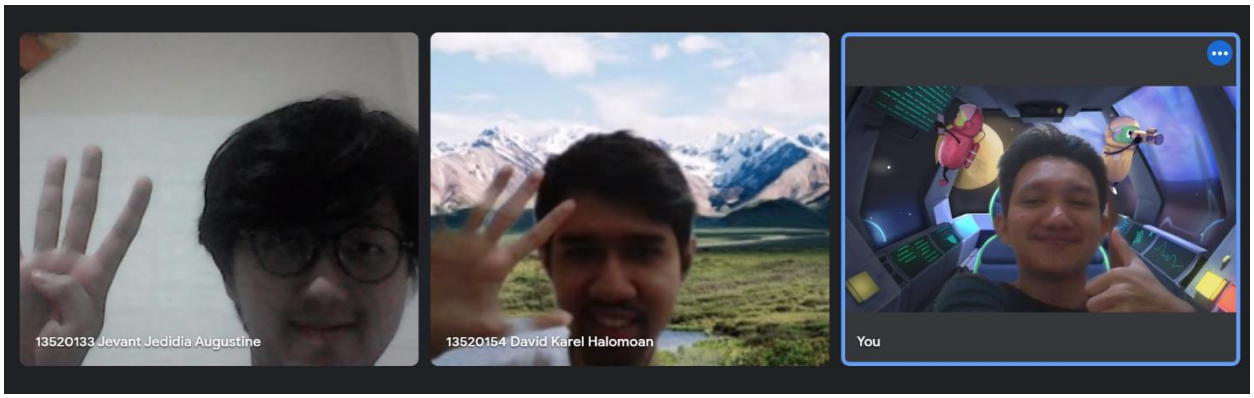
Tugas Besar 1 IF2211 Strategi Algoritma

Semester II Tahun 2021/2022

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan ”Overdrive”



Dipersiapkan oleh:

Kelompok 45 (Lightning MisQueen)

Nama	NIM
Taufan Fajarama Putrawansyah R	13520031
Jevant Jedidia Augustine	13520133
David Karel Halomoan	13520154

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB 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.

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/gamerules.md

BAB 2: LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Hanya ada dua macam persoalan optimasi, yaitu maksimasi dan minimasi. Secara umum, algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah akan diambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa melihat konsekuensi ke depan dan berharap dengan mengambil optimum lokal, akan didapatkan optimum global.

Algoritma *greedy* memiliki beberapa elemen, yaitu:

1. Himpunan Kandidat (C): berisi kandidat yang akan dipilih pada setiap langkah
2. Himpunan Solusi (S): berisi kandidat yang sudah dipilih
3. Fungsi Solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi Seleksi: memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik
5. Fungsi Kelayakan: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi
6. Fungsi Objektif: memaksimumkan atau meminimumkan

2.2 Cara Kerja Program Secara Umum

Program akan melakukan pertandingan antara 2 bot mobil dalam sebuah ajang balapan. Pemenang dari duel adalah bot mobil yang mencapai garis finish lebih dahulu. Jika kedua bot mobil bersamaan sampai di garis finish, maka pemenangnya akan ditentukan dari kecepatan terakhir yang dimiliki bot mobil saat finish. Jika kecepatannya juga sama, maka pemenangnya adalah yang memiliki skor/poin lebih besar.

Program akan berjalan sesuai panjang trek yang ditentukan dan akan berjalan dalam beberapa *round* yang telah ditentukan. Pada jalur balapan terdapat beberapa *obstacle* yang menghalangi dan *powerup* yang dapat diambil untuk digunakan. Program akan mengeksekusi *command* sesuai yang terdapat dalam strategi masing-masing bot untuk setiap *round*. Setiap *command* yang dilakukan dapat menguntungkan (seperti mendapat *powerup*, menambah kecepatan, menambah skor) atau merugikan (seperti bertemu *obstacle*, kecepatan melambat, mobil terkena *damage*). Algoritma *greedy* diimplementasikan ke dalam bot mobil pada *path* ..\starter-bots\java\src\main\java\za\co\entelect\challenge\Bot.java yang terdapat di folder starter-pack dengan memanfaatkan *command* yang tersedia untuk memaksimasi keuntungan atau meminimasi kerugian pada setiap *round*.

Setelah mengimplementasikan strategi algoritma *greedy* pada bot, bot mobil kemudian di-*build* menggunakan Intellij IDEA. Bot di-*build* dengan cara klik kanan pada file `pom.xml` pada direktori yang sama lalu klik `Add as Maven Project`, arahkan kursor pada kotak di bagian bawah kiri Intellij IDEA, akan muncul daftar pilihan menu, klik `Maven`, lalu klik java-starter-bot pada pilihan di kanan layar, klik `Lifecycle`, klik compile, tunggu hingga proses selesai, lalu klik install. Lalu program dijalankan dengan menggunakan `run.bat` pada root directory.

BAB 3: APLIKASI STRATEGI GREEDY

3.1 Pemetaan Persoalan

Elemen Algoritma *Greedy*	Pemetaan pada *Overdrive*
Himpunan Kandidat (C)	Perintah-perintah atau *Command* yang dijalankan pada setiap *round*. *Command* yang dapat dijalankan adalah penggunaan *power-ups, accelerate, decelerate, turn_left, turn_right, fix,* dan *do_nothing.*
Himpunan Solusi (S)	Perintah/*command* terbaik yang dipilih berdasarkan kondisi/*state* dari setiap *round* serta strategi yang ingin diaplikasikan.
Fungsi Solusi	Memeriksa apakah *command* yang dipilih sudah memenuhi tujuan yang ingin dilakukan/dipenuhi berdasarkan *state* dari suatu *round,* seperti menghindar dari *collision*, memaksimasi kecepatan kendaraan, atau mendapatkan *power-up.*
Fungsi Seleksi	Memilih *command* yang terbaik berdasarkan kondisi/*state* dari *round* tertentu. Prioritas pemilihan *command* didasarkan strategi yang ingin diaplikasikan pada algoritma *greedy*.
Fungsi Kelayakan	Memeriksa apakah *command* yang dimasukkan merupakan *command* yang valid berdasarkan *state* dari setiap *round.* Pemeriksaan dilakukan terhadap ada atau tidaknya *power-ups,* nilai *damage* dari kendaraan, *lane* dari kendaraan, dan kecepatan dari kendaraan
Fungsi Objektif	Memenangkan permainan O*verdrive* dengan menjadi yang pertama kali menyentuh garis akhir atau memiliki kecepatan tercepat apabila kedua kendaraan menyentuh garis akhir pada waktu yang sama, atau memiliki *point* tertinggi apabila kedua kendaraan menyentuh garis akhir secara bersamaan dan memiliki kecepatan yang sama ketika menyentuh garis akhir.

3.2 Eksplorasi Alternatif Solusi

Terdapat banyak elemen yang ada pada permainan *Overdrive.* Berdasarkan elemen-elemen tersebut, terdapat beberapa alternatif solusi yang dapat diterapkan pada permainan *Overdrive*.

a. Utamakan *power-ups*

Poin merupakan salah satu faktor yang diperhitungkan untuk memenangkan permainan *Overdrive*. Apabila pemain dan lawan sampai pada garis akhir pada waktu yang bersamaan dan memiliki kecepatan yang sama, kendaraan yang memiliki poin tertinggi akan memenangkan pertandingan tersebut. Mendapatkan dan menggunakan *power-ups* akan memberikan sejumlah poin yang signifikan kepada pemain yang melakukannya. Mengetahui hal tersebut, strategi *greedy* yang mengutamakan pendapatan dan penggunaan *power-up* dengan tujuan memaksimasi poin yang didapat menjadi salah satu alternatif solusi dari permainan *Overdrive*.

Kendaraan akan mengutamakan pergerakan menuju *power-up* yang berada pada peta untuk mendapatkan *power-up* tersebut dengan tujuan mendapatkan poin sebanyak-banyaknya. Setelah

kendaraan pemain mendapatkan *power-up* tersebut, pemain akan langsung menggunakan *power-up* tersebut untuk memaksimasi poin yang didapat oleh pemain.

b. Strategi menyerang

Terdapat beberapa *power-up* yang dapat digunakan untuk menyerang musuh. *Power-up* yang dimaksud adalah EMP, OIL, dan TWEET. Strategi ini mengutamakan penyerangan terhadap musuh dengan *power-up* tersebut. Penyerangan yang dilakukan akan menyebabkan kendaraan musuh untuk menjadi lebih lambat (OIL), dipaksa untuk pergi ke *lane* di sebelahnya dan diperlambat (EMP), dan dihalangi sehingga kendaraan akan bertabrakan dengan halangan tersebut yang menyebabkan kendaraan terhenti dan mengalami kerusakan (TWEET). Efek-efek tersebut diharapkan dapat menyebabkan lawan tertinggal jauh sehingga kendaraan yang menyerang dapat meraih kemenangan.

Untuk menjalankan strategi tersebut, penggunaan *power*-up akan digunakan semaksimum mungkin supaya efek dari *power-up* dapat dimanfaatkan sebaik mungkin. OIL akan digunakan ketika lawan berada di-*lane* yang sama dengan kendaraan pemain dan lawan berada dibelakang pemain. EMP digunakan apabila lawan mendahului pemain dan berada di lane yang tepat untuk dikenai EMP. TWEET akan meletakkan *Cybertruck* tepat didepan lawan, sehingga lawan pasti akan menabrak *Cybertruck* tersebut.

c. Bergerak secara aman

Saat bergerak, akan ada banyak *collision* yang terdapat di peta. Menabrak atau melewati *collision* tersebut tentunya merugikan karena dapat menyebabkan rusaknya kendaraan dan berkurangnya kecepatan kendaraan, serta pengurangan poin. Hal tersebut dapat menyebakan pemain kalah dari pertandingan. Oleh karena itu, salah satu strategi yang dapat diterapkan adalah bergerak dengan keselamatan kendaraan sebagai prioritasnya.

Kendaraan akan mengambil jalur yang paling aman untuk meminimasi *collision*. Apabila kendaraan melihat bahwa kendaraan akan melewati suatu *collision* seperti *mud* atau menabrak suatu *collision* seperti *wall*, kendaraan akan menggunakan *power-up* LIZARD atau berbelok ke kanan atau ke kiri. Kendaraan juga akan melihat apakah mungkin untuk berbelok ke kanan atau ke kiri tanpa menabrak *collision* yang ada pada *lane* di sebelah kanan maupun kiri. Apabila kendaraan terpaksa untuk melewati atau menabrak suatu *collision*, maka akan ada prioritas dari *collision* tersebut. Kendaraan akan melewati *collision* yang memberikan nilai kerusakan paling kecil untuk meminimasi kerusakan dari kendaraan.

d. Maksimasi kecepatan

Kecepatan merupakan salah satu aspek yang paling penting dalam permainan *Overdrive*. Apabila pemain dapat memaksimasi kecepatan kendaraannya selama permainan berlangsung, pemain dapat melewati lawan dan memenangkan permainan dengan menyentuh garis akhir terlebih dahulu.

Kecepatan kendaraan pemain dipengaruhi oleh beberapa hal. Dengan melakukan *command* ACCELERATE, kecepatan kendaraan akan menjadi lebih cepat, sedangkan *command* DECCELERATE akan mengurangi kecepatan kendaraan. Nilai kerusakan dari kendaraan memengaruhi kecepatan maksimal yang dapat dicapai oleh kendaraan. Nilai kerusakan tersebut dapat dikurangi dengan menggunakan command FIX, dengan demikian, kendaraan dapat mencapai kecepatan maksimal yang lebih tinggi.

Untuk memaksimasi kecepatan dari kendaraan pemain, maka akan diterapkan beberapa hal. Kendaraan akan sangat sering melakukan *command* ACCELERATE untuk mempercepat kendaraan. *Command* DECCELERATE tidak akan pernah digunakan karena *command* tersebut tidak memaksimasi kecepatan dari kendaraan. *Power-up* BOOST akan langsung digunakan apabila nilai kerusakan kendaraan sama dengan 0, bila tidak, akan dilakukan *command* FIX terus menerus hingga nilai kerusakan sama dengan 0. *Command* FIX juga akan digunakan apabila nilai kerusakan kendaraan menyebabkan kendaraan untuk tidak dapat meningkatkan kecepatannya lebih tinggi lagi apabila nilai kerusakan kendaraan tidak sama dengan 0. Penghindaran *collision* juga menjadi hal yang diperhatikan pada solusi ini, akan tetapi hal tersebut memiliki prioritas yang lebih rendah bila dibandingkan dengan hal-hal yang menambahkan kecepatan secara langsung.

3.3 Analisis Efisiensi

Solusi mengutamakan *power-ups* mengharuskan pemain untuk melakukan iterasi kepada peta untuk memeriksa lokasi *power-up* pada peta. Selain itu, pemain juga harus memeriksa *power-up* yang memiliki prioritas tertinggi serta jarak dari *power-up* tersebut dalam menentukan pengambilan *power-up.* Hal-hal tersebut menyebabkan strategi ini untuk memiliki kompleksitas yang cukup tinggi yaitu $O(n^2)$.

Strategi menyerang akan lebih fokus ke pengambilan informasi berupa lokasi musuh pada peta agar kendaraan pemain dapat menggunakan *power-up offensive* semaksimal mungkin. Selain itu, akan diperiksa juga urutan prioritas dari *power-up* yang akan digunakan. Berdasarkan hal tersebut, diketahui kompleksitas dari solusi ini, yaitu O(n).

Solusi selanjutnya adalah strategi bergerak secara aman. Strategi ini akan mencari jalur yang paling aman yang akan ditempuh oleh kendaraan pemain. Untuk melakukan hal tersebut, kendaraan akan memeriksa suatu jalur kemudian akan mengiterasi jalur tersebut untuk memeriksa apakah ada *collision* pada jalur tersebut atau tidak. Iterasi tersebut akan dilakukan pada jalur yang sedang diambil oleh pemain, jalur kanan, dan jalur kiri. Kemudian akan dibandingkan jalur mana yang paling aman (jarak *collision* paling jauh atau jalur mana yang tidak memiliki *collision*). Kendaraan juga akan melakukan pemeriksaan prioritas apabila kendaraan terpaksa harus menabrak suatu *collision*. Oleh karena itu, kompleksitas dari strategi ini *best-case scenario*-nya adalah $O(n^2)$ dan *worst-case scenario*-nya adalah $O(n^3)$.

Solusi yang terakhir, maksimasi kecepatan, memiliki kompleksitas yang tidak tinggi, yaitu O(n). Hal tersebut dikarenakan strategi ini sebagian besar hanya membutuhkan kendaraan untuk memeriksa nilai dari kerusakan kendaraan serta memeriksa apakah kendaraan memiliki BOOST atau tidak. Memang pada strategi ini diaplikasikan sebagian kecil strategi bergerak secara aman, akan tetapi pada kebanyakan kasus, kendaraan akan lebih sering untuk melakukan *command* yang menyebabkan kecepatan kendaraan untuk bertambah secara langsung daripada mencari jalur yang paling aman. Sehingga bila disimpulkan, strategi ini pada *best-case scenario*-nya memiliki kompleksitas O(n) dan pada *worst-case scenario*-nya memiliki kompleksitas $O(n^2)$.

3.4 Analisis Efektivitas

Alternatif solusi yang pertama yaitu mengutamakan *power-up*, menurut kami merupakan alternatif solusi yang kurang efektif. Hal tersebut dikarenakan mengutamakan pendapatan *power-up* untuk mengumpulkan poin sebanyak-banyaknya bukanlah strategi yang baik dalam permainan *Overdrive*. Poin memang merupakan salah satu faktor yang mempengaruhi kemenangan pada

permainan *Overdrive*, akan tetapi poin merupakan faktor dengan prioritas paling rendah dalam faktor memenangkan permainan bila dibandingkan dengan faktor yang lain seperti kecepatan. Selain itu, menggunakan *power-up* tanpa memperhatikan keadaan dari *round* pada saat itu bukanlah merupakan cara yang efektif dalam menggunakan *power-up*.

Solusi selanjutnya menggunakan power-up untuk menyerang musuh sebagai tujuan utamanya, dalam kata lain, algoritma menyerang akan memaksimalkan efektivitas dari *power-up offensive* yang didapat. *Power-up* OIL kurang efektif dalam menyerang musuh karena untuk menggunakan *power-up* tersebut secara efektif, banyak syarat yang perlu dipenuhi terlebih dahulu dan efek dari *power-up* tersebut tidaklah sepadan dengan syarat yang perlu dipenuhi. Sebaliknya, p*ower-up offensive* yang lain, yaitu EMP dan TWEET, merupakan *power-up* yang cukup efektif dalam penyerangan lawan. Syarat yang perlu dipenuhi agar penggunaan kedua *power-up* tersebut dimaksimalkan tidaklah banyak dan mudah untuk dipenuhi. Dampak dari *power-up* tersebut juga cukup ampuh dalam melumpuhkan musuh. Tentunya strategi ini sangat bergantung kepada kemunculan *power-up offensive* pada peta. Solusi ini dapat menjadi solusi yang sangat efektif dan dapat juga menjadi solusi yang paling tidak efektif tergantung dengan kemunculan *power-up*, sehingga kami menganggap alternatif ini kurang konsisten dalam efektivitasnya.

Bergerak secara aman menurut kami adalah solusi yang paling efektif di antara alternatif-alternatif solusi yang dieksplorasi. Hal tersebut dikarenakan solusi ini merupakan solusi yang paling konsisten dalam mempertahankan kecepatan kendaraan sepanjang pertandingan bila dibandingkan dengan solusi yang lain. Memang solusi lain memiliki *best-case scenario* yang lebih baik daripada solusi ini, akan tetapi solusi tersebut bisa saja memiliki *worst-case scenario* yang jauh lebih buruk ketimbang solusi bergerak secara aman, sehingga solusi tersebut kurang konsisten bila dibandingkan dengan solusi bergerak secara aman. Solusi bergerak secara aman dengan memilih jalur yang paling aman untuk ditempuh akan memaksimalkan *average-case scenario* yang menurut kami merupakan solusi yang cukup efektif untuk memenangkan pertandingan *Overdrive*.

Solusi yang terakhir adalah solusi yang akan memaksimalkan kecepatan. Pada solusi yang terakhir ini, idealnya kecepatan dari kendaraan akan selalu sama dengan atau mendekati nilai dari kecepatan maksimum yang dapat dimiliki oleh suatu kendaraan. Seperti strategi yang bergantung kepada *power-up*, strategi ini bergantung kepada peta yang di-*generate* selama pertandingan berlangsung. Kecepatan akan dimaksimasi dengan sangat mudah apabila jalur yang ditempuh oleh kendaraan memiliki *collision* yang minim, sebaliknya, apabila jalur yang diambil oleh kendaraan memiliki *collision* yang relatif cukup banyak, kendaraan akan sulit memaksimasi kecepatan kendaraan. Melihat hal tersebut, solusi ini sangat efektif apabila jalur yang ditempuh memiliki sedikit *collision* dan tidak efektif apabila jalur yang diambil memiliki banyak *collision.*

3.5 Strategi yang Dipilih

Pada akhirnya, kami menggabungkan alternatif solusi di atas dengan prioritas tertentu untuk menghasilkan strategi *greedy* yang menurut kami paling baik. Urutan prioritasnya adalah sebagai berikut: memperbaiki kendaraan apabila tidak bisa bergerak, mempercepat kendaraan apabila kecepatan kendaraan sama dengan 0, bergerak menghindari *collision* dengan berbelok atau menggunakan *power-up*, menyerang musuh, mempercepat serta memperbaiki kendaraan, dan bergerak menuju *power-ups*.

Kami memilih strategi tersebut karena walaupun alternatif solusi yang ditawarkan memiliki hal-hal positif yang dapat memenangkan pertandingan, tiap alternatif solusi juga

memiliki hal-hal negatif yang dapat menyebabkan kekalahan akan pertandingan. Melihat natur permainan *Overdrive* yang tidak dapat diduga dan acak, beberapa alternatif solusi dapat menjadi sangat unggul dibandingkan dengan yang lain dan juga dapat menjadi sangat buruk bila dibandingkan dengan yang lain. Dalam kata lain, alternatif solusi tersebut tidaklah konsisten. Maka dari itu untuk menghasilkan suatu solusi yang konsisten, kami menarik dan menggabungkan beberapa elemen yang terbaik dari alternatif-alternatif yang sudah dijabarkan untuk menghasilkan solusi yang menurut kami paling baik.

Pengurutan prioritas dari strategi *greedy* yang kami terapkan memiliki tujuan untuk mengutamakan kecepatan dari kendaraan kemudian pengumpulan poin. Secara umum, kendaraan yang memiliki kecepatan yang tertinggi akan memiliki kemungkinan untuk menang yang lebih besar. Kami ingin algoritma yang kami terapkan dapat menghasilkan kendaraan yang memiliki kecepatan yang konsisten selama lajunya pertandingan. Untuk merealisasikan hal tersebut, kami meletakan keselamatan kendaraan pada prioritas paling tinggi sedangkan pengambilan *power-up* pada prioritas paling rendah karena menurut kami poin merupakan elemen yang tidak terlalu penting dalam permainan *Overdrive.*

Pada permainan *Overdrive*, kami menarik kesimpulan bahwa efisiensi dari algoritma yang diimplementasikan tidak terlalu diperhatikan. Hal yang sangat diperhatikan adalah efektivitas dari algoritma tersebut, sehingga kami lebih mengutamakan efektivitas algoritma ketimbang efisiensi algortima.

Kami juga memastikan bahwa *command* yang dimasukkan merupakan *command* yang valid sehingga tidak akan ada pengurangan poin akibat masukan *command* yang tidak valid. Kami memeriksa keberadaan dari kendaraan pada *lane* untuk memeriksa apakah mungkin untuk melakukan *command* TURN_LEFT dan TURN_RIGHT serta memeriksa apakah kendaraan memiliki *power-up* sebelum menggunakan *power-up* tersebut.

BAB 4: IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

Mendeteksi blok di sekitar

```
public Command run(GameState gameState) {
    Car myCar = gameState.player;
    Car opponent = gameState.opponent;
    // Mendapatkan blok maksimum yang dapat dicapai saat myCar (player) bergerak lurus
    List<Lane> blocksMax = getBlocksInFront(myCar.position.lane, myCar.position.block, gameState, laneIndicator: 0);

    // Mendapatkan blok maksimum yang dapat dicapai saat myCar (player) bergerak ke lane kanan
    List<Lane> rightBlocks = getBlocksInFront(myCar.position.lane, myCar.position.block, gameState, laneIndicator: 1);

    // Mendapatkan blok maksimum yang dapat dicapai saat myCar (player) bergerak ke lane kiri
    List<Lane> leftBlocks = getBlocksInFront(myCar.position.lane, myCar.position.block, gameState, laneIndicator: -1);

    List<Lane> blocks;
    if (blocksMax.size() >= myCar.speed && myCar.boostCounter == 1) {
        // Boost di turn selanjutnya habis kembali ke max speed
        blocks = blocksMax.subList(0, Bot.MAXIMUM_SPEED);
    } else if (blocksMax.size() >= myCar.speed && myCar.boostCounter == 0) {
        // Tidak sedang dalam boost
        blocks = blocksMax.subList(0, myCar.speed);
    } else {
        blocks = blocksMax;
    }

    // Memperbaiki mobil saat tidak bisa bergerak lagi
    if (myCar.damage >= 5) {// Fix if car completely broken
        return FIX;
    }

    // Mempercepat myCar (player) jika kecepatan terlalu rendah, jika bisa boost secara efektif, boost dilakukan
    if (myCar.speed == MINIMUM_SPEED ) {// ACCELERATE IF SPEED 0
        if (canReachBoostSpeed(gameState)) {
            return BOOST;
        }
        return ACCELERATE;
    }
```

Mendeteksi bahaya yang ada di depan bot dan menggunakan powerup jika ada

```
    // Mengecek bahaya yang langsung berada di depan myCar (player)
    String immediateDanger = checkImmediateDanger(blocks, gameState, laneIndicator: 0);

    // Mekanisme menghindar (pindah jalur (lane) dan pwer up lizard) serta sedikit mekanisme menyerang
    // Jika ada bahaya di depan
    if (immediateDanger != null) {// Lizard mechanism and Avoidance mechanism
        String immediateDangerLeft;
        String immediateDangerRight;

        // Melakukan pengecekan bahaya yang akan langsung membahayakan myCar (player) jika bergerak ke kanan dan ke kiri
        if (myCar.position.lane == 1) {
            immediateDangerLeft = null;
        } else {
            immediateDangerLeft = checkImmediateDanger(leftBlocks, gameState, laneIndicator: 1);
        }

        if (myCar.position.lane == 4) {
            immediateDangerRight = null;
        } else {
            immediateDangerRight = checkImmediateDanger(rightBlocks, gameState, laneIndicator: 1);
        }
```

```
// Jika dideteksi lawan ada di depan dan myCar (player) diperkirakan tidak akan menabrak lawan dan memiliki power up EMP
if (immediateDanger == "OPPONENT" && Math.abs(myCar.position.lane - opponent.position.lane) <= 1) {
    if (countPowerUp(PowerUps.EMP, myCar.powerups) > 0 && ((myCar.position.lane != opponent.position.lane)
            || (myCar.position.block <= opponent.position.block - myCar.speed + 3))) {
        return EMP;
    }
}

// Jika punya lizard, gunakan lizard untuk menghindari bahaya sekaligus menambah poin dan menjaga myCar (player) tetap lurus
if (countPowerUp(PowerUps.LIZARD, myCar.powerups) > 0) {
    return LIZARD;
}
```

Menghitung damage yang akan mempengaruhi bot

```
// menghitung damage yang dari bahaya yang akan langsung ditabrak oleh myCar (player)
int damage = countImmediateDamage(immediateDanger);

// Jika player berada di lajur (lane) pinggir, damage sebelahnya akan dijadikan 999 untuk mencegah keluar jalur (lane)
int rightDamage = myCar.position.lane == 4 ? 999 : countImmediateDamage(immediateDangerRight);
int leftDamage = myCar.position.lane == 1 ? 999 : countImmediateDamage(immediateDangerLeft);

// Menghitung damage yang paling kecil dari 3 jalur (lane) berbeda
int minDamage = min(damage, min(rightDamage, leftDamage));
```

Menghitung powerup yang ada di jalur

```
// Menghitung jumlah power up di jalur (lane) lurus, kiri, dan kanan
int countPower = countPowerUpInBlocks(blocks);
int countPowerLeft = countPowerUpInBlocks(leftBlocks);
int countPowerRight = countPowerUpInBlocks(rightBlocks);

// Menghitung powerup terbanyak
int maxPower = max(countPower, max(countPowerLeft, countPowerRight));

// Menghitung power up kedua terbanyak
int midPower = maxPower;
if ((countPowerLeft <= countPower && countPower <= countPowerRight) || (countPowerRight <= countPower && countPower <= countPowerLeft)) {
    midPower = countPower;
} else if ((countPower <= countPowerLeft && countPowerLeft <= countPowerRight)
        || (countPowerRight <= countPowerLeft && countPowerLeft <= countPower)) {
    midPower = countPowerLeft;
} else if ((countPower <= countPowerRight && countPowerRight <= countPowerLeft)
        || (countPowerLeft <= countPowerRight && countPowerRight <= countPower)) {
    midPower = countPowerRight;
}
```

Berpindah jalur jika damage lebih kecil atau terdapat powerup

```
// Jika saat lurus damagenya adalah yang paling kecil dan tidak ada cyber truck didepan, tidak usah lakukan apa-apa
if (damage == minDamage && !hasCyberTruck(blocks)) {
    return DO_NOTHING;
}

// Jika berada di lane paling kanan dan lajur kiri tidak mempunyai cyber truck, belok kiri
if (myCar.position.lane == 4 && !hasCyberTruck(leftBlocks)) {
    return TURN_LEFT;
}

// Jika berada di lane paling kiri dan lajur kanan tidak mempunyai cyber truck, belok kanan
if (myCar.position.lane == 1 && !hasCyberTruck(rightBlocks)) {
    return TURN_RIGHT;
}

// Jika power paling banyak ternyata berada di tengah, ubah power paling banyak menjadi yang kedua terbanyak,
// hal ini dilakukan karena di bagian kode di bawah, hanya ada dua pilihan, belok kiri atau kanan
if (maxPower == countPower) {
    maxPower = midPower;
}
```

```
        // Damage kiri dan kanan sama
        if (leftDamage == rightDamage) {
            if (!hasCyberTruck(leftBlocks) && countPowerLeft == maxPower) {
                return TURN_LEFT;
            }
            if (!hasCyberTruck(rightBlocks) && countPowerRight == maxPower) {
                return TURN_RIGHT;
            }
            return DO_NOTHING; // Default return value
        }

        if (!hasCyberTruck(leftBlocks) && leftDamage == minDamage) {
            return TURN_LEFT;
        }

        if (!hasCyberTruck(rightBlocks) && rightDamage == minDamage) {
            return TURN_RIGHT;
        }

        return DO_NOTHING; // Default return value
    }
```

Menggunakan powerup yang dimiliki

```
    // Bagian menyerang
    // Gunakan EMP jika lawan ada di jalur pinggir dan kita ada di jalur sebelah lawan, lawan tidak mungkin menghindar dari EMP
    if (countPowerUp(PowerUps.EMP, myCar.powerups) > 0 && (myCar.position.block <= opponent.position.block - myCar.speed + 2)
            && (opponent.position.lane == 1 && myCar.position.lane == 2)){
        return EMP;
    }

    if (countPowerUp(PowerUps.EMP, myCar.powerups) > 0 && (myCar.position.block <= opponent.position.block - myCar.speed + 2)
            && (opponent.position.lane == 4 && myCar.position.lane == 3)){
        return EMP;
    }

    // Menempatkan Cyber Truck jika punya, penempatan dilakukan dengan melakukan prediksi posisi lawan
    if (countPowerUp(PowerUps.TWEET, myCar.powerups) > 0) {
        return new TweetCommand(opponent.position.lane, opponent.position.block + faster(opponent.speed) + 3);
    }

    // Gunakan power up oil jika lawan diprediksi akan melewati posisi myCar (player) berada sekarang
    if ((countPowerUp(PowerUps.OIL, myCar.powerups) > 0) && (myCar.position.block > opponent.position.block)
            && (myCar.position.block <= opponent.position.block + opponent.speed) && myCar.position.lane == opponent.position.lane) {
        return OIL;
    }
```

```
     // Jika mempunyai boost
     if (countPowerUp(PowerUps.BOOST, myCar.powerups) > 0) {// Boost mechanism
         // Jika bisa sampai booost speed
         if (canReachBoostSpeed(gameState)) {
             return BOOST;
         } else if (myCar.damage > 0) { // Perbaiki agar bisa mencapai kecepatan boost
             return FIX;
         }
     }
```

Akselerasi jika bisa atau perbaiki bot jika damagenya besar

```
// Akselerasi
if (canAccelerate(gameState)) {
    return ACCELERATE;
}

// Perbaiki karena tidak mungkin akselerasi lagi
if (myCar.damage == 4 && myCar.speed == SPEED_STATE_1 ) {
    return FIX;
}

// Perbaiki karena tidak mungkin akselerasi lagi
if (myCar.damage == 3 && myCar.speed == SPEED_STATE_2 ) {
    return FIX;
}

// Perbaiki karena tidak mungkin akselerasi lagi
if (myCar.damage == 2 && myCar.speed == SPEED_STATE_3) {
    return FIX;
}
```

Mekanisme menambah skor

```
// Mekanisme pengambilan bonus untuk menambah skor sekaligus menghindari damage
String immediateDangerLeft;
String immediateDangerRight;

if (myCar.position.lane == 1) {
    immediateDangerLeft = null;
} else {
    immediateDangerLeft = checkImmediateDanger(leftBlocks, gameState,  laneIndicator: 1);
}

if (myCar.position.lane == 4) {
    immediateDangerRight = null;
} else {
    immediateDangerRight = checkImmediateDanger(rightBlocks, gameState,  laneIndicator: 1);
}
```

Menghitung damage dan powerup

```
    int damage = countImmediateDamage(immediateDanger);
    int rightDamage = myCar.position.lane == 4 ? 999 : countImmediateDamage(immediateDangerRight);
    int leftDamage = myCar.position.lane == 1 ? 999 : countImmediateDamage(immediateDangerLeft);

    int countPower = countPowerUpInBlocks(blocks);
    int countPowerLeft = countPowerUpInBlocks(leftBlocks);
    int countPowerRight = countPowerUpInBlocks(rightBlocks);
    int maxPower = max(countPower, max(countPowerLeft, countPowerRight));

    if (countPower == maxPower && damage == 0) {
        return DO_NOTHING;
    }

    if (countPowerLeft == maxPower && leftDamage == 0) {
        return TURN_LEFT;
    }

    if (countPowerRight == maxPower && rightDamage == 0) {
        return TURN_RIGHT;
    }

    return DO_NOTHING;
}
```

Fungsi untuk menghitung powerup

```
private int countPowerUpInBlocks(List<Lane> blocks) {
    // Menghitung power up dalam suatu list of Lane
    int ret = 0;
    for (Lane block: blocks) {
        if (block.terrain.equals(Terrain.BOOST) || block.terrain.equals(Terrain.OIL_POWER) || block.terrain.equals(Terrain.LIZARD)
                || block.terrain.equals(Terrain.TWEET) || block.terrain.equals(Terrain.EMP)) {
            ret++;
        }
    }
    return ret;
}

private int countPowerUp(PowerUps powerUpToCheck, PowerUps[] available) {
    // Menghitung power up tertentu yang dimiliki
    int ret = 0;
    for (PowerUps powerUp: available) {
        if (powerUp.equals(powerUpToCheck)) {
            ret += 1;
        }
    }
    return ret;
}
```

Fungsi untuk memeriksa bahaya di sekitar

```
private String checkImmediateDanger(List<Lane> blocks, GameState gameState, int laneIndicator) {
    // Mengecek bahaya terdekat yang akan ditemui langsung oleh myCar (player)
    List<Terrain> terrains = blocks.stream().map(element -> element.terrain).collect(Collectors.toList());
    Car player = gameState.player;
    Car opponent = gameState.opponent;
    String nearestTerrain = null;
    int nearestTerrainX = player.position.block + 1;
    int nearestCyberTruckX = player.position.block + 1;
    int speed = laneIndicator == 0 ? player.speed : player.speed - 1;

    boolean opponentExist = blocks.stream().map(element -> element.occupiedByPlayerId).collect(Collectors.toList()).contains(opponent.id)
            && (speed + player.position.block >= faster(gameState.opponent.speed) + opponent.position.block);
    int cyberTruck = blocks.stream().map(element -> element.cyberTruck).collect(Collectors.toList()).indexOf(true);

    if (cyberTruck != -1) {
        nearestCyberTruckX += cyberTruck;
    }
      for (Terrain terrain: terrains) {
          if (terrain.equals(Terrain.MUD)) {
              nearestTerrain = "MUD";
              break;
          } else if (terrain.equals(Terrain.WALL)) {
              nearestTerrain = "WALL";
              break;
          } else if (terrain.equals(Terrain.OIL_SPILL)) {
              nearestTerrain = "OIL";
              break;
          }
          nearestTerrainX++;
      }

      if (nearestCyberTruckX <= nearestTerrainX) {
          nearestTerrain = null;
      } else {
          cyberTruck = -1;
      }
```

```
    if (nearestTerrain != null || cyberTruck != -1 || opponentExist) {
        if (nearestTerrain != null) {
            if (opponent.position.block <= nearestTerrainX) {
                return "OPPONENT";
            } else {
                return nearestTerrain;
            }
        }

        if (cyberTruck != -1) {
            if (opponent.position.block < nearestTerrainX) {
                return "OPPONENT";
            } else {
                return "CYBERTRUCK";
            }
        }
        return "CYBERTRUCK";
    }

    return null;
}
```

Fungsi untuk menghitung damage

```
private int countImmediateDamage(String immediateDamage) {
    // Menghitung damage yang diberikan berdasarkan jenisny
    if (immediateDamage == "MUD" || immediateDamage == "OIL") {
        return 1;
    }

    if (immediateDamage == "WALL") {
        return 2;
    }

    // Cyber Truck diberi damage 3, walaupun sebenarnya memiiki damage 2,
    // hal ini dilakukan karena Cyber Truck dapat menghentikan mobil sehingga lebih baik dihindari
    if (immediateDamage == "CYBERTRUCK") {
        return 3;
    }

    return 0;
}
```

Fungsi untuk memerksa apakah bisa mempercepat laju

```
private boolean canAccelerate(GameState gameState) {
    // Melakukan pengecekan bisa akselerasi atau tidak berdasarkan damage yang dimiliki, bukan boost
    int damage = gameState.player.damage;
    int speed = gameState.player.speed;
    if (damage == 0 && speed <= Bot.SPEED_STATE_3) {
        return true;
    }
    if (damage == 1 && speed <= Bot.SPEED_STATE_3) {
        return true;
    }
    if (damage == 2 && speed <= Bot.SPEED_STATE_2) {
        return true;
    }
    if (damage == 3 && speed <= Bot.SPEED_STATE_1) {
        return true;
    }
    if (damage == 4 && speed <= Bot.MINIMUM_SPEED) {
        return true;
    }
    return false;
}
```

Fungsi untuk menambah kecepatan

```
private int faster(int speed) {
    // Mengembalikan kecepatan satu tingkat di atas kecepatan yang dimasukkan
    if (speed == Bot.INITIAL_SPEED ) {
        return Bot.SPEED_STATE_2 ;
    }
    if (speed == Bot.BOOST_SPEED) {
        return speed;
    }
    for (int i = 0; i < 5; i++) {
        if (speed == Bot.SPEED_STATE[i]) {
            return Bot.SPEED_STATE[i + 1];
        }
    }
    return speed; // default return
}
```

Fungsi untuk memeriksa apakah ada cybertruck, dan apakah bisa mencapai top speed

```
private boolean hasCyberTruck(List<Lane> blocks) {
    // Mengecek apakah di dalam suatu list of Lane terdapat Cyber Truck
    return blocks.stream().map(element -> element.cyberTruck).collect(Collectors.toList()).contains(true);
}

private boolean canReachBoostSpeed(GameState gameState) {
    // Mengecek jika boost speed dapat dicapai
    if (countPowerUp(PowerUps.BOOST, gameState.player.powerups) > 0) {
        if (gameState.player.damage > 0 || gameState.player.boostCounter > 1) {
            return false;
        }
        return true;
    }
    return false;
}
```

Fungsi untuk mendapat block di sekitar

```
private List<Lane> getBlocksInFront(int lane, int block, GameState gameState, int laneIndicator) {
    // mengambil blok (list of Lane) yang terdapat di depan
    List<Lane[]> map = gameState.lanes;
    List<Lane> blocks = new ArrayList<>();
    int startBlock = map.get(0)[0].position.block;
    // Jika belok, speed dikurangi satu
    int speed = laneIndicator == 0 ? gameState.player.speed : gameState.player.speed - 1;
    if (laneIndicator == 0) {
        if (canReachBoostSpeed(gameState)) {
            speed = Bot.BOOST_SPEED ;
        }
        else if (canAccelerate(gameState)) {
            speed = faster(speed);
        }
    }
    else if (laneIndicator == -1) {
        if (lane == 1) {
            return blocks;
        }
        // Dikurangi karena belok
        block--;
    }
    else {
        if (lane == 4) {
            return blocks;
        }
        // Dikurangi karena belok
        block--;
    }
    Lane[] laneList = map.get(lane + laneIndicator - 1);
    for (int i = max(block - startBlock, 0) + 1; i <= block - startBlock + speed; i++) {
        if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) {
            break;
        }
        blocks.add(laneList[i]);
    }
    return blocks;
}
}
```

4.2 Struktur Data yang Digunakan

Permainan *Overdrive* menggunakan struktur data berbasis *class*. Kelas-kelas yang ada pada permainan *Overdrive* telah dibuat dan disediakan oleh *Entelect*. Kelas-kelas tersebut dibagi menjadi 5, yaitu: C*ommand, Entities, Enums, Bot*, dan *Main.*

a. Command

Kelas-kelas yang berada di bagian ini berguna untuk menghasilkan *command* yang akan dieksekusi oleh program.

i. AccelerateCommand
Kelas untuk menghasilkan masukan berupa *command* ACCELERATE.

ii. BoostCommand
Kelas untuk menghasilkan masukan berupa *command* USE_BOOST.

iii. ChangeLaneCommand
Kelas untuk menghasilkan masukan berupa *command* TURN_LEFT atau TURN_RIGHT.

iv. DecelerateCommand
Kelas untuk menghasilkan masukan berupa *command* DECELERATE.

v. DoNothingCommand
Kelas untuk menghasilkan masukan berupa *command* DO_NOTHING.

vi. EmpCommand
Kelas untuk menghasilkan masukan berupa *command* USE_EMP.

vii. FixCommand
Kelas untuk menghasilkan masukan berupa *command* FIX.

viii. OilCommand
Kelas untuk menghasilkan masukan berupa *command* USE_OIL.

ix. Command
Kelas yang digunakan untuk menghasilkan sebuah input *command*.

b. Entities

Kelas-kelas bagian ini menyimpan entity yang berada di dalam permainan Overdrive

i. Car
Kelas untuk menyimpan atribut kendaraan seperti ID, posisi, kecepatan, *power-up* yang dimiliki kendaraan, dan lain-lain.

ii. GameState
Kelas untuk menyimpan atribut dari keadaan permainan pada *round* tertentu, yaitu *round* pada saat ini serta *round* maksimum dari permainan, kendaraan dari pemain dan lawan, serta peta permainan pada *round* saat itu.

iii. Lane
Kelas untuk menyimpan atribut dari *lane*, yaitu *position, terrain,* dan informasi apakah lane tersebut ditempati oleh pemain dengan ID tertentu.

iv. Position
Kelas untuk menyimpan posisi sebagai atribut y(*lane*) dan x(*block*).

c. Enums
Berisikan kelas-kelas yang menyimpan arah pergerakan, *power-up, state* dari kendaraan, dan *terrain* pada peta.

i. Direction
Kelas untuk menyimpan pergerakan dari kendaraan, baik itu berbelok ke kanan dan ke kiri, serta bergerak maju dan mundur.

ii. PowerUps
Kelas untuk menyimpan *power-up* yang dimiliki kendaraan.

iii. State
Kelas untuk menyimpan keadaan dari kendaraan pada *round* tertentu, seperti berakselerasi, berbelok ke kanan, menggunakan BOOST, dan lain-lain.

iv. Terrain
Kelas yang menyimpan medan yang berada di peta seperti MUD, BOOST, garis akhir, dan lain-lain.

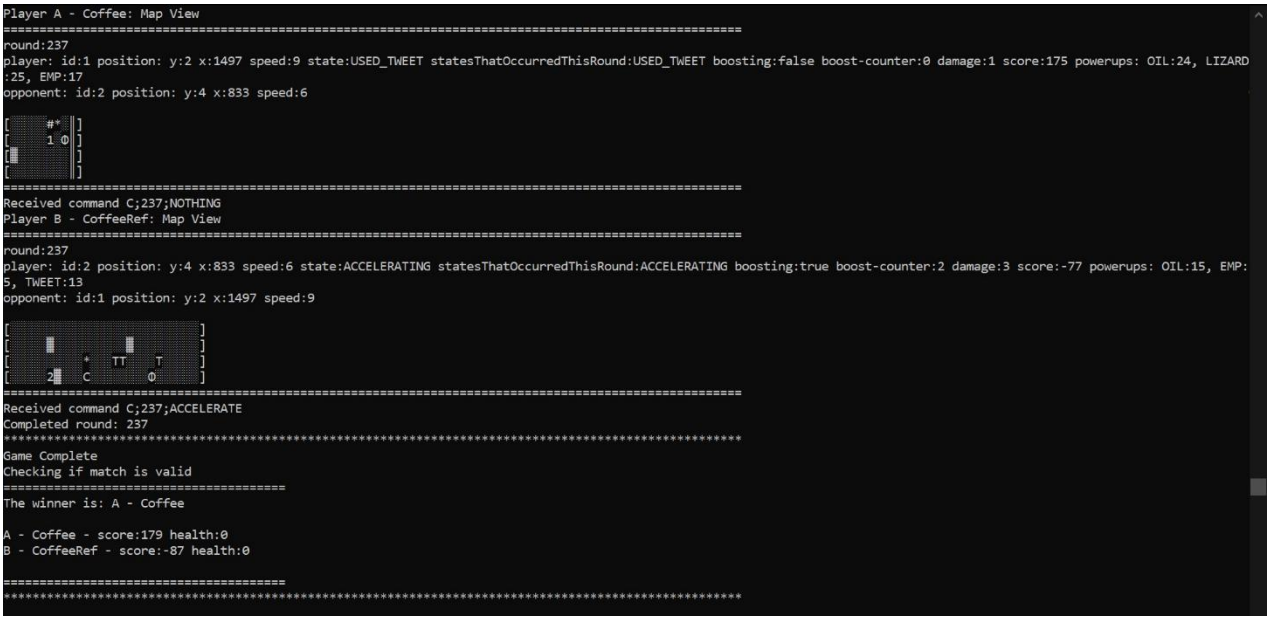
d. Bot
Kelas ini berisi implementasi dari algoritma *greedy* yang telah kami rancang untuk memenangkan permainan *Overdrive*.

e. Main
Kelas yang memanggil semua kelas dan algoritma yang ada serta mengolah masukan-masukan yang diterima untuk menjalankan program *Overdrive*.

4.3 Analisis Desain Solusi

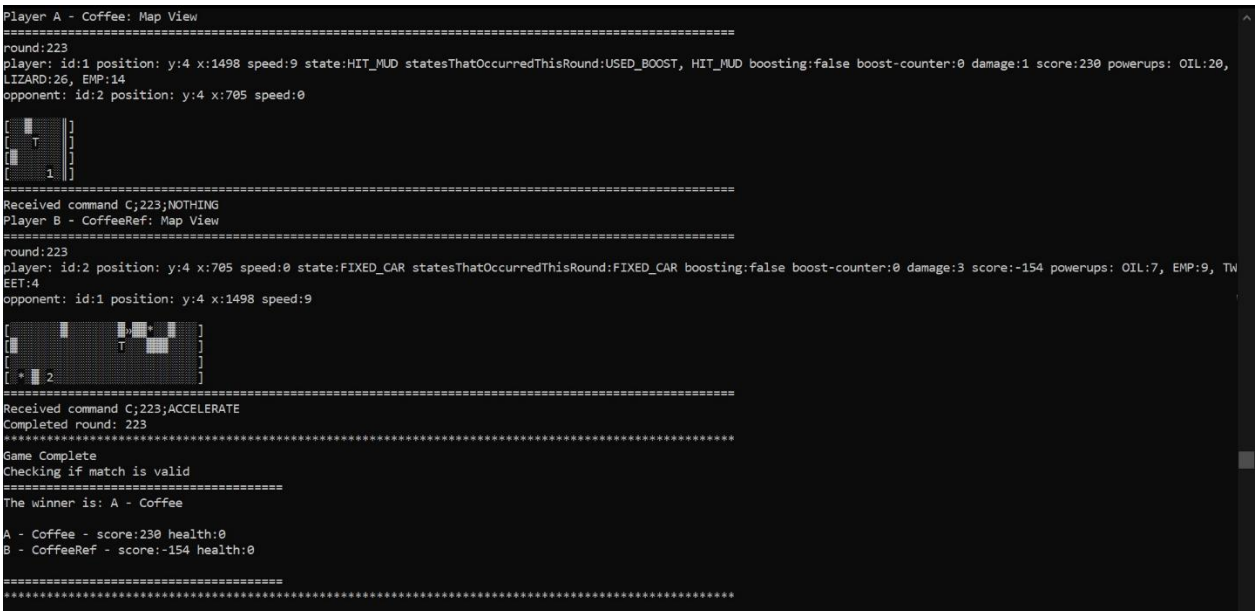
Untuk menguji desain solusi algoritma *greedy* yang telah diimplementasikan, akan digunakan *reference-bot* yang telah disediakan saat men-*download starter-pack* sebagai lawan. Akan dilakukan 3 kali pengujian terhadap *reference-bot.* Untuk mengevaluasi hasil dari pengujian, akan dilihat selisih dari lokasi *block* lawan dan pemain dan selisih dari poin lawan dan pemain.

a. Pengujian pertama



Pada pengujian pertama, *bot* yang kami telah rancang berhasil mengalahkan *reference-bot* dalam 237 *round*. Bila dilihat dari lokasi akhir kendaraan, dapat dilihat bahwa kendaraan kami berada pada *block* 1497 dan kendaraan musuh berada pada *block* 833. Selisih yang besar antara kedua nilai tersebut menyimpulkan bahwa algoritma yang kami rancang sudah cukup efektif dalam mendahului kendaraan musuh. Dapat dilihat juga jumlah poin yang dimiliki oleh kendaran kami dan kendaraan lawan. Selisih poin yang cukup signifikan menyimpulkan bahwa algoritma kami telah menvalidasi input *command* dan menghindari *collision* yang ada pada peta dengan cukup baik.

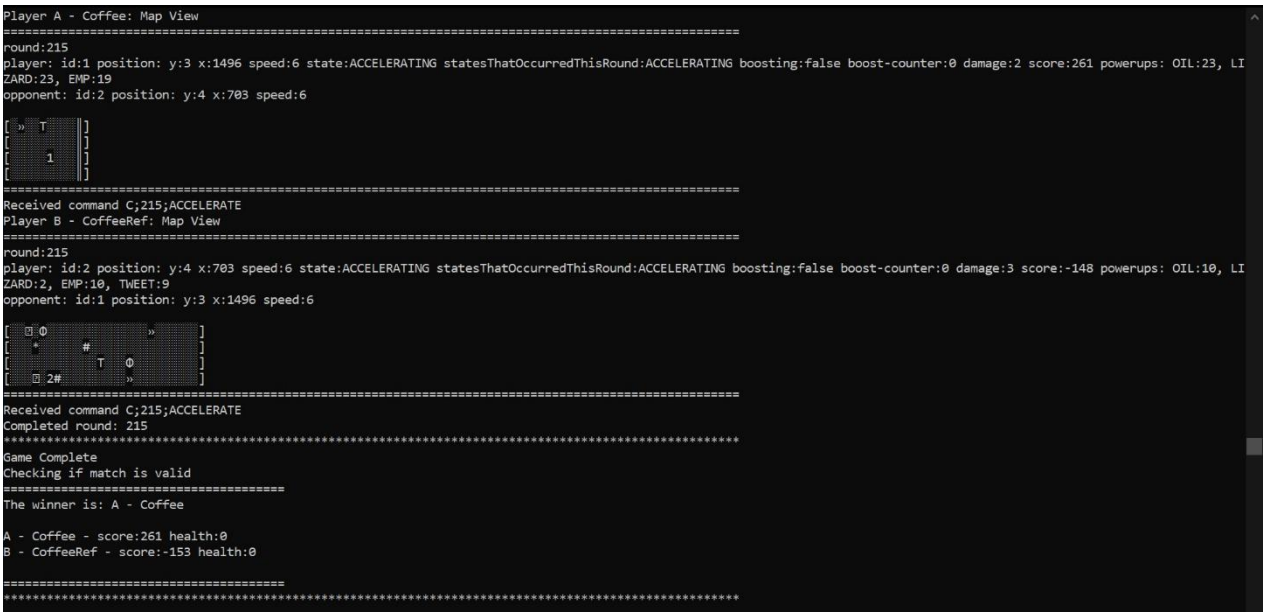
b. Pengujian kedua



Pada pengujian kedua, kami berhasil mengalahkan *bot* dalam 223 *round*. Selisih dari *block* akhir antara *bot* kami dan lawan juga cukup besar, yaitu 1498-705 = 793. Selisih poin yang didapat juga cukup besar, yaitu 384. Untuk kedua selisih tersebut, kendaraan kami memiliki nilai yang lebih

besar dari lawan, sehingga diketahui bahwa kendaraan kami jauh lebih unggul dibandingkan kendaraan lawan.

c. Pengujian ketiga



Pada pengujian ketiga, hasil yang didapat relatif sama dengan hasil pengujian pertama dan kedua. Selisih dari *block* dan poin antara kedua kendaraan cukup besar dengan kendaraan kami memiliki poin dan nilai *block* yang paling besar. Hal tersebut mengatakan bahwa algoritma yang kami implementasikan sudah cukup efektif untuk memenangkan pertandingan.

Dari ketiga pengujian diatas, diketahui bahwa algoritma yang kami implementasikan sudah cukup efektif dalam memenangkan pertandingan *Overdrive*. Hal tersebut ditandai dengan keunggulan yang kendaraan kami miliki saat pertandingan selesai. Keunggulan yang kendaraan kami miliki cukup signifikan baik dalam jumlah *block* (yang berarti kendaraan kami lumayan cepat selama pertandingan berlangsung) dan jumlah poin. Akan tetapi, perlu diperhatikan bahwa *reference-bot* yang digunakan sebagai lawan kami tidak memiliki algoritma yang kompleks dan mendalam seperti milik kami. Sehingga hasil pertandingan mungkin saja berbeda apabila *bot* kami dipertandingkan dengan *bot* lain yang memiliki kompleksitas dan kedalaman algoritma yang setara.

BAB 5: KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kami berhasil mengimplementasikan bot untuk permainan *Overdrive* dengan algoritma *greedy* dengan tujuan memenangkan permainan *Overdrive*. Implementasi yang kami buat sudah cukup baik untuk mengalahkan *reference-bot* yang disediakan oleh *Entelect* sebagai uji coba. Kami memiliki strategi untuk memaksimalkan konsistensi kecepatan dari kendaraan kami. Untuk merealisasikan strategi tersebut, kami mencari beberapa alternatif solusi dan strategi yang dapat diterapkan ke permainan *Overdrive* lalu mengambil elemen-elemen yang terbaik dari alternatif-alternatif tersebut untuk merancang suatu *bot* yang cukup efektif dalam memenangkan permainan *Overdrive*.

5.2 Saran

Pengaplikasian algoritma *greedy* yang kami buat untuk memenangkan permainan *Overdrive* tidaklah sempurna. Saran kami untuk meningkatkan performa algoritma *greedy* baik dalam efektivitas dan efisiensi adalah melakukan pengujian dengan berbagai macam lawan. Dengan melakukan hal tersebut, akan terlihat secara lebih jelas kelemahan dari algoritma *greedy* yang diimplementasikan. Kelemahan-kelemahan tersebut kemudian dapat diatasi dan diperbaiki, sehingga kualitas dari algoritma yang diimplementasikan meningkat secara signifikan. Proses tersebut akan diulangi terus-menerus hingga algoritma *greedy* untuk permainan *Overdrive* memiliki kualitas yang sangat baik.

DAFTAR PUSTAKA

Munir, R. (2020). Algoritma Greedy Bagian 1[PDF]. Institut Teknologi Bandung. Diakses pada 13 Februari 2022 pukul 20.13 WIB melalui *https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf*.

LAMPIRAN

Link Github: https://github.com/davidkarelhp/Lightning_MisQueen
Link Video Youtube: https://youtu.be/2TvilW1PoSA