

# **Laporan Tugas Besar I**

## **IF2211 Strategi Algoritma**

### **Semester II Tahun 2021/2022**



Anggota:

Marchotridyo	13520119
Mahesa Lizardy	13520116
Muhammad Rakha Athaya	13520108

**Institut Teknologi Bandung**  
**2022**

# Daftar Isi

<b>Daftar Isi</b>	<b>1</b>
<b>Bab I</b>	
<b>Deskripsi Tugas</b>	<b>2</b>
<b>Bab II</b>	
<b>Landasan Teori</b>	<b>5</b>
1. Algoritma Greedy	5
2. Cara Kerja Program	5
<b>Bab III</b>	
<b>Aplikasi Strategi Greedy</b>	<b>6</b>
1. Pemetaan Persoalan Overdrive ke Elemen Algoritma Greedy	6
2. Alternatif Solusi Greedy	6
3. Analisis Efisiensi dan Efektivitas Alternatif Solusi Greedy	6
4. Strategi Greedy yang Dipilih	6
<b>Bab IV</b>	
<b>Implementasi dan Pengujian</b>	<b>7</b>
1. Implementasi Algoritma pada Program Bot	7
2. Struktur Data pada Program Bot	7
3. Analisis Hasil Pengujian	7
<b>Bab V</b>	
<b>Kesimpulan dan Saran</b>	<b>8</b>
1. Kesimpulan	8
2. Saran	8
<b>Daftar Pustaka</b>	<b>9</b>

# Bab I

## Deskripsi Tugas

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



Gambar 1. Ilustrasi permainan *Overdrive*

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 power ups. 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>

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 bernilaiimbang. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menggunakan power ups 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* adalah salah satu cara menyelesaikan persoalan optimasi, yaitu maksimasi atau minimasi. Ciri dari algoritma *greedy* adalah solusi dibuat langkah per langkah dengan setiap akan melakukan sebuah langkah akan terdapat banyak pilihan yang perlu dievaluasi. Algoritma *greedy* bersifat tidak bisa kembali ke langkah sebelumnya sehingga diperlukan pilihan yang paling optimal di setiap langkahnya. Setiap pilihan disebut sebagai optimum lokal dengan suatu harapan bahwa langkah-langkah optimum lokal ini akan membawakan hasil yang optimum secara global.

Dalam menyusun algoritma *greedy*, dibuat beberapa elemen-elemen untuk mendefinisikan algoritma tersebut, yaitu:

1. Himpunan kandidat, *C*: berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, *S*: berisi kandidat yang sudah dipilih.
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi (*selection function*): memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
5. Fungsi kelayakan (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi obyektif: memaksimumkan atau meminimumkan.

Menggunakan elemen-elemen di atas, algoritma *greedy* dapat didefinisikan sebagai algoritma yang melibatkan pencarian sebuah himpunan bagian, *S*, dari himpunan kandidat, *C*, di mana *S* harus memenuhi kriteria yang ditentukan, yaitu *S* adalah suatu solusi yang dioptimasi oleh fungsi obyektif.

#### 2.2 Cara Kerja Program

##### 2.2.1 Cara Bot Melakukan Aksinya

Bot melakukan aksinya melalui sekumpulan *command* yang sudah didefinisikan pada folder */command* yang berisikan *command-command* berikut serta rinciannya:

1. Command ACCELERATE berfungsi untuk menambahkan kecepatan mobil sampai kecepatan maksimumnya.
2. Command DECELERATE berfungsi untuk mengurangi kecepatan mobil.
3. Command TURN\_LEFT berfungsi untuk pindah ke lajur sebelah kiri (secara matematis, lajur dikurangi satu) jika memungkinkan.
4. Command TURN\_RIGHT berfungsi untuk pindah ke lajur sebelah kanan (secara matematis, lajur ditambah satu) jika memungkinkan.
5. Command USE\_BOOST akan membuat mobil bergerak dengan kecepatan 15 selama 5 ronde apabila memiliki *power up* BOOST. Jika terkena suatu *obstacle*, maka kecepatan menjadi kecepatan maksimum mobil.
6. Command USE\_OIL meletakkan suatu tumpahan minyak tepat di blok mobil sekarang.
7. Command USE\_TWEET X Y meletakkan sebuah *cyber truck* di lajur X blok Y untuk menutup lajur musuh.
8. Command USE\_LIZARD membuat mobil terbang untuk satu ronde sehingga tidak menabrak *obstacle* maupun mendapat *power up* dari blok-blok di depannya.
9. Command USE\_EMP menembakkan EMP ke mobil musuh, menyebabkan mobilnya terhenti selama 1 ronde dan kecepatannya dikurangi menjadi 3.
10. Command FIX mengurangi dua poin *damage* dari mobil.
11. Command NOTHING membiarkan mobil untuk melaju dengan kecepatan sekarang, tanpa mengurangi atau menambahkan kecepatan.

Perihal ACCELERATE dan DECELERATE mengikuti skala kecepatan berikut:

1. Kecepatan minimum mobil adalah 0 (berhenti).
2. Speed mobil bergerak paling lambat adalah 3 (SPEED\_STATE\_1).
3. Di awal permainan, mobil bergerak dengan kecepatan 5.
4. SPEED\_STATE\_2 adalah 6.
5. SPEED\_STATE\_3 adalah 3.
6. MAXIMUM\_SPEED adalah 9.
7. BOOST\_SPEED adalah 15.
- 8.

ACCELERATE akan menyebabkan menambah kecepatan ke SPEED\_STATE selanjutnya (jika ada) dan DECELERATE akan mengurangi kecepatan ke SPEED\_STATE sebelumnya (jika ada). Skala kecepatan tersebut juga akan dipakai saat mobil mengenai *obstacle*.

Selama mobil bergerak, dimungkinkan untuk mobil menabrak beberapa *obstacle* yang memiliki sifat-sifat yang berbeda, yaitu:

1. MUD dan OIL\_SPILL mengurangi SPEED\_STATE sekarang sebanyak satu, apabila sudah kecepataannya 3, tetap 3. *Obstacle* ini juga menambahkan *damage* sebanyak 1.
2. WALL mengurangi SPEED\_STATE menjadi SPEED\_STATE\_1, tidak peduli berapa kecepatan sebelumnya. *Obstacle* ini juga menambahkan *damage* sebanyak 2.
3. CYBERTRUCK membuat mobil berhenti untuk satu ronde dan membuat kecepataannya menjadi SPEED\_STATE\_1 pada ronde sebelumnya, tidak peduli berapa kecepatan sebelumnya. *Obstacle* ini juga menambahkan *damage* sebanyak 2.

Bot bergerak dengan tujuan mencapai *finish line* pada posisi pertama. Apabila bot pemain dan bot lawan mencapai *finish line* pada saat bersamaan, maka skor dari mobil diperhitungkan. Skor dari bot diatur oleh beberapa aturan berikut:

1. Menabrak MUD mengurangi skor sebanyak 3.
2. Menabrak OIL\_SPILL mengurangi skor sebanyak 4.
3. Mengambil *powerup* menambah skor sebanyak 4.
4. Menggunakan *powerup* menambah skor sebanyak 4.
5. *Command* yang tidak valid mengurangi skor sebanyak 5.

### 2.2.2 Cara Implementasi Algoritma Greedy ke dalam Bot

Permainan memperbolehkan suatu pemain melihat kondisi di sekitarnya, lebih spesifiknya 5 blok di belakang mobil dan 20 blok di depan mobil untuk setiap lajur yang ada. Misalnya, pemain dapat mengetahui bahwa di lajur depan terdapat 4 buah *obstacle*, sedangkan di lajur kanan tidak terdapat *obstacle*. Adanya informasi ini dapat digunakan oleh pemrogram untuk mengatur arah gerak bot selanjutnya.

Contoh berikutnya adalah permainan menyediakan beberapa atribut yang bisa diakses oleh pemrogram, misal kecepatan mobil sekarang, *powerup* apa saja yang dimiliki mobil sekarang, *state* apa yang dimiliki mobil sekarang (apakah sedang BOOST, apakah terkena TWEET, apakah ronde terakhir menggunakan TWEET, dan lain lain). Data-data ini digunakan sebagai pertimbangan untuk pemrogram mendefinisikan algoritma *greedy*-nya, misalnya jangan melakukan BOOST lagi apabila ronde sebelumnya sudah melakukan BOOST, tetapi tunggu BOOST-nya selesai dahulu baru menggunakan *powerup* lagi.

### 2.2.3 Cara menjalankan *game engine*

Setelah mengunduh *starter pack* yang ada di laman <https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>, jalankan



file *run.bat* (untuk Windows) dan *game engine* akan berjalan. Untuk mengonfigurasi bot apa yang ingin digunakan dalam pertandingan, *file* yang perlu diubah adalah *file* *game-runner-config.json*, khususnya mengganti nilai dari atribut *player-a* dan *player-b*.

## Bab III

### Aplikasi Strategi *Greedy*

#### 3.1 Pemetaan Persoalan Overdrive ke Elemen Algoritma *Greedy*

Elemen-elemen algoritma *greedy* untuk persoalan Overdrive didefinisikan oleh beberapa poin berikut:

1. Himpunan kandidat, *C*, yaitu kandidat yang akan dipilih pada setiap langkah. Pada kasus Overdrive ini, himpunan ini adalah *command* apa saja yang bisa dipilih pada setiap *round*, yaitu  $C = \{\text{DO\_NOTHING, ACCELERATE, DECELERATE, TURN\_LEFT, TURN\_RIGHT, USE\_BOOST, USE\_OIL, USE\_TWEET, USE\_LIZARD}\}$
2. Himpunan solusi, *S*, yaitu kandidat yang sudah dipilih. Pada kasus Overdrive, himpunan ini hanya berisikan satu *command* saja yang dipilih dari himpunan kandidat *C*.
3. Fungsi solusi, yaitu fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi. Pada kasus Overdrive, himpunan ini berisikan fungsi yang mengecek apakah solusi yang diberikan dapat dijalankan oleh bot.
4. Fungsi seleksi, yaitu fungsi yang memilih kandidat berdasarkan strategi *greedy* tertentu. Pada kasus Overdrive ini, fungsi seleksi adalah kumpulan strategi heuristik untuk menentukan *command* mana yang layak dipakai, yaitu *command-command* yang memaksimalkan peluang bot untuk memenangkan pertandingan.
5. Fungsi kelayakan, yaitu fungsi yang memeriksa apakah kandidat yang dipilih layak dimasukkan sebagai solusi. Pada kasus Overdrive, fungsi kelayakan ini mirip dengan fungsi solusi yaitu mengecek apakah *command* yang akan dilakukan valid atau tidak.
6. Fungsi obyektif, yaitu fungsi yang memaksimumkan atau meminimumkan. Pada kasus Overdrive ini, fungsi obyektif adalah mencapai garis finis secepat mungkin dengan prioritas kedua mendapatkan skor sebanyak mungkin.

#### 3.2 Alternatif Solusi *Greedy*

##### 3.2.1 Alternatif 1

Alternatif pertama adalah *greedy by distance travelled*. Menggunakan alternatif ini, bot akan mengabaikan banyaknya *powerup* maupun *block* yang ada di sekitarnya. Bot hanya akan memanggil *ACCELERATE* dan *BOOST* serta memanggil *FIX* apabila dibutuhkan. Mobil hanya akan bergerak pada satu jalur saja mengingat *ACCELERATE*

dan *BOOST* adalah dua *command* yang akan memberikan jarak terjauh setiap rondonya.

### 3.2.2 Alternatif 2

Alternatif kedua adalah *greedy by offense and defense*. Menggunakan alternatif ini, bot akan fokus untuk menggunakan power up yang bersifat offensif seperti *OIL POWER*, *TWEET*, dan *EMP*. Bot akan melakukan command tersebut terlebih dahulu jika memungkinkan. Bot juga akan menyimpan power up *LIZARD* yang didapatkan dan hanya menggunakan jika memang rintangan yang didepan tidak bisa dilewati dengan belok ke kiri atau kanan (blok kiri dan kanan juga terdapat rintangan) sehingga memaksimalkan pemakaian *LIZARD*. Dengan menyimpan power up tersebut dapat meminimalkan pemakaian command *FIX*.

### 3.2.3 Alternatif 3

Alternatif ketiga adalah *greedy by power up*. Menggunakan alternatif ini, bot akan lebih menyukai bergerak ke lajur yang memiliki banyak power up tanpa memedulikan jarak yang ia tempuh ataupun *obstacle* yang akan ada pada jalurnya. Setiap bot mendapat power up yang layak untuk bot pakai, bot akan langsung menggunakan *power up* dengan tujuan menambah skor dan mengganggu pergerakan bot musuh. Bot akan melakukan *FIX* jika *damage* yang dimiliki mobil terlalu besar.

### 3.2.4 Alternatif 4

Alternatif keempat adalah *greedy by minimizing damage* (tapi tetap bergerak setiap ronde, tidak *DECELERATE* sampai berhenti dan tidak bergerak). Menggunakan alternatif ini, bot akan lebih menyukai melakukan *command* yang membuatnya tetap bergerak dengan meminimalisasi *damage* yang dikenainya, misal apabila di depan ada tembok tapi di lajur kiri kosong, maka bot akan melakukan *TURN\_LEFT*. Bot tidak mempertimbangkan jarak tempuh ataupun banyak *power up* di lajur lainnya. Bot akan melakukan *FIX* jika *damage* yang dimiliki mobil terlalu besar.

### 3.2.5 Alternatif 5

Alternatif kelima adalah *greedy by choosing the best lane by score without attacking*. Alternatif ini menganalisis ketiga aspek yang disebutkan pada alternatif 1, 3, dan 4 yaitu banyaknya jarak yang akan ditempuh, banyaknya *power up* yang terambil, dan sedikitnya *damage* yang dikenai oleh mobil. Setiap *command* akan dianalisis lintasannya dan lintasan dengan skor terbaik akan dipilih oleh mobil. Bot lebih memprioritaskan gerakan daripada menyerang, sehingga *power up* menyerang seperti *OIL\_SPILL*, *TWEET*, dan *EMP* tidak digunakan, hanya *BOOST* yang berguna dalam strategi ini. Bot akan melakukan *FIX* jika *damage* yang dimiliki mobil terlalu besar.

### 3.2.6 Alternatif 6

Alternatif keenam adalah gabungan alternatif kelima dengan prioritas *offense*. Pada alternatif ini, jika bot tidak memiliki *power up* menyerang, bot akan memfokuskan gerakannya sesuai dengan alternatif 5. Tetapi, ketika bot memiliki *power up* menyerang, maka *bot* akan menggunakannya langsung tanpa memperhatikan kondisi mobilnya sekarang, seperti kecepatan mobil sekarang atau apakah mobil akan menabrak *obstacle* setelah menggunakan *power up*. Bot akan melakukan *FIX* jika *damage* yang dimiliki mobil terlalu besar.

### 3.2.7 Alternatif 7

Alternatif ketujuh adalah gabungan alternatif kelima dengan prioritas *movement*, tapi tetap melakukan *offense*. Pada alternatif ini, bot hanya akan menggunakan *power up* menyerang ketika *command* gerakan yang dipilih oleh bot adalah *DO\_NOTHING*, karena artinya penggunaan *power up* tetap akan menggerakkan bot ke lajur terbaiknya dengan tambahan menggunakan *power up*. Bot akan melakukan *FIX* jika *damage* yang dimiliki mobil terlalu besar.

### 3.2.8 Alternatif 8

Alternatif kedelapan adalah alternatif ketujuh yang diberikan beberapa heuristik terhadap beberapa kondisi khusus. Kondisi tersebut meliputi (dari prioritas tertinggi):

- a. Jika bot bisa bergerak ke *FINISH\_LINE* dalam satu gerakan, jangan melakukan apapun selain gerakan tersebut.
- b. Jika mobil berhenti (memiliki *speed* 0) utamakan penggunaan *ACCELERATE* atau *USE\_BOOST* agar mobil dapat bergerak kembali secepat mungkin.
- c. Bot melakukan *FIX* jika *damage*  $\geq 2$  atau *damage*  $\geq 1$  jika mobil memiliki boost dengan tujuan agar bot dapat bergerak dengan kecepatan maksimum sebanyak mungkin.
- d. Jika musuh berada pada posisi jauh dan sedang bergerak dengan kecepatan cepat (9 atau 15), gunakan *EMP* untuk memperlambat gerakannya, tidak peduli skor lajur apabila bot melakukan *command* *DO\_NOTHING*.

Jika keempat kondisi tersebut tidak dipenuhi, maka bot akan melakukan logika *movement* dan *offense* seperti yang dijelaskan pada alternatif ketujuh.

### 3.3 Analisis Efisiensi dan Efektivitas Alternatif Solusi Greedy

#### 3.3.1 Alternatif 1

##### 3.3.1.1 Efisiensi

Kalkulasi yang dilakukan hanya memeriksa apakah bot memiliki *power up boost* (kompleksitasnya sebanding dengan banyak *power up* yang dimiliki bot sekarang) dan mengecek apakah mobil membutuhkan command *FIX*, yaitu pengecekan *damage* yang kompleksitasnya konstan. Secara efisiensi, algoritma ini sangat cepat, kompleksitasnya hanyalah  $O(P)$  dengan  $P$  adalah banyak *power up* yang dimiliki oleh mobil sekarang.

##### 3.3.1.2 Efektivitas

Walaupun sangat efisien, algoritma ini sangat tidak dianjurkan untuk digunakan karena walaupun secara teoritis algoritma ini akan mengantarkan bot ke garis finis dalam waktu tercepat, sistem *obstacle* pada permainan ini memberikan hukuman yang cukup berat bagi pemain yang sering menabrak *obstacle* dan mengabaikan sistem *power up* yang ada pada permainan akan membuat bot tertinggal jauh.

#### 3.3.2 Alternatif 2

##### 3.3.2.1 Efisiensi

Blok melakukan kalkulasi terhadap rintangan yang ada didepan bot serta samping kanan dan kiri untuk menentukan apakah bot harus menggunakan *LIZARD* atau menyimpannya sehingga komputasi yang dilakukan lebih kompleks.

##### 3.3.2.1 Efektivitas

Walaupun sangat efisien untuk menghindari rintangan yang ada. Namun efisien tersebut berbanding terbalik dengan kecepatan. Karena bot akan sering melakukan perpindahan lane sehingga kecepatan bot akan lebih kecil dibandingkan dengan yang tidak berbelok oleh karena itu diperlukan nya *power up offense* seperti *EMP* dan *TWEET* untuk mengurangi kecepatan lawan yang berada di depan

#### 3.3.3 Alternatif 3

##### 3.3.3.1 Efisiensi

Bot mulai melakukan kalkulasi terhadap blok-blok yang ada di depannya sehingga komputasi lebih kompleks. Bot akan memeriksa blok-blok yang akan dilalui oleh mobil apabila memanggil *command* gerakan  $C = \{DO\_NOTHING, ACCELERATE, DECELERATE, TURN\_LEFT, TURN\_RIGHT, USE\_BOOST, USE\_LIZARD\}$ .

Masing-masing *command* akan dinilai dan dianalisis blok-blok yang dilewatinya, misalnya apabila ACCELERATE membuat mobil bergerak 9 blok, akan dilakukan iterasi pada tiap blok tersebut untuk dicek apakah blok mengandung *power up* atau tidak. Setelah itu, dilakukan komparasi untuk mencari mana *command* yang terbaik, proses ini waktunya konstan. Sehingga, kompleksitas alternatif ini adalah  $O(CB)$  dengan  $C$  adalah banyak *command* yang perlu dicek dan  $B$  adalah banyak *block* yang harus dicek pada *command*  $C$ .

### 3.3.3.2 Efektivitas

Penambahan *powerup* tidak terlalu membantu apabila mobil sering menabrak *obstacle*. Walaupun mobil bisa menyerang dan berpindah lajur untuk mendapat *power up*, *damage* yang terakumulasi akan sangat memperlambat gerak mobil untuk menuju garis finis.

## 3.3.4 Alternatif 4

### 3.3.4.1 Efisiensi

Kalkulasi yang dilakukan sama persis dengan alternatif 3, hanya yang dihitung bukanlah banyak *power up* tapi banyak *damage* yang dimiliki setiap lajur.

### 3.3.4.2 Efektivitas

Alternatif ini mulai cukup baik karena lebih berperilaku seperti manusia yang berhati-hati mengendarai mobilnya. Namun, karena bot tidak mempertimbangkan penggunaan *power up*, hanya meminimalisasi *damage* yang didapat, pada akhirnya bot akan tertinggal oleh bot lawan.

## 3.3.5 Alternatif 5

### 3.3.5.1 Efisiensi

Kalkulasi yang dilakukan sama persis dengan alternatif 3, hanya saja yang dihitung ada beberapa faktor (banyak iterasinya sama, tetapi setiap iterasi akan memiliki lebih banyak operasi karena yang di *keep track* ada 3 variabel, operasinya tapi waktunya konstan dan tidak memengaruhi kompleksitas).

### 3.3.5.2 Efektivitas

Bot ini mulai menunjukkan kebolehannya dalam permainan karena pilihan yang dia ambil setiap perjalanan cukup teroptimasi, tidak lagi sering menabrak dan mengabaikan *power up*. Hanya saja, bot ini terlalu mementingkan gerakan sehingga

tidak sempat mempertimbangkan penggunaan *power up* yang dapat merugikan bot jika memang ada kesempatan yang seharusnya bot menyerang.

### 3.3.6 Alternatif 6

#### 3.3.6.1 Efisiensi

Kalkulasi yang dilakukan sama persis dengan alternatif 5, dengan menambah suatu pengecekan terhadap *powerup* yang dimiliki yaitu kompleksitas  $O(P)$  ketika bot akan melakukan penyerangan. Kompleksitas menjadi  $O(BC + P)$ . Definisi C, B, dan P sama seperti yang sudah dijelaskan pada alternatif 1 dan alternatif 3.

#### 3.3.6.2 Efektivitas

Bot ini mulai bisa menyerang dan bergerak dengan baik, hanya saja bot dapat bersifat terlalu *reckless* saat melakukan penyerangan, misal tetap menyerang walaupun efek menyerang akan membuatnya menabrak *cybertruck* atau *wall*.

### 3.3.7 Alternatif 7

#### 3.3.7.1 Efisiensi

Kalkulasi yang dilakukan sama dengan alternatif 6, dengan perbedaan hanyalah algoritma pengecekan *powerup* hanya dipanggil ketika *command* yang dipilih oleh bot adalah DO\_NOTHING.

#### 3.3.7.2 Efektivitas

Bot sudah cukup *mature*, dalam artian dapat bergerak dan menyerang dengan tetap memperhatikan sekitarnya. Namun, bot terkesan terlalu sabar dalam artian mengabaikan beberapa kasus di mana bot lebih baik menyerang langsung (misal menggunakan EMP jika musuh sudah cukup jauh) dibandingkan menunggu saat yang aman untuk menyerang.

### 3.3.8 Alternatif 8

#### 3.3.8.1 Efisiensi

Kalkulasi yang dilakukan sama dengan alternatif 6, dengan ditambah beberapa pengecekan yang konstan seperti heuristik poin a., b., c., dan d. Walaupun banyak pengecekan bertambah, kompleksitas algoritmanya secara umum sama saja.

### 3.3.8.2 Efektivitas

Bot ini merupakan bentuk perbaikan terhadap bot yang ada di alternatif 7. Menggunakan heuristik, bot dapat *meninggalkan* algoritma gerakan *default*-nya untuk melakukan beberapa hal yang dirancang untuk memaksimalkan apa yang diberikan oleh permainan untuk bot dengan tujuan agar bot memiliki kesempatan untuk memenangkan pertandingan.

## 3.4 Strategi Greedy yang Dipilih

Pada tugas besar ini kami memilih strategi bot alternatif 8 dengan alasan alternatif 8 adalah optimasi dari alternatif-alternatif sebelumnya yang juga merupakan optimasi dari alternatif-alternatif yang ada. Harapannya adalah dengan bot bermain aman, tidak terlalu *reckless* tapi tetap bisa melakukan penyerangan ketika memang dirasa lebih penting untuk menyerang daripada memprioritaskan gerakan.



## Bab IV

# Implementasi dan Pengujian

### 4.1 Implementasi Algoritma pada Program Bot

Pada subbab ini, algoritma-algoritma fungsi pembantu akan dijelaskan terlebih dahulu sebelum algoritma utama bot.

#### 4.1.1 Fungsi getAcceleratingSpeed

Fungsi ini mengembalikan kecepatan akhir mobil apabila melakukan *command* ACCELERATE.

```
function getAcceleratingSpeed(speed, boostCounter: integer) ->
integer
Algoritma
depend on(speed)
    speed = 3 or speed = 5: -> 6
    speed = 6: -> 8
    speed = 8 or speed = 9: -> 9
    speed = 15: if (boostCounter > 1) then
        -> 15
    else
        -> 9
```

#### 4.1.2 Fungsi getDeceleratingSpeed

Sama seperti getAcceleratingSpeed, tapi untuk *command* DECELERATE.

```
function getDeceleratingSpeed(speed: integer) -> integer
Algoritma
depend on(speed)
    speed = 3 or speed = 5: -> 0
    speed = 6: -> 3
    speed = 8: -> 6
    speed = 9: -> 8
    speed = 15: -> 9
```

#### 4.1.3 Fungsi getLaneScore

Fungsi ini mengembalikan sebuah tupel (X, Y, Z) dengan X = total *damage* dari lajur yang akan dipilih, Y = skor total *power up* yang diambil, Z = jumlah dari kecepatan akhir mobil dan jarak yang ditempuh setelah melakukan *command* yang didefinisikan oleh parameter *type*.

Contoh dari tupel yang dihasilkan oleh pengecekan ini ditunjukkan oleh ilustrasi berikut:



Gambar 2. Ilustrasi hasil dari getLaneScore ke berbagai *type* yang dipanggil.

```
function getLaneScore(gameState: GameState, type: string) ->
laneScore
Deklarasi
    car, opponent: Car
    lane: Lane
    laneDamage, pickUpScore, checkDistance, finalSpeed, laneID, i:
integer
    isLizard: boolean
Algoritma
    { Inisialisasi skor lane dahulu }
    laneDamage <- 0; pickUpScore <- 0
    checkDistance <- 0; finalSpeed <- 0
    isLizard <- false;
    { Inisialisasi mobil }
    car <- gameState.player; opponent <- gameState.opponent;
    laneID <- car.position.lane;
    { Set up sesuai tipe }
    if (tipe = "DO_NOTHING") then
        checkDistance <- car.speed
        if boost_akan_habis then
            checkDistance <- 9
    if (tipe = "ACCELERATE") then
        checkDistance <- getAcceleratingSpeed(car.speed,
```

```

car.boostCounter)
  if (tipe == "TURN_LEFT") then
    checkDistance <- car.speed - 1;
    laneID <- laneID - 1 { bergerak ke kiri }
  if (tipe == "TURN_RIGHT") then
    checkDistance <- car.speed - 1;
    laneID <- laneID + 1 { bergerak ke kanan }
  if (tipe == "BOOST") then
    checkDistance <- 15
  if (tipe == "DECELERATE") then
    checkDistance <- getDeceleratingSpeed(car.speed)
  { Iterasi lane yang dimaksud }
  lane <- ambil_info_dari_gameState_sesuai_idLane
  i <- 1 { i adalah iterator untuk blok ke-i yang dicek }
  repeat
    if (lane[i] = "MUD" or "OIL") then
      finalSpeed <- cari_state_sebelumnya
      laneDamage <- laneDamage + 1
    if (lane[i] = "WALL") then
      finalSpeed <- 3
      laneDamage <- laneDamage + 2
    if (lane[i] = "CYBERTRUCK") then
      finalSpeed <- 0
      laneDamage <- laneDamage + 12
    if (lane[i] = "OPPONENT") then
      laneDamage <- laneDamage + 1
    if (lane[i] = "OIL_POWER") then
      pickUpScore <- pickUpScore + 1
    if (lane[i] = "TWEET" or "EMP" or "LIZARD") then
      pickUpScore <- pickUpScore + 2
    if (lane[i] = "BOOST") then
      pickUpScore <- pickUpScore + 5
    i <- i + 1
  until i > checkDistance
  { Kembalikan tupel laneScore }
  -> [laneDamage, pickUpScore, checkDistance + finalSpeed]

```

#### 4.1.4 Fungsi getBestLane

Fungsi getBestLane mengambil dua buah tupel skor sebuah lajur untuk dibandingkan mana yang terbaik. Prioritas dimulai dari *laneDamage* > *pickUpScore* > *checkDistance* + *finalSpeed*.

```

function getBestLane(lane1, lane2: laneScore) -> laneScore
Algoritma
  { Bandingkan damage di lane }
  if (lane1[0] < lane2[0])
    -> lane1
  else if (lane2[0] < lane1[0])

```

```

        -> lane2
    else
        { Bandingkan skor pick up }
        if (lane1[1] > lane2[1])
            -> lane1
        else if (lane2[1] > lane1[1])
            -> lane2
        else
            { Bandingkan checkDistance + finalSpeed }
            if (lane1[2] >= lane2[2])
                -> lane1
            else
                -> lane2

```

#### 4.1.5 Fungsi shouldEMPEarly

Fungsi shouldEMPEarly menganalisis *state game* untuk menentukan apakah mobil lebih baik melakukan EMP atau melakukan logika gerakan biasa.

```

function shouldEMPEarly(gameState: GameState) -> boolean
Deklarasi
    car, opponent: Car
Algoritma
    ambil data mobil pemain (car) dan musuh (opponent) dari
    gameState
    if (car_memiliki_EMP) then
        if (getLaneScore(gameState, "DO_NOTHING") = 0) then
            -> (opponent.speed >= 8 and beda_satu_lajur and
            opponent.block > car.block + car.speed)
        else if (getLaneScore(gameState, "DO_NOTHING") = 1) then
            -> (opponent.speed == 15 and beda_satu_lajur and
            opponent.block > car.block + car.speed)
        { Yang diperhatikan: tidak melakukan EMP jika menggunakan EMP
        akan membuat kita menabrak mobil musuh }

```

#### 4.1.6 Fungsi shouldUseOil

Fungsi shouldUseOil menganalisis *gameState* untuk melihat apakah peletakan OIL\_SPILL akan menghalangi jalan musuh, diilustrasikan oleh gambar di bawah.

```

YYY
 x
ZZZ

```

dengan x adalah mobil kita. Jika salah satu dari y dan salah satu dari z merupakan *obstacle*, maka peletakan OIL\_SPILL pada blok x akan menutup jalur untuk musuh sehingga musuh harus menabrak *obstacle*.

```
function shouldUseOil(gameState: GameState) -> boolean
Algoritma
    car <- ambil data mobil pemain
    i <- blok mobil sekarang
    if (car.lane = 1) then
        lane <- ambil data lane sebelah kanannya
        if (isObstacle(lane[i- 1]) or isObstacle(lane[i]) or
isObstacle(lane[i + 1]))
            -> true
        else if (car.lane = 4) then
            lane <- ambil data lane sebelah kirinya
            if (isObstacle(lane[i- 1]) or isObstacle(lane[i]) or
isObstacle(lane[i + 1]))
                -> true
        else
            lane1 <- ambil data lane sebelah kanan
            lane2 <- ambil data lane sebelah kiri
            kondisi1 <- false; kondisi2 <- false
            { kondisi1 dan kondisi2 adalah kondisi untuk lane1 dan
lane2 }
            if (isObstacle(lane1[i- 1]) or isObstacle(lane1[i]) or
isObstacle(lane1[i + 1]))
                kondisi1 <- true
            if (isObstacle(lane2[i- 1]) or isObstacle(lane2[i]) or
isObstacle(lane2[i + 1]))
                kondisi2 <- true
            -> kondisi1 and kondisi2
        -> false
```

#### 4.1.7 shouldCarAttack

Fungsi shouldCarAttack menganalisis apa *command attack* yang sebaiknya dilakukan apabila lajur terbaik adalah untuk tidak melakukan apa-apa (DO\_NOTHING). Prioritasnya adalah menggunakan *command* USE\_EMP > USE\_TWEET > USE\_OIL.

EMP digunakan ketika musuh berada di depan. Tweet diletakkan pada 'tebakan' posisi musuh yaitu kecepatan jika ACCELERATE ditambah 3 dan mobil berada di depan lawan (agar tidak menghalangi jalan kita). Oil diletakkan jika memiliki > 2, jika musuh dekat di belakang, atau jika shouldUseOil bernilai true.

```
function shouldCarAttack(gameState: GameState) -> command
Algoritma
```

```

    car <- ambil data mobil pemain; opponent <- ambil data mobil
lawan
    if (car.punyaEMP) then
        { Jika beda 1 lane, langsung EMP }
        if (opponent.block > car.block and |opponent.lane -
car.lane| = 1)
            -> EMP
        { Jika di lane yang sama, pastikan kita tidak menabrak
mobil musuh setelah EMP }
        if (opponent.lane = car.lane and car.block + car.speed <
opponent.block)
            -> EMP
        if (car.punyaTWEET) then
            if (car.block > opponent.block) then
                -> TWEET di opponent.lane pada block opponent.block
+ getAcceleratingSpeed(opponent.speed) + 3
            if (car.punyaOil and car.block > opponent.block) then
                if (car.banyakOil > 2) { Oil banyak }
                    -> OIL
                if (opponent.lane = car.lane and car.block -
opponent.block <= 15) { Musuh dekat }
                    -> OIL
                if (shouldUseOil(gameState))
                    -> OIL
            { Jika tidak bisa melakukan command attack, DO_NOTHING }
            -> DO_NOTHING

```

#### 4.1.8 run

Fungsi *run* adalah fungsi utama dari bot yang dipanggil oleh *game runner*, yaitu otak dari bot yang kami buat. Kode ini merupakan implementasi dari heuristik-heuristik dan logika bergerak yang sudah dijelaskan sebelumnya pada penjelasan alternatif 8.

```

function run(gameState: GameState) -> command
Algoritma
    car <- ambil data mobil pemain

    { Heuristik #1. Jika mobil di depan garis finish dan accelerate
membuatnya finish, lakukan. }
    { getAcceleratingSpeedWithDamage sama seperti
getAcceleratingSpeed, tapi mempertimbangkan damage. }
    if (car.block + getAcceleratingSpeedWithDamage(gameState) >=
1500) then
        -> ACCELERATE

    { Logika umum: Jika damage terlalu besar, maka harus FIX dahulu
}
    if (car.damage >= 5)

```

```

-> FIX

{ Heuristik #2. Jika mobil berhenti, upayakan ACCELERATE atau BOOST }
if (car.speed = 0)
    { Kalau punya BOOST dan penggunaan BOOST memberikan damage <= 1, lakukan BOOST }
    if (car.punyaBOOST AND getLaneScore(gameState, "BOOST").get(0) <= 1) then
        -> BOOST
    -> ACCELERATE

{ Heuristik #3. FIX secepat mungkin untuk mendapat kecepatan maksimal }
if (car.damage >= 2 or car.damage >= 1 AND car.punyaBOOST)
    -> FIX

{ Heuristik #4. Jika lebih baik melakukan EMP, lakukan EMP }
if (shouldEMPEarly(gameState))
    -> EMP

{ Logika bergerak }
nothingScore <- getLaneScore(gameState, "DO_NOTHING")
accelScore <- getLaneScore(gameState, "ACCELERATE")
if (car.speed == 9 || car.boostCounter tinggal 1) then
    { Decelerate hanya dilakukan jika kecepatan sudah tinggi }
    decelScore <- getLaneScore(gameState, "DECELERATE")
if (car.lane != 1) then
    { Bukan paling kiri, bisa cari skor sebelah kiri }
    leftScore <- getLaneScore(gameState, "TURN_LEFT")
if (car.lane != 4) then
    { Bukan paling kanan, bisa cari skor sebelah kanan }
    rightScore <- getLaneScore(gameState, "TURN_RIGHT")
boostScore <- getLaneScore(gameState, "BOOST")
lizardScore <- getLaneScore(gameState, "LIZARD")
{ Definisikan maxScore }
maxScore <- pemanggilan getBestLane berulang kepada nothingScore, accelScore, decelScore, leftScore, dan rightScore
{ Cek apabila BOOST lebih baik dari maxScore }
if (getBestLane(maxScore, boostScore) = boostScore) then
    -> BOOST
{ Cek apabila LIZARD lebih baik dari maxScore }
if (getBestLane(maxScore, lizardScore) = lizardScore) then
    -> LIZARD
{ Apabila maxScore bukan BOOST atau LIZARD, pilih command yang bersesuaian }
if (maxScore = accelScore)
    if (myCar.speed >= 9)
        { Sudah kecepatan maksimum, lakukan logic attacking }
}

-> shouldCarAttack(gameState)

```

```

        -> ACCELERATE
    if (maxScore = nothingScore)
        -> shouldCarAttack(gameState)
    if (maxScore = rightScore)
        -> TURN_RIGHT
    if (maxScore = leftScore)
        -> TURN_LEFT
    if (maxScore = decelScore)
        -> DECELERATE

```

## 4.2 Struktur Data pada Program Bot

### 4.2.1 command

#### 4.2.1.1 AccelerateCommand

```

public class AccelerateCommand implements Command {

    @Override
    public String render();
}

```

#### 4.2.1.2 BoostCommand

```

public class BoostCommand implements Command {

    @Override
    public String render();
}

```

#### 4.2.1.3 ChangeLaneCommand

```

public class ChangeLaneCommand implements Command {

    private Direction direction;

    public ChangeLaneCommand(int laneIndicator;
    @Override
    public String render();
}

```



#### 4.2.1.4 Command

```
public interface Command {  
    String render();  
}
```

#### 4.2.1.5 DecelerateCommand

```
public class DecelerateCommand implements Command {  
  
    @Override  
    public String render();  
}
```

#### 4.2.1.6 DoNothingCommand

```
public class DoNothingCommand implements Command {  
    @Override  
    public String render();  
}
```

#### 4.2.1.7 EmpCommand

```
public class EmpCommand implements Command {  
  
    @Override  
    public String render();  
}
```

#### 4.2.1.8 FixCommand

```
public class FixCommand implements Command {  
  
    @Override  
    public String render();  
}
```

#### 4.2.1.9 LizardCommand

```
public class LizardCommand implements Command {  
  
    @Override  
    public String render();  
}
```

#### 4.2.1.10 OilCommand

```
public class OilCommand implements Command {  
  
    @Override
```

```

    public String render();
}

```

#### 4.2.1.11 TweetCommand

```

public class TweetCommand implements Command {
    private int lane;
    private int block;

    public TweetCommand (int lane, int block);
    @Override
    public String render();
}

```

### 4.2.2 entities

#### 4.2.2.1 Car

```

public class Car {
    @SerializedName("id")
    public int id;

    @SerializedName("position")
    public Position position;

    @SerializedName("speed")
    public int speed;

    @SerializedName("damage")
    public int damage;

    @SerializedName("state")
    public State state;

    @SerializedName("powerups")
    public PowerUps[] powerups;

    @SerializedName("boosting")
    public Boolean boosting;

    @SerializedName("boostCounter")
    public int boostCounter;
}

```

#### 4.2.2.2 Gamestate

```

public class GameState {

    @SerializedName("currentRound")
    public int currentRound;
}

```

```

    @SerializedName("maxRounds")
    public int maxRounds;

    @SerializedName("player")
    public Car player;

    @SerializedName("opponent")
    public Car opponent;

    @SerializedName("worldMap")
    public List<Lane[]> lanes;
}

```

#### 4.2.2.3 Lane

```

public class Lane {
    @SerializedName("position")
    public Position position;

    @SerializedName("surfaceObject")
    public Terrain terrain;

    @SerializedName("occupiedByPlayerId")
    public int occupiedByPlayerId;

    @SerializedName("isOccupiedByCyberTruck")
    public boolean isOccupiedByCyberTruck;
}

```

#### 4.2.2.4 Position

```

public class Position {
    @SerializedName("y")
    public int lane;

    @SerializedName("x")
    public int block;
}

```

### 4.2.3 enums

#### 4.2.3.1 Direction

```

public enum Direction {

    FORWARD(0, 1, "FORWARD"),
    BACKWARD(0, -1, "BACKWARD"),
    LEFT(-1, 0, "LEFT"),
    RIGHT(1, 0, "RIGHT");
}

```

```

public final int lane;
public final int block;
public final String label;

Direction(int lane, int block, String label);
public String getLabel();
}

```

#### 4.2.3.2 PowerUps

```

public enum PowerUps {
    @SerializedName("BOOST")
    BOOST,
    @SerializedName("OIL")
    OIL,
    @SerializedName("TWEET")
    TWEET,
    @SerializedName("LIZARD")
    LIZARD,
    @SerializedName("EMP")
    EMP;
}

```

#### 4.2.3.3 State

```

public enum State {
    @SerializedName("ACCELERATING")
    ACCELERATING,
    @SerializedName("READY")
    READY,
    @SerializedName("NOTHING")
    NOTHING,
    @SerializedName("TURNING_RIGHT")
    TURNING_RIGHT,
    @SerializedName("TURNING_LEFT")
    TURNING_LEFT,
    @SerializedName("HIT_MUD")
    HIT_MUD,
    @SerializedName("HIT_OIL")
    HIT_OIL,
    @SerializedName("DECELERATING")
    DECELERATING,
    @SerializedName("PICKED_UP_POWERUP")
    PICKED_UP_POWERUP,
    @SerializedName("USED_BOOST")
    USED_BOOST,
    @SerializedName("USED_OIL")
    USED_OIL,
    @SerializedName("USED_LIZARD")
    USED_LIZARD,
    @SerializedName("USED_TWEET")
    USED_TWEET,
}

```

```

@SerializedName("HIT_WALL")
HIT_WALL,
@SerializedName("HIT_CYBER_TRUCK")
HIT_CYBER_TRUCK,
@SerializedName("HIT_EMP")
HIT_EMP,
@SerializedName("USED_EMP")
USED_EMP,
@SerializedName("FINISHED")
FINISHED
}

```

#### 4.2.3.4 Terrain

```

public enum Terrain {
    @SerializedName("0")
    EMPTY,
    @SerializedName("1")
    MUD,
    @SerializedName("2")
    OIL_SPILL,
    @SerializedName("3")
    OIL_POWER,
    @SerializedName("4")
    FINISH,
    @SerializedName("5")
    BOOST,
    @SerializedName("6")
    WALL,
    @SerializedName("7")
    LIZARD,
    @SerializedName("8")
    TWEET,
    @SerializedName("9")
    EMP
}

```

### 4.2.4 Bot

#### 4.2.4.1 Bot

```

public class Bot {
    /* TO DO: shouldCarBoost, shouldCarEMP, fungsionalitas DECELERATE */
    private List<Command> directionList = new ArrayList<>();

    private final Random random;

    private final static Command ACCELERATE = new AccelerateCommand();
    private final static Command DECELERATE = new DecelerateCommand();
    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 FIX = new FixCommand();

private final static Command TURN_RIGHT = new ChangeLaneCommand(1);
private final static Command TURN_LEFT = new ChangeLaneCommand(-1);

private final static Command DO_NOTHING = new DoNothingCommand();
public Bot();

private int obstacleAmount(List<Object> blocks);

private int powerUpAmount(PowerUps powerUpToCheck, PowerUps[] available);
private int getAcceleratingSpeedWithDamage(GameState gameState);
public Command run(GameState gameState);
private Boolean isLaneObstacle(Lane l);
private Boolean shouldEMPEarly(GameState gameState);
private Boolean shouldUseOil(GameState gameState);
private Command shouldCarAttack(GameState gameState);
private Boolean hasPowerUp(PowerUps powerUpToCheck, PowerUps[]
available);
private int getAcceleratingSpeed(int speed, int boostCounter);
private int getDeceleratingSpeed(int speed);
private ArrayList<Integer> badLane();
private ArrayList<Integer> getLaneScore(GameState gameState, String
type);
private ArrayList<Integer> getBestLane(ArrayList<Integer> lane1,
ArrayList<Integer> lane2);

```

## 4.2.5 Main

### 4.2.5.1 main

```

public class Main {

    private static final String ROUNDS_DIRECTORY = "rounds";
    private static final String STATE_FILE_NAME = "state.json";

    public static void main(String[] args);

```

## 4.3 Analisis Hasil Pengujian

Bot didesain sedeterministik mungkin (tidak ada penggunaan *random*, benar-benar ada alasan yang jelas untuk melakukan setiap hal) untuk menciptakan kekonsistenan dan memaksimalkan pilihan yang bisa ia dapat. Bot dapat melakukan apa yang diinginkan oleh alternatif 8 secara baik, contoh, pada algoritma *pathfinding*:

## Round 029

[Reset](#) [Remove this match](#)

[227, 1]	[228, 1]	[229, 1]	[230, 1] <SUD>	[231, 1]	[232, 1] 95,1738>	[233, 1]	[234, 1]	[235, 1] <SUD>	[236, 1]	[237, 1]	[238, 1]	[239, 1]	[240, 1] <SUD>	[241, 1] <SUD>	[242, 1]	[243, 1]	[244, 1]	[245, 1]	[246, 1]	[247, 1]	[248, 1]	[249, 1]	[250, 1]	[251, 1]	[252, 1]
[227, 2]	[228, 2] <SUD>	[229, 2] <SUD>	[230, 2]	[231, 2]	[232, 2]	[233, 2]	[234, 2]	[235, 2] <SUD>	[236, 2]	[237, 2]	[238, 2]	[239, 2]	[240, 2]	[241, 2]	[242, 2]	[243, 2] <SUD>	[244, 2]	[245, 2]	[246, 2]	[247, 2]	[248, 2]	[249, 2]	[250, 2]	[251, 2]	[252, 2]
[227, 3]	[228, 3]	[229, 3]	[230, 3]	[231, 3]	[232, 3]	[233, 3]	[234, 3]	[235, 3]	[236, 3]	[237, 3]	[238, 3]	[239, 3]	[240, 3]	[241, 3]	[242, 3] <SUD>	[243, 3]	[244, 3]	[245, 3]	[246, 3]	[247, 3]	[248, 3]	[249, 3]	[250, 3]	[251, 3]	[252, 3]
[227, 4] <SUD>	[228, 4]	[229, 4]	[230, 4]	[231, 4]	[232, 4]	[233, 4]	[234, 4]	[235, 4]	[236, 4]	[237, 4] <SUD>	[238, 4]	[239, 4]	[240, 4]	[241, 4]	[242, 4]	[243, 4]	[244, 4]	[245, 4]	[246, 4]	[247, 4]	[248, 4] <SUD>	[249, 4]	[250, 4]	[251, 4]	[252, 4] <SUD>

[First](#) [Prev](#) [29](#) [Next](#) [Last](#)

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

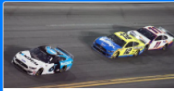






## Round Details

Max Rounds: 600

Current Round: 29

## A - mobil-mogok

(selected) ✓

						
Position <b>2</b> [x: 232, y: 1]	Speed <b>9</b>	Lane <b>1</b>	Distance <b>232</b>	Boosts <b>0</b>	Boosting <b>No</b>	Powerups TWEET,EMP,EMP,TWE ET,LIZARD,EMP

## Bot Command

Command: TURN\_RIGHT

Execution time: 1ms

Exception: null

Gambar 3. Salah satu contoh *pathfinding* yang bot lakukan

Pada kondisi ini, bot yang berwarna biru lebih memilih untuk melakukan TURN\_RIGHT karena memanggil DO\_NOTHING atau ACCELERATE akan menambah *damage* sebesar 2 (menabrak 2 MUD) sedangkan *command* TURN\_RIGHT hanya akan menabrak satu buah MUD. Kasus lain dari algoritma *pathfinding* ditunjukkan oleh kasus berikut:

[816, 1]	[817, 1]	[818, 1]	[819, 1]	[820, 1]	[821, 1]	[822, 1]	[823, 1]	[824, 1]	[825, 1]	[826, 1]	[827, 1]	[828, 1]	[829, 1]	[830, 1]	[831, 1]	[832, 1]	[833, 1]	[834, 1]	[835, 1]	[836, 1]	[837, 1]	[838, 1]	[839, 1]	[840, 1]	[841, 1]	[842, 1]	[843, 1]	[844, 1]	[845, 1]	[846, 1]	[847, 1]	[848, 1]	[849, 1]	[850, 1]	[851, 1]	[852, 1]	[853, 1]	[854, 1]	[855, 1]	[856, 1]	[857, 1]	[858, 1]	[859, 1]	[860, 1]	[861, 1]
[816, 2]	[817, 2]	[818, 2]	[819, 2]	[820, 2]	[821, 2]	[822, 2]	[823, 2]	[824, 2]	[825, 2]	[826, 2]	[827, 2]	[828, 2]	[829, 2]	[830, 2]	[831, 2]	[832, 2]	[833, 2]	[834, 2]	[835, 2]	[836, 2]	[837, 2]	[838, 2]	[839, 2]	[840, 2]	[841, 2]	[842, 2]	[843, 2]	[844, 2]	[845, 2]	[846, 2]	[847, 2]	[848, 2]	[849, 2]	[850, 2]	[851, 2]	[852, 2]	[853, 2]	[854, 2]	[855, 2]	[856, 2]	[857, 2]	[858, 2]	[859, 2]	[860, 2]	[861, 2]
[816, 3]	[817, 3]	[818, 3]	[819, 3]	[820, 3]	[821, 3]	[822, 3]	[823, 3]	[824, 3]	[825, 3]	[826, 3]	[827, 3]	[828, 3]	[829, 3]	[830, 3]	[831, 3]	[832, 3]	[833, 3]	[834, 3]	[835, 3]	[836, 3]	[837, 3]	[838, 3]	[839, 3]	[840, 3]	[841, 3]	[842, 3]	[843, 3]	[844, 3]	[845, 3]	[846, 3]	[847, 3]	[848, 3]	[849, 3]	[850, 3]	[851, 3]	[852, 3]	[853, 3]	[854, 3]	[855, 3]	[856, 3]	[857, 3]	[858, 3]	[859, 3]	[860, 3]	[861, 3]
[816, 4]	[817, 4]	[818, 4]	[819, 4]	[820, 4]	[821, 4]	[822, 4]	[823, 4]	[824, 4]	[825, 4]	[826, 4]	[827, 4]	[828, 4]	[829, 4]	[830, 4]	[831, 4]	[832, 4]	[833, 4]	[834, 4]	[835, 4]	[836, 4]	[837, 4]	[838, 4]	[839, 4]	[840, 4]	[841, 4]	[842, 4]	[843, 4]	[844, 4]	[845, 4]	[846, 4]	[847, 4]	[848, 4]	[849, 4]	[850, 4]	[851, 4]	[852, 4]	[853, 4]	[854, 4]	[855, 4]	[856, 4]	[857, 4]	[858, 4]	[859, 4]	[860, 4]	[861, 4]

[First] < Prev 72 Next > [Last]

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








## Round Details

Max Rounds: 600

Current Round: 72

## A - mobil-mogok

(selected) v

						
Position 1 [x: 641, y: 3]	Speed 9	Lane 3	Distance 641	Boosts 0	Boosting No	Powerups OIL,OIL,BOOST,BOOS T,TWEET,TWEET,OIL,B OOST,TWEET,OIL,EM P,BOOST,OIL,LIZARD,L IZARD,TWEET,OIL,OIL ,LIZARD,TWEET
<b>Bot Command</b> Command: USE_BOOST Execution time: 1ms						

Gambar 4. Salah satu contoh *pathfinding* yang bot lakukan

Dalam kasus ini, TURN\_LEFT, TURN\_RIGHT, ACCELERATE, DO\_NOTHING, DECELERATE, dan BOOST sama-sama memiliki skor *damage* 1. Bot memilih BOOST karena memaksimalkan jumlah *powerup* yang diambil (1) serta jarak tempuh yang paling jauh.

Kelakuan *offensive* dari bot juga teroptimasi dengan baik, contoh penggunaan TWEET pada mobil lawan yang berwarna merah ini. Dua gambar di bawah ini menunjukkan fungsionalitas TWEET dapat memprediksi gerakan musuh dengan benar untuk memblokir lajur lawan.

## Round 074

Res

[800, 1]	[801, 1]	[802, 1]	[803, 1]	[804, 1]	[805, 1]	[806, 1]	[807, 1]	[808, 1]	[809, 1]	[810, 1]	[811, 1]	[812, 1]	[813, 1]	[814, 1]	[815, 1]	[816, 1]	[817, 1]	[818, 1]	[819, 1]	[820, 1]	[821, 1]	[822, 1]	[823, 1]	[824, 1]	[825, 1]
[800, 2]	[801, 2]	[802, 2]	[803, 2]	[804, 2]	[805, 2]	[806, 2]	[807, 2]	[808, 2]	[809, 2]	[810, 2]	[811, 2]	[812, 2]	[813, 2]	[814, 2]	[815, 2]	[816, 2]	[817, 2]	[818, 2]	[819, 2]	[820, 2]	[821, 2]	[822, 2]	[823, 2]	[824, 2]	[825, 2]
[800, 3]	[801, 3]	[802, 3]	[803, 3]	[804, 3]	[805, 3]	[806, 3]	[807, 3]	[808, 3]	[809, 3]	[810, 3]	[811, 3]	[812, 3]	[813, 3]	[814, 3]	[815, 3]	[816, 3]	[817, 3]	[818, 3]	[819, 3]	[820, 3]	[821, 3]	[822, 3]	[823, 3]	[824, 3]	[825, 3]
[800, 4]	[801, 4]	[802, 4]	[803, 4]	[804, 4]	[805, 4]	[806, 4]	[807, 4]	[808, 4]	[809, 4]	[810, 4]	[811, 4]	[812, 4]	[813, 4]	[814, 4]	[815, 4]	[816, 4]	[817, 4]	[818, 4]	[819, 4]	[820, 4]	[821, 4]	[822, 4]	[823, 4]	[824, 4]	[825, 4]

Gambar 5. Mobil lawan yang memiliki *path* kosong di depannya



Round 075

Reset

[809, 1]	[810, 1]	[811, 1]	[812, 1]	[813, 1]	[814, 1]	[815, 1]	[816, 1]	[817, 1]	[818, 1]	[819, 1]	[820, 1]	[821, 1]	[822, 1]	[823, 1]	[824, 1]	[825, 1]	[826, 1]	[827, 1]	[828, 1]	[829, 1]	[830, 1]	[831, 1]	[832, 1]	[833, 1]	[834, 1]
[809, 2]	[810, 2]	[811, 2]	[812, 2]	[813, 2]	[814, 2]	[815, 2]	[816, 2]	[817, 2]	[818, 2]	[819, 2]	[820, 2]	[821, 2]	[822, 2]	[823, 2]	[824, 2]	[825, 2]	[826, 2]	[827, 2]	[828, 2]	[829, 2]	[830, 2]	[831, 2]	[832, 2]	[833, 2]	[834, 2]
[809, 3]	[810, 3]	[811, 3]	[812, 3]	[813, 3]	[814, 3]	[815, 3]	[816, 3]	[817, 3]	[818, 3]	[819, 3]	[820, 3]	[821, 3]	[822, 3]	[823, 3]	[824, 3]	[825, 3]	[826, 3]	[827, 3]	[828, 3]	[829, 3]	[830, 3]	[831, 3]	[832, 3]	[833, 3]	[834, 3]
[809, 4]	[810, 4]	[811, 4]	[812, 4]	[813, 4]	[814, 4]	[815, 4]	[816, 4]	[817, 4]	[818, 4]	[819, 4]	[820, 4]	[821, 4]	[822, 4]	[823, 4]	[824, 4]	[825, 4]	[826, 4]	[827, 4]	[828, 4]	[829, 4]	[830, 4]	[831, 4]	[832, 4]	[833, 4]	[834, 4]

Gambar 6. Mobil lawan dipaksa belok ke kiri karena ada *cybertruck* yang diletakkan di depannya

Namun, penggunaan algoritma *greedy* tentu saja ada kekurangannya. Kita tidak bisa melakukan *backtracking* ataupun melihat beberapa langkah di depan kita. Kita hanya bisa melihat langkah terbaik dari langkah kita sekarang. Perhatikan gambar berikut.

Round 026

Reset

Remove this match

[209, 1]	[210, 1]	[211, 1]	[212, 1]	[213, 1]	[214, 1]	[215, 1]	[216, 1]	[217, 1]	[218, 1]	[219, 1]	[220, 1]	[221, 1]	[222, 1]	[223, 1]	[224, 1]	[225, 1]
[209, 2]	[210, 2]	[211, 2]	[212, 2]	[213, 2]	[214, 2]	[215, 2]	[216, 2]	[217, 2]	[218, 2]	[219, 2]	[220, 2]	[221, 2]	[222, 2]	[223, 2]	[224, 2]	[225, 2]
[209, 3]	[210, 3]	[211, 3]	[212, 3]	[213, 3]	[214, 3]	[215, 3]	[216, 3]	[217, 3]	[218, 3]	[219, 3]	[220, 3]	[221, 3]	[222, 3]	[223, 3]	[224, 3]	[225, 3]
[209, 4]	[210, 4]	[211, 4]	[212, 4]	[213, 4]	[214, 4]	[215, 4]	[216, 4]	[217, 4]	[218, 4]	[219, 4]	[220, 4]	[221, 4]	[222, 4]	[223, 4]	[224, 4]	[225, 4]

Gambar 7. Bot memilih untuk maju

Apabila bot dapat melihat beberapa langkah ke depan, akan lebih baik jika bot mengambil TURN\_RIGHT pada ronde tersebut agar pada ronde selanjutnya mendapat banyak *power up* serta mendapatkan lajur yang kosong. Menggunakan algoritma *greedy*, hal ini tidak bisa dilakukan.

## Bab V

# Kesimpulan dan Saran

### 1. Kesimpulan

Pada Tugas Besar IF2211 Strategi Algoritma 2021/2022 berjudul “Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan *Overdrive*,” kami berhasil membuat sebuah bot dalam permainan “Overdrive” yang memanfaatkan implementasi dari algoritma *greedy*. Program yang dibuat berasal dari beberapa alternatif algoritma *greedy* yang dikembangkan dan dikombinasikan untuk menghasilkan strategi akhir yang terbaik setelah melalui proses analisis efisiensi dan efektivitas.

### 2. Saran

Beberapa saran yang dapat kami berikan untuk Tugas Besar IF2211 Strategi Algoritma 2021/2022 ini adalah:

1. Algoritma *greedy* yang dipilih dalam Tugas Besar ini meskipun sudah melalui proses uji coba dan analisis sejatinya masih memiliki ruang untuk dikembangkan lagi, seperti meningkatkan kembali keefisienan program atau mengeksplorasi pendekatan strategi yang berbeda.
2. Spesifikasi tugas dapat lebih diperjelas agar menghindari kemungkinan adanya multitafsir dalam batasan-batasan program. Demonstrasi langsung dari permainan yang menjadi tema tugas juga dapat menjadi pendamping file spesifikasi tugas yang sangat membantu dalam pengerjaan.

## Daftar Pustaka

- [1] EntelectChallenge. (2020). 2020-Overdrive.<https://github.com/EntelectChallenge/2020-Overdrive> : Diakses pada tanggal 4 Februari 2021 pukul 21:08.
- [2] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung. 2009.

<h3>Link Repository Github Kelompok Mobil Mogok</h3>
<a href="https://github.com/acomarcho/stima-tubes-1">https://github.com/acomarcho/stima-tubes-1</a>

<h3>Link Video Demonstrasi</h3>
<a href="https://youtu.be/3ZLv24gcLLA">https://youtu.be/3ZLv24gcLLA</a>