

# **Tugas Besar 1 IF2211 Strategi Algoritma**

**Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan**

**“Overdrive”**



**Disusun oleh:**

**Kelompok fastabiqul-racing**

**13520016 - Gagas Praharsa Bahar**

**13520029 - Muhammad Garebaldhie ER Rahman**

**13520159 - Atabik Muhammad Azfa Shofi**

**INSTITUT TEKNOLOGI BANDUNG**

**BANDUNG**

**2022**

## DAFTAR ISI

<b>BAB 1 - DESKRIPSI MASALAH</b>	<b>4</b>
<b>BAB 2 - TEORI SINGKAT</b>	<b>7</b>
Algoritma Greedy	7
Elemen-elemen Algoritma Greedy	7
Game Engine	8
Game Engine Overdrive	8
Struktur Game Engine	8
Alur Mulai Permainan	10
Alur Pengembangan Bot	11
<b>BAB 3 - APLIKASI STRATEGI GREEDY</b>	<b>14</b>
Alternatif Solusi Greedy	14
Greedy By Point	14
Greedy By Damage	15
Greedy By Speed	16
Greedy By Enemy Speed	17
Pendekatan Lainnya	18
Solusi Greedy yang Diterapkan dan Pertimbangannya	19
<b>BAB 4 - IMPLEMENTASI DAN PENGUJIAN</b>	<b>21</b>
Repository Github	21
Implementasi Dalam Pseudocode	21
Struktur Data Program	27
Analisis dan Pengujian	32
<b>BAB 5 - KESIMPULAN DAN SARAN</b>	<b>42</b>
Kesimpulan	42
Saran	42
<b>DAFTAR PUSTAKA</b>	<b>43</b>

## BAB 1

### DESKRIPSI MASALAH

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.



Gambar 1. Ilustrasi permainan *Overdrive*

Overdrive adalah game yang diperlombakan pada Entelect Challenge yang diadakan oleh Entelect. Inti dari perlomba ini adalah membuat sebuah *bot* yang dapat berinteraksi dengan *game engine* yang telah disediakan, dan diadu dengan *bot* lain untuk menentukan siapa yang menjadi pemenangnya. Tujuan dari diadakannya perlomba ini adalah untuk memperkenalkan dunia coding dan AI kepada para pemula.

Pada Tugas Besar 1 IF2211 Strategi Algoritma ini, kami diminta untuk mengimplementasikan bot yang dapat memainkan permainan *Overdrive* dengan pendekatan strategi *greedy* untuk memenangkan permainan. Bot akan di implementasi dalam bahasa Java.

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Overdrive. Beberapa aturan umum adalah sebagai berikut:

1. Peta permainan memiliki bentuk array 2 dimensi yang memiliki 4 jalur lurus. Setiap jalur dibentuk oleh block yang saling berurutan, panjang peta terdiri atas 1500 block. Terdapat 5 tipe block, yaitu Empty, Mud, Oil Spill, Flimsy Wall, dan Finish Line yang masing-masing karakteristik dan efek berbeda. Block dapat memuat powerups yang bisa diambil oleh mobil yang melewati block tersebut.
2. Beberapa powerups yang tersedia adalah:
  - a. Oil item, dapat menumpahkan oli di bawah mobil anda berada.
  - b. Boost, dapat mempercepat kecepatan mobil anda secara drastis.
  - c. Lizard, berguna untuk menghindari lizard yang mengganggu jalan mobil anda.
  - d. Tweet, dapat menjatuhkan truk di block spesifik yang anda inginkan.
  - e. EMP, dapat menembakkan EMP ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 lane yang sama) akan terus berada di lane yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.
3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 block untuk setiap round. Game state akan memberikan jarak pandang hingga 20 block di depan dan 5 block di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.
4. Terdapat command yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan power up. Pada setiap round, masing-masing pemain dapat memberikan satu buah command untuk mobil mereka. Berikut jenis-jenis command yang ada pada permainan:
  - a. NOTHING
  - b. ACCELERATE
  - c. DECELERATE
  - d. TURN\_LEFT

- e. TURN\_RIGHT
  - f. USE\_BOOST
  - g. USE\_OIL
  - h. USE\_LIZARD
  - i. USE\_TWEET <lane> <block>
  - j. USE\_EMP
  - k. FIX
5. Command dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor
  6. Bot pemain yang pertama kali mencapai garis finish akan memenangkan pertandingan. Jika kedua bot mencapai garis finish secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar.

## BAB 2

### TEORI SINGKAT

#### 2.1 Algoritma Greedy

Algoritma greedy adalah sebuah algoritma yang dibuat dengan pendekatan *greedy*. Pendekatan *greedy* identik dengan pengambilan solusi optimal langkah per langkah. Pada setiap langkah atau iterasi, akan dipilih keputusan yang paling optimal berdasarkan batasan. Keputusan tersebut tidak perlu memperhatikan kemungkinan-kemungkinan keputusan selanjutnya yang akan diambil atau yang dapat diambil, dan tidak dapat diubah lagi pada langkah selanjutnya. Ini sejalan dengan prinsip utama algoritma greedy yaitu “take what you can get now!” dan berharap dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global (namun nyatanya tidak selalu).

#### 2.2 Elemen-elemen Algoritma Greedy

Algoritma greedy memiliki beberapa elemen atau istilah yang sering menyertainya, diantaranya:

1. Himpunan kandidat, C: berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, S: berisi kandidat yang sudah dipilih.
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi: memilih kandidat berdasarkan strategi greedy tertentu. Strategi ini bersifat heuristik.
5. Fungsi kelayakan: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen di atas, maka algoritma greedy dapat dinyatakan sebagai:

Algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S di optimisasi oleh fungsi objektif

## 2.3 Game Engine

*Game engine* adalah *framework* perangkat lunak yang didesain untuk pembuatan dan pengembangan *video game* dan biasanya mencakup banyak *library* dan program pendukung lainnya. *Game engine* dibuat dengan tujuan mempermudah proses pengembangan video game dan juga mengurangi biaya pengembangan, karena dengan satu *game engine* dapat dibuat *video game* yang berbeda-beda, dan para pengembang tidak perlu membuat dari nol.

## 2.4 Game Engine Overdrive

Dalam permainan Overdrive yang menjadi subjek Tugas Besar kali ini, terdapat beberapa komponen yang diperlukan agar permainan dapat berjalan:

1. *Game engine*: Game engine disini berlaku sebagai *platform* untuk kedua bot untuk bermain. Game engine akan meneruskan perintah yang dikeluarkan oleh bot ke dalam game untuk diproses, dengan memperhatikan peraturan game yang sudah diatur sebelumnya.
2. *Game runner*: *Game runner* berlaku sebagai perangkat lunak yang menjalankan atau memulai pertandingan antar pemain dengan konfigurasi yang telah ditentukan.
3. *Reference bot*: Reference bot adalah bot yang disediakan oleh pembuat Overdrive sebagai lawan. Tersedia dalam bahasa java.
4. *Starter bot*: Bot awal dengan logika dasar dengan beberapa kode *boilerplate* yang dapat digunakan sebagai referensi awal pengembangan bot. Tersedia dalam bahasa C++, C# (.NET), Golang, Java, JavaScript, Lisp, Python3, dan Rust.

## 2.5 Struktur Game Engine

Untuk mulai membangun bot untuk game Overdrive, diperlukan untuk mengunduh starter-pack yang telah disediakan oleh pembuat permainan Overdrive, EntelectChallenge, pada tautan <https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>. Pada tugas besar ini, akan dibangun sebuah *bot* yang dibuat menggunakan bahasa Java, sehingga yang akan dijelaskan pada dokumen ini hanyalah yang terkait dengan Java.

Starter-pack berisi beberapa file, yaitu:

- Folder reference-bot: Folder ini berisi reference bot dalam bahasa Java.
- Folder starter-bot: Folder ini berisi starter bot dalam berbagai bahasa pemrograman.
- File game-engine.jar: File ini berisi *executable software* Java yang berguna sebagai game engine dari permainan Overdrive.
- File game-runner.jar dan game-runner-with-dependencies.jar: File ini berisi *executable software* java yang berguna sebagai game runner yang memulai sebuah game dari Overdrive.
- File run.bat dan makefile: File ini berguna untuk menjalankan permainan dengan *command* yang lebih simpel.

Pada folder reference-bot dan starter-bot, terdapat *template* untuk pengembangan bot. Untuk bot yang menggunakan bahasa Java, terdapat beberapa file dan folder didalamnya yang berlaku sebagai bot itu sendiri. Pada root folder, terdapat bot.json yang berisi identitas dari bot dan pom.xml sebagai *Project Object Model* yang diperlukan oleh Maven, kakas otomasi *build* untuk Java yang digunakan pada bot ini.

Selain itu, terdapat folder src yang berisi *source code* dari bot. Source code ini lebih spesifiknya terdapat pada folder src/main/java/za/co/entelect/challenge, yang berisi:

- Folder *command*: Folder ini berisikan kelas dari perintah yang dapat digunakan oleh *bot*. Perintah yang disediakan adalah Accelerate, Boost, ChangeLane, Decelerate, DoNothing, EMP, Fix, Lizard, Oil, dan Tweet.
- Folder entities: Folder ini berisikan kelas yang mendefinisikan entitas yang ada pada game, yaitu Car, GameState, Lane, dan Position.
- Folder enums: Folder ini berisikan enum yang menyatakan entitas yang memiliki nilai konstan, yaitu Direction, PowerUps, State, Terrain.
- File Bot.java: File ini berisikan logika yang dimiliki oleh bot yang dibangun. Implementasi algoritma strategi greedy akan berada pada file ini, dengan memanfaatkan kelas-kelas lain yang sudah ada sebelumnya. Semua strategi akan berada pada function run() yang akan dijalankan pada saat permainan dimulai.

- File Main.java: File ini berisikan program utama yang akan menjalankan *bot* dengan menginstansiasi *bot* pada setiap ronde, dan menerima perintah dari bot untuk kemudian dikirimkan kepada game engine untuk dijalankan dan diproses.

## 2.6 Alur Mulai Permainan

Untuk dapat menjalankan permainan dengan bot, dibutuhkan beberapa kakis yang perlu di-*install*, yaitu Java Development Kit 8, NodeJS, Maven, dan IDE (opsional, IntelliJ). Untuk mendapatkan *file executable* Java dengan ekstensi .jar, source code harus di *compile* dan *install* menggunakan Maven, yang telah ada sebagai program bawaan pada IDE IntelliJ dan sebagai extension pada VSCode. Setelah itu, diperlukan beberapa konfigurasi, yaitu:

- game-runner-config.json: File ini terletak pada folder starter-pack. File ini digunakan untuk mengatur lokasi *file executable* bot pemain satu dan pemain dua, *file executable game engine*, directory penyimpanan hasil permainan, set-up mode turnamen, dan lain-lain.
- game-config.json: File ini terletak pada folder starter-pack. File ini digunakan untuk mengatur konstanta-konstanta yang ada pada pertandingan, seperti misalnya TRACK\_LENGTH, NUMBER\_OF\_LANES, dan konstanta lain yang dapat berpengaruh pada jalannya permainan.
- Bot.json: File ini terletak pada folder bot masing-masing dan digunakan sebagai *signature* dari bot, yaitu nama bot, lokasi executable, dan nama beserta email pembuat bot.

Apabila semuanya sudah dikonfigurasi, maka untuk menjalankan permainan cukup dengan menjalankan run.bat yang terletak pada folder utama, atau dengan `make run` apabila memiliki *software* make. Permainan akan berjalan dengan CLI (Command Line Interface) pada terminal.

Untuk membantu visualisasi permainan, dapat digunakan website visualizer yang dibuat oleh Raezor pada tautan (<https://entelect-replay.raezor.co.za/>) . Untuk memvisualisasikan jalannya permainan pada website tersebut, diperlukan log hasil permainan yang tersimpan secara *default* di folder match-logs.

## 2.7 Alur Pengembangan Bot

Untuk mulai mengembangkan bot, ada beberapa hal yang dapat dilakukan dan diperhatikan. Pada starter-bot, sudah terdapat beberapa *boilerplate* dari bot itu sendiri, namun *boilerplate* ini masih sangat kosong dan belum lengkap. Berikut ini adalah *boilerplate* awal yang terdapat pada file Bot.java:

```
package za.co.entelect.challenge;
import za.co.entelect.challenge.command.*;
import za.co.entelect.challenge.entities.*;
import za.co.entelect.challenge.enums.Terrain;
import java.util.*;
import static java.lang.Math.max;
public class Bot {
    private static final int maxSpeed = 9;
    private List<Integer> directionList = new ArrayList<>();
    private Random random;
    private GameState gameState;
    private Car opponent;
    private Car myCar;
    private final static Command FIX = new FixCommand();
    public Bot(Random random, GameState gameState)
        {Konstruktor Bot}
    public Command run()
        {Strategi Greedy akan diletakkan disini}
}
```

Langkah pertama yang dapat dilakukan adalah menginstansiasi semua kelas Command yang ada pada entitas folder command dengan menambahkan atribut dengan template berikut pada bot:

```
private final static Command <NAMA_COMMAND> new <NAMA_KELAS_COMMAND>
```

Dikarenakan pada bot hanya ada command FIX pada saat ini, maka kita dapat menambahkan command-command lain, diantaranya ACCELERATE, TURN\_LEFT, TURN\_RIGHT, DECELERATE, OIL, LIZARD, BOOST, EMP, FIX, dan NOTHING. Untuk

command TWEET, dikarenakan kelas tersebut harus diinstansiasi dengan lane dan block yang ditarget, maka instansiasi dilakukan saat sebelum pemanggilan command.

Selain itu, pada entitas Lane, terdapat satu buah atribut yang belum masuk ke dalam Lane, yaitu isOccupiedByCyberTruck, yang digunakan untuk mengecek apakah di lane-block tertentu terdapat cybertruck atau tidak. Untuk menambahkannya, tambahkan line berikut ini pada class Lane:

```
@SerializedName("isOccupiedByCyberTruck")
public Boolean isOccupiedByCyberTruck;
```

Setelah mempersiapkan semua command yang diperlukan, langkah berikutnya adalah mengimplementasikan strategi greedy pada fungsi run(). Selain itu, dapat diperhatikan juga data-data yang bisa diakses. Data ini dapat diakses melalui atribut gameState. gameState sendiri memiliki format sebagai berikut:

```
{
    "currentRound":1,
    "maxRounds":600,
    "player":{
        "id":1,
        "position":{
            "y":1,
            "x":1
        },
        "speed":5,
        "state":"READY",
        "statesThatOccurredThisRound":[
            "READY"
        ],
        "powerups":[],
        "boosting":false,
        "boostCounter":0,
        "damage":0,
        "score":0
    },
    "opponent":{
        "id":2,
```

```
"position":{  
    "y":4,  
    "x":1  
},  
"speed":5  
,  
"worldMap": [  
    [  
        {  
            "position":{  
                "y":1,  
                "x":1  
            },  
            "surfaceObject":0,  
            "occupiedByPlayerId":1,  
            "isOccupiedByCyberTruck":false  
        },  
        ...  
    ]  
]
```

Dengan mempertimbangkan struktur data diatas, maka bot sudah dapat dikembangkan dengan menambahkan fungsi-fungsi dan juga logika pada Bot.java.

## **BAB 3**

### **APLIKASI STRATEGI GREEDY**

#### **3.1. Alternatif Solusi Greedy**

##### **3.2.1. Greedy By Point**

Greedy by Point adalah pendekatan algoritma greedy yang memprioritaskan penambahan point sebagai pendekatannya. Point didapatkan ketika pemain mendapatkan dan memakai power up, dan berkurang ketika melewati obstacle seperti mud, wall, dan lain-lain. Pada setiap ronde, prioritaskan penambahan point (mengambil power up atau memakai power up) dan hindari pengurangan poin (terkena obstacle)

###### a. Mapping Elemen Greedy

- i. Himpunan kandidat: Semua command yang tersedia.
- ii. Himpunan solusi: Command yang terpilih.
- iii. Fungsi solusi: Memeriksa apakah command yang dipilih memberikan point terbanyak dan dapat dilakukan pada saat itu.
- iv. Fungsi seleksi: Pilih command yang menghasilkan poin terbanyak
- v. Fungsi kelayakan: Memeriksa apakah command yang dipilih valid dan dapat dijalankan
- vi. Fungsi objektif: Maksimumkan point yang didapat pada sebuah ronde

###### b. Analisis Efisiensi Solusi

Terdapat tiga command utama yang harus dicek (apakah stay di lane, turn\_left, atau turn\_right). Pada setiap command, diperlukan pengecekan *block* sejauh *speed* yang sekarang tengah dijalankan untuk pengambilan power up. Apabila memungkinkan, gunakan power up yang dimiliki (boost, oil, emp, tweet, lizard) atau ambil power up baru. Tanpa algoritma penghindaran dari *obstacle*, maka kompleksitasnya adalah

$$T(n) = 3*m*1 + 5*1 = O(m). \text{ (dengan } m = \text{current speed})$$

###### c. Analisis Efektivitas Solusi

Greedy by Point berasal dari adanya kriteria point sebagai salah satu metric kemenangan. Namun, karena urutan prioritas kriteria kemenangan adalah Distance > Speed > Point, solusi ini bukanlah yang paling efektif.

Strategi ini efektif apabila:

- Map yang di-generate oleh game engine lebih banyak power up dibandingkan obstacle.

Strategi ini tidak efektif apabila:

- Terdapat sedikit power up pada map
- Terdapat banyak obstacle yang seringkali tertabrak karena memprioritaskan penambahan poin.

### 3.2.2. Greedy By Damage

Greedy by Damage adalah pendekatan algoritma greedy yang memprioritaskan agar terkena damage sesedikit mungkin. Damage dapat bertambah ketika pemain terkena obstacle, dan dapat berkurang saat pemain melakukan FIX. Apabila memiliki damage, maka maximum speed akan turun, sehingga greedy by damage juga salah satu strategi yang dapat dijalankan untuk memaksimumkan kemenangan.

#### a. Mapping Elemen Greedy

- i. Himpunan kandidat: Semua command yang tersedia.
- ii. Himpunan solusi: Command yang terpilih.
- iii. Fungsi solusi: Memeriksa apakah command yang dilakukan memberikan damage paling sedikit pada mobil dan dapat dilakukan
- iv. Fungsi seleksi: Pilih command dengan damage terkecil.
- v. Fungsi kelayakan: Memeriksa apakah command yang terpilih dapat dijalankan
- vi. Fungsi objektif: Minimumkan damage yang didapat pada sebuah ronde

#### b. Analisis Efisiensi Solusi

Terdapat tiga command utama yang diprioritaskan: ACCELERATE, TURN\_RIGHT, TURN\_LEFT. Bot akan memprioritaskan untuk menghindari *obstacle* dibandingkan

menyerang atau mempercepat musuh. Untuk itu, diperlukan pengecekan *obstacle* pada ketiga lane tergantung dari *speed* saat ini. Kompleksitasnya adalah:

$$T(n) = 3*m*1 = O(m). \text{ (dengan } m = \text{current speed})$$

c. Analisis Efektivitas Solusi

Greedy by Damage berasal dari adanya sistem *damage* pada permainan overdrive, dimana kecepatan maksimum pemain akan terbatas dengan sebanyak apa *damage* yang ada pada mobilnya. Hal ini mengimplikasikan bahwa dengan memprioritaskan damage sekecil mungkin, rata-rata *speed* seharusnya naik dikarenakan naiknya maximum speed.

Strategi ini efektif apabila:

- Algoritma penyerangan lawan (dengan menggunakan power up) termasuk lemah, sehingga damage yang diterima lebih sedikit secara umum.
- Posisi bot sudah berada di depan sejak awal

Strategi ini tidak efektif apabila:

- Algoritma penyerangan lawan baik (seperti penggunaan EMP), sehingga banyak *damage* yang tak terhindarkan dan banyak waktu yang terbuang untuk FIX.
- Posisi bot berada di belakang, karena bot tidak memprioritaskan menyerang musuh, maka akan lebih sulit untuk mengejar ketertinggalan.

### 3.2.3. Greedy By Speed

Greedy by Speed adalah pendekatan algoritma greedy yang memprioritaskan kecepatan. Kecepatan dapat ditambahkan dengan command ACCELERATE dan dikurangi dengan command DECELERATE. Power up BOOST juga dapat digunakan untuk mencapai BOOST\_SPEED (*speed* = 15) untuk 5 ronde kedepan.

a. Mapping Elemen Greedy

- i. Himpunan kandidat: Semua command yang tersedia.
- ii. Himpunan solusi: Command yang terpilih.
- iii. Fungsi solusi: Memeriksa apakah command yang dilakukan memberikan kecepatan paling besar pada ronde berikutnya dan command tersebut dapat dijalankan

- iv. Fungsi seleksi: Pilih command dengan kecepatan paling besar.
  - v. Fungsi kelayakan: Memeriksa apakah command valid.
  - vi. Fungsi objektif: Maksimumkan speed pada tiap ronde.
- b. Analisis Efisiensi Solusi
- Greedy by Speed akan memprioritaskan BOOST dan ACCELERATE, maka kompleksitasnya tanpa penghindaran *obstacle* dan pengambilan power up memiliki kompleksitas sebesar O(1).
- c. Analisis Efektivitas Solusi
- Greedy by Speed berasal dari prioritas speed yang cukup tinggi pada penentuan pemenang (Distance > Speed > Point). Selain itu, dengan memiliki kecepatan rata-rata yang tinggi maka pasti distance yang dicakup juga besar.
- Strategi ini efektif apabila:
- Bot sudah berada di depan musuh sejak awal (memiliki *seed* awal yang baik).
  - Banyak terdapat powerup BOOST dan LIZARD pada map.
- Strategi ini tidak efektif apabila:
- Posisi bot berada di belakang, karena bot tidak memprioritaskan menyerang musuh, maka akan lebih sulit untuk mengejar ketertinggalan.
  - Bot tidak mendapatkan banyak BOOST dan LIZARD, sehingga tidak berada dalam kecepatan maksimum sebanyak yang diinginkan.

#### 3.2.4. Greedy By Enemy Speed

Greedy by Enemy Speed adalah pendekatan algoritma greedy yang memprioritaskan membuat kecepatan musuh sekecil mungkin. Hal ini berarti bot lebih memprioritaskan menyerang musuh dengan power-up yang ada dibandingkan dengan mempercepat kendaraan sendiri. Hal ini dapat membuat musuh dengan lebih mudah disalip.

- a. Mapping Elemen Greedy
- i. Himpunan kandidat: Semua command yang tersedia.
  - ii. Himpunan solusi: Command yang terpilih.

- iii. Fungsi solusi: Memeriksa apakah command yang dilakukan mengurangi kecepatan musuh paling besar.
- iv. Fungsi seleksi: Pilih command dengan pengurangan kecepatan musuh paling besar.
- v. Fungsi kelayakan: Memeriksa apakah command valid.
- vi. Fungsi objektif: Minimumkan speed musuh pada tiap ronde.

b. Analisis Efisiensi Solusi

Greedy by Enemy Speed butuh untuk mengecek posisi dari lawan, dan menyerang musuh apabila ada kesempatan. Kompleksitas algoritma dari Greedy by Enemy Speed adalah  $O(1)$ , karena untuk lookup kecepatan musuh hanya butuh satu pemanggilan.

c. Analisis Efektivitas Solusi

Greedy by Enemy Speed berasal dari prioritas untuk membuat lawan selambat mungkin. Hal ini dapat dicapai dengan menggunakan power up secara strategis. Selain itu, dengan menggunakan power up yang banyak, point yang didapat juga akan banyak, karena menggunakan power up dapat memberikan poin sebesar 4.

Strategi ini efektif apabila:

- Algoritma penghindaran musuh buruk.
- Terdapat banyak power up EMP, TWEET, dan OIL pada map.

Strategi ini tidak efektif apabila:

- Algoritma penghindaran obstacle musuh baik. Apabila musuh dapat menghindari semua serangan, maka bot yang memprioritaskan Enemy Speed akan menjadi kurang efektif.

### 3.3. Pendekatan Lainnya

Pada bagian ini, akan dijelaskan beberapa strategi lain yang menjadi pertimbangan, namun bukan sebuah strategi greedy yang utuh.

- Prioritaskan lane 2 dan 3

Dengan memprioritaskan lane 2 dan 3 lebih dari 1 dan 4, maka bot akan memiliki lebih banyak pilihan untuk manuver (`turn_left` dan `turn_right`) dibandingkan pada lane 1 dan lane 4, karena kedua lane tersebut berada di ujung dan hanya dapat melakukan salah satu manuver saja dari `turn_left` dan `turn_right`.

- Pembobotan setiap obstacle

Pada algoritma penghindaran (banyak dipakai di Greedy by Speed dan Greedy by Damage), dilakukan pembobotan obstacle, dikarenakan ada obstacle yang memiliki efek yang lebih fatal dibanding yang lainnya, misalkan MUD memiliki bobot yang lebih kecil dibandingkan WALL dikarenakan efeknya yang lebih ringan.

- Perubahan strategi *on-the-flight*

Berdasarkan analisis yang telah disampaikan diatas, terkadang ada beberapa solusi yang lebih baik dalam kondisi tertentu. Oleh karena itu, dapat dibuat kondisi secara heuristik strategi yang lebih cenderung kemanakah yang dilakukan pada suatu saat tertentu, sehingga strategi bot dapat berubah secara dinamis.

### 3.4. Solusi Greedy yang Diterapkan dan Pertimbangannya

Seperti yang telah dipaparkan pada poin-poin sebelumnya, terdapat banyak pendekatan *greedy* yang dapat diimplementasikan tergantung pada situasi, beserta kekurangan dan kelebihannya masing-masing. Oleh karena itu, akan lebih baik apabila bot dapat mempertimbangkan masing-masing pendekatan *greedy* dalam penentuan pilihan Command yang akan dijalankan.

Pada implementasi program bot kami, alur berpikir yang kami pakai adalah sebagai berikut:

1. Prioritaskan speed setinggi mungkin.
2. Apabila tidak bisa lebih cepat lagi, prioritaskan mendapatkan poin sebanyak mungkin.
3. Apabila sudah tidak bisa lurus, pindah jalur, dengan prioritas jalur tengah (lane 2 dan 3).
4. Apabila sudah tidak bisa lurus dan belok, gunakan Lizard apabila tersedia.
5. Apabila tidak ada lagi yang dapat dilakukan, ambil jalur yang *obstacle*-nya memiliki bobot paling kecil.

6. Apabila sedang dalam kondisi dibelakang musuh, gunakan EMP untuk dapat melemahkan musuh.

Poin-poin di atas didasari dari alternatif-alternatif solusi greedy yang telah dieksplorasi sebelumnya. Poin 1 mencakup Greedy By Speed, dimana bot akan memprioritaskan speed setinggi mungkin. Poin 2 mencakup Greedy By Point, artinya bot akan memprioritaskan perpindahan jalur yang akan memberikan poin paling tinggi kepada bot. Poin 3, 4, dan 5 mencakup Greedy By Damage, dimana apabila tidak ada aksi lagi yang dilakukan, hindari *obstacle* untuk mendapatkan damage sekecil mungkin. Poin 6 mencakup Greedy By Enemy Speed, apabila bot dalam posisi dibelakang musuh, gunakan EMP untuk dapat menghentikan musuh beberapa saat.

## BAB 4

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Repository Github

Bot yang kami buat dapat diakses melalui repository Github pada tautan berikut:  
<https://github.com/IloveNoooodles/fastabiqul-racing>

#### 4.2 Implementasi Dalam *Pseudocode*

```
function run() → Command

Algoritma

{Handle Last Move}

if (myCarBlock+myCarSpeed ≥ 1500) then

    if (hasBoost and myCar.damage = 0 and blockerForward ≤ 1) then return BOOST

    if (MOVE_FORWARD() ≠ null) then return MOVE_FORWARD()

    if (hasLizard and not(lastBlocked)) then return LIZARD

    if (blockerLeft = 0) then return TURN_LEFT

    if (blockerRight = 0) then return TURN_RIGHT

{Fix worst condition}

if (myCar.damage ≥ 3) then

    if (myCarSpeed < listSpeed[myCar.damage] and canAccelerate) then

        return ACCELERATE

    else return FIX

if (myCarSpeed = 0) then return ACCELERATE
```

```

{Endgame mode : prio speed but still turn right and left}

if (InEndGame) then return ENDGAME()

{Boost while min speed}

if (myCarSpeed ≤ 3 and hasBoost and myCar.damage = 0 and blockerForwardFar = 0) then
    return BOOST

if (myCarSpeed ≤ 5 and canAccelerate) then return ACCELERATE

{ Prio use emp if losing}

if ((hasEMP) and

    ((opponentBlock > myCarBlock + 20) or (opponentSpeed > myCarSpeed)) and
    (opponentBlock > myCarBlock) and (myCarSpeed ≥ 6) and
    (blockerForward = 0) and (Math.abs(opponentLane - myCarLane) ≤ 1))
    then return EMP

{Boost always number 1}

if ((hasBoost and !boosting) and (blockerForwardFar = 0)) then
    if (myCar.damage ≠ 0) then return FIX
    else return BOOST

{Prio take boost}

if (LOOKFOR(BOOST) ≠ null) then return LOOKFOR(BOOST)

{Prio take emp}

if (LOOKFOR(EMP) ≠ null) then return LOOKFOR(EMP)

```

```

{Prio midlane}

if (PRIO_MID_LANE() ≠ null) then return PRIO_MID_LANE()

{Prio speed}

if (canAccelerate) then return ACCELERATE

{While fullspeed and no-blocker then use powerup}

if (USE_POWERUP() ≠ null and blockerForward = 0) then return USE_POWERUP()

{Prio pickup powerup}

if (LOOKFOR(POWERUP) ≠ null) then return LOOKFOR(POWERUP)

{Boost instead Low_speed}

if ((hasBoost) and (myCarSpeed < 8) and (blockerForwardFar ≤ 1)) then
    if (myCar.damage ≠ 0) then return FIX
    else return BOOST

{Do nothing if can forward}

if (blockerForward = 0) then return SKIP

{If cant forward but can turn}

if (blockerForward > 0 and (blockerRight = 0 or blockerLeft = 0)) then return bestLane

{If cant turn but has lizard}

if (hasLizard and !lastBlocked) then return LIZARD

```

```

{Last attempt, prio to take powerup on far side}

if (PRIO_POWERUP_FARLAND() ≠ null) then return PRIO_POWERUP_FARLAND()

{If can Decelerate}

if (myCarSpeed ≥ 6 and canDecelerate) then return DECELERATE

{Boost instead Low_speed}

if ((hasBoost) and (myCar.damage = 0) and not(boosting) and (blockerForwardFar ≤ 1))
then return BOOST

{If cant do anything}

return bestLane

```

### **function MOVE\_FORWARD() → Command**

#### Algoritma

```

if (blockerForward = 0) then

    if (canAccelerate) then return ACCELERATE

    if (USE_POWERUP() ≠ null) then return USE_POWERUP

    else return SKIP

else return null

```

### **function ENDGAME() → Command**

#### Algoritma

```

{EMP if losing in endgame}

if ((hasEMP) and not((myCarLane = opponentLane) and (myCarBlock + myCarSpeed >
opponentBlock)) and (opponentBlock > myCarBlock) and myCarSpeed ≥
listSpeed[myCar.damage + 1] and (Math.abs(opponentLane - myCarLane) ≤ 1) and

```

```

blockerForward = 0) then return EMP

{Boosting condition}

if (boosting and hasLizard and not(lastBlocked)) then return LIZARD

if (myCarBlock ≥ 1485 and hasBoost and myCar.damage = 0 and blockerForwardFar < 2)
then return BOOST

if ((hasBoost) and (not(boosting)) and (blockerForwardFar = 0)) then

    if (myCar.damage ≠ 0) then return FIX

    else return BOOST

if ((hasBoost) and (myCarSpeed < 8) and (blockerForwardFar ≤ 1)) then

    if (myCar.damage ≠ 0) then return FIX

    else return BOOST

{Minimum speed}

if (myCarSpeed ≤ 3 and canAccelerate) then return ACCELERATE

{Prio take boost}

if (LOOKFOR(BOOST) ≠ null) then return LOOKFOR(BOOST)

{Prio speeding}

if (canAccelerate) then return ACCELERATE

{Moving forward}

if (MOVE_FORWARD ≠ null) then return MOVE_FORWARD

{Lizard if cant forward}

```

```
if (blockerForward > 0 and blockerForward = blockerForwardFar and hasLizard and  
not(lastBlocked)) then return LIZARD
```

```
{If lizard not effective}
```

```
else return bestLane
```

```
function LOOKFOR(option) → Command
```

**Algoritma**

```
if (containBoost(blocksFront)) then
```

```
    if (MOVE_FORWARD() ≠ null) then return MOVE_FORWARD()
```

```
    if (blocksRightContain(option) and WorthTurnRight) then return TURN_RIGHT
```

```
    if (blocksLeftContain(option) and WorthTurnLeft) then return TURN_LEFT
```

```
else return null
```

```
function USE_POWERUP() → Command
```

**Algoritma**

```
if ((hasOil) and (Math.abs(opponentLane - myCarLane) ≤ 1) and (opponentBlock ≤  
myCarBlock) and ((opponentLane = myCarLane) or (blockerLeft ≠ 0 and blockerRight ≠  
0))) then return OIL
```

```
if ((hasTweet) and not(myCarLane = opponentLane and myCarBlock ≤ opponentBlock +  
opponentSpeed + 3) and (myCarLane ≠ opponentLane or opponentSpeed ≥ 8)) then
```

```
    if (opponentBlock ≥ 1490) then
```

```
        TWEET = TweetCommand(opponentLane, 1500)
```

```
    else
```

```
        TWEET = TweetCommand(opponentLane, opponentBlock + opponentSpeed + 3)
```

```
    return TWEET
```

```
{If Oil and Tweet doesn't meet condition then use over oil}
```

```
if (countOil > 3) then return OIL
```

```
else return null
```

```
function PRIO_MID_LANE() → Command
```

Algoritma

```
if (myCarLane = 1 and blockerRight ≤ blockerForward) then return TURN_RIGHT  
if (myCarLane = 4 and blockerLeft ≤ blockerForward) then return TURN_LEFT  
if (myCarLane = 1 and blockerRightFar ≤ blockerForwardFar) return TURN_RIGHT  
if (myCarLane = 4 and blockerLeftFar ≤ blockerForwardFar) then return TURN_LEFT  
else return null
```

```
function PRIO_POWERUP_FARLAND() → Command
```

Algoritma

```
if (containBoost(blocksRightFar) and blockerRightFar ≤ blockerForwardFar and  
blockerRightFar ≤ blockerLeftFar) then return TURN_RIGHT  
if (containBoost(blocksLeftFar) and blockerLeftFar ≤ blockerForwardFar and  
blockerLeftFar ≤ blockerRightFar) then return TURN_LEFT  
if (containEmp(blocksRightFar) and blockerRightFar ≤ blockerForwardFar and  
blockerRightFar ≤ blockerLeftFar) then return TURN_RIGHT  
if (containEmp(blocksLeftFar) and blockerLeftFar ≤ blockerForwardFar and  
blockerLeftFar ≤ blockerRightFar) then return TURN_LEFT  
if (containPowerUps(blocksRightFar) and blockerRightFar ≤ blockerForwardFar and  
blockerRightFar ≤ blockerLeftFar) then return TURN_RIGHT  
if (containPowerUps(blocksLeftFar) and blockerLeftFar ≤ blockerForwardFar and  
blockerLeftFar ≤ blockerRightFar) then return TURN_LEFT  
else return null
```

### 4.3 Struktur Data Program

Pada bot *Overdrive* yang berbasis bahasa Java, pendekatan yang digunakan adalah dengan pemrograman berbasis objek, dimana data akan dienkapsulasi dalam kelas-kelas. Dalam proses pembuatannya, kami juga menambahkan beberapa kelas dan struktur data dalam

keseluruhan program untuk melengkapi implementasi bot. Program yang kami kembangkan secara umum terbagi menjadi lima bagian besar, yaitu kelas-kelas Command, kelas-kelas Entities, kelas-kelas Enums, Bot, dan Main.

a. Command

Pada bagian Command, kelas-kelas yang terdapat di dalamnya melambangkan Command yang dapat dilakukan oleh bot. Kelas-kelas tersebut diantaranya:

- Command

Kelas Command berisi kelas utama yang nantinya akan di *inherit* oleh command-command lainnya. Kelas ini memiliki atribut render yang nantinya akan dimiliki juga oleh semua kelas command lain yang diturunkan dari kelas ini.

- AccelerateCommand

Kelas AccelerateCommand berisi implementasi dari command ACCELERATE, yaitu menambah speed pemain ketika dapat ditambahkan.

- BoostCommand

Kelas BoostCommand berisi implementasi dari command BOOST, yaitu menambah speed pemain menjadi BOOST\_SPEED.

- ChangeLaneCommand

Kelas ChangeLaneCommand berisi implementasi dari command TURN\_RIGHT dan TURN\_LEFT, yaitu command untuk belok ke lane lain.

- DecelerateCommand

Kelas DecelerateCommand berisi implementasi dari command DECELERATE, yaitu mengurangi speed pemain sebanyak satu tingkat.

- DoNothingCommand

Kelas DoNothingCommand berisi implementasi dari command NOTHING, yaitu tidak melakukan apa apa pada ronde tertentu.

- EmpCommand

Kelas EmpCommand berisi implementasi dari command USE\_EMP, yaitu menggunakan PowerUp EMP yang dapat menghentikan musuh.

- FixCommand

- Kelas FixCommand berisi implementasi dari command FIX, yaitu berhenti sejenak dan mengurangi damage yang ada pada mobil.
- LizardCommand  
Kelas LizardCommand berisi implementasi dari command USE\_LIZARD, yaitu menggunakan PowerUp Lizard untuk meloncati jalanan, menjadi kebal terhadap *obstacle* yang terlewati pada ronde tersebut.
  - OilCommand  
Kelas OilCommand berisi implementasi dari command USE\_OIL, yaitu menggunakan PowerUp Oil untuk menurunkan OIL\_SPILL pada position sekarang, yang dapat men-*damage* dan mengurangi kecepatan mobil lawan.
  - TweetCommand  
Kelas TweetCommand berisi implementasi dari command USE\_TWEET <lane> <block>, yaitu menggunakan PowerUp TWEET untuk menaruh CYBERTRUCK pada lane dan block yang ditentukan oleh pemain untuk mencelakakan musuh.

b. Entities

Pada bagian Entities, kelas-kelas yang terdapat di dalamnya melambangkan entitas yang berada pada permainan *Overdrive* beserta dengan atribut yang dimilikinya. Kelas-kelas tersebut antara lain:

- Car  
Kelas Car berisi atribut-atribut yang dapat dimiliki oleh mobil pemain yang disuplai dari game engine. Atribut-atribut tersebut ialah id, position, speed, damage, state, powerups, boosting, dan boostCounter.
- GameState  
Kelas GameState berisi atribut-atribut yang menggambarkan kondisi permainan pada suatu ronde. Atribut-atribut tersebut ialah currentRound, maxRounds, player, opponent, dan worldMap.
- Lane  
Kelas Lane berisi atribut-atribut yang menggambarkan kondisi lane-block pada permainan. Atribut-atribut tersebut ialah position, surfaceObject, occupiedById, dan isOccupiedByCyberTruck. Penggambaran kondisi

dilakukan dengan menyebutkan posisi lane <block, lane>, lalu apa yang berada di block tersebut digambarkan oleh atribut-atribut lainnya.

- Position

Kelas Position berisi atribut-atribut yang menggambarkan posisi pada peta permainan. Atribut-atribut tersebut ialah lane yang menggambarkan sumbu y, dan block yang menggambarkan sumbu x.

- c. Enums

Pada bagian Enums, kelas-kelas yang terdapat di dalamnya melambangkan konstanta yang ada pada permainan *Overdrive* agar dapat dibaca dengan lebih baik oleh bot. Kelas-kelas tersebut antara lain:

- Direction

Kelas Direction berisi enumerasi arah-arah yang dapat terjadi pada game ini, yaitu FORWARD, BACKWARD, LEFT, dan RIGHT.

- PowerUps

Kelas PowerUps berisi enumerasi dari semua PowerUp yang berada pada game *Overdrive*. Powerups tersebut antara lain BOOST, OIL, TWEET, LIZARD, dan EMP.

- State

Kelas State berisi enumerasi dari semua state yang mungkin dimiliki oleh seorang pemain, yaitu ACCELERATING, READY, NOTHING, TURNING\_RIGHT, TURNING\_LEFT, HIT\_MUD, HIT\_OIL, DECELERATING, PICKED\_UP\_POWERUP, USED\_BOOST, USED\_OIL, USED\_LIZARD, USED\_TWEET, HIT\_WALL, HIT\_CYBER\_TRUCK, dan FINISHED.

- Terrain

Kelas Terrain berisi enumerasi dari semua kemungkinan isi block, yaitu EMPTY, MUD, OIL\_SPILL, OIL\_POWER, FINISH, BOOST, WALL, LIZARD, TWEET, dan EMP.

- d. Bot

Pada bagian Bot, terdapat satu kelas Bot pada file Bot.java yang berisi implementasi inti dari bot. Pada kelas ini, terdapat beberapa atribut dan fungsi yang berguna untuk keberjalanannya sendiri. Atribut-atribut tersebut dapat dibagi menjadi beberapa kelompok, diantaranya:

- Instance dari Command

Kelas Bot berisi beberapa atribut yang melambangkan instansiasi dari command untuk mempermudah akses. Command tersebut sama dengan yang telah disebutkan pada kolom Command.

- Informasi Mobil Pemain dan Lawan

Kelas Bot juga berisi beberapa atribut yang melambangkan kondisi dari mobil pemain dan lawan, yang merupakan pemanggilan dari atribut-atribut yang berada pada Entity Car. Atribut-atribut tersebut antara lain myCar, opponent, myCarLane, myCarBlock, myCarSpeed, opponentLane, opponentBlock, opponentSpeed, dan boosting.

- Informasi Block

Kelas Bot juga berisi beberapa atribut yang melambangkan kondisi dari block yang dapat dilihat pemain. Atribut-atribut tersebut antara lain map, startBlock, currentBlock, idxBlock, blocksFront, blocksFrontFar, blocksRight, blocksRightFar, blocksLeft, blocksLeftFar. Untuk map, digunakan tipe data List<Lane[]> dan untuk atribut blocks<Direction> menggunakan tipe data List<Object>.

- Informasi Status dan Decision

Kelas Bot juga berisi beberapa atribut yang melambangkan status game saat ini dan juga kemungkinan yang dapat diambil oleh bot, sebagai keluaran dari fungsi logika pada bot. Atribut-atribut tersebut ialah canRight, canRightFar, canLeft, canLeftFar, canForward, dan canForwardFar yang bertipe integer, lalu endGame ,lastBlocked, dan canAccelerate yang bertipe boolean, dan bestLane yang bertipe Command, sebagai output akhir dari pertimbangan pembelokan.

- Informasi Power Up

Kelas Bot juga berisi beberapa atribut yang melambangkan informasi power up yang sedang dimiliki oleh pemain. Atribut ini ada yang bersifat singuler (dilambangkan dengan tipe boolean) yaitu hasLizard, hasBoost, hasEMP, hasOil, dan hasTweet, dan jumlah yang dimiliki (dilambangkan dengan tipe integer) yaitu countLizard, countBoost, countOil, dan countTweet.

- Informasi Tambahan

Kelas Bot juga berisi atribut yang melambangkan informasi tambahan, seperti array yang melambangkan indeks kecepatan (int listSpeed[]).

e. Main

Pada bagian Main, terdapat *entrypoint* dari bot, atau dengan kata lain bagian kode pertama yang dijalankan saat bot pertama kali dipanggil dari *executable*-nya. Isi dari kelas Main tidak terlalu banyak, hanya terdapat inisialisasi hal-hal teknis yang berkaitan dengan penerimaan game state dan penerjemahan command dari bot menuju game engine.

#### 4.4 Analisis dan Pengujian

Kami melakukan analisis dan pengujian dengan cara mempertarungkan bot yang kami punya dengan bot kami sendiri. Setelah itu kami amati *behavior* dari bot tersebut dengan menggunakan *visualizer* yang dibuat oleh community dari entelect 2020. Visualizer tersebut dapat dilihat pada link berikut: <https://entelect-replay.raezor.co.za/>.

Alasan kami menggunakan tool tersebut adalah tool tersebut memiliki *UI/UX* yang bagus, website tersebut sangat mudah untuk dimengerti, dan sangat mudah untuk kebutuhan testing dan analisis. Pengujian dilakukan dengan hal yang serupa yaitu setelah kami membuat algoritma yang berkaitan dengan strategi yang telah dibuat, lalu akan dilakukan *testing* dengan melawan bot kami sendiri. Hal ini pun dapat memudahkan kita dalam melakukan debugging, Karena jika bot yang kita buat memiliki *behavior* yang tidak sesuai dengan kode yang telah kita buat maka kemungkinan terdapat sebuah bug pada kode tersebut.

Terdapat alternatif lain yang bisa kita gunakan, yaitu menggunakan *seed* yang sama setiap pertandingan. Awalnya seed yang kita gunakan bersifat random sehingga tampak bahwa map di generate secara random, namun kita bisa melakukan sedikit modifikasi pada bot agar seed yang digunakan selalu sama sehingga map yang dibentuk selalu sama setiap pertandingan.

Approach diatas memiliki keuntungan dan kekurangannya masing masing, seperti kita tidak bisa mendapatkan semua kemungkinan map dan bot yang kita buat tidak *general* untuk setiap map. Berdasarkan Analisis yang telah dipaparkan diatas, kami memutuskan untuk menggunakan *approach* yang general.

Berikut adalah analisis yang kami lakukan terhadap suatu pertandingan melawan bot sendiri menggunakan *tools* yang telah disebutkan di atas.

[1, 1] <PLAYER>	[2, 1]	[3, 1]	[4, 1]	[5, 1]	[6, 1]	[7, 1]	[8, 1]	[9, 1]	[10, 1]	[11, 1]	[12, 1]	[13, 1]	[14, 1]	[15, 1]	[16, 1]	[17, 1] <MOOD>	[18, 1]	[20, 1]	
[1, 2]	[2, 2]	[3, 2]	[4, 2]	[5, 2]	[6, 2]	[7, 2]	[8, 2]	[9, 2]	[10, 2]	[11, 2]	[12, 2] <MOOD>	[13, 2]	[14, 2]	[15, 2]	[16, 2]	[17, 2]	[18, 2]	[19, 2]	[20, 2] <LIZARD>
[1, 3]	[2, 3]	[3, 3]	[4, 3]	[5, 3]	[6, 3]	[7, 3]	[8, 3]	[9, 3] <MOOD>	[10, 3]	[11, 3]	[12, 3]	[13, 3]	[14, 3]	[15, 3]	[16, 3]	[17, 3]	[18, 3]	[19, 3]	[20, 3] <MOOD>
[1, 4] <PLAYER>	[2, 4]	[3, 4]	[4, 4]	[5, 4]	[6, 4]	[7, 4]	[8, 4] <MOOD>	[9, 4]	[10, 4]	[11, 4] <MOOD>	[12, 4]	[13, 4]	[14, 4] <MOOD>	[15, 4]	[16, 4]	[17, 4]	[18, 4]	[19, 4]	[20, 4]

(Click the button that displays the round number to quickly switch rounds)

Round Details		Max Rounds: 500	Current Round: 1
<b>A - FASTabiqul-racing-SUPERIOR</b> <span style="float: right;">(selected) ▾</span>			
Position <b>1</b> [x: 1, y: 1]	Speed <b>5</b>	Lane <b>1</b>	Distance <b>1</b>
Boosts <b>0</b>	Boosting <b>No</b>	Powerups	
<b>Bot Command</b> Command: ACCELERATE Execution time: 329ms Exception: null			

Gambar 2 State awal pertandingan bot

The screenshot shows a racing game interface. At the top is a 4x6 grid of lane numbers from [1,1] to [24,1]. Below this is a 4x6 grid of lane numbers from [1,2] to [24,2]. Lane 11, row 2, has a red border and contains the text "[11,2] <OBSTACLE>". Lane 12, row 2, has a blue border and contains the text "[12,2] <MOULD>". Lane 10, row 3, has a green border and contains the text "[10,3] <OBSTACLES>". Lane 11, row 3, has a red border and contains the text "[11,3] <OBSTACLES>". Lane 12, row 3, has a blue border and contains the text "[12,3] <OBSTACLES>". Lane 13, row 3, has a green border and contains the text "[13,3] <OBSTACLES>". Lane 14, row 3, has a red border and contains the text "[14,3] <OBSTACLES>". Lane 15, row 3, has a blue border and contains the text "[15,3] <OBSTACLES>". Lane 16, row 3, has a green border and contains the text "[16,3] <OBSTACLES>". Lane 17, row 3, has a red border and contains the text "[17,3] <OBSTACLES>". Lane 18, row 3, has a blue border and contains the text "[18,3] <OBSTACLES>". Lane 19, row 3, has a green border and contains the text "[19,3] <OBSTACLES>". Lane 20, row 3, has a red border and contains the text "[20,3] <OBSTACLES>". Lane 21, row 3, has a blue border and contains the text "[21,3] <OBSTACLES>". Lane 22, row 3, has a green border and contains the text "[22,3] <OBSTACLES>". Lane 23, row 3, has a red border and contains the text "[23,3] <OBSTACLES>". Lane 24, row 3, has a blue border and contains the text "[24,3] <OBSTACLES>".

**Round Details**

A - FASTabiquil-racing-SUPERIOR (selected) ▾

Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
1 [x: 7, y: 1]	6	1	7	0	No	

**Bot Command**

Command: TURN\_RIGHT  
Execution time: 23ms  
Exception: null

Gambar 3 Pertandingan bot ronde awal

Menurut Logika normal seharusnya lebih efektif untuk lurus dahulu lalu berbelok ke kanan, namun kami memprioritaskan mobil untuk berada pada lane 2 ataupun 3 karena untuk kedepannya terdapat banyak *possibility* yang bagus jika mobil kita berada di tengah, bukan di ujung lane.

Pada ronde 5 terlihat bahwa *decision* yang diambil ialah melakukan belok kanan, karena jika kita perhatikan pada *lane* kita terdapat blocker yaitu *mud* dan pada *lane* kiri terdapat *obstacle* yaitu *wall* sehingga dengan menggunakan *greedy by damage* kita dapatkan keputusan terbaik yaitu untuk melakukan belok ke kanan.

Round 028

Reset Remove this match

[195,1]	[196,1]	[197,1]	[198,1]	[199,1]	[200,1] (MUD)	[201,1]	[202,1]	[203,1]	[204,1]	[205,1]	[206,1]	[207,1]	[208,1]	[209,1]	[210,1]	[211,1] (MUD)	[212,1] (MUD)	[213,1]	[214,1]
[195,2]	[196,2]	[197,2]	[198,2]	[199,2] (PLAYER)	[200,2]	[201,2]	[202,2]	[203,2]	[204,2]	[205,2]	[206,2]	[207,2]	[208,2] (PLAYER)	[209,2]	[210,2]	[211,2]	[212,2]	[213,2]	[214,2]
[195,3]	[196,3]	[197,3]	[198,3]	[199,3]	[200,3]	[201,3]	[202,3]	[203,3]	[204,3]	[205,3]	[206,3]	[207,3]	[208,3]	[209,3]	[210,3]	[211,3] (MUD)	[212,3]	[213,3]	[214,3]
[195,4]	[196,4]	[197,4]	[198,4]	[199,4] (PLAYER)	[200,4]	[201,4]	[202,4]	[203,4]	[204,4]	[205,4]	[206,4]	[207,4]	[208,4]	[209,4]	[210,4]	[211,4] (MUD)	[212,4]	[213,4]	[214,4]

[ First ] [ < Prev ] 28 [ Next > ] [ Last ]  
(Click the button that displays the round number to quickly switch rounds)

Round Details

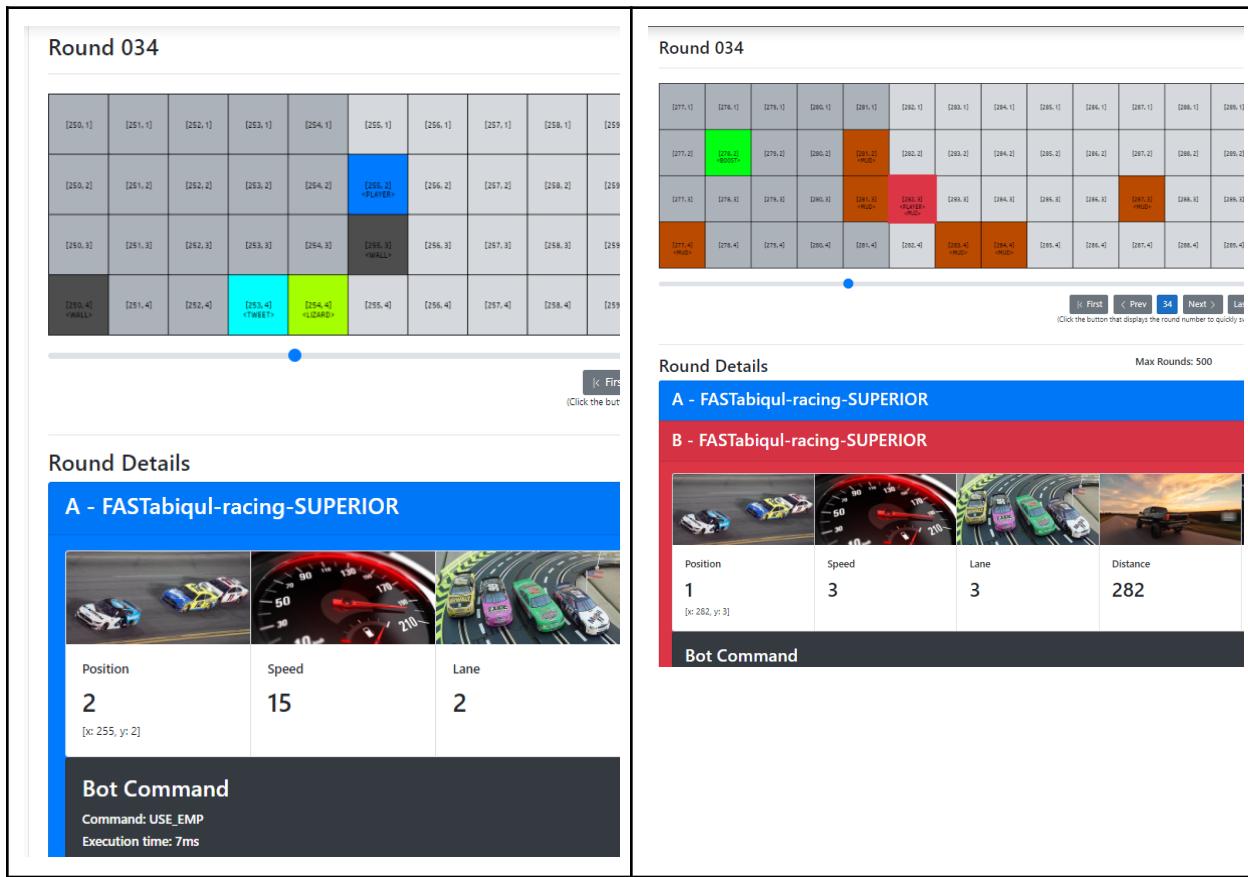
Current Round: 28

A - FASTabiquil-racing-SUPERIOR (selected) ▾

Position 2 [x: 194, y: 2]	Speed 9	Lane 2	Distance 194	Boosts 0	Boosting No	Powerups LIZARD,LIZARD,OIL,OIL,OIL,BOOST	
Bot Command Command: FIX Execution time: 14ms Exception: null							

Gambar 4 State pertandingan bot ronde 28

Pada ronde 28, terlihat bahwa ketika kita memiliki sebuah boost namun kita masih memiliki *damage* pada kendaraan kita maka hal yang pertama kita lakukan adalah melakukan *command fix* karena *boost* akan maksimal jika kendaraan kita tidak memiliki *damage* sama sekali yang akan menaikkan *speed* kendaraan kita menjadi *boost speed* yaitu 15. Hal yang baru saja dilakukan termasuk ke dalam kategori *greedy by speed* dengan melakukan fix terlebih dahulu speed yang didapat pada ronde selanjutnya akan lebih maksimal dibandingkan dengan langsung melakukan *boost*.



Gambar 5 State pertandingan bot ronde 34

Pada ronde 34 ini terlihat bahwa posisi musuh berada di depan kendaraan kita. Terdapat beberapa keputusan yang kita ambil misalnya kita dapat melakukan pergerakan lebih jauh ke depan dengan menggunakan *command lizard* ataupun *do nothing* memang hal itu memberikan kita keuntungan yaitu posisi kita akan semakin mendekati garis *finish* namun akan lebih baik lagi jika kita menggunakan *sisi offensive* nya atau bisa kita sebut *greedy by enemy speed*. Agar kesempatan kendaraan kita menang lebih tinggi maka jarak antar kendaraan tidak boleh jauh. Maka kami menetapkan suatu *threshold* yaitu 20 *block*. Jika kendaraan lawan sudah berada didepan kita dan lebih dari 20 *block* maka hal yang bisa kita lakukan ialah mencegah musuh untuk melakukan pergerakan dengan menggunakan *command emp*.

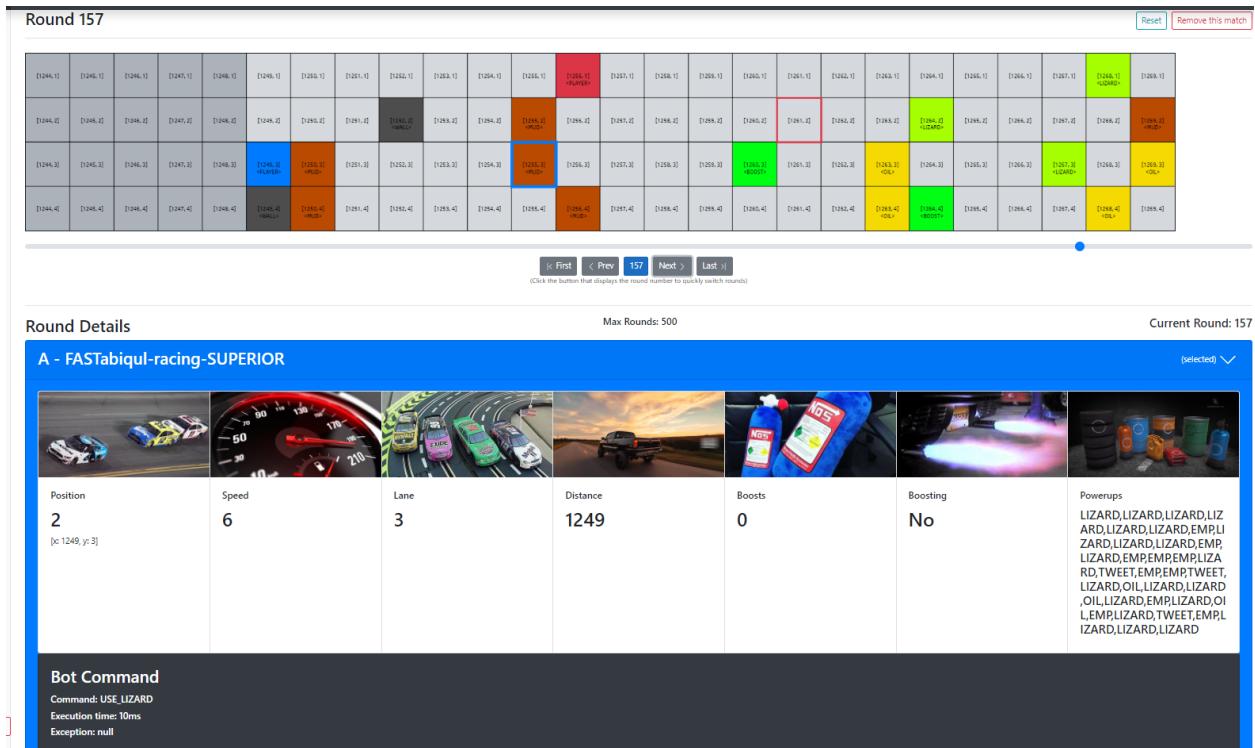
Round 047		Reset		Remove this match																					
[346, 1]	[347, 1]	[348, 1]	[349, 1]	[350, 1] + [350, 2]	[351, 1]	[352, 1]	[353, 1]	[354, 1]	[355, 1]	[356, 1]	[357, 1] + [357, 2]	[358, 1]	[359, 1]	[360, 1]	[361, 1]	[362, 1]	[363, 1] + [363, 2]	[364, 1]	[365, 1]	[366, 1]	[367, 1] + [367, 2]	[368, 1]	[369, 1]	[370, 1]	[371, 1]
[346, 2]	[347, 2]	[348, 2] + [348, 3]	[349, 2] + [349, 3]	[350, 2]	[351, 2] + [351, 3]	[352, 2]	[353, 2]	[354, 2]	[355, 2]	[356, 2]	[357, 2]	[358, 2]	[359, 2]	[360, 2]	[361, 2]	[362, 2]	[363, 2] + [363, 3]	[364, 2]	[365, 2] + [365, 3]	[366, 2]	[367, 2]	[368, 2]	[369, 2]	[370, 2]	[371, 2]
[346, 3]	[347, 3]	[348, 3]	[349, 3]	[350, 3]	[351, 3] + [351, 4]	[352, 3]	[353, 3]	[354, 3]	[355, 3]	[356, 3]	[357, 3]	[358, 3]	[359, 3]	[360, 3]	[361, 3]	[362, 3]	[363, 3]	[364, 3]	[365, 3]	[366, 3]	[367, 3]	[368, 3]	[369, 3]	[370, 3]	[371, 3]
[346, 4] + [346, 5]	[347, 4]	[348, 4]	[349, 4]	[350, 4]	[351, 4]	[352, 4]	[353, 4]	[354, 4]	[355, 4]	[356, 4]	[357, 4]	[358, 4]	[359, 4]	[360, 4]	[361, 4]	[362, 4]	[363, 4] + [363, 5]	[364, 4]	[365, 4]	[366, 4]	[367, 4]	[368, 4]	[369, 4]	[370, 4]	[371, 4]
47						<a href="#">First</a>	<a href="#">Prev</a>	47	<a href="#">Next</a>	<a href="#">Last</a>															
(Click the button that displays the round number to quickly switch rounds)																									

Gambar 6 State pertandingan bot ronde 47

Pada ronde 47 terdapat hal yang menarik. Kendaraan kita memiliki alternatif *command* yang dapat digunakan yaitu ***accelerate*** namun kondisi tersebut menurut kami kurang optimal karena masih ada hal yang bisa kita lakukan agar mendapatkan outcome terbaik dari ronde ini dengan menggunakan ***greedy by point***. Yaitu ketika kendaraan lawan berada pada *lane* yang sama pada kendaraan kita maka kita gunakan *oil*. *Oil* berfungsi sebagai *blocker* yang bisa kita dapatkan dari pengambilan *powerup*. *Oil* dapat menurunkan *speed state* by 1 sehingga outcome ini akan lebih menguntungkan karena pada ronde ini, kita berhasil menggunakan *power up* maka score kita akan bertambah sebesar 4 dan kita berhasil membuat pergerakan musuh terhambat dengan blocker kita. *Outcome* ini jelas lebih menguntungkan jika dibandingkan dengan melakukan *command accelerate*.

Gambar 7 State pertandingan bot ronde 106

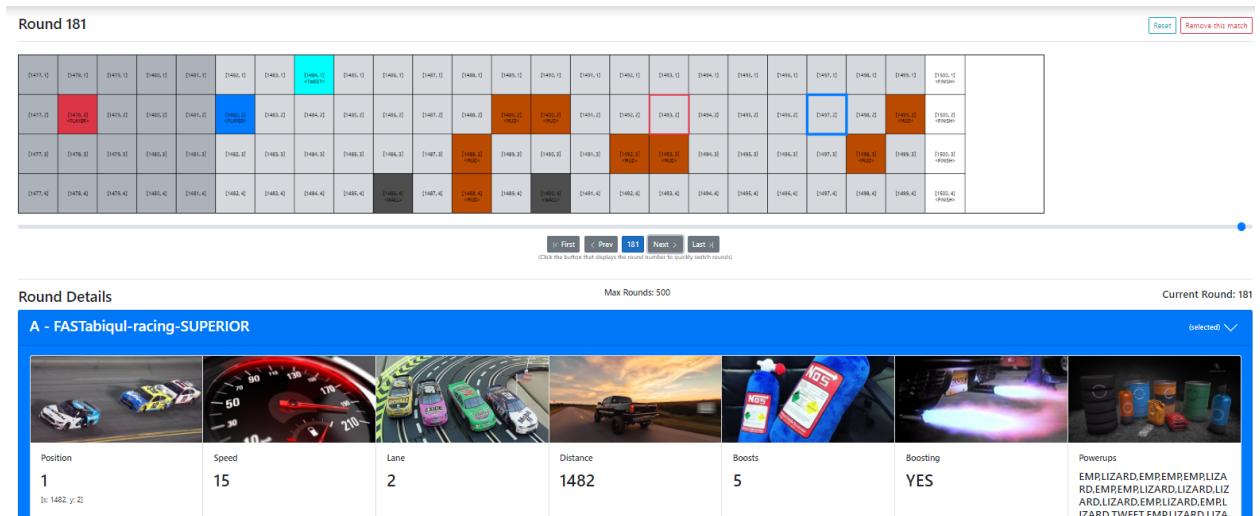
Pada ronde ini seharusnya menerapkan ***greedy by speed*** namun kendaraan kita tidak memiliki ***power up*** boost maka dilakukanlah ***greedy by point***. Karena EMP kita prioritaskan untuk jarak yang agak jauh dan oil hanya bisa kita gunakan jika kita berada di depan kendaraan lawan maka hal yang terbaik yang bisa kendaraan kita lakukan ialah menggunakan ***command TWEET***. Command ini akan membuat sebuah blocker pada *lane* serta *block* yang kita tentukan sendiri. Pada kasus ini kita menggunakan **USE\_TWEET 2 289** artinya akan ditaruh sebuah *cybertruck* pada *block* 289 yang berada pada *lane* 2. Mengapa dipilih *block* 289, hal ini karena kami membuat prediksi *block* terdekat yang mana yang akan dilewati musuh. Prediksi tersebut bisa diambil dengan menggunakan *opponentSpeed* serta *opponentPosition*. Strategi kendaraan kita ialah untuk menaruh *cybertruck* pada posisi *opponentSpeed + opponent.Position.block + 3*. Mengapa terdapat penambahan konstan 3? Hal itu dilakukan karena sewaktu - waktu kendaraan dapat melakukan ***ACCELERATE*** yang akan menambah speed kendaraan tersebut, maka agar prediksi yang dilakukan lebih akurat kami menggunakan angka konstan 3 tersebut.



Gambar 8 State pertandingan bot ronde 117

Pada ronde 157 ini terdapat kasus yang menarik. Pada *lane* kendaraan kita baik tengah, kiri maupun kanan terdapat sebuah blocker yang menghalangi pergerakan kendaraan kita. Maka saat ini merupakan saat yang tepat untuk melakukan *greedy by damage* namun greedy by damage saja tidak cukup, perlu adanya pembobotan setiap obstacle karena setiap *obstacle* memiliki efek nya masing masing. Karena kita melakukan *approach* dengan menggunakan greedy by damage maka akan dicari damage minimum pada ronde ini bagaimanapun caranya tentunya setelah ditambah dengan pembobotan setiap *obstacle*. Maka *outcome* yang terbaik adalah menggunakan *LIZARD*. *Lizard* dapat meloncati *block* namun masih terkena efek dari *block* ketika kita mendarat. Tentunya ini lebih baik dibandingkan kita *ACCELERATE* dan terkena lebih banyak *damage*.

Gambar 9 State pertandingan bot ronde 169 dan 175



Gambar 10 State pertandingan bot ronde 181

Pada ronde 169, 175, 181 terdapat kasus menarik. Karena posisi *block* sudah lebih besar dari 1350, maka akan masuk kedalam mode *endgame*. Mode ini akan mengutamakan *speed* serta penggunaan *power up* secara *offensive*. Pada kasus normal, kendaraan kita akan melakukan *command turn\_right* pada ronde 169 dan *command turn\_left* pada ronde 175 dan 181. Namun, pada kondisi *endgame* ini karena diutamakan kecepatan maka akan lebih baik jika kita menggunakan *power up lizard* karena akan memberikan distance lebih dekat ke pada garis finish.

Setelah melakukan analisis dan pengujian secara iteratif, maka kelompok kami memutuskan bahwa **algoritma yang kami desain sudah cukup baik** meskipun masih ada *room for improvement*. Hal hal yang bisa ditingkatkan yaitu, pada ronde 2 kita bisa memprioritaskan untuk speed terlebih dahulu dibandingkan dengan pindah ke tengah lane, serta meningkatkan *usage* dari *power up* yang didapat karena dapat dilihat bahwa laporan yang dimiliki oleh kendaraan kita pada garis *finish* masih cukup banyak dan masih bisa dibuat lebih efektif sehingga mendapatkan *score* yang optimal.

## **BAB 5**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Dari tugas besar ini, kami berhasil membuat bot untuk permainan *Overdrive* dengan pendekatan algoritma greedy. Program akhir yang dibuat tidak hanya memanfaatkan satu strategi greedy, melainkan menggabungkan beberapa elemen dari banyak alternatif solusi greedy untuk menghasilkan strategi terbaik versi kami.

#### **5.2 Saran**

Saran untuk kelompok kami di antaranya :

1. Pembagian tugas dilakukan dengan lebih baik lagi, agar alur bekerja lebih jelas.
2. Algoritma yang digunakan pada tugas besar ini masih dapat dikembangkan lebih lanjut lagi, baik dari segi format kode sampai efektivitas algoritma
3. Kode program diberi komentar yang lebih jelas lagi. Agar kode yang digunakan lebih mudah dimengerti dan semua anggota bisa melakukan *debugging* pada algoritma yang bermasalah

## **DAFTAR PUSTAKA**

[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)

<https://github.com/EntelectChallenge/2020-Overdrive>