

**LAPORAN TUGAS BESAR I  
IF2211 STRATEGI ALGORITMA**

**PEMANFAATAN ALGORITMA GREEDY  
DALAM APLIKASI PERMAINAN  
“OVERDRIVE”**



Dipersiapkan oleh:

**Kelompok Overbrain (44)**

Eiffel Aqila Amarendra 13520074

Nelsen Putra 13520130

Willy Wilsen 13520160

**Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2022**

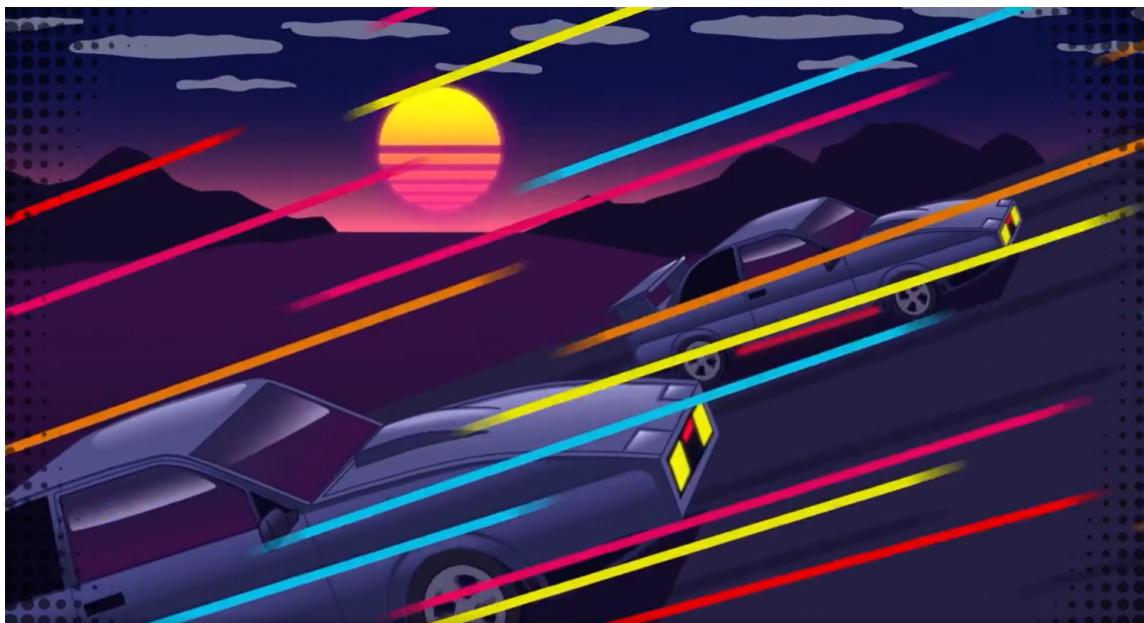
## DAFTAR ISI

<b>DAFTAR ISI</b>	<i>i</i>
<b>BAB I</b>	<b>1</b>
<b>BAB II</b>	<b>4</b>
Algoritma Greedy	4
Cara Kerja Program Secara Umum	5
<b>BAB III</b>	<b>8</b>
Pemetaan Algoritma Greedy pada Permasalahan Permainan Overdrive	8
Eksplorasi Alternatif Strategi Greedy pada Permasalahan Permainan Overdrive	13
Analisis Efisiensi dan Efektivitas Strategi Greedy	15
Pemilihan Algoritma Greedy	18
<b>BAB IV</b>	<b>20</b>
Implementasi Algoritma Greedy	20
Struktur Data Program Overdrive	27
Analisis Pengujian Algoritma Greedy	29
<b>BAB V</b>	<b>31</b>
Kesimpulan	31
Saran	32
<b>LAMPIRAN</b>	<b>34</b>
<b>DAFTAR PUSTAKA</b>	<b>35</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, kelompok diminta untuk menggunakan sebuah *game engine* yang mengimplementasikan permainan *Overdrive*. *Game engine* tersebut dapat diperoleh pada laman berikut: <https://github.com/EntelectChallenge/2020-Overdrive>.

Tugas kelompok ialah mengimplementasikan bot mobil dalam permainan *Overdrive* dengan menggunakan strategi *Greedy* untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, kami disarankan untuk 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 *blocks*. Terdapat 5 tipe *block*, yaitu *Empty*, *Mud*, *Oil Spill*, *Flimsy Wall*, dan *Finish Line* yang masing-masing karakteristik dan efek berbeda. *Block* dapat memuat *powerups* yang bisa diambil oleh mobil yang melewati *block* tersebut.
2. Beberapa *powerups* yang tersedia adalah:
  - a. *Oil item*, dapat menumpahkan oli di bawah mobil anda berada.
  - b. *Boost*, dapat mempercepat kecepatan mobil anda secara drastis.
  - c. *Lizard*, berguna untuk menghindari lizard yang mengganggu jalan mobil anda.
  - d. *Tweet*, dapat menjatuhkan truk di block spesifik yang anda inginkan.
  - e. *EMP*, dapat menembakkan EMP ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 lane yang sama) akan terus berada di lane yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.
3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 block untuk setiap round. Game state akan memberikan jarak pandang hingga 20 block di depan dan 5 block di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.
4. Terdapat command yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan powerups. Pada setiap round, masing-masing pemain dapat memberikan satu buah command untuk mobil mereka. Berikut jenis-jenis command yang ada pada permainan:
  - a. NOTHING
  - b. ACCELERATE
  - c. DECELERATE
  - d. TURN\_LEFT
  - e. TURN\_RIGHT

- f. USE\_BOOST
  - g. USE\_OIL
  - h. USE\_LIZARD
  - i. USE\_TWEET <lane> <block>
  - j. USE\_EMP
  - k. FIX
5. Command dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.
  6. Bot pemain yang pertama kali mencapai garis finish akan memenangkan pertandingan. Jika kedua bot mencapai garis finish secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar.

Adapun peraturan yang lebih lengkap dari permainan Overdrive, dapat dilihat pada laman :

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

## BAB II

### LANDASAN TEORI

#### 2.1. Algoritma *Greedy*

Algoritma *Greedy* merupakan salah satu algoritma/metode yang paling populer dalam pencarian solusi dari permasalahan yang membutuhkan optimisasi atau dengan kata lain, mencari solusi optimal. Permasalahan optimisasi dapat dibagi menjadi dua jenis, yaitu permasalahan memaksimasi (*maximization*) dan permasalahan meminimasi (*minimization*).

Algoritma *Greedy* yang memiliki prinsip “*Take what you can get now!*” dapat didefinisikan sebagai algoritma yang memecahkan permasalahan dengan cara membentuk himpunan solusi langkah demi langkah dari himpunan kosong. Pada setiap langkah, algoritma ini, sebagaimana prinsipnya, mengambil pilihan terbaik yang dapat diperoleh saat itu, tanpa memperhatikan konsekuensi ke depan dan tidak dapat mundur ke langkah sebelumnya. Kemudian, algoritma ini akan memilih solusi optimum lokal, dengan harapan, setiap langkah tersebut akan mengarah kepada optimum global. Dengan demikian, hasil solusi algoritma *greedy* belum tentu merupakan solusi optimum global (terbaik), bisa jadi merupakan solusi sub-optimum atau pseudo-optimum.

Berikut merupakan elemen-elemen yang menggambarkan sebuah Algoritma Greedy.

1. Himpunan Kandidat ( $C$ )

Himpunan yang berisi kandidat yang akan dipilih pada setiap langkah.

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 yang layak atau tidak.

6. Fungsi Obyektif

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

## 2.2. Cara Kerja Program Secara Umum

Pada projek permainan *Overdrive*, *Entelect*, penyelenggara challenge ini, menyediakan *starter pack* yang di dalamnya berisi *folder-folder* berikut, (1) *game-engine*, yang bertanggung jawab atas penegakkan berbagai aturan di dalam permainan ini dengan menerapkan berbagai *command* untuk *bot* yang valid ke dalam status permainan; (2) *game-runner*, yang bertanggung jawab atas penyelenggaraan permainan, pemanggilan *command* yang sesuai, dan menyerahkannya kepada *game-engine*; (3) *reference-bot*, yakni *bot* yang berisi kumpulan *AI logic* yang telah didefinisikan sebelumnya dan dapat digunakan sebagai lawan dalam permainan; (4) *starter-bots*, yakni *bot* sederhana yang menjadi *bot* pemain dan dapat dimodifikasi. Selain itu, sebagai tambahan, permainan ini juga dilengkapi dengan *visualizer* yang dapat diunduh secara terpisah yang berfungsi untuk menampilkan permainan dengan grafis layaknya permainan komputer lainnya.

*Game Engine*, seperti namanya, merupakan sebuah mesin yang dirancang secara khusus untuk menjalankan program atau aplikasi permainan. Pada permainan *Overdrive*, *game engine* yang digunakan dibuat dengan *Unity* dan sudah tersedia dari *starter pack* yang berasal dari *Entelect*, penyelenggara *challenge* ini. Dalam berinteraksi dengan program yang kita bangun, *game engine* ini menggunakan *file-file* berformat *.json* untuk mengambil maupun menyimpan *state* permainan dan atribut-atribut penting lainnya dalam permainan. Misalnya, pada `game-config.json`, *developer* telah mendefinisikan atribut-atribut yang akan dipakai di dalam permainan dan *game engine* menjadikan *file* ini sebagai *source file* permainan.

*Game engine* yang disediakan oleh *Entelect* telah memiliki konfigurasi *default* permainan yang disimpan dalam *file* dengan format tersebut. Artinya, kita tidak perlu lagi mengatur *settings* yang ada dalam permainan tersebut dan bisa langsung menjalankan permainan yang ada. Dalam hal ini, bot dasar permainan sudah ada melalui konfigurasi tersebut dan tidak perlu ditambahkan dengan sendirinya. Namun, kita tetap bisa melakukan perubahan pada bagian-bagian tertentu dalam permainan dengan memodifikasi *file-file .json* ini. Misalnya, untuk mengubah nama bot atau nama pemain yang kita inginkan sebagai *in-game name* kita, kita dapat mengaplikasikannya dengan mengubah atribut `nickName` di dalam file `bot.json` yang ada pada direktori Java maupun bahasa lainnya. Di *file* ini kita juga bisa mengganti direktori, nama, atau bahasa yang digunakan dalam pembuatan bot mobil pada permainan *Overdrive* ini.

Ketika kita ingin melakukan *testing* atau *sparring* dengan bot baru selain dari *Reference Bot* yang tersedia, kita bisa menambahkan bot ini dengan mengganti atribut `player-a` dan `player-b` yang ada di dalam permainan dengan direktori dari bot yang ingin dipertandingkan. Kita juga bisa mengganti target direktori yang digunakan untuk menyimpan *state* permainan tiap rondenya.

Awalnya, program bot yang kita miliki belum mempunyai atribut permainan yang lengkap. Misalnya, terdapat beberapa *command* khusus seperti USE\_BOOST, USE\_OIL, dan USE\_LIZARD, juga termasuk atribut-atribut yang digunakan untuk mengecek keberhasilan *command* ini yang sebenarnya masih belum terdefinisi di dalam bot. *Method-method* yang sudah ada pada *file* juga masih bisa diperbaiki dan dikembangkan agar dapat menjadi lebih baik.

Oleh karena itu, sebelum melakukan implementasi lebih lanjut, kelompok berinisiatif untuk melengkapi bagian-bagian yang hilang ini terlebih dahulu. Dalam bahasa Java, bagian ini dibuat dengan paradigma berorientasi objek, sehingga didefinisikan kelas-kelas baru untuk menampung atribut tersebut. Kita tidak bisa langsung membuat atribut baru yang ada di dalam permainan. Hal ini akan mengakibatkan atribut tersebut menjadi tidak terdefinisi. Dengan demikian, program

harus mengambil atribut-atribut yang ada di dalam file .json yang digunakan *game engine*. Hal ini dilakukan dengan menggunakan anotasi `SerializedName` pada JavaJSONObject untuk mengambil atribut-atribut pada file .json tersebut dan memasukkannya pada atribut kelas tertentu tanpa perlu diinisialisasi. Dengan demikian, program yang kita bangun sekarang bisa berinteraksi dengan *state game engine* dan permainan tersebut dengan perantara file .json ini.

Untuk memudahkan dalam proses *build* program, dapat digunakan IntelliJ IDEA. IntelliJ IDEA telah menyediakan *Maven Toolbox* dan kita bisa memanfaatkan fitur `Install` di dalam `Lifecycle` untuk menghasilkan file .jar. File ini akan digunakan dalam proses *running* permainan. Dengan melakukan konfigurasi yang telah disebutkan sebelumnya, kita bisa melakukan *running* dengan menjalankan program `run.bat`. Program ini akan menjalankan *game engine* secara otomatis pada *Command Prompt* atau Terminal dan permainan akan ditampilkan dalam format *text-based*.

## BAB III

### APLIKASI STRATEGI *GREEDY*

#### 3.1. Pemetaan Algoritma *Greedy* pada Permasalahan Permainan *Overdrive*

##### 3.1.1. Pemetaan Umum Permasalahan Permainan

Elemen Algoritma <i>Greedy</i>	Pemetaan pada Permainan <i>Overdrive</i>
Himpunan Kandidat ( $C$ )	Perintah-perintah yang mungkin dijalankan oleh bot mobil di dalam permainan. Perintah yang dimaksud dapat merupakan perintah untuk meningkatkan kecepatan, memaksimalkan kecepatan, memperbaiki damage pada mobil, menjalankan turbo, mengambil power-ups, menggunakan power-ups untuk mengusik lawan.
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap komponen-komponen permainan, misalnya jumlah <i>damage</i> mobil saat ini, <i>speed</i> mobil saat ini, ketersediaan <i>boost</i> , ketersediaan <i>lizard</i> , posisi <i>obstacle</i> dari mobil, posisi dan ketersediaan <i>power-ups</i> lainnya seperti <i>OIL</i> , <i>EMP</i> , dan <i>TWEET</i> , maupun valid atau tidaknya koordinat yang ingin dituju oleh mobil pemain saat melakukan

	pergerakan.
Fungsi Obyektif	Memenangkan permainan <i>Overdrive</i> , baik dengan mencapai garis <i>finish</i> pertama kali maupun dengan mengumpulkan poin sebanyak-banyaknya dengan harapan bahwa ketika mencapai ronde akhir, yakni ketika mobil pemain dengan mobil lawan mencapai garis <i>finish</i> di saat yang bersamaan, total poin yang kita miliki dapat lebih banyak dibandingkan poin lawan sehingga kita dinyatakan memenangkan permainan.

### 3.1.2. Pemetaan Strategi Pemrioritasan Perbaikan ke dalam Algoritma *Greedy*

Elemen Algoritma <i>Greedy</i>	Pemetaan pada Permainan <i>Overdrive</i>
Himpunan Kandidat ( $C$ )	Perintah-perintah yang berkaitan dengan pemaksimalan perbaikan dalam permainan <i>Overdrive</i> , antara lain <i>FIX</i> .
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung dan mobil terkena <i>damage</i> .
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari jumlah <i>damage</i> yang diterima mobil.
Fungsi Obyektif	Memaksimalkan perbaikan yang bisa dilakukan mobil pada setiap ronde permainan. Dalam hal ini, juga harus dimaksimalkan efek kecepatan

	maksimal yang bisa didapat dari <i>boost</i> .
--	--

3.1.3. Pemetaan Strategi Pemrioritasan Penghindaran *Obstacle* ke dalam Algoritma *Greedy*

Elemen Algoritma <i>Greedy</i>	Pemetaan pada Permainan <i>Overdrive</i>
Himpunan Kandidat ( $C$ )	Perintah-perintah yang berkaitan dengan pemaksimalan penghindaran <i>obstacle</i> dalam permainan <i>Overdrive</i> , antara lain <i>TURN_RIGHT</i> , <i>TURN_LEFT</i> , dan <i>LIZARD</i> .
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung dan terdapat <i>obstacle</i> di sepanjang jalur mobil.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari <i>obstacle</i> yang terdapat di sepanjang jalur mobil dan ketersediaan <i>lizard</i> pada mobil.
Fungsi Obyektif	Memaksimalkan penghindaran <i>obstacle</i> yang bisa dilakukan mobil pada setiap ronde permainan. Dalam hal ini, juga harus dimaksimalkan kemungkinan mengambil jalur yang yang paling menguntungkan untuk mobil.

3.1.4. Pemetaan Strategi Pemrioritasan Pemaksimalan Kecepatan ke dalam Algoritma *Greddy*

Elemen Algoritma <i>Greddy</i>	Pemetaan pada Permainan <i>Overdrive</i>
Himpunan Kandidat ( $C$ )	Perintah-perintah yang berkaitan dengan pemaksimalan kecepatan dalam permainan <i>Overdrive</i> , antara lain <i>ACCELERATE</i> dan <i>USE_BOOST</i> .
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung dan mobil belum mencapai kecepatan maksimal.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari jumlah <i>damage</i> , kecepatan, dan ketersediaan <i>boost</i> pada mobil.
Fungsi Obyektif	Memaksimalkan kecepatan yang bisa didapatkan mobil pada setiap ronde permainan. Dalam hal ini, juga harus diminimalisir kemungkinan untuk mobil tidak melakukan apa-apa (karena tidak menghasilkan poin dan merugikan) dan mengurangi bahaya yang mungkin ditemui pada setiap ronde permainan.

3.1.5. Pemetaan Strategi Pemrioritasan Pencarian *Power-Ups* ke dalam Algoritma *Greedy*

Elemen Algoritma <i>Greedy</i>	Pemetaan pada Permainan <i>Overdrive</i>
Himpunan Kandidat ( $C$ )	Perintah-perintah yang berkaitan dengan pencarian power-ups dalam permainan <i>Overdrive</i> , antara lain <i>TURN_RIGHT</i> dan <i>TURN_LEFT</i> .
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung dan terdapat power-ups di sepanjang jalur mobil.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari power-ups yang terdapat di sepanjang jalur mobil.
Fungsi Obyektif	Memaksimalkan pencarian power-ups yang bisa didapatkan mobil pada setiap ronde permainan. Dalam hal ini, juga harus diminimalisir kemungkinan mobil untuk menabrak <i>obstacle</i> saat mengambil power-ups.

3.1.6. Pemetaan Strategi Pemrioritasan Pemanfaatan *Power-Ups* ke dalam Algoritma *Greedy*

Elemen Algoritma <i>Greedy</i>	Pemetaan pada Permainan <i>Overdrive</i>
--------------------------------	--

Himpunan Kandidat ( $C$ )	Perintah-perintah yang berkaitan dengan pemanfaatan power-ups dalam permainan <i>Overdrive</i> , antara lain <i>TWEET</i> , <i>EMP</i> , dan <i>OIL</i> .
Himpunan Solusi ( $S$ )	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung dan tidak terdapat <i>obstacle</i> di sepanjang jalur mobil serta mobil sudah mencapai kecepatan maksimal.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan oleh <i>game engine</i> .
Fungsi Seleksi ( <i>Selection Function</i> )	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam <i>bot</i> yang telah dibangun dalam implementasi permainan <i>Overdrive</i> . Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan ( <i>Feasibility</i> )	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap validitas dari posisi musuh dan ketersediaan power-ups yang dimiliki mobil.
Fungsi Obyektif	Memaksimalkan pemanfaatan power-ups yang bisa didapatkan mobil pada setiap ronde permainan. Dalam hal ini, juga harus dimaksimalkan kemungkinan mobil untuk menggunakan power-ups dengan tepat sasaran.

### 3.2. Eksplorasi Alternatif Strategi *Greedy* pada Permasalahan Permainan *Overdrive*

*Overdrive* memiliki mekanisme permainan yang tergolong cukup kompleks. Kompleksitas tersebut membuka peluang bagi kelompok kami untuk menemukan beragam strategi yang dapat diimplementasikan dengan tujuan untuk memenangkan permainan ini.

#### 3.2.1. Strategi Pemrioritasan Perbaikan

Strategi Pemrioritasan Perbaikan atau *Save The Car Greedy* merupakan strategi untuk mempertahankan mobil dalam kondisi tidak memiliki *damage* sama sekali atau *damage* sebesar 0 dengan cara memperbaiki mobil ketika menerima *damage*. Pada setiap ronde, jika mobil menabrak *obstacle*, pilih *command FIX* hingga *damage* pada mobil mencapai 0. Jika mobil tidak menerima *damage* atau *damage* sebesar 0, strategi ini tidak dapat digunakan.

### 3.2.2. Strategi Pemrioritasan Penghindaran *Obstacle*

Strategi Pemrioritasan Penghindaran *Obstacle* atau *Avoid Greedy* merupakan strategi untuk mempertahankan mobil dalam kondisi tidak memiliki *damage* sama sekali atau *damage* sebesar 0 dengan cara menghindari *obstacle*. Pada setiap ronde, jika mobil melihat adanya *obstacle* di hadapannya, mobil akan berbelok ke jalur yang bersebelahan, yakni arah kiri atau kanan, yang tidak memiliki *obstacle* di jalur tersebut dengan memilih *command TURN\_LEFT* atau *TURN\_RIGHT*. Jika terdapat *obstacle* di hadapan dan jalur sebelah mobil dan mobil memiliki *LIZARD*, mobil akan memilih *command USE\_LIZARD* sedangkan jika mobil tidak memiliki *LIZARD*, mobil akan memilih jalur dengan *damage* yang paling rendah untuk berbelok. Jika mobil tidak melihat adanya *obstacle* di hadapannya, strategi ini tidak dapat digunakan.

### 3.2.3. Strategi Pemrioritasan Pemaksimalan Kecepatan

Strategi Pemrioritasan Pemaksimalan Kecepatan atau *Max Speed and Turbo Greedy* merupakan strategi untuk meningkatkan kecepatan *bot* mobil hingga mencapai batas kecepatan maksimal. Pada setiap ronde, jika mobil tidak memiliki *damage* sebesar 5, pilih *command USE\_BOOST* jika memiliki *BOOST* atau *ACCELERATE* jika tidak dengan tujuan untuk mempercepat kecepatan mobil hingga mencapai *MAX SPEED*. Jika mobil memiliki *damage* sebesar 5, strategi ini tidak dapat digunakan.

### 3.2.4. Strategi Pemrioritasan Pencarian *Power-Ups*

Strategi Pemrioritasan Pencarian *Power-Ups* atau *GetPowerUp Greedy* merupakan strategi untuk mencari power-ups yang bisa didapatkan mobil pada setiap ronde permainan. Pada setiap ronde, jika mobil menemukan power up di jalur hadapannya, mobil akan tetap melaju. Namun, jika mobil menemukan *power up* di jalur sebelah kiri, mobil akan memanggil *command TURN\_LEFT*, sebaliknya jika menemukan *power up* di jalur sebelah kanan, mobil akan memanggil *command TURN\_RIGHT*. Jika tidak ada *power up* di sekitar mobil, strategi ini tidak dapat digunakan.

### 3.2.5. Strategi Pemrioritasan Pemanfaatan *Power-Ups*

Strategi Pemrioritasan Pemanfaatan *Power-Ups* atau *USE\_TWEET - USE\_OIL - USE\_EMP Greedy* merupakan strategi untuk memaksimalkan pemanfaatan power-ups yang bisa didapatkan mobil pada setiap ronde permainan. Penggunaan *power up* memiliki syarat, yakni mobil telah mencapai kecepatan maksimum dan tidak terdapat *obstacle* di hadapan mobil. Pada setiap ronde, jika syarat terpenuhi dan mobil memiliki *power up* TWEET, mobil akan memanggil *command USE\_TWEET* dan menempatkan *Cyber Truck* di jalur hadapan mobil musuh. Jika syarat terpenuhi, mobil memiliki *power up* OIL, dan musuh berada di belakang mobil pemain, mobil akan memanggil *command USE\_OIL*. Jika syarat terpenuhi dan mobil memiliki *power up* EMP, dan musuh berada di depan mobil pemain, mobil akan memanggil *command USE\_EMP*. Jika kondisi di atas tidak dipenuhi, strategi ini tidak dapat digunakan.

## 3.3. Analisis Efisiensi dan Efektivitas Strategi *Greedy*

### 3.3.1. Analisis Efisiensi

#### 3.3.1.1. Strategi Pemrioritasan Perbaikan

Untuk strategi pemrioritasan perbaikan, mobil hanya perlu memeriksakan pertama-tama akan memeriksa *damage* dan *current speed* atau pemeriksaan keberadaan *power up* BOOST jika kondisi pertama tidak memenuhi sehingga:

- *Best case*: kondisi 1 memenuhi, maka efisiensi  $O(1)$

- *Worst case*: kondisi 1 tidak memenuhi, sehingga diperlukan pemeriksaan keberadaan *power up* BOOST maka  $O(p)$  dengan  $p$  adalah letak elemen BOOST pada array of Power Up.

#### 3.3.1.2. Strategi Pemrioritasan Penghindaran *Obstacle*

Untuk strategi pemrioritasan penghindaran *obstacle*, mobil akan memeriksakan keberadaan obstacle di jalur, keberadaan *power up* LIZARD, dan menghitung total *damage* jika perlu sehingga efisiensi strategi ini adalah  $O(l + p)$  dengan  $l$  adalah letak obstacle pada array of Lane dan  $p$  adalah letak elemen LIZARD pada array of Power Up.

#### 3.3.1.3. Strategi Pemrioritasan Pemaksimalan Kecepatan

Untuk strategi pemrioritasan pemaksimalan kecepatan, mobil akan memeriksakan keberadaan obstacle di jalur, keberadaan *power up* LIZARD, dan *current speed* sehingga efisiensi strategi ini adalah  $O(l + p)$  dengan  $l$  letak obstacle pada array of Lane dan  $p$  letak elemen BOOST pada array of Power Up.

#### 3.3.1.4. Strategi Pemrioritasan Pencarian *Power-Ups*

Untuk strategi pemrioritasan pencarian *power-ups*, mobil akan memeriksakan keberadaan *power-up* di jalur mobil sehingga efisiensi strategi ini adalah  $O(l)$  dengan  $l$  adalah letak obstacle pada array of Lane.

#### 3.3.1.5. Strategi Pemrioritasan Pemanfaatan *Power-Ups*

Untuk strategi pemrioritasan pencarian *power-ups*, mobil akan memeriksakan keberadaan *power-up* di dalam array of Power Up dan posisi mobil musuh sehingga efisiensi strategi ini adalah  $O(p)$  dengan  $p$  adalah letak *power up* yang dibutuhkan pada array of Power Up.

### 3.3.2. Analisis Efektivitas

#### 3.3.2.1. Strategi Pemrioritasan Perbaikan

Strategi pemrioritasan perbaikan hanya efektif ketika mobil menerima *damage* lebih besar dari 2 atau menerima *damage* berapapun jika memperoleh *power up* BOOST karena *command* FIX dapat menghilangkan *damage* sebanyak 2 dari mobil. Selain kondisi tersebut, *command* FIX akan menjadi tidak efektif karena mobil akan berada dalam kondisi *stationary/diam* di tempat.

#### 3.3.2.2. Strategi Pemrioritasan Penghindaran *Obstacle*

Strategi pemrioritasan penghindaran *obstacle* efektif ketika mobil mendapati adanya *obstacle* di hadapannya. Mengingat bahwa ketika mobil menabrak *obstacle*, mobil akan mengalami perlambatan dan menerima *damage* yang dapat menurunkan kecepatan maksimum. Selain kondisi tersebut, *command* yang digunakan di dalam strategi ini, yakni TURN\_LEFT, TURN\_RIGHT, atau USE\_LIZARD akan menjadi tidak efektif karena *power up* akan menghilang secara sia-sia atau pembelokan mobil akan merugikan mobil karena berbelok akan menurunkan kecepatan mobil.

#### 3.3.2.3. Strategi Pemrioritasan Pemaksimalan Kecepatan

Strategi pemrioritasan pemaksimalan kecepatan efektif ketika mobil belum mencapai kecepatan maksimum dan tidak menerima *damage* dari *obstacle*. Selain kondisi itu, strategi ini akan menjadi tidak efektif terutama apabila mobil telah memperoleh *damage* sebesar 5, *command* ACCELERATE dan USE\_BOOST akan menjadi sia-sia karena mobil akan tetap diam.

#### 3.3.2.4. Strategi Pemrioritasan Pencarian *Power-Ups*

Strategi pemrioritasan pencarian *power-ups* efektif ketika mobil menemukan adanya *power-up* di salah satu jalur yang dapat dilalui mobil dan di jalur tersebut tidak ada *obstacle*. Selain kondisi itu, strategi ini akan menjadi tidak efektif apabila tidak ada *power-up* yang tersedia atau mobil

mempertaruhkan penerimaan *damage* demi mendapatkan *power-up* tersebut.

### 3.3.2.5. Strategi Pemrioritasan Pemanfaatan *Power-Ups*

Strategi pemrioritasan pemanfaatan *power-ups* efektif ketika mobil memiliki *power-up* dan di jalur yang dilalui mobil tidak ada *obstacle*. Selain kondisi itu, strategi ini akan menjadi tidak efektif apabila ketika mobil mempertaruhkan penerimaan *damage* demi menggunakan *power-up* yang bisa jadi tidak berhasil mengenai mobil lawan.

## 3.4. Pemilihan Algoritma *Greedy*

Berdasarkan eksplorasi alternatif Algoritma *Greedy* di atas, kelompok kami memutuskan untuk menggabungkan strategi-strategi tersebut. Hal yang menjadi pertimbangan utama atas keputusan itu adalah fakta bahwa setiap strategi di atas hanya efektif dan dapat digunakan dalam beberapa kasus saja untuk setiap rondenya. Dengan demikian, penggabungan strategi-strategi tersebut dinilai lebih mampu secara efektif untuk menangani seluruh kasus atau skema yang ada selama permainan ketimbang memilih salah satu yang terbaik di antaranya.

Pada implementasi program, akhirnya kami menggabungkan dan memodifikasi kelima strategi tersebut. Untuk menangani kasus-kasus tertentu, kami berpikir bahwa diperlukan adanya pengaturan urutan prioritas strategi yang kemudian coba kami kombinasikan untuk menemukan mana kombinasi urutan yang paling optimal. Hasil yang kelompok dapatkan ialah kombinasi strategi dengan urutan sebagai berikut: strategi pemrioritasan perbaikan, strategi pemrioritasan penghindaran *obstacle*, strategi pemrioritasan pemaksimalan kecepatan, strategi pemrioritasan pencarian *power-ups*, dan yang terakhir strategi pemrioritasan pemanfaatan *power-ups*. Di samping itu, apabila berdasarkan *command* yang digunakan, urutan prioritasnya, yaitu perintah SAVETHECAR, AVOID, TURBO, MAXSPEED, GETPOWERUP, dan USEOIL/USEEMP/USETWEET.

Urutan prioritas yang kami tentukan terakhir kali berasal dari proses *trial and error* yang kami ujikan pada *reference bot*. Hasilnya, prioritas yang disusun demikian menghasilkan *win rate* tertinggi dibandingkan dengan variasi-variasi lainnya. Berdasarkan pengujian yang telah kami lakukan juga, terdapat kondisi-kondisi tertentu yang tidak dapat dihindari dalam mempengaruhi hasil akhir dari pertandingan mobil kami di permainan *Overdrive* ini, misalnya, bentuk atau kondisi *maps* yang dipakai pada pertandingan tersebut.

Algoritma *Greedy* yang kami pilih untuk diimplementasikan adalah berdasarkan fokus/*goal* utama dari kelompok untuk memenangkan pertandingan, yakni untuk mencapai garis finish lebih dulu daripada lawan. Kemudian, pertimbangan kedua ialah dengan tujuan memaksimalkan poin/skor yang bisa didapat setiap ronde melalui susunan prioritas perintah yang diberikan kepada bot mobil. Kami juga memaksimalkan skor yang didapat dari strategi penghindaran *obstacle* dengan memaksimalkan pemanfaatan *command* AVOID yang dimiliki oleh bot mobil kami.

Selain itu, algoritma *Greedy* kami juga didesain untuk menghindari munculnya respon *do nothing* ataupun *invalid command* yang bisa muncul akibat beberapa hal atau kondisi. Oleh karena itu, ketimbang menjalankan command DO NOTHING, kelompok kami lebih memilih untuk selalu me-return command ACCELERATE apabila terdapat kondisi yang memaksa mobil untuk tidak bisa menggunakan strategi lainnya. Strategi yang demikian dibuat dengan memprioritaskan pergerakan sebagai strategi *default* dari bot mobil kami. Kami menjadikan pergerakan sebagai strategi yang fundamental (selain perbaikan) dalam mengumpulkan poin sebanyak-banyaknya karena strategi ini secara rata-rata memiliki *case* yang paling optimal untuk dijadikan sebagai strategi *default*.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1. Implementasi Algoritma *Greedy*

Pada program kami, hasil implementasi algoritma *greedy* terdapat pada file bot.java. Di dalam file tersebut terdapat sebuah fungsi pemanggilan yang utama, yakni run.

##### 4.1.1. Public Command run

```
function run() → Command
{Fungsi utama untuk mencapai garis finish secepat mungkin berdasarkan
strategi yang telah dipersiapkan dan diurutkan berdasarkan skala prioritas}

KAMUS LOKAL
    blocks, leftBlocks, rightBlocks: List<Object>
    countLeft, countRight, countFront: integer

ALGORITMA
    blocks ← getBlocksInFront(myCar.position.lane,
        myCar.position.block, getSpeedAfterAccelerate())
    countFront ← getDamageInFront(myCar.position.lane,
        myCar.position.block, getSpeedAfterAccelerate())

    {Prioritas 1: Strategi Save The Car}
    {Strategi ini berfokus pada pemanggilan command FIX dan digunakan
    dalam 2 kondisi:
    1. menerima damage ≥ 2 dan kecepatan mobil = kecepatan maksimum di
    damage tersebut, atau
    2. menerima damage > 0 dan mendapatkan power up boost}

    if (((myCar.damage = 2 and myCar.speed = 8) or
        (myCar.damage = 3 and myCar.speed = 6) or
        (myCar.damage = 4 and myCar.speed = 3) or
        (myCar.damage >= 5)) and
        (not hasPowerUp(PowerUps.BOOST, myCar.powerups))) then
        → FIX
    else if (myCar.damage > 0 and hasPowerUp(PowerUps.BOOST,
        myCar.powerups)) then
        → FIX

    {Prioritas 2: Strategi Avoid}
    {Strategi ini berfokus pada penghindaran obstacle atau minimalisasi
    damage yang didapat ketika berhadapan dengan beberapa obstacle dan
    digunakan dalam kondisi ketika terdapat obstacle di jalur mobil}
    if (myCar.speed ≠ 0 && isBlocks(blocks)) then

        {Kasus 1: mobil memiliki LIZARD
        - ketika mobil berhadapan dengan obstacle tetapi salah satu
        jalur di sebelahnya aman, mobil akan berbelok ke jalur
        tersebut
        - ketika mobil berhadapan dengan obstacle di hadapannya dan di
```

```

jalur sebelahnya juga ada, mobil akan memanggil command
LIZARD
- ketika mobil berhadapan dengan obstacle di hadapannya dan
jalur di kiri dan kanannya aman, mobil akan memilih jalur
yang lebih beneficial untuknya (bersambung ke strategi di
bawah)}

if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)) then
    {Kasus 1.1: jika mobil berada di lane 1}
    if (myCar.position.lane = 1) then
        rightBlocks ←
        getBlocksInFront(myCar.position.lane + 1,
        myCar.position.block, myCar.speed - 1)
        if (isBlocks(rightBlocks)) then
            → LIZARD
        else
            → TURN_RIGHT

    {Kasus 1.2: jika mobil berada di lane 4}
    else if (myCar.position.lane = 4) then
        leftBlocks ←
        getBlocksInFront(myCar.position.lane - 1,
        myCar.position.block, myCar.speed - 1)
        if (isBlocks(leftBlocks)) then
            → LIZARD
        else
            → TURN_LEFT

    {Kasus 1.3: jika mobil berada di lane 2/3}
    else
        leftBlocks ←
        getBlocksInFront(myCar.position.lane - 1,
        myCar.position.block, myCar.speed - 1)
        rightBlocks ←
        getBlocksInFront(myCar.position.lane + 1,
        myCar.position.block, myCar.speed - 1)
        if (isBlocks(leftBlocks)) then
            if (isBlocks(rightBlocks)) then
                → LIZARD
            else
                → TURN_RIGHT
        else
            if (isBlocks(rightBlocks)) then
                → TURN_LEFT

    {Kasus 2: mobil tidak memiliki LIZARD
    - ketika mobil berhadapan dengan obstacle tetapi salah satu
    jalur di sebelahnya aman, mobil akan berbelok ke jalur
    tersebut
    - ketika mobil berhadapan dengan obstacle di hadapannya dan di
    jalur sebelahnya juga ada, mobil akan memilih jalur dengan
    jumlah damage paling rendah
    - ketika mobil berhadapan dengan obstacle di hadapannya dan
    jalur di kiri dan kanannya aman, mobil akan memilih jalur
    yang lebih beneficial untuknya (bersambung ke strategi di
    bawah)}

else
    {Kasus 2.1: jika mobil berada di lane 1}
    if (myCar.position.lane = 1) then

```

```

rightBlocks ←
getBlocksInFront(myCar.position.lane + 1,
myCar.position.block, myCar.speed - 1)
if (not isBlocks(rightBlocks)) then
    → TURN_RIGHT
else
    countRight ←
    getDamageInFront(myCar.position.lane + 1,
    myCar.position.block, myCar.speed - 1)
    if (countRight < countFront) then
        → TURN_RIGHT

{Kasus 2.2: jika mobil berada di lane 4}
else if (myCar.position.lane = 4) then
    leftBlocks ←
    getBlocksInFront(myCar.position.lane - 1,
    myCar.position.block, myCar.speed - 1)
    if (not isBlocks(leftBlocks)) then
        → TURN_LEFT
else
    countLeft ←
    getDamageInFront(myCar.position.lane - 1,
    myCar.position.block, myCar.speed - 1)
    if (countLeft < countFront) then
        → TURN_LEFT
else
    leftBlocks ←
    getBlocksInFront(myCar.position.lane - 1,
    myCar.position.block,
    myCar.speed - 1)
    rightBlocks ←
    getBlocksInFront(myCar.position.lane + 1,
    myCar.position.block,
    myCar.speed - 1)
    if (isBlocks(leftBlocks)) then
        if (not
            isBlocks(rightBlocks))
        then
            → TURN_RIGHT

{Kasus 2.3: jika mobil berada di lane 2/3}
else
    countRight ←
    getDamageInFront(myCar.position.lane + 1,
    myCar.position.block, myCar.speed - 1)
    countLeft ←
    getDamageInFront(myCar.position.lane - 1,
    myCar.position.block, myCar.speed - 1)
    if (countLeft ≤ countRight and countLeft <
        countFront) then
        → TURN_LEFT
    else if (countRight < countLeft and countRight <
        countFront) then
        → TURN_RIGHT
    else
        if (isBlocks(rightBlocks)) then
            → TURN_LEFT

{Prioritas 3: Strategi Turbo}
{Strategi ini berfokus pada pemanggilan command USE_BOOST dan}

```

```

digunakan ketika mobil tidak memiliki damage dan memiliki BOOST}
if (myCar.damage = 0 and hasPowerUp(PowerUps.BOOST,
    myCar.powerups) and myCar.speed ≠ 15 and not isBlocks(blocks))
then
    → BOOST

{Prioritas 4: Strategi Max Speed}
{Strategi ini berfokus pada pemanggilan command ACCELERATE dan
digunakan ketika kecepatan mobil belum mencapai kecepatan maksimum
pada damage terkait dan tidak terdapat obstacle di jalur mobil}
if (((myCar.speed < 9 and myCar.damage < 2) or
    (myCar.speed < 8 and myCar.damage = 2) or
    (myCar.speed < 6 and myCar.damage = 3) or
    (myCar.speed < 3 and myCar.damage = 4)) and
not isBlocks(blocks)) then
    → ACCELERATE

{Prioritas 5: Strategi Get Power Up}
{Strategi ini berfokus pada pengambilan power up BOOST/LIZARD dan
digunakan ketika:
- sudah mencapai kecepatan maksimal, tidak memiliki boost, dan tidak
ada obstacle di hadapannya, atau
- terdapat obstacle di hadapannya, tetapi kedua jalur di sebelahnya
tidak ada}

if (myCar.position.lane = 1) then
    rightBlocks ← getBlocksInFront(myCar.position.lane + 1,
        myCar.position.block, myCar.speed - 1)
    if (not isPowerUp(blocks)) then
        if (isPowerUp(rightBlocks) and not
            isBlocks(rightBlocks)) then
                → TURN_RIGHT
    else if (myCar.position.lane = 4) then
        leftBlocks ← getBlocksInFront(myCar.position.lane - 1,
            myCar.position.block, myCar.speed - 1)
        if (!isPowerUp(blocks)) then
            if (isPowerUp(leftBlocks) and not
                isBlocks(leftBlocks)) then
                    → TURN_LEFT
    else
        leftBlocks ← getBlocksInFront(myCar.position.lane - 1,
            myCar.position.block, myCar.speed - 1)
        rightBlocks ← getBlocksInFront(myCar.position.lane + 1,
            myCar.position.block, myCar.speed - 1)
        if (isBlocks(blocks)) then
            if (not isPowerUp(rightBlocks)) then
                if (isPowerUp(leftBlocks)) then
                    → TURN_LEFT
            else
                if (myCar.position.lane = 2) then
                    → TURN_RIGHT
                else
                    → TURN_LEFT
        else
            if (isPowerUp(leftBlocks)) {
                if (myCar.position.lane = 2) then
                    → TURN_RIGHT
                else
                    → TURN_LEFT

```

```

        else
            → TURN_RIGHT
    else
        if (not isPowerUp(blocks)) then
            if (not isPowerUp(rightBlocks)) then
                if (isPowerUp(leftBlocks) and not
                    isBlocks(leftBlocks)) then
                    → TURN_LEFT
            else
                if (not
                    isBlocks(rightBlocks))
                then
                    → TURN_RIGHT

{Prioritas 6: Strategi USE_TWEET, USE_OIL, USE_EMP}
{Strategi ini digunakan ketika prioritas 1-5 sudah tidak memenuhi dan
mobil memenuhi kondisi:
- USE_TWEET: ketika mobil memiliki power up TWEET
- USE_OIL: ketika mobil berada di depan mobil musuh dan
memiliki power up OIL
- USE_EMP: ketika mobil berada di depan mobil musuh dan
memiliki power up OIL}

if (hasPowerUp(PowerUps.TWEET, myCar.powerups)) then
    TWEET ← new TweetCommand(opponent.position.lane,
    opponent.position.block + getSpeedAfterAccelerate(opponent) +
    1)
    → TWEET

if (myCar.position.block > opponent.position.block + opponent.speed)
then
    if (hasPowerUp(PowerUps.OIL, myCar.powerups))
        → OIL
    else if (myCar.position.block < opponent.position.block) then
        if (myCar.position.lane = opponent.position.lane and
            hasPowerUp(PowerUps.EMP, myCar.powerups)) then
                → EMP

{Jika seluruh prioritas tidak memenuhi, mobil kami akan menjalankan
ACCELERATE}
→ ACCELERATE

```

#### 4.1.2. Private Boolean hasPowerUp

```

function hasPowerUp(powerUpToCheck: PowerUps,
                     available: array of PowerUps) → Boolean
{mengembalikan True jika mobil memiliki power up 'PowerUpToCheck' atau False
jika mobil tidak memilikinya}

KAMUS LOKAL
powerUp: PowerUps

ALGORITMA
for (powerUp: available) do
    if (powerUp = NULL) then
        → false
    if (powerUp = powerUpToCheck) then
        → true

```

→ false

#### 4.1.3. Private int getSpeedAfterAccelerate

```
function getSpeedAfterAccelerate() → integer
{mengembalikan kecepatan mobil setelah melakukan akselerasi/memanggil command
ACCELERATE}

KAMUS LOKAL
-
ALGORITMA
    if (myCar.speed = 3 or myCar.speed = 5) then
        → 6
    else if (myCar.speed = 6) then
        → 8

    else if (myCar.speed = 8 or
            (myCar.speed == 9 and not hasPowerUp(PowerUps.BOOST,
            myCar.powerups))) then
        → 9
    else if ((myCar.speed = 9 and hasPowerUp(PowerUps.BOOST,
            myCar.powerups)) or myCar.speed = 15) then
        → 15
    else
        → 3
```

#### 4.1.4. Private int getDamageInFront

```
function getDamageInFront(lane, block, speed: integer) → integer
{mengembalikan jumlah damage yang diberikan obstacle di jalur terkait}

KAMUS LOKAL
    map: List<array of Lane>
    sumDamage: integer
    startBlock: integer
    laneList: array of Lane
    i: integer

ALGORITMA
    map ← gameState.lanes
    sumDamage ← 0
    startBlock ← map.get(0).position.block
    laneList = map.get(lane - 1)

    i traversal [max(block - startBlock, 0)..(block - startBlock + 1)]
        if (laneListi = NULL or laneListi.terrain = Terrain.FINISH) then
            → sumDamage
        else
            if (laneListi.terrain = Terrain.MUD or
                laneListi.terrain = Terrain.OIL_SPILL) then
                    sumDamage ← sumDamage + 1
            else if (laneListi.terrain = Terrain.WALL) then
                sumDamage ← sumDamage + 2
    → sumDamage
```

#### 4.1.5. Private List<Object> getBlocksInFront

```
function getBlocksInFront(lane, block, speed: integer) → List<Object>
{mengembalikan list yang berisi block-block yang ada di jalur terkait}

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

ALGORITMA
    map ← gameState.lanes
    blocks ← new ArrayList<>()
    startBlock ← map.get(0).position.block
    laneList ← map.get(lane - 1)

    i traversal [max(block - startBlock, 0)..block - startBlock + speed]
        if (laneListi = null || laneListi.terrain = Terrain.FINISH) then
            → blocks
        else
            blocks.add(laneList[i].terrain)
            if (laneListi.isOccupiedByCyberTruck) then
                blocks.add("TRUCK")
            if (laneListi.occupiedById = opponent.id) then
                blocks.add("OPPONENT")
            → blocks
```

#### 4.1.6. Private Boolean isBlocks

```
function isBlocks(block: List<Object>) → Boolean
{mengembalikan True jika sebuah block merupakan obstacle atau False jika bukan}
KAMUS LOKAL
    -

ALGORITMA
    → (block.contains(Terrain.MUD) or block.contains(Terrain.WALL) or
       block.contains(Terrain.OIL_SPILL) or block.contains("TRUCK") or
       block.contains("OPPONENT"))
```

#### 4.1.7. Private Boolean isPowerUp

```
function isPowerUp(block: List<Object>) → Boolean
{mengembalikan True jika sebuah block berisi power up BOOST/LIZARD atau False jika bukan}
KAMUS LOKAL
    -

ALGORITMA
    → (block.contains(Terrain.BOOST) or block.contains(Terrain.LIZARD))
```

## 4.2. Struktur Data Program *Overdrive*

Pada permainan “Overdrive” ini, struktur data yang digunakan berbasis pada kelas. Di dalam permainan ini, telah terdefinisi beberapa kelas, antara lain *Command*, *Entities*, *Enums*, *Bot*, dan *Main*. Kelas-kelas tersebut kemudian kami kembangkan dan lengkapi dengan penjelasan sebagai berikut.

### 4.2.1. Command

Kelas ini berisi kumpulan aksi dan *command* yang dapat digunakan pemain untuk mengelola bot mobil.

#### 4.2.1.1. AccelerateCommand

Kelas ini berguna dalam pemanggilan *command* ‘ACCELERATE’.

#### 4.2.1.2. BoostCommand

Kelas ini berguna dalam pemanggilan *command* ‘USE\_BOOST’.

#### 4.2.1.3. ChangeLaneCommand

Kelas ini berguna dalam pemanggilan *command* ‘TURN\_LEFT’ atau ‘TURN\_RIGHT’.

#### 4.2.1.4. DecelerateCommand

Kelas ini berguna dalam pemanggilan *command* ‘DECELERATE’.

#### 4.2.1.5. DoNothingCommand

Kelas ini berguna dalam pemanggilan *command* ‘NOTHING’.

#### 4.2.1.6. EmpCommand

Kelas ini berguna dalam pemanggilan *command* ‘USE\_EMP’.

#### 4.2.1.7. FixCommand

Kelas ini berguna dalam pemanggilan *command* ‘FIX’.

#### 4.2.1.8. LizardCommand

Kelas ini berguna dalam pemanggilan *command* ‘USE\_LIZARD’.

#### 4.2.1.9. OilCommand

Kelas ini berguna dalam pemanggilan *command* ‘USE\_OIL’.

#### 4.2.1.10. TweetCommand

Kelas ini berguna dalam pemanggilan *command* ‘USE\_TWEET’.

#### 4.2.2. Entities

Kelas ini berisi kumpulan data *entity* yang tersedia pada di dalam permainan, seperti data yang dimiliki bot mobil dan data keadaan permainan.

##### 4.2.2.1. Car

Kelas ini berisi kumpulan data Car (bot mobil), antara lain *id*, *position*, *speed*, *state*, *damage*, *power up*, *boosting*, dan *boostCounter*.

##### 4.2.2.2. GameState

Kelas ini berisi kumpulan data keadaan permainan, antara lain *currentRound*, *maxRounds*, *player*, *opponent*, dan *worldMap*.

##### 4.2.2.3. Lane

Kelas ini berisi kumpulan data jalur, antara lain *position*, *surfaceObject*, *occupiedByPlayer*, dan *isOccupiedByCyberTruck*.

##### 4.2.2.4. Position

Kelas ini berisi kumpulan data posisi, antara lain *x* dan *y*.

#### 4.2.3. Enums

Kelas ini berisi kumpulan data khusus yang tersedia pada di dalam permainan, seperti data arah dan tipe blok.

##### 4.2.3.1. Direction

Kelas ini berisi kumpulan data khusus arah pergerakan yang tersedia di dalam permainan.

##### 4.2.3.2. PowerUps

Kelas ini berisi kumpulan data khusus variasi *power up* yang tersedia di dalam permainan, antara lain BOOST, OIL, TWEET, LIZARD, dan EMP.

##### 4.2.3.3. State

Kelas ini berisi kumpulan data khusus keadaan suatu mobil selama permainan, seperti ACCELERATING, TURNING\_RIGHT, dan PICKED\_UP\_POWERUP.

##### 4.2.3.4. Terrain

Kelas ini berisi kumpulan data khusus variasi block jalan yang tersedia di dalam permainan.

#### 4.2.4. Bot

Kelas ini berisi hasil pengembangan dan implementasi algoritma *Greedy* yang akan digunakan di dalam permainan dengan tujuan untuk memenangkan permainan. Kelas ini memanfaatkan berbagai kelas-kelas yang telah disebutkan di atas.

#### 4.2.5. Main

Kelas ini berfungsi untuk memanggil kelas Bot dan menjalankan program dengan menjalankan seluruh algoritma dan *command-command* yang telah dikembangkan.

### 4.3. Analisis Pengujian Algoritma *Greedy*

Percobaan untuk mengambil data uji ini dilakukan cukup dengan menjalankan `run.bat` pada folder starter-pack (src jika sudah diganti) yang berisi bot yang telah dikembangkan. Pada kesempatan kali ini, kami melakukan tiga kali pengujian terhadap bot yang telah kami rancang.

#### 4.3.1. Pengujian I

```
[#      *      #  0]
[>    2     *    T ]
[      B      ]
[      >  ]=====
Received command C;124;ACCELERATE
Completed round: 124
*****
Game Complete
Checking if match is valid
=====
The winner is: A - Overbrain
```

Pada pengujian pertama, *bot* yang kami rancang berhasil mengalahkan *reference bot* yang disediakan oleh Entelect hanya dalam 124 ronde. Hasil akhir ini setelah dilihat pada visualizer dikarenakan *car* kami banyak mendapatkan power-ups *boost* di sepanjang jalurnya sehingga kecepatan maksimal pada *car* dapat tercapai.

#### 4.3.2. Pengujian II

```
[      2 ||]
[      ]
[      ]
[      ]
=====
Received command C;144;TURN_RIGHT
Completed round: 144
*****
Game Complete
Checking if match is valid
=====
The winner is: B - Coffee Ref
```

Namun, pada pengujian kedua, *bot* yang kami rancang mengalami kekalahan terhadap *reference bot* dalam 144 ronde. Dapat dilihat disini bahwa meskipun *car* kami menggunakan strategi pemaksimalan kecepatan, kecepatan *car* kami tetap dapat dihentikan dengan beragam power-ups pada permainan dan akhirnya kalah. Setelah dilihat pada visualizer, *car* kami tidak berhasil menghindari truck yang dikeluarkan oleh *car reference bot*. Kasus ini adalah kasus terburuk karena truck pada permainan *overdrive* memiliki *damage* yang cukup besar dan dapat mengurangi kecepatan pada *car*.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### 5.1. Kesimpulan

Telah berhasil diimplementasikan sebuah program berupa bot mobil pada permainan *Overdrive* yang dirancang dan dikembangkan untuk memenangkan permainan sesuai dengan yang diminta dalam spesifikasi Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022. Hal mengenai bot mobil yang berhasil diimplementasikan dalam program ini meliputi:

1. Konsep algoritma Greedy dan penerapannya dalam implementasi bot mobil pada permainan *Overdrive*,
2. Eksplorasi alternatif strategi yang berkaitan dengan permasalahan optimisasi, baik itu *maximization* maupun *minimization*,
3. Analisis efisiensi dan efektivitas strategi *Greedy* yang berhasil dieksplorasi dan ditemukan untuk kemudian ditentukan mana yang layak dipakai pada suatu kondisi tertentu,
4. Pemetaan dan pengurutan prioritas strategi berdasarkan kondisi mobil dan lingkungan di sekitar mobil,
5. Implementasi program dengan paradigma pemrograman berorientasi objek untuk mengembangkan bot mobil pada permainan *Overdrive*.

Semua implementasi dari konsep-konsep di atas kemudian berhasil digunakan untuk menyelesaikan seluruh *command* yang ada di dalam spesifikasi maupun *game rules* dari permainan *Overdrive* itu sendiri. Setidaknya terdapat 9 *command* utama (pada spesifikasi permainan terdapat 11 *command*, namun yang kami pilih untuk kami gunakan hanya 9) yang dapat digunakan pada bot mobil permainan *Overdrive* kami. *Command-command* ini dapat digunakan dan telah diatur pada bot mobil permainan *Overdrive* kami yang sudah diimplementasikan sedemikian rupa dengan memanfaatkan strategi *Greedy* yang telah kelompok kami eksplorasi. Dengan demikian, strategi yang kami gunakan diharapkan dapat memampukan *player* untuk memenangkan pertandingan.

Dengan pengimplementasian berbagai strategi pemrioritasan pada bot mobil permainan *Overdrive*, khususnya dengan menggunakan konsep algoritma *Greedy*, kita dapat memecahkan permasalahan optimisasi yang dapat ditemukan ketika melakukan eksplorasi alternatif strategi untuk memenangkan pertandingan. Konsep yang telah diajarkan di perkuliahan IF2211 dapat dengan baik diterapkan dalam penggerjaan Tugas Besar 1 IF2211 Strategi Algoritma ini. Selain melakukan eksplorasi terhadap berbagai alternatif algoritma Greedy yang dapat dipakai, kelompok juga mengatur skala prioritas terhadap setiap strategi yang telah ditemukan. Bahkan, ada pula *command-command*

yang tersedia pada spesifikasi, namun tidak kami implementasikan mengingat *command* tersebut tidak memberikan *impact* yang signifikan terhadap kemenangan bot mobil pada permainan *Overdrive* ini. Hal ini tentu kami putuskan setelah melakukan analisis efisiensi dan efektivitas suatu *command* maupun strategi yang ada.

Dengan demikian, kelompok menyimpulkan bahwa dengan mengerjakan Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 ini, dapat diketahui bahwa untuk menyelesaikan suatu masalah yang mungkin ditemukan dalam kehidupan sehari-hari, dalam hal ini misalnya, memenangkan pertandingan balap mobil pada permainan *Overdrive*, dapat diimplementasikan strategi pada bot mobil dengan mencari solusi paling optimal dari suatu permasalahan yang membutuhkan optimisasi sebagai bentuk penerapan dari konsep algoritma *Greedy* yang telah dipelajari pada kuliah IF2211.

## 5.2. Saran

Tugas Besar 1 IF2211 Strategi Algoritma Semester 2 Tahun 2021/2022 menjadi salah satu proses pembelajaran bagi kelompok dalam menerapkan ilmu-ilmu yang diajarkan pada kuliah maupun melakukan eksplorasi materi secara mandiri. Berikut ini merupakan sejumlah saran dari kelompok untuk pihak-pihak yang ingin melakukan atau mengerjakan hal serupa.

1. Program yang diminta adalah program dengan menggunakan bahasa pemrograman Java, yakni salah satu bahasa pemrograman yang belum dikuasai secara menyeluruh oleh ketiga anggota kelompok yang terlibat dalam penggeraan tugas besar ini. Dengan demikian, kelompok merekomendasikan agar disediakan waktu yang cukup untuk melakukan eksplorasi terkait bahasa pemrograman yang digunakan sebelum mengimplementasikannya ke dalam sebuah program. Hal ini akan meningkatkan efektivitas kerja tim dalam pembuatan suatu program. Di samping itu, perlu dipertimbangkan pula waktu yang dimiliki untuk melakukan eksplorasi terhadap suatu bahasa pemrograman atau *framework* tertentu sehingga tidak membebani *programmer* dalam penggeraan proyek dengan jangka waktu yang singkat. Gunakan kemampuan berpikir serta bekerja yang elaboratif dan koordinatif di antara seluruh anggota kelompok yang terlibat.
2. Modularitas menjadi hal yang krusial dalam menciptakan suatu program secara efektif dan efisien. Dalam jangka waktu yang singkat, pemrograman secara modular dapat membantu *programmer* untuk memudahkan proses pencarian kesalahan/*error* serta *debugging*. Pada dasarnya, memrogram secara modular berarti memecah-mecah program menjadi modul-modul kecil di mana masing-masing modul berinteraksi melalui antarmuka modul. Masalah yang awalnya kompleks dapat dibagi menjadi bagian-bagian kecil yang lebih sederhana dan dapat diselesaikan dalam lingkup yang lebih kecil. Akibatnya, apabila terdapat *error/bug* pada program, kesalahan dapat dengan mudah ditemukan

karena alur logika yang jelas serta dapat dilokalisasi dalam satu modul. Lebih dari pada itu, modifikasi program dapat dilakukan tanpa mengganggu *body* program secara keseluruhan. Oleh karena itu, kelompok sangat menyarankan untuk melakukan pemrograman secara modular dalam mengimplementasikan fungsi-fungsi pada bot mobil dalam permainan *Overdrive*.

3. Penting bagi kelompok untuk memiliki strategi serta distribusi tugas yang baik. Ketika membuat program dalam sebuah tim, kesamaan cara menulis kode serta kemampuan untuk menulis komentar menjadi hal yang sangat penting. Hal ini diperlukan agar memudahkan anggota kelompok dalam menyatukan dan melanjutkan sebuah program. Kemampuan tersebut tentunya didukung juga dengan adanya *version control system* yang baik yang dapat digunakan oleh *programmer* dalam membuat sebuah program secara bersama-sama. Untuk itu, kami sangat menyarankan ‘GitHub’ untuk digunakan sebagai *version control system* dalam penggerjaan tugas-tugas besar pada mata kuliah IF2211 ini, maupun pada pembuatan program dan penggerjaan proyek yang lainnya.
4. Kelompok menyadari bahwa pada implementasi bot mobil pada permainan *Overdrive* yang telah kami buat, masih banyak aspek yang dapat dikembangkan lebih lagi. Salah satunya ialah dengan mengoptimalkan algoritma *Greedy* yang digunakan agar mobil pada permainan *Overdrive* kami ini dapat memenangkan pertandingan bersama dengan mobil lawan. Hal ini tentu menjadi ruang untuk *programmer* agar dapat melakukan improvisasi terhadap implementasi dan pengembangan program pada bot mobil permainan *Overdrive*, terutama dalam hal eksplorasi strategi. Selain itu, kelompok juga merekomendasikan untuk menjalankan permainan ini dengan *visualizer* yang tersedia untuk mempermudah pemahaman *programmer* terhadap permainan *Overdrive* itu sendiri sebelum kembali mengembangkan strategi dan implementasi bot yang sudah ada.

## **LAMPIRAN**

Link *repository* GitHub:

[https://github.com/eiffelaqila/Tubes1\\_Overbrain](https://github.com/eiffelaqila/Tubes1_Overbrain)

Link *video* YouTube:

<https://youtu.be/GgN5ofoHDuU>

## **DAFTAR PUSTAKA**

- Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 1). Institut Teknologi Bandung.  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf). Diakses pada 12 Februari 2022.
- Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 2). Institut Teknologi Bandung.  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf). Diakses pada 12 Februari 2022.
- Munir, Rinaldi. (2020). Algoritma Greedy (Bagian 3). Institut Teknologi Bandung.  
[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf). Diakses pada 12 Februari 2022.