

LAPORAN TUGAS BESAR 1 PEMANFAATAN ALGORITMA *GREEDY* DALAM GAME OVERDRIVE

IF2211/Strategi Algoritma



Hotwheels

1. Fadil Fauzani - 13520032
2. Jason Kanggara - 13520080
3. Nayotama Pradipta - 13520089

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

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.

Pada tugas besar pertama Strategi Algoritma ini, sebuah game engine digunakan untuk 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
- 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 adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) sedemikian sehingga, pada setiap langkah mengambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Algoritma Greedy merupakan metode paling populer dan sederhana untuk memecahkan persoalan optimasi (persoalan yang mencari solusi optimal), yang dapat dibagi menjadi dua macam persoalan, yaitu Maksimasi dan Minimasi.

2.1.1 Elemen Algoritma Greedy

Elemen - elemen yang digunakan dalam penerapan *Algoritma Greedy* antara lain:

1. Himpunan kandidat, adalah himpunan yang berisi elemen pembentuk solusi.
2. Himpunan solusi, adalah himpunan yang terpilih sebagai solusi persoalan.
3. Fungsi solusi, menentukan apakah himpunan kandidat yang sudah dipilih memberikan solusi
4. Fungsi seleksi, memilih kandidat berdasarkan strategi greedy tertentu yang paling mungkin mencapai solusi optimal.
5. Fungsi kelayakan, memeriksa apakah suatu kandidat yang dipilih itu layak atau tidak (jika layak, maka kandidat dapat dimasukkan ke dalam himpunan solusi).
6. Fungsi Objektif, adalah fungsi yang mengoptimalkan suatu solusi, baik secara maksimasi maupun minimasi.

2.2 Game Engine Entelect Challenge - Overworks

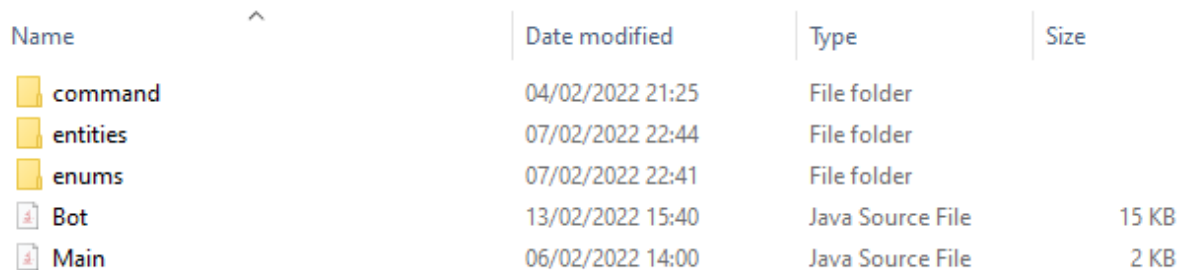
Dalam *game* Entelect Challenge 2020 - Overworks, terdapat beberapa komponen yang diperlukan agar *game* dapat berjalan, yaitu:

1. *Game engine* : bertanggung jawab untuk melaksanakan segala peraturan *game* yang ada, dengan memberikan *command* pada bot dalam *game state* jika *command* valid.
2. *Game runner* : bertanggung jawab untuk menjalankan permainan antar *player*, menerima *commands* yang diberikan oleh bot, lalu diberikan kepada *game engine* untuk dijalankan *command* yang telah diterima.
3. *Reference bot* : Bot referensi yang sudah disediakan oleh pembuat game sebagai referensi lawan untuk bot yang akan dibuat oleh *player*. Tersedia dalam bahasa Java.
4. *Starter bot* : Bot awal dengan beberapa logika dasar yang digunakan sebagai referensi dalam pengembangan bot.

2.2.1 Starter Bot dalam Bahasa Java

Starter Bot merupakan folder yang berisi file - file yang akan menjadi dasar dalam pembangunan bot oleh pemain. Folder *Starter Bot* beserta folder *Reference Bot* dapat diperoleh dengan mengunduh file zip yang tersedia pada starter pack di (<https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4>). Dalam folder *Starter Bot* terdapat berbagai macam pilihan bahasa pemrograman yang dapat digunakan. Pada tugas besar ini, bahasa yang akan digunakan selama proses pengembangan bot adalah bahasa pemrograman Java. Pada folder *java*, terdapat tiga folder dan dua file yang menjadi *base* dari bot yang akan dikembangkan, yaitu:

1. *Folder command* : berisi kelas dan perintah yang dapat digunakan oleh pemain.
2. *Folder entities* : berisi kelas entitas yang ada pada game
3. *Folder enums* : berisi entitas yang memiliki nilai berupa konstanta
4. *Bot.java* : *File* ini berisikan logika dari bot yang akan dibangun. Pada file ini akan diimplementasikan strategi greedy dengan memanfaatkan kelas - kelas yang ada. Pada file *Bot.java*, *method* utama yang kemudian akan dijalankan adalah *method run()*.
5. *Main.java* : *File* ini bertanggung jawab untuk menjalankan bot dengan menerima perintah setiap ronde dari file *Bot.java* yang akan dikirimkan ke *game engine* untuk kemudian diproses.



| Name | Date modified | Type | Size |
|----------|------------------|------------------|-------|
| command | 04/02/2022 21:25 | File folder | |
| entities | 07/02/2022 22:44 | File folder | |
| enums | 07/02/2022 22:41 | File folder | |
| Bot | 13/02/2022 15:40 | Java Source File | 15 KB |
| Main | 06/02/2022 14:00 | Java Source File | 2 KB |

Gambar xx. Struktur folder java

Untuk dapat menjalankan bot nya, diperlukan beberapa *tools* yang perlu di-*install* terlebih dahulu, yaitu Java JDK (min versi 8), IntelliJ IDEA, dan NodeJS. Untuk mendapatkan *executable file* java (.jar), *file* yang disediakan harus di-*build* terlebih dahulu menggunakan IntelliJ IDEA. Setelah itu, data lokasi “*player-a*” diubah ke *path* “*./starter-bots/java*” pada file *game-runner-config.json*. Jika sudah melakukan konfigurasi dan build file .jar, maka *game overdrive* dapat dijalankan dengan menjalankan file *run.bat* di *root directory*. Berikut adalah penjelasan mengenai file konfigurasi yang ada di permainan *overdrive*:

1. *game-runner-config.json*: Terletak di *root folder starter-pack*, file ini digunakan untuk mengatur lokasi file bot pemain satu, lokasi file bot pemain dua, lokasi keluaran dari hasil *match*, lokasi *game engine*, lokasi *game config*, dan konfigurasi lain yang berhubungan dengan keberjalanan dari *game*.
2. *game-config.json*: Terletak di *root folder starter-pack*, file ini digunakan untuk melakukan konfigurasi *game*-nya itu sendiri, seperti mengatur panjang track, mengatur

banyaknya ronde, entitas pada peta game, *command* - *command* yang terdapat pada *game*, *state* yang ada pada *game*, persentase dari *terrain generation*, dan masih banyak lagi terkait.

3. bot.json: Terletak di folder pengembangan bot, dalam contoh tugas besar ini, file bot.json terletak di folder starter-bots/java. File ini bertujuan untuk memberikan identitas pada bot seperti nama pencipta bot, email pencipta bot, nama bot, lokasi file executable dari bot dan bahasa pemrograman dari bot.

Hasil eksekusi dari permainan *Overdrive* ini memiliki tampilan *command line* secara default. Akan tetapi, ada metode untuk melihat tampilan dari hasil permainan agar terlihat lebih baik dan lebih interaktif, yaitu dengan menggunakan *visualizer*. Untuk menggunakan perangkat *visualizer*, unduh *visualizer* di (<https://github.com/Affuta/overdrive-round-runner>). Cara menggunakan *visualizer* terdapat di *repository* pada link yang sudah tertera sebelumnya.

2.2.2 Mengembangkan Starter Bot

Walaupun starter bot sudah menyediakan logic yang dibutuhkan untuk menjalankan bot, logic yang disediakan masih sangat sederhana. *Class* dan *Method* yang tersedia masih belum menggunakan seluruh fitur yang tersedia, dan *method* yang tersedia hanya berisi *command* - *command* sederhana seperti *command* belok kanan, belok kiri, dan akselerasi. Akan tetapi, sudah disediakan *method* untuk mendapatkan *block* - *block* apa saja yang terdapat di depan *bot* untuk melakukan fungsi sederhana seperti menghindari *block* lumpur. Pada folder inti dari Starter Bot juga sudah terdapat file file yang berisi *command*, entitas, dan *enums*. Walaupun begitu, konten dari file tersebut masih ada yang belum lengkap, seperti pada folder *command*, masih terdapat *command* yang belum disediakan, yaitu *command* *Lizard* dan *command* *Tweet*. Untuk melengkapi file file yang dibutuhkan dalam pengembangan bot, dapat melihat di (<https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/state-files.md>)

Kedua *command* baru yang ditambahkan sebelumnya pada folder *command* dapat dituliskan dengan sintaks sebagai berikut,

| | |
|-------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| LizardCommand.java package za.co.entelect.challenge.command; public class LizardCommand implements Command { | TweetCommand.java package za.co.entelect.challenge.command; public class TweetCommand implements Command { private int lane; |
|-------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|

| | |
|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> @Override public String render() { return String.format("USE_LIZARD"); } </pre> | <pre> private int block; public TweetCommand (int lane, int block) { this.lane = lane; this.block = block; } @Override public String render() { return String.format("USE_TWEET %d %d", lane, block); } } </pre> |
|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Berdasarkan contoh di atas, terdapat *method render()* yang mengembalikan suatu *string*. *Method* tersebut akan mengembalikan *command* bertipe string, dimana *command* tersebut akan dikirimkan ke file *Main.java* yang nanti akan dikirimkan ke game engine untuk menjalankan *command tersebut* jika dikembalikan pada suatu ronde.

Jika sudah menambahkan *class* dan elemen baru yang diperlukan, langkah selanjutnya adalah mengimplementasikan strategi greedy pada file *Bot.java*. File *Bot.java* akan menjadi file inti yang akan mempengaruhi bagaimana suatu bot bekerja, seperti belok kiri, belok kanan, menggunakan *powerup*, dan sebagainya. *Method* utama pada file *Bot.java* adalah *method run()* yang akan mengembalikan *command* yang akan dijalankan oleh bot setiap ronde. *Method run()* tersebut akan dipanggil di file *Main.java* untuk kemudian dikirimkan ke game engine untuk memeriksa apakah valid atau tidak, jika valid maka *command* tersebut dijalankan, jika tidak maka bot tidak akan melakukan apa - apa. Dalam file *Bot.java*, player dapat menambah *method - method* pendukung yang dapat dimanfaatkan pada *method run()*. Isi dari file *Bot.java* secara mendasar adalah seperti berikut.

| |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> package za.co.entelect.challenge; import za.co.entelect.challenge.command.*; import za.co.entelect.challenge.entities.*; import za.co.entelect.challenge.enums.PowerUps; import za.co.entelect.challenge.enums.State; import za.co.entelect.challenge.enums.Terrain; // Import hal hal yang diperlukan public class Bot { </pre> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

// Atribut bawaan dan tambahan

public Bot(GameState gameState) {
    this.random = new SecureRandom();
    this.gameState = gameState;
    this.myCar = gameState.player;
    this.opponent = gameState.opponent;
    // Atribut yang dapat digunakan oleh seluruh method dalam
class Bot
    }

    public Class Something() {
        // method method tambahan/pendukung
    }

    public Command run() {
        // method utama untuk menjalankan command setiap ronde
        return Command;
    }
}

```

BAB III

Aplikasi Strategi Greedy

3.1 Solusi Strategi Greedy yang Mungkin Dipilih

3.1.1 Greedy by Obstacle

Greedy by Obstacle adalah strategi yang berfokus pada menghindari suatu *Terrain* yang memberikan damage ke mobil. Pada setiap ronde, periksalah apakah di depan mobil terdapat *Terrain* yang bersifat *obstacle* atau tidak. Tingkatan menghindari suatu *Terrain* berdasarkan besarnya *slowing* dari *terrain*, semakin besar *slowing* maka itu menjadi prioritas utama untuk dihindari. Jika ada *obstacle*, maka jalankan *command* untuk menghindari *Terrain*, jika tidak, maka jalankan *command* yang lain. Akan tetapi, jika memiliki powerup Lizard, maka saat tidak ditemukan lane kosong tetapi memiliki powerup Lizard, maka *command* lizard dijalankan untuk melompati obstacle

a. Mapping Elemen - Elemen Greedy

- Himpunan Kandidat: *Terrain* di depan mobil
- Himpunan Solusi: *Terrain* yang memperlambat speed
- Fungsi Solusi: Memeriksa apakah ada lane dalam jangkauan speed dari player yang memiliki *Terrain* dengan efek paling minimal, atau ada suatu lane dalam jangkauan yang tidak ada *terrain* buruk sama sekali.
- Fungsi Seleksi: Jika ada lane tanpa *terrain* buruk, maka pilih lane tersebut, jika setiap lane ada *terrain* buruk, pilihlah lane dengan *terrain* yang memiliki efek paling minimal. Jika tidak ada pilihan lane kosong, maka gunakan powerup lizard
- Fungsi Kelayakan: memeriksa apakah lane kosong.
- Fungsi Objektif: Meminimalkan efek memperlambat speed.

b. Analisis Efisiensi Solusi

Terdapat 2 / 3 lane sesuai dengan posisi bot sekarang. Untuk memeriksa obstacle yang ada, maka setiap rondanya, block yang terdapat di depan bot dimasukan ke dalam list of block. Panjang dari list of blocks sesuai dengan speed yang dimiliki oleh bot (N Kecepatan). Jika bot berada di lane 1 atau 4, maka pengecekan block sebanyak 2N. Jika bot berada di lane 2 atau 3, maka pengecekan block sebanyak 3N. Maka Efisiensi waktu adalah:

$$T(n) = 2N, \text{ untuk lane} = 1 \text{ atau lane} = 4,$$

$$T(n) = 3N, \text{ untuk } 1 < \text{lane} < 4$$

Kasus jika tidak terdapat lane kosong maka dilakukan pengecekan apakah terdapat powerup lizard atau tidak. Pengecekan dilakukan sebanyak n dalam list powerup

$$T(n) = 2N + N = 3N, \text{ untuk lane} = 1 \text{ atau lane} = 4,$$

$$T(n) = 3N + N = 4N, \text{ untuk } 1 < \text{lane} < 4,$$

Maka Big O Notationnya adalah $O(N)$

c. Analisis Efektivitas Solusi

Strategi ini Efektif apabila

- Terdapat lane kosong dalam jangkauan bot
- Terdapat lane dengan *obstacle* paling sedikit dalam jangkauan bot

Strategi ini kurang Efektif apabila

- Seluruh lane dalam jangkauan bot tertutup oleh *obstacle*

3.1.2 Greedy by PowerUp

Greedy by PowerUp adalah strategi greedy yang mengutamakan pada penggunaan *power-up* jika terdapat pada inventory (*list of powerups*) dari bot. Pada setiap ronde, periksa apakah di *inventory* tersedia suatu *powerup*. Jika bot memiliki *powerup*, maka *powerup* akan digunakan terlebih dahulu sebelum menjalankan *command* yang lain. Untuk meningkatkan efektivitas dari penggunaan *powerup*, maka *powerup* digunakan dengan syarat tertentu. Selain menggunakan *power-up* yang ada di *inventory*, *Greedy by Powerup* juga berusaha untuk mencari *powerup* yang ada pada lane dalam jangkauan dari bot.

a. Mapping Elemen - Elemen Greedy

- Himpunan Kandidat: *Powerup* yang ada, baik dalam *inventory* maupun tidak
- Himpunan Solusi: Menggunakan *powerup*
- Fungsi Solusi: Memeriksa apakah terdapat *power-up* yang bisa dipakai oleh player, jika ada maka gunakan *power-up* dengan ketentuan tertentu, jika tidak lakukan hal lain.
- Fungsi Seleksi: Memilih *powerup* berdasarkan situasi yang ada, mengambil *powerup* berdasarkan situasi yang ada
- Fungsi Kelayakan: Memeriksa apakah ada *powerup* dalam *inventory*
- Fungsi Objektif: Menggunakan *powerup* yang dapat meminimalkan jarak antara player dengan lawan jika player di belakang lawan atau memaksimalkan jarak jika player berada di depan musuh.

b. Analisis Efisiensi Solusi

Untuk penggunaan *powerup*, melakukan looping pada *list of powerup* sebanyak *n* *powerup* untuk memeriksa apakah ada *powerup* yang ingin digunakan.

$$T(n) = N$$

Jika diinginkan untuk mencari *powerup* maka efisiensinya menyerupai dengan pengecekan *obstacle* pada strategi *greedy by obstacle*.

Maka Big O Notationnya adalah $O(n)$.

c. Analisis Efektivitas Solusi

Strategi ini Efektif apabila

- Berada pada posisi sesuai dengan kegunaan dari *powerup*

Strategi ini kurang Efektif apabila

- Bila *powerup* digunakan terus dan *powerup* ada banyak, maka akan mengganggu pergerakan dari bot, sehingga sulit untuk mengejar.

3.1.3 Greedy by Damage

Greedy by Damage adalah strategi greedy yang mengutamakan dalam meminimumkan besar *damage* yang diterima oleh bot. Bot akan memilih lane yang memberikan *damage* terkecil. Algoritma ini hanya akan dijalankan apabila sudah dipastikan bahwa untuk *turn* selanjutnya bot akan menabrak. Apabila memungkinkan untuk tidak menabrak, maka *greedy by obstacle* akan dijalankan. Sekilas strategi greedy ini menyerupai strategi *Greedy by Obstacle*. Hal ini dikarenakan strategi *Greedy by Damage* merupakan substrategi dari *Greedy by Obstacle*.

a. Mapping Elemen - Elemen Greedy

- Himpunan Kandidat: Terrain sepanjang lane dalam jangkauan block
- Himpunan Solusi: memilih lane dengan damage terkecil dari semua lane yang memiliki obstacle
- Fungsi Solusi: Memeriksa total damage pada obstacle untuk lane depan, kiri, dan kanan
- Fungsi Seleksi: Pilih damage minimum kemudian melakukan command menuju lane dengan damage minimum. Jika total damage sama, maka accelerate.
- Fungsi Kelayakan: Memeriksa jika lane memiliki damage minimum maka mobil akan melaju menuju lane tersebut.
- Fungsi Objektif: Minimalisir damage yang diberikan kepada mobil jika sudah pasti akan menabrak

b. Analisis Efisiensi Solusi

Seleksi lane dimulai dengan traversal searching sepanjang N dengan N adalah kecepatan mobil. Kecepatan mobil menentukan berapa blocks yang harus dicari karena kemungkinan mobil hanya akan menabrak pada range tersebut. Apabila mobil berada di ujung, maka looping akan dilakukan sebanyak 2N, sedangkan apabila mobil berada di tengah lane (ada lane kiri dan lane kanan), maka looping dilakukan sebanyak 3N. Penjumlahan dilakukan sebanyak 3N (Cari wall, cari mud, dan cari oil). Cek kelayakan dilakukan dengan mengurutkan damage dari total damage masing-masing. Efisiensi waktu algoritma adalah

$$T(n) = 6N^2, 2 \leq \text{lane} \leq 3$$

$$T(n) = 9N^2, \text{lane} = 1 \text{ or lane} = 4$$

Maka Big O Notationnya adalah $O(N^2)$

c. Analisis Efektivitas Solusi

Strategi ini Efektif apabila

- Mobil pasti menabrak akan tetapi obstacle di depan berbeda-beda sehingga bisa memilih lane dengan damage yang paling kecil

Strategi ini kurang Efektif apabila

- Mobil pasti menabrak akan tetapi damage untuk semua lane sama. Strategi akan secara langsung meminta mobil untuk accelerate dan tidak akan efektif.

3.1.4 Greedy by Speed

Greedy by Speed adalah strategi greedy yang berusaha untuk memaksimalkan speed yang dimiliki dari bot. Strategi ini juga memanfaatkan *power-up boost* sebagai prioritas powerup karena efeknya yang dapat meningkatkan speed sampai range speed maksimum. Pada strategi greedy ini memfokuskan pada penggunaan command *ACCELERATE*, *BOOST*, dan *FIX*.

a. Mapping Elemen - Elemen Greedy

- Himpunan Kandidat: command *ACCELERATE*, *BOOST*, dan *FIX*.
- Himpunan Solusi: Speed dari player
- Fungsi Solusi: Melihat kondisi speed pada bot player, memastikan agar speed dari bot tidak berada dibawah batas yang telah ditentukan oleh player

- Fungsi Seleksi: Menjalankan command berdasarkan kondisi yang sedang dialami oleh bot
 - Fungsi Kelayakan: Memeriksa apakah command perlu digunakan pada situasi yang sedang dialami oleh bot
 - Fungsi Objektif: Memaksimalkan kecepatan bot pada ronde yang sedang dijalani.
- b. Analisis Efisiensi Solusi
- Karena hanya memeriksa speed dari bot, maka efisiensi dari strategi ini adalah $T(n) = 1$
Maka Big O Notationnya adalah $O(1)$.
- c. Analisis Efektivitas Solusi
- Strategi ini Efektif apabila
- Tidak ada obstacle di depan
- Strategi ini kurang Efektif apabila
- Banyak obstacle

3.2 Solusi Algoritma Greedy yang Dipilih dan Pertimbangannya

Pada game “Overdrive” ini, terdapat beberapa strategi greedy yang kami terapkan pada bot yang kami kembangkan. Strategi Greedy yang digunakan juga menyesuaikan dengan situasi yang sedang dialami oleh bot setiap rondanya. Strategi Greedy yang digunakan dalam pengembangan bot adalah *Greedy by Obstacle*, *Greedy by Damage*, dan *Greedy by Powerup*.

Pertimbangan pertama memilih strategi *Greedy by Obstacle* adalah untuk menghindari suatu obstacle yang dapat memperlambat speed dari bot, atau bahkan membuat bot berhenti. Kondisi itu menjadi fokus utama dalam pengembangan bot agar bot dapat mempertahankan kecepatannya pada kecepatan yang optimal. Dari pertimbangan pertama yang sudah disebutkan sebelumnya, untuk mengoptimalkan strategi sebelumnya, maka kami mengkombinasikannya dengan strategi *Greedy by Damage*.

Pertimbangan kami mengkombinasikan strategi *Greedy by Damage* dengan *Greedy by Obstacle* adalah karena adanya kemungkinan bahwa semua opsi lane ditutup oleh obstacle sehingga sulit untuk menentukan obstacle mana yang akan memberikan solusi paling optimal. Dengan adanya strategi *Greedy by Damage*, bot dapat menentukan dari semua alternatif lane yang berada dalam jangkauan, lane mana yang memberikan damage paling sedikit. Jika pada strategi *Greedy by Obstacle* hanya menghindari obstacle berdasarkan tingkatan yang ada pada *game rules*, yaitu fokus utama yang dihindari adalah *Cybertruck*, lalu diikuti oleh *Wall*, *Mud*, dan terakhir *Oil Spill*, dengan adanya strategi *Greedy by Damage*, jika setiap lane ada obstacle yang duplikat, maka dapat ditentukan untuk beralih ke lane yang memberikan *damage* paling kecil.

Jika obstacle di depan bot kosong, maka dapat diterapkan strategi *Greedy by Powerup* yaitu menggunakan power up jika ada dan sesuai dengan kondisi, atau jika kondisi penggunaan power up tidak terpenuhi, maka lakukan proses pencarian power up dalam jangkauan bot dengan syarat tidak ada obstacle. Hal ini dapat membantu bot mengumpulkan powerup dan menggunakannya sehingga saat obstacle kosong, bot tidak harus menjalankan *command accelerate* saja.

BAB IV

Implementasi & Pengujian

4.1 Pseudocode

Fungsi `getManyBlocks` untuk mengembalikan blocks pada lane dan block tertentu

```
function getManyBlocks(input lane: integer, block: integer) →  
List<object>
```

DEKLARASI VARIABEL

```
    map: List<Lane[]>  
    blocks: List<Object>  
    startBlock: integer  
    laneList: Lane[]
```

ALGORITMA

```
    map ← gameState.lanes  
    blocks ← ArrayList<>()  
    startBlock ← map.get(0)[0].position.block  
    laneList ← map.get(lane = 1)  
    i traversal [max(block - startBlock, 0)...block-startBlock +  
16]  
        if (laneList[i] = null or laneList[i].terrain =  
Terrain.FINISH) then  
            break  
            blocks.add(laneList[i].terrain)  
    → blocks
```

Fungsi *isOppInRange* untuk mengecek apakah musuh berada pada jangkauan

function isOppInRange() → boolean

ALGORITMA

```
    if (myCar.position.lane = opponent.position.lane) then
        range: int
        range ← opponent.position.block -
myCar.position.block
        if (myCar.speed ≤ 9 and range ≤ 9) then
            → true
        if (myCar.speed = 15 and) then
            → true
    → false
```

Function *countTerrain* untuk menghitung jumlah terrain yang bernama “*terrainToCheck*” pada blocks

```
function countTerrain(input blocks: List<object>,  
terrainToCheck: Terrain) → integer
```

DEKLARASI VARIABEL

```
    count: int
```

```
    block: Object
```

ALGORITMA

```
    count ← 0
```

```
    block ← blocks
```

```
    i traversal[1...block.size()]
```

```
        if (block = terrainToCheck) then
```

```
            count ← count + 1
```

```
→ count
```


Fungsi *hasCyberTruck* mengembalikan true jika terdapat cybertruck pada lane

```
function hasCyberTruck(input lane: integer) → boolean
```

```
DEKLARASI VARIABEL
```

```
    block : integer
```

```
    map : List<Lane[]>
```

```
    startBlock : integer
```

```
    laneList : Lane[]
```

```
ALGORITMA
```

```
    block ← myCar.position.block
```

```
    map ← gameState.lanes
```

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

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

```
    i traversal [max(block - startBlock, 0)...block-startBlock +  
16]
```

```
        if (laneList[i] = null or laneList[i].terrain =  
Terrain.FINISH) then
```

```
            break
```

```
            if (laneList[i].isOccupiedByCyberTruck) then
```

```
                → true
```

```
    → false
```

Fungsi *checkEmpRange* mengirimkan true jika jarak mobil musuh cukup dekat untuk di EMP

```
function checkEmpRange() → boolean
```

DEKLARASI VARIABEL

-

ALGORITMA

```
→ myCar.position.lane = opponent.position.lane or  
opponent.position.lane - myCar.position.lane = 1 or  
myCar.position.lane - opponent.position.lane = 1
```

Fungsi *obstacles* mengirimkan true jika terdapat obstacles pada blocks

```
function obstacles(input blocks: List<Object>) → boolean
```

DEKLARASI VARIABEL

-

ALGORITMA

```
→ blocks.contains(Terrain.MUD) or  
blocks.contains(Terrain.WALL) or  
blocks.contains(Terrain.OIL_SPILL)
```

Fungsi *containPowerUp* mengirimkan true jika terdapat power up pada blocks

```
function containPowerUp(input blocks: List<Object>) → boolean
```

DEKLARASI VARIABEL

-

ALGORITMA

```
→ blocks.contains(Terrain.BOOST) or  
blocks.contains(Terrain.LIZARD) or blocks.contains(Terrain.EMP)  
or blocks.contains(Terrain.TWEET) or  
blocks.contains(Terrain.OIL_POWER)
```

Fungsi *definiteCrash* mengembalikan true jika mobil dipastikan akan menabrak obstacle

```
function definiteCrash(input lane: integer) → boolean

DEKLARASI VARIABEL

-

ALGORITMA

    if (lane = 1) then

        → (hasCyberTruck(myCar.position.lane) or
        obstacles(getBlocksInFront(myCar.position.lane,
        myCar.position.block))) and (hasCyberTruck(myCar.position.lane
        + 1) or obstacles(getBlocksInFront(myCar.position.lane + 1,
        myCar.position.block)))

    else if (lane = 2 or lane = 3) then

        → (hasCyberTruck(myCar.position.lane) or
        obstacles(getBlocksInFront(myCar.position.lane,
        myCar.position.block))) and (hasCyberTruck(myCar.position.lane
        + 1) or obstacles(getBlocksInFront(myCar.position.lane + 1,
        myCar.position.block))) and (hasCyberTruck(myCar.position.lane
        - 1) or obstacles(getBlocksInFront(myCar.position.lane-1,
        myCar.position.block)))

    else

        → (hasCyberTruck(myCar.position.lane) or
        obstacles(getBlocksInFront(myCar.position.lane,
        myCar.position.block))) and (hasCyberTruck(myCar.position.lane
        - 1) or obstacles(getBlocksInFront(myCar.position.lane-1,
        myCar.position.block)))
```

Fungsi *getDamageBlocks* mengembalikan jumlah damage yang diberikan total obstacle pada blocks yang berada di lane

```
function getDamageBlocks(input blocks: List<Object>, lane:
integer) → integer

DEKLARASI VARIABEL

    damage : integer

ALGORITMA

    damage ← 0

    damage ← damage + countTerrain(blocks, Terrain.OIL_SPILL)

    damage ← damage + countTerrain(blocks, Terrain.MUD)

    damage ← damage + countTerrain(blocks, Terrain.WALL) * 2

    if (hasCyberTruck(lane)) then

        damage ← damage + 2

    → damage
```

```
function findPowerUp(input blocks: List<Object>) → Command
```

```
DEKLARASI VARIABEL
```

```
-
```

```
ALGORITMA
```

```
    if (not obstacles(blocks and not  
hasCyberTruck(myCar.position.lane)) then  
  
        if (myCar.position.lane = 1) then  
  
            blocksRight : List<Object>  
  
            blocksRight ←  
getBlocksInFront(myCar.position.lane + 1, myCar.position.block)  
  
            if (containPowerUp(blocks)) then  
  
                → ACCELERATE  
  
            if (containPowerUp(blocksRight)) then  
  
                if (not obstacles(blocksRight) and not  
hasCyberTruck(myCar.position.lane + 1)) then  
  
                    → TURN_RIGHT  
  
            if (myCar.position.lane = getMaxLane()) then  
  
                blocksLeft : List<Object>  
  
                blocksLeft ←  
getBlocksInFront(myCar.position.lane - 1, myCar.position.block)  
  
                if (containPowerUp(blocks)) then  
  
                    → ACCELERATE  
  
                if (containPowerUp(blocksLeft)) then  
  
                    if (not obstacles(blocksLeft) and not  
hasCyberTruck(myCar.position.lane - 1)) then  
  
                        → TURN_LEFT
```

```

        if (myCar.position.lane > 1 and myCar.position.lane <
getMaxLane()) then

            blocksLeft : List<Object>

            blocksRight: List<Object>

            blocksLeft ←
getBlocksInFront(myCar.position.lane - 1, myCar.position.block)

            blocksRight ←
getBlocksInFront(myCar.position.lane + 1, myCar.position.block)

            if (containPowerUp(blocks) then

                → ACCELERATE

            if (containPowerUp(blocksLeft)) then

                if (not obstacles(blocksLeft) and not
hasCyberTruck(myCar.position.lane - 1)) then

                    → TURN_LEFT

            if (containPowerUp(blocksRight)) then

                if (not obstacles(blocksRight) and not
hasCyberTruck(myCar.position.lane + 1)) then

                    → TURN_RIGHT

            if (containPowerUp(blocksLeft) and
containPowerUp(blocksRight)) then

                if (not obstacles(blocksLeft) and not
obstacles(blocksRight) and not
hasCyberTruck(myCar.position.lane - 1) and not
hasCyberTruck(myCar.position.lane + 1)) then

                    i : integer

                    i ←
random.nextInt(directionList.size())

                    → directionList.get(i)

```

→ ACCELERATE

```
function changeLane(input blocks: List<Object>) → Command
```

```
DEKLARASI VARIABEL
```

```
-
```

```
ALGORITMA
```

```
    if (not obstacles(blocks) and not  
hasCyberTruck(myCar.position.lane)) then
```

```
        → ACCELERATE
```

```
    if (myCar.position.lane = 1) then
```

```
        blocksRight : List<Object>
```

```
        blocksRight ← getBlocksInFront(myCar.position.lane +  
1, myCar.position.block)
```

```
        if (not obstacles(blocksRight) and not  
hasCyberTruck(myCar.position.lane + 1)) then
```

```
            → TURN_RIGHT
```

```
            if (countTerrain(blocks, Terrain.MUD) ≥ 2 or  
blocks.contains(Terrain.Wall) or  
hasCyberTruck(myCar.position.lane) or isOppInRange()) then
```

```
                if (hasPowerUp(PowerUps.LIZARD,  
myCar.powerups)) then
```

```
                    → LIZARD
```

```
            if (definiteCrash(1)) then
```

```
                damageRight : integer
```

```
                damageStraight : integer
```

```
                damageRight ← getDamageBlocks(blocksRight,  
myCar.position.lane + 1)
```

```
                damageStraight ← getDamageBlocks(blocks,  
myCar.position.lane)
```



```

    if (damageRight < damageStraight) then
        → TURN_RIGHT
    else
        → ACCELERATE

    if (hasCyberTruck(myCar.position.lane)) then
        → TURN_RIGHT

    if (hasCyberTruck(myCar.position.lane + 1)) then
        → ACCELERATE

    if (blocksRight.contains(Terrain.WALL) and not
blocks.contains(Terrain.WALL)) then
        → ACCELERATE

    if (not blocksRight.contains(Terrain.WALL and not
blocks.contains(Terrain.WALL)) then
        → TURN_RIGHT

    if (blocksRight.contains(Terrain.MUD)) then
        if (countTerrain(blocks, Terrain.MUD) >
countTerrain(blocksRight, Terrain.MUD)) then
            → TURN_RIGHT
        → ACCELERATE

    if (isOppInRange())
        → TURN_RIGHT
    → TURN_RIGHT

    if (myCar.position.lane = getMaxLane()) then

        blocksLeft : List<object>

```

```

        blocksLeft ← getBlocksInFront(myCar.position.lane -
1, myCar.position.block)

        if (not obstacles(blocksLeft) and not
hasCyberTruck(myCar.position.lane - 1)) then

            → TURN_LEFT

        if (countTerrain(blocks, Terrain.MUD) ≥ 2 or
blocks.contains(Terrain.Wall) or
hasCyberTruck(myCar.position.lane) or isOppInRange()) then

            if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) then

                → LIZARD

        if (definiteCrash(getMaxLane())) then

            damageLeft : integer

            damageStraight : integer

            damageLeft ← getDamageBlocks(blocksLeft,
myCar.position.lane - 1)

            damageStraight ← getDamageBlocks(blocks,
myCar.position.lane)

            if (damageLeft < damageStraight) then

                → TURN_LEFT

            else

                → ACCELERATE

        if (hasCyberTruck(myCar.position.lane)) then

            → TURN_LEFT

        if (hasCyberTruck(myCar.position.lane - 1)) then

            → ACCELERATE

```

```

        if (blocksLeft.contains(Terrain.WALL) and not
blocks.contains(Terrain.WALL)) then

            → ACCELERATE

        if (not blocksLeft.contains(Terrain.WALL) and
blocks.contains(Terrain.WALL)) then

            → TURN_LEFT

        if (blocksLeft.contains(Terrain.MUD)) then

            if (countTerrain(blocks, Terrain.MUD) >
countTerrain(blocksLeft, Terrain.MUD)) then

                → TURN_LEFT

            → ACCELERATE

        if (isOppInRange()) then

            → TURN_LEFT

        → TURN_LEFT

    if (myCar.position.lane > 1 and myCar.position.lane <
getMaxLane()) then

        blocksRight : List<Object>

        blocksLeft  : List<Object>

        blocksRight ← getBlocksInFront(myCar.position.lane +
1, myCar.position.block)

        blocksLeft  ← getBlocksInFront(myCar.position.lane -
1, myCar.position.block)

        if (not obstacles(blocksLeft) and not
hasCyberTruck(myCar.position.lane - 1)) then

            → TURN_LEFT

        if (not obstacles(blocksRight) and not
hasCyberTruck(myCar.position.lane + 1)) then

```

```

→ TURN_RIGHT

    if (countTerrain(blocks, Terrain.MUD ≥ 3 or
blocks.contains(Terrain.Wall) or
hasCyberTruck(myCar.position.lane) or isOppInRange()) then

        if (hasPowerUp(PowerUps.LIZARD,
myCar.powerups)) then

            → LIZARD

            if (myCar.position.lane = 2) then

                if (definiteCrash(2)) then

                    damageRight : integer

                    damageLeft : integer

                    damageStraight : integer

                    damageRight ← getDamageBlocks(blocksRight,
myCar.position.lane + 1

                    damageLeft ← getDamageBlocks(blocksLeft,
myCar.position.lane - 1)

                    damageStraight ← getDamageBlocks(blocks,
myCar.position.lane)

                    min : integer

                    min ← Math.min(Math.min(damageRight,
damageLeft), damageStraight)

                    if (min == damageStraight) then

                        → ACCELERATE

                    else if (min = damageRight) then

                        → TURN_RIGHT

                    else

                        → TURN_LEFT

```

```

else
    if (definiteCrash(3)) then
        damageRight : integer
        damageLeft : integer
        damageStraight : integer
        damageRight ← getDamageBlocks(blocksRight,
myCar.position.lane + 1
        damageLeft ← getDamageBlocks(blocksLeft,
myCar.position.lane - 1)
        damageStraight ← getDamageBlocks(blocks,
myCar.position.lane)
        min : integer
        min ← Math.min(Math.min(damageRight,
damageLeft), damageStraight)
        if (min == damageStraight) then
            → ACCELERATE
        else if (min = damageRight) then
            → TURN_RIGHT
        else
            → TURN_LEFT

→ ACCELERATE

```

```
function run() → Command
```

```
DEKLARASI VARIABEL
```

```
    blocks : List<Object>
```

```
    manyBlocks : List<Object>
```

```
    blocks ← getBlocksInFront(myCar.position.lane,  
myCar.position.block)
```

```
    manyBlocks ← getManyBlocks(myCar.position.lane,  
myCar.position.block)
```

```
    if (myCar.damage ≥ 2) then
```

```
        → FIX
```

```
    if (obstacles(blocks) or  
hasCyberTruck(myCar.position.lane)) then
```

```
        → changeLane(blocks)
```

```
    if (hasPowerUp(PowerUps.BOOST, myCar.powerups)) then
```

```
        if (not obstacles(manyBlocks) and not  
hasCyberTruck(myCar.position.lane)) then
```

```
            if (not myCar.boosting) then
```

```
                if (myCar.damage ≠ 0) then
```

```
                    → FIX
```

```
                → BOOST
```

```
    if (myCar.speed ≤ 5) then
```

```
        → ACCELERATE
```

```
    if (myCar.position.block < opponent.position.block) then
```

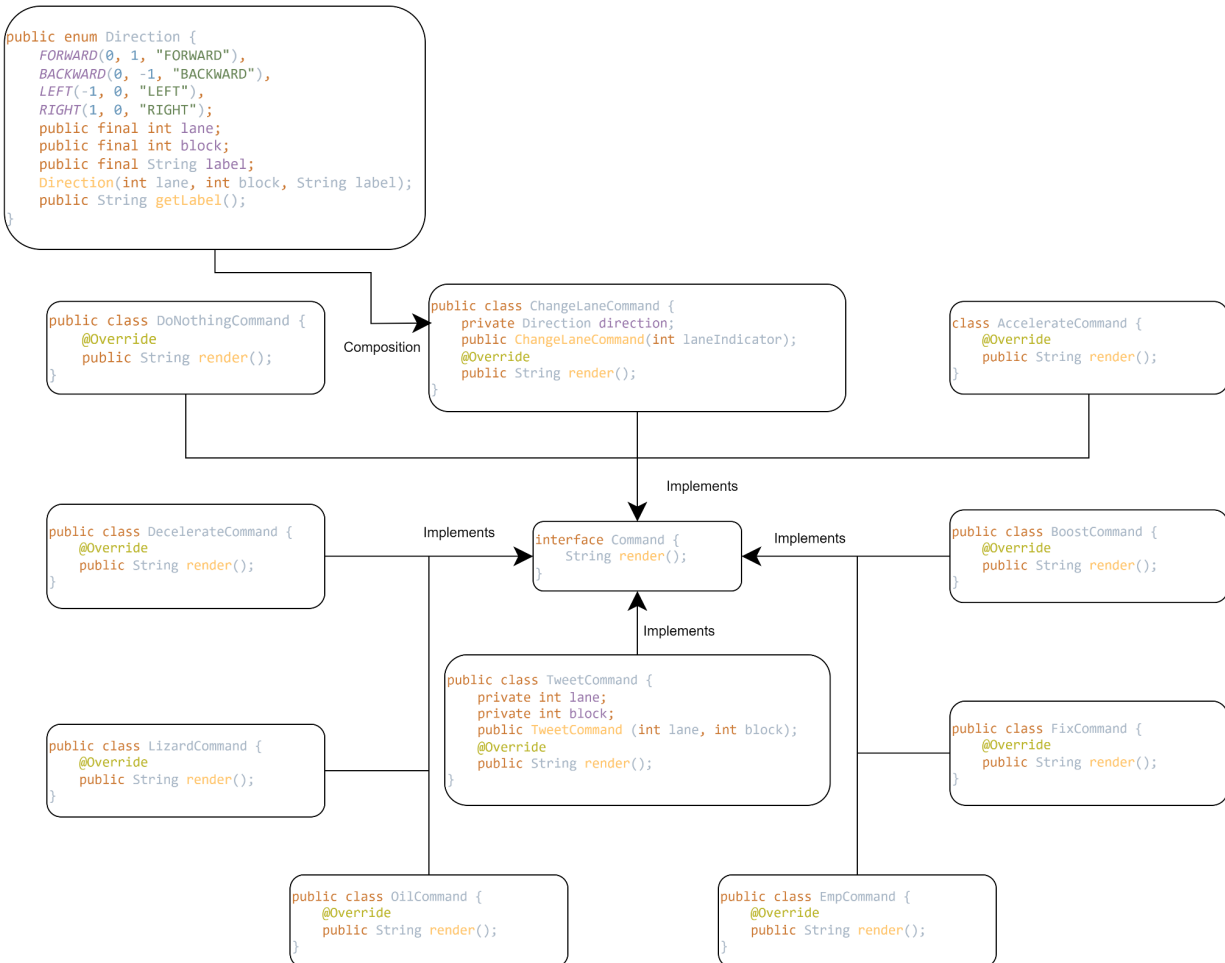
```
        → if (checkEmpRange()) then
```

→ EMP

```
if (myCar.position.block > opponent.position.block) then
    if (hasPowerUp(PowerUps.TWEET, myCar.powerups)) then
        if ((myCar.position.block -
opponent.position.block ≥ Bot.maxBoostSpeed) then
            → new TweetCommand(opponent.position.lane,
opponent.position.block + opponent.speed + 5)
        if (hasPowerUp(PowerUps.OIL, myCar.powerups)) then
            → OIL
    → findPowerUp(blocks)
```

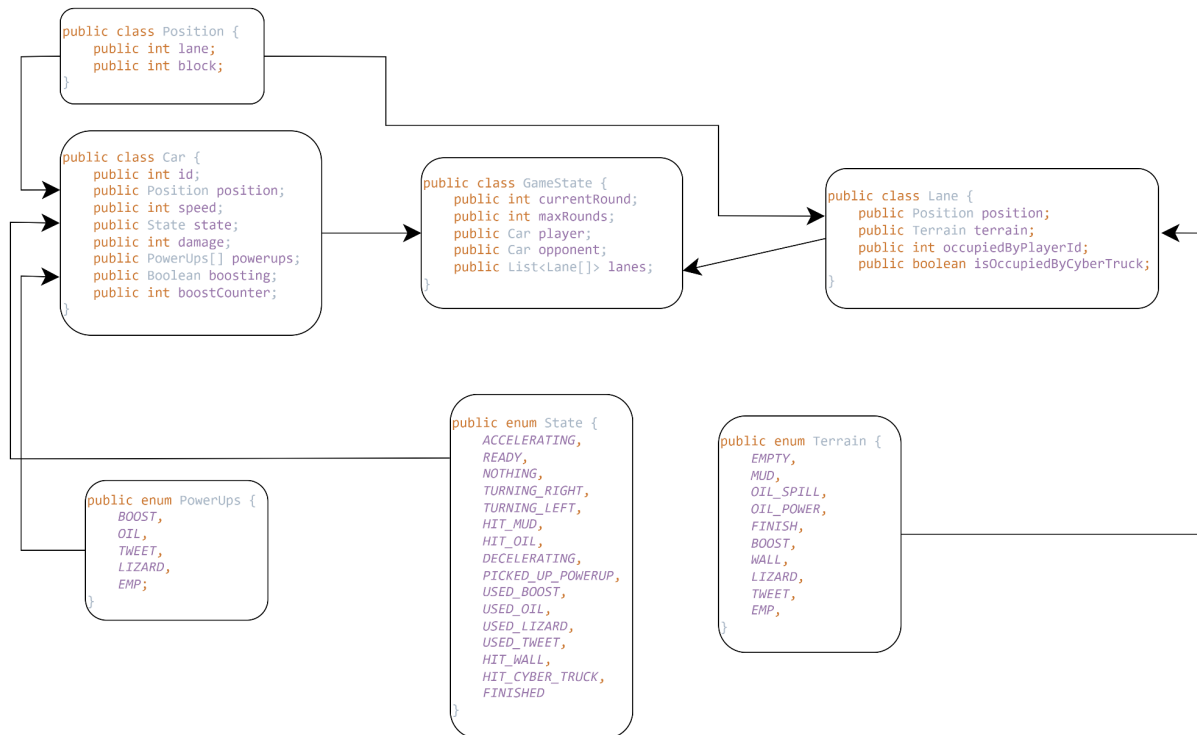
4.2 Struktur Data

Command



Command merupakan kelas yang berisikan objek-objek yang dapat dipanggil memerintahkan car untuk melakukan suatu aksi. Semua kelas Command akan menggunakan metode render dimana tiap command akan mempunyai implementasi render yang berbeda-beda. Metode render sendiri adalah metode yang akan mengirimkan pesan ke game engine.

Entities



Entities adalah kelas yang berisikan objek-objek yang dapat di buat dalam game overdrive. Di game overdrive ada 4 entitas, yaitu *gameState*, dibuat untuk mencatat tiap properti status yang ada pada game, car, dibuat sebagai representasi mobil pada game dengan mencatat properti properti mobil, Lane, dibuat untuk merepresentasikan jalur yang ada dan mencatat properti khusus yang ada pada lane, dan Posisi sebagai representasi data posisi menyimpan informasi lane dan blok.

4.3 Analisis dan Pengujian

Berikut adalah hasil analisis dan pengujian strategi yang diimplementasi beserta analisisnya.

Pengujian I

```
Starting round: 2
Player A - HotWheels: Map View
=====
round:2
player: id:1 position: y:2 x:5 speed:5 state:TURNING_RIGHT statesThatOccurredThisRound:TURNING_RIGHT boosting:false boost-counter:0 damage:0 score:0 powerups:
opponent: id:2 position: y:4 x:7 speed:6

[#####f#####]
[#####>>#####]
[#####>#####]
[#####2#####]
```

Pada round 1, mobil bot menjalankan command TURN_RIGHT. Alasan menjalankan command tersebut adalah karena di depan bot terdapat terrain WALL, dan di sebelah kanan, berdasarkan method contain (untuk memeriksa apakah ada block yang dicari ada pada list of object dengan panjang 9 jika kecepatan block ≤ 9 dan 15 jika kecepatan block 15), hanya terdapat terrain MUD. Karena efek *slowing* dari WALL lebih besar dibanding MUD, maka block menjalankan command TURN_RIGHT untuk menghindari menabrak terrain WALL.

Pengujian II

```
round:46
player: id:1 position: y:1 x:584 speed:15 state:USED_TWEET statesThatOccurredThisRound:USED_TWEET boosting:true boost-counter:3 damage:0 score:170 powerups: BOOST:2, LIZARD:7, EMP:6
opponent: id:2 position: y:4 x:140 speed:6

[#####T#####f#####]
[#####*#####]
[#####]
[#####f#####]
=====

round:47
player: id:1 position: y:1 x:519 speed:15 state:USED_LIZARD statesThatOccurredThisRound:USED_LIZARD boosting:true boost-counter:2 damage:0 score:170 powerups: BOOST:2, LIZARD:6, EMP:6
opponent: id:2 position: y:4 x:144 speed:0

[#####T#####f*#####]
[#####f#####f#####]
[#####]
[#####f#####]
=====
```

Pada ronde 46, di depan dan di sebelah kanan block terdapat *obstacle*. Karena di dalam inventory bot memiliki power up Lizard, maka command LIZARD dijalankan untuk melompati obstacle yang ada didepannya.

Pengujian III

```
Starting round: 74
Player A - HotWheels: Map View
=====
round:74
player: id:1 position: y:3 x:828 speed:15 state:ACCELERATING statesThatOccurredThisRound:ACCELERATING boosting:true boost-counter:2 damage:0 score:238 powerups: BOOST:3, LIZARD:9, EMP:6
opponent: id:2 position: y:4 x:263 speed:6

[#####]
[#####]
[#####]
[#####]
```

Pada round 74, posisi bot berada di lane 3 dan di depannya terdapat obstacle yaitu Terrain MUD. Karena di depan terdapat obstacle, maka dijalankan strategi *Greedy by Obstacle*. Setelah menjalankan strategi tersebut, ternyata pada lane sebelah kiri dan kanan terdapat obstacle juga yang sama sama hanya memiliki Terrain MUD. Maka selanjutnya dijalankan strategi *Greedy by Damage*, yaitu memeriksa dari ketiga lane yang ada, mana yang memberikan damage terkecil. Setelah dilakukan pemeriksaan, lane di depan memiliki damage terkecil, diikuti dengan lane kanan, lalu lane kiri dengan damage terbesar (kondisi mobil sedang boosting, maka pengecekan block sebanyak 15 block). Maka lane yang dipilih adalah lane di depan sehingga bot menjalankan command ACCELERATE.

Pengujian IV

```
Starting round: 97
Player A - HotWheels: Map View
=====
round:97
player: id:1 position: y:2 x:1102 speed:9 state:USED_OIL statesThatOccurredThisRound:NOTHING, USED_OIL boosting:false boost-counter:0 damage:0 score:330 powerups: BOOST:3, LIZARD:13, EMP:12
opponent: id:2 position: y:4 x:336 speed:0

[#####]
[#####]
[#####]
[#####]

Starting round: 98
Player A - HotWheels: Map View
=====
round:98
player: id:1 position: y:3 x:1110 speed:9 state:TURNING_RIGHT statesThatOccurredThisRound:TURNING_RIGHT boosting:false boost-counter:0 damage:0 score:334 powerups: BOOST:4, LIZARD:13, EMP:12
opponent: id:2 position: y:4 x:336 speed:0

[#####]
[#####]
[#####]
[#####]
```

Pada round 97, di depan bot tidak ada obstacle sama sekali, oleh karena itu, bot dapat menjalankan strategi *Greedy by Powerup*. Kondisi bot saat ini tidak terlalu membutuhkan menggunakan suatu power up, sehingga dapat dilakukan prose pencarian power up disekitarnya. Karena bot berada di lane 2, maka dilakukan pemeriksaan pada lane 1 dan 3 apakah terdapat powerup dan bebas dari obstacle. Setelah dilakukan pemeriksaan, lane 3 memiliki power up, oleh karena itu, bot menjalankan command TURN_RIGHT untuk mengambil powerup.

BAB V

Kesimpulan & Saran

5.1 Kesimpulan

Dari Tugas Besar IF2211 Strategi Algoritma semester 2 2021/2022 berjudul “Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan ‘Overdrive’”, kami berhasil mengembangkan sebuah bot dalam permainan “Overdrive” dari logic dasar yang sudah ada dengan memanfaatkan strategi greedy untuk mengembangkan kemampuan bot saat dijalankan. Program tidak hanya menggunakan satu Algoritma Greedy, melainkan dikombinasikan dengan strategi greedy yang lain untuk menghasilkan strategi yang optimal. Meskipun demikian, strategi yang telah kami buat tentunya bukan merupakan strategi yang paling ideal. Hal ini tentunya dikarenakan strategi greedy tidak selalu menghasilkan solusi yang paling baik. Secara garis besar, tugas ini telah mengajarkan kami untuk berpikir kreatif dalam menerapkan algoritma greedy.

5.2 Saran

Saran yang dapat kami berikan untuk tugas besar IF 2211 Strategi Algoritma Semester 2 2021/2022 meliputi:

1. Algoritma greedy yang diterapkan masih bisa lebih dikembangkan lagi menjadi lebih efisien. Algoritma greedy yang kami telah buat sudah bisa memenangkan pertandingan melawan bot lain, akan tetapi tidak konsisten. Kombinasi greedy dengan algoritma untuk memprediksi command bot musuh menjadi salah satu pengembangan yang dapat dilakukan
2. Komentar pada program masih kurang maksimal dan efektif
3. Penulisan pseudocode sepertinya juga masih kurang maksimal dan detail.

Daftar Pustaka

Entellect Challenge (2020, Agustus 17). Entellect Challenge 2020 - Overdrive. Tersedia di: <https://github.com/EntellectChallenge/2020-Overdrive> (Diakses tanggal 11 Februari 2022)

Munir, Rinaldi (2022). Algoritma Greedy (Bagian 1). Tersedia di: [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 tanggal 13 Februari 2022)

Munir, Rinaldi (2022). Algoritma Greedy (Bagian 2). Tersedia di: [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 tanggal 15 Februari 2022)

Munir, Rinaldi (2022). Algoritma Greedy (Bagian 3). Tersedia di: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf) (Diakses tanggal 16 Februari 2022)

Link Repository

<https://github.com/jasonk19/Tubes-STIMA-1.git>