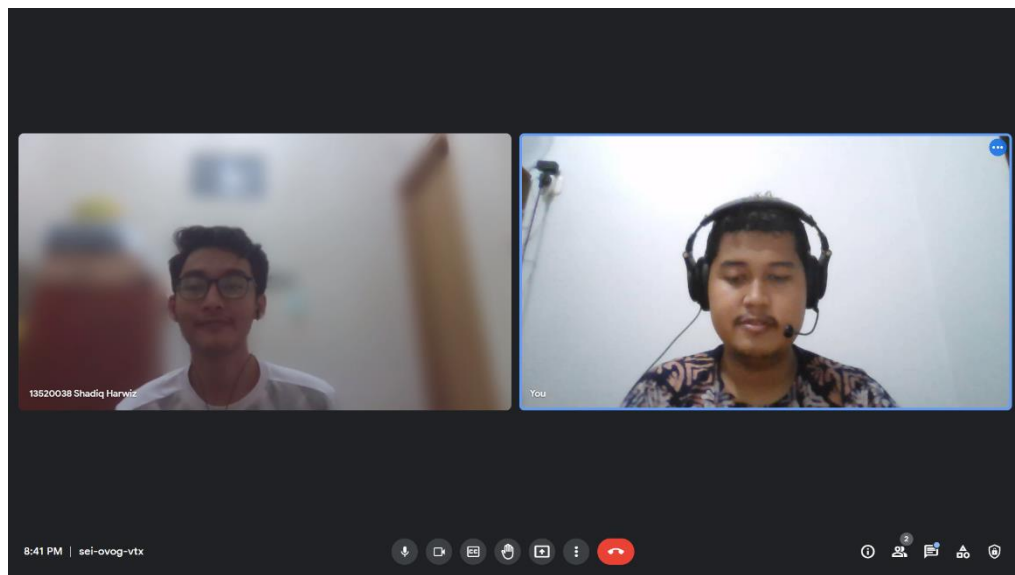


Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Overdrive”

LAPORAN TUGAS BESAR

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma
Semester II Tahun 2021/2022



Disusun oleh

Shadiq Harwiz (13520038)

Andika Naufal Hilmy (13520098)

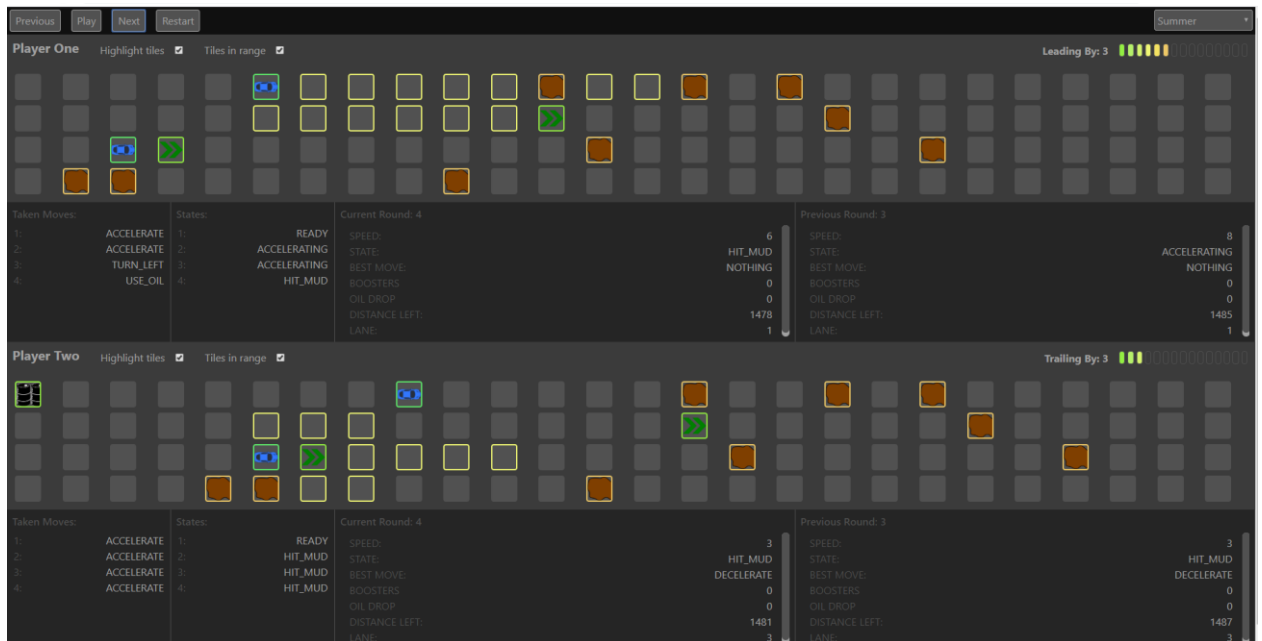
**TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG**

2021

BAB I

DESKRIPSI TUGAS

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



Gambar 1. Contoh tampilan permainan Overdriver

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* untuk mengimplementasikan permainan *Overdrive*. *Game engine* dapat diperoleh pada laman berikut: <https://github.com/EntelectChallenge/2020-Overdrive>. Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan *Overdrive* dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter bot* di dalam *starter pack* pada laman berikut ini: <https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>

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 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 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
 - 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.

Adapun peraturan yang lebih lengkap dari permainan Overdriver, dapat dilihat pada laman: <https://github.com/EntelectChallenge/2020-Overdrive/blob/master/game-engine/game-rules.md#the-car>

BAB II

LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma greedy adalah algoritma yang digunakan untuk membentuk solusi langkah perlangkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya.

2.1.1 Prinsip Utama Algoritma Greedy

Prinsip utama algoritma greedy adalah “*take what you can get now*”. Maksud dari prinsip tersebut adalah pada setiap langkah dalam algoritma greedy, diambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya. Solusi tersebut disebut dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.

2.1.2 Elemen Algoritma Greedy

Elemen-elemen yang digunakan dalam penerapan algoritma greedy 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

2.2 Game Engine

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

2.3 Game Engine Entellect Overdriver

Dalam *game* Entellect 2020 – Overdriver terdapat beberapa komponen yang diperlukan agar game dapat berjalan:

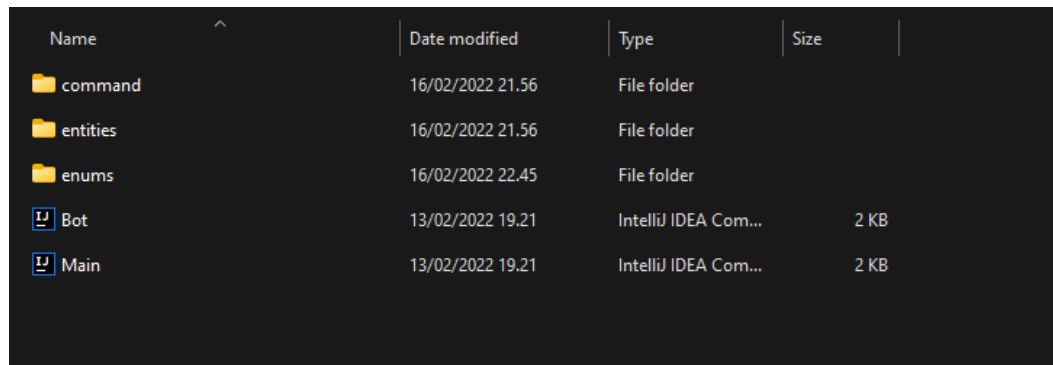
1. *Game engine interface* : *Interface*(antarmuka) yang digunakan *game runner* untuk dapat menjalankan atau mengembangkan *game* dengan *game engine* yang berbeda.
2. *Game engine* : *Game engine* bertanggung jawab untuk memaksa *bot* yang dibuat pemain agar mematuhi aturan yang didefinisikan. *Game engine* akan menyalurkan perintah yang dikirimkan oleh *bot* kedalam game untuk diproses jika perintah yang diberikan valid.
3. *Game runner* : *Game runner* bertanggung jawab untuk menjalankan pertandingan antarpemain. *Game runner* menerima perintah dari *bot* (dengan berbagai bahasa) dan mengirimkannya ke *game engine* untuk dieksekusi
4. *Reference bot* : *bot* yang disediakan dengan beberapa strategi agar dapat diujikan dengan *bot* yang akan dibuat. Tersedia dalam bahasa java.
5. *Starter bot* : *bot* awal dengan logika dasar yang bisa digunakan sebagai referensi dalam pengembangan *bot*.

2.3.1 Mulai Membangun Starter Bot Dalam Bahasa Java

Hal pertama yang harus dilakukan untuk membangun *bot* adalah melakukan *clone* terhadap program entellect starter-pack di (<https://github.com/EntellectChallenge/2020-Overdrive/releases/tag/2020.3.4>). Dalam membangun bot terdapat banyak bahasa pilihan yang dapat digunakan. Pada tugas besar ini akan dibangun sebuah *bot* dengan bahasa pemrograman Java. Pada *folder* Java terdapat tiga *folder* dan dua *file* awal yang disediakan, penjelasannya adalah sebagai berikut:

1. *Folder command*: *Folder* ini berisikan kelas dari perintah yang dapat digunakan oleh pemain. Perintah yang disediakan adalah *Accelerate*, *Boost*, *ChangeLane*, *Declerate*, *DoNothing*, *Emp*, *Fix*, dan *Oil*. Semua kelas tersebut mengimplementasikan interface *command*.
2. *Folder entities*: *Folder* ini berisikan kelas entitas (makhluk) yang ada pada game, contohnya adalah *Car*, *GameState*, *Lane*, *Position*, dan lain-lain.
3. *Folder enums*: *Folder* ini berisikan entitas yang memiliki nilai berupa konstanta. Contohnya adalah *Direction*, *PowerUps*, *State*, *Terrain*, dan lain-lain.

4. Bot.Java: *File* ini berisikan logika dari *bot* yang dibangun. Pada *file* ini akan diimplementasikan strategi greedy dengan memanfaatkan kelas lain yang ada. Semua strategi akan ditulis pada *method* `run()` di kelas *bot*.
5. Main.Java: *File* ini adalah *file* yang bertanggung jawab untuk menjalankan Bot.Java dengan menginstansiasi *bot* pada setiap ronde dan menerima perintah dari bot tersebut untuk dikirimkan ke game engine untuk kemudian diproses.



Name	Date modified	Type	Size
command	16/02/2022 21.56	File folder	
entities	16/02/2022 21.56	File folder	
enums	16/02/2022 22.45	File folder	
Bot	13/02/2022 19.21	IntelliJ IDEA Com...	2 KB
Main	13/02/2022 19.21	IntelliJ IDEA Com...	2 KB

Gambar 2. Struktur *Folder* Java

Untuk dapat menjalankan bot nya maka dibutuhkan beberapa aplikasi yang harus di-*install*, yaitu Java JDK 8, *code editor* IntelliJ IDEA, dan NodeJs. Untuk mendapatkan *file executable* java (.jar), *file* yang disediakan harus di-*build* menggunakan IntelliJ IDEA. Setelah itu data lokasi “player-a” diubah menjadi “./starter-bots/java” pada *file* `game-runner-config.json`. Setelah itu, jalankan `run.bat` untuk menjalankan *game-engine*. Berikut ini adalah informasi mengenai beberapa kegunaan *file* konfigurasi yang ada di game:

1. `game-runner-config.json`: Letaknya ada di dalam *folder* starter pack, digunakan untuk mengatur lokasi *file bot* pemain satu, pemain dua, *file* hasil pertandingan, konfigurasi *game*, dan *game engine*.
2. `game-config.json`: Letaknya ada di dalam *folder* starter-pack, digunakan untuk mengatur konfigurasi dasar dari game Overdriver seperti atribut dari setiap jenis mobil, akumulasi poin dari setiap perintah dalam game, dan lain-lain.
3. `bot.json` : Letaknya ada di dalam *folder* java dan digunakan untuk mengganti nama *bot* dan lokasi *executable* (.jar) dari bot yang dibuat.

Untuk dapat menjalankan *visualizer*, harus mendownload *visualizernya* di <https://github.com/Affuta/overdrive-round-runner>. Setelah menjalankan pertandingan melalui cmd maka hasilnya secara *default* akan tersimpan di *folder* `match-logs`.

Kemudian *folder* pertandingan yang ingin divisualisasikan harus dipindahkan ke dalam *folder* `matches` di *visualizer*. *Folder* pertandingan memiliki format nama seperti *timestamp* pertandingannya sehingga hanya perlu mengingat kapan pertandingan tersebut dijalankan untuk mendapatkan *folder* yang tepat. Setelah memindahkan *folder* pertandingan ke *visualizer*, jalankan aplikasi *visualizer* dan pertandingan sudah siap ditonton dengan menggunakan tampilan GUI

2.3.2 Mengembangkan Starter Bot

Meskipun *starter bot* sudah menyediakan berbagai macam kelas beserta *method* dan *attributenya*, kelas tersebut masih bisa disempurnakan lagi dan ditambah jumlahnya. Contohnya starter-bot sudah menyediakan kelas untuk perintah *Accelerate*, *Boost*, *ChangeLane*, *Declerate*, *DoNothing*, *Emp*, *Fix*, dan *Oil*, akan tetapi belum menyediakan perintah untuk *Lizard* dan *Tweet*. Selain kelas *command*, kelas *entity* dan yang lainnya juga masih bisa ditambahkan lagi. Dalam membuat kelas baru, *method* dan *attribute* yang dibutuhkan dapat ditambahkan ke kelas tersebut. Perhatikan bahwa *method* render perlu dituliskan supaya *command* bisa dibaca oleh *game engine* melalui *main.java*.

Dalam pertukaran data dengan *game engine*, Overdriver memanfaatkan *file json* yaitu *state.json*. Penjelasan tentang *state* yang disediakan, namanya, dan isinya dapat dibaca di <https://github.com/EntelectChallenge/2020-Overdrive/blob/master/game-engine/state-files.md>

Untuk menggunakan *file* dengan ekstensi *.json* di Java maka diperlukan modul *json* yang harus di-*import*. Sintaks penggunaannya adalah sebagai berikut:

```
com.google.gson.annotations.SerializedName;  
@SerializedName(<nama_value_di_file_state.json>  
    <keyword_modifier> <type_varibel> <nama_variabel>;  
Contohnya  
com.google.gson.annotations.SerializedName;  
@SerializedName("maxRounds")  
    public int maxRounds;
```

Setelah mempersiapkan semua kelas dan elemen baru yang diperlukan, langkah selanjutnya adalah memasukkan strategi greedy yang diinginkan pada *file bot.java*. *Starter bot* sudah menyediakan beberapa *method* dan *attribute* awal yang bisa digunakan dalam pengembangan *bot*. *Method* dan *attribute* yang disediakan juga bisa diubah dan ditambah lagi untuk melengkapi strategi greedy. Semua strategi greedy yang ditulis akan diimplementasikan pada *method run* dan pastikan *method run* harus memberikan *return* sebuah perintah yang valid agar dideteksi oleh *game engine*. Manfaatkan fitur OOP yang disediakan Java untuk membuat strategi greedy. Isi dari *bot.java* adalah sebagai berikut:


```

package za.co.entelect.challenge;

import za.co.entelect.challenge.command.*;
import za.co.entelect.challenge.entities.*;
import java.util.*;
import java.util.stream.Collectors;
// Import semua hal yang dibutuhkan

public class Bot {
    private Random random;
    private GameState gameState;
    private Opponent opponent;
    // Attribute bawaan dan tambahan

    public class A{ Nested Class baru }
    // Nested Class jika dibutuhkan

    private Direction resolveDirection(Position a, Position b){ Isi method }
    private int euclideanDistance(int aX, int aY, int bX, int bY) {.....}
    private List<CellandFreezeCount> getFreezedLocation() {
        // Method bawaan dan tambahan sendiri untuk membuat strategi greedy
    }

    public Command run() {Strategi Greedy diletakkan disini}
}

```

BAB III

PEMANFAATAN STRATEGI GREEDY

3.1 Solusi Algoritma Greedy yang Mungkin Dipilih

3.1.1. *Greedy by Obstacles*

Greedy by Obstacles adalah strategi untuk menghindari berbagai *block* yang muncul pada *map* dan berada di depan posisi mobil. Adapun *block* yang dimaksud adalah *block* yang jika dilewati oleh mobil akan memperlambat mobil atau hal lain yang dapat merugikan mobil selama belum mencapai garis *finish*. *Block* yang akan dihindari ini adalah salah satu dari jenis *Mud*, *Oil Spill*, *Opponent*, atau *Wall*. Apabila tidak ada hal tersebut di depan posisi si mobil, maka greedy ini tidak perlu dilakukan.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat : Semua *block* di depan posisi mobil yang dapat dilewati atau tidak ada *block* yang dipilih.
- Himpunan Solusi : Salah satu *block* disekitar mobil yang dapat dilewati atau tidak ada *block* yang dipilih.
- Fungsi Solusi : Memeriksa apakah *block* yang dipilih bagian dari *obstacles* atau tidak.
- Fungsi Seleksi : Memilih *block* di depan posisi mobil yang dapat dilewati dan tidak merupakan bagian dari *obstacles*. Apabila tidak ada *block* yang bisa dipilih, maka mobil akan tetap bergerak tepat ke depan.
- Fungsi Kelayakan : Mengevaluasi apakah mobil perlu untuk menghindari *block* yang mengandung *obstacles*. Apabila tidak abaikan *block* yang dipilih.
- Fungsi Objektif : Meminimalkan resiko yang terjadi pada mobil saat mengambil *block* yang akan dilewati mobil nantinya.

b. Analisis Efisiensi Solusi

Seleksi yang dilakukan adalah mengecek setiap *block* di peta yang berada di depan mobil ($4 \times n$), apabila di antara *block* tersebut ada yang mengandung *obstacles*, maka diambil *block* disekitar mobil yang terdekat dan mengandung *obstacle* (4), lalu dicek apakah mobil akan mengurangi kecepatannya atau mobil akan mengambil *block* yang akan dilewatinya (1).

$$T(n) = 16n = O(n)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Hampir efektif selama mobil belum mencapai *finish* karena *blocks* yang mengandung *obstacles* dapat dihindari dan *blocks* yang akan dilewati nantinya tidak memberikan kerugian pada mobil.

Strategi ini tidak efektif apabila:

- Bot lawan sangat pintar sehingga dapat membuat mobil terpaksa memilih salah satu *block* yang memiliki resiko minim

3.1.2. Greedy by Powerups

Greedy by Powerups adalah strategi untuk mengambil berbagai *block* yang muncul pada *map* dan berada di depan posisi mobil. Adapun *block* yang dimaksud adalah *block* yang jika dilewati oleh mobil akan memberikan keuntungan pada mobil sewaktu mobil akan menggunakannya dan mobil belum mencapai garis *finish*. *Block* yang akan diambil ini adalah salah satu dari jenis *Boost*, *Lizard*, *Tweet*, *EMP*, *Oil Item* atau *Wall*. Apabila tidak ada hal tersebut di depan posisi si mobil, maka greedy ini tidak perlu dilakukan.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat : Semua *block* di depan posisi mobil yang dapat dipilih atau tidak ada *block* yang dipilih.
- Himpunan Solusi : Salah satu *block* disekitar mobil yang dapat diambil atau tidak ada *block* yang dipilih.
- Fungsi Solusi : Memeriksa apakah *block* yang dipilih bagian dari *Powerups* atau tidak.
- Fungsi Seleksi : Memilih *block* di depan posisi mobil yang dapat dilewati dan merupakan bagian dari *Powerups*. Apabila tidak ada *block* yang bisa dipilih, maka mobil akan tetap bergerak tepat ke depan.
- Fungsi Kelayakan : Mengevaluasi apakah mobil perlu untuk melewati *block* yang mengandung *Powerups*. Apabila tidak abaikan *block* yang dipilih.
- Fungsi Objektif : Memaksimalkan keuntungan yang ada pada mobil saat mengambil *block* yang akan dilewati mobil nantinya.

b. Analisis Efisiensi Solusi

Seleksi yang dilakukan adalah mengecek setiap *block* di peta yang berada di depan mobil ($4 \cdot n$), apabila di antara *block* tersebut ada yang mengandung *Powerups*, maka diambil *block* disekitar mobil yang terdekat dan mengandung *Powerups* (4), lalu dicek apakah mobil akan mengambil *block* yang akan dilewatinya atau tidak (1).

$$T(n) = 16n = O(n)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Hampir efektif selama mobil belum mencapai *finish* karena *blocks* yang mengandung *Powerups* dapat diambil dan *blocks* yang akan dilewati nantinya akan memberikan keuntungan pada mobil.

Strategi ini tidak efektif apabila:

- Bot lawan sangat pintar sehingga dapat membuat mobil terkena jebakan saat melewati *block* yang telah dipilih.

3.2 Solusi Algoritma Greedy yang Dipilih dan Pertimbangannya

Pada *game Overdriver* ini terdapat beberapa solusi algoritma greedy yang bisa diimplementasikan dan digunakan tergantung pada kondisi mobil yang sedang aktif bergerak untuk setiap rondennya. Sehingga, diperlukan suatu pertimbangan kapan suatu solusi algoritma greedy harus dipilih. Jika dilihat dari perspektif yang lebih luas, terdapat beberapa skema inti yang harus dimiliki sebuah *bot*, antara lain skema untuk menangani *block* yang mengandung *obstacles* dan skema untuk memprioritaskan memilih *block* yang mengandung *Powerups*.

Pada implementasi program yang kami lakukan, kami hanya memilih satu alternatif solusi algoritma greedy untuk menangani salah satu skema diatas, yaitu menggunakan *greedy by obstacles* untuk menghindari mobil saat berada di dekat *block* yang mengandung *obstacles*. Kondisi ini harus ditangani karena mobil akan cepat rusak nantinya jika tidak dihindari dari *block-block* yang mengandung *obstacles*. Jika mobil telah rusak parah, akan terjadi dua kemungkinan. Yaitu mobil tidak dapat bergerak lagi atau mobil harus diperbaiki terlebih dahulu yang tentunya akan memakan waktu yang lama sehingga mobil lawan akan semakin mudah untuk bergerak dalam mencapai garis *finish*.

Kami tidak menggunakan algoritma *greedy by powerups* dengan beberapa pertimbangan, yaitu pada sewaktu mobil akan memilih suatu *block* yang mengandung *powerups*, mobil tidak tahu apakah selanjutnya terdapat *obstacles* atau perangkap yang telah diberikan oleh mobil lawan sehingga akan memperlambat mobil dalam bergerak untuk menghindari perangkap atau *block* yang mengandung *obstacle* setelahnya jika mobil tidak memiliki salah satu jenis *powerups* yang dapat digunakannya.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Pseudocode

```
// Main Strategy function run() → CommandDeklarasi
blocks : list of Object
nextBlocks : list of Object
isOpponentInFront : Boolean
isMudInFront : Boolean
isOilSpillInFront : Boolean
isWallInFront : Boolean
rightBlocks : list of Object
leftBlocks : list of Object
oilSpillDistance : integer
mudDistance : integer
opponentDistance : integer
wallDistance : integer
rightOilSpillDistance : integer
rightMudDistance : integer
rightOpponentDistance : integer
leftWallDistance : integer
leftOilSpillDistance : integer
leftMudDistance : integer
leftOpponentDistance : integer
leftWallDistance : integer
nearestObstacleInFrontDistance : integer
nearestObstacleInRightDistance : integer
nearestObstacleInLeftDistance : integer

Body
blocks ← getBlocksInFront(mycar.position.lane, mycar.position.block)
nextBlocks = blocks.subList(0, 1)
isOpponentInFront = blocks.contains(opponent)
isMudInFront = blocks.contains(Terrain.MUD)
isOilSpillInFront = blocks.contains(Terrain.OIL_SPILL)
isWallInFront = blocks.contains(Terrain.WALL)

if (myCar.damage >= 3) then
    → Fix
endif

if (myCar.speed = 0) then
    → new AccelerateCommand()
Endif

if (isMudInFront or isOilSpillInFront or isOpponentInFront or isWallInFront) then
    if (myCar.position.lane == 1) then
        rightBlocks = getBlocksInFront(myCar.position.lane + 1, myCar.position.block)

        oilSpillDistance = getDistanceToObject(blocks, Terrain.OIL_SPILL)
        mudDistance = getDistanceToObject(blocks, Terrain.MUD)
        opponentDistance = getDistanceToObject(blocks, opponent)
        wallDistance = getDistanceToObject(blocks, Terrain.WALL)

        rightOilSpillDistance = getDistanceToObject(rightBlocks, Terrain.OIL_SPILL)
        rightMudDistance = getDistanceToObject(rightBlocks, Terrain.MUD)
        rightOpponentDistance = getDistanceToObject(rightBlocks, opponent)
        rightWallDistance = getDistanceToObject(rightBlocks, Terrain.WALL)

        // nearest obstacle nearby
        nearestObstacleInFrontDistance = min(min(oilSpillDistance, mudDistance),
        min(opponentDistance, wallDistance))
        nearestObstacleInRightDistance = min(min(rightOilSpillDistance, rightMudDistance),
        min(rightOpponentDistance, rightWallDistance));
```

```

    // decide to turn right or just decelerate
    if (nearestObstacleInFrontDistance < nearestObstacleInRightDistance && myCar.speed
<= 3) then
    → TURN_RIGHT
    else
    → new DecelerateCommand()
    endif
else if (myCar.position.lane == 4) then
    leftBlocks = getBlocksInFront(myCar.position.lane - 1, myCar.position.block)

    oilSpillDistance = getDistanceToObject(blocks, Terrain.OIL_SPILL)
    mudDistance = getDistanceToObject(blocks, Terrain.MUD)
    opponentDistance = getDistanceToObject(blocks, opponent)
    wallDistance = getDistanceToObject(blocks, Terrain.WALL)

    rightOilSpillDistance = getDistanceToObject(rightBlocks, Terrain.OIL_SPILL)
    rightMudDistance = getDistanceToObject(rightBlocks, Terrain.MUD)
    rightOpponentDistance = getDistanceToObject(rightBlocks, opponent)
    rightWallDistance = getDistanceToObject(rightBlocks, Terrain.WALL)

    leftOilSpillDistance = getDistanceToObject(rightBlocks, Terrain.OIL_SPILL)
    leftMudDistance = getDistanceToObject(rightBlocks, Terrain.MUD)
    leftOpponentDistance = getDistanceToObject(rightBlocks, opponent)
    leftWallDistance = getDistanceToObject(rightBlocks, Terrain.WALL)

    // nearest obstacle nearby
    nearestObstacleInFrontDistance = min(min(oilSpillDistance, mudDistance),
min(opponentDistance, wallDistance))
    nearestObstacleInLeftDistance = min(min(LeftOilSpillDistance, LeftMudDistance),
min(LeftOpponentDistance, LeftWallDistance));

    // decide to turn right or just decelerate
    if (nearestObstacleInFrontDistance < nearestObstacleInLeftDistance && myCar.speed
<= 3) then
    → TURN_LEFT
    else
    → new DecelerateCommand()
    endif
else
    rightBlocks = getBlocksInFront(myCar.position.lane + 1, myCar.position.block)
    leftBlocks = getBlocksInFront(myCar.position.lane - 1, myCar.position.block)

    oilSpillDistance = getDistanceToObject(blocks, Terrain.OIL_SPILL)
    mudDistance = getDistanceToObject(blocks, Terrain.MUD)
    opponentDistance = getDistanceToObject(blocks, opponent)
    wallDistance = getDistanceToObject(blocks, Terrain.WALL)

    leftOilSpillDistance = getDistanceToObject(rightBlocks, Terrain.OIL_SPILL)
    leftMudDistance = getDistanceToObject(rightBlocks, Terrain.MUD)
    leftOpponentDistance = getDistanceToObject(rightBlocks, opponent)
    leftWallDistance = getDistanceToObject(rightBlocks, Terrain.WALL)

    // nearest obstacle nearby
    nearestObstacleInFrontDistance = min(min(oilSpillDistance, mudDistance),
min(opponentDistance, wallDistance))
    nearestObstacleInRightDistance = min(min(rightOilSpillDistance, rightMudDistance),
min(rightOpponentDistance, rightWallDistance));
    nearestObstacleInLeftDistance = min(min(LeftOilSpillDistance, LeftMudDistance),
min(LeftOpponentDistance, LeftWallDistance));

    // decide whether to turn left, right, or just decelerate
    // cases:
    // front > left && front > right -> decelerate
    // front < left && front < right -> turn right if left < right else turn left
    // left <= front && front < right -> turn right
    // left > front && front >= right -> turn left

```

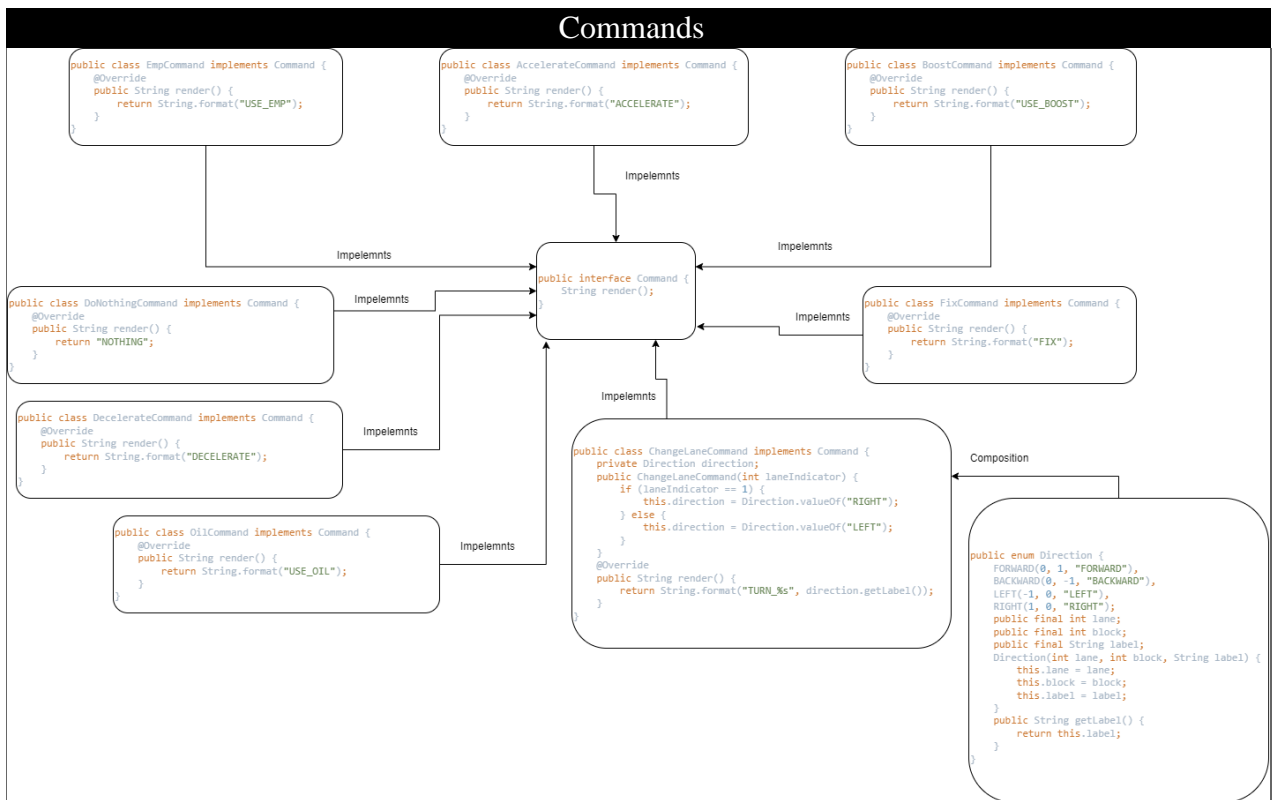
```

// left == front && front == right -> turn left if car is in lane 3 else turn right
if (nearestObstacleInFrontDistance > nearestObstacleInLeftDistance &&
nearestObstacleInFrontDistance > nearestObstacleInRightDistance ) then
    → new DecelerateCommand()
    else if (nearestObstacleInFrontDistance < nearestObstacleInLeftDistance &&
nearestObstacleInFrontDistance < nearestObstacleInRightDistance )

        if (myCar.position.lane == 3) then
            → TURN_LEFT
        else
            → TURN_RIGHT
        endif
    else if (nearestObstacleInFrontDistance >= nearestObstacleInLeftDistance &&
nearestObstacleInFrontDistance < nearestObstacleInRightDistance )
        → TURN_RIGHT
    else if (nearestObstacleInFrontDistance < nearestObstacleInLeftDistance &&
nearestObstacleInFrontDistance >= nearestObstacleInRightDistance )
        → TURN_LEFT
    else
        if (myCar.position.lane == 3) then
            → TURN_LEFT
        else
            → TURN_RIGHT
        endif
    endif
endif
endif

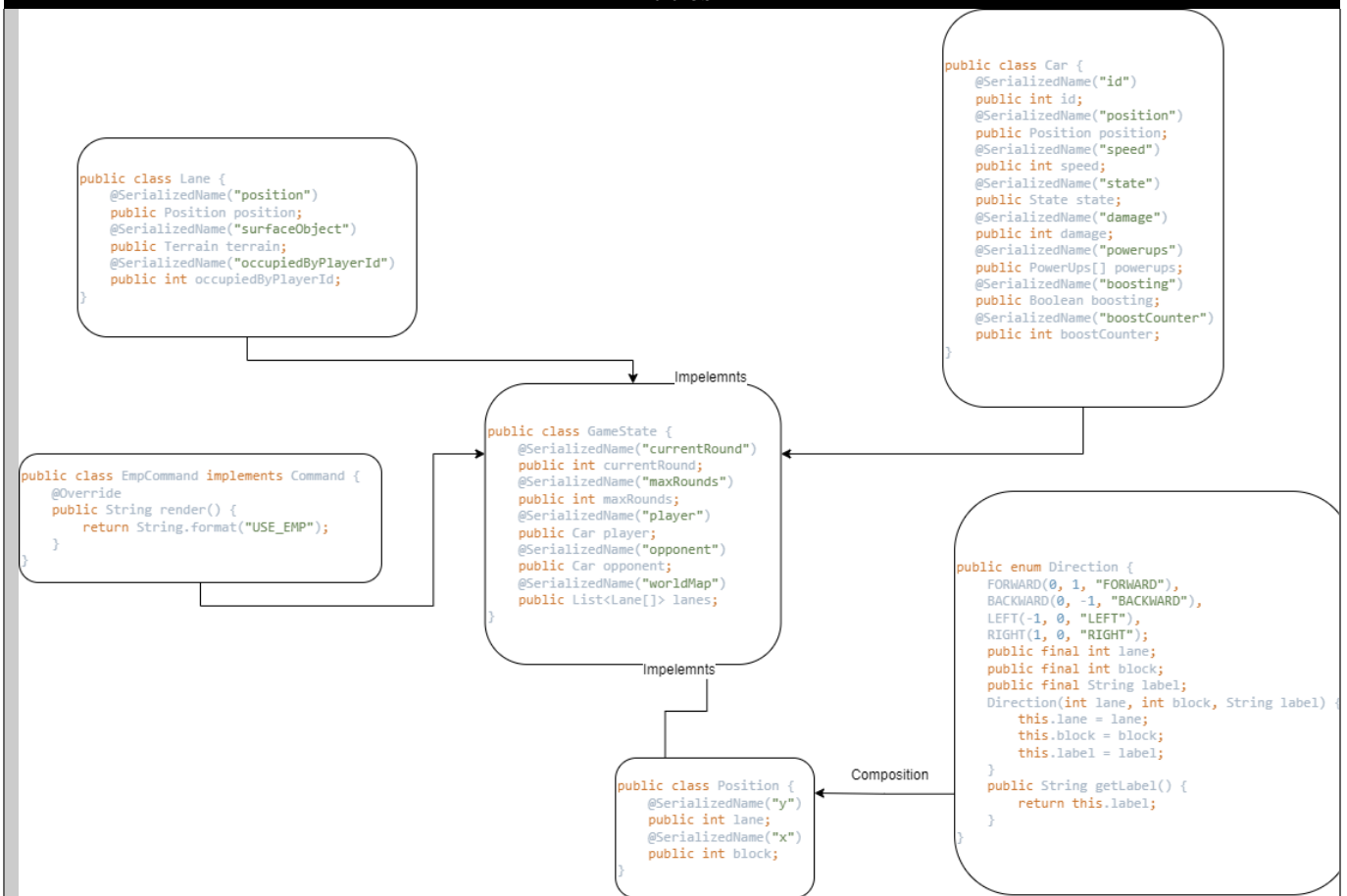
```


4.2. Struktur Data



Terdapat delapan kelas terkait *command* yang digunakan untuk mengirim perintah kepada mobil untuk melakukan suatu aksi. Perintah disampaikan menggunakan *method render* yang harus di-*override* oleh semua *class* karena semua *class* mengimplementasikan *interface command* yang mempunyai *abstract method render*. *Method render* akan dipanggil di program utama untuk mengirimkan pesan *command* kepada *game engine*. Setiap *class* (kecuali *do nothing*) minimal mempunyai target melakukan aksi.

Entities



Entities merupakan *class* yang berisikan objek-objek yang dapat di instansiasi pada *game overdriver*. *Class* paling utama dalam game adalah *GameState*. Dari *GameState* kita bisa mengetahui data peta pada game, data *player* kita, dan *player* musuh. Dari data *player* tersebut kita juga bisa mengetahui data mobil kita maupun mobil musuh.

4.3. Analisis dan Pengujian

Berdasarkan tiga kali pengujian, strategi greedy berhasil mendapat nilai optimal dengan kemenangan seratus persen dalam melawan bot referensi yang disediakan dalam starter pack. Secara umum, algoritma ini berhasil memprediksi lajur yang dipilih, memastikan gerak yang tepat pada lajur tepi, dan mempercepat bot hingga kecepatan maksimum saat jalanan kosong tanpa ada rintangan di depan. Namun, terdapat beberapa kelemahan, yaitu algoritma ini gagal ketika bot menemui lebih dari satu rintangan pada tile di depan bot sehingga bot akan terhalang oleh rintangan pada gerak tertentu. Dengan adanya kelemahan tersebut, maka pemrogram memperbaiki kelemahan tersebut dengan kemampuan bot memperbaiki dirinya saat damage-nya sebesar minimal 3 sehingga laju bot kembali optimal.

BAB V

KESIMPULAN DAN SARAN

1. Kesimpulan

Dari tugas besar IF 2211 Strategi Algoritma semester 2 2021/2022 berjudul Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Overdrive”, kami berhasil membuat sebuah *bot* dalam permainan Overdrive yang memanfaatkan algoritma greedy. Program yang dibuat tidak hanya menggunakan satu algoritma greedy melainkan beberapa algoritma greedy yang dikombinasikan untuk menghasilkan strategi yang terbaik.

2. Saran

Saran-saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma semester 2 2021/2022 adalah:

- a. Algoritma yang digunakan pada Tugas Besar ini masih memiliki banyak kekurangan sehingga sangat memungkinkan untuk dilakukan efisiensi, misalnya dengan tidak menggunakan fungsi yang sama berulang-ulang. Oleh karena itu, dalam pengembangan program ini, masih bisa dilakukan efisiensi kinerja.
- b. Memperjelas spesifikasi dan batasan-batasan setiap program pada *file* tugas besar untuk mencegah adanya multitafsir dan kesalahpahaman pada proses pembuatan program.
- c. Penulisan *pseudocode* tampak kurang perlu dikarenakan program yang lumayan panjang dan membaca program lebih mudah daripada membaca *pseudocode* dengan asumsi program sudah *well commented*.

DAFTAR PUSTAKA

- Apa itu game engine. (2012, March 08). <https://rickykurn.wordpress.com/2012/03/08/apa-itu-game-engine/> (diakses tanggal 17 Februari 2022).
- Efendi, I. (2016, December 18). Pengertian ALGORITMA Greedy. <https://www.it-jurnal.com/pengertian-algoritma-greedy/> (diakses tanggal 17 Februari 2022).
- EntelectChallenge, B. (2019, September 02). Entelectchallenge/2020-Overdriver. <https://github.com/EntelectChallenge/2020-Overdrive> (diakses tanggal 17 Februari 2022).
- Munir,Rinaldi."Algoritma Greedy (Bagian 1)". 2021. [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) (diakses pada 17 Februari 2022).
- Munir,Rinaldi."Algoritma Greedy (Bagian 2)". 2021. [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) (diakses pada 17 Februari 2022).
- Munir,Rinaldi."Algoritma Greedy (Bagian 3)". 2022. [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) (diakses pada 17 Februari 2022).
- .