TUGAS BESAR I IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2021/2022

PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN “OVERDRIVE”

Disusun Oleh Kelompok:

Kelompok 36 “Usada Pawapua Steel”

Andreas Indra Kurniawan
13520091

Thirafi Najwan Kurniatama
13520157

Farrel Farandieka Fibriyanto
13520054

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

17 FEBRUARI 2022

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.

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 <lane> <block>

j. USE_EMP

k. FIX

5. Command dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.

6. Bot pemain yang pertama kali mencapai garis finish akan memenangkan pertandingan. Jika kedua bot mencapai garis finish secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar. Adapun peraturan yang lebih lengkap dari permainan Overdrive, dapat dilihat pada laman :https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md

Bab 2

Landasan Teori

2.1 Algoritma Greedy

Algoritma Greedy merupakan salah satu metode yang paling populer untuk menyelesaikan persoalan optimasi. Persoalan optimasi sendiri merupakan suatu persoalan untuk mencari solusi optimal. Terdapat dua jenis persoalan optimasi, yaitu maksimasi (maximization) dan minimasi (minimization).

Algoritma Greedy merupakan algoritma yang selalu mengambil solusi terbaik yang dapat diperoleh pada waktu itu tanpa memperhatikan konsekuensi ke depan. Algoritma greedy dapat mencari solusi optimal global untuk beberapa permasalahan optimisasi, tetapi hanya akan mencari solusi yang kurang optimal untuk beberapa permasalahan lainnya.

Dalam algoritma greedy, terdapat beberapa elemen:

1. Himpunan Kandidat (C)
   himpunan yang berisi kandidat yang akan dipilih pada setiap langkah
2. Himpunan Solusi (S)
   himpunan berisi kandidat yang sudah dipilih
3. Fungsi Solusi
   fungsi untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi Seleksi (*selection function*)
   fungsi untuk memilih kandidat berdasarkan strategi greedy tertentu
5. Fungsi Kelayakan (*feasibility function*)
   fungsi untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi
6. Fungsi Obyektif
   fungsi untuk memaksimumkan atau meminimumkan

2.2 Permainan Overdrive

Permainan Overdrive memiliki beberapa program yang berjalan ketika permainan sedang berjalan:

1. Game Engine, yang berfungsi untuk menegakkan peraturan-peraturan yang telah ditetapkan dengan mengecek apakah peraturan tiap bot valid atau tidak.
2. Game Runner, yang berfungsi untuk mengadu dua bot dengan meminta perintah yang dikeluarkan tiap bot dan memberikannya kepada Game Engine
3. Bot, yang berfungsi untuk mengeluarkan perintah permainan sesuai dengan hasil pemograman pembuatnya berdasarkan informasi permainan yang diberikan.

Pemain dapat mengubah logika-logika dalam bot sehingga bot tersebut akan berperilaku sesuai dengan yang pemain inginkan. Pemain dapat memilih bahasa pemograman apa yang ingin dipakai untuk membuat botnya dan menggunakan template yang telah disediakan di folder starter-bot

Untuk menjalankan permainan dengan pengaturan default, kita bisa menjalankan file run.bat, dengan ini akan terlihat suatu console dengan output yang dikeluarkan oleh Game Engine. Pada ujung permainan akan diberikan simpulan tiap pertandingan dan diberikan penjelasan siapa pemenang pada permainan tersebut.

Untuk menentukan bot mana yang dipakai dalam permainan, yang menjalankan program dapat mengubah folder directory bot di dalam file game-runner-config.json di dalam folder utama permainan. Kolom ‘player-a’ dan ‘player-b’ menjelaskan lokasi folder yang memiliki .jar bot untuk player-a dan player-b secara berurutan. Kolom ini lah yang perlu diubah apabila ingin menjalankan bot yang berbeda.

Bab 3

Aplikasi Strategy Greedy

3.1 Mapping Persoalan

Berikut merupakan pemetaan permasalahan permainan Worms secara umum menjadi elemen-elemen penyusun algoritma greedy:

Elemen Algoritma Greedy	Pemetaan pada Overdrive
Himpunan Kandidat	perintah *NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET Y X, USE_LIZARD* pada permainan
Himpunan Solusi	perintah *NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET Y X, USE_LIZARD* pada permainan yang diatur sehingga membuat mobil kita memenangkan permainan.
Fungsi Solusi	Fungsi untuk menentukan apakah perintah yang dikirimkan bot merupakan perintah yang terdefinisi atau tidak dalam permainan. Karena apabila tidak terdefinisi maka perintah bukanlah solusi yang valid
Fungsi Seleksi	Memilih perintah sesuai dengan prioritas perintah yang telah ditetapkan dalam bot yang dibuat. Prioritas perintah dalam bahasa pemograman dapat dilakukan dengan membuat fungsi penyeleksian yang terurut menurut dengan prioritas strategi yang dimiliki.
Fungsi Kelayakan	Memeriksa bahwa semua kondisi yang diperlukan untuk mengirim perintah tertentu sudah terpenuhi. Pemeriksaan dapat dilakukan terhadap posisi mobil sendiri dan mobil lawan, kecepatan mobil, damage mobil, power up yang dimiliki mobil, serta obstacles yang ada di lane di depan mobil.
Fungsi Obyektif	Memastikan mobil akan memenangkan pertandingan dengan memaksimalkan beberapa hal

	yaitu mencapai garis finish terlebih dahulu, jarak yang ditempuh, kecepatan yang dilaju, serta skor yang didapatkan.

3.2 Alternatif-Alternatif Solusi Greedy

3.2.1 Memaksimalkan Lane Weight

Lane weight dapat didefinisikan sebagai nilai dari sebuah lane. Ketika lane memiliki *weight* yang besar maka lane tersebut akan lebih dipilih oleh mobil. Untuk setiap power yang berada di lane, *weight* dari lane akan bertambah sedangkan untuk setiap *obstacle* yang ada di lane akan mengurangi *weight* dari lane. Strategi ini akan mengecek *weight* dari 3 lane(lane kiri,lane mobil, lane kanan jika ada) lalu *weight* dari setiap lane akan dibandingkan sebelum melakukan aksi. Setiap *obstacle* dan *power* memiliki berat yang berbeda di *lane* agar *risk and reward* yang diambil merupakan yang terbaik.

3.2.2 Memaksimalkan Waktu di mana Kecepatan Mobil Maksimum

Kecepatan normal mobil biasanya merupakan 9 *lane per round* (LPR), tetapi dengan boost kita bisa mendapatkan kecepatan maksimum 15 LPR selama 5 ronde. Tidak ada prasyarat lain untuk menggunakan boost kecuali power up boost sudah dimiliki mobil. Dengan ini kita tidak perlu meminimalisasi damage mobil, cukup mengirim pesan fix sehingga mobil tidak akan berhenti total (damage = 5). Strategi ini akan memprioritaskan belok ke jalur yang terdapat power up boost sehingga dapat memakai boost secara terus menerus.

3.2.3 Memaksimalkan Kecepatan Mobil Rata-Rata

Strategi ini memprioritaskan rata-rata kecepatan mobil berada di sekitar 9 RPL, agar kecepatan mobil rata-rata merupakan kecepatan mobil maksimum lokal. Agar kecepatan mobil dapat berada di ambang 9, mobil perlu memiliki nilai damage 0 sehingga strategi ini juga akan meminimalisasi nilai damage mobil. Setelah mobil memiliki damage 0, kita dapat fokus memaksimalkan kecepatan lokal mobil. Setelah itu strategi fokus untuk menjaga kecepatan tersebut dengan menghindari semua obstacle dalam lane mobil.

3.2.4 Meminimalisasi Kecepatan Lawan

Strategi ini fokus pada penyerangan agar mobil lawan tidak bisa menyaingi kecepatan mobil kita. Strategi ini dilakukan dengan cara mengambil power penyerangan sebanyak mungkin dan terus menggunakan power menyerang ketika power tersedia. Jika power sedang tidak tersedia maka mobil akan melakukan accelerate. Pada strategi ini yang perlu diperhatikan adalah urutan penggunaan accelerate, turn, dan power. Penggunaannya harus seimbang agar mobil bisa tetap maju.

3.3 Analisis Efisiensi

Efisiensi dari algoritma kami dalam satu pengambilan keputusan command adalah O(Speed) dengan Speed adalah kecepatan mobil kami pada ronde tersebut. Jika dikalkulasi untuk keseluruhan game, maka algoritma kami berjalan pada O(Speed * Round) dengan Round adalah jumlah ronde yang terjadi pada game tersebut.

3.4 Analisis Efektivitas

Untuk alternatif solusi memaksimalkan lane weight, didapatkan cukup efektif dibandingkan dengan solusi lainnya. Dalam percobaan kami, solusi ini akan memberikan kondisi di mana mobil akan mempertahankan kecepatan yang tinggi (di sekitar 9 RPL) serta mendapatkan power-up dalam frekuensi yang lumayan sering. Solusi ini kami lihat memiliki keseimbangan agresif (dalam arti memiliki kesempatan menyerang lawan) dan pasif (dalam arti memfokuskan dalam perkembangan mobil sendiri) yang cukup seimbang.

Untuk alternatif solusi memaksimalkan waktu kecepatan mobil maksimum, didapatkan strategi ini kurang efektif. Dalam percobaan kami, untuk mendapatkan power up boost sebanyak mungkin, bot akan memilih lane yang memiliki banyak obstacles. Oleh karena itu bot biasanya memiliki nilai damage yang lumayan tinggi sehingga hanya memiliki kecepatan yang rendah.

Untuk alternatif solusi memaksimalkan kecepatan mobil rata-rata, kami dapatkan strategi ini cukup baik. Dalam percobaan kami, bot meminimalisasi terjadinya tabrakan dengan obstacles sehingga bot memiliki kecepatan rata-rata 9. Karena bot jarang menabrak obstacles, bot juga meminimalisasi ronde yang dibutuhkan untuk mengirim perintah FIX sehingga bot jarang perlu diam dalam suatu ronde untuk memperbaiki mobil sendiri. Solusi ini kami lihat sudah bagus apabila lawan kami memiliki strategi pasif yang sama, karena tidak melakukan perlawanan terhadap lawan yang tidak bertindak pasif.

Untuk alternatif solusi meminimalisasi kecepatan lawan, kami dapatkan strategi ini cukup baik. Dalam percobaan kami, bot berhasil memperlambat laju mobil lawan dengan mengambil semua power up yang dapat menyerang mobil lawan. Akan tetapi, kami juga dapatkan bahwa mobil menabrak cukup banyak rintangan dalam tujuan mengambil power up ofensif sebanyak mungkin. Karena itu mobil memiliki rata-rata kecepatan yang rendah dan rata-rata damage yang lumayan tinggi. Solusi ini bisa dibilang cukup baik untuk lawan yang hanya bersifat pasif, tetapi hasil akhir untuk solusi ini bisa dibilang bersifat *coin flip* atau 50:50 meskipun lawan yang cukup dasar (bersifat pasif, tidak ada parameter lain).

3.5 Solusi yang dipilih

Solusi yang dipilih pada tugas ini merupakan gabungan dari 3 solusi yang ditawarkan yaitu memaksimalkan lane weight, memaksimalkan waktu di mana kecepatan mobil maksimum, serta meminimasi kecepatan lawan. Solusi yang didahulukan adalah memaksimalkan waktu ketika kecepatan mobil maksimum. Hal ini diterapkan dengan cara mengecek kecepatan mobil serta mengecek apakah di depan mobil ada *obstacle* atau tidak, jika ada obstacle maka akan digunakan powerup lizard. Selain itu bot juga akan diusahakan untuk terus melakukan boost jika tersedia dan boost akan efektif jika digunakan.

Solusi meminimasi kecepatan lawan merupakan prioritas kedua ketika memaksimalkan kecepatan tidak bisa dilakukan. Power yang diutamakan untuk digunakan adalah EMP, tweet, serta oil secara berurutan. Sebelum menggunakan power juga akan dicek terlebih dahulu apakah di depan mobil ada *obstacle* sehingga mobil tidak akan menabrak apapun ketika menggunakan power. Power oil juga hanya akan digunakan ketika bot sedang memimpin sehingga tidak akan membuang move dan power secara sia-sia.

Solusi memaksimalkan *lane weight* dipilih karena dengan menggunakan *lane weight* kita dapat melihat “nilai” dari tiap lane sehingga kita tidak akan melakukan pergerakan yang sia-sia. *Lane weight* juga bisa sekaligus menghindari *obstacle* yang ada. Dengan begitu bot bisa melakukan pergerakan yang paling efektif oleh karena itu solusi ini digunakan untuk setiap akan dilakukan pergerakan.

Bab 4

Implementasi dan Pengujian

Source code yang dijelaskan di bab ini tersedia di https://github.com/reverseon/Tubes-1-IF2110-2021-Overdrive pada direktori ./src/starter-bots/java/src/main/java/za/co/entelect/challenge/Bot.java

4.1 Implementasi Algoritma Greedy

Implementasi algoritma greedy diimplementasikan pada file bot.java. Di dalam file tersebut terdapat beberapa fungsi dengan fungsi run sebagai fungsi utamanya.

4.1.1 public Command run(GameState gameState)

```
    function run(GameState gameState)→Command

    {Fungsi utama yang digunakan untuk menentukan command yang akan digunakan}



    Kamus Lokal

    myCar,opCar : Car

    curLaneW, truck, wallo, opb, mud, oilsp, mx, my, ox, oy, oSpd, desiredX, desiredY,
rightLaneW, lizardW, leftLaneW : Integer

    curLane, blockIFBoost, rightLane, leftLane : List<Object>

    posiblane : ArrayList<Integer>



    Algoritma

          Car myCar = gameState.player;

         Car opCar = gameState.opponent;

         // OBVIOUS AND IMMEDIATE ACTION

         if (myCar.damage > 3) return FIX; // Middle Ground

         if (myCar.speed < 3) return ACCELERATE; //  Accelerate to prevent
0 speed stuck
```

```
        List<Object> curLane = getBlocksInFront(myCar.position.lane,
myCar.position.block, myCar.speed, gameState, false);

        Integer curLaneW = laneWeight(curLane);

        if (myCar.speed == 15 && hasPowerUp(PowerUps.LIZARD,
myCar.powerups) && hasObstacles(curLane)) return LIZARD;

        if (!hasObstacles(curLane) && myCar.damage == 3) return FIX;



        // POWERUPS (EXCEPT LIZARD) DECISION HERE

        // BOOST -> when not and leading

        if (hasPowerUp(PowerUps.BOOST, myCar.powerups) && !myCar.boosting)
{

            List<Object> blockIFBoost =
getBlocksInFront(myCar.position.lane, myCar.position.block,
maxSpd(myCar.damage), gameState, false);

            Integer truck = cf(blockIFBoost, Terrain.TRUCK);

            Integer wallo = cf(blockIFBoost, Terrain.WALL);

            Integer opb = cf(blockIFBoost, Terrain.OPPONENT);

            Integer mud = cf(blockIFBoost, Terrain.MUD);

            Integer oilsp = cf(blockIFBoost, Terrain.OIL_SPILL);

            if (!hasObstacles(blockIFBoost)) {

                if (myCar.damage == 2 || myCar.damage == 1) return FIX;

                else return BOOST;

            } else if (myCar.damage > 2 && (mud + oilsp <= 1 ) &&
(truck+wallo+opb == 0) && (Math.abs(maxSpd(myCar.damage-1) - myCar.speed)
> 3)) {

                return BOOST;

            }
```

```
        }


        if (myCar.speed <= 3 &&
!hasObstacles(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) return ACCELERATE;


        if (!hasObstacles(curLane)) { // prioritize change lane / accel
first if speed is too low before using power ups
            // EMP -> when not leading
            if (hasPowerUp(PowerUps.EMP, myCar.powerups)) {
                ArrayList<Integer> posibLane = new ArrayList<>();
                if (myCar.position.lane == 1) {posibLane.add(1);
posibLane.add(2);}
                else if (myCar.position.lane == 4) {posibLane.add(3);
posibLane.add(4);}
                else {posibLane.add(myCar.position.lane);
posibLane.add(myCar.position.lane-1);
posibLane.add(myCar.position.lane+1);}
                if (myCar.position.block < opCar.position.block &&
posibLane.contains(opCar.position.lane)) {
                    return EMP;
                }
            }
            // TWEET -> when not and leading
            if (hasPowerUp(PowerUps.TWEET, myCar.powerups)) {
                int mx = myCar.position.block;
                int my = myCar.position.lane;
```

```
                int ox = opCar.position.block;
                int oy = opCar.position.lane;
                int oSpd = opCar.speed;
                int desiredX = ox + (nextSpdState(oSpd) == 50 ? 15 :
nextSpdState(oSpd)) + 1;
                int desiredY = oy;
                if (desiredY != my) {
                    return TWEET(desiredX, desiredY);
                }
            }
            // OIL -> when leading
            if (hasPowerUp(PowerUps.OIL, myCar.powerups)) {
                if (myCar.position.block > opCar.position.block &&
myCar.position.lane == opCar.position.lane) {
                    return OIL;
                }
            }
        }

        // LANE CHANGING DECISION
        if (myCar.position.lane == 1) {
            // can only turn right
            List<Object> rightLane =
getBlocksInFront(myCar.position.lane+1, myCar.position.block,
myCar.speed-1, gameState, true);
            Integer rightLaneW = laneWeight(rightLane);
```

```
            Integer lizardW = laneWeight(curLane.subList(curLane.size()-1,
curLane.size()));

            if (curLaneW < lizardW && rightLaneW < lizardW &&
hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {

                return LIZARD; // Two Lane aren't worth it and has a
Lizard, use LIZARD TO JUMP

            }

            if (curLaneW >= rightLaneW) { // BEST CONTENDER -> ACCELERATE,
DO NOTHING

                // CHECK IF ITS WORTH IT TO ACCELERATE

                if (nextSpdState(myCar.speed) != 50 /* ERROR CODE */ &&
curLaneW <= laneWeight(getBlocksInFront(myCar.position.lane,
myCar.position.block, nextSpdState(myCar.speed), gameState, false))) {

                    if (myCar.speed == maxSpd(myCar.damage) &&
!hasObstacles(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) {

                        return FIX;

                    }



                    return ACCELERATE; // WHY NOT

                } else if (prevSpdState(myCar.speed) != 50 && curLaneW <
laneWeight(getBlocksInFront(myCar.position.lane, myCar.position.block,
prevSpdState(myCar.speed), gameState, false))) {

                    return DECELERATE;

                } else {

                    if (nextSpdState(myCar.speed) == 50) {

                        return ACCELERATE;

                    }
```

```
                    return DO_NOTHING;

                }

            } else {

                    return TURN_RIGHT; // BETTER TURN RIGHT NOW

            }

        } else if (myCar.position.lane == 4) {

            // can only turn left

            List<Object> leftLane =
getBlocksInFront(myCar.position.lane-1, myCar.position.block,
myCar.speed-1, gameState, true);

            Integer leftLaneW = laneWeight(leftLane);

            Integer lizardW = laneWeight(curLane.subList(curLane.size()-1,
curLane.size()));

            if (curLaneW < lizardW && leftLaneW < lizardW &&
hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {

                return LIZARD; // Two Lane aren't worth it and has a
Lizard, use LIZARD TO JUMP

            }

            if (curLaneW >= leftLaneW) { // BEST CONTENDER -> ACCELERATE,
DO NOTHING

                // CHECK IF ITS WORTH IT TO ACCELERATE OR DECELERATE

                if (nextSpdState(myCar.speed) != 50 && curLaneW <=
laneWeight(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) {

                    if (myCar.speed == maxSpd(myCar.damage) &&
!hasObstacles(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) {

                        return FIX;
```

```
                    }

                    return ACCELERATE; // WHY NOT

                } else if (prevSpdState(myCar.speed) != 50 && curLaneW <
laneWeight(getBlocksInFront(myCar.position.lane, myCar.position.block,
prevSpdState(myCar.speed), gameState, false))) {

                    return DECELERATE;

                } else {

                    if (nextSpdState(myCar.speed) == 50) {

                        return ACCELERATE;

                    }

                    return DO_NOTHING;

                }

            } else {

                return TURN_LEFT; // BETTER TURN LEFT NOW

            }



        } else {

            // can be both

            List<Object> rightLane =
getBlocksInFront(myCar.position.lane+1, myCar.position.block,
myCar.speed-1, gameState, true);

            List<Object> leftLane =
getBlocksInFront(myCar.position.lane-1, myCar.position.block,
myCar.speed-1, gameState, true);

            Integer rightLaneW = laneWeight(rightLane);

            Integer leftLaneW = laneWeight(leftLane);
```

```
            Integer lizardW = laneWeight(curLane.subList(curLane.size()-1,
curLane.size()));

            if (curLaneW < lizardW && leftLaneW < lizardW && rightLaneW <
lizardW && hasPowerUp(PowerUps.LIZARD, myCar.powerups)) {

                return LIZARD; // Three Lane aren't worth it and has a
Lizard, use LIZARD TO JUMP

            }

            if (curLaneW >= rightLaneW && curLaneW >= leftLaneW) { // BEST
CONTENDER -> ACCELERATE, DO NOTHING

                if (nextSpdState(myCar.speed) != 50 && curLaneW <=
laneWeight(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) {

                    if (myCar.speed == maxSpd(myCar.damage) &&
!hasObstacles(getBlocksInFront(myCar.position.lane, myCar.position.block,
nextSpdState(myCar.speed), gameState, false))) {

                        return FIX;

                    }

                    return ACCELERATE; // WHY NOT

                } else if (prevSpdState(myCar.speed) != 50 && curLaneW <
laneWeight(getBlocksInFront(myCar.position.lane, myCar.position.block,
prevSpdState(myCar.speed), gameState, false))) {

                    return DECELERATE;

                } else {

                    if (nextSpdState(myCar.speed) == 50) {

                        return ACCELERATE;

                    }

                    return DO_NOTHING;

                }
```

```
            } else if (rightLaneW >= leftLaneW && rightLaneW >= curLaneW)
{

                return TURN_RIGHT;

            } else {

                return TURN_LEFT;

            }

        }
```

4.1.2 public Bot()

```
procedure Bot()

    {Menambahkan angka random serta memberikan arah gerak ke kiri dan
kanan}

    public Bot() {

        this.random = new SecureRandom();

        directionList.add(TURN_LEFT);

        directionList.add(TURN_RIGHT);

    }
```

4.1.3 private boolean hasPowerUps(PowerUps powerupToCheck, PowerUps[] available)

```
    function hasPowerUp(PowerUps powerUpToCheck, PowerUps[]
available)→Boolean

    {Mengecek jika powerup dimiliki}

    ALGORITMA

    for (PowerUps powerUp: available) {
```

```
            if (powerUp.equals(powerUpToCheck)) {
                return true;
            }
        }
        return false;
    }
```

4.1.4 private List<object> getBlocksInFront()

```
    function getBlocksInFront(int lane, int block, int viewDistance,
GameState gameState, Boolean includeFirstBlock) → List<Object>
    {Mendapatkan isi block di depan sejauh speed}

    Kamus Lokal
        map : List<Lane[]>
        blocks : List<Object>
        startBlock, wheretoStart, wheretoEnd, i : integer
        laneList : Lane[]
    Algoritma
        List<Lane[]> map = gameState.lanes;
        List<Object> blocks = new ArrayList<>();
        int startBlock = map.get(0)[0].position.block;
        Lane[] laneList = map.get(lane - 1);
        int wheretoStart = max(block - startBlock + (includeFirstBlock ? 0
: 1), 0);
```

```
        int wheretoEnd = block - startBlock + viewDistance;

        for (int i = wheretoStart; i <= wheretoEnd; i++) {

            if (laneList[i] == null || laneList[i].terrain ==
Terrain.FINISH) {

                break;

            }

            if (laneList[i].cyberTruck) blocks.add(Terrain.TRUCK);

            else if (laneList[i].occupiedByPlayerId ==
gameState.opponent.id) blocks.add(Terrain.OPPONENT);

            else blocks.add(laneList[i].terrain);



        }

        return blocks;
```

4.1.4 private int laneWeight()

```
    function laneWeight(List<Object> laneTerrain) → integer

    {Menghitung beban lane lalu mengembalikan bebannya}

    Kamus Lokal

        opIDX, positionPts, raw, mudfr, oilfr, weight : integer

        scale : double

    Algoritma

                int opIDX = laneTerrain.indexOf(Terrain.OPPONENT);

        int positionPts = 0;

        if (opIDX != -1) {
```

```
            int raw = laneTerrain.size() - opIDX;
            double scale = (raw / laneTerrain.size() * 7) - 3;
            positionPts = (int)(sigmoid(scale) * -15) - 1;
        }
        int mudfr = Collections.frequency(laneTerrain, Terrain.MUD);
        int oilfr = Collections.frequency(laneTerrain, Terrain.OIL_SPILL);
        int weight = Collections.frequency(laneTerrain, Terrain.BOOST) * 6
+
                Collections.frequency(laneTerrain, Terrain.LIZARD) * 5
+
                Collections.frequency(laneTerrain, Terrain.TWEET) * 5
+
                Collections.frequency(laneTerrain, Terrain.OIL_POWER)
* 2 +
                Collections.frequency(laneTerrain, Terrain.EMP) * 5 +
                oilfr*oilfr * -3 +
                mudfr*mudfr * -3 +
                Collections.frequency(laneTerrain, Terrain.WALL) * -13
+
                Collections.frequency(laneTerrain, Terrain.TRUCK) *
-13 +
                positionPts;
        // WEIGHT STILL ARBITRARY, MIGHT UPDATED LATER
        return weight;
```

4.1.5 private int prevSpdState()

```
function prevSpdState(int curSpd) → integer

{Mengembalikan state kecepatan sebelum kecepatan sekarang}

Algoritma

switch (curSpd) {

        case 0:

            return 50; // ERROR CODE

        case 3:

            return 0;

        case 5:

            return 3;

        case 6:

            return 3;

        case 8:

            return 6;

        case 9:

            return 8;

        case 15:

            return 9;

        default:

            return -5000;
```

4.1.6 private int nextSpdState()

```
function nextSpdState(int curSpd) → integer

{Mengembalikan state kecepatan setelah kecepatan sekarang}

Algoritma

Depend on(curSpd)

        switch (curSpd) {

        case 0:

            return 3;

        case 3:

            return 6;

        case 5:

            return 6;

        case 6:

            return 8;

        case 8:

            return 9;

        case 9:

            return 50; // ERROR CODE

        case 15:

            return 50; // ERROR CODE

        default:

            return -5000;

    }
```

4.1.7 private int maxSpd()

```
function maxSpd(int dmg) → integer

{Mengembalikan kecepatan maksimal dari damage yang diterima}

Algoritma

switch(dmg) {

        case 0:

            return 15;

        case 1:

            return 9;

        case 2:

            return 8;

        case 3:

            return 6;

        case 4:

            return 3;

        case 5:

            return 0;

        default:

            return -999;

    }
```

4.1.8 private int cf()

```
    function cf(List<Object> list, Object any) → int

    {Mengembalikan jumlah objek dalam list}

    Algoritma

        return Collections.frequency(list, any);
```

4.1.9 private double sigmoid()

```
    function sigmoid(double x) → double

    {Mengembalikan fungsi sigmoid}

    Algoritma

        return 1 / (1 + Math.exp(-1*x));
```

4.1.10 private boolean hasObstacles()

```
    function hasObstacles(List<Object> lane) → boolean

    {Mengembalikan true jika terdapat obstacle di lane}

    Algoritma

        return lane.contains(Terrain.OPPONENT) ||
lane.contains(Terrain.MUD) || lane.contains(Terrain.OIL_SPILL) ||
lane.contains(Terrain.WALL) || lane.contains(Terrain.TRUCK);
```

4.2 Penjelasan Struktur Data

Program bot overdrive memiliki beberapa kelas yang akan dijelaskan lebih lanjut di bawah

4.2.1 Command

Kelas ini berfungsi untuk mengirimkan perintah yang diberikan bot kepada Game Runner. Kelas ini sering dipakai untuk menyimpan perintah-perintah yang dapat dilakukan seperti perintah ACCELERATE, TURN_RIGHT, TURN_LEFT, dst.

4.2.2 Car

Kelas ini berisikan data umum suatu mobil seperti id bertipe integer, posisi mobil bertipe position, kecepatan mobil bertipe integer, damage mobil bertipe integer, *state* mobil bertipe state, power up mobil bertipe array of powerups, kondisi boost mobil bertipe boolean, dan jumlah boost mobile bertipe integer.

4.2.3 Gamestate

Kelas ini berisikan detail permainan seperti jumlah ronde yang sudah berlalu bertipe integer, jumlah ronde maksimal bertipe integer, mobil pemain bertipe car, mobil lawan bertipe car, dan jalur-jalur dalam permainan bertipe List of Lanes.

4.2.4 Lane

Kelas ini berisikan detail jalur-jalur dalam permainan seperti posisi bertipe position, tipe block bertipe terrain, dan id player yang mengambil block tersebut bertipe integer.

4.2.5 Position

Kelas ini berisikan dengan detail suatu posisi yang terdapat dalam permainan seperti jalurnya (posisi dalam axis y) yang bertipe integer dan posisi dalam axis x yang bertipe integer.

Selain kelas-kelas, program bot overdrive juga memiliki enum yang menjelaskan lebih lanjut beberapa hal seperti

4.2.6 Direction

Enum ini berisikan penjelasan tentang arah yang dilakukan dalam bentuk jalur bertipe integer, block bertipe integer, serta label arah bertipe string.

4.2.7 Power Ups

Enum ini berisikan penjelasan tentang nama-nama power up yang bisa dikirimkan berupa perintah bot seperti BOOST, OIL, dst.

4.2.8 State

Enum ini berisikan penjelasan tentang state mobil setelah suatu ronde seperti ACCELERATING, TURN_RIGHT, dst.

4.2.9 Terrain

Enum ini berisian penjelasan tentang tipe block yang berada dalam jalur-jalur permainan seperti tipe block EMPTY (kosong), MUD, OIL_SPILL, BOOST, dst

Dan untuk menggunakan struktur-struktur data di atas, akan digunakan kelas Bot

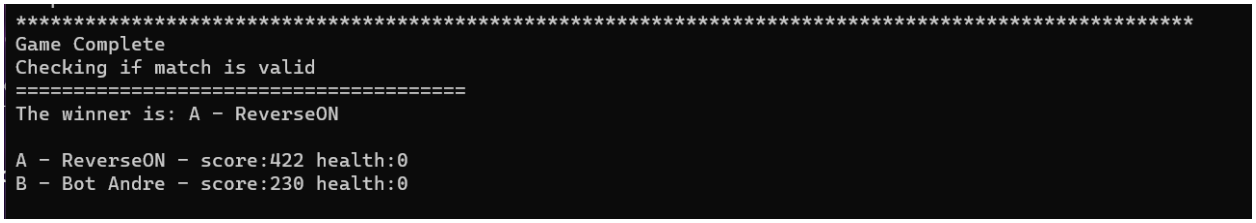
4.2.10 Bot

Kelas ini berisikan logika-logika bot yang akan bermain, seperti kapan akan mengirimkan perintah ACCELERATE, kapan perlu belok kanan atau mengirimkan perintah TURN_RIGHT, dst

4.3 Analisis Desain Solusi Algoritma Greedy

Secara umum, algoritma greedy yang digunakan dalam tugas ini dapat memenangkan sebagian besar permainan. Penggunaan sistem *lane weight* untuk mengukur bobot jalur yang ditempuh dapat memilih jalur optimal untuk dilalui hampir tiap ronde. Pengurutan prioritas perintah juga telah dibuat sedemikian rupa sehingga perintah yang dilakukan terlebih dahulu merupakan perintah yang akan membuat mobil bot berada di kondisi paling optimal.

Berikut merupakan beberapa sampel permainan yang dilakukan bot kami:



Dari beberapa sampel di atas, bisa disimpulkan bahwa bot kami dapat memenangkan pertandingan melawan beberapa bot lain yang kami buat dan juga memaksimalkan skor serta kecepatan mobil apabila pertandingan berakhir dengan kedua mobil bot memiliki jarak tempuh yang sama.

Bab 5

Kesimpulan

5.1 Kesimpulan

Secara umum, bot yang kami program dapat memenangkan sebagian besar pertandingan yang dijalankan. Dalam pengujian kami telah dibuktikan juga bahwa bot kami lebih baik daripada algoritma greedy alternatif lain. Kami simpulkan bahwa program kami dapat memilih solusi optimal pada hampir semua ronde.

5.2 Saran

Aplikasi algoritma greedy dalam permainan Overdrive dapat dibilang cukup efektif. Saran kami untuk kedepannya adalah untuk melanjutkan mencari-cari susunan prioritas perintah yang lebih efektif, serta mencari kembali rasio jarak:skor yang bagus karena sebagian besar permainan yang kami jalankan, skor tidak berfungsi sebagai *tie-breaker.*

Daftar Pustaka

- Rinaldi Munir, Diakses 15 Februari 2022 https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
- Black, Paul E. (2 February 2005). "greedy algorithm". *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology (NIST).
- https://github.com/EntelectChallenge/2020-Overdrive

Link Repository

https://github.com/reverseon/Tubes-1-IF2110-2021-Overdrive