 = 3
- c. INITIAL_SPEED = 5
- d. SPEED_STATE_2 = 6
- e. SPEED_STATE_3 = 8
- f. MAXIMUM_SPEED = 9
- g. BOOST_SPEED = 15

Pada setiap ronde, setiap pemain akan memberikan sebuah perintah alias *command* untuk *botnya* yang akan dieksekusi pada waktu yang bersamaan bagi perintah dari kedua pemain (tidak sekuensial). Setiap perintah dari pemain akan divalidasi sebelum perintah tersebut dieksekusi. Perintah yang tidak valid akan mengakibatkan *bot* untuk tidak melakukan apa-apa selama satu ronde. Terdapat 11 *command* yang dapat dilakukan oleh *bot* yakni sebagai berikut:

- a. NOTHING: tidak melakukan apa-apa (*speed* dan *lane* tetap sama)
- b. ACCELERATE: menambah *speed* menjadi *speed state* setelahnya
- c. DECELERATE: mengurangi *speed* menjadi *speed state* sebelumnya
- d. TURN_LEFT: *lane* menjadi *lane* yang ada di kiri
- e. TURN_RIGHT: *lane* menjadi *lane* yang ada di kanan
- f. USE_BOOST: menggunakan *powerup* boost
- g. USE_OIL: menggunakan *powerup* oil
- h. USE_TWEET <*lane*> <*block*>: menggunakan *powerup* tweet
- i. USE_LIZARD: menggunakan *powerup* lizard
- j. USE_EMP: menggunakan *powerup* EMP
- k. FIX: mengurangi *damage*

Setiap kali *bot* mengambil atau menggunakan *powerup*, skor akan bertambah sebanyak 4. Berikut 5 *powerups* yang terdapat pada permainan ini:

a. Oil item

Menaruh *oil block* pada *block* yang sedang ditempati olehnya. *Block* tersebut kemudian akan menjadi *block* bertipe *oil spill* dan apabila *bot* lawan terkena *oil spill* maka akan mendapatkan efek seperti pada deskripsi tipe *block oil spill*

b. Boost

Meningkatkan *speed bot* menjadi BOOST_SPEED selama 5 turn. Apabila dilakukan *decelerate* secara manual atau karena efek dari *obstacle*, maka *speed* akan berkurang menjadi MAXIMUM_SPEED dan efek *boost* akan gagal

c. Lizard

Memungkinkan *bot* untuk mengabaikan semua tabrakan atau efek yang menambah *damage* kecuali pada *block* terakhir yang ditempati *bot* setelah berjalan

d. Tweet

Meminta Elon Musk untuk menaruh *cybertruck* pada *block* dan *lane* tertentu. Apabila terkena *cybertruck*, suatu *bot* akan tertahan di belakang *cybertruck* hingga akhir ronde, *speed* akan menjadi SPEED_STATE_1, dan *damage* akan bertambah sebesar 2

e. EMP

Menembak EMP ke depan *lane* yang sedang ditempati *bot* serta *lane* yang berada di kiri dan kanan dari *lane bot*. Apabila terkena EMP, suatu *bot* akan berhenti selama satu ronde dan *speed* akan menjadi SPEED_STATE_1

Damage berpengaruh terhadap *speed* maksimal dari suatu *bot*. Untuk mengurangi *damage*, dapat digunakan *command FIX* untuk memperbaiki *bot* dan mengurangi *damage* sebesar 2 dengan catatan *bot* akan diam selama satu ronde dan akan melanjutkan ronde berikutnya dengan *speed* sebesar *speed* sebelum dilakukan FIX. Berikut penskalaan dari *speed* maksimal suatu *bot* berdasarkan *damage* yang dimiliki olehnya:

- a. Damage 5 = Max Speed 0
- b. Damage 4 = Max Speed 3
- c. Damage 3 = Max Speed 6
- d. Damage 2 = Max Speed 8
- e. Damage 1 = Max Speed 9
- f. Damage 0 = Max Speed 15

BAB III

APLIKASI STRATEGI GREEDY

1. Solusi Algoritma Greedy yang Mungkin Dipilih

1.1. *Cautious Greedy*

Cautious Greedy adalah strategi yang bertujuan meminimalkan total *damage* yang diterima *bot*. Strategi ini didasarkan pada *terrain* yang terdapat pada setiap *lane* yang mungkin dilalui oleh *bot*. Dalam implementasinya, strategi ini sama sekali tidak menghiraukan keberadaan *power-ups* dan hanya memilih *lane* dengan mempertimbangkan *terrain* yang ada pada setiap *lane* dan total *damage*-nya.

Secara umum, strategi ini dilakukan dengan mengambil informasi banyaknya *terrain* pada *block* yang berada di *lane* yang sedang ditempati oleh *bot* serta *lane* yang berada di kanan dan kirinya. Untuk setiap *lane*, akan dilakukan pengecekan kondisi dari *block* sebanyak *speed* dari *bot* untuk kasus *command* NOTHING, TURN_LEFT, dan TURN_RIGHT. Kemudian, akan dilakukan dua pengecekan tambahan untuk *lane* yang sedang ditempati oleh *bot* sebesar *speed* sebelumnya dan *speed* setelahnya untuk mensimulasikan hasil dari *command* ACCELERATE dan DECELERATE. Setelah itu, total *damage* dari setiap *lane* akan dihitung dengan mengkategorisasi jenis *terrain* dan menambahkan *damage* sesuai efek dari setiap jenis *terrain*. Total *damage* dari setiap *lane* akan dimasukkan ke dalam sebuah *arraylist of integer* dan akan diurut sehingga didapatkan *lane* dengan total *damage* terkecil.

a. Mapping Element Greedy

i. Himpunan Kandidat

Command yang berkaitan dengan pergerakan dari *bot* dalam permainan Overdrive, yaitu FIX, ACCELERATE, DECELERATE, TURN_RIGHT, TURN_LEFT, dan juga NOTHING.

ii. Himpunan Solusi

Command yang dipilih sesuai dengan kondisi yang memenuhi dan paling optimum dibanding *command* lainnya.

iii. Fungsi Solusi

Memeriksa apakah perintah yang terpilih merupakan perintah yang terdefinisi dalam permainan atau tidak. Jika tidak, maka perintah bukan solusi yang valid dan secara *default* akan mengembalikan NOTHING.

iv. Fungsi Seleksi

Pemilihan *command* berdasarkan jumlah *damage* dari setiap *lane* yang dapat dilalui *bot* serta *speed* dari *bot*. Akan dipilih *command* yang paling memenuhi kondisi pada suatu ronde dengan mengutamakan yang diseleksi melalui *count damage* yang paling minimum sehingga didapat hasil paling optimum.

v. Fungsi Kelayakan

Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi *command* yang dipilih sudah terpenuhi. Validasi dilakukan terhadap validitas dari jumlah *damage* pada *lane*, posisi *bot*, *speed bot*, *speed state* sebelumnya, *speed state* setelahnya, serta *terrain* pada *blocks* yang akan dilewati.

vi. Fungsi Objektif

Meminimalkan total *damage* yang diterima *bot* dengan menghindari *terrain* serta mengutamakan *path planning* yang paling optimal bagi *bot*. Dalam hal ini memanfaatkan pemilihan *speed* dan *lane* dari *bot*.

b. Analisis Efisiensi Solusi

Cautious greedy didasarkan pada pengecekan terhadap jumlah *terrain* pada suatu *lane* yang akan dilalui *bot*. Penghitungan *terrain* dilakukan pada ketiga *lane* yang dapat dilalui oleh pemain dengan jangkauan *block* yang dicek sebesar *speed* dari pemain. Jika dianggap pada setiap *lane* dilakukan pengecekan sebanyak *speed* n, maka proses tersebut akan memakan waktu sebesar 3n. Akan dilakukan dua pengecekan tambahan bagi *speed* sebelumnya dan *speed* setelahnya sehingga didapat $3n + n + n = 5n$. Setelah didapat jumlah *terrain*, akan dilakukan pengecekan jenis pada setiap *terrain*

untuk menghitung total *damage* dari setiap *lane* sehingga untuk m buah *terrain* dibutuhkan $3n + 3m$. Kemudian, seluruh total *damage* akan diurutkan pada suatu *array* sehingga didapat *lane* dengan total *damage* terkecil. Dengan adanya sebanyak 5 total *damage* yang diurutkan dengan 2 berasal dari *speed* setelah dan *speed* sebelum dan 3 dari setiap *lane* dengan *speed* sesuai *speed* dari *bot*, didapat total kompleksitas waktu sebesar $3n + 3m + 5 = O(n)$ sehingga $T(n) = 3n + 3m + 5 = O(n)$ dengan n adalah *speed* dan m adalah banyaknya *terrain*.

c. Analisis Efektivitas Solusi

Strategi *cautious greedy* efektif pada kasus dimana:

- Karena strategi ini mengakibatkan *bot* untuk menjaga maksimum *speed* dengan meminimalisir *damage*, strategi ini akan berguna apabila *bot* lawan tidak menghiraukan *damage* sehingga maksimum *speed* lawan akan dibawah

Namun, strategi ini tidak efektif pada kasus dimana

- *Bot* lawan menggunakan *powerup*, sehingga kesenjangan keuntungan dengan *bot* lawan akan membesar dikarenakan sifat *powerup* yang dapat menguntungkan *bot* lawan dan menghadang *bot* lain selain *bot* lawan

1.2. *Destructive Greedy*

Destructive Greedy adalah strategi yang didasarkan pada *power-ups* yang bersifat destruktif yakni EMP, Tweet, dan Oil. Strategi ini bertujuan memaksimalkan *damage* yang diberikan kepada *bot* lawan melalui ketiga *power-ups* tersebut. Dalam implementasinya, strategi ini secara umum hanya fokus pada *lane* yang penuh *power-ups* yang bersifat destruktif seperti EMP, Tweet, dan Oil serta menentukan kapan *power-ups* tersebut digunakan.

Secara umum, strategi ini dilakukan dengan melakukan pengecekan terhadap setiap *power-ups* destruktif yaitu EMP, Tweet, dan Oil. Untuk setiap *power-ups* tersebut dilakukan pengecekan kondisional untuk menentukan apakah kondisi sekarang merupakan kondisi yang optimal dalam penggunaan setiap *power-ups*

tersebut. Jika kondisi terpenuhi untuk suatu *power-ups* maka *command* untuk *power-ups* akan dimasukkan terlebih dahulu ke dalam sebuah *array*. Setiap *command* yang dimasukkan memiliki sebuah *weight* sebagai prioritas antara *command*. Sebagai salah satu contoh, EMP memiliki *weight* paling tinggi diantara Tweet dan Oil. Akhirnya, *array* yang berisi *weighted command* akan diambil yang paling optimumnya, jika *array* kosong maka akan mengembalikan *command* Nothing (tidak melakukan apapun).

a. Mapping Element Greedy

i. Himpunan Kandidat

Command yang berkaitan dengan penyerangan dalam permainan Overdrive, yaitu USE_EMP, USE_TWEET, USE_OIL, dan juga NOTHING.

ii. Himpunan Solusi

Command yang dipilih sesuai dengan kondisi yang memenuhi dan paling optimum dibanding *command* lainnya.

iii. Fungsi Solusi

Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid dijalankan oleh *game engine* dan secara *default* akan mengembalikan Nothing.

iv. Fungsi Seleksi

Pemilihan *command* berdasarkan keberadaan dari *power-ups* dan juga kondisi saat ini yang paling memenuhi untuk suatu *command*. Prioritas antara *command* juga digunakan dalam fungsi seleksi ini dengan cara menyeleksi *weighted command* yang paling optimum.

v. Fungsi Kelayakan

Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi *command* yang dipilih sudah terpenuhi. Validasi

dilakukan terhadap validitas dari keberadaan *power-ups*, posisi *bot*, posisi musuh, *speed bot*, *speed* musuh, dan kondisi *terrain*.

vi. Fungsi Objektif

Memaksimalkan *damage* dan pengurangan *speed* terhadap bot lawan ketika melakukan penyerangan. Dalam hal ini memanfaatkan penempatan Cyber Truck(Tweet), Oil, dan penggunaan EMP untuk sebisa mungkin memperlambat pergerakan dari bot lawan.

b. Analisis Efisiensi Solusi

Dasar dari *destructive greedy* adalah melakukan pengecekan terhadap banyaknya *power-ups* yang dimiliki oleh *bot*, misalkan banyaknya *power-ups* yang dimiliki adalah n . Pada algoritma strategi *greedy* ini dilakukan tiga kali pengecekan terhadap masing-masing *command*, yaitu Tweet, Oil, dan EMP. Sejauh ini kompleksitas waktu yang dibutuhkan adalah tiga kali n sehingga menjadi $3n$. Lalu, untuk setiap pengecekan *power-ups* terdapat kondisional untuk menentukan apakah *power-ups* tersebut optimal untuk digunakan. Anggap untuk tiap *command* dilakukan pengecekan kondisional sebanyak empat kali, sehingga total kompleksitas waktunya sekarang adalah $3n + 16$. Terakhir dilakukan pengambilan terhadap *weighted command* dalam *array*, yang berarti kompleksitas waktunya adalah banyaknya elemen pada *array*, dalam hal ini bisa dianggap *size* dari *array* adalah 6 (berdasarkan *weighted command* yang tersedia). Jadi, total kompleksitas waktunya adalah $3n + 22$, yang berarti $T(n) = 3n + 22 = O(n)$, dengan n banyaknya *power-ups* yang dimiliki.

c. Analisis Efektivitas Solusi

Strategi *destructive greedy* ini efektif apabila:

- *Bot* lawan berada dekat di belakang dari *bot* kita, sehingga dimungkinkan penyerangan Oil agar *speed* dari *bot* lawan berkurang.
- *Bot* lawan berada di depan dari *bot* kita, sehingga dimungkinkan penyerangan EMP agar *bot* lawan terpaksa diam selama satu *round* dan *speed* lawan berkurang secara signifikan.

- *Bot* lawan berada di belakang dari *bot* kita dan *lane* lurus, kiri, ataupun kanan dari *bot* lawan memiliki banyak halangan, sehingga dimungkinkan penyerangan Tweet agar *lane* manapun yang dipilih lawan akan menjadi buruk dan lawan terpaksa mengorbankan *speed* yang dimilikinya.

Sedangkan, strategi *destructive greedy* ini kurang efektif apabila:

- *Bot* kita sedang berada tepat di belakang *bot* lawan dan berada pada *lane* yang sama, sehingga penyerangan apapun yang dilakukan akan mengakibatkan dampak negatif pada *bot* kita juga.

1.3. *Opportunist Greedy*

Opportunist Greedy adalah strategi yang bertujuan meningkatkan performa *bot* dan didasarkan pada *power-ups* yang bersifat menguntungkan bagi bot seperti Boost yang dapat dan Lizard. Secara umum, strategi ini diimplementasikan hanya dengan mempertimbangkan *lane* yang berisi *power-up* Boost atau Lizard serta pemilihan waktu yang tepat untuk menggunakan *power-ups* tersebut sehingga strategi ini tidak menghiraukan aspek lainnya seperti *power-ups* lain yang bersifat destruktif. Pemilihan langkah juga didasari oleh jumlah *power-ups* yang terdapat pada setiap *lane* dan langkah mana yang memaksimalkan *power-ups* yang dikumpulkan.

a. Mapping Element Greedy

i. Himpunan Kandidat

Command yang berkaitan dengan penggunaan dan pengumpulan *power-ups* yang bersifat menguntungkan, yaitu USE_BOOST, USE_LIZARD, ACCELERATE, DECELERATE, NOTHING, TURN_LEFT, dan TURN_RIGHT.

ii. Himpunan Solusi

Command yang dipilih sebagai command yang paling optimum dibanding *command* yang lainnya.

iii. Fungsi Solusi

Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid dijalankan oleh *game engine* dan secara *default* akan mengembalikan Nothing.

iv. Fungsi Seleksi

Pemilihan *command* berdasarkan keberadaan dari *power-ups* pada ketiga lane di depan *player* dan juga jumlah *power-ups* yang dimiliki oleh *player*. Prioritas antara *command* juga digunakan dalam fungsi seleksi ini dengan cara menyeleksi *weighted command* yang paling optimum.

v. Fungsi Kelayakan

Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi *command* yang dipilih sudah terpenuhi. Validasi dilakukan terhadap validitas dari keberadaan *power-ups*, posisi *bot*, *speed bot*, dan kondisi *terrain*.

vi. Fungsi Objektif

Memaksimalkan *maxSpeed player* dan pengumpulan *power-ups* sebanyak-banyaknya. Dalam hal ini memanfaatkan penggunaan *boost* dan *lizard* untuk se bisa mungkin menjaga *speed player* semaksimal mungkin.

b. Analisis Efisiensi Solusi

Algoritma *opportunist greedy* melakukan pengecekan terhadap jumlah dan jenis *power-ups* yang dimiliki oleh *player bot*, misalkan jumlah *power-ups* yang dimiliki adalah n . Karena akan dicari *boost* dan *lizard*, dilakukan sebanyak $2n$ pengecekan. Selain itu, dilakukan penghitungan jumlah *boost* pada ketiga *lane* di depan player dengan jangkauan blok sebesar *speed player*. Anggap pada tiap *lane* dilakukan m pengecekan, maka untuk ketiga *lane* akan dilakukan $3m$ pengecekan. Kemudian, jumlah *boost* yang terdapat pada ketiga *lane* ini akan diurutkan untuk memilih yang terbesar. Dengan

bubble search, pengurutan ini maksimal 3 kali. Dilakukan juga pengecekan kondisi penggunaan *boost* dan *lizard* masing-masing 1 kali. Jadi, total kompleksitas algoritmanya adalah $2n + 3m + 5$, yang berarti $T(n) = 2n + 3m + 5 = O(n)$, dengan n banyaknya *power-ups* yang dimiliki dan m besar *speed player*.

c. Analisis Efektivitas Solusi

Strategi *opportunist greedy* ini efektif apabila:

- *Terrain* cenderung tidak dipenuhi blok yang merugikan.
- Posisi *player* sudah jauh di depan posisi lawan.
- Bot lawan menggunakan approach konservatif secara penuh.

Sedangkan, strategi *opportunist greedy* ini kurang efektif apabila:

- *Terrain* cenderung dipenuhi oleh blok yang merugikan dan posisi lawan dekat di depan player pada *lane* yang sama.
- Bot lawan menggunakan *approach* destruktif sehingga lane dipenuhi blok yang merugikan.

2. Solusi Algoritma Greedy yang Dipilih dan Pertimbangannya

Pada *game* Overdrive ini, terdapat tiga aspek penting yang menurut kelompok kami bisa diimplementasikan strategi *greedy*, yaitu *destructive*, *cautious*, dan *opportunist*. Berdasarkan tiga aspek tersebut, dilakukan percobaan yaitu dengan dilakukan pertandingan antara ketiga aspek itu, tetapi hasil dari pertandingan tersebut memberikan hasil yang tidak memuaskan karena rasanya ketiga aspek tersebut merupakan aspek dasar dari *game* Overdrive. Sehingga, tidak bisa hanya dipilih yang terbaik antara salah satu dari ketiga aspek tersebut. Berdasarkan realisasi tersebut, kelompok kami memutuskan bahwa pilihan terbaik adalah untuk menggabungkan ketiga aspek tersebut sehingga menghasilkan *bot* Overdrive yang cukup *all-rounder* yaitu *bot* yang dapat mengambil banyak *power-ups* dan menggunakan untuk mempercepat *bot* kami, menghindari segala macam *obstacle*, dan melakukan penyerangan pada *bot* lawan.

Namun, untuk menggabungkan ketiga aspek tersebut tentunya membutuhkan kompromi dari setiap aspeknya. Setiap aspek, pasti harus menyesuaikan prioritasnya sehingga

menghasilkan strategi *greedy* secara umum yang seimbang. Yang menjadi prioritas dari strategi ini adalah *cautious*. Alasan pemilihan aspek *cautious* yang menjadi *pivot* pada *bot* yang kami buat adalah karena dengan pemilihan *lane* yang baik tentunya akan memberikan peluang terhadap *bot* kami sendiri untuk menggunakan *power-ups* dalam hal ini adalah, *opportunist* dan *destructive*. Aspek *cautious* berperan sebagai penyeimbang antara dua aspek lainnya. Sementara, antara *opportunist* dan *destructive* cenderung lebih diprioritaskan *opportunist* dikarenakan kemampuan yang sinkron dengan *cautious* yaitu penggunaan Lizard dan Boost pada *lane* yang memungkinkan. Setelah *bot* dapat menentukan *lane* yang terbaik dan memberikan peluang untuk melakukan optimisasi (dalam hal ini *opportunist*), aspek *destructive* muncul ketika *bot* sudah berada pada *lane* yang optimum dan *bot* berada pada posisi aman. Aspek *destructive* memberikan kemampuan pada *bot* untuk dapat mengejar lawan yang sudah ada di depan ataupun makin memperlambat ketertinggalan lawan yang berada di belakang.

Gabungan dari ketiga aspek ini (*cautious*, *opportunist*, dan *destructive*) beserta hasil kompromi ketiganya menjadikan strategi *greedy* yang kelompok kami terapkan pada *bot* Overdrive ini memenuhi ekspektasi. Kemampuan *bot* kami menjadi *all-rounder* sesuai dengan perkiraan awal kami saat merumuskan strategi yang ingin digunakan. Tentunya, menjadi *all-rounder* tidaklah menjadikan strategi ini yang terbaik, mungkin terdapat alternatif lain yang dapat menghasilkan hasil yang lebih baik.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

1. Pseudocode

```
// Main Strategi
function run( GameState ) -> Command

Deklarasi

myCar : Car
check : int[]
offCommand : Command
compare, viscompare : double[]
bestRoute, visBestRoute : double
damage, curLane, curBlock : int
blocks, rightblocks, leftblocks : List<Object>
curSpeed, lowerSpeed, higherSpeed : int
frontWeight, leftWeight, rightWeight : double
frontPowerUps, leftPowerUps, rightPowerUps : int
visfrontWeight, visleftWeight, visrightWeight : double
visfrontdamage, visleftdamage, visrightdamage : int
curdamage, leftdamage, rightdamage, checkdamage : int
boostdamage, lessdamage, acclrtdamage, dclrtdamage : int

Body

// menginisialisasi data dari bot
myCar <- gameState.player
damage <- myCar.damage
curLane <- myCar.position.lane
curBlock <- myCar.position.block
curSpeed <- myCar.speed

// menginisialisasi kemungkinan speed serta lane dari bot
lowerSpeed <- getNearbySpeed(curSpeed, -1)
higherSpeed <- getNearbySpeed(curSpeed, 1)
blocks <- getBlocksInFront(curLane, curBlock, 1, gameState)
leftblocks <- getBlocksInFront(curLane, curBlock, 2, gameState)
rightblocks <- getBlocksInFront(curLane, curBlock, 0, gameState)
```

```
// menginisialisasi pencarian command yang bersifat menyerang
offCommand <- offensiveSearch(gameState)

// kasus dimana bot sudah dalam jangkauan finish line
// command lain dihiraukan karena diutamakan mencapai finish line
if (curBlock + higherSpeed >= 1499) then
    -> ACCELERATE
endif

// pengecekan terhadap damage car
// diusahakan agar damage tidak mencapai 2 agar maximum speed tetap di angka 9
if (damage >= 2) then
    -> FIX
endif

// pengecekan terhadap speed
// diutamakan speed untuk diatas 0 agar bot bergerak
if (curSpeed == 0) then
    -> ACCELERATE
endif

// menginisialisasi powerups dari setiap lane
frontPowerUps <- countPowerUps(blocks, curSpeed + 1)
visfrontPowerUps <- countPowerUps(blocks, visibility)
if not(leftblocks.isEmpty()) then
    leftPowerUps <- countPowerUps(leftblocks, curSpeed)
    visleftPowerUps <- countPowerUps(leftblocks, visibility)
else
    leftPowerUps <- -1
    visleftPowerUps <- -1
endif

if not(rightblocks.isEmpty()) then
    rightPowerUps <- countPowerUps(rightblocks, curSpeed)
    visrightPowerUps <- countPowerUps(rightblocks, visibility)
else
    rightPowerUps <- -1
    visrightPowerUps <- -1
endif
```

```
// menginisialisasi damage dari setiap lane
curdamage <- countTotalDamage(blocks, curSpeed + 1)
visfrontdamage <- countTotalDamage(blocks, visibility)
if (leftblocks.isEmpty()) then
    leftdamage <- nullBlocks
    visleftdamage <- nullBlocks
else
    leftdamage <- countTotalDamage(leftblocks, curSpeed)
    visleftdamage <- countTotalDamage(leftblocks, visibility)
endif

if (rightblocks.isEmpty()) then
    rightdamage <- nullBlocks
    visrightdamage <- nullBlocks
else
    rightdamage <- countTotalDamage(rightblocks, curSpeed)
    visrightdamage <- countTotalDamage(rightblocks, visibility)
endif

// menginisialisasi weight dan visweight powerup per damage dari setiap lane
if (curdamage = 0) then
    frontWeight <- frontPowerUps / 0.1
    visfrontWeight <- visfrontPowerUps / 0.1
else
    frontWeight <- frontPowerUps / curdamage
    visfrontWeight <- visfrontPowerUps / visfrontdamage
endif
if (leftdamage = 0 or leftdamage = nullBlocks) then
    leftWeight <- leftPowerUps / 0.1
    visleftWeight <- visleftPowerUps / 0.1
else
    leftWeight <- leftPowerUps / leftdamage
    visleftWeight <- visleftPowerUps / visleftdamage
endif

if (rightdamage = 0 or rightdamage = nullBlocks) then
    rightWeight <- rightPowerUps / 0.1
    visrightWeight <- visrightPowerUps / 0.1
else
    rightWeight <- rightPowerUps / rightdamage
```

```
    visrightWeight <- visrightPowerUps / visrightdamage
endif

// membandingkan total damage dan memilih lane dengan total damage terkecil
check <- [curdamage, leftdamage, rightdamage]
check.sort()
lessdamage <- check[0]

// membandingkan weight dan memilih weight terbesar
compare <- [frontWeight, leftWeight, rightWeight]
compare.sort()
bestRoute <- compare[2]

// membandingkan visibility weight dan memilih weight terbesar
viscompare <- [visfrontWeight, visleftWeight, visrightWeight]
viscompare.sort()
visBestRoute <- viscompare[2]

// mengecek kondisi apabila memilih command boost
boostdamage <- countTotalDamage(blocks, 16)
if ((hasPowerUp(PowerUps.BOOST, myCar.powerups)) and (boostdamage <= 2) and (curSpeed != maxBoostSpeed)) then
    -> BOOST
endif

if ((curdamage = 0) and (curSpeed > 3)) then
    if (offCommand != NOTHING) then
        -> offCommand
    endif
endif

if (((curSpeed = maxBoostSpeed) and (curdamage != 0)) or ((curSpeed >= 8) and (curdamage != 0) and (leftdamage != 0) and (rightdamage != 0))) then
    if (hasPowerUp(PowerUps.LIZARD, myCar.powerups))
        -> LIZARD
    endif
endif

// memilih lane berdasarkan weight dan visibility weight
if ((bestRoute = frontWeight) and (bestRoute = leftWeight) and (bestRoute =
```

```
rightWeight))  
    if (visBestRoute = visfrontWeight) then  
        bestRoute <- frontWeight  
    else  
        if (visBestRoute = visleftWeight) then  
            bestRoute <- leftWeight  
        else  
            if (visBestRoute = visrightWeight) then  
                bestRoute <- rightWeight  
            endif  
        endif  
    endif  
else  
    if ((bestRoute = frontWeight) and (bestRoute = leftWeight)) then  
        if (visBestRoute = visfrontWeight) then  
            bestRoute <- frontWeight  
        else  
            if (visBestRoute = visleftWeight) then  
                bestRoute <- leftWeight  
            endif  
        endif  
    endif  
else  
    if ((bestRoute = leftWeight) and (bestRoute = rightWeight)) then  
        if (visBestRoute = visleftWeight) then  
            bestRoute <- leftWeight  
        else  
            if (visBestRoute = visrightWeight) then  
                bestRoute <- rightWeight  
            endif  
        endif  
    endif  
else  
    if ((bestRoute = frontWeight) and (bestRoute = rightWeight)) then  
        if (visBestRoute = visfrontWeight) then  
            bestRoute <- frontWeight  
        else  
            if (visBestRoute = visrightWeight) then  
                bestRoute <- rightWeight  
            endif  
        endif  
    endif  
endif
```

```
        endif
    endif
endif

if (((bestRoute = frontWeight) and (curdamage <= 3)) or (lessdamage = curdamage)) then
    if (higherSpeed != -1) then
        acclrtdamage <- countTotalDamage(blocks, higherSpeed + 1)
        if ((acclrtdamage <= 3) and (curSpeed < 9)) then
            -> ACCELERATE
        endif
    endif
    -> offCommand
else
    if (bestRoute = leftWeight) then
        checkdamage <- leftdamage
    else
        checkdamage <- rightdamage
    endif

    if ((lowerSpeed != -1) and (lowerSpeed > 5) and ((checkdamage >= 2) or
(lessdamage >= 2))) then
        dclrtdamage <- countTotalDamage(blocks, lowerSpeed + 1)
        if (dclrtdamage = 0) then
            -> DECELERATE
        endif
    endif

    if ((not leftblocks.isEmpty()) and (((bestRoute = leftWeight) and (leftdamage <=
3)) or (lessdamage = leftdamage)))
        -> TURN_LEFT
    else
        if ((not rightblocks.isEmpty()) and (((bestRoute = rightWeight) and
(rightdamage <= 3)) or (lessdamage = rightdamage)))
            -> TURN_RIGHT
        else
            -> offCommand
        endif
    endif
endif
```

```
// mengecek apakah power up ada
function hasPowerUp(PowerUps powerUpToCheck, PowerUps[] available) -> boolean

Deklarasi

Body

for (PowerUps powerUp : available) do
    if (powerUp.equals(powerUpToCheck)) then
        -> true
    endif
-> false
endfor

// menghitung jumlah power up
function countPowerUp(PowerUps powerUpToCheck, PowerUps[] available) -> int

Deklarasi

count : int

Body

count <- 0
for (PowerUps powerUp : available) do
    if (powerUp.equals(powerUpToCheck)) then
        count -> count + 1
    endif
    -> count
endfor

// mengembalikan map of blocks dan objek-objek pada lane tertentu, mengembalikan jumlah
blok yang bisa dijalankan pada max speed

function getBlocksInFront(int lane, int block, int dir, GameState gameState) ->
List<Object>

Deklarasi

map      : List<Lane[]>
blocks   : List<Object>
startBlock : int
laneList : Lane[]
i        : int

Body

map <- gameState.lanes
```

```
if (((dir = 2) and (lane != 1)) or (dir = 1) or ((dir = 0) and (lane != 4))) then
    startBlock <- map.get(0)[0].position.block
    laneList <- map.get(lane - dir)
    i traversal[max(block-startBlock, 0)..block-startBlock+Bot.visibility] do
        if ((laneList[i] = null) or (laneList[i].terrain = Terrain.FINISH)) then
            break
        endif
        if (laneList[i].cybertruck) then
            blocks.add("CYBERTRUCK")
        else
            blocks.add(laneList[i].terrain)
        endif
    endif
-> blocks

// mendapatkan speed state terdekat
function getNearbySpeed(int curSpeed, int dir) -> int
Deklarasi
curIdx : int
Body
if (((curSpeed = 0) and (dir = -1)) or ((curSpeed = 15) and (dir = 1))) then
    -> -1
else
    curIdx <- speedState.indexOf(curSpeed)
    -> speedState.get(curIdx + dir)
endif

// menghitung total damage pada tiap lanes
function countTotalDamage(List<Object> blocks, int steps) -> int
Deklarasi
count : int
Body
count <- 0
if (blocks.size() < steps) then
    steps <- blocks.size()
endif
i traversal[0..steps] do
    if ((block.get(i) = Terrain.MUD) or (blocks.get(i) = Terrain.OIL_SPILL)) then
        count <- count + 1
    else
```

```
        if ((blocks.get(i) = Terrain.WALL) or (blocks.get(i) = "CYBERTRUCK")) then
            count <- count + 2
        endif
    endif
-> count

function countPowerUps (List<Object> blocks, int speed) -> int
Deklarasi
count      : int
i          : int

Body
count <- 0
if (blocks.size() < speed) then
    speed <- blocks.size()
endif
i <- 0
itraversal[0..speed]
    if (blocks.get(i) = BOOST or blocks.get(i) = LIZARD or blocks.get(i) = EMP or
blocks.get(i) = TWEET or blocks.get(i) = OIL_POWER) then
        count += 1
    endif
-> count + 1
```

```
// Bagian Destruktif
// Strategi Destructive
function offensiveSearch (GameState gameState) -> Command

Deklarasi
myCar       : Car
opponent    : Car
actions     : list of tuples of <int, Command>
curLane     : Lane
curBlock    : Block
curSpeed    : Speed
blocks      : list of Terrain
leftBlocks  : list of Terrain
```

```
rightBlocks    : list of Terrain
curDamage      : int
leftDamage     : int
rightDamage    : int
check          : list of int
oppNextSpeed   : int

Body
// Lakukan inisialisasi terhadap deklarasi
// deklarasi untuk melihat state dari bot sendiri
myCar <- gameState.player
// deklarasi untuk melihat state dari lawan sehingga bisa melakukan prediksi
opponent <- gameState.opponent

// logika terhadap power ups Oil
if (hasPowerUp(OIL, myCar.powerups) then
    // jika terlalu banyak oil, drop aja
    if (countPowerUp(OIL, myCar.powerups) > 3) then
        // weight dari command ini 10, yang berarti prioritas paling kecil
        actions.add((10, OIL))
    endif

    // jika ada musuh di belakang
    if (opponent.position.lane = myCar.position.lane) then
        // posisi musuh tepat di belakang kita
        if (opponent.position.block = myCar.position.block - 1) then
            // salah satu weight yang tinggi
            // artinya ini merupakan command yang efektif
            actions.add((1, OIL))

        // posisi musuh tidak tepat di belakang namun cukup dekat
        else if (1 <= (myCar.position.block - opponent.position.block) <= 15)
    then
        actions.add(3, OIL)

    endif
endif
endif
```

```
// logika terhadap power ups cybertruck
if (hasPowerUp(TWEET, myCar.powerUps) then

    // jika kita berada di depan, maka bisa melakukan prediksi
    // terlalu berisiko untuk menggunakan tweet ketika sedang ketinggalan
    if (myCar.position.block > opponent.position.lane) then

        // deklarasi lane, block, dan speed bot musuh
        curLane <- opponent.position.lane
        curBlock <- opponent.position.block
        curSpeed <- opponent.speed

        // deklarasi lane depan, kiri, dan kanan dari bot musuh
        blocks <- getBlocksInFront(curLane, curBlock, 1, gameState)
        leftBlocks <- getBlocksInFront(curLane, curBlock, 2, gameState)
        rightBlocks <- getBlocksInFront(curLane, curBlock, 1, gameState)

        // menghitung damage yang diberikan setiap lane terhadap bot musuh
        // untuk memprediksi lane yang paling tepat untuk diberikan
        // cybertruck

        curDamage <- countTotalDamage(blocks, curSpeed + 1)
        if (leftBlocks.isEmpty()) then
            // kiri lane dari musuh berarti sudah tidak ada lane
            leftDamage <- 9999;
        else
            leftDamage <- countTotalDamage(rightBlocks, curSpeed)
        endif

        if (rightBlocks.isEmpty()) then
            // kanan lane dari musuh berarti sudah tidak ada lane
            rightDamage <- 9999;
        else
            rightDamage <- countTotalDamage(rightBlocks, curSpeed)
        endif

        // cari lane terbaik dari 3 lane
```

```
check <- (curDamage, leftDamage, rightDamage)
check.sort()
bestRoute <- check[0]

// tempatkan cybertruck di lane terbaik musuh
if (bestRoute = curDamage) then
    // opponent mungkin melakukan akselerasi ataupun boost
    // pada lane lurus
    // perkiraan speed state berikutnya dari opponent
    oppNextSpeed <- getNearbySpeed(opponent.speed, 1)

    // jika oppNextSpeed = -1 opponent sedang dalam state boost
    if (oppNextSpeed != -1) then

        // opponent kemungkinan menggunakan boost
        if (opponent.boostCounter >= 1) then
            //prediksi cybertruck sesuai speed boost
            action.add((4,
                        TWEET(opponent.position.lane,
                              opponent.position.block +
                              maxBoostSpeed + 1))

        else
            // prediksi cybertruck sesuai next speed state
            // state maksimum tanpa boost
            if (oppNextSpeed = 9) then
                action.add((4,
                            TWEET(opponent.position.lane,
                                  opponent.position.block +
                                  opponentNextSpeed + 1))

            else
                action.add((4,
                            TWEET(opponent.position.lane,
                                  opponent.position.block +
                                  opponent.speed + 1))
            endif
        endif
    // opponent sudah dalam state boost
    else
        action.add((4, TWEET(opponent.position.lane,
```

```
        opponent.position.block + opponent.speed + 1))

    endif

    else if(bestRoute = leftDamage) then
        // prediksi pada lane sebelah kiri dari lane musuh sekarang
        action.add((4, TWEET(opponent.position.lane - 1,
            opponent.position.block + opponent.speed))

    else
        // prediksi pada lane sebelah kanan
        action.add((4, TWEET(opponent.position.lane + 1,
            opponent.position.block + opponent.speed))

    endif

    endif

endif

// logika memakai EMP
// gunakan EMP hanya ketika di belakang
if (hasPowerUp(EMP, myCar.powerUps) && myCar.position.block < opponent.position.block)
then

    // bot musuh berada dalam range untuk penggunaan EMP
    if (abs(myCar.position.lane - opponent.position.lane) <= 1) then
        // command dengan prioritas tertinggi
        // karena mampu menghentikan pergerakan lawan saat itu juga
        actions.add((0, EMP))

    endif
endif

// sorting untuk command dengan prioritas terbaik (minimum)
actions.sort()
if (actions.size() > 0) then
    // ambil command dengan prioritas optimum
    -> actions[0][1]
else
    // command nothing jika tidak bisa menyerang
    -> Nothing
endif
```

2. Struktur Data Program Overdrive

Pada permainan Overdrive ini, struktur data yang digunakan berbasis pada *class/kelas*. *Starter pack* yang diberikan dari Entelect sudah cukup lengkap, namun pada bagian *lane* perlu ditambahkan atribut bahwa *lane* ditempati oleh Cybetruck. Struktur data yang terdapat pada program Overdrive terdiri atas tiga *folder* yaitu Command, Entities, dan Enums serta dua *file* utama untuk melakukan strategi yaitu Bot.java dan Main.java.

a. Bagian Command

Kelas-kelas pada bagian ini adalah kelas yang berguna untuk menyimpan aksi gerakan atau kemampuan yang dapat digunakan oleh bot pada program Overdrive.

i. Accelerate

Kelas untuk menghasilkan *new Command* yaitu menaikkan *speed state* ke *state* berikutnya.

ii. Boost

Kelas untuk menghasilkan *new Command* yaitu mengubah *speed state* ke *boost speed state* yaitu *speed maksimum* pada permainan.

iii. ChangeLane

Kelas untuk menghasilkan *new Command* yaitu mengubah *lane* yang sedang ditempati oleh *bot* saat ini. Lane yang tersedia adalah kiri dan kanan dari *lane* saat ini.

iv. Command

Kelas pembentuk Command untuk digunakan pada kelas lain.

v. Decelerate

Kelas untuk menghasilkan *new Command* yaitu mengubah *speed state* ke *speed state* untuk menghindari *obstacle* di depan. Jika digunakan ketika dalam *boost speed state*, maka *boost* hilang.

vi. DoNothing

Kelas untuk menghasilkan *new Command* yaitu tidak melakukan apapun.

vii. Emp

Kelas untuk menghasilkan *new Command* yaitu USE_EMP yang berguna untuk memberhentikan lawan selama satu ronde dan merubah *speed* lawan menjadi 3.

viii. Fix

Kelas untuk menghasilkan *new Command* yaitu FIX yang berguna untuk menghapus *damage* yang diterima oleh *bot*. Fix menggunakan satu ronde untuk dilakukan.

ix. Lizard

Kelas untuk menghasilkan *new Command* yaitu USE_LIZARD yang berguna untuk melewati *terrain* apapun dengan *speed* sekarang.

x. Oil

Kelas untuk menghasilkan *new Command* yaitu USE_OIL yang berguna untuk menambahkan Oil yang berfungsi untuk melambatkan *speed* lawan.

xi. Tweet

Kelas untuk menghasilkan *new Command* yaitu USE_TWEET yang berguna untuk memblokir *lane* lawan dengan Cybertruck. Jika lawan menabrak Cybertruck, maka lawan akan berhenti selama satu ronde dan *speed state* lawan menjadi 1.

b. Bagian Entities

Kelas-kelas pada bagian ini berguna untuk menyimpan data *entity* dari permainan.

i. Car

Kelas untuk menyimpan semua atribut yang diperlukan dari *bot* yaitu *speed*, *damage*, *state*, *powerups*, *boosting*, dan *boostCounter*.

ii. GameState

Kelas untuk menyimpan status permainan yang sedang terjadi yaitu *currentRound*, *maxRounds*, *player*, *opponent*, dan *worldMap*.

iii. Lane

Kelas untuk menyimpan *lane* yang berisi *map* tempat permainan sedang dilakukan.

iv. Position

Kelas untuk menyimpan objek *position* posisi bot dalam bentuk kartesian (x dan y).

c. Bagian Enums

Bagian ini berisi kelas-kelas yang menyimpan data-data seperti *power-ups* dan *terrain* yang terdapat pada permainan.

i. Direction

Kelas untuk menyimpan enumerasi tipe-tipe arah yang terdapat pada permainan yaitu, FORWARD, BACKWARD, LEFT, dan RIGHT.

ii. PowerUps

Kelas untuk menyimpan enumerasi tipe-tipe *power-ups* yang terdapat pada permainan yaitu, BOOST, OIL, TWEET, LIZARD, dan EMP.

iii. State

Kelas untuk menyimpan enumerasi tipe-tipe *state* yang terdapat pada permainan salah satunya adalah ACCELERATING, READY, NOTHING, dan TURNING_RIGHT.

iv. Terrain

Kelas untuk menyimpan enumerasi tipe-tipe *terrain* yang terdapat pada permainan salah satunya adalah MUD_SPILL, FLIMSY_WALL, OIL_SPILL, dan FINISH.

d. Kelas Bot

Kelas ini berisi pengembangan dari implementasi algoritma *greedy* yang kami rancang untuk memenangkan permainan Overdrive, memanfaatkan seluruh kelas-kelas sebelumnya. Semua strategi yang ditulis akan terdapat pada method run() pada kelas Bot.

e. Kelas Main

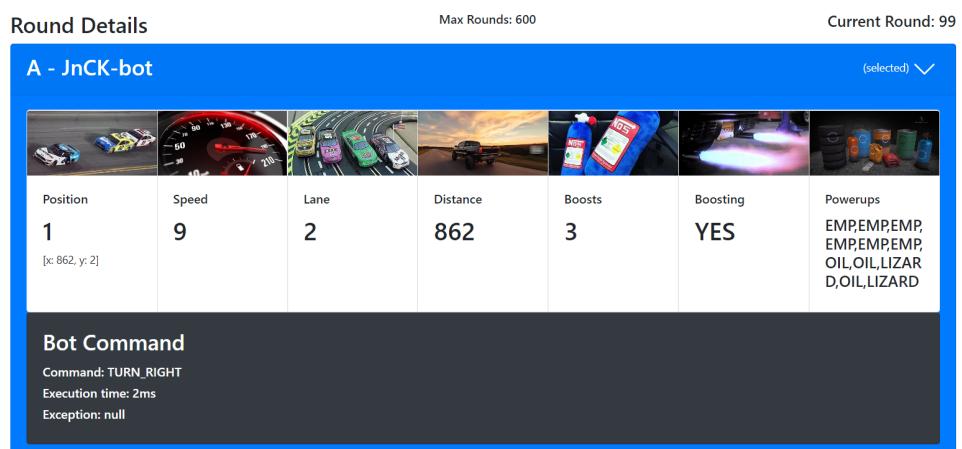
Kelas untuk memanggil kelas *Bot* dan melaksanakan program dengan menjalankan seluruh algoritma dan *command-command* yang sudah dibentuk untuk memulai permainan. Kelas ini akan menerima perintah dari kelas Bot lalu diproses lebih lanjut oleh *game engine* sesuai logika *game* yang telah dibuat.

3. Analisis dan Pengujian

a. Pengujian Greedy Cautious



Gambar 4.1 Map Pengujian Greedy Cautious



Gambar 4.2. State Bot Pengujian Greedy Cautious

Pada pengujian *greedy* secara *cautious*, terlihat dari gambar 4.1 bahwa bot kami berada pada *lane 2*. *Lane 2* tersebut memiliki 3 *terrain* yang mengandung Mud, sehingga jika *bot* kami melewati *lane* tersebut akan mengalami kerugian. Ditinjau dari *lane* sebelah kiri dan kanannya, yaitu 1 dan 3. Untuk *lane 1*, terdapat satu *power-ups* dan 1 *terrain* yang mengandung Mud. Untuk *lane 3*, terdapat 1 *power-ups*, tetapi tidak mengandung *terrain* apapun. Dari ketiga *lane* tersebut, yang paling aman merupakan *lane* yang ketiga (belok kanan dari *lane* saat ini) karena pada *lane 3* bisa mendapat 1 *power-ups* dan tidak terdapat pengurangan *speed* ataupun terkena *damage* yang bisa merugikan *bot* kami. Interaksi *bot* ini merupakan hasil optimal yang sesuai dengan yang kami harapkan.

b. Pengujian Greedy Destructive



Gambar 4.3. Map Pengujian Greedy Destruktif

Round Details		Max Rounds: 600		Current Round: 122															
A - JnCK-bot																			
<table border="1"> <thead> <tr> <th>Position</th> <th>Speed</th> <th>Lane</th> <th>Distance</th> <th>Boosts</th> <th>Boosting</th> <th>Powerups</th> </tr> </thead> <tbody> <tr> <td>1 [x: 1066, y: 3]</td> <td>9</td> <td>3</td> <td>1066</td> <td>5</td> <td>YES</td> <td>EMP,EMP,EMP, EMP,EMP,EMP, OIL,LIZARD,LI ZARD,OIL,OIL, TWEET</td> </tr> </tbody> </table>						Position	Speed	Lane	Distance	Boosts	Boosting	Powerups	1 [x: 1066, y: 3]	9	3	1066	5	YES	EMP,EMP,EMP, EMP,EMP,EMP, OIL,LIZARD,LI ZARD,OIL,OIL, TWEET
Position	Speed	Lane	Distance	Boosts	Boosting	Powerups													
1 [x: 1066, y: 3]	9	3	1066	5	YES	EMP,EMP,EMP, EMP,EMP,EMP, OIL,LIZARD,LI ZARD,OIL,OIL, TWEET													
Bot Command Command: USE_TWEET 4 407 Execution time: 2ms Exception: null																			

Gambar 4.4. State Bot Pengujian Greedy Destruktif

Pada pengujian *greedy* secara *destructive*, terlihat pada gambar 4.3 bahwa *lane* yang ditempat saat ini merupakan *lane* yang aman untuk melakukan gerakan ofensif sehingga pada gambar 4.4. *command* yang dilakukan adalah USE_TWEET, yaitu *command* untuk menempatkan Cybertruck ke suatu posisi.



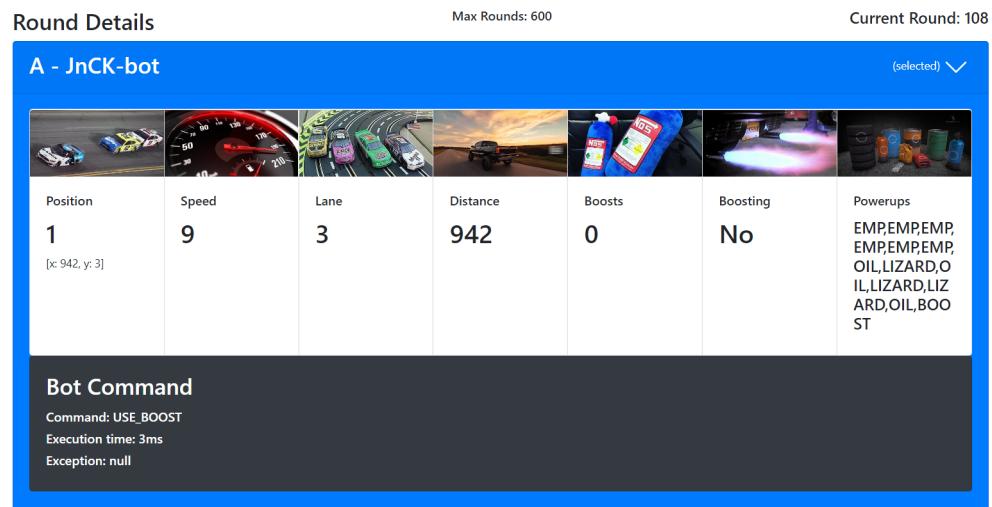
Gambar 4.5 Map Bot Lawan Pengujian Greedy Destruktif

Dari gambar 4.4 terlihat bahwa *command* USE_TWEET ditempatkan pada *lane* 4 dan *block* 407, hal ini sudah didasari dengan melakukan analisis terhadap kondisi *bot* lawan sehingga Cybertruck ditempatkan pada *lane* dan *block* tersebut dengan tujuan untuk menyulitkan lawan. Interaksi *bot* ini merupakan hasil optimal yang sesuai dengan yang kami harapkan.

c. Pengujian Greedy Opportunist



Gambar 4.6 Map Bot Penggunaan Boost Greedy Opportunist

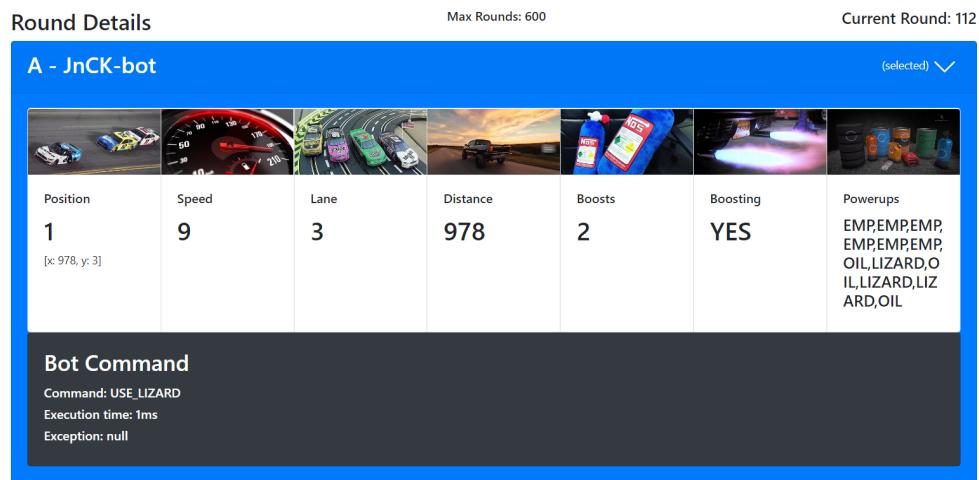


Gambar 4.7 State Bot Penggunaan Boost Greedy Opportunist

Pada pengujian algoritma *greedy opportunist* ini, dapat dilihat pada gambar 4.7 bahwa bot sedang berada di *lane* yang bebas *damage* bahkan hingga batas *visibility* dan memiliki *powerups boost*. Hal ini menyebabkan bot akan menggunakan *command* USE_BOOST. Keputusan ini merupakan keputusan terbaik pada kondisi tersebut karena bisa meningkatkan *speed* menjadi 15 dan semakin menjauhkan diri dari lawan. Interaksi bot ini sesuai dengan apa yang kami harapkan mengenai algoritma ini.



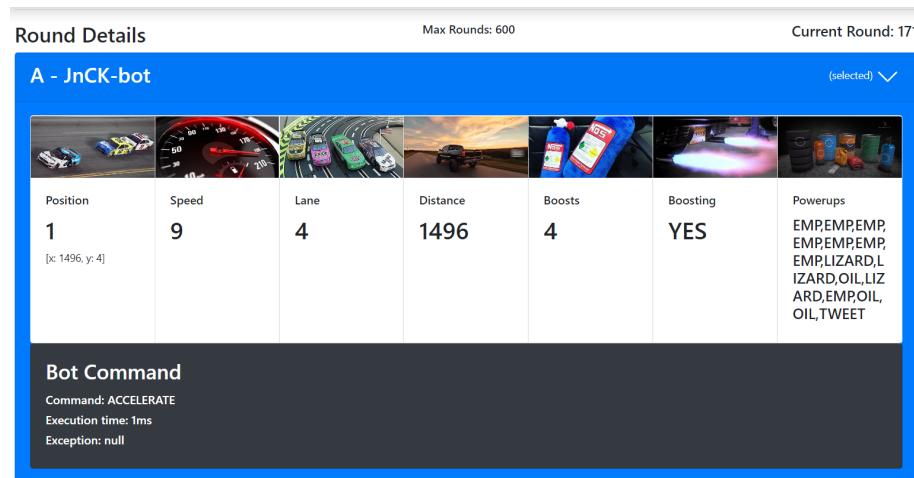
Gambar 4.8 Map Bot Penggunaan Lizard Greedy Opportunist



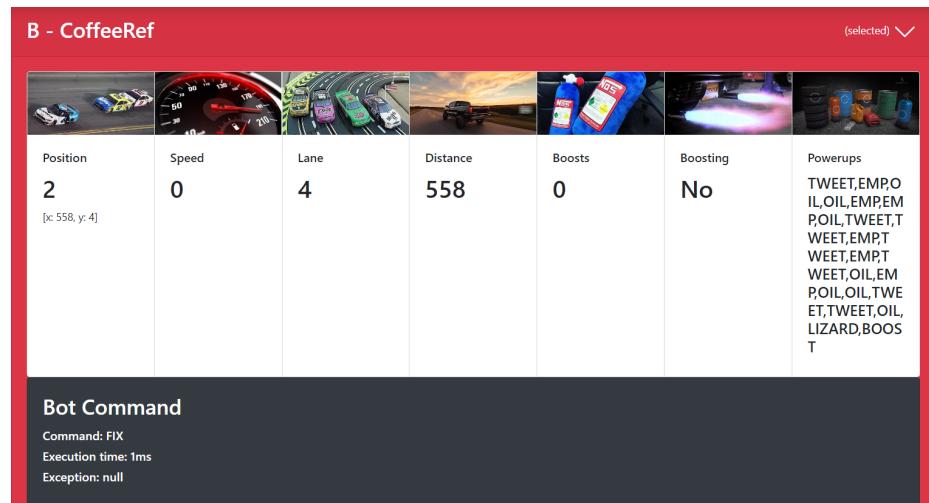
Gambar 4.8 State Bot Penggunaan Lizard Greedy Opportunist

Pada pengujian algoritma *greedy opportunist* ini, dapat dilihat pada gambar 4.8 bot sedang berada pada posisi dimana *speed*-nya merupakan speed maksimal tanpa *boost* dan ketiga *lane* di sekitarnya menyebabkan *damage* yang kurang lebih sama. Hal ini menyebabkan bot akan menggunakan *command* USE_LIZARD dimana penggunaan *lizard* akan menyebabkan bot melompati *obstacle* di depannya. Keputusan ini merupakan keputusan yang sesuai pada kondisi ini karena dengan menggunakan lizard bot bisa menjaga *speed*-nya yang cukup tinggi dan menghindari *damage* yang ada. Interaksi bot ini sesuai dengan apa yang kami harapkan dengan algoritma ini.

d. Pengujian Greedy Keseluruhan



Gambar 4.9 State Final Bot



Gambar 4.10 State Final Bot Lawan

Pada pengujian algoritma *greedy* yang menggabungkan ketiga aspek, dapat terlihat pada gambar 4.9 sekaligus gambar 4.10 bahwa bot yang kami buat (JnCK-bot) mampu mengalahkan *reference bot* dari Entelect dengan selisih yang cukup jauh yaitu sekitar 1000. Dari setiap ronde juga, diamati bahwa *command* yang dihasilkan dari bot kami semuanya sudah hampir sesuai dengan yang kami harapkan. Sehingga, pada akhirnya hasil yang kelompok kami harapkan yaitu untuk menang dapat terealisasikan.

BAB V

KESIMPULAN DAN SARAN

I. Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2021/2022 ini, kami berhasil menciptakan sebuah bot dalam permainan “Overdrive” dengan memanfaatkan algoritma greedy. Untuk *source code* dari *bot* kami dapat dilihat pada [laman berikut](#). Untuk video pengenalan bot dapat dilihat pada [laman berikut](#). Dalam pembuatan bot ini, kami tidak hanya mengimplementasikan satu strategi greedy, melainkan menggabungkan ketiga strategi greedy yang ada. Dengan menggabungkan ketiga strategi tersebut dan menentukan pada situasi apa bot harus mengutamakan approach apa, diperoleh algoritma yang lebih sangkil dan optimum.

II. Saran

Saran-saran yang dapat kami berikan untuk Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun Ajaran 2021/2022 :

- a. Strategi algoritma yang digunakan pada bot ini merupakan strategi *greedy* sebagai pemenuhan kriteria tugas besar. Pada dasarnya algoritma *greedy* belum tentu menghasilkan hasil yang terbaik, sehingga untuk *improvement* bisa dilakukan percobaan terhadap jenis-jenis algoritma lainnya.
- b. Jenis *greedy* yang kelompok kami gunakan juga mungkin belum menjadi yang paling optimum, sehingga bisa dilakukan eksplorasi terhadap alternatif-alternatif *greedy* yang lainnya.
- c. Bot kami merupakan bot *all-rounder* yang tidak menggunakan strategi khusus pada skenario tertentu, sehingga mungkin bisa dilakukan berbagai macam strategi *cheesy* yang memungkinkan.

DAFTAR PUSTAKA

Github.com. (2020). Entelect Challenge 2020 - Overdrive. Diakses pada 5 Februari 2022, dari
<https://github.com/EntelectChallenge/2020-Overdrive>

Github.com. (2020). Game Rules. Diakses pada 3 Februari 2022, dari
<https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md>

Informatika.stei.itb.ac.id/~rinaldi.munir. (2022). Tugas Besar 1 (Tubes 1): Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive". Diakses pada 5 Februari 2022, dari
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Tugas-Besar-1-IF2211-Strategi-Algoritma-2022.pdf>

Informatika.stei.itb.ac.id/~rinaldi.munir. (2021). Algoritma Greedy Bagian 1. Diakses pada 8 Februari 2022, dari
[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)

Informatika.stei.itb.ac.id/~rinaldi.munir. (2021). Algoritma Greedy Bagian 2. Diakses pada 8 Februari 2022, dari
[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)

Informatika.stei.itb.ac.id/~rinaldi.munir. (2021). Algoritma Greedy Bagian 3. Diakses pada 8 Februari 2022, dari
[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)