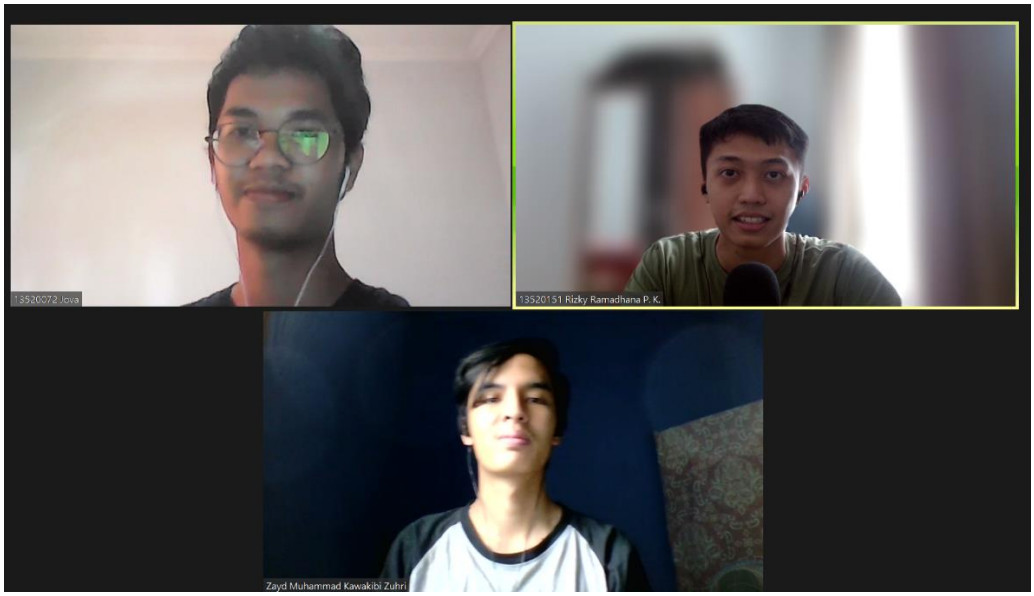
LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA



Disusun oleh:

Jova Andres Riski Sirait	13520072
Zayd Muhammad Kawakibi Zuhri	13520144
Rizky Ramadhana P. K.	13520151

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 2 TAHUN 2021/2022

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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

Gambar 1. Ilustrasi permainan Overdrive

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

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan strategi greedy untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada starter-bots di dalam starter-pack pada laman berikut ini: https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Overdrive pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

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

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. IntelIiJ 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 IntelIiJ 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

A. Greedy Algorithm

Algoritma *Greedy* merupakan suatu strategi algoritma yang mengikuti pola pemecahan masalah secara heuristik, dengan selalu memilih pilihan yang optimal pada tiap langkah masalah. Algoritma ini cukup populer karena metode yang cukup sederhana dan mudah dipahami, sehingga menjadi pilihan yang layak digunakan untuk memecahkan persoalan optimasi. Masalah optimasi yang dapat menggunakan algoritma *greedy* adalah maksimasi dan minimasi.

Dalam algoritma *greedy* persoalan dipecahkan secara berlangkah sehingga pada setiap langkah algoritma mengambil pilihan yang menghasilkan hasil terbaik pada saat itu juga, tanpa mempertimbangkan langkah-langkah selanjutnya. Karena inilah algoritma disebut *greedy*, karena memegang prinsip *"take what you can get now"* pada setiap langkah, sehingga bersifat rakus. Pada setiap langkah kita hanya bisa berharap bahwa dengan memilih optimum lokal tersebut, algoritma akan menghasilkan hasil akhir optimum global.

Algoritma *Greedy* dapat dipecah menjadi beberapa elemen: himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi obyektif. Jika kita mengambil contoh persoalan *Knapsack* problem, himpunan kandidat berisi kandidat yang dapat dipilih di setiap langkah, atau dalam masalah ini setiap objek yang dapat dimasukkan dalam tas. Himpunan solusi berisi kandidat yang terpilih dalam setiap langkah dan akan menjadi solusi, dalam contoh ini setiap objek yang sudah dimasukkan ke dalam tas. Fungsi solusi menentukan apakah himpunan solusi sudah pantas menjadi solusi atau tidak, yang dalam knapsack problem adalah apakah tidak ada objek lagi yang tidak dapat dimasukkan ke dalam tas, dan algoritma akan berhenti. Fungsi seleksi memilih kandidat dari himpunan kandidat dan memasukkannya ke dalam himpunan solusi. Fungsi ini dapat menggunakan strategi tertentu yang *greedy*, seperti memilih objek dengan profit tertinggi dalam knapsack problem. Fungsi kelayakan menentukan apakah kandidat yang dipilih memang dapat dimasukkan ke himpunan solusi, misal berat yang terbatas pada tas. Terakhir adalah fungsi objektif yang memaksimumkan atau meminimumkan algoritma berdasarkan tujuan dari persoalan, atau dalam contoh knapsack adalah memaksimumkan profit yang didapatkan.

B. Cara Kerja Program

Permainan overdrive membutuhkan dua bot yang akan dipertandingkan algoritmanya dalam hal melewati rintangan dan menggunakan power ups yang dimilikinya guna mencapai garis finish terlebih dahulu dibanding bot lawan.

Pada *starter-pack* permainan ini sendiri sudah disediakan *game engine* dimana tugas bot adalah memberikan perintah (*command*) ke *game engine* untuk melakukan aksi yang akan diambil oleh pemain. Perintah yang akan diambil oleh pemain akan disesuaikan dengan kondisi/*state* pemain pada saat tertentu. Misalnya jika terdapat halangan di depan, bot akan memberikan perintah "TURN_LEFT" ataupun "TURN_RIGHT" untuk menghindari halangan tersebut, atau bisa juga perintah lainnya tergantung algoritma yang diimplementasikan.

Untuk mengimplementasikan algoritma greedy yang dibuat ke dalam bot, strategi yang telah dibuat perlu ditransformasikan ke dalam bahasa yang dipilih. Contohnya pada bahasa Java, algoritma greedy dituliskan pada kelas *Bot.Java*, dengan menambahkan fungsi baru untuk mengecek state, menghitung kejadian paling menguntungkan, dan lain lain serta menuliskan kode utama pada fungsi *run.*

Untuk mempertandingkan dua buah bot, konfigurasi dapat dilakukan pada file "game-runner-config.json" dengan menentukan folder kedua bot pada *field* "player-a" dan "player-b".

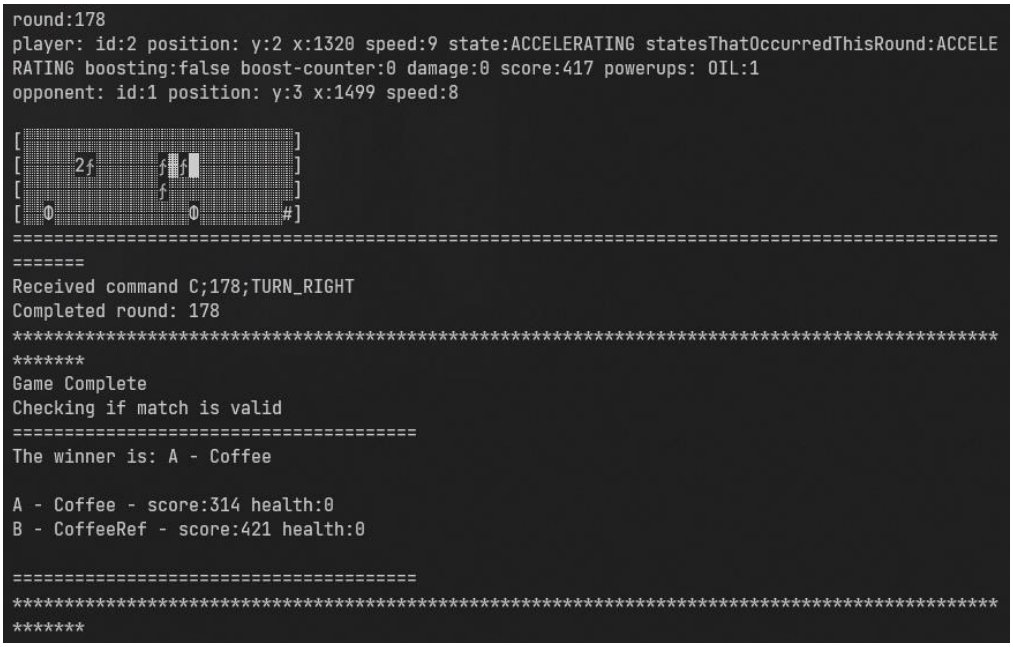
Pada bahasa Java (yang akan digunakan pada tugas ini), file yang dibutuhkan adalah hasil kompilasi program yang berupa file jar pada folder "target". Compile dapat dilakukan dengan bantuan Maven sebagai *build automation tool*. Perintah selanjutnya yang perlu dijalankan pada terminal adalah sebagai berikut:

```
chcp 65001

java -Dfile.encoding=UTF-8 -jar ./game-runner-jar-with-dependencies.jar
```

```
pause
```

Setelah itu, *game engine* akan menjalankan perintah yang diberikan oleh kedua bot pada setiap putaran yang berlangsung, memberikan informasi detail pemain (*damage, power ups, state, dll*). Berikut adalah tampilan akhir ketika program dijalankan:



BAB III
APLIKASI STRATEGI GREEDY

A. Pemetaan Permasalahan *Overdrive* Ke Elemen Algoritma Greedy

Algoritma greedy memiliki beberapa elemen yang nantinya bisa membedakan satu algoritma greedy dengan algoritma greedy lainnya. Elemen-elemen tersebut adalah himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Pada persoalan *overdrive* ini himpunan kandidat adalah kumpulan dari berbagai perintah yang bisa diberikan kepada *game engine*, yaitu NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP, dan FIX. Sedangkan himpunan solusi adalah himpunan yang isinya urutan perintah yang akan dieksekusi *game engine*. Fungsi solusi pada permasalahan *overdrive* adalah fungsi yang mengecek apakah urutan urutan pada himpunan solusi dapat membawa mobil ke garis finish. Fungsi ini juga tidak akan diimplementasi pada tugas ini. Untuk fungsi seleksi pada persoalan *overdrive* ini adalah sebuah algoritma yang menentukan perintah mana yang paling baik dieksekusi saat ini, perintah yang paling baik tersebut akan ditambahkan ke himpunan solusi. Kriteria "paling baik" akan dijelaskan lebih lanjut pada bagian C. Fungsi inilah yang akan diimplementasikan pada tugas ini. Fungsi ini juga akan dijelaskan lebih lanjut pada bab III bagian C. Pada permasalahan *overdrive*, fungsi kelayakan adalah fungsi yang menyatakan apakah suatu calon solusi layak menjadi solusi. Perlu diingat bahwa tidak semua calon solusi layak menjadi solusi. Contohnya, ketika mobil berada pada lintasan paling kiri maka perintah TURN_LEFT menjadi solusi yang tidak layak walaupun *game engine* tetap akan menerima perintah tersebut. Fungsi kelayakan ini tidak akan diimplementasikan secara khusus karena sudah menjadi bagian dari *game engine*. Verifikasi sebuah perintah dapat dilaksanakan atau tidak sepenuhnya menjadi tanggung jawab *game engine*. Selanjutnya fungsi objektif pada permasalahan overdrive adalah fungsi memenangkan game yang bisa dicapai dengan melewati garis finish lebih dulu.

B. Alternatif Solusi Greedy

Alternatif solusi greedy yang pertama adalah dengan membuat urutan prioritas langkah mana yang harus dieksekusi terlebih dahulu. Urutan prioritas tersebut dieksekusi menurut keadaan pemain ataupun lawan saatini. Secara garis besar alternatif solusi greedy yang pertama adalah sebagai berikut. Bila mobil rusak berat, maka perbaiki terlebih dahulu. Selanjutnya akan diperiksa bila mobil tidak rusak berat, stok powerups menipis, dan terdapat powerups yang dapat dijangkau maka akan diprioritaskan untuk mengambil powerups tersebut. Bila mobil tidak dalam keadaan rusak berat dan tidak ada powerups yang bisa diambil, maka strategi terbaiknya adalah menghindar dari rintangan yang ada.

Bila masih tidak ada rintangan di depan mobil yang ingin dihindari, maka bisa digunakan powerups seperti OIL, TWEET, EMP, ataupun BOOST.

Alternatif solusi greedy yang kedua adalah dengan membandingkan poin yang didapat bila mengeksekusi tiap-tiap perintah. Pada alternatif akan dibandingkan berapa poin yang didapat bila seandainya saat ini dieksekusi suatu perintah dengan berapa poin yang didapat bila seandainya saat ini dieksekusi suatu perintah yang lain. Perintah yang menghasilkan poin paling tinggi dianggap sebagai perintah yang paling baik untuk saat ini dan bisa ditambahkan ke himpunan solusi. Perhitungan poin untuk tiap-tiap alternatif dihitung berdasarkan berapa obstacle yang ia tabrak dan berapa powerups yang bisa ia ambil. Makin banyak obstacle yang ia tabrak, makin kecil poinnya. Makin banyak powerups yang ia ambil maka makin besar poinnya.

C. Solusi Greedy Yang Dipilih

Alternatif solusi greedy yang dipilih adalah alternatif kedua karena pertimbangan berikut. Setelah diamati dari beberapa pertandingan, alternatif pertama memiliki prioritas yang kurang fleksibel. Contohnya, bila terdapat powerups yang bisa diambil dan stok powerups tersisa sedikit, maka mobil akan memutuskan untuk mengambil *powerups* tersebut tidak peduli sebanyak apapun *damage* yang diterima. Karena itulah dipilih alternatif solusi greedy yang kedua dimana setiap *lane* yang akan dilewati akan diperhitungkan.

Pada solusi greedy yang dipilih ini, akan diperiksa terlebih dahulu apakah mobil memiliki kerusakan lebih dari 3, bila iya maka perintah FIX akan dieksekusi. Lalu diperiksa apakah kecepatan mobil saat ini 0, bila iya maka perintah ACCELERATE akan dieksekusi. Dua pemeriksaan diatas dianggap sebagai kondisi yang darurat sehingga harus cepat ditangani tanpa memperhatikan faktor lain. Bila dua kondisi di atas tidak terpenuhi, maka akan dihitung berapa poin yang akan diterima bila mobil belok kiri, belok kanan, dan tetap lurus. Yang dimaksudkan mobil tetap lurus adalah tetap lurus dengan kecepatan saat ini, tetap lurus dengan kecepatan yang dipercepat, menggunakan *powerups* lizard, ataupun menggunakan *powerups* BOOST.Untuk setiap *lane* yang akan dilewati, jenis lane tersebut akan menambah atau mengurangi poin. Penjelasan detail mengenai besar poin untuk tiap jenis *lane* akan dijelaskan pada bab empat. Bila ternyata tetap lurus merupakan pilihan yang menghasilkan poin paling tinggi, maka baru akan dipertimbangkan untuk menggunakan powerups lain seperti OIL, EMP, ataupun TWEET bergantung dengan stok yang dimiliki dan posisi lawan. *Powerups* tersebut hanya akan digunakan bila masih ada stok dan posisi lawan yang cocok sedemikian hingga serangan memiliki kemungkinan besar untuk berhasil. Kemudian akan dicek apakah kecepatan saat ini sama dengan kecepatan maksimum mobil yang berkorelasi dengan kerusakan mobil. Bila iya, maka akan dieksekusi perintah FIX agar kerusakan mobil tidak menjadi penghambat akselerasi mobil.

Bila kerusakan mobil 0 dan sudah mencapai batas maksimum kecepatannya, maka mobil akan menggunakan *powerups* dengan acak. Selain itu mobil akan mengeksekusi perintah ACCELERATE.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma Greedy

Bot ini menggunakan banyak fungsi antara untuk mengimplementasi solusi algoritma greedy yang dipilih. Untuk mengerti implementasinya lebih baik, 4 fungsi terpenting yaitu run(), getMaximumPoint(), getMaximumPointIfStraight(), dan countPoint() diimplementasikan sebagai berikut dalam *pseudocode*.

```
function run(gameState) -> command {
// fungsi bot yang dipanggil setiap ronde ketika harus memberi command ke game engine
// menerima gameState atau state dari permainan pada suatu ronde dan mengembalikan command

    Declaration:
        // mendapatkan state dari mobil bot sendiri dan state mobil lawan
        myCar = gameState.player;
        opponent = gameState.opponent;

    Algorithm:
        // jika masih menggunakan boost, prioritaskan strategi menghindari rintangan
        // perbaiki bila rusak darurat
        if ((myCar.damage == 4 && myCar.speed == 3) || myCar.damage == 5) {
            return FIX;
        }

        // tambahkan kecepatan dan jalan terlebih dahulu bila berhenti
        if(myCar.speed==0){
            return ACCELERATE;
        }

        // menghitung mana yang paling menguntungkan, belok kiri, belok kanan, atau tetap pada lane yang sama
        // memanggil fungsi seleksi untuk algoritma greedy berupa
        // fungsi getMaximumPoint() untuk mendapatkan keputusan command yang optimal lokal
        decision = getMaximumPoint(gameState);
```

```
        // mengembalikan command sesuai pilihan yang didapatkan dari fungsi seleksi
        if(decision == "LEFT"){
            return TURN_LEFT;
        }else if(decision == "RIGHT"){
            return TURN_RIGHT;
        }else if(decision == "LIZARD"){
            return LIZARD;
        }else if(decision == "BOOST"){
            return BOOST;
        } else if (decision == "ACCELERATE") {
            return ACCELERATE;
        }
        // decision "NOTHING" akan lanjut ke code yang di bawah ini

        // jika keputusan optimal pada ronde ini adalah tetap jalan lurus, maka gunakan power-up sesuai kondisi
        // pakai EMP jika memiliki EMP, lawan berada di lane sama atau sebelah, dan lawan berada di block lebih depan
        if (hasPowerUp(EMP, myCar.powerups)
            && Math.abs(myCar.position.lane - opponent.position.lane) <= 1
            && myCar.position.block < opponent.position.block) {
            return EMP;
        }

        // pakai TWEET jika memiliki TWEET, letakkan cybertruck di jalur lawan
        if (hasPowerUp(PowerUps.TWEET, myCar.powerups)){
            return TWEET(opponent.position.lane, opponent.position.block + opponent.speed + 3);
        }

        // pakai OIL jika memiliki OIL, lawan berada di belakang bot, dan jarak ke lawan kurang dari kecepatan maksimum
        if (hasPowerUp(PowerUps.OIL, myCar.powerups)
            && myCar.position.block > opponent.position.block
            && myCar.position.block - opponent.position.block < maxSpeed) {
            return OIL;
        }

        // jika tidak ada kondisi optimal untuk menggunakan power-up di atas,
        // maka lakukan perbaikan opsional agar boost / accelerate tidak sia sia
        if ((myCar.damage == 1 && myCar.speed == 9)
            || (myCar.damage == 2 && myCar.speed == 8)
            || (myCar.damage == 3 && myCar.speed == 6)) {
            return FIX;
        }
```

```
        // jika tidak perlu melakukan perbaikan dan kecepatan sudah maksimal,
        // maka gunakan power-up apapun yang ada untuk menambah poin
        if (myCar.speed >= maxSpeed) {
            //pakai EMP
            if (hasPowerUp(PowerUps.EMP, myCar.powerups)){
                return EMP;
            }
            //pakai TWEET
            if (hasPowerUp(PowerUps.TWEET, myCar.powerups)){
                return new TweetCommand(opponent.position.lane, opponent.position.block + opponent.speed + 3);
            }
            //pakai OIL
            if (hasPowerUp(PowerUps.OIL, myCar.powerups)){
                return OIL;
            }
            //pakai LIZARD
            if (hasPowerUp(PowerUps.LIZARD, myCar.powerups)){
                return LIZARD;
            }
        }

        // default comman adalah accelerate
        return ACCELERATE;
    }
```

```
function getMaximumPoint(gameState) -> decision {
// fungsi seleksi yang dipanggil dalam fungsi run untuk mengambil keputusan berdasarkan algoritma greedy
// menerima state ronde dalam gameState dan mengembalikan keputusan yang optimum lokal
    Declaration:
        block = gameState.player.position.x;    // mendapatkan posisi di sumbu x dari mobil bot
        currentSpeed = gameState.player.speed;  // mendapatkan kecepatan mobil bot
        lane = gameState.player.position.y;     // mendapatkan lane mobil bot dari sumbu y
        map = gameState.lanes;                  // mendapatkan data map dari gameState
        startBlock = 0;                         // mendapatkan posisi awal dari map
        current = map[lane];                    // mendapatkan data dari lane yang ditempati mobil bot
        point_current = countPoint(current);    // menilai poin dari data lane yang ditempati

    Algorithm:
        // jika mobil berada pada lane paling atas/kiri
        if (lane == 1) {
            // mendapatkan data lane sebelah kanan
            right = map[lane + 1];
            // mengevaluasi lane tersebut dengan fungsi penilai poin
            point_right = countPoint(right);
            // jika poin dari lane sebelah kanan lebih tinggi, maka keputusan adalah belok kanan
            if (point_right > point_current) {
                return "RIGHT";
            } else {
                // jika tidak, pertimbangkan apa yang dilakukan jika lurus
                return getMaximumPointIfStraight(gameState);
            }
```

```
        // jika mobil berada pada lane paling bawah/kanan
        } else if (lane == 4) {
            // mendapatkan data lane sebelah kiri
            left = map[lane + 1];
            // mengevaluasi lane tersebut dengan fungsi penilai poin
            point_left = countPoint(left);
            // jika poin dari lane sebelah kiri lebih tinggi, maka keputusan adalah belok kiri
            if (point_left > point_current) {
                return "LEFT";
            } else {
                // jika tidak, pertimbangkan apa yang dilakukan jika lurus
                return getMaximumPointIfStraight(gameState);
            }
        // jika mobil berada pada 2 lane tengah
        } else {
            // mendapatkan data lane kiri dan kanan
            // lalu mengevaluasi nilai poin keduanya
            right = map[lane - 1];
            left = map[lane + 1];
            point_left = countPoint(left);
            point_right = countPoint(right);
            // jika poin dari lane kiri lebih tinggi, maka keputusan adalah belok kiri
            if (point_left > point_current && point_left > point_right) {
                return "LEFT";
            // jika poin dari lane kanan lebih tinggi, maka keputusan adalah belok kanan
            } else if (point_right > point_current && point_right > point_left) {
                return "RIGHT";
            // jika tidak, pertimbangkan apa yang dilakukan jika lurus
            } else {
                return getMaximumPointIfStraight(gameState);
            }
        }
    }
```

```
function getMaximumPointIfStraight(gameState) -> decision {
// fungsi untuk mengambil keputusan berdasarkan algoritma greedy jika mobil tidak belok kanan atau kiri
    Declaration:
        // inisialisasi perhitungan bobot poin setiap command
        point_nothing = 0;
        point_boost = 0;
        point_accelerate = 0;
        point_lizard = 0;
        available = gameState.player.powerups;  // mendapatkan data power-up yang dimiliki bot
        block = gameState.player.position.x;    // mendapatkan posisi di sumbu x dari mobil bot
        currentSpeed = gameState.player.speed;  // mendapatkan kecepatan mobil bot
        lane = gameState.player.position.y;     // mendapatkan lane mobil bot dari sumbu y
        map = gameState.lanes;                  // mendapatkan data map dari gameState
        startBlock = 0;                         // mendapatkan posisi awal dari map

    Algorithm:
        // jika tidak memiliki BOOST maka kurangi bobot poin untuk command BOOST agar tidak digunakan
        if (!hasPowerUp(BOOST, available)) {
            point_boost -= 100;
        }

        // jika tidak memiliki LIZARD maka kurangi bobot poin untuk command LIZARD agar tidak digunakan
        if (!hasPowerUp(LIZARD, available)) {
            point_lizard -= 100;
        }

        // tambahkan poin untuk tidak menjalankan command,
        // dengan menghitung bobot poin jalan saat lurus tanpa percepatan
        point_nothing += countPoint(map[lane]));
```

```
        // jika bot sedang menggunakan boost, kurangi bobot poin untuk command BOOST agar tidak digunakan
        point_boost -= gameState.player.boosting ? 100 : 0;
        // tambahkan bobot poin untuk boost untuk setiap powerup boost yang dimiliki, agar segera digunakan
        point_boost += countPowerups(available, PowerUps.BOOST) * 2;
        // tambahkan bobot poin untuk boost dari perhitungan poin jika jalan lurus dengan kecepatan boost
        point_boost += countPoint(map[lane]);

        // jika kecepatan bot sudah maximum dengan kondisi bot,
        // kurangi bobot untuk command accelerate agar tidak digunakan
        point_accelerate -= (currentSpeed >= getDamagedMaxSpeed(gameState.player.damage)) ? 100 : 0;
        // tambahkan poin untuk accelerate dari perhitungan poin jika jalan lurus dengan percepatan tambahan;
        point_accelerate += countPoint(map[lane]);

        // tambahkan poin untuk LIZARD jika digunakan di jalur lurus
        point_lizard += countPoint(map[lane]);

        // dapatkan nilai poin tertinggi dari semua command
        max_point = max(max(point_nothing, point_accelerate), max(point_lizard, point_boost));

        // kembalikan command dengan nilai poin tertinggi
        if (max_point == point_nothing) {
            return "NOTHING";
        } else if (max_point == point_accelerate) {
            return "ACCELERATE";
        } else if (max_point == point_lizard) {
            return "LIZARD";
        } else {
            return "BOOST";
        }
    }
```

```
function countPoint(lanes) -> int {
    Declaration:
        // inisialisasi perhitungan poin
        count = 0;

    Algorithm:
        // untuk setiap lane yang dipertimbangkan, hitung poinnya
        for (lane in lanes) {
            // kurangi poin jika terdapat mud, oil, wall, atau cybertruck pada lane
            if (lane.terrain = MUD){
                count -= 2;
            } else if (lane.terrain = OIL_SPILL){
                count -= 2;
            } else if (lane.terrain = WALL)){
                count -= 5;
            } else if (lane.terrain = CyberTruck){
                count -= 10;
            // tambahkan poin jika terdapat powerup oil, boost, lizard, tweet, atau emp pada lane
            } else if (lane.terrain = OIL_POWER)){
                count += 1;
            } else if (lane.terrain = BOOST)){
                count += 1;
            } else if (lane.terrain = LIZARD)){
                count += 1;
            } else if (lane.terrain = TWEET)){
                count += 1;
            } else if (lane.terrain = EMP)){
                count += 1;
            }
        }
        // kembalikan nilai poin
        return count;
}
```

B. Penjelasan Struktur Data

Struktur data yang digunakan pada program Overdrive secara umum terdiri dari command, entities, enums, serta dua kelas utama yaitu Bot.Java yang berisi implementasi algoritma greedy dan Main.java yang merupakan program utama.

1. Command

Di dalam command, terdapat sebelas kelas yang merupakan perintah yang akan dikirimkan bot ke dalam game engine untuk dieksekusi. Berikut adalah penjelasan singkat tiap kelas yang ada:

a. AccelerateCommand.java, perintah untuk menambah kecepatan mobil.
b. BoostCommand.java, perintah untuk menggunakan *boost*.
c. ChangeLaneCommand.java, perintah untuk berpindah jalur ke kanan atau ke kiri.
d. Command.java, merupakan *parent* dari semua kelas *command*.
e. DecelerateCommand.java, perintah untuk mengurangi kecepatan mobil.
f. DoNothingCommand.java, tidak melakukan aksi apa-apa.
g. EmpCommand.java, perintah untuk menembakkan ledakan EMP ke depan mobil.
h. FixCommand.java, perintah untuk memperbaiki mobil.
i. LizardCommand.java, perintah untuk membuat mobil melompat untuk menghindari *lizard*.
j. OilCommand.java, perintah untuk menjatuhkan *oil* di bawah lintasan mobil saat ini.
k. TweetCommand.java, perintah untuk memunculkan truk cyber.

2. Entities

Di dalam entities, terdapat empat kelas yang merupakan objek yang ada pada game Overdrive. Berikut adalah penjelasan singkat tiap kelas yang ada:

a. Car.java, menyimpan informasi tentang mobil yang tediri atas atribut id, position, speed, damage, state, powerups, boosting, dan boostCounter.
b. GameState.java, menyimpan informasi tentang status game yang sedang berjalan saat ini yang terdiri dari atribut currentRound, maxRounds, player, opponent, dan lanes.
c. Lane.java, menyimpan informasi tentang objek yang ada pada tiap state tertentu pada game yang terdiri dari atribut position, terrain, dan occupiedByPlayerId.
d. Position.java, menyimpan informasi posisi mobil saat ini dalam koordinat x dan y (atribut block dan lane).

3. Enums

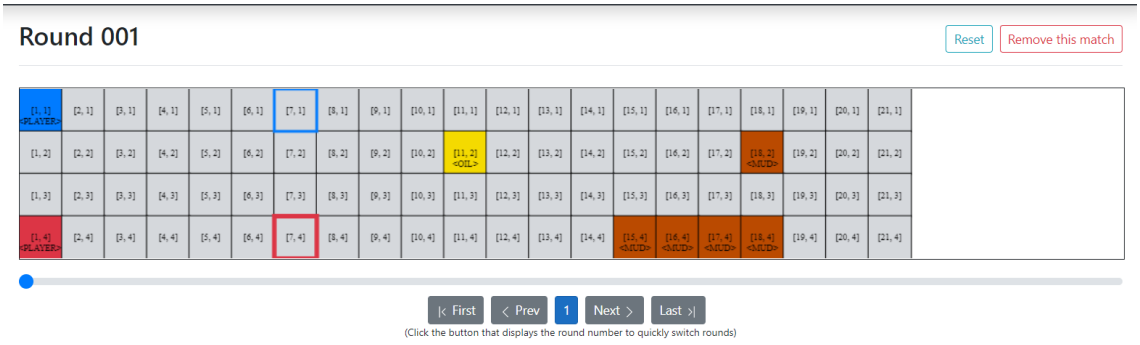
Di dalam enums, terdapat empat kelas yang menyimpan variabel tertentu menjadi suatu tipe data konstanta yang bertujuan agar pemanggilannya lebih mudah dari kelas lain.

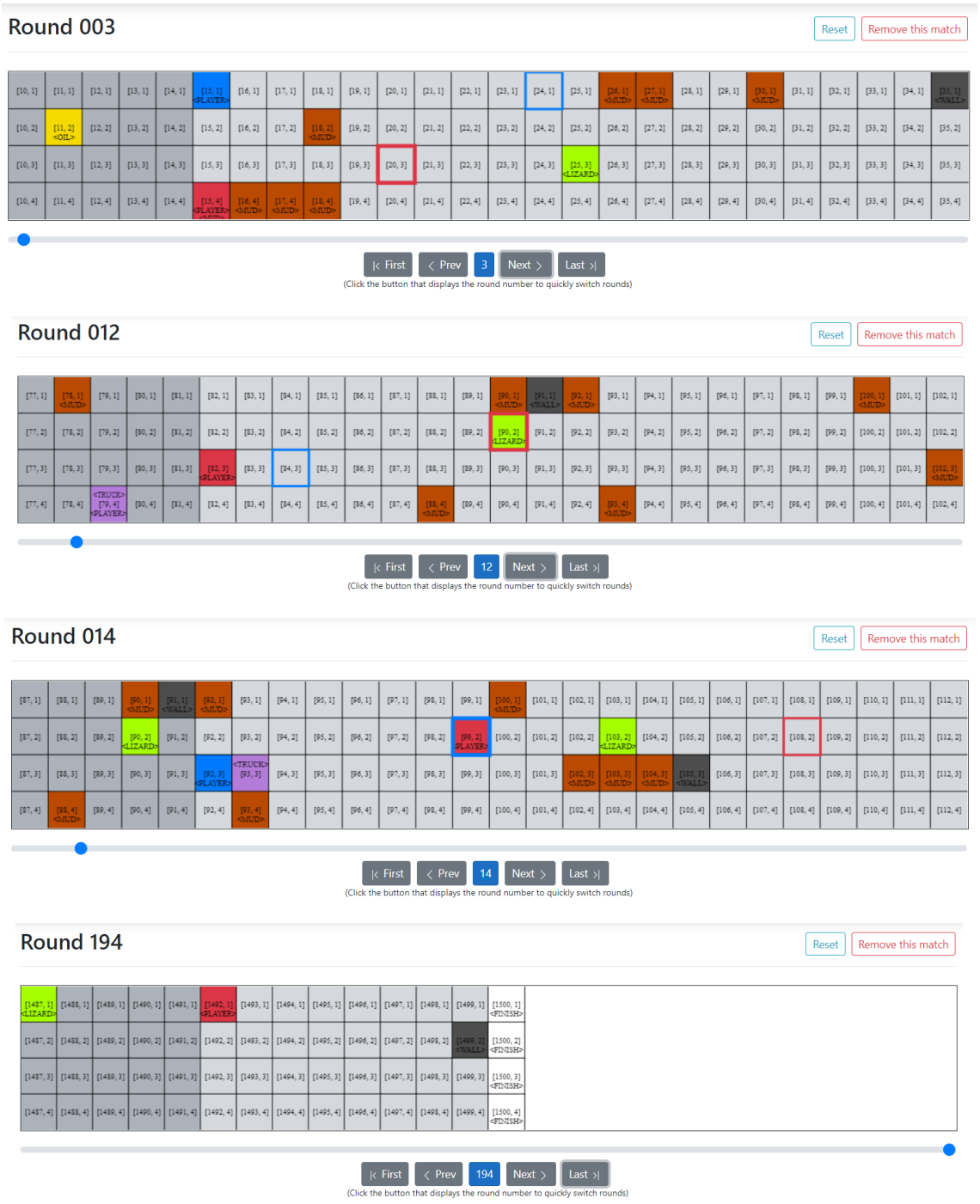
a. Direction.java, berisi konstanta sebagai representasi arah gerak mobil yaitu FORWARD, BACKWARD, LEFT, dan RIGHT.
b. PowerUps.java, berisi konstanta yang menyimpan *powerups* yang bisa digunakan mobil dalam permainan.
c. State.java, berisi konstanta yang menyimpan status dari mobil di dalam permainan yang sedang berlangsung.
d. Terrain.java, berisi konstanta yang menyimpan objek-objek yang terdapat dalam game.

4. **Bot.java**
   Kelas Bot merupakan kelas implementasi utama dimana semua strategi (algoritma greedy) dituliskan. Beberapa method bisa ditambahkan dan akan dipanggil dari method run.
5. **Main.java**
   Kelas Main merupakan kelas utama yang akan diproses oleh game engine untuk memuat dalam lintasan, state, dan bot yang akan ditandingkan.

C. Analisis Desain Solusi Algoritma Greedy

Berikut adalah beberapa tangkapan layar dari visualisasi pengujian program:





Dari beberapa pengujian yang telah dilakukan, algoritma greedy yang diterapkan pada bot menghasilkan keputusan yang optimal pada banyak kasus. Dari gambar di atas, terlihat bot telah memberikan perintah ke game engine sesuai dengan algoritma greedy yang diimplementasikan pada bot tersebut. Misalnya pad Round 001, karena kondisi lintasan yang tidak terdapat halangan, maka bot akan mengutamakan penambahan kecepatan jika belum maksimal yaitu dengan perintah

ACCELERATE. Pada Round 003, terlihat beberapa objek MUD yang menghalangi mobil sehingga bot akan memperhitungkan keputusan yang paling optimal apakah berbelok atau tetap pada lintasan tersebut dan hasilnya bot memberi perintah TURN_LEFT sesuai yang diharapkan. Pada Round 012, kerja algoritma greedy lebih terlihat dimana mobil bergerak ke arah kiri, padahal tidak ada halangan apapun jika mobil tetap berada di jalurnya. Keputusan tersebut dihasilkan karena perpindahan jalur ke kiri memberikan hasil yang lebih optimal karena mendapatkan satu ability yang terhitung sebagai point. Di Round 014, pada saat mobil memiliki ability dan posisi mobil yang cukup aman (tidak banyak rintangan), bot mengutamakan penggunaan *powerup* TWEET untuk menghalangi lintasan musuh.

Pada kasus tertentu, bot belum berhasil memberikan perintah yang paling efektif. Contohnya ketika sudah hampir sampai di garis finish dan mobil memiliki *damage* tertentu, kadang bot memberikan perintah FIX, padahal kecepatan mobil masih cukup untuk sampai ke garis finish dalam satu putaran. Meskipun cukup jarang terjadi, mobil lawan bisa saja mendahului dan terlebih dahulu sampai di garis finish pada putaran tersebut, yang awalnya berada di posisi kalah.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Dari pengerjaan Tugas Besar 1 Strategi Algoritma, kami telah berhasil membuat strategi atau algoritma greedy dan mengimplementasikannya ke dalam sebuah bot dalam permainan Overdrive. Bot yang dibuat dapat memberikan perintah yang optimal ke dalam game engine untuk memenangkan permainan ini. Strategi *greedy* yang dipakai adalah dengan menghitung keuntungan yang paling optimal jika melewati setiap lane tertentu.

B. Saran

1. Pemberian poin untuk tiap jenis blok perlu diperhitungkan kembali untuk mencari poin yang proporsional untuk masing-masing jenis blok
2. Sebaiknya perhitungan poin melibatkan beberapa variabel lain seperti seberapa jauh mobil akan melaju, kecepatan mobil di giliran selanjutnya, ataupun kecepatan lawan di giliran selanjutnya
3. Perlunya sistem pemberian poin yang terpisah saat mobil sedang menggunakan BOOST agar memprioritaskan penghindaran rintangan. Hal tersebut sebaiknya dilakukan supaya penggunaan powerups BOOST tidak sia-sia

DAFTAR PUSTAKA

1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

2. https://brilliant.org/wiki/greedy-algorithm/

REPOSITORY

https://github.com/rizkyramadhana26/Tubes-Stima-1