Laporan Tugas Besar 1
IF2211 Strategi Algoritma
Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"

Oleh:

Kelompok 43 - Wheel of Fortune

Muhammad Akyas David Al Aleey	13520011
Adzka Ahmadetya Zaidan	13520127
Muhammad Fahmi Irfan	13520152

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

SEMESTER II 2021/2022

Daftar Isi

Bab I
Deskripsi Tugas

1.1. Spesifikasi Tugas

Pada tugas besar kali ini, anda diminta untuk membuat sebuah bot untuk bermain permainan Overdrive yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

1. Download latest release starter pack.zip dari tautan berikut https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4
2. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut.
    a. Java (minimal Java 8): https://www.oracle.com/java/technologies/downloads/#java8
    b. IntelIiJ IDEA: https://www.jetbrains.com/idea/
    c. NodeJS: https://nodejs.org/en/download/
3. Untuk menjalankan permainan, kalian dapat membuka file "run.bat" (Untuk Windows dapat buka dengan double-click, Untuk Linux/Mac dapat menjalankan command "make run").
4. Secara default, permainan akan dilakukan diantara reference bot (default-nya berbahasa Java) dan starter bot (default-nya berbahasa JavaScript) yang disediakan. Untuk mengubah hal tersebut, silahkan edit file "game-runner-config.json". Anda juga dapat mengubah file "bot.json" dalam direktori "starter-bots" untuk mengatur informasi terkait bot anda.
5. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di starter-bots. Ingat bahwa bot kalian harus menggunakan bahasa Java dan di-build menggunakan IntelIiJ sebelum menjalankan permainan kembali. Dilarang menggunakan kode program yang sudah ada untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan visualizer berikut https://github.com/Affuta/overdrive-round-runner
7. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut https://github.com/EntelectChallenge/2020-Overdrive.

Strategi greedy yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mencapai garis finish lebih awal atau mencapai garis finish bersamaan tetapi dengan kecepatan lebih besar atau memiliki skor terbesar jika kedua komponen tersebut masih bernilai imbang. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menggunakan powerups begitu ada untuk mengganggu mobil musuh. Buatlah strategi greedy

terbaik, karena setiap bot dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (TBD).

Strategi greedy harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

Bab II
Landasan Teori

2.1. Algoritma Greedy

Algoritma Greedy merupakan algoritma yang cukup populer. Algoritma ini menyelesaikan persoalan dengan cara langkah demi langkah sedemikian sehingga opsi terbaik pada suatu langkah (atau biasa disebut optimum lokal) akan dipilih tanpa meninjau konsekuensi ke depannya. Untuk beberapa permasalahan, pemilihan optimum lokal di tiap langkahnya dapat menghasilkan optimum global, namun ini tidak berlaku untuk semua permasalahan. Elemen-elemen pada algoritma ini ialah

1. Himpunan kandidat , yaitu himpunan yang berisi opsi-opsi yang dapat dipilih pada setiap langkah,
2. Himpunan solusi, yaitu himpunan yang berisi opsi-opsi yang telah dipilih
3. Fungsi solusi, yaitu fungsi yang menentukan apakah opsi-opsi yang telah dipilih sudah memberikan solusi
4. Fungsi seleksi, yaitu fungsi yang memilih kandidat opsi menggunakan algoritma greedy
5. Fungsi kelayakan, yaitu fungsi yang memeriksa apakah kandidat opsi layak dimasukkan dalam himpunan solusi,
6. Fungsi obyektif, yang menyatakan suatu persoalan merupakan persoalan minimasi atau maksimasi.

2.2. Cara Kerja Program

2.2.1. Struktur Program

Secara umum, program ini terdiri dari

1. *Game Engine*, yaitu program yang bertanggung jawab atas keberjalanan permainan sesuai dengan aturan permainan.
2. *Game Runner*, yaitu program yang bertanggung jawab atas keberlangsungan permainan sehingga perintah yang dipanggil bot dapat dieksekusi dengan baik oleh *game engine.*
3. *Reference bot*, yaitu bot referensi yang berisi contoh penggunaan perintah-perintah dasar beserta contoh strategi yang digunakan untuk memenangkan permainan. Bot ini dapat digunakan untuk menguji bot yang telah dibuat.
4. *Starter bot*, yaitu bot yang akan diolah dan digunakan pada tugas ini.

Direktori *starter bot* memiliki struktur seperti berikut.

|--- bot.json

|--- src

```
|--- main
  |--- {package_directories}
    |--- Bot.java
    |--- Main.java
    |--- entities
      |--- Car.java
      |--- GameState.java
      |--- Lane.java
      |--- Position.java
    |--- enums
      |--- Powerups.java
      |--- Direction.java
                        |--- State.java
                        |--- Terrain.java
              |--- command
                        |--- AccelerateCommand.java
                        |--- BoostCommand.java
                        |--- ChangeLaneCommand.java
                        |--- Command.java
                        |--- DecelerateCommand.java
                        |--- DoNothingCommand.java
                        |--- OilCommand.java
```

2.2.2. Cara Kerja Bot

Ketika permainan dijalankan, program akan membaca *game-runner-config.json* untuk mengetahui informasi-informasi tertentu, seperti lokasi luaran *match-log* dan lokasi *bot.json* masing-masing bot. *File bot.json* berisi informasi-informasi bot seperti nama bot, lokasi bot, dan bahasa yang digunakan untuk membuat bot tersebut. Selanjutnya, kedua bot akan dijalankan.

Untuk setiap ronde, bot akan melakukan langkah-langkah berikut.

1. Bot akan membaca nomor ronde melalui stdin.
2. Bot akan membaca *state.json* yang berisi *map* dari permainan tersebut dan keadaan dari masing-masing bot.
3. Bot akan menentukan langkah selanjutnya berdasarkan keadaan yang telah dibaca bot.
4. Bot akan mengeluarkan luaran melalui stdout dengan format C;<nomor ronde>;<*command*>.

Untuk mengimplementasikan algoritma greedy ke dalam bot, sunting *file Bot.java* yang ada di dalam folder *starter-bot*. *File* tersebut berisi kelas Bot yang di dalamnya akan diisi logika dan strategi yang nantinya akan dioperasikan. Bot dapat memanggil beberapa class command yang terletak di dalam folder command dan juga mengakses entitas yang diperlukan seperti keadaan mobil bot, status musuh, posisi kedua bot, keadaan jalur, dan kondisi game. File *Bot.java* nantinya akan mengeksekusi algoritma yang diimplementasikan kemudian diakhir akan mengembalikan sebuah *command*/perintah yang akan menentukan aksi dari mobil bot pemain pada ronde berikutnya.

Untuk menjalankan *game*, jalankan *file run.bat* dalam direktori *game-runner* untuk Sistem Operasi Windows, atau jalankan *command* **make** pada terminal di dalam direktori *game-runner* untuk Sistem Operasi Linux.

Bab III
Aplikasi Strategi Greedy

3.1. *Mapping* Persoalan menjadi Elemen-elemen Algoritma Greedy

- Himpunan Kandidat

  Himpunan kandidat pada persoalan ini ialah *command-command* yang dapat dilakukan oleh bot, antara lain NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET, USE_LIZARD.

- Himpunan Solusi

  Himpunan solusi pada persoalan ini berisi kemungkinan urutan pemilihan *command* oleh bot untuk mengalahkan bot lawan.

- Fungsi solusi

  Fungsi solusi pada persoalan ini menentukan apakah urutan *command* pada himpunan solusi sudah membuat bot mencapai garis akhir dan mengalahkan bot lawan.

- Fungsi seleksi

  Fungsi seleksi pada persoalan ini ialah memilih *command* yang paling menguntungkan dalam setiap langkah.

- Fungsi kelayakan

  Fungsi kelayakan pada persoalan ini ialah menentukan apakah suatu *command* yang paling menguntungkan tersebut memungkinkan atau tidak. Jika tidak, akan dipilih prioritas pilihan setelahnya hingga ditemukan pilihan yang memungkinkan.

- Fungsi obyektif

  Fungsi obyektif pada persoalan ini ialah meminimasi ronde yang dilalui bot agar dapat mencapai garis akhir secepat mungkin.

3.2. Alternatif Solusi Greedy pada Persoalan

Algoritma dapat dibagi menjadi beberapa bagian, di mana bagian-bagian tersebut memiliki tingkat prioritas yang berbeda-beda.

- Mempercepat bot
- Menghindari *obstacle*
- Mengambil *power up*
- Memperbaiki mobil
- Menggunakan *power up* untuk keuntungan bot sendiri

Dari pembagian ini, terdapat 5!=120 kemungkinan algoritma greedy jika pembagian prioritas ini tidak dibagi-bagi lagi.

3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi Greedy

Berdasarkan pembagian di atas, memperbaiki mobil merupakan hal yang seharusnya paling diprioritaskan, karena jika *damage* yang diterima mobil lebih besar dari 4, mobil tidak dapat berjalan. Akan tetapi, memperbaiki mobil ketika *damage* yang diterima mobil tidak begitu berat tidak terlalu efisien.

Selanjutnya ialah menghindari *obstacle*, karena *obstacle* akan memberikan *damage* kepada mobil. Setelah menghindar dari *obstacle*, terdapat tiga opsi tersisa. Dari ketiga opsi tersebut, mengambil *power up* merupakan opsi yang optimal karena pengambilan *power up* hanya bisa dilakukan ketika di *lane* yang dapat dilalui terdapat *power up*. Kemudian, menggunakan *power up* akan diprioritaskan dibandingkan mempercepat bot di *lane* kosong,

karena *power up* tidak selalu dimiliki bot. Dengan kata lain, dari 120 kemungkinan algoritma *greedy* di atas, algoritma yang paling efisien ialah

- Memperbaiki mobil
- Menghindari *obstacle*
- Mengambil *power up*
- Menggunakan *power up*
- Mempercepat mobil

Akan tetapi, algoritma ini masih belum cukup efisien jika tidak membagi-bagi lagi pembagian prioritas algoritma. Keefektifan dapat ditingkatkan dengan membagi bagian "memperbaiki mobil" menjadi "memperbaiki mobil dengan *damage* berat" dan "memperbaiki mobil dengan *damage* rigan", dan "menggunakan *power up*" dapat dibagi menjadi "menggunakan EMP", "menggunakan *tweet*", "menggunakan *oil*", "Menggunakan *boost*" dan "menggunakan *lizard*", karena kondisi efektif penggunaan masing-masing *power up* berbeda-beda.

3.4. Strategi Greedy yang Dipilih

Beberapa pembagian prioritas di atas dibagi-bagi lagi sehingga berikut adalah urutan prioritas yang perlu dilakukan bot.

- Jika *damage* yang diterima lebih besar atau sama dengan 4 (sangat berat), bot akan memperbaiki mobilnya.
- Jika *speed* tidak lebih dari 3 (sangat lambat), bot akan mempercepat mobil
    - Jika ada *boost*, gunakan *boost*
    - Jika tidak ada, gunakan *accelerate*
- Jika terdapat *obstacle* pada *lane*, bot akan menghindar dari *obstacle* jika bisa
    - Jika ada *lizard*, gunakan *lizard*
    - Jika *lane* 2 atau 3 tidak terdapat *obstacle* dan mobil dapat berpindah ke *lane* tersebut, pindah ke *lane* tersebut
    - Jika *lane* 1 atau 4 tidak terdapat *obstacle* dan mobil dapat berpindah ke *lane* tersebut, pindah ke *lane* tersebut
    - Jika *obstacle* tidak dapat dihindari, pindah ke *lane* yang memiliki *power up*
- Jika pada *lane* sebelah terdapat *power up*, pindah *lane* untuk mengambil *power up*
- Jika memiliki EMP, berada di belakang lawan dan maksimal perbedaan *lane* ialah 1, gunakan EMP
- Jika *damage* mobil lebih besar sama dengan 3 (cukup parah) atau *damage* mobil ringan namun mobil akan mencapai *max speed*, perbaiki mobil.
- Jika bot berada di depan lawan dan memiliki *tweet*, gunakan *tweet*
- Jika bot berada di depan lawan dan memiliki *oil, gunakan oil*
- Gunakan *accelerate*

Pada dasarnya, prioritas bot ini ialah meminimalisasi *damage* pada mobil, karena *damage* akan mengurangi kecepatan maksimum mobil. Hal ini dapat dilihat dari prioritas bot untuk memperbaiki mobil dan menghindar dari *obstacle*. Jika *obstacle* tidak dapat dihindari, bot akan memilih *lane* yang menguntungkan, yaitu *lane* yang memiliki *power up* paling banyak. Lalu, bot

juga memprioritaskan kecepatan, bukan berarti mobil bot harus memiliki kecepatan maksimum, namun mobil bot tidak boleh terlalu pelan. Hal ini diperlukan karena sesuatu yang perlu dimiliki saat balapan ialah kecepatan yang besar (tidak harus tercepat) dan stabil.

Penggunaan EMP sangat diprioritaskan apabila tidak ada *power up* yang dapat diambil, karena pada permainan ini EMP merupakan *power up* yang sangat kuat karena dapat menembakkan ke depan maksimal di tiga *lane* dengan jarak tak terhingga. Karena jangkauannya yang luas, EMP diprioritaskan di atas perbaikan mobil dengan *damage* menengah.

Bab IV
Implementasi dan Pengujian

4.1. Implementasi Algoritma *Greedy* pada Program Bot

Implementasi dalam bentuk kode program *bot* dalam *game engine* yang digunakan dilakukan pada *file bot.java* yang tersedia pada *starter-pack game* Overdrive. Berikut isi dari *file bot.java* tersebut setelah diimplementasikan algoritmanya:

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

import za.co.entelect.challenge.command.*;
import za.co.entelect.challenge.entities.Car;
import za.co.entelect.challenge.entities.GameState;
import za.co.entelect.challenge.entities.Lane;
import za.co.entelect.challenge.enums.PowerUps;
import za.co.entelect.challenge.enums.Terrain;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

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();
  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);

  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() {
    // myCar Position
    int carLane = myCar.position.lane;
    int carBlock = myCar.position.block;

    // opponent Position
    int oppLane = opponent.position.lane;
    int oppBlock = opponent.position.block;

    // Get blocks information front and sides (if exist)
    List<Object> blocks = getBlocksMax(carLane, myCar.position.block); // Get information for
15 blocks
    List<Object> nextBlocks = getBlocksReach(carLane, myCar.position.block); // Get
information for 9 blocks

    // Default initialization, unused if Lane does not exist
    List<Object> blocksRight = getBlocksMax(carLane, myCar.position.block);
    List<Object> blocksLeft = getBlocksMax(carLane, myCar.position.block);
    List<Object> nextBlocksRight = getBlocksReach(carLane, carBlock);
    List<Object> nextBlocksLeft = getBlocksReach(carLane, carBlock);
    if (carLane != 4) {
      blocksRight = getBlocksMax(carLane+1, carBlock);
      nextBlocksRight = getBlocksReach(carLane+1, carBlock);
    }
    if (carLane != 1) {
      blocksLeft = getBlocksMax(carLane-1, carBlock);
      nextBlocksLeft = getBlocksReach(carLane-1, carBlock);
    }

    boolean canTurnRight = (carLane != 4); // If there's a lane on the right
    boolean canTurnLeft = (carLane != 1); // If there's a lane on the left
    boolean isAhead = carBlock > oppBlock; // If myCar is ahead of opponent
    boolean isBehind = !isAhead; // If myCar is behind opponent
    boolean isFarFromEachOther = (carBlock - oppBlock) * (carBlock - oppBlock) > 2500; //
myCar and opponent is more than 50 blocks apart
    boolean onEMPRange = carBlock < oppBlock && (carLane - oppLane) * (carLane -
oppLane) < 4;

    // If car is too damaged (and can't move quickly)
```

```
    // -> FIX
    if (myCar.damage >= 4){
      return FIX;
    }

    // If car is moving too slow
    // -> Use BOOST or ACCELERATE
    if (myCar.speed <= 3){
      // If the lane is clear, not too damaged, and have BOOST PowerUp
      if (hasPowerUp(PowerUps.BOOST, myCar.powerups) && myCar.damage < 3 &&
isClear(blocks)){
        return BOOST;
      } else {
        return ACCELERATE;
      }
    }

    // BOOST CAN BE USED MAXIMALLY
    // Current lane is clear, car is undamaged, boost is running out / inactive, have BOOST
    // -> Use BOOST
    if (isClear(blocks) && myCar.damage == 0 && hasPowerUp(PowerUps.BOOST,
myCar.powerups) && myCar.boostCounter <= 1) {
      return BOOST;
    }

    // MOVING TO EMPTY LANE
    // If car is boosting, current lane has obstacle
    // -> Turn to empty Lane, prioritizing going to Lane 2 or 3
    if (myCar.boostCounter > 1 && !isClear(blocks)){
      if (isClear(blocksRight) && canTurnRight){
        if (carLane == 3 && isClear(blocksLeft)){
          return TURN_LEFT;
        } else {
          return TURN_RIGHT;
        }
      } else if (canTurnLeft && isClear(blocksLeft)){
        return TURN_LEFT;
      }
    }

    // DODGING OBSTACLE USING LIZARD
    // If car is boosting, current lane has obstacle, (adjacent lanes have obstacle), have LIZARD
   // -> Use LIZARD
    if (myCar.boostCounter > 1 && hasPowerUp(PowerUps.LIZARD, myCar.powerups)){
      if (!blocks.contains(Terrain.FINISH)){
        if (!isClear(blocks.subList(0, blocks.size()-1))){
```

```
        return LIZARD;
      }
    } else if (!isClear(blocks)){
      return LIZARD;
    }
  }

  // MOVING TO EMPTY LANE
  // If car is not boosting, current lane has obstacle, (adjacent lanes have obstacle)
  // -> Turn to empty Lane, prioritizing Lane 2 or 3
  if (myCar.boostCounter <= 1 && !isClear(nextBlocks)){
    if (isClear(nextBlocksRight) && canTurnRight){
      if (isClear(nextBlocksLeft) && carLane == 3){
        return TURN_LEFT;
      } else {
        return TURN_RIGHT;
      }
    } else if (isClear(nextBlocksLeft) && canTurnLeft){
      return TURN_LEFT;
    }
  }

  // MOVING TO HIGHER VALUE LANE
  // If current lane is clear but doesn't have any PowerUp to pick up
  // -> Go to empty Lane with PowerUp, prioritizing Lane 2 or 3
  if (isClear(blocksRight) && canTurnRight){
    if (isClear(blocksLeft) && carLane == 3){
      if (isAhead && containsPowerUp1(nextBlocksLeft) && !containsPowerUp1(nextBlocks))
{
        return TURN_LEFT;
      } else if (isBehind && containsPowerUp2(nextBlocksLeft) &&
!containsPowerUp2(nextBlocks)){
        return TURN_LEFT;
      }
    } else {
      if (isAhead && containsPowerUp1(nextBlocksRight) &&
!containsPowerUp1(nextBlocks)) {
        return TURN_RIGHT;
      } else if (isBehind && containsPowerUp2(nextBlocksRight) &&
!containsPowerUp2(nextBlocks)){
        return TURN_RIGHT;
      }
    }
  } else if (isClear(blocksLeft) && canTurnLeft){
    if (isAhead && containsPowerUp1(nextBlocksLeft) && !containsPowerUp1(nextBlocks)) {
      return TURN_LEFT;
```

```
    } else if (isBehind && containsPowerUp2(nextBlocksLeft) &&
!containsPowerUp2(nextBlocks)){
      return TURN_LEFT;
    }
  }

  // DODGING OBSTACLE USING LIZARD
  // If car speed = 9, have LIZARD, current Lane has obstacle
  // -> Use LIZARD
  if (myCar.boostCounter <= 1 && myCar.speed > 8 && hasPowerUp(PowerUps.LIZARD,
myCar.powerups)){
    if (!nextBlocks.contains(Terrain.FINISH)){
      if (!isClear(nextBlocks.subList(0, nextBlocks.size()-1))){
        return LIZARD;
      }
    } else if (!isClear(nextBlocks)){
      return LIZARD;
    }
  }

  // MOVING TO HIGHER VALUE LANE
  // If current lane has obstacle and doesn't have any PowerUp to pick up
  // -> Turn to lane with PowerUp, prioritizing Lane 2 or 3
  if (!(containsPowerUp1(nextBlocks)) && !isClear(nextBlocks) && !myCar.boosting){
    if (isAhead) {
      if (canTurnRight && containsPowerUp1(nextBlocksRight)){
        if (carLane == 3 && containsPowerUp1(nextBlocksLeft)){
          return TURN_LEFT;
        } else {
          return TURN_RIGHT;
        }
      } else if (canTurnLeft && containsPowerUp1(nextBlocksLeft)){
        return TURN_LEFT;
      }
    } else {
      if (canTurnRight && containsPowerUp2(nextBlocksRight)){
        if (carLane == 3 && containsPowerUp2(nextBlocksLeft)){
          return TURN_LEFT;
        } else {
          return TURN_RIGHT;
        }
      } else if (canTurnLeft && containsPowerUp2(nextBlocksLeft)){
        return TURN_LEFT;
      }
    }
  } else if (!(containsPowerUp1(blocks)) && !isClear(blocks) && myCar.boosting){
```

```
    if (isAhead) {
      if (canTurnRight && containsPowerUp1(blocksRight)){
        if (canTurnLeft && containsPowerUp1(blocksLeft)){
          return TURN_LEFT;
        } else {
          return TURN_RIGHT;
        }
      } else if (canTurnLeft && containsPowerUp1(blocksLeft)){
        return TURN_LEFT;
      }
    } else {
      if (canTurnRight && containsPowerUp2(blocksRight)){
        if (canTurnLeft && carLane == 3 && containsPowerUp2(blocksLeft)){
          return TURN_LEFT;
        } else {
          return TURN_RIGHT;
        }
      } else if (canTurnLeft && containsPowerUp2(blocksLeft)){
        return TURN_LEFT;
      }
    }
  }

  // USING EMP
  // If have EMP, behind enemy, current Lane is equal or adjacent to enemy Lane, enemy
moving faster or equal to speed 6
  // -> Use EMP
  if (hasPowerUp(PowerUps.EMP, myCar.powerups) && onEMPRange && opponent.speed >=
6){
    return EMP;
  }

  // Car is still damaged by more than 3 -> FIX
  if (myCar.damage >= 3){
    return FIX;
  }

  // Car is still damaged, and is almost or going at max speed of 9 or it can boost
  // -> FIX
  if (myCar.damage != 0 && (myCar.speed >= 8 || hasPowerUp(PowerUps.BOOST,
myCar.powerups))){
    return FIX;
  }

  // USING TWEET
```

```
    // If car moving at max speed or more, have TWEET, far away from each other or placed behind (to prevent tweet from backfiring)
    // -> Use TWEET depending on enemy speed and location
    if (myCar.speed >= 9 && hasPowerUp(PowerUps.TWEET, myCar.powerups) && (isFarFromEachOther || carBlock > oppBlock+16)){
      if (opponent.speed > 9) {
        return new TweetCommand(oppLane, oppBlock+16);
      } else if (opponent.speed == 9 && opponent.damage == 1){
        return new TweetCommand(oppLane, oppBlock+10);
      } else if (opponent.speed == 8 && opponent.damage == 2){
        return new TweetCommand(oppLane, oppBlock+9);
      } else if (opponent.speed == 3){
        return new TweetCommand(oppLane, oppBlock+7);
      } else if (opponent.speed == 6){
        return new TweetCommand(oppLane, oppBlock+9);
      } else if (opponent.speed == 8){
        return new TweetCommand(oppLane, oppBlock+10);
      } else if (opponent.damage == 5){
        return new TweetCommand(oppLane, oppBlock+1);
      }
    }

    // USING OIL
    // If car has OIL PowerUp and a bit far ahead of enemy
    // -> Use OIL PowerUp
    if (hasPowerUp(PowerUps.OIL, myCar.powerups) && (carBlock - oppBlock) > 16){
      return OIL;
    }

    // None of the condition above is met
    // -> simply Accelerate
    return ACCELERATE;
  }

  /**
   * Returns map of blocks and the objects in the for the current lanes, returns the amount of blocks that can be
   * traversed at max speed.
   **/
  // Get information for 15 blocks of a lane
  private List<Object> getBlocksMax(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);
```

```
    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);
      }
    return blocks;
  }
  // Get information for 9 blocks of a lane
  private List<Object> getBlocksReach(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);
    for (int i = max(block - startBlock, 0); i <= block - startBlock + maxSpeed; i++) {
      if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) {
        break;
      }

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

      }
    return blocks;
  }
  // Check if range of blocks of a lane is clear of obstacles (excluding Cyber Truck)
  private boolean isClear(List<Object> blocks){
    return !(blocks.contains(Terrain.MUD) || blocks.contains(Terrain.OIL_SPILL) ||
blocks.contains(Terrain.WALL));
  }
  // Check if range of blocks of a lane contain PowerUp LIZARD or BOOST
  private boolean containsPowerUp1(List<Object> blocks) {
    return (blocks.contains(Terrain.LIZARD) || blocks.contains(Terrain.BOOST));
  }
  // EMP only useful when myCar is behind enemy, check if range of blocks of a lane contain
PowerUp EMP, LIZARD, or BOOST
  private boolean containsPowerUp2(List<Object> blocks){
    return (blocks.contains(Terrain.EMP) || blocks.contains(Terrain.LIZARD) ||
blocks.contains(Terrain.BOOST));
  }

  // Check if car has the PowerUp selected
  private Boolean hasPowerUp(PowerUps powerUpToCheck, PowerUps[] available) {
    for (PowerUps powerUp: available) {
      if (powerUp.equals(powerUpToCheck)) {
        return true;
```

```
    }
   }
   return false;
  }

}
```

4.2. Penjelasan Struktur Data

Bot ini memiliki struktur data seperti berikut

a. Kelas Bot

Kelas bot berisi beberapa atribut-atribut berikut

- maxSpeed (int), menyatakan kecepatan maksimum yang dapat ditempuh bot
- directionList (List<Integer>), senarai yang berisi -1 dan 1 yang masing-masing menandakan arah mobil (-1 untuk belok kiri, 1 untuk belok kanan)
- gameState (GameState), berisi *state* permainan
- Opponent (Car), merupakan data mobil lawan
- myCar (Car), merupakan data mobil pemain
- FIX (Command), merupakan objek *Command* FIX
- ACCELERATE(Command), merupakan objek *Command* ACCELERATE
- LIZARD(Command), merupakan objek *Command* LIZARD
- OIL(Command), merupakan objek *Command* OIL
- BOOST(Command), merupakan objek *Command* BOOST
- EMP(Command), merupakan objek *command* EMP
- TURN_RIGHT(Command), merupakan objek *command* TURN_RIGHT
- TURN_LEFT(Command), merupakan objek *command* TURN_LEFT

Kelas bot juga memiliki metode-metode berikut

- Bot(), merupakan *constructor* kelas Bot
- run(), merupakan metode untuk menjalankan bot yang telah berisi logika-logika bot
- getBlocksMax(), mengembalikan senarai informasi dalam suatu jarak *blocks* dari suatu *lane* sebesar 15 *blocks* depan *bot*
- getBlocksReach(), mengembalikan senarai informasi dalam suatu jarak *blocks* dari suatu *lane* sebesar 9 *blocks* depan *bot*
- isClear(*lane*), menghasilkan *true* jika *lane* bersih dari *obstacle*
- containsPowerUp1(), menghasilkan *true* jika pada suatu jarak *blocks* dari suatu *lane* terdapat *power up lizard* atau *boost*.
- containsPowerUp2(), menghasilkan *true* jika pada suatu jarak *blocks* dari suatu *lane* terdapat *power up EMP, lizard* atau *boost*
- hasPowerUp(*powerUp,powerUpList*), akan menghasilkan *true* jika terdapat *powerUp* pada *powerUpList*

b. Kelas Main

Kelas Main hanya berisi atribut ROUNDS_DIRECTORY yang menyimpan direktori ronde dan STATE_FILE_NAME yang menyimpan nama file *state*, serta satu konstruktor. Kelas ini digunakan untuk membaca *state* saat ini, menginformasikan ke bot, lalu bot akan memberikan *output* dan kelas ini akan mencetak *output* tersebut ke stdout.

c. Direktori Command

Direktori ini berisi kelas Command dan anak-anaknya, antara lain

- AccelerateCommand
- BoostCommand
- ChangeLanecommand
- DecelerateCommand
- DoNothingcommand
- EmpCommand
- FixCommand
- LizardCommand
- OilCommand
- TweetCommand

Masing-masing subkelas hanya memiliki satu metode, yaitu render(), yang berguna untuk menerjemahkan perintah bot menjadi *string* yang akan dikeluarkan oleh kelas Main di stdout.

d. Direktori Entities

Direktori ini berisi empat kelas, yaitu

- Car, kelas yang menerjemahkan informasi mobil
- GameState, kelas yang menerjemahkan keadaan *game* (seperti ronde dan map)
- Lane, kelas yang menerjemahkan keadaan *lane*
- Position, kelas yang menerjemahkan posisi

Keempat kelas ini akan menerjemahkan data-data dari *state.json*/*game_state.json* menjadi data-data yang valid dalam java.

e. Direktori Enums

Direktori ini berisi empat kelas, yaitu

- Direction, kelas yang menerjemahkan arah dari kumpulan data menjadi kata yang mudah dibaca
- PowerUps, kelas yang menerjemahkan nama-nama *power ups*
- State, kelas yang menerjemahkan semua keadaan yang mungkin dari suatu mobil
- Terrain, kelas yang menerjemahkan semua status yang mungkin dari suatu *tile*.

4.3. Analisis dari Desain Solusi Algoritma *Greedy*

Secara garis besar, implementasi pada setiap pengujian yang dilakukan sudah berhasil mendekati nilai optimal. Dalam 1500 *blocks* yang perlu dilalui oleh *bot* serta rintangannya, *bot* yang dibuat dapat menempuhnya dalam sekitar 120-140 ronde. Artinya, rata-rata, *bot* bergerak sekitar sebanyak 11.5 blocks per ronde. Pada saat yang sama, ketika bot tidak bisa bergerak lebih cepat lagi, *bot* akan menggunakan *power up offensive*-nya untuk memperlambat musuh. Selain itu, *bot* akan mencoba untuk mendapatkan nilai yang optimal untuk ronde selanjutnya jika saat tersebut, *bot* tidak dapat bergerak lebih cepat lagi. Implementasi kode kami juga memasukkan pemilihan *lane* yang memiliki keuntungan yang lebih besar dibanding yang lainnya, dengan prioritas bergerak di *lane* yang tidak memiliki rintangan.

Namun, *bot* yang telah kami buat tidak dapat mendeteksi *Cyber Truck* yang dipasang oleh lawan. Jika, *bot* lawan menggunakan *power up tweet* pada *lane* kosong *bot*, *bot* kami tidak dapat menghindar dari *Cyber Truck* tersebut secara sengaja. Selain itu, perhitungan nilai setiap *lane* kurang akurat karena tidak membedakan jumlah rintangan pada 2 *lane* atau lebih yang memiliki rintangan. Jadi, *bot* mungkin saja berbelok ke *lane* yang lebih merugikan jika lokasi *lane bot* pada saat tertentu dan *lane* sebelahnya memiliki rintangan.

Bab V
Kesimpulan dan Saran

5.1. Kesimpulan

Algoritma greedy sejatinya merupakan algoritma yang digunakan untuk memecahkan suatu persoalan secara langkah per langkah yang menggunakan pendekatan penyelesaian masalah dengan mencari nilai optimum pada setiap langkahnya. Algoritma ini memegang prinsip "take what you can get now" yang memaksimalkan pengambilan keputusan pada langkah yang sedang dihadapi tanpa memperhatikan konsekuensi pada langkah berikutnya. Harapannya, algoritma dapat berakhir dengan solusi yang merupakan optimum global. Algoritma greedy dapat diterapkan sebagai strategi bermain pada berbagai macam permainan salah satunya yaitu permainan Overdrive yang menjadi objek utama pada tugas besar ini.

Meskipun keputusan pada suatu langkah dirasa telah optimal, algoritma ini tidak menjamin solusi yang dihasilkan merupakan solusi yang optimum secara global. Faktanya, terdapat berbagai macam strategi dalam pengambilan keputusan demi mencapai syarat optimal pada setiap langkahnya. Hal tersebut dapat terlihat pada permainan Overdrive yang telah kami kerjakan. Walaupun strategi yang diterapkan pada tiap langkah sudah berjalan dengan baik, hal ini tidak menjamin suatu kemenangan pada setiap pertandingan yang diselenggarakan.

5.2. Saran

Pengerjaan kelompok seharusnya dapat dilakukan dengan lebih menekankan pemahaman dasar terlebih dahulu terhadap permasalahan yang diberikan. Untuk kelompok yang akan mengerjakan hal serupa, alangkah lebih baik jika pengerjaan diawali dengan membaca dan memahami bahan-bahan yang sekiranya akan dibutuhkan dalam pengerjaan. Hal ini dilakukan dari jauh-jauh hari sebelum melakukan implementasi algoritma. Dengan itu, pembuatan algoritma pada *bot* dapat dilaksanakan lebih mudah dan teratur.

Selain itu, kode yang kami buat seharusnya bisa lebih dibuat lebih rapi lagi. Pahami terlebih dahulu *method* dan *variable* atau *state* yang tersedia pada *starter-pack* sebelum memulai. Rancang terlebih dahulu atau *brainstorming* dengan kelompok apa saja *fungsi* dan algoritma yang akan dibuat pada program. Terakhir, lakukan tes melawan *reference bot*, *bot* sendiri, dan *bot* orang lain supaya dapat mengetahui lebih mudah apa saja yang perlu diperbaiki.

Link Repository Github

https://github.com/DeeGeeDow/Wheel-of-Fortune

Daftar Pustaka

Algoritma Greedy (itb.ac.id)

Entelect Challenge 2020-Overdrive Game Rules (github.com)