LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

Disusun oleh:

Kelompok 33 Setima

13520051 – Flavia Beatrix Leoni A. S.

13520057 – Marcellus Michael Herman K

13520087 – Dimas Shidqi Parikesit

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

Daftar Isi

BAB 1
Deskripsi Tugas

Overdrive adalah sebuah *game* yang mempertandingkan 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.

Beberapa aturan umum dari *game* ini 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.

BAB 2

Landasan Teori

2.1 Dasar Teori

Algoritma Greedy adalah algoritma yang membentuk solusi langkah per langkah dengan mencari nilai maksimum pada seitap langkahnya. Keputusan atas solusi yang diambil tidak memperhatikan keputusan selanjutnya yang akan diambil. Keputusan yang telah diambil, tidak dapat dikembalikan.

Prinsip utama algoritma Greedy adalah "take what you can get now" atau ambil yang dapat diambil saat ini. Arti dari prinsip ini adalah pada setiap langkah algoritma Greedy, setiap langkah tersebut merupakan solusi optimum lokal. Dengan mengambil langkah optimum lokal, diharapkan solusi akhir yang terbentuk dapat menjadi optimum global, yaitu solusi optimum yang melibatkan setiap langkah dari awal sampai akhir.

Berikut merupakan elemen-elemen yang menggambarkan sebuah Algoritma Greedy.

1. Himpunan Kandidat (C)

   Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah, misalnya simpul sisi di dalam graf, job, task, koin, benda, karakter.

2. Himpunan Solusi (S)

   Himpunan yang berisi kandidat yang sudah dipilih.

3. Fungsi Solusi

   Fungsi yang digunakan untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi paling optimal.

4. Fungsi Seleksi (Selection Function)

   Fungsi yang digunakan untuk memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.

5. Fungsi kelayakan (Feasibility)

   Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).

6. Fungsi Obyektif

Fungsi yang digunakan untuk memaksimumkan atau meminimumkan solusi yang kita miliki.

2.2 Cara Kerja Program

2.2.1. Struktur Program Bot

Program bot ini dapat dibagi menjadi beberapa bagian:

1. Folder Command
   Berisi file AccelerateCommand.java, DecelerateCommand.java, DecelerateCommand.java, ChangeLaneCommand.java, DoNothingCommand.java, dan FixCommand.java yang menjalankan *basic command*.
   Command yang mengaktifkan *powerup* terdapat pada file BoostCommand.java, EmpCommand.java, LizardCommand.java, OilCommand.java, dan TweetCommand.java.
   Sementara itu, file Command.java berisi *interface* dari semua *command* yang lain.
2. Folder Entities
   Folder ini berisi definisi entitas game, yaitu *car* yang berada pada file Car.java, *Game State* yang berada pada file GameState.java, *lane* yang berada pada file Lane.java, dan *position* yang berada pada file Position.java.
3. Folder Enums
   Pada fitur game yang memiliki banyak variasi seperti *direction, power ups, state,* dan *terrain,* dilakukan enumerasi agar untuk menyederhanakan penggunaannya oleh pemain. Enumerasi ini diakukan pada file yang berada di folder ini.
4. File Bot.java
   File ini berisi implementasi algoritma pada bot
5. File Main,java
   File ini berisi *pipeline* untuk menjalankan bot, yaitu alur penggunaan bot yang berada pada file Bot.java

2.2.2. Alur Kerja Bot

Bot adalah sebuah objek yang diinstansiasi dari *Class Bot* yang berada pada file Bot.java. Pada class ini terdapat tiga bagian utama, yaitu atribut *private* yang berisi data internal, atribut *public* Bot yang berisi kondisi dari dunia, dan method *public* run yang mengeluarkan

output Command yang dipakai oleh bot pada round tersebut. Selain itu terdapat pula method yang dibuat untuk kebutuhan algoritma yang digunakan.

Class Bot kemudian digunakan pada file Main.java. Tahap pertama penggunaannya adalah dengan menginstansiasi objek Bot dari class Bot dengan melakukan passing gamestate dan nilai random. Kemudian dijalankan method run dari objek tersebut sehingga didapatkan command yang akan dijalankan bot pada round tersebut. Kemudian command yang didapatkan dari objek bot tersebut diprint ke terminal.

2.2.3. Cara Implementasi Algoritma Greedy pada Bot

Algoritma Greedy diimplementasikan pada class Bot bagian method run, meskipun apabila diperlukan dapat membuat atribut *private* dan method public lainnya untuk memodularisasi kode.

2.2.4. Cara Menjalankan Game Engine

Terdapat dua tahap untuk menjalankan game dan bot yang telah dibuat. Pertama adalah pembuatan build dari bot. Untuk membuat build dari bot pertama harus melakukan build menggunakan file konfigurasi Maven, dimana Maven adalah salah satu build system untuk program Java. Build dapat dibuat dengan menggunakan *command* pada terminal

```
mvn clean install
```

atau dengan menggunakan fitur build pada IntelliJ IDEA.

Kemudian perlu dilakukan tahap kedua yaitu menjalankan game. Untuk menjalankan game diperlukan enam hal, yaitu hasil build kedua pihak, file game-config.json, file game-engine.jar, file game-runner-config.json, dan file game-runner-jar-with-dependencies.jar. Apabila ke-enam komponen tersebut ada, game dapat dijalankan dengan menjalankan file run.bat atau dengan mengetik *command* pada terminal

```
java -Dfile.encoding=UTF-8 -jar ./game-runner-jar-with-dependencies.jar
```

BAB 3

Aplikasi Strategi *Greedy*

3.1 Elemen-Elemen Algoritma *Greedy*

1. Himpunan kandidat

   Command-command yang dapat diberikan kepada mobil, meliputi *command* NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP, dan FIX.

2. Himpunan solusi

   Command terpilih yang dianggap paling mendesak untuk digunakan agar *bot* dapat memenangkan pertandingan.

3. Fungsi solusi

   Memeriksa apakah *command* yang dipilih dapat membuat mobil kami semakin dekat dengan *finish line* serta dapat menghambat mobil lawan dalam mencapai *finish line.*

4. Fungsi seleksi

   Memilih *command* yang dapat membuat mobil bergerak lebih cepat sesuai dengan kondisi yang ada. Jika mobil telah bergerak dengan cukup cepat, mobil akan melakukan penyerangan terhadap mobil lawan.

5. Fungsi kelayakan

   Memeriksa apakah *command* yang dipilih merupakan *command* yang valid dan bukan merupakan *command* TURN_LEFT ketika mobil berada di *lane* 1, *command* TURN_RIGHT ketika mobil berada di *lane* 4, atau *command* untuk menggunakan *power up* yang tidak dimiliki.

6. Fungsi objektif

   Command yang dipilih dapat memaksimumkan kecepatan mobil dan *score* yang dimiliki serta dapat meminiumkan potensi bertabrakan dengan *object* seperti mud, oil, wall, dan cybertruck.

3.2 Eksplorasi Alternatif Solusi *Greedy*

Dalam melakukan eksplorasi solusi, masing-masing orang membuat dasar bot versi sendiri. Kemudian dari ketiga bot tersebut diujikan terhadap bot lain untuk menentukan komponen mana yang akan diambil dari masing-masing bot sehingga akan didapatkan bot baru yang memiliki kemampuan terbaik dari bot yang lain. Berikut detail masing-masing bot yang telah dibuat

3.2.1. Alternatif Dimas

Fitur yang ada pada alternatif yang dibuat Dimas adalah prioritas penggunaan *command*, prosedur menghindari rintangan termasuk cyber truck, dan prosedur pemilihan jalur untuk mengambil powerup. Prioritas penggunaan command pertama adalah melakukan FIX apabila damage mobil mencapai lima atau lebih dari sama dengan dua dan kecepatannya sudah mencapai kecepatan maksimum, percepat jika terlalu lamban, hindari rintangan yang ada, gunakan tweet, gunakan emp, gunakan boost, gunakan oil, pindah jalur jika jalur sebelah ada powerup, dan default fallback yaitu akselerasi. Pada prosedur menghindari rintangan, prioritas utama adalah menggunakan lizard dan pindah ke jalur yang tidak memiliki rintangan apapun. Namun apabila semua jalur sekarang dan jalur sebelah memiliki rintangan, maka diutamakan pindah ke jalur yang rintangan nya bukan Wall ataupun Cyber Truck. Apabila tidak ada solusi yang baik maka tidak akan pindah ke jalur manapun. Prosedur pindah jalur untuk mengambil powerup dilakukan apabila tidak ada hal yang dilakukan tetapi jalur sebelah tidak ada rintangan dan ada powerup maka akan berpindah jalur.

3.2.2. Alternatif Leoni

Pada alternatif ini, bot akan selalu berusaha untuk memiliki kecepatan yang tinggi dengan cara memperbaiki mobilnya jika kerusakan mobil lebih dari satu dan melakukan accelerate jika kecepatannya kurang dari 4. Bot juga akan berusaha mendapatkan powerup boost tanpa memperhatikan obstacle yang ada di jalurnya. Jika tidak ada powerups boost yang dapat diambil, bot akan menghindari obstacle, seperti mud, wall, dan oil spill yang ada di jalur tersebut dengan cara berpindah ke jalur lain atau menggunakan powerup lizard. Namun, powerup lizard diutamakan untuk menghindari mud. Apabila tidak terdapat obstacle pada jalur tersebut, bot akan menggunakan powerups lain yang dimilikinya. Penggunaan powerups diprioritaskan untuk mempercepat mobil dengan menggunakan boost, kemudian barulah bot akan melakukan penyerangan terhadap mobil lawan, yaitu dengan menggunakan powerups emp, tweet, dan oil.

3.2.3. Alternatif Michael

Pada alternatif yang diusulkan oleh Michael, dia menggunakan pembobotan untuk setiap jalur yang dapat ditempuh oleh mobil. Semakin tinggi skor dari jalur tersebut, mobil akan cenderung untuk melewati jalur tersebut. Untuk setiap Power Ups, nilai dari pembobotan akan bertambah. Akan tetapi, untuk setiap Obstacles, nilai dari pembobotan akan berkurang. Jika ternyata nilai dari ketiga jalur tersebut di bawah 0, program akan mengecek apakah mobil memiliki Lizard atau tidak. Jika memiliki, mobil akan menggunakan Lizard. Jika tidak memiliki, mobil akan menuju jalur yang memiliki skor paling tinggi di antara ketiga jalur tersebut.

3.3 Analisis Efisiensi dari Alternatif Solusi *Greedy* yang Dirumuskan

3.3.1. Alternatif Dimas

Pada setiap runtime alternatif ini dijalankan empat kali looping, yaitu tiga kali melihat keberadaan cyber truck pada lane saat ini dan lane sebelah, ditambah dengan satu kali looping untuk melihat lane didepan. Kemudian apabila percabangan yang dipilih adalah prosedur menghindar atau mengambil powerup, maka akan dilakukan dua kali looping tambahan untuk melihat lane sebelah, sehingga dalam kasus terburuk nya, efisiensi dari alternatif ini apabila dituliskan dalam notasi Big O adalah O(6n) atau sama dengan O(n).

3.3.2. Alternatif Leoni

Implementasi dari alternatif solusi ini cukup sederhana karena pada setiap eksekusi, akan dilakukan satu kali perulangan untuk melihat kondisi di depan mobil. Jika bot memutuskan untuk berpindah ke jalur yang lain, akan dilakukan dua kali perulangan lagi untuk melihat kondisi di jalur kanan dan kiri mobil. Sementara jika bot ingin mengecek ada tidaknya powerups yang dimiliki, akan dilakukan satu kali perulangan lagi. Untuk kasus yang lain, hanya akan dilakukan pengecekan kondisi sederhana. Oleh karena itu, efisiensi dari alternatif ini dalam notasi Big O adalah O(n).

3.3.3. Alternatif Michael

Efisiensi dari alternatif yang diusulkan oleh Michael bisa dikatakan sebagai alternatif dengan efisiensi rendah, sebab untuk setiap langkahnya, harus dilakukan iterasi untuk kondisi di depan mobil sebanyak tiga kali, yaitu untuk jalur tepat di depan mobil, jalur di sebelah kanan mobil,

jalur di sebelah kiri mobil. Selain itu, untuk setiap langkah mobil bergerak, dilakukan pencarian Obstacle ataupun Power Ups dengan notasi Big O O(n).

3.4 Analisis Efektivitas dari Alternatif Solusi *Greedy* yang Dirumuskan

3.4.1 Alternatif Dimas

Alternatif ini cukup efektif untuk menghindari rintangan, menggunakan dan mengambil powerup. Akan tetapi, alternatif ini masih memiliki kekurangan yaitu strategi pengambilan keputusan dalam menghindar yang masih dapat diperbaiki. Pada alternatif ini, powerup Lizard selalu digunakan apabila pemain memiliki powerup tersebut. Akan tetapi, kenyataannya strategi itu tidak selalu menghasilkan keputusan yang terbaik untuk ronde itu. Perbaikan dari algoritma ini merupakan salah satu hal yang diperhatikan dalam membuat bot versi final.

3.4.2 Alternatif Leoni

Alternatif solusi ini cukup efektif dalam memenangkan permainan karena mobil akan selalu berusaha untuk memiliki kecepatan yang tinggi yang dapat membuatnya mencapai garis finish dengan lebih cepat. Bot juga akan menggunakan boost jika tidak ada obstacle lain di jalurnya dan tidak sedang boosting. Namun, alternatif ini kurang mampu memilih jalur yang paling aman ketika bot ingin berpindah ke jalur yang lain. Alternatif ini juga kurang efektif jika lawan sering menggunakan powerup tweet karena alternatif ini tidak memiliki strategi khusus dalam menghindari cybertruck.

3.4.3 Alternatif Michael

Efektivitas dari alternatif yang diusulkan oleh Michael cukup efektif untuk menghindari obstacle ataupun pencarian power ups. Akan tetapi, alternatif yang diusulkan oleh Michael tidak dapat menghindari keberadaaan Cyber Truck yang kerap kali justru menghalangi laju keberjalanan mobil. Selain itu, terkadang ada beberapa kondisi keputusan yang diambil bukanlah keputusan yang optimum. Misalkan ketika ada Mud pada ketiga jalur, mobil justru menggunakan Use Boost alih-alih melakukan tindakan biasa tanpa mengurangi Power Ups seperti Accelerate.

3.5 Strategi *Greedy* yang Dipilih

Kami mencoba untuk menggabungkan strategi dari alternatif yang diusulkan oleh masing-masing anggota hingga akhirnya menemukan hasil yang paling optimal dalam memenangkan permainan ini, yaitu dengan urutan prioritas perintah untuk memperbaiki mobil, menghindari cybertruck, memilih jalur yang memiliki powerups yang dapat diambil dan tidak memiliki obstacle, melakukan accelerate atau menghindari mud yang ada jika mobil bergerak dengan kecepatan kurang dari sama dengan tiga, menghindari obstacle lain, seperti mud, wall, dan oil spill jika mobil berjalan dengan cukup cepat, dan menggunakan powerups lain yang ada. Penggunaan powerups ini juga memiliki urutan prioritas, yaitu emp, boost, tweet, dan oil yang diurutkan dari powerups yang menurut kelompok kami paling berdampak dalam memenangkan permainan. Apabila tidak ada kondisi yang dipenuhi, bot akan melakukan accelerate. Strategi kami tidak menggunakan command nothing dan decelerate karena kami menganggap command tersebut kurang berguna dalam memenangkan permainan ini.

Strategi greedy yang kami pilih untuk diimplementasikan adalah strategi yang berusaha untuk memaksimalkan kecepatan dan memilih jalur yang paling menguntungkan dalam arti mengandung obstacle seminimal mungkin atau memiliki powerups untuk diambil yang lebih menguntungkan. Pemilihan jalur ini dilakukan dengan metode pembobotan tiap jalur yang ada, seperti yang digunakan pada alternatif solusi Michael.

Strategi ini merupakan strategi yang paling efektif menurut kami karena kami berusaha memadukan kelebihan dari masing-masing alternatif solusi dan mengatasi permasalahan yang kurang dapat ditangani oleh masing-masing alternatif. Strategi ini telah dapat memilih jalur yang paling tepat dan menghindari cybertruck yang tentunya berdampak cukup besar terhadap kecepatan mobil. Strategi ini juga dapat memilih kondisi yang paling tepat dalam menggunakan powerups yang dimiliki.

Kecepatan mobil merupakan salah satu hal yang paling penting dalam memenangkan permainan sehingga kecepatan mobil menjadi prioritas utama dalam strategi yang kami pilih. Terdapat beberapa faktor yang mempengaruhi kecepatan mobil dalam permainan ini, yaitu damage mobil dan pemilihan jalur yang tepat. Strategi kami selalu berusaha agar mobil memiliki damage yang kecil serta kecepatan yang tinggi. Pemilihan jalur yang tepat menjadi penting untuk diperhatikan karena dengan memilih jalur yang tepat, mobil akan memiliki potensi yang lebih kecil dalam menabrak obstacle yang dapat menambah damage mobil dan menurunkan kecepatan mobil. Selain itu, pemilihan jalur yang tepat juga membuat bot kami

memiliki score yang cukup tinggi karena mengurangi kemungkinan bertabrakan dengan mud dan oil spill serta menambah kemungkinan dalam memperoleh powerup yang berpengaruh terhadap score yang dimiliki.

BAB 4

Implementasi dan Pengujian

4.1 Implementasi Algoritma *Greedy*

Implementasi algoritma Greedy pada permainan ini terdapat pada file Bot.java. Berikut adalah notasi algoritma dari algoritma Greedy yang telah kami buat.

4.1.1 Public Command Run

```
function run(GameState: gameState) → Command
{ Fungsi utama dalam menentukan command apa yang akan dilakukan oleh mobil }

KAMUS LOKAL
        truckExists, truckLeft, truckRight : boolean
        myCar, opponent : Car
        result : int

ALGORITMA
        { Kalau damage >=2 difix karena terlalu lamban }
    if myCar.damage >= 2 and myCar.speed==maxSpeed then
        → FIX

        { Cek apakah ada truck, kalau ada hindari atau loncati pakai lizard }
    if truckExists or truckLeft or truckRight then
        result ← doTurn(myCar, gameState)
        if result > 5 then
            if hasPowerUp(PowerUps.LIZARD, myCar.powerups) then
                → LIZARD
            else
                result ← 5

        if result = 1 then
            → TURN_RIGHT
        if result = 2 then
            → TURN_LEFT

        { Kalau kencang dan lane sebelah lebih hijau (ada powerup tanpa obstacle), pindah
}
    if myCar.speed > 8 then
        result ← doTurn(myCar, gameState)
        if result > 5 then
            if hasPowerUp(PowerUps.LIZARD, myCar.powerups) then
                → LIZARD
            else
                result ← 5

        if result = 1 then
            → TURN_RIGHT
```

```
        if result = 2 then
          → TURN_LEFT

        { Kalau lamban dan didepan ada lumpur, hindari }
      if myCar.speed <= 3 and blocks.contains(Terrain.MUD) then
        result ← doTurn(myCar, gameState)
        if result > 5 then
          if hasPowerUp(PowerUps.LIZARD, myCar.powerups) then
            → LIZARD
          else
            result ← 5

        if result = 1 then
          → TURN_RIGHT
        if result = 2 then
          → TURN_LEFT

        { Kalau terlalu lamban, dipercepat }
        if myCar.speed <= 3 then
              → ACCELERATE

        { Kalau didepan ada obstacle, hindarin }
      if blocks.contains(Terrain.MUD) or blocks.contains(Terrain.WALL) or
blocks.contains(Terrain.OIL_SPILL) then
        result ← doTurn(myCar, gameState)
        if result > 5 then
          if hasPowerUp(PowerUps.LIZARD, myCar.powerups) then
            → LIZARD
          else
            result ← 5

        if result = 1 then
          → TURN_RIGHT
        if result = 2 then
          → TURN_LEFT

        { Pakai EMP kalau kondisi oke }
        if hasPowerUp(PowerUps.EMP, myCar.powerups) and (opponent.position.lane =
myCar.position.lane or opponent.position.lane = myCar.position.lane - 1 or
opponent.position.lane = myCar.position.lane + 1) and opponent.position.block >=
myCar.position.block and myCar.speed >= 6 then
        → EMP

      if myCar.speed = 9 and not myCar.damage = 0 then
        → FIX

        { Pakai boost kalau punya dan ga lagi boost dan gak ada rintangan }
      if hasPowerUp(PowerUps.BOOST, myCar.powerups) and not myCar.boosting and not
(blocks.contains(Terrain.MUD) or blocks.contains(Terrain.WALL) or
blocks.contains(Terrain.OIL_SPILL)) then
```

```
        → BOOST

        { Kalau kencang dan punya tweet dan syarat kita didepan musuh, dan tidak dijalur sama, pakai tweet }
      if myCar.speed > 8 and hasPowerUp(PowerUps.TWEET, myCar.powerups) then
        if not (opponent.position.lane = myCar.position.lane and opponent.position.block < myCar.position.block) then
          → new TweetCommand(opponent.position.lane, opponent.position.block + opponent.speed)

        { Taruh oil }
      if myCar.speed = 15 and opponent.position.block < myCar.position.block and hasPowerUp(PowerUps.OIL, myCar.powerups) then
        → OIL

        { Jika tidak ada kondisi yang memenuhi, ACCELERATE saja }
        → ACCELERATE
```

4.1.2. Private Int checkMax

```
function checkMax(Car: myCar) → integer

KAMUS LOKAL

ALGORITMA
        depend on (myCar.damage)
                myCar.damage = 0 : → 15
                myCar.damage = 1 : → 9
                myCar.damage = 2 : → 8
                myCar.damage = 3 : → 6
                myCar.damage = 4 : → 3
                myCar.damage = 5 : → 0
        else
                → 9
```

4.1.3. Private Boolean hasPowerUp

```
function hasPowerUp(PowerUps: powerUpToCheck, PowerUps[]: available) → boolean


KAMUS LOKAL


ALGORIMA
        while PowerUps powerUp: available do
                if powerUp.equals(powerUpToCheck) then
                        → true
        → false
```

4.1.4. Private List<Object> getBlocks

```
function getBlocks(int: lane, block, GameState: gameState, Car: myCar, String: direction) → List<Object>
KAMUS LOKAL
        startBlock : integer
        map : List<Lane[]>
        blocks : List<Object>
        laneList : Lane[]

ALGORITMA
    map ← gameState.lanes
    blocks ← new ArrayList<>()
    startBlock ← map.get(0)[0].position.block

    Lane[] laneList

    if direction.equals("right") then
      if lane>3 then
        → blocks

      laneList ← map.get(lane)
    else
            if(direction.equals("left") then
            if lane-2<0 then
                → blocks

            laneList ← map.get(lane - 2)
      else
            laneList ← map.get(lane - 1)


    interval [max(block - startBlock, 0)...block - startBlock + myCar.speed + getAccelerate(myCar.speed, hasPowerUp(PowerUps.BOOST, myCar.powerups)] do
      if laneList[i] = null or laneList[i].terrain = Terrain.FINISH then
        break
      blocks.add(laneList[i].terrain)

    → blocks
```

4.1.5. Private Boolean checkTruck

```
function checkTruck(int: lane, block, GameState: gameState, String: direction, Car: myCar) → boolean

KAMUS LOKAL
        startBlock : integer
        map : List<Lane[]>
        laneList : Lane[]

ALGORITMA
    map ← gameState.lanes
    startBlock ← map.get(0)[0].position.block
```

```
    if direction.equals("left") then
        if lane-2<0 then
            → false
        laneList ← map.get(lane - 2)
    else if direction.equals("right") then
        if lane>3 then
            → false

        laneList ← map.get(lane)
    else
        laneList ← map.get(lane - 1)

    interval [max(block - startBlock -5, 0)...min(block - startBlock + myCar.speed + getAccelerate(myCar.speed, hasPowerUp(PowerUps.BOOST, myCar.powerups)), 1500)] then
        if laneList[i] = null or laneList[i].terrain == Terrain.FINISH then
            break

        if laneList[i].isOccupiedByCyberTruck then
            → true

    → false
```

4.1.6. Private Integer doTurn

```
function doTurn(Car: myCar, GameState: gameState) → integer

KAMUS LOKAL
        blocksFront, blocksLeft, blocksRight : List<Object>
        opponent : Car
        truckExists, truckLeft, truckRight : boolean
        valueLeft, valueRight, valueFront : boolean
        LizardLeft, LizardFront, LizardRight : integer
        TweetLeft, TweetFront, TweetRight : integer
        BOOSTLeft, BOOSTFront, BOOSTRight : integer
        WallLeft, WallFront, WallRight : integer
        MudLeft, MudFront, MudRight : integer
        OilLeft, OilFront, OilRight : integer
        wall, mud, oil : float
        valueLeft, valueRight, valueFront : float

ALGORITMA
    blocksFront ← getBlocks(myCar.position.lane, myCar.position.block, gameState, myCar, "front")
    blocksLeft ← getBlocks(myCar.position.lane, myCar.position.block, gameState, myCar, "left")
    blocksRight ← getBlocks(myCar.position.lane, myCar.position.block, gameState, myCar, "right")
    opponent ← gameState.opponent

        truckExists ← checkTruck(myCar.position.lane, myCar.position.block, gameState, "front", myCar)
    truckLeft ← checkTruck(myCar.position.lane, myCar.position.block, gameState, "left", myCar)
    truckRight ← checkTruck(myCar.position.lane, myCar.position.block, gameState, "right", myCar)
```

```
    LizardLeft ← Collections.frequency(blocksLeft, Terrain.LIZARD)
    LizardFront ← Collections.frequency(blocksFront, Terrain.LIZARD)
    LizardRight ← Collections.frequency(blocksRight, Terrain.LIZARD)

    TweetLeft ← Collections.frequency(blocksLeft, Terrain.TWEET)
    TweetFront ← Collections.frequency(blocksFront, Terrain.TWEET)
    TweetRight ← Collections.frequency(blocksRight, Terrain.TWEET)

    BOOSTLeft ← Collections.frequency(blocksLeft, Terrain.BOOST)
    BOOSTFront ← Collections.frequency(blocksFront, Terrain.BOOST)
    BOOSTRight ← Collections.frequency(blocksRight, Terrain.BOOST)

    WallLeft ← Collections.frequency(blocksLeft, Terrain.WALL)
    WallFront ← Collections.frequency(blocksFront, Terrain.WALL)
    WallRight ← Collections.frequency(blocksRight, Terrain.WALL)

    MudLeft ← Collections.frequency(blocksLeft, Terrain.MUD)
    MudFront ← Collections.frequency(blocksFront, Terrain.MUD)
    MudRight ← Collections.frequency(blocksRight, Terrain.MUD)

    OilLeft ← Collections.frequency(blocksLeft, Terrain.OIL_SPILL)
    OilFront ← Collections.frequency(blocksFront, Terrain.OIL_SPILL)
    OilRight ← Collections.frequency(blocksRight, Terrain.OIL_SPILL)

    valueLeft ← 0
    valueRight ← 0
    valueFront ← 0

    if truckExists then
      valueFront ← valueFront - 1500

    if truckLeft then
      valueLeft ← valueLeft - 1500

    if truckRight then
      valueRight ← valueRigh - 1500

    if (opponent.position.lane = myCar.position.lane) and opponent.position.block >=
myCar.position.block and (opponent.position.block + opponent.speed +
getAccelerate(opponent.speed, false)) <= (myCar.position.block + myCar.speed +
getAccelerate(myCar.speed, hasPowerUp(PowerUps.BOOST, myCar.powerups))) then
      valueFront ← valueFront - 1000

    wall ← 0
    mud ← 0
    oil ← 0
    if myCar.boosting then
        wall ← 5
        mud ← 5
        oil ← 5
    else
        wall ← 3
        mud ← 1.5
        oil ← 1.5
```

```
valueLeft ← valueLeft + (LizardLeft * 1) + (BOOSTLeft * 2) + (TweetLeft * 0.5)
valueRight ← valueRight + (LizardRight * 1)  + (BOOSTRight * 2) + (TweetRight * 0.5)
valueFront ← valueFront + (LizardFront * 1)  + (BOOSTFront * 2) + (TweetFront * 0.5)

valueLeft ← valueLeft - ((WallLeft * (wall)) + (MudLeft * (mud)) + (OilLeft * (oil)))
valueRight ← valueRight - ((WallRight * (wall)) + (MudRight * (mud)) + (OilRight * (oil)))
valueFront ← valueFront - ((WallFront * (wall)) + (MudFront * (mud)) + (OilFront * (oil)))

→ getNumber(valueFront, valueLeft, valueRight, WallLeft, WallRight, WallFront, myCar, BOOSTLeft, BOOSTRight, BOOSTFront)
```

4.1.7. Private Integer getNumber

```
function getNumber(double valueFront, double valueLeft, double valueRight, int WallLeft, int WallRight, int WallFront, Car myCar, int BOOSTLeft, int BOOSTRight, int BOOSTFront) → integer

KAMUS LOKAL

ALGORIMA
    if myCar.position.lane = 1 then
        if(valueRight > valueFront) then
            if(valueRight < 0) then
                → 6
            → 1
        else
            if(valueFront < 0) then
                return 8
            → 3

    if(myCar.position.lane = 4) then
        if(valueLeft > valueFront) then
            if(valueLeft < 0) then
                → 7
            → 2
        else
            if(valueFront < 0) then
                → 8
            → 3

    if(valueFront < 0 and valueLeft < 0 and valueRight < 0 ) then
        if(valueRight > valueLeft and valueRight > valueFront and not myCar.position.lane = 4) then
            → 6

        if(valueLeft > valueRight and valueLeft > valueFront and myCar.position.lane != 1) then
            → 7

        if(valueFront > valueLeft && valueFront > valueRight) then
            → 8
        if(valueLeft = valueFront) then
            → 8
        if(valueRight = valueFront) then
            → 8
        if(valueLeft = valueRight) then
```

```
        if(myCar.position.lane == 4) then
            → 7
        else if (myCar.position.lane == 1)
            → 8
        → 7

    if(valueRight > valueLeft and valueRight > valueFront and not myCar.position.lane = 4) then
        → 1

    if(valueLeft > valueRight and valueLeft > valueFront and myCar.position.lane != 1) then
        → 2

    if(valueFront > valueLeft and valueFront > valueRight) then
        → 3

    if(valueLeft = valueRight) then
        valueLeft ← valueLeft + WallLeft * (-1) + BOOSTLeft * 1
        valueRight ← valueRight WallRight * (-1) + BOOSTRight * 1

    if(valueLeft = valueFront) then
        valueLeft += WallLeft * (-1) + BOOSTLeft * 1
        valueFront += WallFront * (-1) + BOOSTFront * 1

    if(valueRight = valueFront) then
        valueRight += WallRight * (-1) + BOOSTRight * 1
        valueFront += WallFront * (-1) + BOOSTFront * 1

    if(valueRight > valueLeft and valueRight > valueFront and not myCar.position.lane = 4) then
        if(valueRight < 0) then
            → 6
        → 1

    if(valueLeft > valueRight and valueLeft > valueFront and not myCar.position.lane = 1) then
        if(valueLeft < 0) then
            → 7
        → 2

    if(valueFront > valueLeft and valueFront > valueRight) then
        if(valueFront < 0) then
            → 8
        → 3

    if(valueLeft = valueRight) then
         if(myCar.position.lane = 4) then
                    → 2
                    if (myCar.position.lane = 1) then
                    → 1
        → 2
    → 3
```

4.1.2. Private Integer getAccelerate

```
function getAccelerate(int speedNow, boolean x) → integer

KAMUS LOKAL
```

```
ALGORIMA
    if(speedNow = 0) then
      → 3
    if (speedNow = 3 or speedNow = 6) then
      → 2
    if (speedNow = 5 or speedNow = 8) then
      → 1
    else
      if(speedNow = 9) then
        if(x) then
          → 7
        else
          → 0
      else
        → 0
```

4.2 Penjelasan Struktur Data yang Digunakan

Terdapat berbagai jenis struktur data yang membangun program ini, khususnya pada file Bot.java. Penulis akan menjabarkan struktur data yang digunakan dengan batasan hanya struktur data yang terdapat pada file Bot.java.

4.2.1 run

Fungsi ini merupakan salah satu fungsi utama dimana penulis diharuskan untuk mengubah-ubah dan mengedit pada bagian ini untuk memenangkan pertandingan. Kelas ini berisi perbandingan if else dengan urutan prioritas *command* yang akan dikembalikan. Struktur data yang digunakan dalam fungsi ini adalah kelas Car, array List<Object>, dan boolean.

4.2.2 hasPowerUp

Fungsi ini merupakan fungsi untuk mengembalikan sebuah boolean yang menunjukkan apakah bot mobil memiliki power up tertentu atau tidak. Fungsi ini tidak memiliki struktur data apapun.

4.2.3 getBlocks

Fungsi ini mengembalikan sebuah array yang berisi blocks-blocks tertentu pada suatu baris tertentu. Fungsi ini memiliki struktur data array List<Lane[]>, array List<Object>, dan integer.

4.2.4 checkTruck

Fungsi ini akan mengembalikan sebuah boolean yang menunjukkan apakah ada cyber truk pada baris tersebut. Fungsi ini memiliki struktur data array List<Lane[]> serta integer.

4.2.5 doTurn

Fungsi ini akan mengembalikan sebuah integer untuk menunjukkan ke arah manakah bot mobil harus bergerak. Fungsi ini menghitung nilai pembobotan dari setiap jalur dan akan memberikan bobot untuk setiap jalur ke fungsi getNumber. Fungsi doTurn memiliki struktur data array List<Object>, kelas Car, boolean, integer, dan double.

4.2.6 getNumber

Fungsi ini akan mengembalikan sebuah integer untuk menunjukkan ke arah manakah bot mobil harus bergerak. Integer tersebut akan diberikan ke function doTurn. Fungsi ini berisi urutan if else yang berisi kondisi-kondisi tertentu yang dipenuhi oleh setiap bobot pada jalur tertentu. Tidak ada struktur data yang digunakan oleh fungsi ini.

4.2.7 getAccelerate

Fungsi ini akan mengembalikan sebuah integer yang menunjukkan kecepatan maksimal yang didapat dari command Accelerate berdasarkan kecepatan saat ini. Tidak ada struktur data yang digunakan oleh fungsi ini.

4.3 Analisis Desain Solusi Algoritma *Greedy*

Untuk melakukan analisis desain solusi yang kami buat, kami menandingkan bot kami terhadap bot reference yang diberikan oleh starter-pack. Pada pengujian yang dilakukan, kami menandingkan bot kami dengan bot referensi sebanyak lima kali. Dari kelima pengujian tersebut, bot kami berhasil memenangkan pertandingan sebanyak lima kali.

Pada seluruh pertandingan, bot kami telah berhasil melakukan pemindahan lane dan penggunaan powerup dengan efektif dan efisien. Meskipun begitu, bot ini masih memiliki beberapa kekurangan.

Pertama, bot kami terlalu agresif dalam melakukan command fix. Bot kami melakukan fix ketika damage bernilai satu dan tidak memiliki boost. Hal tersebut tidak efisien waktu karena keuntungan memiliki damage bernilai nol hanyalah untuk memaksimalkan kecepatan boost.

Kedua, bot kami belum memanfaatkan command tweet dengan benar. Command tweet sangat merugikan musuh apabila dilakukan dengan benar. Akan tetapi command ini sangat mudah dihindari oleh musuh. Penggunaan command tweet pada bot kami belum menutup semua kemungkinan penhindaran musuh.

BAB 5

Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan implementasi algoritma Greedy permainan “Overdrive” yang telah dibuat, kami menyimpulkan bahwa algoritma Greedy dapat digunakan untuk memainkan permainan ini. Algoritma solusi yang kami buat menggunakan algoritma Greedy, yaitu mengambil solusi paling optimum berdasarkan bobot nilai dari setiap jalur. Bobot yang digunakan mengunjukkan bahwa semakin positif bobot tersebut, bot mobil akan cenderung berbelok ke arah tersebut. Walaupun bisa jadi setelah keputusan pada optimum lokal tersebut justru membawa konsekuensi yang kurang baik pada keputusan berikutnya, misal saat ini berbelok ke arah tertentu untuk mengambil Power Ups, ternyata justru terjebak di dalam sekumpulan hambatan seperti lumpur.

5.2 Saran

Pengaplikasian algoritma Greedy pada program kami tentu masih jauh dari kesempurnaan sehingga saran kami pada tugas besar kali ini adalah dapat mengembangkan algoritma Greedy dengan lebih baik lagi sehingga kondisi optimum lokal dapat mendekati kondisi optimum globalnya.

Link Github : https://github.com/dParikesit/TubesStima1

Link Youtube : https://youtu.be/HRQKqzPfrDA

Daftar Pustaka

Munir, R. (2020). Algoritma Greedy Bagian 1[PDF]. Institut Teknologi Bandung. Diakses pada 6 Februari 2020 pukul 11.21 WIB melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf.

Munir, R. (2020). Algoritma Greedy Bagian 2[PDF]. Institut Teknologi Bandung. Diakses pada 6 Februari 2020 pukul 11.33 WIB melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Gree dy-(2021)-Bag2.pdf.

Munir, R. (2020). Algoritma Greedy Bagian 3[PDF]. Institut Teknologi Bandung. Diakses pada 6 Februari 2020 pukul 11.47 WIB melalui https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf.