LAPORAN TUGAS BESAR 1

IF2211 Strategi Algoritma

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"

Disusun oleh:
Kelompok 20

1. Gede Sumerta Yoga (13520021)
2. Johannes Winson Sukiatmodjo (13520123)
3. Ignasius Ferry Priguna (13520126)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2022

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

DESKRIPSI TUGAS

1.1 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 1.1 Ilustrasi permainan *Overdrive*

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Overdrive*. *Game engine* dapat diperoleh pada laman berikut:
https://github.com/EntelectChallenge/2020-Overdrive.

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini:
https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4
Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Overdrive* pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan memiliki bentuk array 2 dimensi yang memiliki 4 jalur lurus. Setiap jalur dibentuk oleh *block* yang saling berurutan, panjang peta terdiri atas 1500 *block*. Terdapat 5 tipe *block*, yaitu *Empty*, *Mud*, *Oil Spill*, *Flimsy Wall*, dan *Finish Line* yang masing-masing karakteristik dan efek berbeda. *Block* dapat memuat *powerups* yang bisa diambil oleh mobil yang melewati *block* tersebut.
2. Beberapa *powerups* yang tersedia adalah:
    a. *Oil item,* dapat menumpahkan oli di bawah mobil anda berada.
    b. *Boost,* dapat mempercepat kecepatan mobil anda secara drastis.
    c. *Lizard,* berguna untuk menghindari *lizard* yang mengganggu jalan mobil anda.
    d. *Tweet*, dapat menjatuhkan truk di *block* spesifik yang anda inginkan.
    e. *EMP,* dapat menembakkan *EMP* ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 *lane* yang sama) akan terus berada di *lane* yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.
3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 *block* untuk setiap *round*. *Game state* akan memberikan jarak pandang hingga 20 *block* di depan dan 5 *block* di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.
4. Terdapat *command* yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan *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

1.2 Spesifikasi tugas

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

1. *Download latest release starter pack.zip* dari tautan berikut
   https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4
2. Untuk menjalankan permainan, kalian butuh beberapa *requirement* dasar sebagai berikut.
   a. Java (minimal Java 8):

https://www.oracle.com/java/technologies/downloads/#java8

b. IntelliJ IDEA: https://www.jetbrains.com/idea/

c. NodeJS: https://nodejs.org/en/download/

3. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk *Windows* dapat buka dengan *double-click*, Untuk Linux/Mac dapat menjalankan *command* “make run”).
4. Secara *default*, permainan akan dilakukan diantara *reference bot* (*default*-nya berbahasa *Java*) dan *starter bot* (*default*-nya berbahasa *JavaScript*) yang disediakan. Untuk mengubah hal tersebut, silahkan edit file “game-runner-config.json”. Anda juga dapat mengubah file “bot.json” dalam direktori “starter-bots” untuk mengatur informasi terkait bot anda.
5. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di *starter-bots*. Ingat bahwa bot kalian harus menggunakan bahasa **Java** dan di-*build* menggunakan **IntelliJ** sebelum menjalankan permainan kembali. **Dilarang** menggunakan kode program yang sudah ada untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan *visualizer* berikut https://github.com/Affuta/overdrive-round-runner
7. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut.

Strategi *greedy* yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mencapai garis *finish* lebih awal atau mencapai garis *finish* bersamaan tetapi dengan kecepatan lebih besar atau memiliki skor terbesar jika kedua komponen tersebut masih bernilai imbang. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menggunakan *powerups* begitu ada untuk mengganggu mobil musuh. Buatlah strategi *greedy* terbaik, karena setiap bot dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (TBD).

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

BAB II

LANDASAN TEORI

2.1 Dasar teori

Algoritma *Greedy* merupakan sebuah algoritma yang sederhana dan cukup populer terutama untuk menyelesaikan permasalahan yang berkaitan dengan optimasi. Optimasi di sini mencakup mencari maksimasi ataupun minimasi dari suatu permasalahan. Algoritma ini berjalan langkah per langkah dengan di setiap langkahnya berusaha untuk mengambil pilihan yang optimal. Di setiap langkah tersebut harapannya telah memilih optimum lokal setiap langkah dan akan berakhir menjadi optimum global dari permasalahan.

Algoritma *Greedy* telah terbukti berhasil di beberapa masalah, seperti Kode Huffman, Algoritma Dijkstra, dan yang lainnya. Namun, ada juga masalah yang tidak berakhir dengan solusi paling optimal dari seluruh solusi yang ada, seperti *Travelling Salesman Problem*. Solusi *Greedy* dari permasalahan tersebut hanya menjadi hampiran dari solusi optimalnya. Cara termudah untuk membuktikan bahwa Algoritma *Greedy* tidak menghasilkan solusi yang optimal di suatu permasalahan adalah dengan CounterExample, yaitu menunjukkan bahwa ada solusi yang lebih optimal.

Pada Algoritma *Greedy*, ada elemen-elemen yang harus ditinjau, yaitu:

- Himpunan kandidat, C: isinya adalah kandidat-kandidat yang bisa dipilih pada setiap langkah.
- Himpunan solusi, S: isinya adalah kandidat-kandidat yang telah dipilih di setiap langkahnya.
- Fungsi solusi: fungsi yang mengecek apakah himpunan solusi sudah menyelesaikan masalah yang ditinjau.
- Fungsi seleksi: fungsi untuk memilih kandidat berdasarkan strategi *greedy* yang telah ditentukan.
- Fungsi kelayakan: fungsi yang memeriksa kelayakan suatu kandidat untuk masuk ke himpunan solusi.
- Fungsi objektif : tujuan dari algoritma *greedy* apakah ingin memaksimumkan atau meminimumkan.

Dapat dikatakan dari elemen-elemen tersebut Algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S di optimisasi oleh fungsi objektif.

2.2 Cara kerja program

Permainan ini bekerja dengan menggunakan *Game Engine*. *Game engine* adalah sebuah perangkat lunak yang dibuat khusus untuk menjalankan sebuah permainan. Pada tugas besar kali ini, Permainan “Overdrive” dijalankan dengan menggunakan *game engine* yang dibuat oleh Entelect. Secara default game ini akan berjalan dalam CLI, tetapi pada *repository github* resminya telah disediakan tiga visualizer untuk menggambarkan *game* ini.

Game engine ini menggunakan file dalam format .json untuk menyimpan komponen-komponen yang diperlukan dalam permainan, seperti *state* permainan, *command*/perintah, dan atribut lainnya. Sebagai contohnya ada game-config.json yang menyimpan atribut permainan seperti *command*, *state*, panjang lintasan, dan lainnya. Selain itu ada game-runner-config.json yang berisi informasi tentang lokasi penyimpanan *output* setiap ronde, lokasi penyimpanan file pemain pertama dan pemain kedua, dan masih ada lainnya.

Pada kondisi awal, *Game engine* ini sudah disiapkan oleh Entelect untuk langsung dimainkan dengan pertandingan antara reference-bot ataupun starter-bot. Kita tidak perlu mengatur konfigurasi untuk pertandingan default ini, tinggal klik file run.bat dan permainan akan dimulai.

Tentunya, kita juga bisa membuat bot kita sendiri dengan mengedit file di folder starter-bot. Di sini kami menggunakan bahasa Java dan kami telah mengedit strategi awal yang dimiliki starter-bot. Strategi tersebut kami implementasikan di file bot.java. Kemudian ada file yang bernama main.java yang akan membaca state saat ini, memberikannya kepada bot, menerima *output*-nya dan mencetaknya. Di tempat yang sama dengan kedua file tersebut juga terdapat tiga folder yang menyimpan atribut-atribut yang diperlukan, yaitu command, entities, dan enums.

Dalam permainan ini juga diperlukan file berekstensi .jar dan untuk membantu pembuatannya kami menggunakan bantuan Intellij IDEA. Dalam Intellij IDEA, terdapat Maven Toolbox yang secara otomatis akan membaca ada tidaknya project Java di Game Engine ini. Kemudian, untuk menghasilkan file .jar, dimanfaatkan fitur install di dalam Lifecycle pada bagian project Java yang muncul di Maven Toolbox. Kemudian akan muncul folder target di direktori starter-bot > java yang di dalamnya akan berisi file .jar yang diperlukan. Dengan ini, segala

sesuatu untuk menjalankan permainan telah terpenuhi dan tinggal menjalankan run.bat untuk memulai permainan. Terakhir, hasil output permainan akan disimpan di folder Match-Logs.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *mapping* persoalan *Overdrive*

a. *Mapping* Umum Permainan Overdrive

Berikut adalah pemetaan/mapping umum permainan Overdrive menjadi elemen-elemen algoritma *Greedy*.

Tabel 3.1 Mapping Umum Permainan Overdrive

Elemen Algoritma *Greedy*	***Mapping* pada Permainan Overdrive**
Himpunan Kandidat (C)	*Command-command* yang bisa dijalankan oleh *player* di setiap rondenya pada permainan. *Command* yang ada memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan *powerups.* Lebih lengkapnya dapat dilihat pada bagian deskripsi tugas.
Himpunan Solusi (S)	Command terbaik yang dapat dilakukan oleh bot mobil di setiap ronde tergantung dengan kondisi di sekitarnya.
Fungsi Solusi	Pemeriksaan terhadap *command* yang dipilih apakah termasuk *command* yang valid sesuai dengan peraturan permainan. Jika ternyata tidak valid, *command* ini bukanlah solusi yang harus dimasukkan ke himpunan solusi.
Fungsi Seleksi	Pemilihan *command* yang sesuai dengan strategi *greedy* yang telah diprogram kepada bot. Pada pemilihan ini telah ditetapkan urutan-urutan

	prioritas pengambilan *command* pada mayoritas situasi yang mungkin dihadapi oleh bot mobil di permainan.
Fungsi Kelayakan	Pemeriksaan terhadap *command* yang dipilih apakah dapat dilakukan pada kondisi yang dialami bot mobil di setiap rondenya. Sebagai contoh, *command* yang berkaitan dengan penggunaan suatu *powerup* tidak dapat dilakukan jika belum mendapatkan *powerup* tersebut.
Fungsi Objektif	Memenangkan balapan pada permainan "Overdrive", baik dengan menjadi yang pertama melewati blok *finish*, memiliki kecepatan akhir yang lebih cepat atau memiliki *score* yang lebih tinggi jika ternyata melewati blok *finish* pada ronde yang sama.

b. Mapping Pergerakan bot dalam Permainan "Overdrive"

Pergerakan bot-mobil pada permainan "Overdrive" merupakan sesuatu yang penting untuk dipikirkan. Pergerakan tersebut bisa membuat bot-mobil mendapatkan *powerups* yang lebih banyak dan bagus, melewati obstacle yang menghalangi, dan memiliki posisi di depan lawan. Berikut adalah mapping pergerakan bot dalam permainan "Overdrive" menjadi elemen-elemen dalam algoritma *Greedy*.

Tabel 3.2 Mapping Pergerakan bot dalam Permainan "Overdrive"

Elemen Algoritma *Greedy*	**Mapping pada Permainan "Overdrive"**
Himpunan Kandidat (C)	*Command-command* yang berkaitan dengan pergerakan, baik yang bertujuan untuk menambah kecepatan maupun menghindari obstacle. *Command-command* tersebut diantaranya ACCELERATE, BOOST, DECELERATE, USE_LIZARD, TURN LEFT, DAN TURN RIGHT

Himpunan Solusi (S)	*Command* yang terbaik dilakukan oleh bot mobil di setiap ronde tergantung dengan kondisi di sekitarnya.
Fungsi Solusi	Pemeriksaan terhadap *command* yang dipilih apakah termasuk *command* yang valid sesuai dengan peraturan permainan. Jika ternyata tidak valid, *command* ini bukanlah solusi yang harus dimasukkan ke himpunan solusi.
Fungsi Seleksi	Pemilihan *command* yang sesuai dengan strategi greedy yang telah diprogram kepada bot. Pada pemilihan ini telah ditetapkan urutan-urutan prioritas pengambilan *command* pada mayoritas situasi yang mungkin dihadapi oleh bot mobil di permainan.
Fungsi Kelayakan	Pemeriksaan terhadap *command* yang dipilih apakah dapat dilakukan pada kondisi yang dialami bot mobil di setiap rondenya. Contohnya: jika berada di *lane* nomor 1, tidak mungkin memilih command TURN_LEFT.
Fungsi Objektif	Memaksimalkan *command* yang ada untuk mendapatkan sebanyak mungkin powerups yang prioritas pengambilannya telah diatur. Selain itu, diminimalisasi penggunaan command NOTHING karena tidak akan menguntungkan bot dan ibarat menyia-nyiakan sebuah ronde dalam permainan

c. *Mapping* Penggunaan *Powerups* Menyerang di Permainan “Overdrive”

Selain ada *powerups* untuk membantu pergerakan bot-mobil, ada juga *powerups* yang fungsinya untuk menyerang atau mengganggu pergerakan bot-mobil lawan. Penggunaan command yang berkaitan dengan hal ini juga tidak kalah penting dengan bagian sebelumnya karena kita bisa mencapai block finish terlebih dahulu jika lawan

pergerakannya terganggu sehingga menjadi lebih lambat. Berikut merupakan mapping Penggunaan *Powerups* untuk Menyerang menjadi elemen-elemen dalam Algoritma *Greedy*.

Tabel 3.3 Mapping Penggunaan Powerups Menyerang di Permainan "Overdrive"

Elemen Algoritma *Greedy*	**Mapping pada Permainan "Overdrive"**
Himpunan Kandidat (C)	*Command-command* yang bersifat menyerang lawan seperti menghentikan pergerakannya dan memberi *obstacle* di lane. *Command* yang termasuk antara lain: USE_EMP, USE_OIL, dan USE_TWEET
Himpunan Solusi (S)	*Command* yang terbaik dilakukan oleh bot mobil di setiap ronde tergantung dengan kondisi di sekitarnya.
Fungsi Solusi	Pemeriksaan terhadap *command* yang dipilih apakah termasuk *command* yang valid sesuai dengan peraturan permainan. Jika ternyata tidak valid, *command* ini bukanlah solusi yang harus dimasukkan ke himpunan solusi.
Fungsi Seleksi	Pemilihan *command* yang sesuai dengan strategi greedy yang telah diprogram kepada bot. Pada pemilihan ini telah ditetapkan urutan-urutan prioritas pengambilan *command* pada mayoritas situasi yang mungkin dihadapi oleh bot mobil di permainan.
Fungsi Kelayakan	Memeriksa bahwa situasi dan kondisi yang dialami bot-mobil tersebut pada round itu memungkinkan untuk menggunakan *command* yang dipilih. Sebagai contoh, penggunaan command USE_EMP dilakukan jika berada di belakang lawan dan *lane* yang dekat.

Fungsi Objektif	Mengganggu lawan dengan mengurangi kecepatannya, memberikan *damage*, dan memberikan *obstacle* yang tidak menguntungkannya. Dengan membuat lawan terganggu seperti ini, tujuan akhir *game* ini akan semakin mudah untuk dicapai.

3.2 Eksplorasi alternatif solusi *greedy*

Mekanisme permainan Overdrive yang cukup banyak dan kompleks membuat banyak sekali strategi yang bisa diimplementasikan untuk mencapai tujuan dari permainan ini, yaitu memenangkan permainan. Oleh karena itu, eksplorasi strategi ini akan dibagi ke dalam kelompok-kelompok strategi untuk memudahkan klasifikasi strategi yang kami terapkan. Berikut merupakan alternatif-alternatif yang bisa diimplementasikan dalam game Overdrive ini.

1. Strategi pergerakan mobil

   Dalam permainan Overdrive, banyak sekali strategi pergerakan yang bisa kita implementasikan. Setiap strategi pergerakan juga bisa dipakai tergantung dari kondisi permainan yang terus berubah-ubah selama permainan berlangsung.

   Salah satu strategi pergerakan yang mungkin untuk dipakai adalah pergerakan menghindari obstacle yang berada di depan, kiri, atau kanan kita sejauh kecepatan mobil kita pada round tersebut. Jika terdapat obstacle di depan, kiri, dan kanan kita, kita bisa menggunakan powerup lizard untuk menghindarinya. Jika tidak memilikinya, mobil akan memilih lane dengan kecepatan akhir maksimum dan kerusakan yang minimum. Jika masih sama juga, mobil akan memilih lane yang memiliki total point prioritas terbanyak. Kami mendefinisikan lizard bernilai 2 poin, boost bernilai 1.75 poin, emp bernilai 1.5 poin, tweet bernilai 1.25 poin, dan yang terakhir oil bernilai 1 poin.

   Selain itu, mobil pemain bisa memilih untuk mengambil power up yang dekat di sekitarnya dengan syarat tidak ada obstacle di depan. Hal ini bertujuan supaya mobil kita memiliki persediaan powerup yang banyak agar memberikan keuntungan yang lebih besar dalam game serta memperoleh poin yang maksimal.

2. Strategi menyerang

Ketika menyerang, kita tentunya berharap tepat mengenai musuh. Dalam hal tersebut kita juga bisa mendapatkan poin ketika menggunakan powerup yang kita miliki. Misalnya, untuk emp harus dipakai jika musuh berada di depan kita dan berada di lane kita saat ini ataupun di lane kiri atau kanan kita. Hal ini bertujuan untuk memaksimalkan potensi yang dimiliki oleh powerup tersebut.

Selain itu, kita juga bisa menggunakan tweet untuk meletakkan cyber truck di posisi yang kita inginkan. Strategi terbaik untuk meletakkannya adalah di posisi lane musuh dan tidak jauh dari block dia sekarang. Hal ini bertujuan untuk memaksa musuh untuk berbelok ke arah yang lain.

Terakhir ada juga powerup oil yang memiliki efek yang sama dengan mud, yaitu menurunkan kecepatan sebesar 1 state dan memberikan damage sebesar 1. Menurut kami, strategi terbaik untuk meletakkan mud adalah jika tidak terdapat obstacle tak jauh di belakang kita ataupun meletakkannya jika posisi mobil kita satu lane dengan musuh, dengan catatan kita menggunakan powerup tersebut ketika musuh berada di belakang kita.

3. Strategi menaikkan speed

Secara garis besar, strategi ini berfokus pada kecepatan mobil kita sendiri. Jika memiliki kesempatan untuk mempercepat, maka command yang akan diprioritaskan yaitu antara boost ataupun accelerate. Tujuan strategi ini adalah memaksimalkan potensi powerup boost yang dimiliki ataupun meningkatkan kecepatan mobil selagi mampu demi mencapai garis finish secepat mungkin.

Strategi ini akan membaca obstacle di depan kita sejauh posisi berhenti mobil kita ketika memanggil command boost ataupun accelerate. Jika kondisinya tidak memungkinkan, maka akan dipilih command yang lain.

Jika ingin memanggil command boost, mobil kita harus dalam keadaan tidak memiliki damage sama sekali. Hal ini karena boost memiliki kecepatan sebesar 15 dan jika damage kita lebih dari 0, maka kecepatan maksimal yang didapat tidak akan menyentuh angka 15. Oleh karena itu, penting bagi kita untuk memastikan mobil kita dalam kondisi prima (tidak memiliki damage sama sekali) ketika ingin memanggil command boost.

3.3 Analisis efisiensi dan efektivitas alternatif solusi *greedy*

Sebenarnya dalam permainan Overdrive ini, kompleksitas waktu algoritma tidak terlalu penting karena game engine telah didesain untuk menerima perintah dari dua pemain dan menjalankannya secara bersamaan. Meskipun demikian, efisiensi dan efektivitas dari algoritma ini perlu dianalisis sehingga menghasilkan algoritma yang lebih greedy dan lebih baik lagi. Berikut merupakan analisis efisiensi dan efektivitas dari setiap strategi di atas (sebelum dikombinasikan).

3.3.1 Analisis efisiensi

Untuk strategi pergerakan mobil, terdapat beberapa looping bersarang, seperti list block di depan, kiri, kanan, ataupun belakang, pengecekan ada tidaknya obstacle, fungsi finalSpeedIfCollide, dan countPowerUpsPrioPoints. Variabel-variabel tersebut terdapat dalam strategi ini karena pada intinya melakukan pengecek terhadap lane yang akan dilalui. Namun, semua variabel tersebut belum tentu dipanggil oleh program kami dikarenakan ada beberapa hal yang tidak memenuhi kasus tertentu. Oleh karena itu, kami simpulkan bahwa kompleksitas yang dimiliki oleh strategi ini adalah $\Omega(n^4)$.

Untuk strategi menyerang, tidak begitu kompleks karena pada intinya hanya mengecek persediaan powerup yang kami miliki. Terdapat juga kasus khusus jika kami ingin menggunakan powerup oil, kami bakal mengecek terlebih dahulu ada tidaknya obstacle di belakang kami. Namun, kedua perulangan tersebut tidaklah bersarang. Oleh karena itu, kami simpulkan bahwa kompleksitas yang dimiliki oleh strategi ini adalah $\Omega(n)$.

Untuk strategi menaikkan speed, mirip dengan kompleksitas strategi menyerang, karena terdapat perulangan di bagian pengecekan persediaan powerup dan pengecekan lane saat kami hendak melakukan accelerate maupun boost. Karena kedua pengecekan tersebut tidak bersarang, maka kami simpulkan bahwa kompleksitas yang dimiliki oleh strategi ini adalah $\Omega(n)$.

3.3.2 Analisis efektivitas

Untuk strategi pergerakan mobil, algoritma yang kami gunakan sudah jauh berkembang dari pertama kali ide tersebut muncul, dari yang berawal hanya memeriksa apakah ada obstacle di depan sampai menganalisis lane yang paling worth it untuk dipilih. Tingkat keefektivitasannya juga sudah sangat baik, mengingat dengan algoritma ini kita bisa menentukan langkah terbaik ketika berbelok dengan mempertimbangkan

aspek damage yang diterima, kecepatan final setelah melewati obstacle, serta total poin prioritas dalam mengambil powerup yang tersedia.

Untuk strategi menyerang, kami lebih memprioritaskan emp terlebih dahulu, baru kemudian tweet dan yang terakhir adalah oil. Berdasarkan testing yang kami lakukan, didapatkan bahwa urutan prioritas inilah yang memberikan efektivitas yang paling tinggi. Hal tersebut sesuai dengan pertimbangan kami di awal memilih urutan tersebut, karena kami berpikir kalau powerup emp akan 100% mengenai musuh apabila digunakan pada saat yang tepat, sedangkan untuk tweet dan oil, musuh masih memiliki kesempatan untuk menghindarinya. Selain itu, emp juga sangat bermanfaat ketika mobil kita tertinggal oleh musuh. Selanjutnya, kami sudah melakukan testing dan mendapatkan keputusan bahwa tweet akan lebih efektif jika digunakan terlebih dahulu ketimbang oil. Hal itu dikarenakan tweet bisa diletakkan di mana saja dan memiliki efek damage dan penurunan kecepatan yang sangat signifikan ketimbang oil.

Untuk strategi menaikkan speed, strategi ini merupakan yang paling simple dibandingkan dengan kedua strategi sebelumnya. Algoritma yang kami terapkan sudah sangat efektif, mengingat kami memanfaatkan kecepatan boost dengan bersiap siaga untuk memperbaiki mobil terlebih dahulu sampai tidak memiliki damage. Setelah itu, barulah kami memprioritaskan accelerate karena saat boost tersebut habis, kecepatan akan turun ke maksimum speed, bukan kecepatan saat kita memanggil command boost. Kami juga memperbaiki mobil setiap kali memiliki damage lebih dari 1. Hal tersebut kami pertimbangkan karena setiap kali fix akan mengurangi damage sebesar 2, jadi akan kurang efektif apabila kita memanggil command fix ketika damage mobil kita masih 1.

3.4 Strategi *greedy* yang dipilih

Pada akhirnya, strategi final yang kami pilih merupakan kombinasi dari ketiga strategi di atas dengan urutan prioritas sebagai berikut beserta alasan dan pertimbangannya.

1. Fix jika memiliki damage lebih dari 1
   Alasannya agar memiliki max speed sebesar 9. Alasan kami tidak memilih damage lebih dari 0 karena setiap kali fix akan mengurangi damage sebesar 2, jadi rugi kalau masih memiliki 1 damage.

2. Accelerate jika speed sama dengan 0
   Alasannya hanya sekadar untuk berjaga-jaga jikalau mobil tersebut berhenti.

3. Belok kiri ataupun kanan jika ada obstacle ataupun musuh di depan
   Alasannya agar bisa meminimalkan damage yang diterima ataupun agar mobil kami tidak stuck di belakang mobil musuh.

   3.1 Gunakan lizard jika terdapat obstacle di depan, kiri, dan kanan mobil
   Alasannya sama seperti poin sebelumnya, yaitu meminimalkan damage yang diterima.

   3.2 Pilih lane dimana final speed setelah menabrak obstacle merupakan yang terbesar
   Alasannya karena tidak ada jalan lain selain terus maju. Oleh karena itu, kami mempertimbangkan hal itu dengan memilih lane yang memiliki final speed yang terbesar, agar bisa memaksimalkan kecepatan setelah menabrak obstacle.

   3.3 Pilih lane dimana total poin prioritas powerup merupakan yang terbesar
   Alasannya karena kami ingin memaksimalkan langkah ketika menentukan lane yang terbaik. Kami mendefinisikan lizard senilai 2 poin, emp senilai 1.75 poin, boost senilai 1.5 poin, tweet senilai 1.25 poin, dan yang terakhir oil senilai 1 poin. Kami memilih angka-angka tersebut karena disesuaikan dengan pertimbangan kami. Menurut kami, lizard merupakan powerup yang paling penting karena bisa dipakai saat tertinggal maupun saat memimpin. Powerup emp kami prioritaskan kedua sebagai amunisi jika mobil kami tertinggal dengan mobil musuh. Powerup boost kami prioritaskan ketiga karena tidak mudah untuk menggunakan boost, belum lagi jika musuh menggunakan emp. Keempat ada powerup tweet agar bisa memaksa musuh untuk berbelok di saat kami memimpin ataupun tertinggal. Terakhir ada powerup oil karena powerup ini memiliki chance terkena musuh yang sangat kecil.

4. Gunakan emp di saat musuh berada di depan, kiri, atau kanan
   Alasannya untuk mengejar ketertinggalan dari musuh. Kami juga memprioritaskan powerup emp dahulu agar membuat musuh tidak tenang dalam berkendara.

5. Gunakan tweet di lane musuh dan blok dimana musuh berhenti saat accelerate
   Alasannya agar mengganggu musuh di mana pun dan kapan pun selagi bisa. Kami meletakkan tweet di lane musuh berada dan blok dimana dia berhenti saat melakukan accelerate plus satu agar memaksimalkan kemungkinan memaksa musuh untuk berbelok.

6. Fix sebelum memanggil command boost jika damage sama dengan 1
   Alasannya agar memaksimalkan kecepatan dari boost itu sendiri. Kami berpikir bahwa sia-sia jika menggunakan boost apabila max speed yang dimiliki tidak sama dengan kecepatan boost.

7. Gunakan boost
   Alasannya agar mengejar ketertinggalan dari musuh ataupun semakin menjauh dari musuh. Kami menggunakan command boost terlebih dahulu sebelum accelerate karena kami ingin memaksimalkan kecepatan setelah boost tersebut habis.

8. Accelerate apabila speed akan bertambah
   Alasannya agar memiliki kecepatan yang konstan di state max speed.

9. Gunakan oil jika musuh berada di belakang
   Alasannya untuk mengganggu pergerakan musuh, sehingga dia akan terpaksa menghindari obstacle tersebut. Powerup oil kami prioritaskan terakhir karena kami merasa bahwa oil memiliki chance yang kecil untuk mengenai musuh.

10. Belok ke lane 2 atau 3 jika berada di pinggir lintasan
    Alasannya agar di round selanjutnya bisa memiliki dua pilihan belok. Selain itu, alasan kami ingin bergerak ke tengah agar memperbesar kemungkinan pemanggilan command emp untuk menyerang musuh di saat kami tertinggal.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi algoritma *greedy*

Implementasi algoritma *greedy* pada program kami terdapat pada file bot.java. Fungsi utama dari program kami adalah run yang mengembalikan command yang akan dijalankan di permainan.

4.1.1 Public Command run

```
Function run(Gamestate gameState) -> Command
{ Mengembalikan command yang akan dieksekusi pada suatu ronde di
permainan }

Kamus Lokal
myCar, opponent : Car
blocksInFront, blocksInFrontLeft, blocksInFrontRight, blocksInBack,
blocksIfAccelerate, blocksIfBoost, blocksIfNoAccelerate,
finalLizardBlock, blocksIfLeft, blocksIfRight : List of Lane
defaultFinalSpeed, leftFinalSpeed, centerFinalSpeed,
accelerateFinalSpeed, rightFinalSpeed, lizardFinalSpeed
: int
powerUpsPrioPointsLeft, powerUpsPrioPointsCenter, powerUpsPrioPointsRight
: float
finalSpeedEqual : boolean

Algoritma
{ Setup blocks dan value yang akan dicek }

myCar <- gameState.player
opponent <- gameState.opponent

defaultFinalSpeed <- getDefaultFinalSpeed(myCar)

blocksInFront <- getBlocksInFront(myCar.position.lane,
myCar.position.block, gameState)
blocksInFrontLeft <- getBlocksSide(myCar.position.lane,
myCar.position.block, gameState, LEFT)
blocksInFrontRight <- getBlocksSide(myCar.position.lane,
myCar.position.block, gameState, RIGHT)
blocksInBack <- getBlocksBack(myCar.position.lane, myCar.position.block,
gameState)
```

```
blocksIfAccelerate <- blocksInFront.subList(0,
min(speedIfAccelerate(defaultFinalSpeed, myCar.damage),
blocksInFront.size()))
blocksIfBoost <- blocksInFront.subList(0, min(15, blocksInFront.size()))
blocksIfNoAccelerate <- blocksInFront.subList(0, min(defaultFinalSpeed,
blocksInFront.size()))
finalLizardBlock <- blocksInFront.subList(max(0,min(defaultFinalSpeed -
1, blocksInFront.size())), min(defaultFinalSpeed, blocksInFront.size()))
blocksIfLeft <- emptyList()
blocksIfRight <- emptyList()

IF (myCar.position.lane != 1) THEN
      blocksIfLeft <- blocksInFrontLeft.subList(0, min(defaultFinalSpeed,
blocksInFrontLeft.size() - 1))

IF (myCar.position.lane != 4) THEN
      blocksIfRight <- blocksInFrontRight.subList(0,
      min(defaultFinalSpeed, blocksInFrontRight.size() - 1));

leftFinalSpeed <- finalSpeedIfCollide(blocksIfLeft, myCar, false)
centerFinalSpeed <- finalSpeedIfCollide(blocksIfNoAccelerate, myCar,
false)
accelerateFinalSpeed <- finalSpeedIfCollide(blocksIfAccelerate, myCar,
true)
rightFinalSpeed <- finalSpeedIfCollide(blocksIfRight, myCar, false)
lizardFinalSpeed <- finalSpeedIfcollide(finalLizardBlock, myCar, false)

powerUpsPrioPointsLeft <- countPowerUpsPrioPoints(blocksIfLeft)
powerUpsPrioPointsCenter <- finalSpeedIfCollide(blocksIfNoAccelerate,
myCar)
powerUpsPrioPointsRight <- countPowerUpsPrioPoints(blocksIfRight)

finalSpeedEqual <- false

{ STRATEGI BOT }

{ Fix jika damage lebih dari 1 }
IF (myCar.damage > 1) THEN
      -> FIX

{ Accelerate jika speed mobil 0 }
IF (myCar.speed = 0) THEN
      -> ACCELERATE

{ GREEDY OBSTACLE AVOIDANCE }
{ Dijalankan ketika ada obstacle atau mobil lawan yang akan ditabrak
pemain jika tidak berbelok dan tidak menggunakan accelerate}
```

```
{ Prioritas belok, menggunakan lizard, mencari jalur dengan speed akhir
terbesar, mencari jalur dengan point prioritas power up terbesar, ke lane
tengah }

IF (isObstaclePresent(blocksIfNoAccelerate)
OR isCollisionWithOpponentPossible(myCar, opponent, myCar.speed)) THEN
      { Belok kiri jika tidak ada obstacle yang menghalangi di kiri, ada
      obstacle di kanan dan mobil tidak di lane 1 }
      IF (!isObstaclePresent(blocksIfLeft)
      AND isObstaclePresent(blocksIfRight)
      AND myCar.position.lane != 1) THEN
            -> TURN_LEFT


      { Belok kanan jika tidak ada obstacle yang menghalangi di kanan,
      ada obstacle di kiri, dan mobil tidak di lane 4 }
      IF (isObstaclePresent(blocksIfLeft)
      AND !isObstaclePresent(blocksIfRight)
      AND myCar.position.lane != 1) THEN
            -> TURN_LEFT

      { Jika kedua lane tidak ada obstacle, ke lane yang power up nya
      lebih banyak, lalu ke tengah }
      IF (!isObstaclePresent(blocksIfLeft)
      AND !isObstaclePresent(blocksIfRight)) THEN
            IF (myCar.position.lane = 1) THEN
                  -> TURN_RIGHT
            IF (myCar.position.lane = 4) THEN
                  -> TURN_LEFT
            IF (powerUpsPrioPointsLeft > powerUpsPrioPointsRight) THEN
                  -> TURN_LEFT
            IF (powerUpsPrioPointsLeft < powerUpsPrioPointsRight) THEN
                  -> TURN_RIGHT
            { powerUpsPrioPoints kiri dan kanan sama }
            { Prioritaskan ke lane tengah }
            IF (myCar.position.lane = 2) THEN
                  -> TURN_RIGHT
            IF (myCar.position.lane = 3) THEN
                  -> TURN_LEFT


      { Ada obstacle di semua lane }
      IF ((isObstaclePresent(blocksIfLeft)
      AND isObstaclePresent(blockIfLeft))
      OR (myCar.position.lane == 1 AND isObstaclePresent(blocksIfRight))
      OR (myCar.position.lane == 4 AND isObstaclePresent(blocksIfLeft)))
      THEN
            { Gunakan lizard jika punya dan speed akhir jika dipakai
            lebih besar dari tidak dipakai atau belok }
```

```
IF (hasPowerUp(PowerUps.LIZARD, myCar.powerups)
AND lizardFinalSpeed > centerFinalSpeed
AND (lizardFinalSpeed > leftFinalSpeed
OR myCar.position.lane = 1)
AND (lizardFinalSpeed > rightFinalSpeed
OR myCar.position.lane = 4)) THEN
      -> LIZARD

{ Jika obstacle tidak bisa dihindari, cari yang final
speednya lebih besar}
{ Kalau harus belok dan final speednya sama, lihat dari point
prioritas powerupnya }

IF (myCar.position.lane = 1) THEN
      IF (centerFinalSpeed < rightFinalSpeed) THEN
            -> TURN_RIGHT
      IF (centerFinalSpeed = rightFinalSpeed) THEN
            finalSpeedEqual <- true
ELSE IF (myCar.position.lane = 4) THEN
      IF (centerFinalSpeed < leftFinalSpeed) THEN
            -> TURN_LEFT
      IF (centerFinalSpeed = leftFinalSpeed) THEN
            finalSpeedEqual <- true
ELSE
      IF (leftFinalSpeed > centerFinalSpeed
      AND leftFinalSpeed > rightFinalSpeed) THEN
            -> TURN_LEFT
      IF (rightFinalSpeed > centerFinalSpeed
      AND rightFinalSpeed > leftFinalSpeed) THEN
            -> TURN_RIGHT
      IF (leftFinalSpeed = centerFinalSpeed
      AND centerFinalSpeed > rightFinalSpeed) THEN
            IF (powerUpsPrioPointsLeft >
            powerUpsPrioPointsCenter) THEN
                  -> TURN_LEFT
      IF (rightFinalSpeed = centerFinalSpeed
      AND centerFinalSpeed > leftFinalSpeed) THEN
            IF (powerUpsPrioPointsRight >
            powerUpsPrioPointsCenter) THEN
                  -> TURN_RIGHT
      IF (leftFinalSpeed = rightFinalSpeed
      AND leftFinalSpeed > centerFinalSpeed) THEN
            IF (powerUpsPrioPointsLeft >
            powerUpsPrioPointsRight) THEN
                  -> TURN_LEFT;
            IF (powerUpsPrioPointsLeft <
            powerUpsPrioPointsRight) THEN
                  -> TURN_RIGHT
            IF (myCar.position.lane = 2) THEN
```

```
                        -> TURN_RIGHT
                  IF (myCar.position.lane = 3) THEN
                        -> TURN_LEFT
            IF (leftFinalSpeed = centerFinalSpeed
            AND centerFinalSpeed = rightFinalSpeed) THEN
                  finalSpeedEqual <- true


{ GREEDY AMBIL POWER UP }
{ Tidak ada obstacle di tengah atau ada obstacle di semua arah dan final
speed sama }
{ Prioritas ke jalur dengan point prioritas power up tertinggi, tetap di
lane yang sama, ke lane tengah }
IF (myCar.position.lane = 1) THEN
      IF (!isObstaclePresent(blocksIfRight) OR finalSpeedEqual) THEN
            IF (powerUpsPrioPointsCenter < powerUpsPrioPointsRight) THEN
                  -> TURN_RIGHT
ELSE IF (myCar.position.lane = 4) THEN
      IF (!isObstaclePresent(blocksIfLeft) OR finalSpeedEqual) THEN
            IF (powerUpsPrioPointsCenter < powerUpsPrioPointsLeft) THEN
                  -> TURN_LEFT
ELSE IF (!isObstaclePresent(blockIfLeft)
AND isObstaclePresent(blocksIfRight)) THEN
      IF (powerUpsPrioPointsCenter < poweUpsPrioPointsLeft) THEN
            -> TURN_LEFT
ELSE IF (isObstaclePresent(blocksIfLeft)
AND !isObstaclePresent(blocksIfRight)) THEN
      IF (powerUpsPrioPointsCenter < powerUpsPrioPointsRight) THEN
            -> TURN_RIGHT
ELSE IF ((!isObstaclePresent(blocksIfLeft)
AND !isObstaclePresent(blocksIfRight))
OR finalSpeedEqual)) THEN
      IF (powerUpsPrioPointsLeft > powerupsPrioPointsCenter
      AND powerUpsPrioPointsLeft > powerUpsPrioPointsRight) THEN
            -> TURN_LEFT
      ELSE IF (powerUpsPrioPointsRight > powerUpsPrioPointsLeft
      AND powerUpsPrioPointsRight > PowerUpsPrioPointsCenter) THEN
            -> TURN_RIGHT
      ELSE IF (powerUpsPrioPointsRight = powerUpsPrioPointsLeft
      AND powerUpsPrioPointsRIght > powerUpsPrioPointsCenter) THEN
            IF (myCar.position.lane = 2) THEN
                  -> TURN_RIGHT
            IF (myCar.position.lane = 3) THEN
                  -> TURN_LEFT

{ Obstacle Placement (EMP dan TWEET) }
{ Gunakan EMP jika punya EMP dan lawan ada di range EMP }
IF (hasPowerUp(PowerUps.EMP, myCar.powerups)
AND isInEmpRange(myCar.position, opponent.position)) THEN
```

```
      -> EMP

{ Gunakan TWEET di depan lawan jika punya TWEET }
IF (hasPowerUp(PowerUps.TWEET, myCar.powerups)) THEN
      -> new TweetCommand(opponent.position.lane, opponent.position.block
      + speedIfAccelerate(opponent.speed, opponent.damage) + 1)

{ BOOST }
{ Gunakan BOOST jika tidak ada obstacle yang menghalangi, punya BOOST,
dan tidak sedang memakai BOOST }
IF (!isObstaclePresent(blocksIfBoost)
&& hasPowerUp(PowerUps.BOOST, myCar.powerups)
&& myCar.boostCounter <= 1) THEN
      { Persiapan penggunaan BOOST jika ada damage}
      IF (myCar.damage = 1) THEN
            -> FIX
      { BOOST digunakan saat tidak ada yang menghalangi dan damage 0 }
      IF (!isCollisionWithOpponentPossible(myCar, opponent,
      maxSpeed(myCar.damage))) THEN
            -> BOOST

{ ACCELERATE }
{ Gunakan ACCELERATE jika speed akan bertambah, tidak ada obstacle, dan
tidak sedang menggunakan BOOST }
IF (speedIfAccelerate(defaultFinalSpeed, myCar.damage) >
defaultFinalSpeed
AND !isObstaclePresent(blocksIfAccelerate)
AND !myCar.boosting) THEN
      -> ACCELERATE

{ OIL }
{ Gunakan OIL jika punya power up OIL, lawan di belakang, dan tidak ada
obstacle di belakang }
IF (hasPowerUp(PowerUps.OIL, myCar.powerups
AND myCar.position.bloack > opponent.position.block
AND (!isObstaclePresent(blocksInBack)
OR myCar.position.lane = opponent.position.lane)) THEN
      -> OIL

{ Coba ke lane tengah kalau tidak ada obstacle dan power up points sama
atau lebih baik }
IF (myCar.position.lane = 1 AND !isObstaclePresent(blocksIfRight) AND
powerUpsPrioPointsRight >= powerUpsPrioPointsCenter) THEN
      -> TURN_RIGHT
IF (myCar.position.lane = 4 AND !isObstaclePresent(blocksIfLeft) AND
powerUpsPrioPointsLeft >= powerUpsPrioPointsCenter) THEN
      -> TURN_LEFT

-> NOTHING
```

4.1.2 Private Boolean hasPowerUp

```
Function hasPowerUp(PowerUps powerUpToCheck, PowerUps[] available) ->
boolean
{ Mengembalikan true jika player memiliki powerup sesuai dengan
powerUpToCheck }

Kamus Lokal

Algoritma
FOR (each PowerUps in available as powerUp)
      IF (powerUp.equals(powerUpToCheck)) THEN
            -> true
-> false
```

4.1.3 Private List<Lane> getBlocksInFront

```
Function getBlocksInFront(int lane, int block, GameState gameState) ->
List of Lane
{ Mengembalikan block di depan mobil yang dapat terlihat di map }

Kamus Lokal
map : List of array of Lane
blocks : List of Lane
startBlock : integer
laneList : array of Lane

Algoritma
map <- gameState.lanes
startBlock <- map.get(0)[0].position.block
laneList <- map.get(lane - 1)

FOR (i = max(block - startBlock + 1, 0) to block - startBlock + 20)
      IF (laneList[i] = null OR laneList[i].terrain = Terrain.FINISH)
      THEN
            break
      blocks.add(laneList[i])
-> blocks
```

4.1.4 Private List<Lane> getBlocksSide

```
Function getBlocksSide(int lane, int block, GameState gameState, int
direction) -> List of Lane
{Mengembalikan block di samping kiri / kanan mobil yang terlihat di map}

Kamus Lokal
map : List of array of Lane
blocks : List of Lane
startBlock : integer
laneList : array of Lane
```

Algoritma

```
IF (lane + direction > 4 OR lane + direction < 1) THEN
      -> emptyList()

map <- gameState.lanes
startBlock <- map.get(0)[0].position.block
laneList <- map.get(lane - 1 + direction)

FOR (i = max(block - startBlock, 0) to block - startBlock + 20)
IF (laneList[i] = null OR laneList[i].terrain = Terrain.FINISH) THEN
            break
      blocks.add(laneList[i])
-> blocks
```

4.1.5 Private List<Lane> getBlocksBack

Function getBlocksBack(int lane, int block, GameState gameState) -> List of Lane
{ Mengembalikan block di belakang mobil yang terlihat di map }

Kamus Lokal

```
map : List of array of Lane
blocks : List of Lane
startBlock : integer
laneList : array of Lane
```

Algoritma

```
map <- gameState.lanes
startBlock <- map.get(0)[0].position.block
laneList <- map.get(lane - 1)

FOR (i = max(block - startBlock - 5, 0) to block - startBlock)
      IF (laneList[i] = null) THEN
            break
      blocks.add(laneList[i])
-> blocks
```

4.1.6 Private int speedIfAccelerate

Function speedIfAccelerate(int speed, int damage) -> integer
{ Mengembalikan speed mobil jika diberikan command ACCELERATE }

Kamus Lokal

```
Final_speed : integer
```

Algoritma

```
CASE (speed) OF
      0: final_speed <- 3
      3 OR 5: final_speed <- 6
      6: final_speed <- 8
```

```
      8 OR 9: final_speed <- 9
      15: final_speed <- 15
      Default: final_speed <- 0
-> min(final_speed, maxSpeed(damage))
```

4.1.7 Private int maxSpeed

```
Function maxSpeed(int damage) -> integer
{ Mengembalikan kecepatan maksimum berdasarkan damage mobil }

Kamus Lokal

Algoritma
CASE (damage) OF
      4: -> 3
      3: -> 6
      2: -> 8
      1: -> 9
      0: -> 15
      Default: -> 0
```

4.1.8 Private int speedIfDecelerate

```
Function speedIfDecelerate(int speed) -> integer
{ Mengembalikan speed mobil jika diberikan command DECELERATE }

Kamus Lokal

Algoritma
CASE (speed) OF
      5 OR 6: -> 3
      8: -> 6
      9: -> 8
      15: -> 9
      Default: -> 0
```

4.1.9 Private Boolean isInEmpRange

```
Function isInEmpRange(Position myCarPosition, Position opponentPosition)
-> boolean
{ Mengembalikan true jika mobil lawan berada di posisi yang akan terkena
EMP }

Kamus Lokal

Algoritma
IF (myCarPosition.block < opponentPosition.block) THEN
      CASE (myCarPosition.lane) OF
            1:
```

```
            IF (opponentPosition.lane = 1 OR opponentPosition.lane = 2)
            THEN
                  -> true
            2:
            IF (opponentPosition.lane = 1 OR opponentPosition.lane = 2
            OR opponentPosition.lane = 3) THEN
                  -> true
            3:
            IF (opponentPosition.lane = 2 OR opponentPosition.lane = 3
            OR opponentPosition.lane = 4) THEN
                  -> true
            4:
            IF (opponentPosition.lane = 3 OR opponentPosition.lane = 4)
            THEN
                  -> true
            Default: -> false
-> false
```

4.1.10 Private Boolean isObstaclePresent

```
Function isObstaclePresent(List<Lane> blocks) -> boolean
{ Mengembalikan true jika ada obstacle di dalam list blocks }

Kamus Lokal
terrain: Terrain

Algoritma
FOR (each Lane in blocks as block)
      terrain <- block.terrain
      IF (terrain = Terrain.WALL OR terrain = Terrain.MUD OR terrain =
      Terrain.OIL_SPILL OR block.isOccupiedByCyberTruck) THEN
            -> true
-> false
```

4.1.11 Private int finalSpeedIfCollide

```
Function finalSpeedIfCollide(List<Lane> blocks, Car myCar, boolean
isAccelerating) -> integer
{ Mengembalikan perhitungan kecepatan akhir jika melewati suatu lane }

Kamus Lokal
damage, speed_reduction. Final_speed: integer
terrain: Terrain

Algoritma
damage <- 0
speed_reduction <- 0
final_speed <- -1

{ Hitung total damage }
```

```
FOR (each Lane in blocks as block)
      terrain <- block.terrain
      IF (block.isOccupiedByCyberTruck) THEN
            -> 0
      IF (terrain = Terrain.Wall) THEN
            damage <- damage + 2
      ELSE IF (terrain = Terrain.MUD OR terrain = Terrain.OIL_SPILL) THEN
            damage <- damage + 1

{ Hitung total pengurangan speed }
FOR (each Lane in blocks as block)
      terrain <- block.terrain
      IF (terrain = Terrain.Wall) THEN
            Final_speed <- 3
      ELSE IF ((terrain = Terrain.MUD OR terrain = Terrain.OIL_SPILL) AND
      final_speed = -1) THEN
            speed_reduction <- speed_reduction + 1

{ Hitung kecepatan akhir }
IF (final_speed = -1) THEN
      IF (isAccelerating) THEN
            final_speed = speedIfAccelerate(getDefaultFinalSpeed(myCar),
            myCar.damage)
      ELSE
            final_speed = getDefaultFinalSpeed(myCar)

      FOR (i = 0 to i = speed_reduction - 1)
            final_speed = speedIfDecelerate(final_speed)
      final_speed <- max(3, final_speed)

-> min(final_speed, maxSpeed(myCar.damage + damage))
```

4.1.12 Private float countPowerUpsPrioPoints

```
Function countPowerUpsPrioPoints(List<Lane> blocks) -> float
{ Mengembalikan point prioritas powerup di dalam list blocks }

Kamus Lokal
powerUpsPrioPoints: float

Algoritma
powerUpsPrioPoints <- 0
FOR (each Lane in blocks as block)
      terrain <- block.terrain
      CASE (terrain) OF
            LIZARD: powerUpsPrioPoints <- powerUpsPrioPoints + 2
            BOOST: powerUpsPrioPoints <- powerUpsPrioPoints + 1.75
            EMP: powerUpsPrioPoints <- powerUpsPrioPoints + 1.5
            TWEET: powerUpsPrioPoints <- powerUpsPrioPoints + 1.25
```

```
            OIL_POWER: powerUpsPrioPoints <- powerUpsPrioPoints + 1
-> powerUpsPrioPoints
```

4.1.13 Private Boolean isCollisionWithOpponentPossible

```
Function isCollistionWithOpponentPossible(Car myCar, Car opponent, int
speed) -> boolean
{ Mengembalikan true jika ada kemungkinan terjadi tabrakan dengan mobil
lawan }
{ Diasumsikan ada kemungkinan terjadi tabrakan jika ada di lane yang
sama, mobil pemain di belakang lawan dan block akhir pemain berada di
depan block akhir lawan (jika tidak ada tabrakan)}

Kamus Lokal

Algoritma
-> myCar.position.lane = opponent.position.lane
AND myCar.position.block < opponent.position.block
AND myCar.position.block + speed >= opponent.position.block +
opponent.speed
```

4.1.14 Private int getDefaultFinalSpeed(Car myCar)

```
Function getDefaultFinalSpeed(Car myCar) -> integer
{ Mengembalikan speed mobil untuk ronde ini jika tidak terjadi tabrakan
dan tidak diberikan command boost atau accelerate }

Kamus Lokal

Algoritma
IF (myCar.speed = 15 AND myCar.booostCounter <= 1 AND myCar.boosting)
THEN
      -> 9
ELSE
      -> myCar.speed
```

4.2 Penjelasan struktur data

Pada permainan “Overdrive”, struktur data yang digunakan berbasis kelas. Kami memanfaatkan kelas-kelas yang sudah disediakan di *starter kit* dan tidak mengubahnya kecuali di bagian Bot. Secara umum, kelas yang digunakan dalam implementasi bot ini dibagi menjadi 5 bagian, yaitu *command, entities, enums, bot,* dan *main*.

4.2.1 Bagian *command*

Kelas-kelas pada bagian ini digunakan untuk mengembalikan string command ke *game engine*.

1. AccelerateCommand

   Kelas untuk menghasilkan *new command* yang mengakibatkan meningkatnya kecepatan mobil pemain.
2. BoostCommand

   Kelas untuk menghasilkan *new command* yang mengaktifkan *power up* Boost pemain.
3. ChangeLaneCommand

   Kelas untuk menghasilkan *new command* yang mengakibatkan pemain berbelok ke kiri atau ke kanan tergantung string yang diberikan (“RIGHT” atau “LEFT”).
4. DecelerateCommand

   Kelas untuk menghasilkan *new command* yang mengakibatkan menurunnya kecepatan mobil pemain.
5. DoNothingCommand

   Kelas untuk menghasilkan *new command* yang mengakibatkan pemain tidak memberikan *command* apapun pada suatu ronde permainan.
6. EmpCommand

   Kelas untuk menghasilkan *new command* yang mengaktifkan *power up* EMP pemain.
7. FixCommand

   Kelas untuk menghasilkan *new command* yang menghentikan gerakan mobil pemain dan mengurangi *damage* pada mobil.
8. LizardCommand

   Kelas untuk menghasilkan *new command* yang mengaktifkan *power up* Lizard pemain.
9. OilCommand

   Kelas untuk menghasilkan *new command* yang mengaktifkan *power up* Oil pemain dan memunculkan *oil spill* di belakangnya.
10. TweetCommand

    Kelas untuk menghasilkan *new command* yang mengaktifkan *power up* Tweet dan memunculkan *cybertruck* pada koordinat yang dispesifikasikan di *command*.

4.2.2 Bagian *entities*

Kelas-kelas pada bagian ini digunakan untuk menyatakan entitas yang ada di permainan.

1. Car

   Kelas untuk menyimpan atribut dan kondisi mobil pemain ataupun mobil lawan. Atribut yang disimpan meliputi id mobil, posisi, kecepatan, damage, state, Array power up yang dimiliki, mobil sedang boosting atau tidak, dan *counter* boost.
2. GameState

   Kelas untuk menyimpan kondisi permainan pada suatu ronde. Atribut yang disimpan meliputi ronde saat ini, ronde maksimum, mobil pemain, mobil lawan, dan peta permainan yang dinyatakan dalam List of Array of Lane.
3. Lane

   Kelas untuk menyimpan atribut suatu blok di koordinat tertentu. Atribut yang disimpan meliputi koordinat posisi, *obstacle* atau *power up* yang ada di blok, id player yang berada di suatu blok (jika ada), dan boolean yang menyatakan ada atau tidaknya *cybertruck* di blok tersebut.
4. Position

   Kelas untuk menyimpan angka koordinat suatu lokasi dalam axis X dan Y.

4.2.3 Bagian *enums*

Kelas-kelas pada bagian ini digunakan untuk menyatakan properti entitas dengan tipe data non primitif.

1. Direction

   Kelas untuk menyimpan enumerasi arah meliputi “FORWARD”, “BACKWARD”, “LEFT”, dan “RIGHT”.
2. PowerUps

   Kelas untuk menyimpan enumerasi *power up* meliputi “BOOST”, “OIL”, “TWEET”, “LIZARD”, dan “EMP”.
3. State

   Kelas untuk menyimpan enumerasi *state* yang bisa dialami suatu mobil antara lain “ACCELERATING”, “READY”, “NOTHING”, TURNING_RIGHT”, “TURNING_LEFT”, “HIT_MUD”, “HIT_OIL”, “DECELERATING”, “PICKED_UP_POWERUP”, “USED_BOOST”, “USED_OIL”, “USED_LIZARD”, “USED_TWEET”, “HIT_WALL”, “HIT_CYBER_TRUCK”, dan “FINISHED”.
4. Terrain

   Kelas untuk menyimpan enumerasi jenis *obstacle* atau *power up* yang ada pada suatu blok. *Obstacle* dan *power up* dinyatakan dengan angka 0-9 yang secara

berturut-turut merepresentasikan blok kosong, *mud, oil spill, oil power, finish, boost, wall, lizard, tweet,* dan *emp*.

4.2.4 Kelas Bot

Kelas Bot berisi implementasi strategi Bot dengan algoritma Greedy. Struktur data yang digunakan dalam kelas ini adalah List, tipe data primitif, serta kelas-kelas yang ada pada bagian *command, entities, dan enums*.

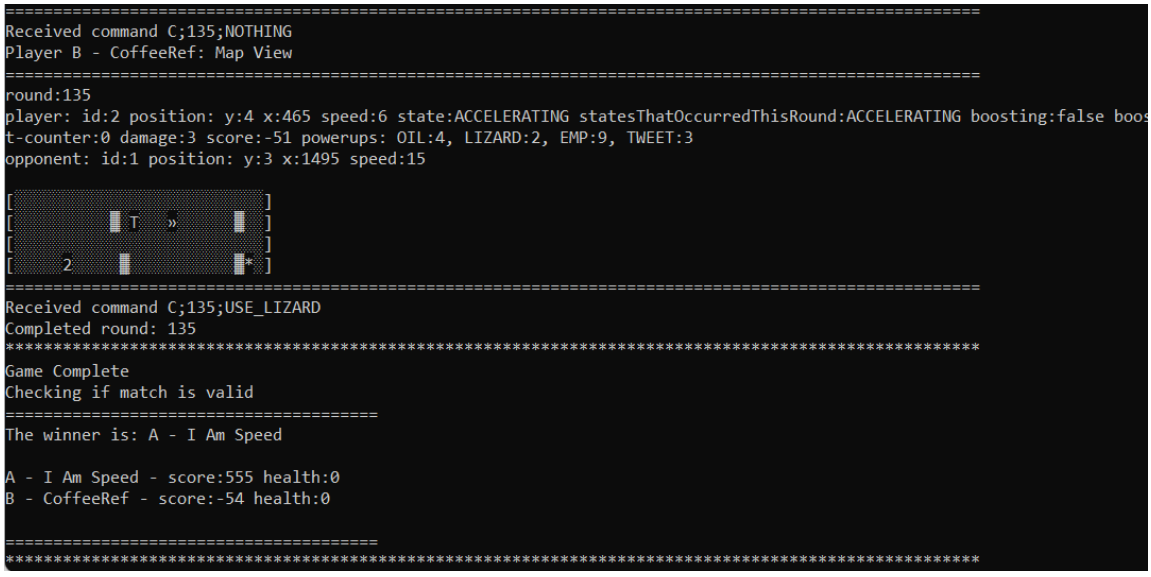
4.2.5 Kelas Main

Kelas Main penghubung bot dengan *game engine* sehingga bot dan *game engine* bisa bertukar informasi. Game engine memberikan kondisi permainan pada suatu ronde dan bot mengembalikan *command* yang ingin dilakukan pada ronde tersebut.

4.3 Analisis dari desain solusi algoritma *greedy*

Percobaan untuk menganalisis solusi algoritma *Greedy* yang kami buat dilakukan dengan menjalankan run.bat. Terdapat tiga kali pengujian yang kami lakukan, yaitu melawan *bot-reference* dan dua bot yang batal menjadi *bot* utama kami.

a. Pengujian I

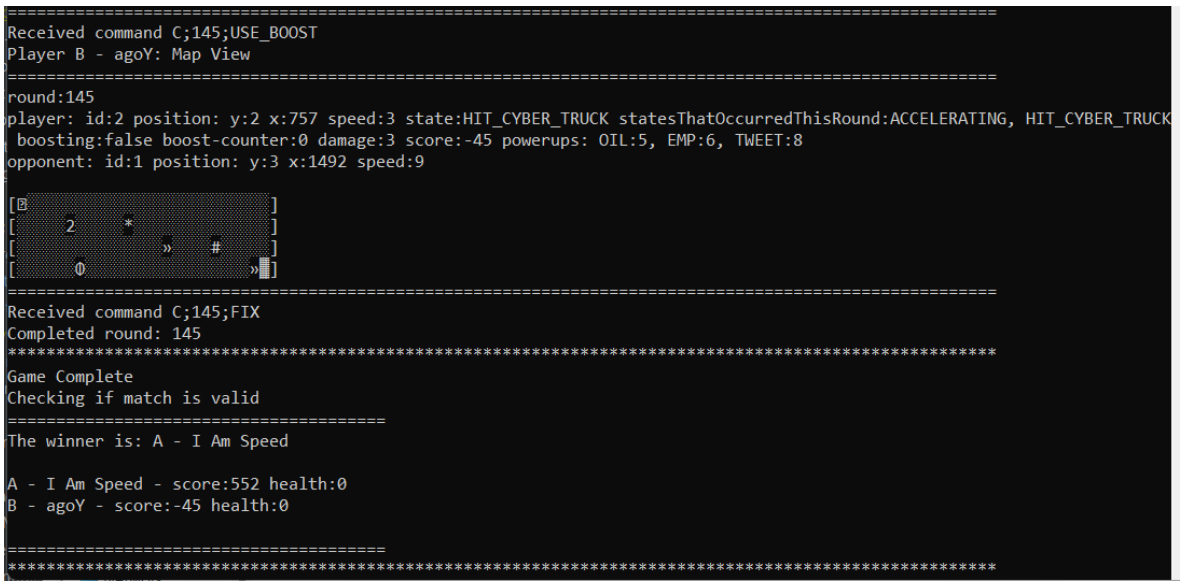


Gambar 4.1 Pengujian I

Percobaan ini dilakukan dengan melawan *reference-bot* yang sudah disiapkan oleh *Entelect*. Total pertandingan ini berjalan selama 135 ronde dengan perbedaan blok kurang lebih 1000 blok dan dimenangkan oleh *bot* kami yang bernama I Am Speed. Jika dilihat dari segi *code*, *reference-bot* sama sekali tidak menggunakan algoritma *Greedy*

bahkan hanya membuat beberapa perintah dasar untuk menghindari *obstacle* dan mempercepat kecepatan. Berbeda dengan bot kami yang sudah menggunakan algoritma *greedy,* baik untuk menyerang lawan maupun menghindari *obstacle*. Oleh karena itu, banyak sekali dapat dilihat di setiap rondenya *reference-bot* mendapat kerugian dan pada akhirnya kalah di pertandingan ini.

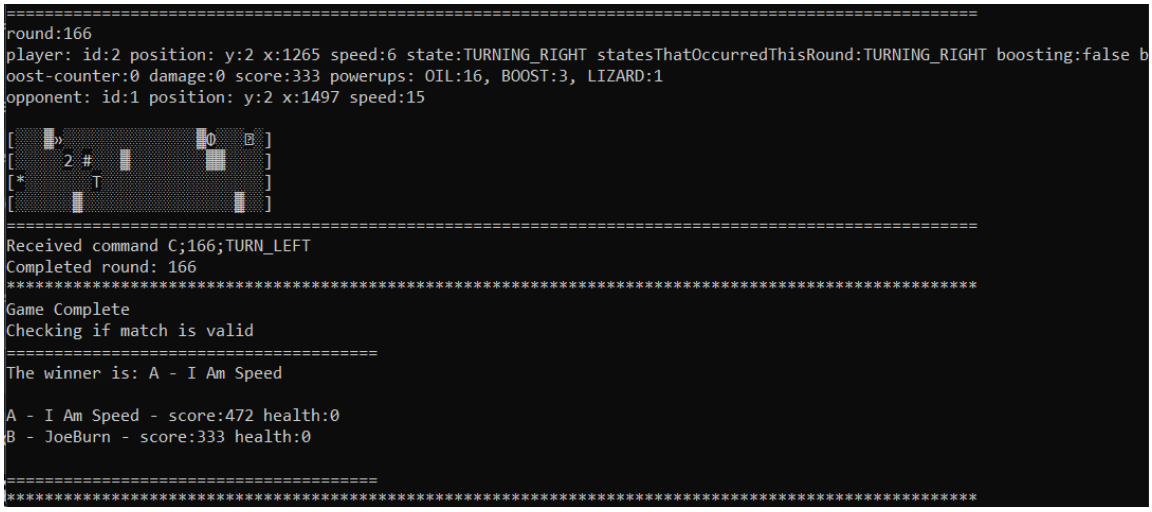
b. Pengujian II



Gambar 4.2 Pengujian II

Pada pertandingan kedua ini, *bot* kami melawan salah satu *bot* lain yang tidak jadi menjadi *bot* utama kami. Pada pertandingan ini *bot* kami menang dalam 145 ronde dengan perbedaan blok lebih dari 700 blok. *Bot* lawan sudah memiliki code yang lebih baik daripada *code* yang dimiliki *reference-bot*. Namun, pengambilan keputusan untuk menggunakan *command* pada setiap rondenya masih kurang menerapkan strategi *Greedy*. Jika pada *bot* kami pergerakan belok kiri atau kanan menggunakan strategi dengan membuat poin prioritas untuk *powerups* yang ada sehingga bisa mengambil *powerups* yang menurut kami paling efektif dan dengan kuantitas yang lebih banyak. Berbeda dengan bot lawan yang pergerakan kanan kiri hanya berdasarkan obstacle yang menghalanginya. Kemudian, banyak *command* yang bertujuan menyerang dari bot lawan yang tidak terpakai karena salahnya penempatan urutan prioritas *command* pada *code*-nya. Hal ini berakibat tidak banyak serangan yang diterima oleh *bot* kami di setiap rondenya.

c. Pengujian III



Gambar 4.3 Pengujian III

Pada pertandingan ini, *bot* kami melawan salah satu *bot* lainnya yang tidak jadi menjadi bot utama kami. *Bot* lawan ini sebenarnya sudah hampir sama penerapan strategi greedynya, tetapi mungkin ada beberapa detail yang kurang di sini. Pada pertandingan ini *bot* kami menang dalam 166 ronde dan berbeda kurang lebih 250 blok dengan lawan. Pada awal pertandingan, jika dilihat *map* di setiap rondenya, *bot* kami dan *bot* lawan sebenarnya berjarak tidak terlalu jauh, tetapi di pertengahan dan akhir, *bot* kami dapat memperlebar jarak. Beberapa perbedaan yang membuat *bot* kami menang adalah adanya strategi pergerakan yang memperhitungkan prioritas *powerups* yang berada disekitarnya sehingga memperoleh *powerups* yang menurut kami lebih penting dan dengan kuantitas yang lebih juga. Selain itu, ada bot kami juga akan memprioritaskan *lane* tengah (*lane* 2 atau 3) jika memang tidak ada *obstacle* yang menghalangi karena berada di *lane* tengah membuat bot memiliki lebih banyak pilihan pergerakan, baik untuk menghindari *obstacle* maupun mencari *powerups*. Perbedaan dengan *bot* lawan sebelumnya, *bot* lawan pada pertandingan ini sudah bisa menyerang *bot* kami dengan cukup baik sehingga memiliki perbedaan blok akhir yang lebih sedikit.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kami berhasil mengimplementasikan algoritma *Greedy* dalam pembuatan *bot*-mobil untuk mencapai objektif di permainan *Overdrive*. Adapun objektif pada permainan ini adalah mencapai blok *finish* secepat-cepatnya dibanding blok lawan, baik dengan melakukan pergerakan yang efektif maupun menyerang lawan sehingga lawan memiliki pergerakan yang tidak efektif. Beberapa pengimplemetasian algotritma *Greedy* yang kami lakukan antara lain, Pemilihan *command* pergerakan yang tepat sehingga berhasil menghindari obstacle yang ada dan mendapatkan *powerups* yang cukup banyak serta beberapa strategi penyerangan.

Walaupun begitu, pada tugas besar kali ini kami menyadari bahwa algoritma *Greedy* yang kami buat bukan merupakan solusi yang paling optimal, setidaknya menurut kelompok kami. Hal ini disebabkan masih ada beberapa strategi *Greedy* yang mungkin lebih optimal dan sempat kami pikirkan, tetapi karena keterbatasan waktu dan lain hal belum bisa kami implementasikan. Selain itu, strategi yang dibuat biasanya masih terpengaruh dengan *map* yang digunakan karena *map* yang selalu berubah-ubah di setiap pertandingan. Hal ini mengakibatkan jika melawan *bot* yang sama-sama handal, kehokian juga merupakan salah satu bagian yang berpengaruh dari pertandingan ini.

5.2 Saran

Pengimpelementasian algoritma *Greedy* pada permainan Overdrive ini masih jauh untuk dikatakan strategi yang paling optimal. Saran kami pada tugas kali ini adalah program ini dapat dikembangkan lebih jauh dengan strategi-strategi *Greedy* lain yang lebih efektif dan efisien serta mungkin dengan menggabungkannya dengan algoritma lain. Selain itu, bisa juga dikembangkan bot-mobil yang menggunakan kecerdasan buatan (Artificial Intelligence) sehingga bot dapat berkembang dari setiap pertandingan yang pernah dilakukannya dan menghasilkan strategi yang jauh lebih baik.

DAFTAR PUSTAKA

1. https://github.com/EntelectChallenge/2020-Overdrive
2. https://www.jetbrains.com/help/idea/compiling-applications.html#package_into_jar
3. https://stackoverflow.com/questions/30204884/creating-a-jar-from-a-maven-project-in-intellij/30210471
4. https://entelect-replay.raezor.co.za/
5. https://brilliant.org/wiki/greedy-algorithm/
6. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

Link GitHub → https://github.com/johannes-ws/Tubes-1-Stima

Link YouTube → https://youtu.be/cThYj4OgZ1E