

**Laporan**  
**Tugas Besar I IF2211 Strategi Algoritma**  
**Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan**  
**“Overdrive”**



Disusun Oleh:

Fransiskus Davin Anwari	13520025
Kent Liusudarso	13520069
Nicholas Budiono	13520121

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

## DAFTAR ISI

DAFTAR ISI.....	2
BAB 1 .....	3
BAB 2 .....	5
2.1    Dasar Teori (Algoritma Greedy) .....	5
2.2    Cara Kerja Program .....	6
BAB 3 .....	8
3.1    Proses Mapping .....	8
3.2    Eksplorasi Alternatif .....	8
3.3    Analisis Efisiensi .....	8
3.4    Analisis Efektivitas .....	9
3.5    Strategi greedy yang dipilih .....	10
BAB 4 .....	12
4.1    Implementasi .....	12
4.2    Struktur Data .....	17
4.3    Analisis Desain .....	18
BAB 5 .....	21
5.1    Kesimpulan .....	21
5.2    Saran .....	21
DAFTAR PUSTAKA .....	22

## 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.



Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang 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-bots 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 <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. Adapun peraturan yang lebih lengkap dari permainan Overdrive, dapat dilihat pada laman :

<https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md>

## BAB 2

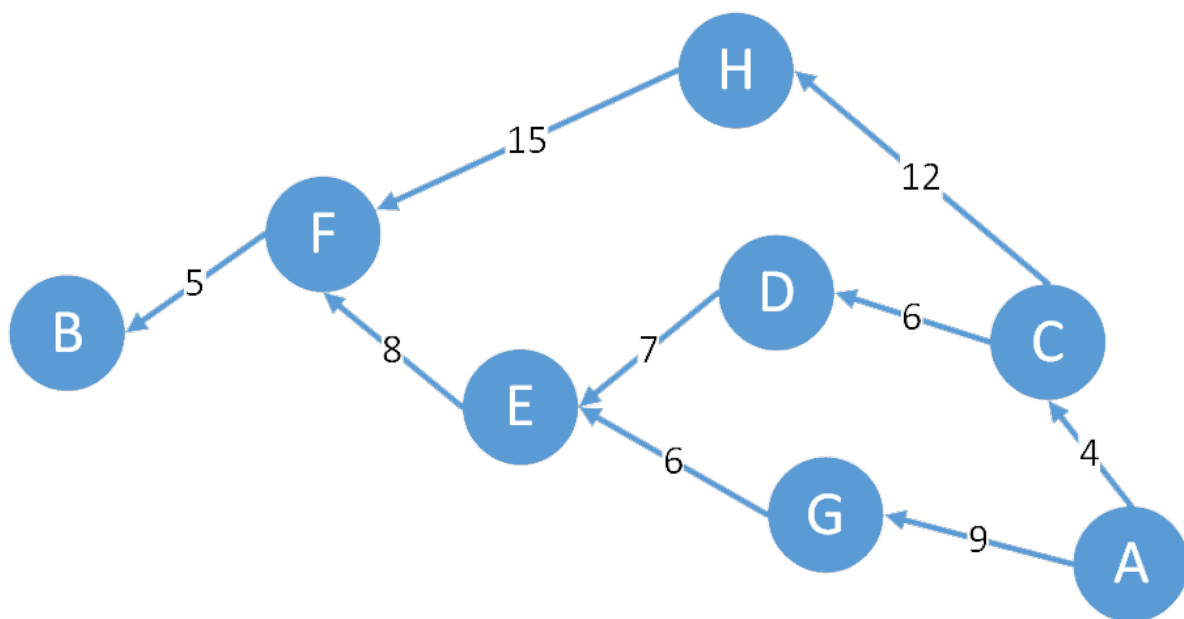
### LANDASAN TEORI

#### 2.1 Dasar Teori (Algoritma Greedy)

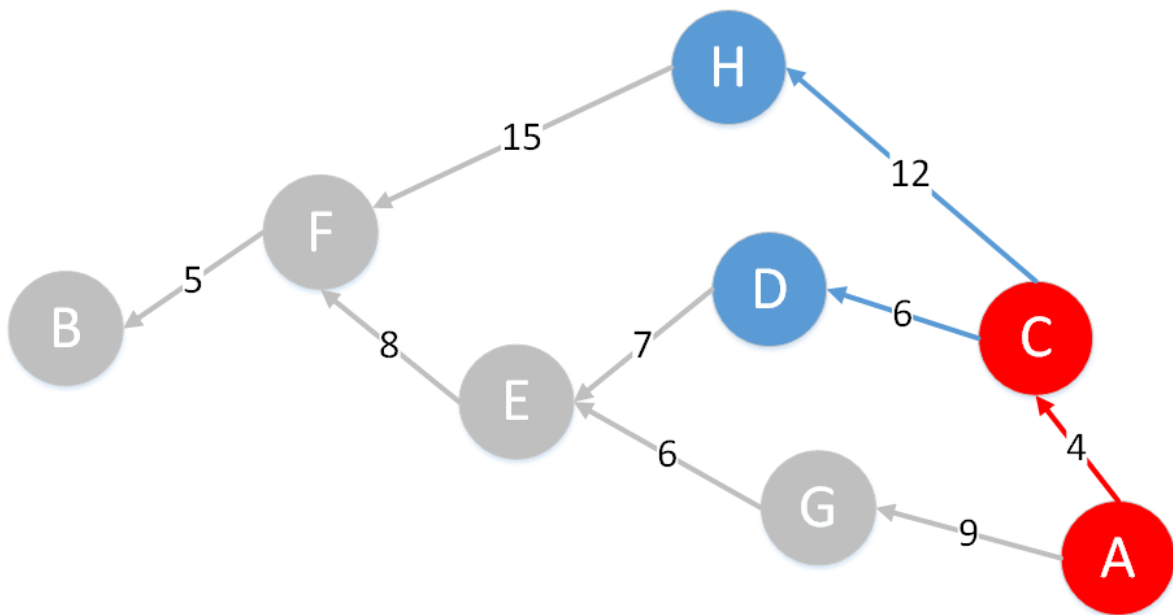
Algoritma greedy merupakan jenis algoritma yang menggunakan pendekatan penyelesaian masalah dengan mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan. Algoritma greedy ini menggunakan cara secara heuristik sehingga pada kasus tertentu akan lebih cepat dibandingkan cara yang lebih primitif seperti brute force. Pada kebanyakan kasus, algoritma greedy tidak akan menghasilkan solusi paling optimal, begitupun algoritma greedy biasanya memberikan solusi yang mendekati nilai optimum dalam waktu yang cukup cepat.

Contoh pengaplikasiannya adalah dalam permasalahan pencarian jalan tercepat untuk pergi dari satu titik ke titik tertentu. Seperti berikut:

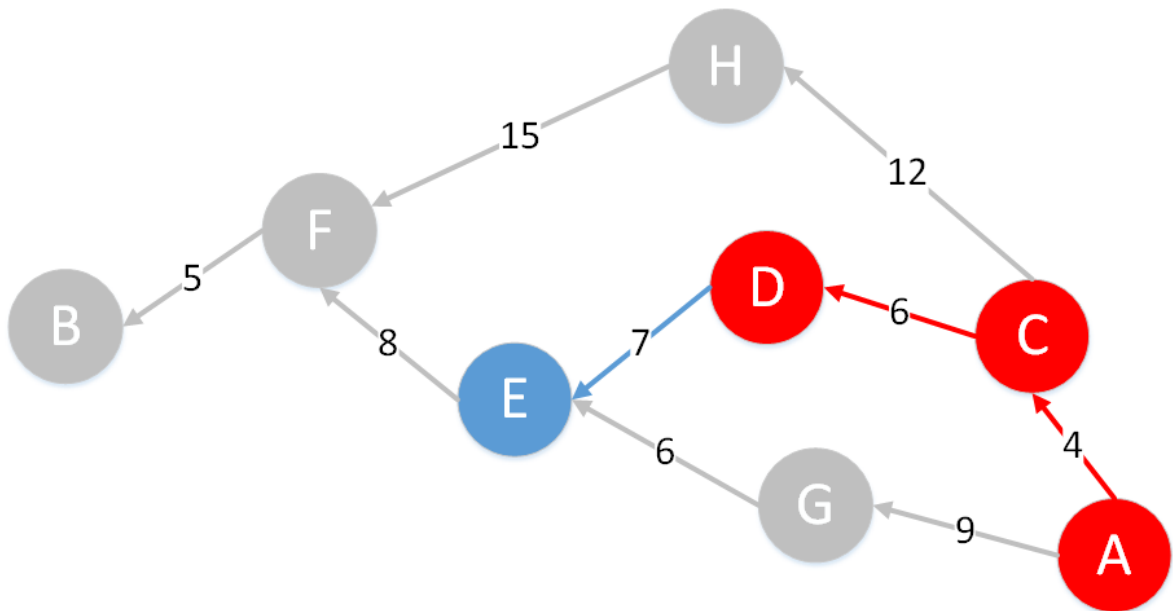
Diketahui sebuah peta dengan huruf abjad melambangkan tempat yang bisa diakses dan panah sebagai jalur yang bisa digunakan dengan jarak nya yang dilambangkan oleh angka.



Dengan menggunakan algoritma greedy, kita tau bahwa dari titik mulai A, tempat selanjutnya yang akan digunakan adalah ke C, karena jarak nya lebih kecil dari jika A pergi ke G.



Sama seperti sebelumnya, akan dipilih dari C ke D, karena lebih kecil dari C ke H.



Cara tersebut akan diteruskan sampai ke tujuan, tetapi dapat terlihat dari graf diatas bahwa algoritma greedy belum tentu menghasilkan solusi ter-mangkus. Dapat dilihat jika menggunakan jalur A – C – D – E – F – B, total jaraknya adalah 30. Sedangkan dengan jalur A – G – E – F – B adalah 28.

## 2.2 Cara Kerja Program

Program yang telah disediakan berisi beberapa bagian, yaitu game-engine, yaitu bagian dari game yang berfungsi untuk menjaga peraturan game tetap terjaga. Game-runner yang bekerja untuk melakukan pertandingan antara player. Reference-bot yang berguna sebagai sebuah referensi dari contoh bot yang bisa digunakan. Serta starter-bot yang digunakan sebagai titik awal dari bot yang ingin kita gunakan.

Untuk melakukan modifikasi pada bot nya, tinggal mengubah command yang dipanggil pada bot.java, command-command yang tersedia dari game-engine juga sudah diberikan dalam folder command, beserta apa saja yang ada di map, yaitu semua keadaan yang ada di jalan dari bot. Dengan map yang dibikin secara random sehingga tidak ada pertandingan yang sama.

Command yang ada berupa ACCELERATE, TURN\_RIGHT, TURN\_LEFT, FIX, LIZARD, OIL BOOST EMP, DECELERATE, dan DO\_NOTHING. Begitu juga dengan map, ada EMPTY, MUD, WALL, dan tempat power up ditempatkan. Dengan command ini, kita bisa memanipulasi starter\_bot sehingga bisa meningkatkan efektivitas dari bot kita, dengan ini juga kita bisa melakukan pertandingan antar bot.

Tidak terlepas juga dengan aturan-aturan yang berlaku, seperti yang dikatakan tadi, game ini berupa 2D array yaitu map, dengan isi nya masing-masing, ada halangan dan ada juga power up. Mobil dapat melihat sejauh 20 blok kedepan, serta 5 blok ke belakang.

Mobil memiliki darah sebanyak 5 dan banyak state yang bisa terjadi pada mobil. Kecepatan mobil dengan maksimal 15, serta darah mobil yaitu 5. Game ini juga memiliki sistem score yang akan bertambah jika menggunakan power up, berkurang jika menggunakan command yang invalid.

## BAB 3

### APLIKASI STRATEGI GREEDY

#### 3.1 Proses Mapping

- Himpunan kandidat : semua command
- Himpunan solusi : command yang terpilih
- Fungsi solusi : memeriksa apakah command yang dipilih merupakan command terbaik untuk mencapai *finish line*
- Fungsi seleksi : memilih command terbaik
- Fungsi kelayakan : memeriksa apakah command yang dipilih valid
- Fungsi objektif : round yang diperlukan untuk mencapai *finish line* minimum

#### 3.2 Eksplorasi Alternatif

- *Greedy by Menghindar*
  - Memprioritaskan untuk menghindari semua rintangan. Mobil akan accelerate ketika di depan tidak ada halangan, tetapi akan berusaha sebaik mungkin untuk menghindari halangan. Karena mengfokuskan menghindari maka mobil akan FIX hanya jika mobil tidak bisa berjalan atau damage = 5.
- *Greedy by PowerUp*
  - Memprioritaskan untuk mengambil dan menggunakan powerup. Mobil akan mencari jalan yang paling efektif untuk mendapatkan powerup. Karena itu mobil akan berbelok ke lane yang terdapat halangan jika lane tersebut terdapat powerup. Dengan prekondisi seperti itu maka pada algoritma ini mobil akan sering mengalami tabrakan dengan halangan. Selain itu beberapa powerup juga tidak akan maksimal digunakan, contohnya powerup BOOST yang paling efektif jika damage mobil = 0 hingga perlu banyak FIX.
- *Greedy by Acceleration*
  - Hanya memprioritaskan acceleration tanpa menghindari rintangan dan fix jika damage mobil lebih dari 0. Dengan algoritma ini perintah TURN RIGHT dan TURN LEFT tidak akan digunakan. Mobil hanya akan maju pada 1 lane dan langsung melakukan FIX jika mobil menabrak halangan. Selain itu powerup yang didapatkan mobil hanya sedikit karena mobil menetap di 1 lane sedangkan powerup tersebar di banyak lane.

#### 3.3 Analisis Efisiensi

- *Greedy by Menghindar*
  - Pada setiap round, algoritma ini perlu mengecek area di depan dan disampingnya untuk memastikan bahwa mobil dapat memilih *lane* yang paling aman untuk dilewati. Karena itu algoritma ini tidak bisa dibilang paling efisien karena tiap round harus menjalankan fungsi *inFront*, *inRight*, dan *inLeft* untuk mencari halangan untuk dihindari.
- *Greedy by PowerUp*



- Sama seperti *greedy by powerup*, algoritma ini juga perlu mengecek area di depan dan disampingnya pada setiap round untuk mencari keberadaan powerup. Karena itu pada satu iterasi round perlu menjalankan 3 fungsi seperti *greedy by* menghindari hingga tidak bisa dibilang sebagai algoritma paling efisien.
- *Greedy by Acceleration*
  - Kode berjalan lumayan cepat karena kode tidak perlu membuang waktu untuk melihat area kira kanan ataupun depan mobil. Mobil hanya melakukan command accelerate selama bisa mencapai kecepatan maksimal, dan FIX jika tidak. Karena setiap round program hanya perlu mengecek damage mobil, maka dapat dibilang antara ketiga strategi *greedy* yang telah temukan strategi ini adalah yang paling efisien.

### 3.4 Analisis Efektivitas

- *Greedy by* menghindar
  - Algoritma *greedy by* menghindar mengabaikan keberadaan powerup pada permainan, selain itu algoritma ini juga tidak terlalu memikirkan mekanik kecepatan maksimum yang dapat dicapai bot. Karena itu, jika bot mencapai kecepatan yang rendah, algoritma tidak akan memilih untuk melakukan command FIX karena bagi bot kecepatan rendah merupakan kecepatan ideal karena dapat menghindari halangan dengan lebih mudah. Tetapi karena itu bot akan memakan waktu lama untuk mencapai tujuan sehingga dapat dibilang algoritma ini kurang efektif.
- *Greedy by powerup*
  - Algoritma *greedy by powerup* mengutamakan pengambilan powerup sehingga mayoritas waktu bot akan mengorbankan mobil menabrak halangan untuk mendapatkan powerup. Hal tersebut menyebabkan bot mengalami banyak tabrakan sehingga kecepatan bot tidak akan optimal. Sama juga seperti kasus *greedy by* menghindar, bot tidak akan mengutamakan command FIX karena kecepatan rendah paling optimal untuk mendapatkan powerup paling banyak. Selain itu beberapa powerup seperti BOOST tidak akan bisa digunakan secara optimal karena untuk mencapai kecepatan yang hanya dapat dicapai BOOST diperlukan banyak FIX oleh bot agar damage mobil = 0. Karena kecepatan bot rendah, maka kemungkinan besar bot akan berposisi dibelakang musuh, walaupun ini kondisi yang baik untuk menggunakan powerup seperti EMP, tetapi hal ini menyebabkan powerup OIL menjadi tidak berguna karena tidak ada musuh yang akan melewati OIL tersebut. Walaupun algoritma ini mendapatkan banyak powerup, beberapa powerup tidak dapat dimaksimalkan sesuai kebutuhannya sehingga algoritmanya tidak efektif.
- *Greedy by Acceleration*
  - Algoritma *greedy by Acceleration* adalah algoritma yang hanya mengutamakan kecepatan dibandingkan semuanya. Karena algoritma ini akan terus FIX hingga dapat mencapai kecepatan maksimal, maka dengan itu algoritma ini langsung lebih efektif dibandingkan *greedy by powerup* dan *greedy by* menghindar yang lebih memilih untuk berkecepatan rendah. Tetapi karena pada algoritma ini bot

hanya tetap di 1 lane maka akan mengalami banyak tabrakan dengan halangan sehingga perlu melakukan banyak FIX. Karena itu algoritma ini juga belum paling efektif karena membuang banyak waktu untuk memperbaiki mobil karena frekuensi tabrakan yang tinggi.

### 3.5 Strategi greedy yang dipilih

Ketiga strategi greedy yang kami temukan bukanlah strategi yang paling efektif. Karena itu kami menggabungkan ketiga strategi greedy tersebut menjadi strategi yang menentukan aksi yang akan dilakukan sesuai dengan situasi pada tiap round. Karena pada dasarnya strategi *greedy* adalah memilih solusi yang paling tepat pada situasi tersebut, maka algoritma gabungan ini masih tergolongkan algoritma *greedy*. Implementasinya dengan membuat prioritas command-command yang digunakan agar dapat menyelesaikan track dengan round sedikit mungkin. Karena itu perlu dibuat terlebih dahulu urutan prioritas command yang akan digunakan untuk merancang algoritma greedynya.

Dengan melakukan analisis dari cara kerja tiap command dan dengan melakukan trial and error, kami mendapatkan urutan prioritas command mobil sebagai berikut.

**FIX > BOOST > LIZARD > TURN LEFT / TURN RIGHT > USE OIL > USE EMP >  
USE TWEET > ACCELERATE**

Dengan urutan sebagai berikut kami mendapatkan hasil penyelesaian track dengan round paling sedikit yaitu rata-ratanya sekitar 160 round. Selain merancang algoritma greedy mengikuti urutan prioritas tersebut, kami juga menambahkan beberapa state dan solusi yang khusus untuk state tertentu.

Agar algoritma dapat didefinisikan sebagai algoritma greedy maka pada setiap state harus terdapat solusi yang paling tepat. Untuk mempermudah kami membuatnya dalam bentuk tabel.

State	Solusi
Jalan depan mobil tidak ada apa-apa	Accelerate
Kecepatan mobil 0	Accelerate
Posisi pada lane 1 / 4 dan dan kiri/kanan mobil kosong	Pindah ke lane 2/ 3
Jalan depan terdapat mud dan speed mobil 1-3	Turn Right / Left
Jalan depan ada mud dan speed 0 dan damage > 1	Accelerate
Jalan depan ada mud dan speed >3 dan	Accelerate
Jalan depan ada wall	Turn Right / Left
Jalan depan ada CT	Turn Right / Left
Boost tersedia dan damage mobil = 0	Use Boost
Boost tersedia dan damage mobil != 0	Fix
Damage Mobil > 2	Fix
Memiliki power up	Menggunakan power up yang bersangkutan (kecuali power up lizard)
Jika mau belok dan memiliki power up lizard	Menggunakan lizard
Musuh memakai TWEET	Menggunakan Lizard jika ada, menghindari cybertruck
Di Lane sebelah mobil terdapat powerup	Belok untuk mengambil power up

Dengan Menggabungkan State-Solution diatas dengan urutan prioritas command yang telah kami buat, kami dapat mendapat algoritma greedy yang paling efektif untuk memenangkan race.

## BAB 4

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi

```
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();
    private final static Command ACCELERATE = new AccelerateCommand();
    private final static Command LIZARD = new LizardCommand();
    private final static Command OIL = new OilCommand();
    private final static Command BOOST = new BoostCommand();
    private final static Command EMP = new EmpCommand();
    private final static Command TURN_RIGHT = new ChangeLaneCommand(1);
    private final static Command TURN_LEFT = new ChangeLaneCommand(-1);
    // private final static Command DECELERATE = new DecelerateCommand();
    // private final static Command DO_NOTHING = new DoNothingCommand();

    public Bot(Random random, GameState gameState) {
        this.random = random;
        this.gameState = gameState;
        this.myCar = gameState.player;
        this.opponent = gameState.opponent;
    }

    public Command run() {
        List<Object> inFront = getBlocks(myCar.position.lane,
myCar.position.block);

        // Situasi Fix -> jika damage 3 ke atas dan tidak punya boost
        (damage tidak
        // perlu sampe 0)
        if (myCar.damage > 2 && !hasPowerUp(PowerUps.BOOST,
myCar.powerups)) {
            return FIX;
        }

        // Situasi jika punya boost -> Fix jika damage belum 0, Boost jika
        damage = 0
        if (hasPowerUp(PowerUps.BOOST, myCar.powerups) && myCar.speed < 15)
        {
            if (myCar.damage != 0) {
                return FIX;
            } else {
                return BOOST;
            }
        }

        // fail safe
        if (myCar.speed == 0) {
            return ACCELERATE;
        }
    }
}
```

```

        // HINDAR
        if ((inFront.contains(Terrain.MUD) ||
inFront.contains(Terrain.OIL_SPILL) || inFront.contains(Terrain.WALL)
        || inFront.contains(Terrain.CYBER_TRUCK))) {
            // Situasi di lane 1 dan ada Obstacle
            if (myCar.position.lane == 1) {
                if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
                    return LIZARD;
                } else {
                    return TURN_RIGHT;
                }
            }
            // Situasi di lane 4 dan ada Obstacle
            else if (myCar.position.lane == 4) {
                if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
                    return LIZARD;
                } else {
                    return TURN_LEFT;
                }
            }
            // Situasi di lane 2 dan ada Obstacle
            else if (myCar.position.lane == 2 || myCar.position.lane == 3)
        {
            List<Object> inRight = getBlocks(myCar.position.lane + 1,
myCar.position.block - 1);
            List<Object> inLeft = getBlocks(myCar.position.lane - 1,
myCar.position.block - 1);
            if (myCar.position.lane == 2) {
                if ((inLeft.contains(Terrain.MUD) ||
inLeft.contains(Terrain.OIL_SPILL)
                || inLeft.contains(Terrain.WALL)
                || inLeft.contains(Terrain.CYBER_TRUCK)) &&
(inRight.contains(Terrain.MUD)
                || inRight.contains(Terrain.OIL_SPILL)
                || inRight.contains(Terrain.WALL) ||
inRight.contains(Terrain.CYBER_TRUCK))) {
                    if (inLeft.size() > inRight.size()) {
                        if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                            return LIZARD;
                        } else {
                            return TURN_RIGHT;
                        }
                    }
                    else if (inLeft.size() < inRight.size()) {
                        if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                            return LIZARD;
                        } else {
                            return TURN_LEFT;
                        }
                    }
                    else {
                        if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                            return LIZARD;
                        } else {
                            return ACCELERATE;
                        }
                    }
                }
            }
            else if (inLeft.contains(Terrain.MUD) ||
inLeft.contains(Terrain.OIL_SPILL)
            || inLeft.contains(Terrain.WALL)

```

```

        || inLeft.contains(Terrain.CYBER_TRUCK)) {
    if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
        return LIZARD;
    } else {
        return TURN_RIGHT;
    }
} else if (inRight.contains(Terrain.MUD)
    || inRight.contains(Terrain.OIL_SPILL)
    || inRight.contains(Terrain.WALL) ||
inRight.contains(Terrain.CYBER_TRUCK)) {
    if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
        return LIZARD;
    } else {
        return TURN_LEFT;
    }
} else {
    if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
        return LIZARD;
    } else {
        return TURN_LEFT;
    }
}
}
// Situasi di lane 3 dan ada Obstacle
if (myCar.position.lane == 3) {
    if ((inLeft.contains(Terrain.MUD) ||
inLeft.contains(Terrain.OIL_SPILL)
    || inLeft.contains(Terrain.WALL)
    || inLeft.contains(Terrain.CYBER_TRUCK)) &&
(inRight.contains(Terrain.MUD)
    || inRight.contains(Terrain.OIL_SPILL)
    || inRight.contains(Terrain.WALL) ||
inRight.contains(Terrain.CYBER_TRUCK))) {
        if (inLeft.size() > inRight.size()) {
            if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                return LIZARD;
            } else {
                return TURN_RIGHT;
            }
        } else if (inLeft.size() < inRight.size()) {
            if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                return LIZARD;
            } else {
                return TURN_LEFT;
            }
        } else {
            if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) {
                return LIZARD;
            } else {
                return ACCELERATE;
            }
        }
    } else if (inLeft.contains(Terrain.MUD) ||
inLeft.contains(Terrain.OIL_SPILL)
    || inLeft.contains(Terrain.WALL)
    || inLeft.contains(Terrain.CYBER_TRUCK)) {
        if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
            return LIZARD;

```

```

        } else {
            return TURN_RIGHT;
        }
    } else if (inRight.contains(Terrain.MUD)
        || inRight.contains(Terrain.OIL_SPILL)
        || inRight.contains(Terrain.WALL) ||
inRight.contains(Terrain.CYBER_TRUCK)) {
        if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
            return LIZARD;
        } else {
            return TURN_LEFT;
        }
    } else {
        if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
            return LIZARD;
        } else {
            return TURN_LEFT;
        }
    }
}
} else {
    if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {
        return LIZARD;
    } else {
        return ACCELERATE;
    }
}

// Situasi jika punya oil -> cek jika menang, jika menang maka pake
oil
if (hasPowerUp(PowerUps.OIL, myCar.powerups) && myCar.speed < 15) {
    if (myCar.position.block > opponent.position.block) {
        return OIL;
    }
}

// Situasi jika punya EMP -> cek jika menang, jika kalah tembak emp
kalo lanenya dekat dengan lane musuh
if (hasPowerUp(PowerUps.EMP, myCar.powerups) && myCar.speed < 15) {
    if (myCar.position.block < opponent.position.block) {
        if (myCar.position.lane == opponent.position.lane ||
myCar.position.lane == opponent.position.lane + 1 || myCar.position.lane ==
opponent.position.lane - 1) {
            return EMP;
        }
    }
}

// Situasi jika punya TWEET -> pake di depan musuh
if (hasPowerUp(PowerUps.TWEET, myCar.powerups) && myCar.speed < 15
/*&& opponent.speed > 5*/) {
    return TWEET(opponent.position.lane, opponent.position.block +
opponent.speed + 1);
}

// Greedy PowerUP
if ((!inFront.contains(Terrain.MUD) ||
!inFront.contains(Terrain.OIL_SPILL) || !inFront.contains(Terrain.WALL) ||
!inFront.contains(Terrain.CYBER_TRUCK))) {
    if (myCar.position.lane == 1) {

```

```

        List<Object> inRight = getBlocks(myCar.position.lane + 1,
myCar.position.block - 1);
        if (inRight.contains(Terrain.BOOST)) {
            return TURN_RIGHT;
        }
        else {
            return ACCELERATE;
        }
    }
    else if (myCar.position.lane == 4) {
        List<Object> inLeft = getBlocks(myCar.position.lane - 1,
myCar.position.block - 1);
        if (inLeft.contains(Terrain.BOOST)) {
            return TURN_LEFT;
        }
        else {
            return ACCELERATE;
        }
    }
    else if (myCar.position.lane == 2 || myCar.position.lane == 3)
{
        List<Object> inRight = getBlocks(myCar.position.lane + 1,
myCar.position.block - 1);
        List<Object> inLeft = getBlocks(myCar.position.lane - 1,
myCar.position.block - 1);
        if (inRight.contains(Terrain.BOOST)) {
            return TURN_RIGHT;
        }
        else if (inLeft.contains(Terrain.BOOST)) {
            return TURN_LEFT;
        }
        else {
            return ACCELERATE;
        }
    }
    return ACCELERATE;
}

return ACCELERATE;
}

// Mengecek apakah kachow memiliki power up
private Boolean hasPowerUp(PowerUps powerUpToCheck, PowerUps[]
available) {
    for (PowerUps powerUp : available) {
        if (powerUp.equals(powerUpToCheck)) {
            return true;
        }
    }
    return false;
}

// Mengecek block disekitar kachow
private List<Object> getBlocks(int lane, int block) {
    List<Lane[]> map = gameState.lanes;
    List<Object> blocks = new ArrayList<>();
    int startBlock = map.get(0)[0].position.block;

    Lane[] laneList = map.get(lane - 1);
    if (myCar.speed == 15) {

```



```

        for (int i = max(block - startBlock, 0); i <= block -
startBlock + 15; i++) {
            if (laneList[i] == null || laneList[i].terrain ==
Terrain.FINISH) {
                break;
            }

            blocks.add(laneList[i].terrain);
        }
    } else {
        for (int i = max(block - startBlock, 0); i <= block -
startBlock + Bot.maxSpeed; i++) {
            if (laneList[i] == null || laneList[i].terrain ==
Terrain.FINISH) {
                break;
            }

            blocks.add(laneList[i].terrain);
        }
    }

    return blocks;
}

// Jalankan TWEET pada koordinat tertentu
private Command TWEET(int lane, int block) {
    return new TweetCommand(lane, block);
}
}

```

## 4.2 Struktur Data

Struktur data yang digunakan dalam bot Overdrive yaitu:

- Car
  - Int id
  - Int speed
  - Int damage
  - Boolean boosting
  - Int boostCounter
- GameState
  - Int currentRound
  - Int maxRounds
- Lane
  - Int occupideByPlayerId
- Position
  - Int lane
  - Int block
- Direction
  - FORWARD
  - BACKWARD
  - LEFT

- RIGHT
- PowerUps
  - BOOST
  - OIL
  - TWEET
  - LIZARD
  - EMP
- State
  - ACCELERATING
  - READY
  - NOTHING
  - TURNING\_RIGHT
  - TURNING\_LEFT
  - HIT\_MID
  - HIT\_OIL
  - DECELERATING
  - PICKED\_UP\_POWERUP
  - USED\_BOOST
  - USED\_OIL
  - USED\_LIZARD
  - USED\_TWEET
  - HIT\_WALL
  - HIT\_CYBER\_TRUCK
  - FINISHED
  - USED\_EMP
  - HIT\_EMP
- Terrain
  - EMPTY
  - MUD
  - OIL\_SPILL
  - OIL\_POWER
  - FINISH
  - BOOST
  - WALL
  - LIZARD
  - TWEET
  - EMP
  - CYBER\_TRUCK

### 4.3 Analisis Desain

Untuk menganalisis Algoritma secara rinci maka berikut adalah beberapa contoh kasus untuk dianalisis solusi yang dipilih:

[27, 1]	[28, 1]	[29, 1]	[30, 1]	[31, 1] <MUD>	[32, 1] <MUD>	[33, 1]	[34, 1]	[35, 1]	[36, 1]	[37, 1]	[38, 1]	[39, 1]	[40, 1] <TWEET>	[41, 1]	[42, 1] <BOOST>	[43, 1]	[44, 1]	[45, 1]	[46, 1]	[47, 1]	[48, 1]	[49, 1]	[50, 1]	[51, 1]	[52, 1]
[27, 2]	[28, 2]	[29, 2]	[30, 2] <MUD>	[31, 2]	[32, 2] <PLAYER>	[33, 2]	[34, 2]	[35, 2]	[36, 2]	[37, 2]	[38, 2]	[39, 2]	[40, 2] <WALL>	[41, 2]	[42, 2]	[43, 2]	[44, 2] <BOOST>	[45, 2]	[46, 2]	[47, 2]	[48, 2]	[49, 2]	[50, 2]	[51, 2]	[52, 2]
[27, 3]	[28, 3]	[29, 3]	[30, 3]	[31, 3]	[32, 3]	[33, 3]	[34, 3]	[35, 3]	[36, 3]	[37, 3]	[38, 3]	[39, 3]	[40, 3]	[41, 3]	[42, 3]	[43, 3]	[44, 3]	[45, 3]	[46, 3]	[47, 3]	[48, 3]	[49, 3] <MUD>	[50, 3]	[51, 3]	[52, 3]
[27, 4]	[28, 4]	[29, 4] <BOOST>	[30, 4] <PLAYER>	[31, 4]	[32, 4]	[33, 4]	[34, 4]	[35, 4]	[36, 4]	[37, 4]	[38, 4]	[39, 4] <OIL>	[40, 4] <OIL>	[41, 4]	[42, 4]	[43, 4]	[44, 4]	[45, 4]	[46, 4] <WALL>	[47, 4]	[48, 4]	[49, 4]	[50, 4]	[51, 4]	[52, 4]

Pada kondisi diatas, bot berada pada state dimana didepannya terdapat suatu obstacle berupa WALL. Karena pada saat ini bot diasumsikan tidak memiliki LIZARD maka bot harus memilih untuk belok ke kanan ataupun ke kiri. Pada instansi ini bot memilih untuk belok ke lane 3 karena di lane 1 terdapat obstacle berupa MUD.

Jika kita melihat hanya situasi itu saja, bot memilih pilihan yang paling tepat yaitu menghindari ke lane 3. Tetapi jika kita melihat round selanjutnya, bot menjadi kehabisan pilihan karena di depan terdapat halangan MUD sedangkan di kedua lane sebelahny terdapat WALL. Sedangkan jika pada ronde sebelumnya bot berpindah ke lane 1, bot hanya akan melewati obstacle MUD sekali dan mendapat powerup berupa TWEET dan BOOST. Jadi jika kita melihat 1 ronde kedepan dapat disimpulkan bahwa langkah yang dipilih bot bukan pilihan paling benar. Hal ini menunjukan kelemahan algoritma *greedy* dimana algoritma tersebut memilih solusi dengan memperhatikan kondisi pada saat itu juga tanpa melihat konsekuensi aksi yang dipilih di masa depan.

Selanjutnya adalah kasus dimana bot menggunakan LIZARD untuk melewati obstacle.

[470, 1]	[471, 1]	[472, 1]	[473, 1]	[474, 1]	[475, 1] <TWEET>	[476, 1]	[477, 1]	[478, 1]	[479, 1]	[480, 1]	[481, 1]	[482, 1]	[483, 1]	[484, 1] <BOOST>	[485, 1]	[486, 1]	[487, 1] <WALL>	[488, 1]	[489, 1] <LIZARD>	[490, 1]	[491, 1]	[492, 1]	[493, 1]	[494, 1]	[495, 1]
[470, 2]	[471, 2]	[472, 2]	[473, 2]	[474, 2]	[475, 2] <PLAYER>	[476, 2]	[477, 2]	[478, 2]	[479, 2]	[480, 2]	[481, 2]	[482, 2]	[483, 2]	[484, 2]	[485, 2]	[486, 2] <MUD>	[487, 2] <TWEET>	[488, 2]	[489, 2]	[490, 2] <LIZARD>	[491, 2]	[492, 2]	[493, 2]	[494, 2]	[495, 2]
[470, 3]	[471, 3]	[472, 3]	[473, 3]	[474, 3]	[475, 3]	[476, 3]	[477, 3]	[478, 3]	[479, 3]	[480, 3]	[481, 3]	[482, 3]	[483, 3]	[484, 3]	[485, 3]	[486, 3]	[487, 3]	[488, 3]	[489, 3]	[490, 3]	[491, 3]	[492, 3]	[493, 3]	[494, 3]	[495, 3]
[470, 4]	[471, 4]	[472, 4]	[473, 4] <MUD>	[474, 4]	[475, 4]	[476, 4]	[477, 4]	[478, 4]	[479, 4]	[480, 4]	[481, 4]	[482, 4]	[483, 4]	[484, 4] <BOOST>	[485, 4]	[486, 4] <LIZARD>	[487, 4]	[488, 4]	[489, 4]	[490, 4]	[491, 4]	[492, 4]	[493, 4] <LIZARD>	[494, 4]	[495, 4]


First Prev 48 Next Last

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

#### Round Details

Max Rounds: 600

Current Round: 48

A - kachow						
						
Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
1 [x: 475, y: 2]	15	2	475	4	YES	TWEET,OIL,TWEET, EMP,OIL,TWEET,EM P,OIL,LIZARD,TWEE T,OIL,LIZARD
Bot Command						
Command: USE_LIZARD						
Execution time: 1ms						

Pada kasus ini bot sedang dalam kondisi BOOSTING dan akan menabrak suatu mud. Karena bot memiliki powerup LIZARD maka bot memilih untuk menggunakan powerup tersebut dibandingkan untuk belok menghindari obstacle. Hal ini dilakukan dengan mengikuti urutan prioritas command yang sebelumnya telah dibuat untuk algoritma *greedy* ini.

Walaupun dapat dibilang bahwa pilihan bot tidak salah, tetapi hal ini tidak memikirkan bahwa di ronde-ronde selanjutnya terdapat kemungkinan bahwa bot akan mencapai suatu

situasi dimana bot tidak dapat berbelok ke kiri ataupun kanan tanpa menabrak obstacle. Pada saat ini sebenarnya bot dapat berbelok ke lane 3 tanpa menabrak MUD sehingga LIZARD dapat tersimpan. Tetapi karena algoritma ini merupakan algoritma *greedy* maka bot tetap akan menggunakan powerup LIZARD jika dianggap akan membantu pada situasi itu juga.

## **BAB 5**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Kami menggunakan algoritma greedy untuk membantu pemilihan keputusan bot pada permainan Overdrive. Algoritma greedy yang dibuat adalah gabungan dari algoritma greedy yang mengutamakan 3 komponen pada permainan yaitu halangan, powerup, dan kecepatan. Hal ini dilakukan dengan membuat urutan prioritas command yang cocok dan juga menjabarkan berbagai macam state yang dapat muncul pada permainan dan solusi yang sesuai. Implementasi algoritma ini menghasilkan bot yang dapat menyelesaikan suatu permainan sepanjang 1500 block dengan banyak ronde kisaran 160-180.

#### **5.2 Saran**

Program kami masih dapat dioptimisasi dengan menganalisis lebih mengenai prioritas algoritma greedy serta struktur dari kode. Dengan program yang lebih optimal, bot kami akan melakukan aksi terbaik terhadap pada setiap round yang dilalui.

## DAFTAR PUSTAKA

Algoritma Greedy. Diakses pada 15 Februari 2022, dari <http://dev.bertzzie.com/knowledge/analisis-algoritma/Greedy.html>

Dr. Ir. Rinaldi Munir, MT. (2022). Strategi Algoritma [Presentasi PowerPoint]. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf)

2020-Overdrive. Diakses pada 1 Februari 2022, dari <https://github.com/EntelectChallenge/2020-Overdrive>

## REPOSITORY DAN VIDEO

Repository Github : <https://github.com/kentlius/Tubes1Stima>

Video Demo : <https://youtu.be/uo4Cc8TpzM8>