LAPORAN TUGAS BESAR 1

IF2211 Strategi Algoritma

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"

Semester II Tahun 2021/2022

Disusun oleh:

Ahmad Romy Zahran	13520009
Aji Andhika Falah	13520012
Farhan Hafiz	13520027

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2021

Daftar Isi

Bab 1 : 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.

Gambar 1 Contoh tampilan permainan Overdrive

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Overdrive pada tautan di atas. 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 ronde. 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 *powerups*. 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 : Landasan Teori

2.1 Algoritma *Greedy*

Algoritma *greedy* merupakan jenis algoritma yang menggunakan pendekatan penyelesaian masalah dengan mencari nilai maksimum sementara pada setiap langkahnya. Nilai maksimum sementara ini dikenal dengan istilah *local optimum*. Solusi yang diharapkan dari memilih *local optimum* adalah solusi yang paling baik secara keseluruhan atau dikenal juga sebagai *global optimum* . Algoritma greedy biasanya digunakan dalam permasalahan optimasi. Persoalan optimasi adalah persoalan-persoalan yang mencari *global optimum* . Terdapat 2 jenis permasalahan optimasi yaitu maksimasi (mencari solusi terbesar) dan minimasi (mencari solusi terkecil).

Dalam algoritma *greedy* , terdapat beberapa elemen yang digunakan untuk membantu memvisualisasikan proses penyelesaian masalah. Elemen-elemen tersebut antara lain :

1. Himpunan Kandidat, adalah himpunan yang berisi elemen pembentuk solusi.
2. Himpunan Solusi, adalah himpunan yang terpilih sebagai solusi persoalan.
3. Fungsi Seleksi, adalah fungsi memilih kandidat yang paling mungkin mencapai solusi optimal
4. Fungsi Kelayakan, adalah fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.
5. Fungsi Solusi, adalah fungsi yang mengembalikan nilai boolean, *True* jika himpunan solusi yang sudah terbentuk merupakan solusi yang lengkap; *False* jika himpunan solusi belum lengkap
6. Fungsi Objektif, adalah fungsi yang mengoptimalkan Solusi

Dengan menggunakan keenam elemen diatas, algoritma *greedy* melakukan proses penyelesaian masalahnya. Dalam tugas ini, algoritma *greedy* akan digunakan untuk menyelesaikan permasalahan optimasi “Memenangkan Balapan dalam Permainan Overdrive”. Permainan tersebut akan dimainkan dalam sebuah game engine yang akan dijelaskan juga dalam laporan ini.

2.2 *Game Engine* Permainan Overdrive

Game engine adalah sistem perangkat lunak yang dirancang untuk menjadi dasar pembuatan permainan video, seperti permainan di komputer, konsol, atau ponsel. Sebuah

game engine dapat digunakan untuk membuat lebih dari satu permainan, dan pengembang permainan dapat mengoptimisasi proses pengembangan dengan cara menggunakan atau mengadaptasi *game engine* yang telah ada sebelumnya. Pada sub-bab ini akan dijelaskan mengenai *game engine* yang digunakan untuk permainan Overdrive, prerekuisit, implementasi algoritma *greedy*, dan cara menjalakan *game engine* pada permainan.

2.2.1 Prerekuisit

Permainan Overdrive dalam tugas ini dijalankan menggunakan game engine yang sudah tersedia. Game engine permainan ini dapat diperoleh pada laman berikut ini : (https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4). Untuk dapat menjalankan permainan, dibutuhkan beberapa aplikasi yang harus di-install sesuai dengan spesifikasi yang tertera yaitu Java (Disarankan JDK 8), code editor IntelliJ IDEA (untuk melakukan build menjadi file exetuable java (.jar)), dan NodeJs.

2.2.2 Komponen Permainan Overdrive

Dalam permainan Overdrive yang telah tersedia, terdapat beberapa komponen yang diperlukan agar permainan dapat berjalan, antara lain :

1. *Game engine* : berupa interface yang digunakan game runner untuk dapat menjalankan atau mengembangkan permainan, membuat bot yang dibuat oleh pemain mematuhi aturan yang didefinisikan, serta memberi perintah yang dikirimkan oleh bot ke dalam permainan untuk diproses

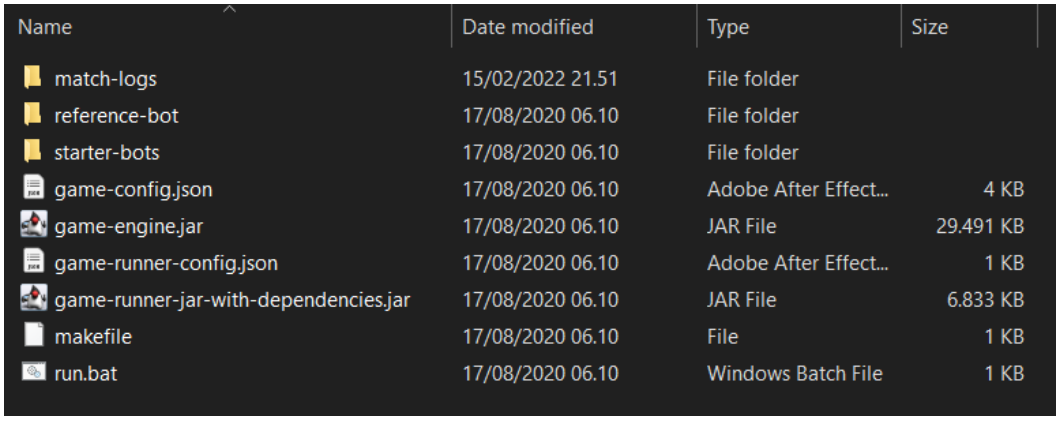
2. *Game runner* : berguna untuk menjalankan permainan antar pemain. Game runner menerima perintah dari bot dan mengirimkannya ke game engine untuk dieksekusi

3. *Reference bot* : bot yang telah disediakan (dibuat dalam Bahasa javascript) agar dapat diujikan dengan bot yang akan dibuat.

4. *Starter bot* : bot awal dengan logika dasar yang akan dikembangkan sesuai kebutuhan (dapat dibuat dalam Bahasa python, C++, Java, dan lain-lain)

2.2.3 Membuat *Starter Bot* dalam Bahasa Java

Pada laman sebelumnya, terdapat file zip yang dapat digunakan yang didalamnya terdapat folder Bernama starter-pack. Tampilan dalam dari folder starter-pack tersebut tampak seperti di bawah ini :



Name	Date modified	Type	Size
match-logs	15/02/2022 21.51	File folder	
reference-bot	17/08/2020 06.10	File folder	
starter-bots	17/08/2020 06.10	File folder	
game-config.json	17/08/2020 06.10	Adobe After Effect...	4 KB
game-engine.jar	17/08/2020 06.10	JAR File	29.491 KB
game-runner-config.json	17/08/2020 06.10	Adobe After Effect...	1 KB
game-runner-jar-with-dependencies.jar	17/08/2020 06.10	JAR File	6.833 KB
makefile	17/08/2020 06.10	File	1 KB
run.bat	17/08/2020 06.10	Windows Batch File	1 KB

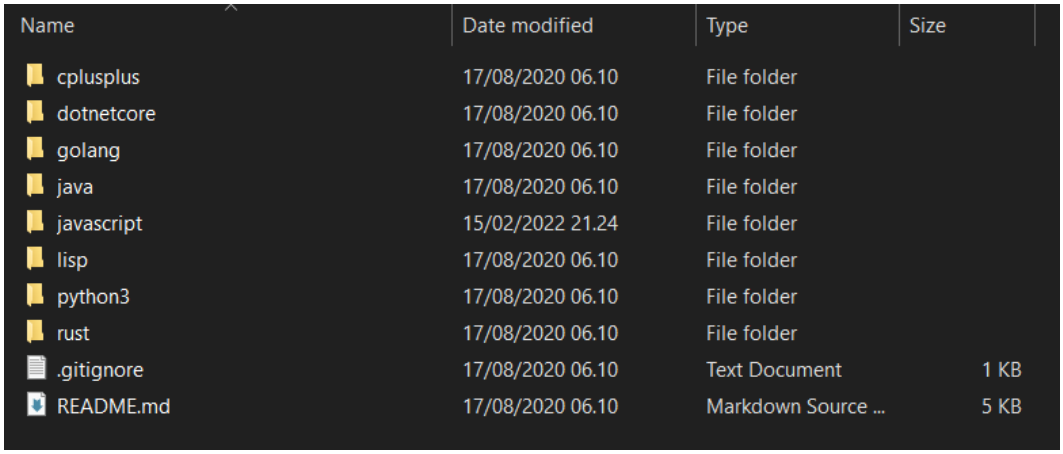
Gambar 2 Struktur folder starter-pack

Pada gambar di atas, terlihat *game engine* sudah tersedia dalam bentuk *executable jar file*. *File* tersebut digunakan untuk menjalankan permainan. Selain itu, terdapat beberapa *file* konfigurasi yang ada pada *starter-pack* antara lain :

1. game-config.json : digunakan untuk mengatur konfigurasi dasar dari game Overdrive berupa atribut-atribut seperti jumlah *lane*, informasi *car*, *powerups*, dan lain-lain

2. game-runner-config.json : digunakan untuk mengatur lokasi *file* konfigurasi *game*, *game engine*, bot pemain satu, bot pemain dua, dan *file* hasil permainan.

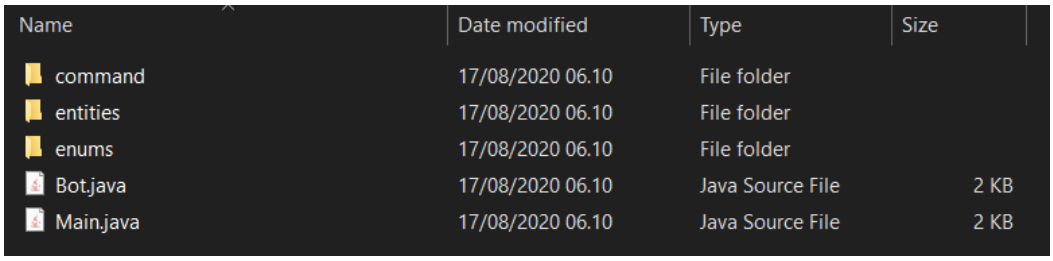
Pada permainan ini, hal yang akan diuji adalah bot yang telah dibuat sesuai kebutuhan dan dimainkan. Seperti yang telah dijelaskan sebelumnya, terdapat dua buah bot yang tersedia. Bot pertama adalah reference bot, bot bawaan dari game engine yang menjalankan program dalam bahasa JavaScript untuk melakukan permainan. Bot kedua adalah starter bot, bot yang dapat diprogram ulang sesuai kebutuhan. Folder starter-bots memiliki beberapa sub-folder lagi yang terdiri dari berbagai Bahasa seperti gambar di bawah ini :



Name	Date modified	Type	Size
cplusplus	17/08/2020 06.10	File folder	
dotnetcore	17/08/2020 06.10	File folder	
golang	17/08/2020 06.10	File folder	
java	17/08/2020 06.10	File folder	
javascript	15/02/2022 21.24	File folder	
lisp	17/08/2020 06.10	File folder	
python3	17/08/2020 06.10	File folder	
rust	17/08/2020 06.10	File folder	
.gitignore	17/08/2020 06.10	Text Document	1 KB
README.md	17/08/2020 06.10	Markdown Source ...	5 KB

Gambar 3 Struktur folder starter-bot

Walaupun starter bot dapat dibuat dalam berbagai Bahasa, pada tugas ini diwajibkan untuk menuliskannya dalam Bahasa pemrograman Java. Pada folder java terdapat tiga buah folder dan dua buah file awal yang telah disediakan yaitu :



Name	Date modified	Type	Size
command	17/08/2020 06.10	File folder	
entities	17/08/2020 06.10	File folder	
enums	17/08/2020 06.10	File folder	
Bot.java	17/08/2020 06.10	Java Source File	2 KB
Main.java	17/08/2020 06.10	Java Source File	2 KB

Gambar 4 Struktur folder Java pada starter-bot

1. Folder command : berisikan kelas dari perintah yang dapat digunakan oleh pemain/bot. Perintah yang tersedia antara lain Accelerate, Boost, ChangeLane, Decelarate, DoNothing, Emp, Fix, dan Oil.

2. Folder entities : berisikan kelas entitas yang ada pada permainan antara lain Car, GameState, Lane, dan Position.

3. Folder enums : berisikan entitas yang memiliki nilai seperti Direction, PowerUps, State, dan Terrain.

4. File Bot.java : berisikan logika dari bot yang dibuat. Pada file ini akan diimplementasikan strategi greedy dengan memanfaatkan kelas lain yang ada.

4. File Main.java : berfungsi untuk menjalankan Bot.java dengan menginstansiasi bot pada setiap ronde dan menerima perintah dari bot tersebut untuk dikirimkan ke game engine kemudian diproses

Untuk menambahkan atau mengedit bot yang diinginkan, file game-runner-config.json yang berada di folder starter-pack. Setelah itu data lokasi “player-a” diubah menjadi “./starter-bots/java” pada file game-runner-config.json seperti di bawah ini :

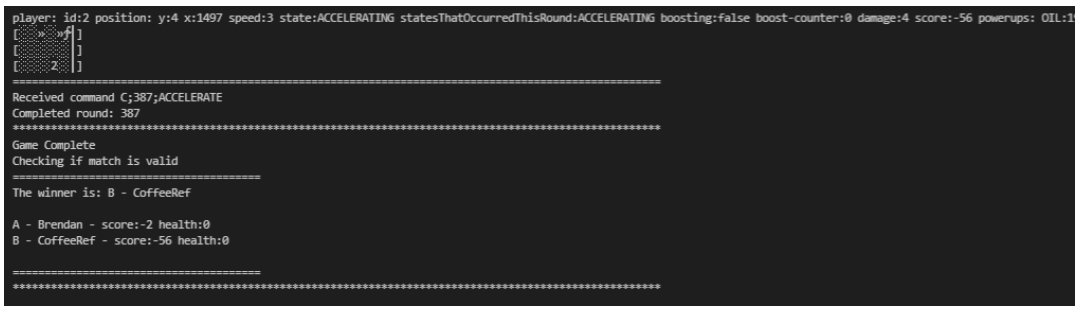
Gambar 5 Tampilan game-runner-config.json

Meskipun starter bot sudah menyediakan berbagai macam kelas beserta method dan attributenya, kelas tersebut masih bisa dikembangan kembali. Didalam bot.java, sudah tersedia beberapa algoritma yang dapat digunakan untuk memainkan Overdrive, akan dikembangan algoritma greedy.

```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) {
        this.random = random;
        this.gameState = gameState;
        this.myCar = gameState.player;
        this.opponent = gameState.opponent;

        directionList.add(-1);
        directionList.add(1);
    }

    public Command run() {
        List<Object> blocks = getBlocksInFront(myCar.position.lane, myCar.position.block);
        if (myCar.damage >= 5) {
            return new FixCommand();
        }
        if (blocks.contains(Terrain.MUD)) {
            int i = random.nextInt(directionList.size());
            return new ChangeLaneCommand(directionList.get(i));
        }
        return new AccelerateCommand();
    }
```

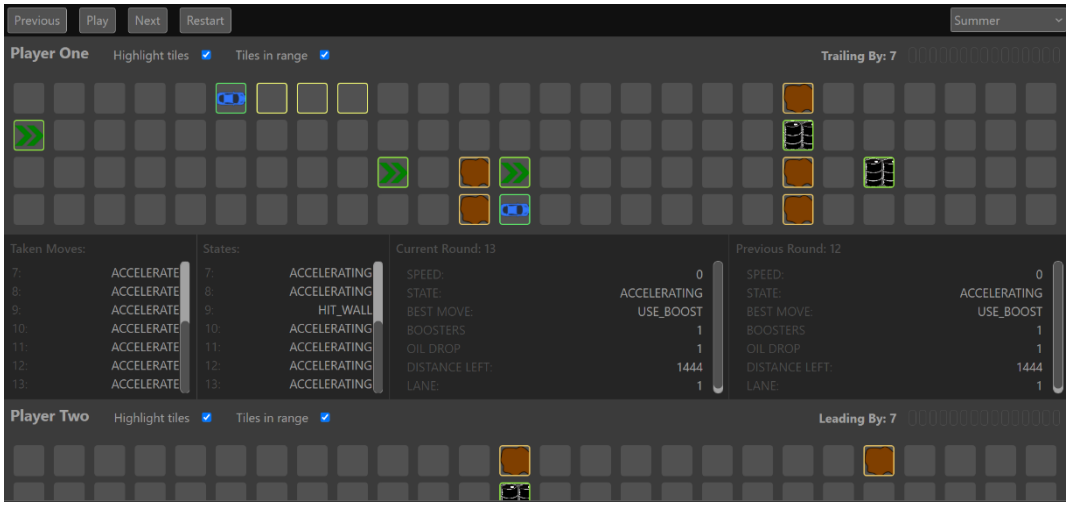
Gambar 6 Tampilan bot.java

Untuk memulai permainan, jalankan file run.bat yang ada pada folder starter-pack. Setelah dijalankan, permainan akan berjalan dan pemenang akan diumumkan pada akhir permainan. Berikut adalah tampilan dari permainan Overdrive



Gambar 7 Tampilan permainan

Untuk meningkatkan kualitas interface dapat menjalankan visualizer yang tersedia pada (https://github.com/Affuta/overdrive-round-runner) yang akan terlihat seperti di bawah ini :



Gambar 8 Tampilan permainan menggunakan visualiser

Bab 3 : Aplikasi Strategi Greedy

3.1 Alternatif Algoritma *Greedy*

3.1.1 *Greedy by Lane*

Greedy by Lane adalah strategi agar car melaju di lane dengan case terbaik. Pada setiap ronde, perhatikan lane yang berada di depan dan selalu hindarkan lane yang memiliki efek buruk terhadar car seperti *Mud, Oil Spill, Flimsy Wall, EMP,* dan *Cybertruck.* Enumerasi semua command dan lakukan command yang sesuai agar selalu tetap di lane yang aman.

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan
- Himpunan Solusi : Lane terpilih dengan case terbaik
- Fungsi Solusi : Memeriksa apakah lane yang dipilih tidak memberi efek buruk atau memberi efek paling minimum terhadap car milik kita
- Fungsi Seleksi : Pilih Lane yang paling tidak memberi efek buruk
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4)
- Fungsi Objektif : Minimumkan efek buruk dari lane yang akan dilewati car

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), ataupun belok kanan (TURN_RIGHT) , sehingga kompleksitasnya adalah :

$$T(n) = 4*3 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Rintangan-rintangan yang bisa memberi efek buruk terhadap car tidak ada atau sangat jarang berada di lane yang sedang car tempati sehingga car tidak perlu mengurangi speed/jumlah block yang dilewatinya saat belok

Strategi ini tidak efektif apabila :

- Rintangan-rintangan yang bisa memberi efek buruk terhadap car berada di posisi yang tidak menguntungkan misalnya pada block delapan terdapat rintangan di lane 1,2,3 kemudian pada block sepuluh terdapat rintangan di lane 2,3,4 sehingga saat berada di block 9, yang awalnya car berada di lane

4 harus berpindah sebanyak tiga kali ke lane 1 yang akan banyak mengurangi kinerja car.

3.1.2 Greedy by Time

Greedy by Time adalah strategi agar car melaju dalam tiap 20 block memiliki waktu yang minimum. Proses dapat dilakukan dengan mengenumerasi semua command, lalu memilih alternatif agar sampai ke 20 blok selanjutnya memiliki waktu yang paling sedikit. Diharapkan, apabila tiap 20 block memiliki waktu yang paling sedikit (minimum local) akan menghasilkan minimum untuk full 1500 block (minimum global).

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan, penggunaan command
- Himpunan Solusi : Alternatif kombinasi command yang menciptakan waktu tercepat
- Fungsi Solusi : Memeriksa apakah alternatif command yang dimasukkan akan menghasilkan waktu yang lebih cepat
- Fungsi Seleksi : Memilih salah satu alternatif kombinasi command dengan waktu terkecil dalam menempuh 20 blok
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4) dan command-command yang dimasukkan valid.
- Fungsi Objektif : Minimumkan waktu yang diperlukan bagi car untuk melewati 1500 block

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), belok kanan (TURN_RIGHT), menambah kecepatan (ACCELERATE), ataupun mengurangi kecepatan (DECELARATE). Akan dicek hingga 20 block kedepan dan dihitung waktu yang diperlukannya hingga mencapai waktu tercepat sehingga kompleksitasnya adalah :

$$T(n) = 4*5*20 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Car lawan tidak memiliki powerups apapun yang akan memberikan efek buruk terhadap car kita

Strategi ini tidak efektif apabila :

- Car lawan memiliki powerup seperti Oil, EMP, ataupu tweet yang akan memberikan efek buruk terhadap car kita dan mengganggu alternatif kombinasi command yang akan digunakan

3.1.3 Greedy by Speed

Greedy by Speed adalah strategi agar car melaju dengan kecepatan yang terus bertambah bahkan hingga maksimal mencapai maximum speed yang didefinisikan, Kecepatan car dapat bertambah dan berkurang baik karena command yang diinput maupun efek dari entitas yang ada. Car akan selalu diusahakan agar ditingkatkan kecepatannya hingga maksimal

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Command move (nothing, accelerate, decelerate)
- Himpunan Solusi : Command accelerate dipilih apabila belum mencapai maksimal
- Fungsi Solusi : Memeriksa apakah kecepatan car telah mencapai maximum_speed
- Fungsi Seleksi : Pilih command accelerate
- Fungsi Kelayakan : Memeriksa apakah command yang digunakan valid
- Fungsi Objektif : Memaksimalkan kecepatan yang dapat dilaju car

b. Analisis Efisiensi Solusi

Mengecek apakah maximum_speed car belum mencapai maksimalnya, apabila belum lakukan command ACCELERATE , sehingga kompleksitasnya adalah:

$$T(n) = 1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Lane yang ditempati car saat itu memiliki rintangan yang bisa memberi efek buruk dengan jumlah minimum sehingga kecepatan yang dimiliki car tidak berkurang atau bahkan rusak

Strategi ini tidak efektif apabila :

- Lane yang ditempati car saat itu memiliki rintangan yang bisa memberi efek buruk dengan jumlah maksimum sehingga kecepatan yang dimiliki car berkurang banyak atau bahkan rusak yang menyebabkan command ACCELERATE sebelumnya menjadi sia-sia.

3.1.4 Greedy by Damage

Greedy by Damage adalah strategi agar car memiliki damage minimum dan mencoba membuat car lawan memiliki damage maksimum. Damage didapatkan saat car mengenai obstacle seperti Mud, Oil, Wall, dan Cybertruck. Car juga harus menghindari tembakan EMP lawan Car harus menghindari obstacle-obstacle yang ada agar tidak memiliki damage yang banyak, apabila damage yang telah diterima car mencapai 3, segera melakukan command FIX agar car tidak rusak yang menyebabkan tidak bisa berjalan lagi.

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan
- Himpunan Solusi : Lane yang tidak memiliki obstacle
- Fungsi Solusi : Memeriksa apakah lane yang dipilih tidak mempunyai obstacle yang bisa memberi damage terhadap car
- Fungsi Seleksi : Pilih Lane yang memiliki damage minimum
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4)
- Fungsi Objektif : Minimumkan damage yang diterima oleh car

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), ataupun belok kanan (TURN_RIGHT), ataupun melakukan perbaikan (FIX), sehingga kompleksitasnya adalah :

$$T(n) = 4*4 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Lane yang sedang ditempati car dan lane didekatnya memiliki jumlah obstacle yang minimum

Strategi ini tidak efektif apabila :

- Tiap lane memiliki obstacle yang sulit dihindari serta car lawan bisa saja memberi powerup berupa tembakan EMP dan cyber_truck yang mengganggu car milik kita.

3.1.5 Greedy by Powerups - Boost

Greedy by Powerups - boost adalah strategi agar selalu mengambil powerups boost yang yang tersedia di lane tertentu dan menggunakannya sebaik mungkin. Apabila car mendapat powerups boost, akan langsung digunakan dengan command USE_BOOST agar melaju kencang.

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan
- Himpunan Solusi : Lane terpilih yang memiliki powerups Boost
- Fungsi Solusi : Memeriksa apakah lane yang dipilih memiliki powerups boost
- Fungsi Seleksi : Pilih Lane yang terdapat powerups boost
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4) serta powerups sudah dimiliki
- Fungsi Objektif : Memaksimalkan penggunaan powerups boost

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), ataupun belok kanan (TURN_RIGHT), setelah mendapatkannya gunakan command USE_BOOST , sehingga kompleksitasnya adalah :

$$T(n) = 4*3*1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Powerups boost berada dalam jangkauan yang relative dekat dari car serta Sebagian besar block kedepan pada lane yang akan digunakan boost tidak memiliki obstacle yang mengurangi kecepatan

Strategi ini tidak efektif apabila :

- Powerups boost berada dalam jangkauan yang jauh serta jalur yang dilewati saat menggunakan boost terdapat obstacle terutama wall yang menghentikan kinerja boost sehingga menyia-nyiakan powerups boost.

3.1.6 Greedy by Powerups – Oil EMP

Greedy by Powerups – Oil EMP adalah strategi agar selalu mengambil powerups oil dan EMP yang yang tersedia di lane tertentu dan menggunakannya sebaik mungkin agar dapat mengenai car lawan. Apabila car mendapat powerups Oil, Pastikan car lawan berada di posisi belakang kita, lalu jika berada di belakang, gunakan command USE_OIL untuk mengganggu lawan. Apabila car mendapat

powerups EMP, Pastikan car lawan berada di posisi depan kita, lalu jika berada di depan, gunakan command USE_EMP untuk menembak lawan.

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan, command powerups
- Himpunan Solusi : Lane terpilih yang memiliki powerups Oil atau EMP, penggunaan comman powerups
- Fungsi Solusi : Memeriksa apakah lane yang dipilih memiliki powerups Oil dan EMP serta posisi car lawan sudah tepat
- Fungsi Seleksi : Pilih Lane yang terdapat powerups Oil dan EMP serta posisi car sesuai
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4) serta powerups sudah dimiliki
- Fungsi Objektif : Memaksimalkan penggunaan powerups Oil dan EMP

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), ataupun belok kanan (TURN_RIGHT) untuk mengambil powerups , setelah itu bandingkan posisi car miliki kita dan milik lawan, jika sesuai lakukan comman powerups yang sesuai, sehingga kompleksitasnya adalah :

$$T(n) = 4*3*2*2 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Powerups Oil ataupun EMP terpakai dengan sebaik mungkin yaitu mengenai car lawan sehingga mengganggu kinerjanya

Strategi ini tidak efektif apabila :

- Powerups Oil ataupun EMP terpakai, namun sama sekali tidak mengenai car lawan sehingga tidak mempengaruhi dan justru mensia-siakan powerup yang diambil.
-

3.1.7 Greedy by Powerups – Tweet

Greedy by Powerups – Tweet adalah strategi agar selalu mengambil powerups Tweet yang yang tersedia di lane tertentu dan menggunakannya sebaik mungkin agar cybertruck dapat mengenai car lawan. Perhatikan posisi car lawan dan gunakan command USE_TWEET <lane> <block> dengan lane dan block merupakan nilai

yang berada di depan car lawan sehingga memperbesar kemungkinan car lawan menabraknya.

a. Mapping Elemen-elemen Greedy

- Himpunan Kandidat : Lane dari arena balapan, command powerups
- Himpunan Solusi : Lane terpilih yang memiliki powerups Tweet, penggunaan comman powerups
- Fungsi Solusi : Memeriksa apakah lane yang dipilih memiliki powerups Tweet dan posisi car lawan
- Fungsi Seleksi : Pilih Lane yang terdapat powerups Tweet serta posisi penggunannya sesuai
- Fungsi Kelayakan : Memeriksa apakah lane valid (terdapat lane 1 sampai lane 4) serta powerups sudah dimiliki
- Fungsi Objektif : Memaksimalkan penggunaan powerups Tweet

b. Analisis Efisiensi Solusi

Terdapat empat lane yang harus diperiksa, lalu mengecek apakah car seharusnya berada di lane yang sama (NOTHING), belok kiri (TURN_LEFT), ataupun belok kanan (TURN_RIGHT) untuk mengambil powerups , setelah itu perhatikan posisi car milik lawan, lakukan command powerups tweet dengan posisi yang sesuai, sehingga kompleksitasnya adalah :

$$T(n) = 4*3*2*1 = O(1)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Powerups Tweet terpakai dengan sebaik mungkin sehingga car lawan menabrak cybertruck yang ada.

Strategi ini tidak efektif apabila :

- Powerups Tweet terpakai, namun sama sekali tidak mengenai car lawan sehingga tidak mempengaruhi dan justru mensia-siakan powerup yang diambil.

3.2 Solusi Algoritma Greedy yang Digunakan

Pada game Overdrive ini terdapat beberapa solusi algoritma greedy seperti yang telah dijelaskan sebelumnya yang bisa diimplementasikan dan digunakan tergantung pada kondisi yang ada untuk setiap rondenya. Terdapat beberapa skema dan keadaan yang harus ditangani oleh bot, antara lain kondisi lane yang sedang dilewati bot, kondisi bot miliki kita sendiri, dan kondisi bot milik lawan. Pada implementasi program yang kami buat, kami

mengambil beberapa alternatif solusi algoritma greedy yang digunakan untuk menangai skema-skema yang ada.

Untuk menangani kondisi lane yang sedang dan akan dilewati, kami mengkombinasikan algoritma greedy by lane, greedy by speed, dan greedy by damage. Greedy by lane akan mencari lane terbaik yang bisa dilewati oleh bot sehingga efisien. Dengan adanya algoritma by lane, algoritma Greedy by speed akan menambah kecepatan secara signifikan yang bisa digunakan bot namun tetap memperhatikan lane yang ada sehingga tidak sembarangan melakukan accelerate. Greedy by damage juga menambah efisiensi agar bot tidak mendapatkan damage yang terlalu besar selama permainan.

Untuk menangani kondisi bot baik milik kita ataupun lawan, semua greedy by powerups dapat kita manfaatkan. Dengan adanya algoritma greedy by powerup (Boost, Tweet, Oil, EMP) ini akan menambah kemampuan dari bot yang kita miliki. Misalnya jika bot sedang tidak mempunyai powerups apapun, bot akan berjalan algoritma greedy sebelumnya. Namun, jika bot memiliki suatu powerups, situasinya akan berbeda. Greedy by Powerups – boost akan menangani jika bot mendapat boost dan akan menggunakannya sebaik mungkin. Greedy by Powerups - Oil, EMP dapat digunakan sesuai dengan kondisi dan posisi bot lawan. Misalnya jika posisi lawan di depan, dapat memanfaatkan EMP, sebaliknya jika posisi lawan di belakang, dapat memanfaatkan Oil. Greedy by Powerups – Tweet juga digunakan untuk menangani bot lawan bahkan memungkinkan untuk membalikkan keadaan.

Algoritma greedy by time tidak kami gunakan dengan beberapa pertimbangan yaitu tidak memperhatikan kondisi bot dan lawan yang bisa berubah-ubah. Selain itu, kombinasi dari beberapa algoritma lainnya diharapkan sudah cukup meng-cover untuk mendapat waktu terbaik. Kombinasi dari beberapa algoritma greedy yang telah dijelaskan diharapkan mendapatkan hasil terbaik dan dapat memenangkan pertandingan.

Bab 4 : Implementasi dan Pengujian

4.1 Implementasi Strategi Greedy pada program Bot yang digunakan

Berikut implementasi strategi greedy dalam metode run pada kelas Bot.

```
public String run() {
    // metode run mengembalikan string Command yang dipilih berdasarkan strategi greedy
    // start: indeks block Car sekarang
    // end: indeks block terakhir yang terlihat
    // curLane: indeks lane Car sekarang
    this.start = gameState.player.position.block-gameState.lanes.get(0)[0].position.block;
    this.end = gameState.lanes.get(0).length-1;
    int curLane = gameState.player.position.lane -1;
    // prekomputasi variabel prefix, ctDamage, dan ctWall. Secara berurut berisi informasi
    // prefix jumlah powerups, jumlah Damage, dan jumlah Wall ditambah cybertruck yang ada pada
lintasan.
    precompute();
    // now berisi informasi Car sekarang, berupa speed, damage, jumlah setiap powerups,
boostcounter
    // dan boosting.
    Result now = new Result(gameState);

    // Jika damage>=5, pilih Fix
    if(now.damage>=5) {
        return "FIX";
    }
    // update kondisi boost
    updateBoostcounter(now);
    // digunakan metode greaterThanV6 sebagai fungsi seleksi utama
    // juga sebagai penentu kriteria optimum lokal
    Comparator<Command> comp = (Command c1, Command c2) -> c1.resi.greaterThanV6(c2.resi);
    // versi greaterThan dapat diganti untuk mencoba kriteria yang lain

    List<Command> commands = new ArrayList<>();
    // untuk setiap command dari 8 command non-attack, metode run mengembalikan boolean
    // fungsi seleksi dan kelayakan khusus setiap command. Bila memenuhi, disimpan atribut
    // resi bertipe Result berisi informasi Car setelah command tersebut dijalankan
    BoostCommand bc = new BoostCommand();
    if(bc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(bc);
    }
    AccelerateCommand ac = new AccelerateCommand();
    if(ac.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(ac);
    }
    NothingCommand nc = new NothingCommand();
    if(nc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(nc);
    }
    LizardCommand lc = new LizardCommand();
    if(lc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall,gameState)) {
        commands.add(lc);
    }
    TurnLeftCommand tlc = new TurnLeftCommand();
    if(tlc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(tlc);
    }
    TurnRightCommand trc = new TurnRightCommand();
    if(trc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(trc);
    }
    DecelerateCommand dc = new DecelerateCommand();
```

```
    if(dc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(dc);
    }
    FixCommand fc = new FixCommand();
    if(fc.run(now,start,curLane,truck,end,prefix,ctDamage,ctWall)) {
        commands.add(fc);
    }
    // Command yang memenuhi diurutkan untuk mencari optimum lokal
    Collections.sort(commands,comp);
    bestCom = commands.get(0).render();
    // Bila command yang terpilih berefek sama dengan nothing, yaitu command nothing atau
    // command accelerate dengan kondisi Car sedang speed maksimum, lanjutkan dengan
    // attackStrategy menggunakan 3 command attack.
    boolean attack = bestCom.equals("NOTHING");
    if(bestCom.equals("ACCELERATE")) {
        if(min(nextSpeed(now.speed),maxSpeedIfDamage[now.damage])==now.speed) attack=true;
    }
    if(attack) {
        attackStrategy(now);
    }
    return bestCom;
}
```

Metode greaterThanV6 sebagai fungsi seleksi utama pada strategi greedy diimplementasikan sebagai berikut dalam Result.java.

```
public int greaterThanV6(Result res) {
    // menentukan prioritas dari dua Result, yaitu this dan res.
    // bila this diprioritaskan, kembalikan bilangan negatif.
    // bila res diprioritaskan, kembalikan bilangan positif.
    // bila sama, kembalikan 0.

    // bila damage tidak sama, prioritaskan damage yang lebih kecil
    if(this.damage!=res.damage) {
        return this.damage-res.damage;
    }
    // bila speed tidak sama, prioritaskan speed yang lebih besar
    if(this.speed!=res.speed) {
        return res.speed-this.speed;
    }
    // bila jarak yang ditempuh berbeda setidaknya 2 blok, prioritaskan yang lebih jauh
    if(abs(res.xr-this.xr)>1) {
        return res.xr-this.xr;
    }
    // bila powerups boost yang dimiliki tidak sama, prioritaskan yang lebih banyak
    if(res.ctBoost!=this.ctBoost) {
        return res.ctBoost-this.ctBoost;
    }
    // bila powerups lizard yang dimiliki tidak sama, prioritaskan yang lebih banyak
    if(res.ctLizard!=this.ctLizard) {
        return res.ctLizard-this.ctLizard;
    }
    // prioritaskan yang memiliki powerups attack lebih banyak
    // Emp dibobot 4, Tweet dibobot 3, Oil dibobot 2.
    return (4*res.ctEmp+3*res.ctTweet+2*res.ctOil)-(4*this.ctEmp+3*this.ctTweet+2*this.ctOil);
}
```

Fungsi seleksi dan kelayakan khusus untuk setiap command non-attack diimplementasikan sebagai kondisional pada metode run setiap command.

- BoostCommand.java

```
    // cek stok powerups. Bila speed sudah max jangan gunakan boost
    if(res.ctBoost>0 && res.speed!=maxSpeedIfDamage[res.damage]) {
```

- AccelerateCommand.java

```
    // Bila selanjutnya masih boosting, jangan gunakan accelerate
    if(!res.boosting) {
```

- NothingCommand.java

```
    // bila speed=0, jangan Nothing
    if(res.speed>0) {
```

- LizardCommand.java

```
    // cek stok Lizard
    if(res.ctLizard>0) {
```

- TurnLeftCommand.java

```
    // bila tidak bisa ke kiri atau speed=0, jangan turn left
    if(y>0 && res.speed>0) {
```

- TurnRightCommand.java

```
    // Bila tidak bisa ke kanan atau speed=0, jangan turn right
    if(y+1<ctLane && res.speed>0) {
```

- DecelerateCommand.java

```
    // menghindari Car berhenti
    if(res.speed>3) {
       ...
        // Pada greedy by damage decelerate kandidat kuat, namun
        // sering merugikan. decelerate layak dipertimbangkan bila lane
        // sebelah bebas dari rintangan
        boolean nextLaneBetter = false;
        if(yi+1<ctLane) {
            if(ctDamage[yi+1][end]-ctDamage[yi+1][xi-1]==0) {
                nextLaneBetter = true;
            }
        }
        if(yi-1>=0) {
            if(ctDamage[yi-1][end]-ctDamage[yi-1][xi-1]==0) {
                nextLaneBetter = true;
            }
        }
        if(!nextLaneBetter) {
            return false;
        }
       ...
}
```

- FixCommand.java

```
    // Bila damage=0, Fix tidak dipertimbangkan
    if(res.damage>0) {
```

Bila command hasil strategi sebelumnya menghasilkan command nothing atau accelerate yang tidak berefek, dilanjutkan strategi attack menggunakan powerups yang diimplementasikan pada metode AttackStrategy di Bot.java sebagai berikut.

```
private void attackStrategy(Result cur) {
    // mengubah bestCom ke command attack sesuai strategi
    int myX = gameState.player.position.block;
    int myY = gameState.player.position.lane-1;
    int otherX = gameState.opponent.position.block;
    int otherY = gameState.opponent.position.lane-1;
    int otherSpeed = gameState.opponent.speed;
    if(myX>otherX) { // kasus Car di depan lawan
        // ketiga commmand attack memiliki metode run yang mengubah bestCom
        // pada bot menjadi command attack bila suatu kondisi terpenuhi

        // oil digunakan bila lawan berada di lane yang sama dan perbedaan jarak
        // kurang dari kecepatannya. Kecepatan normal lawan di lane yang sama akan
        // membuatnya hit_oil ronde berikutnya
        OilCommand oc = new OilCommand();
        oc.run(cur,myX,myY,otherX,otherY,otherSpeed,this);
        // tweet digunakan bila kondisi oil tadi tidak memenuhi
        // tweet ditempatkan di dekat Car bila area sekitar ramai rintangan untuk
        // mempersulit musuh
        // Selain itu, tweet ditempatkan di sekitar otherX+2*otherSpeed sebagai
        // rintangan ronde berikutnya
        TweetCommand tc = new TweetCommand();
        tc.run(cur,myX,myY,otherX,otherY,otherSpeed,this,gameState.lanes,ctDamage);
        if(bestCom.equals("NOTHING") || bestCom.equals("ACCELERATE")) {
            // Memanfaatkan oil agar terpakai maksimal
            if(cur.ctOil*15+myX>=track_length) {
                bestCom = "USE_OIL";
            }
        }
    } else if(myX<otherX){ // Kasus Car di belakang lawan
        // Emp digunakan saat dapat membalap lawan
        // saat emp > 2 dan kecepatan lawan 15 agar kembali ke 3
        // saat emp berlimpah sehingga bisa jadi tidak habis di akhir game
        EmpCommand ec = new EmpCommand();
        ec.run(cur,myX,myY,otherX,otherY,otherSpeed,this);
    }
}
```

4.2 Penjelasan Struktur Data yang digunakan dalam program

Struktur data dari starter bot ini dibagi menjadi tiga folder yang berisi class/.java dan beberapa class diluarnya. Tiga folder tersebut adalah folder command yang berisi perintah-perintah yang dapat dilakukan oleh bot, folder entities yang berisi objek-objek yang ada di dalam permainan, dan folder enums yang berisi enumerator yang digunakan di bot. Lalu file java di luar folder ada Bot, Main, Result, dan util.

4.2.1 Folder command

Folder command adalah folder yang berisi perintah-perintah yang dapat dilakukan oleh bot, seperti ACCELERATE, FIX, USE_OIL, dll. Command-command tersebut memiliki parent class Command yang digunakan untuk pertimbangan tiap command. Class command non-attack ada delapan, dan di setiap class tersebut terdapat fungsi seleksi khusus kelayakan penggunaanya pada metode run. Class-class tersebut adalah AccelerateCommand, BoostCommand, DecelerateCommand, FixCommand,

LizardCommand, NothingCommand, TurnLeftCommand, dan TurnRightCommand. Selain delapan class non-attack tersebut, ada tiga class command attack. Perhitungan penggunaan ketiga class tersebut ada di metode AttackStrategy di bot.java. Ketiga class tersebut adalah EmpCommand, OilCommand, dan TweetCommand.

4.2.2 Folder entities

Folder entities adalah folder yang berisi objek-objek yang ada di dalam permainan. Terdapat 4 class di dalam folder ini, Car, GameState, Lane, dan Position.

4.2.2.1 Car

Class Car adalah class object mobil yang ada dalam permainan. Class ini digunakan untuk mobil player dan mobil musuh. Terdapat beberapa informasi yang disimpan di class ini, yaitu id, position, speed, state, damage, powerups, boosting, dan boostCounter. Informasi score sengaja diabaikan.

4.2.2.2 GameState

Class GameState menyipan informasi tentang permainan yang sedang dimainkan, seperti nomor ronde dan world map. Variabel yang disimpan di dalam class ini adalah currentRound, maxRound, player yaitu mobil player, opponent yaitu mobil musuh, dan lanes adalah peta yang disimpan informasinya tiap baris.

4.2.2.3 Lane

Class Lane adalah Class peta permainan yang dibagi tiap barisnya. Class ini menggunakan class Position dan class terrain. Selain itu, class ini juga memiliki variabel occupiedByPlayerId dan isOccupiedByCuberTruck.

4.2.2.4 Position

Class Position adalah class yang menyimpan koordinat dari suatu object, x dengan nama block dan y dengan nama lane.

4.2.3 Folder enums

Folder enums adalah folder yang enumerator yang digunakan di bot ini. Terdapat tiga class di folder enums, Direction, PowerUps, State, dan Terrain.

4.2.3.1 PowerUps

Class PowerUps adalah class yang menyimpan tipe data PowerUps. Tipe data ini digunakan untuk menyimpan power up yang dimiliki bot player dan musuh. Constant yang ada di tipe data ini adalah BOOST, OIL, LIZARD, TWEET, dan EMP.

4.2.3.2 State

Class State adalah class yang menyimpan tipe data State. State ini adalah state mobil setelah ada input. Misal setelah diberikan command “ACCELERATE”, maka State nya akan menjadi ACCELERATING. Jika mobil menabrak dinding, maka State nya akan menjadi HIT_WALL. Constant yang ada di tipe data ini adalah ACCELERATING, READY, NOTHING, TURNING_RIGHT, TURNING_LEFT, HIT_MUD, HIT_OIL, HIT_WALL, HIT_CYBER_TRUCK, HIT_EMP, DECELERATING, PICKED_UP_POWERUP, USED_BOOST, USED_OIL, USED_LIZARD, USED_TWEET, USED_EMP, dan FINISHED.

4.2.3.3 Terrain

Class Terrain adalah class yang menyimpan tipe data dari sebuah lane. Class ini akan mengecek apakah pada lane dan block tertentu berupa jalan bertipe Empty, Mud, Oil_Spill, ataupun Finish, juga bisa berupa powerups bertipe Oil_Power, Boost, Wall, Lizard, Tweet, ataupun EMP.

4.2.4 Bot

Class Bot adalah kelas untuk mengembangkan strategi Bot. Objek kelas ini dibuat setiap ronde baru dengan parameter random dan GameState. Kelas Bot memiliki metode run untuk mengembalikan command yang dipilih berdasarkan strategi. Dari atribut gameState yang disimpan, informasi diambil dan disimpan ke atribut lain. Informasi prefix powerups disimpan di prefix, informasi prefix damage disimpan di ctDamage, informasi prefix wall disimpan di ctWall, informasi truk disimpan di truck. Informasi indeks Car terhadap block disimpan di start, end adalah indeks terakhir dari block yang terlihat, ctLane adalah banyak lane. Kelas ini juga memiliki metode AttackStrategy yang dipanggil di run dan metode precompute untuk mengisi prefix,ctDamage, dan ctWall.

4.2.5 Main

Kelas main adalah program utama yang pada setiap rondenya membaca file state.json dan menuliskan command yang dipilih. Class ini akan memanggil method run dari class bot untuk mendapatkan command yang dipilih.

4.2.6 Result

Kelas ini merupakan versi lebih sederhana dan ringkas dari GameState. Kelas ini dapat meringkas informasi blok arena dan Car. Atribut yang disimpan dan dioperasikan adalah speed, damage, 5 powerups (ctBoost, ctLizard, ctEmp, ctTweet, ctOil), boosting, dan boostcounter. Kelas ini dapat diinisialisasi dari Terrain dan GameState. Kelas ini memiliki metode add dan minus untuk penjumlahan dan pengurangan dengan sesamanya. Kelas ini juga memiliki metode greaterThanV6 yang digunakan untuk membandingkan Result yang satu dengan yang lain dan menjadi komparator dalam mencari optimum lokal.

4.2.7 Util

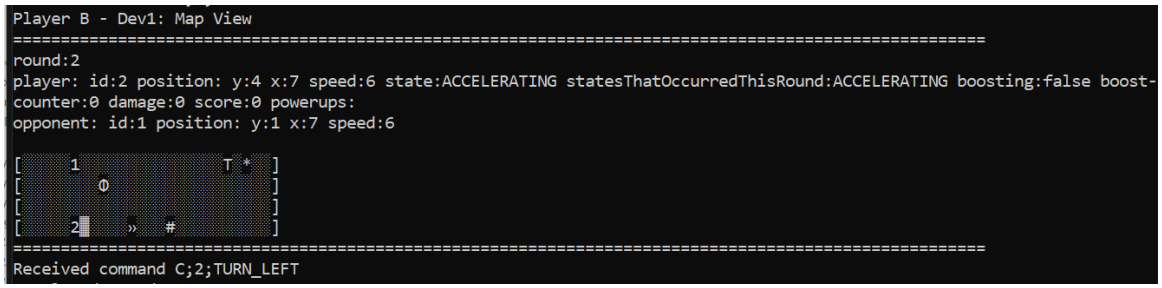
Kelas Util adalah kelas yang digunakan untuk membantu perhitungan dalam bot ini. Kelas ini menyimpan konstanta seperti track_length (1500) dan maxSpeedIfDamage. maxSpeedIfDamage[i] adalah kecepatan maksimum Car bila damagenya i (i dari 0 sampai 5). Kelas ini memiliki 4 metode, yaitu updateBoostCounter, nextSpeed, prevSpeed, dan updateXiYi. updateBoostCounter adalah prosedur yang menerima objek Result dan mengurangi 1 boostcounter dan memperbarui boosting. nextSpeed dan prevSpeed adalah fungsi yang menerima int x dan mengembalikan speed sebelum atau setelah x menurut aturan decelerate/accelerate. updateXiYi adalah fungsi yang menerima parameter x, xi, yi, resi, dan informasi lainnya untuk memperbarui resi (bertipe Result) dari segala hal yang ditemui di lane yi dari block x+1 hingga xi.

4.3 Analisis dan Pengujian

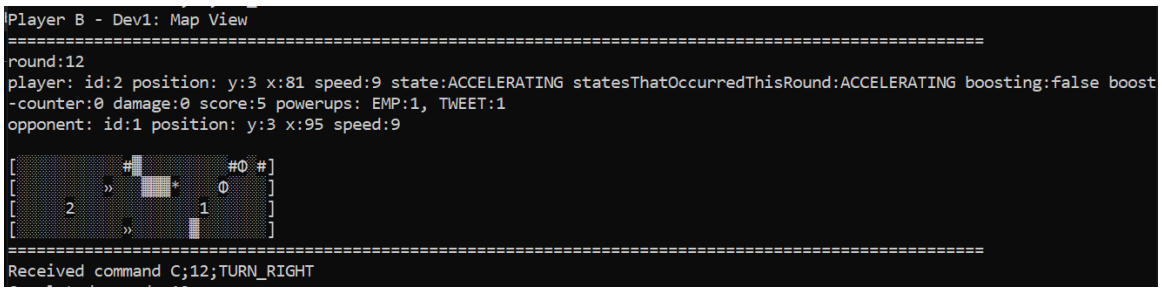
Setelah implementasi, Bot diuji dengan reference-bot dan bot lain buatan sendiri. Ketika melawan reference-bot, Bot berhasil finish dalam 120-130 ronde. Sedangkan ketika melawan bot lain dapat mencapai 150-180 ronde. Hal ini karena kedua bot saling serang, baik dengan oil, tweet, maupun Emp.

Dari hasil 120-130 ronde, didapat kecepatan rata-rata Bot adalah 1500/125= 12 yang mendekati kecepatan maksimum, yaitu 15. Dan hanya berbeda 20-30 ronde dari banyak ronde minimal 100.

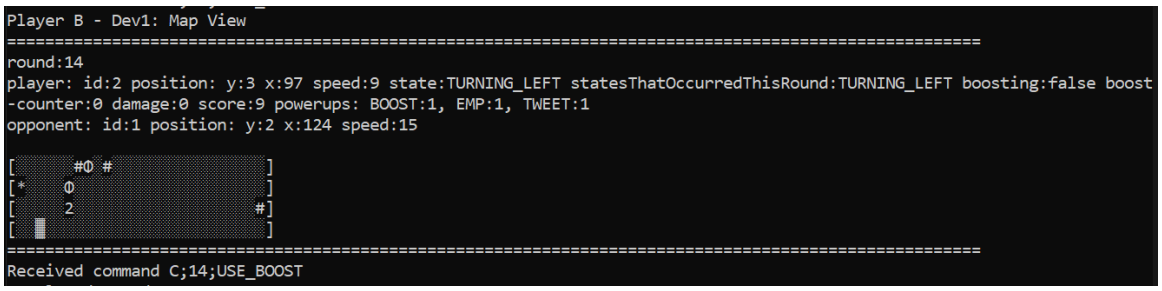
Berikut beberapa perilaku Bot dan analisisnya.



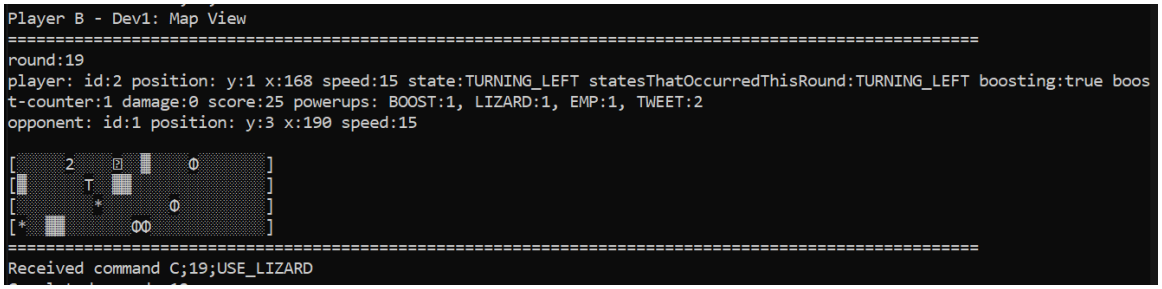
Bot menghindari Mud walaupun ada powerups boost dibelakangnya. Hal ini karena damage lebih diprioritaskan. Hal ini juga menyebabkan perilaku bot untuk mem-fix setiap terkena damage.



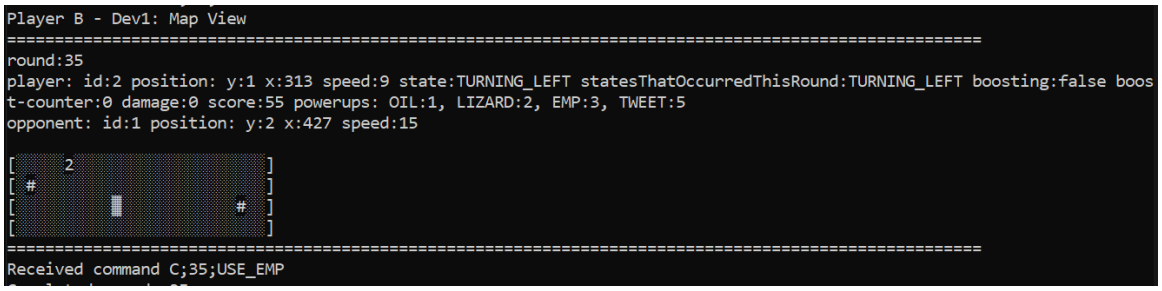
Bot memilih mengambil boost di kanan walau jadi terlambat 1 block. Hal ini karena ketika kedua command memiliki efek damage, speed yang sama serta beda blok hanya satu. Bot akan memilih powerups boost yang lebih banyak.



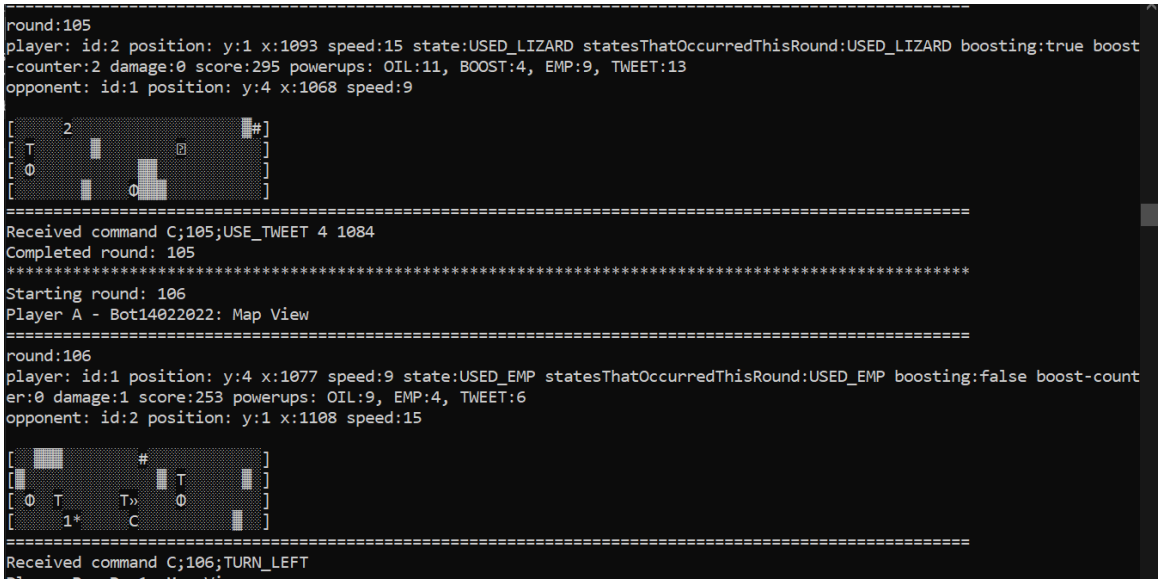
Bot menggunakan boost untuk speed yang lebih besar. Dengan meminimumkan damage, bot dapat memakai boost hingga ronde 19.



Bot menggunakan lizard untuk menghindari mud di kedua lane. Bot berhasil meminimumkan damage.

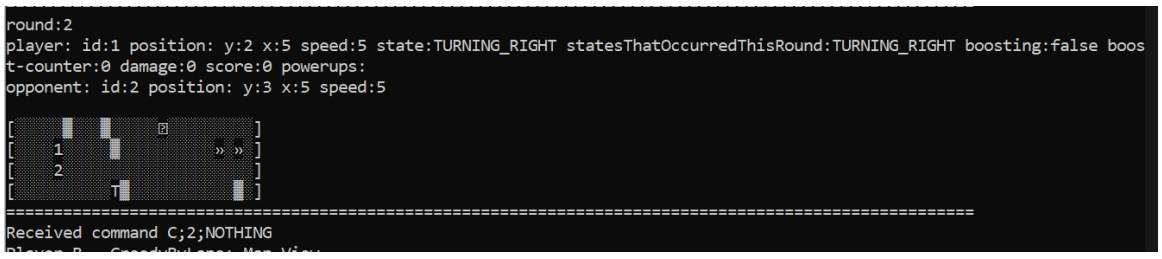


Bot menjalankan AttackStrategy dan menggunakan EMP. Pada ronde 35, command non-attack yang terpilih tidak memberi efek apapun sehingga dilanjutkan dengan strategi attack. Karena bot tertinggal (posisi bot 313, lawan 427) dan kecepatan lawan 15 serta emp ada 3 (lebih dari 2), digunakanlah emp. Lawan akan freeze dan kecepatan kembali ke 3 yang sangat merugikan.



Bot menggunakan AttackStrategy tweet. Kecepatan lawan 9, bot menaruh truk di posisi lawan + 2*kecepatan lawan -2= 1068+2*9-2=1084. Namun bot lawan dapat menghindar ke kiri karena kosong. Hal ini tidak bisa dihindari, namun dari banyak percobaaan akan muncul kesempatan truk memaksa lawan terjebak dan terkena damage.

Bot menggunakan AttackStrategi oil. Dengan oil ini lawan akan terkena di ronde saat ini juga.



Bot memilih nothing lalu turn right, namun lebih optimal memilih turn right lalu accelerate. Hal ini karena bot tidak mempertimbangkan langkah kedua. Kekurangan ini dapat diperbaiki sebagian dengan mempertimbangkan sejumlah blok akhir yang terlihat namun belum sempat diimplementasi.

Kesimpulannya perilaku bot sudah sesuai dengan kode. Adapun performa Bot sudah baik dan sudah mendekati optimum walaupun belum mencapai optimum. Bot dapat

menang terutama dalam konfigurasi 10 blok di awal banyak rintangan dan di 10 blok di akhir banyak powerups. Sedangkan kekalahan bot biasa disebabkan oleh konfigurasi 10 blok di awal banyak powerups tetapi 10 blok di akhir banyak rintangan karena sulit dideteksi.

Bab 5 : Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan penjelasan yang telah dijelaskan pada bab-bab sebelumnya, kami berhasil membuat sebuah bot dalam permainan Overdrive yang memanfaatkan algoritma *greedy*. Implementasi dari beberapa algoritma *greedy* yang dikombinasikan terbukti berhasil memenangkan pertandingan melawan bot lain sebagai perbandingannya.

5.2 Saran

Saran-saran yang dapat diberikan pada Tugas Besar IF2211 Strategi Algoritma Semester 2 2020/2021 ini antara lain :

1. Algoritma yang digunakan pada tugas besar ini masih memiliki kekurangan sehingga diharapkan selanjutnya untuk dilakukan efisensi dari kinerja bot yang dibuat. Algoritma juga sebaiknya dibuat dengan *comment* yang lebih *well-commented* agar orang-orang yang membaca algoritma program yang dibuat dapat memahami dengan baik.

2. Pesan untuk diri kami sendiri khususnya, dan semua mahasiswa pada umumnya, agar mengurangi sifat menunda-nunda pekerjaan karena akan sangat menyulitkan saat mendekati *deadline*.

Link Github Program

https://github.com/ARomygithub/Tubes1_Race-Algo/tree/main

Daftar Pustaka

Munir, R. 2022. *Algoritma Greedy (2022) Bag 1* . Slide kuliah IF2211 Strategi Algoritma, Institut Teknologi Bandung

https://creatormedia.my.id/pengertian-algoritma-greedy/ (*diakses pada 15 Februari 2022 Pukul 22.00)*

https://github.com/EntelectChallenge/2020-Overdrive *(terakhir diakses 18 Februari 2022 pukul 20.00)*

https://www.javaprogramto.com/2020/04/java-collection-sort-custom-sorting.html *(terakhir diakses 17 Februari Pukul 20.00)*