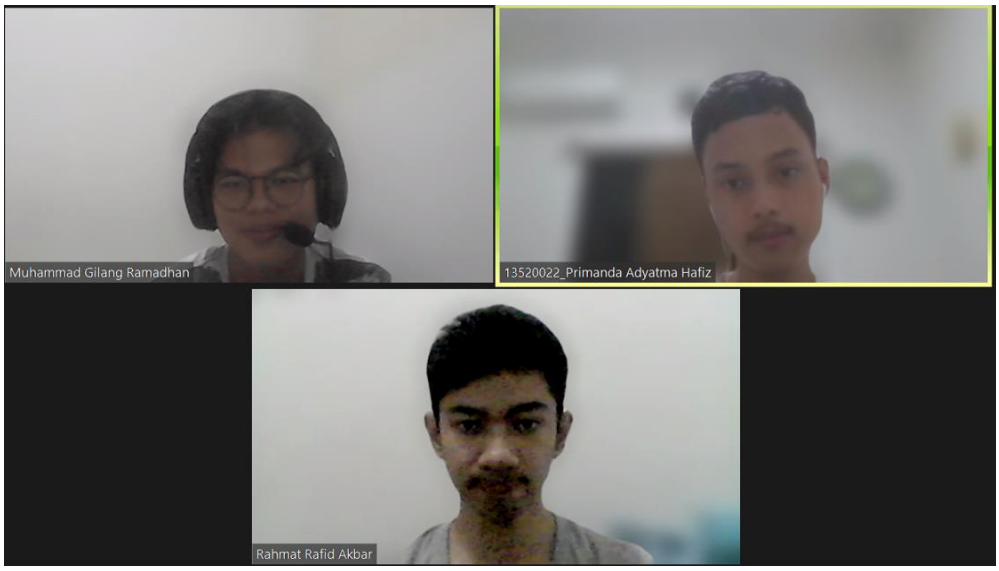
Laporan Penerapan Algoritma *Greedy* pada Permainan Overdrive

TUGAS BESAR STRATEGI ALGORITMA

Oleh:

Primanda Adyatma Hafiz - 13520022
Rahmat Rafid Akbar - 13520090
Muhammad Gilang Ramadhan – 13520137



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 **algoritma 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 Dasar Teori

Dalam menyelesaikan suatu persoalan tentunya diinginkan sebuah solusi terbaik yang dapat digunakan dengan semaksimal mungkin. Dalam proses pencarian solusi atas persoalan dilakukan beberapa pendekatan dengan berbagai strategi penyelesaian. Salah satu strategi penyelesaian yang sering digunakan adalah dengan memanfaatkan algoritma *greedy*.

Algoritma greedy adalah algoritma sederhana yang intuitif untuk mengoptimalisasi penyelesaian sebuah persoalan. Algoritma ini akan berusaha menemukan solusi yang merupakan solusi optimum lokal pada suatu tahap/ bagian persoalan, dengan harapan salah satu dari kumpulan solusi lokal ini akan memuat sebuah solusi optimum global yang merupakan solusi paling optimal untuk keseluruhan persoalan. Sehingga, algoritma ini tidak dapat menjamin ditemukannya sebuah solusi paling optimum yang dapat menyelesaikan persoalan secara global. Namun, algoritma ini sangat berguna untuk melakukan aproksimasi terhadap suatau keadaan tertentu yang tidak membutuhkan hasil pasti (eksak).

Sesuai namanya, algoritma greedy, (Eng, rakus/tamak/loba), merupakan algoritma yang melakukan pengambilan solusi yang paling optimal pada suatu tahap/bagian persoalan dan mengklaim solusi itu sebagai solusi yang optimal. Algoritma ini dapat diungkapkan dalam suatu slogan, “Take what you can take now”, yang berarti ambil apa yang bisa diambil saat ini. Dengan kata lain, untuk setiap tahap/bagian persoalan, algoritma greedy akan mengambil solusi paling optimal pada saat suatu tahap dieksekusi, dan diharapkan dengan mengumpulkan solusi-solusi paling optimal saat itu, maka akan diperoleh sebuah solusi optimal dari kumpulan solusi tersebut yang dapat berlaku secara global untuk keseluruhan persoalan.

Implementasi algoritma greedy yang banyak dikenal adalah algoritma djikstra (algoritma yang digunakan untuk menemukan jarak terpendek dari satu node ke node lain pada suatu graf) dan algoritma Huffman Coding (sebuah algoritma untuk melakukan encoding dan decoding terhadap sebuah pesan/data untuk menghemat penggunaan memori sekaligus menjamin keamanan

data tersebut). Algoritma greedy juga bisa dimanfaatkan untuk menyelesaikan persoalan yang lebih kompleks seperti Knapsack problem dan juga coin exchange problem, namun algoritma greedy tidak dapat secara sempurna memperoleh solusi paling optimal untuk setiap kemungkinan yang ada, maka dari itu untuk memperoleh solusi yang lebih optimal untuk persoalan-persoalan yang lebih kompleks seringkali diperlukan algoritma yang lebih kompleks juga.

Algoritma greedy memiliki beberapa elemen yang harus dipenuhi, yaitu sebagai berikut:

1. Himpunan Kandidat

Himpunan kandidat berisi kumpulan solusi yang mungkin akan dipilih pada setiap langkah, atau semua ruang sampel yang dapat kita pilih sebagai suatu solusi.

2. Himpunan Solusi

Himpunan solusi adalah himpunan bagian dari himpunan kandidat yang merupakan hasil final yang sudah dipilih berdasarkan pada setiap tahap/ bagian persoalan.

3. Fungsi Seleksi

Fungsi seleksi digunakan untuk menyeleksi anggota-anggota dari himpunan kandidat yang nantinya akan diproses kembali oleh fungsi kelayakan dan akhirnya dapat dijadikan himpunan solusi. Fungsi seleksi merupakan ciri khas dari algoritma greedy karena bersifat intuitif sehingga dapat berbeda untuk setiap implementasinya.

4. Fungsi Kelayakan

Fungsi kelayakan digunakan untuk memeriksa apakah sebuah solusi yang dipilih merupakan solusi yang layak untuk dimasukkan ke dalam himpunan solusi atau tidak. Biasanya fungsi ini ditentukan dari kondisi-kondisi yang harus dipenuhi dan membatasi persoalan pada program.

5. Fungsi Objektif

Fungsi objektif adalah fungsi yang merepresentasikan tujuan dari persoalan yang mencari solusi paling optimal, biasanya diantara memaksimalkan atau meminimalkan nilai suatu random variabel.

2.2 Cara Kerja Program

Game engine Overdrive akan menjalankan file run.bat yang akan mengeksekusi file game-runner-jar-with-dependencies.jar yang akan membaca konfigurasi file game-runner-config.json untuk menjalankan game. File game-runner-config.json akan membaca keberadaan 2 object Bot

yang merupakan pemain yang akan bertanding serta skema dasar permainan pada file game-config.json. File konfiugrasi ini juga akan mengeksekusi file game-engine.jar yang merupakan mesin yang digunakan untuk menjalankan permainan.

Pada saat membaca keberadaan 2 object Bot, file setiap Bot akan dieksekusi secara terpisah pada file Main.java yang berkesesuaian. Sementara itu, implementasi algoritma greedy dilakukan di file Bot.java pada function member run() yang nantinya akan dipanggil pada file Main.java. Implementasi algoritma greedy ini dilakukan secara terurut dan sekuensial dengan menentukan return value dari fungsi untuk setiap command dan membandingkannya berdasarkan fungsi seleksi. Command yang memiliki prioritas dan return value tertinggi akan direturn pada saat pemanggilan function run() di Main.java dan program pun akan menjalankan command yang sesuai.

BAB 3
APLIKASI STRATEGI GREEDY

3.1 *Mapping* Persoalan

Pada permainan Overdrive, akan terdapat 2 buah bot yang dipertandingkan. Bot-bot ini mengeksekusi program yang telah dibuat sebelumnya. Dalam proses pembuatan program dilakukan abstraksi persoalan terlebih dahulu dengan menentukan elemen-elemen pokok pada algortima greedy berupa himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif.

Salah satu faktor yang kami gunakan dalam abstraksi program kami adalah dengan memanfaatkan pembagian command-command yang ada. Pada permainan kali ini, terdapat 11 command yang tersedia dan dapat digunakan disetiap round pertandingan. Penentuan elemen-elemen pokok algoritma ini sebagai berikut:

- Himpunan kandidiat dapat diperoleh dari kartesian produk terhadap pengambilan command pada setiap round. Apabila terdapat n buah round maka akan terdapat 11^n elemen pada himpunan kandidat yang dapat dipilih.
- Himpunan solusi merupakan keseluruhan pemilihan command yang telah dilakukan pada sebuah pertandingan (match). Anggota himpunan solusi ini merupakan salah satu elemen dari himpunan kandidat yang terpilih berdasarkan fungsi seleksi, fungsi kelayakan, dan fungsi objektif.
- Fungsi seleksi yang digunakan pada program kami adalah pembuatan urutan prioritas berdasarkan fungsi objektif yang dipilih. Pembuatan fungsi seleksi (urutan prioritas) ini didasari oleh intuisi dan kesepakatan anggota kelompok.
- Fungsi kelayakan dibentuk berdasarkan kondisi yang telah ditetapkan sebelumnya dengan memenuhi beberapa constrains (batasan) : kerusakan yang diterima, kecepatan mobil, keberadaan obstacle dalam lintasan dan ketersediaan power up. Penentuan fungsi kelayakan dilakukan sesuai dengan command yang ditangani oleh anggota kelompok.
- Fungsi objektif, pada spesifikasi program telah dijelaskan secara sederhana bahwa tujuan permainan ini adalah mencapai garis finish secepat mungkin dan meraih skor setinggi

mungkin. Dalam merealisasikan tujuan tersebut, kami mengambil langkah prioritas untuk cenderung meningkatkan kecepatan mobil semaksimal mungkin dan kerusakan yang diterima mobil seminimal mungkin. Sehingga, tujuan utama dapat tercapai dengan round yang seminimal mungkin dan dapat memenangkan pertandingan.

3.2 Alternatif Solusi

Salah satu alternatif solusi yang dapat digunakan pada program Bot permainan Overdrive ini adalah dengan selalu berusaha mengambil *powerup* dan menghindari *obstacle* sehingga diperoleh skor yang maksimum. Dengan menggunakan alternatif solusi ini, kendaraan kita memiliki kemungkinan besar untuk mencapai garis akhir terlebih dahulu dibanding musuh karena dengan mendapatkan *powerup* dan menghindari *obstacle* kita dapat memiliki lebih banyak opsi *command* dan kecepatan maksimal mobil bisa menjadi lebih tinggi. Selain itu, skor yang lebih besar juga akan memberikan kita keuntungan pada akhir permainan bila musuh sampai di garis akhir pada waktu yang bersamaan dengan kita.

3.3 Analisis Efektivitas dan Efisiensi Alternatif Solusi

Alternatif solusi di atas akan menguntungkan kita bila powerup yang diperoleh dapat memaksimalkan kemungkinan kita untuk mencapai garis akhir terlebih dahulu dibandingkan musuh. Sehingga jika strateginya adalah mengumpulkan powerups sebanyak-banyaknya terdapat beberapa kasus di mana powerups tidak terlalu menguntungkan karena pada dasarnya terdapat beberapa powerups yang lebih superior dibandingkan powerup lainnya antara lain adalah Booster dan Emp. Selain itu jika kita hanya berusaha menghindari obstacle saja akan terdapat beberapa kasus di mana kita gagal mendapatkan powerup yang menguntungkan padahal obstacle yang dihindari misalnya hanyalah sebuah mud yang hanya berdampak kecil baik ke kerusakan maupun ke kecepatan dari kendaraan. Oleh karena itu, semestinya solusi yang baik adalah solusi yang mempertimbangkan baik powerup maupun obstacle yang masing-masing jenisnya memiliki nilai positif dan negatif yang berbeda sehingga keseluruhan jenisnya tidak disamaratakan.

3.4 Strategi Greedy yang dipilih

Strategi greedy yang kami pakai dalam permainan ini adalah dengan berusaha memaksimalkan kecepatan dan penggunaan *powerup*, serta mendapatkan *powerup* dengan

menggunakan skala prioritas dari 0 sampai 5 pada masing-masing command. Pada skala prioritas tersebut, semakin tinggi nilainya maka semakin recommended command tersebut dipakai pada suatu kondisi. Jadi, sesuai dengan strategi greedy, untuk setiap langkahnya dapat diambil skala prioritas yang paling tinggi (maksimum lokal) untuk kondisi tertentu. Adapun jika terdapat nilai prioritas yang sama pada beberapa command, sesuai dengan strategi kami yaitu dengan memaksimalkan kecepatan dan penggunaan *powerup* (terutama pada *powerup* yang berkaitan dengan kecepatan), urutan yang diprioritaskan dari yang terbesar ke terkecil adalah FIX, BOOST, ACCELERATE, EMP, LIZARD, TURN_RIGHT, TURN_LEFT, OIL, NOTHING, DECELERATE.

Strategi yang kami terapkan tersebut cenderung lebih ke strategi *offensive healing*. Hal tersebut terlihat dari urutan prioritas dari command yang kami susun, dimana FIX, BOOST, ACCELERATE, dan EMP menempati urutan teratas. Sesuai dengan strategi kami, yaitu mengandalkan kekuatan *powerup*, pada awal game memang strategi ini memang tidak terlalu efektif, karena di *early game, powerup* yang terkumpul cenderung lebih sedikit dibandingkan pada saat *late game*. Selain itu, karena strategi kami juga berusaha mencari *powerup* maka score yang terkumpul pun cenderung lebih banyak.

Untuk masalah penyerangan terhadap musuh, kendaraan kami menyerang kendaraan musuh melalui EMP ataupun OIL (utamanya EMP), tapi kami disini tidak memakai command TWEET. Hal tersebut dikarenakan CYBER_TRUCK kita pada suatu kondisi dapat mengahalangi kendaraan kita untuk melaju.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma

Selection_Command

```
function Selection_Command () → Command
{ Fungsi untuk menyeleksi command}
Deklarasi
      int com[] : new int[11]
Algoritma
      // Ambil angka prioritas dari setiap command
      com[0] = UseFix();
      com[1] = UseBoost();
      com[2] = UseAccelerate();
      com[3] = UseEMP();
      com[4] = UseLizard();
      // Turn Right
      com[5] = UseTurn_Right(myCar.boosting, myCar.boostCounter>1);
      // Turn Left
      com[6] = UseTurn_Left(myCar.boosting, myCar.boostCounter>1);
      com[7] = UseOil();
      com[8] = UseDo_Nothing();
      com[9] = UseDecelerate();

      // Cari command dengan prioritas terbesar
```

```
int idxMax = 0;
int maks = com[0];
for(int i = 1; i < 11; i++){
    if(com[i] > maks){
        maks = com[i];
        idxMax = i;
    }
}
// Fungsi seleksi
if(idxMax == 0){
    return FIX;
}
else if(idxMax == 1){
    return BOOST;
}
else if(idxMax == 2){
    return ACCELERATE;
}
else if(idxMax == 3){
    return EMP;
}
else if(idxMax == 4){
    return LIZARD;
}
```

```
      else if(idxMax == 5){
          return TURN_RIGHT;
      }
      else if(idxMax == 6){
          return TURN_LEFT;
      }
      else if(idxMax == 7){
          return OIL;
      }
      else if(idxMax == 8){
          return NOTHING;
      }
      else if(idxMax == 9){
          return DECELERATE;
      }
      else{
          return ACCELERATE;
      }
```

hasPowerUp

```
function hasPowerUp(PowerUps a) → boolean
{ mereturn true jika car punya powerup }
Deklarasi
      -
```

```
Algoritma
        for (PowerUps x:myCar.powerups){
           if(x.equals(a)){
              return true;
           }
        }
        return false;
```

getMaxSpeedByDamage

```
function getMaxSpeedByDamage() → Int
{ mereturn integer sebagai kecepatan maksimum yang bisa dicapai Car }
Deklarasi
      -
Algoritma
        if(myCar.damage==5){
           return 0;
        }else if(myCar.damage==4){
           return 3;
        }else if(myCar.damage==3){
           return 6;
        }else if(myCar.damage==2){
           return 8;
        }else if(myCar.damage==1){
```

```
        return 9;
    }else{
        return 15;
    }
```

getCurSpeed

```
function  getCurSpeed(int numLevelUp,boolean isBooster) → Int

{ mengembalikan integer sebagai kecepatan Car jika levelnya ditambah
sebesar numLevelUp dan berdasarkan apakah booster diaktifkan }

Deklarasi

    -

Algoritma

    if(isBooster){
        return getMaxSpeedByDamage();
    }
    int[] allSpeed={0,3,5,6,8,9,15};
    int idx=0;
    for(;idx<7;idx++){
        if(allSpeed[idx]==myCar.speed){
            break;
        }
    }
    if(idx+numLevelUp>5){
        idx=5;
```

```
        }else{
            idx=max(idx+numLevelUp,0);
        }
        return min(allSpeed[idx],getMaxSpeedByDamage());
```

getMaxPos

```
function getMaxPos() → Int
{ mengembalikan integer sebagai maksimum posisi yang bisa dicapai }
Deklarasi
        Lane[] curLane     :  gameState.lanes.get(myCar.position.lane-1)
        int startBlock     :  gameState.lanes.get(0)[0].position.block
Algoritma
        for (int i=max(myCar.position.block-startBlock+1,0);i<=myCar.position.block-
        startBlock+20;i++){
            if (curLane[i]==null || curLane[i].terrain==Terrain.FINISH){
                return i;
            }
        }
        return 20;
```

UseOil

```
function UseOil() → Int
{ mengembalikan integer sebagai skala prioritas command}
```

```
Deklarasi

        boolean canHit       : false

        Lane[] curLane       : gameState.lanes.get(myCar.position.lane-1)

        int startBlock       : gameState.lanes.get(0)[0].position.block

        int obstacleDMG   : 0

Algoritma

        // Mengecek apakah lawan berada di belakang kita

        if(myCar.position.block > opponent.position.block){

           canHit = true;

        }

        // Mengecek total damage dari obstacle di depan

        for(int i=max(myCar.position.block-
        startBlock+1,0);i<=min(getMaxPos(),myCar.position.block-
        startBlock+myCar.speed);i++){

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

              break;

           }else if(curLane[i].terrain==Terrain.WALL ||
        curLane[i].isOccupiedByCyberTruck){

              obstacleDMG += 2;

           }

           else if(curLane[i].terrain==Terrain.MUD ||
        curLane[i].terrain==Terrain.OIL_SPILL){

              obstacleDMG += 1;

           }

        }

        boolean hasPU = hasPowerUp(PowerUps.OIL);
```

```
// ~Split CASE~
if(!(hasPU && canHit)){
    return 0;
}
else if(hasPU && canHit && obstacleDMG > 2){
    return 1;
}
else if(hasPU && canHit && obstacleDMG > 0){
    return 2;
}
else if(hasPU && canHit && obstacleDMG == 0){
    return 3;
}
else if(hasPU && canHit && myCar.speed==maxSpeed && obstacleDMG > 0){
    return 4;
}
else if(hasPU && canHit && myCar.speed==maxSpeed && obstacleDMG == 0){
    return 5;
}
else{
    return 0;
}
```

UseEMP

```
function UseEMP() → Int
{ mengembalikan integer sebagai skala prioritas command}
Deklarasi
        int curLaneIdx  : myCar.position.lane - 1
        boolean canHit  :  false
Algoritma
        // Mengecek apakah lawan berada dalam ruang tembak dari EMP
        if(myCar.position.block < opponent.position.block){
           canHit = true;
        }
        if(canHit){
           for(int i = curLaneIdx-1; i <= curLaneIdx+1; i++){
              canHit = false;
              if(0<=i && i <=3){
                 if(i+1 == opponent.position.lane){
                    canHit = true;
                    break;
                 }
              }
           }
        }
        boolean hasPU = hasPowerUp(PowerUps.EMP);
        // Menghitung total damage dari obstacle
```

```
Lane[] curLane = gameState.lanes.get(myCar.position.lane-1);

int startBlock=gameState.lanes.get(0)[0].position.block;

int obstacleDMG = 0;

for(int i=max(myCar.position.block-
startBlock+1,0);i<=min(getMaxPos(),myCar.position.block-
startBlock+myCar.speed);i++){

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

      break;

   }else if(curLane[i].terrain==Terrain.WALL ||
curLane[i].isOccupiedByCyberTruck){

      obstacleDMG += 2;

   }

   else if(curLane[i].terrain==Terrain.MUD ||
curLane[i].terrain==Terrain.OIL_SPILL){

      obstacleDMG += 1;

   }

}

// ~Split CASE~

if(!(hasPU && canHit)){

   return 0;

}

else if(hasPU && canHit && obstacleDMG > 2){

   return 1;

}

else if(hasPU && canHit && obstacleDMG > 0){

   return 2;
```

```
        }
        else if(hasPU && canHit && obstacleDMG == 0){
            return 3;
        }
        else if(hasPU && canHit && myCar.speed==maxSpeed && obstacleDMG > 0){
            return 4;
        }
        else if(hasPU && canHit && myCar.speed==maxSpeed && obstacleDMG == 0
        && myCar.position.lane==opponent.position.lane){
            return 5;
        }
        else{
            return 0;
        }
```

UseLizard

```
function UseLizard() → Int
{ mengembalikan integer sebagai skala prioritas command}
Deklarasi
        boolean hasPU      : hasPowerUp(PowerUps.LIZARD)
        Lane[] curLane     : gameState.lanes.get(myCar.position.lane-1)
        int startBlock     : gameState.lanes.get(0)[0].position.block
        int obstacleDMG    : Inisialisasi Damage Obstacle
        int numPowerup     : Inisialisasi banyaknya Powerup yang telah dikumpulkan
```

```
Algoritma

        obstacleDMG = 0;

        numPowerup=0;

        boolean safeLanding=myCar.position.block-
        startBlock+myCar.speed>getMaxPos()?true:(curLane[myCar.position.block-
        startBlock+myCar.speed].terrain!=Terrain.MUD &&
        curLane[myCar.position.block-
        startBlock+myCar.speed].terrain!=Terrain.OIL_SPILL &&
        curLane[myCar.position.block-startBlock+myCar.speed].terrain!=Terrain.WALL
        && curLane[myCar.position.block-
        startBlock+myCar.speed].isOccupiedByCyberTruck);

        for(int i=max(myCar.position.block-
        startBlock+1,0);i<min(getMaxPos(),myCar.position.block-
        startBlock+myCar.speed);i++){

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

              break;

           }else if(curLane[i].terrain==Terrain.WALL ||
        curLane[i].isOccupiedByCyberTruck){

              obstacleDMG += 2;

           }

           else if(curLane[i].terrain==Terrain.MUD ||
        curLane[i].terrain==Terrain.OIL_SPILL){

              obstacleDMG += 1;

           }else if(curLane[i].terrain==Terrain.BOOST ||
        curLane[i].terrain==Terrain.EMP){

              numPowerup++;

           }

        }

        int turn=max(UseTurn_Left(gameState.player.boosting,
        gameState.player.boostCounter>1),UseTurn_Right(gameState.player.boosting,
        gameState.player.boostCounter>1));
```

```
        if(!hasPU && turn>=4){
            return 0;
        }else if(!safeLanding && obstacleDMG<=2){
            return 1;
        }else if(!safeLanding && obstacleDMG>2){
            return 3;
        }else if(obstacleDMG>=2 && numPowerup>0){
            return 4;
        }else if((obstacleDMG>=2 && numPowerup==0) || obstacleDMG>3){
            return 5;
        }else{
            return 0;
        }
```

isNabrakObstacleInfront_atCurrentLane

```
function isNabrakObstacleInfront_atCurrentLane (int lane, int curblock,
Boolean isBooster, Boolean isNextBooster) → boolean

{ mengembalikan true jika menabrak dan false jika tidak menabrak }

Deklarasi

      kerusakan      : Damage dari myCar

      int NextSpeed  : NextSpeed dari Car pada NextRound

      Lane[] curLane : gameState.lanes.get(lane-1)

      int block      : gameState.lanes.get(0)[0].position.block

Algoritma
```

```
Lane[] curLane = gameState.lanes.get(lane-1);

int block = gameState.lanes.get(0)[0].position.block;

for (int i = max(curblock - block, 0); i <=
min(getMaxPos(),curblock - block + getCurSpeed(0, isBooster)); i++)
{

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

        break;

    }

    else if (curLane[i].terrain==Terrain.MUD ||
curLane[i].terrain==Terrain.OIL_SPILL){

        kerusakan += 1;

    }

    else if (curLane[i].terrain==Terrain.WALL ||
curLane[i].terrain==Terrain.CYBER_TRUCK){

        kerusakan += 2;

    }

}

// Kecepatan mobil di next round ketika setelah satu round

if (kerusakan == 5){

    NextSpeed = min(getCurSpeed(0, isNextBooster), 0);

}

else if (kerusakan == 4){

    NextSpeed = min(getCurSpeed(0, isNextBooster), 3);

}

else if (kerusakan == 3){

    NextSpeed = min(getCurSpeed(0, isNextBooster), 6);
```

```
    }

    else if (kerusakan == 2){

        NextSpeed = min(getCurSpeed(0, isNextBooster), 8);

    }

    else if (kerusakan == 1){

        NextSpeed = min(getCurSpeed(0, isNextBooster), 9);

    }

    else{

        NextSpeed = min(getCurSpeed(0, isNextBooster), 15);

    }

    // Buat cek apakah ada obstacle yang bakal ketabrak di depan
    bloknya

    for (int i = curblock - block + getCurSpeed(0, isBooster) + 1; i
    <=min(getMaxPos(),curblock - block + getCurSpeed(0, isBooster) +
    NextSpeed); i++){

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

            break;

        }

        else if (curLane[i].terrain==Terrain.MUD ||
    curLane[i].terrain==Terrain.OIL_SPILL ||
    curLane[i].terrain==Terrain.WALL ||
    curLane[i].terrain==Terrain.CYBER_TRUCK){

            return true;

        }

    }

    return false;
```

isNabrakObstacle_turning

```
function isNabrakObstacleInfront_atCurrentLane (int lane, int curblock,
Boolean isBooster, Boolean isNextBooster) → boolean

{ mengembalikan true jika menabrak dan false jika tidak menabrak }

Deklarasi

        Lane[] curLane  : gameState.lanes.get(lane-1)

        int block           : gameState.lanes.get(0)[0].position.block

Algoritma

        for (int i = max(curblock - block, 0); i <= min(getMaxPos(),curblock - block +
        getCurSpeed(0, isBooster)); i++) {

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

              break;

           }

           else if (curLane[i].terrain==Terrain.MUD ||
        curLane[i].terrain==Terrain.OIL_SPILL || curLane[i].terrain==Terrain.WALL ||
        curLane[i].terrain==Terrain.CYBER_TRUCK){

              return true;

           }

        }

        return false;
```

isNabrakPowerUpInfront_atCurrentLane

```
function isNabrakObstacleInfront_atCurrentLane (int lane, int curblock,
Boolean isBooster, Boolean isNextBooster) → boolean

{ mengembalikan true jika menabrak dan false jika tidak menabrak }

Deklarasi
```

```
      kerusakan      : Damage dari myCar

      int NextSpeed  : NextSpeed dari Car pada NextRound

      Lane[] curLane : gameState.lanes.get(lane-1)

      int block      : gameState.lanes.get(0)[0].position.block

Algoritma

      for (int i = max(curblock - block, 0); i <= min(getMaxPos(),curblock - block +
      getCurSpeed(0, isBooster)); i++) {

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

            break;

         }

         else if (curLane[i].terrain==Terrain.MUD ||
      curLane[i].terrain==Terrain.OIL_SPILL){

            kerusakan += 1;

         }

         else if (curLane[i].terrain==Terrain.WALL ||
      curLane[i].terrain==Terrain.CYBER_TRUCK){

            kerusakan += 2;

         }

      }

      // Kecepatan mobil di next round ketika setelah satu round

      if (kerusakan == 5){

         NextSpeed = min(getCurSpeed(0, isNextBooster), 0);

      }

      else if (kerusakan == 4){

         NextSpeed = min(getCurSpeed(0, isNextBooster), 3);

      }
```

```
else if (kerusakan == 3){
   NextSpeed = min(getCurSpeed(0, isNextBooster), 6);
}
else if (kerusakan == 2){
   NextSpeed = min(getCurSpeed(0, isNextBooster), 8);
}
else if (kerusakan == 1){
   NextSpeed = min(getCurSpeed(0, isNextBooster), 9);
}
else{
   NextSpeed = min(getCurSpeed(0, isNextBooster), 15);
}
// Buat cek apakah ada obstacle yang bakal ketabrak di depan bloknya
for (int i = curblock - block + getCurSpeed(0, isBooster) + 1; i
<=min(getMaxPos(),curblock - block + getCurSpeed(0, isBooster) + NextSpeed);
i++){
   if (curLane[i] == null || curLane[i].terrain == Terrain.FINISH) {
      break;
   }
   else if (curLane[i].terrain==Terrain.OIL_POWER ||
curLane[i].terrain==Terrain.BOOST || curLane[i].terrain==Terrain.EMP ||
curLane[i].terrain==Terrain.TWEET || curLane[i].terrain==Terrain.LIZARD){
      return true;
   }
}
return false;
```

isNabrakpowerUp_turning

```
function isNabrakpowerUp_turning (int lane, int curblock, Boolean
isBooster, Boolean isNextBooster) → boolean

{ mengembalikan true jika menabrak dan false jika tidak menabrak }

Deklarasi

        Lane[] curLane  : gameState.lanes.get(lane-1)

        int block           : gameState.lanes.get(0)[0].position.block

Algoritma

        Lane[] curLane=gameState.lanes.get(lane-1);

        int block = gameState.lanes.get(0)[0].position.block;

        for (int i = max(curblock - block, 0); i <= min(getMaxPos(),curblock - block +
        getCurSpeed(0, isBooster)); i++) {

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

              break;

           }

           else if (curLane[i].terrain==Terrain.MUD ||
        curLane[i].terrain==Terrain.OIL_SPILL || curLane[i].terrain==Terrain.WALL ||
        curLane[i].terrain==Terrain.CYBER_TRUCK){

              return true;

           }

        }

        return false;
```

UseTurn_Left

```
function  UseTurn_Left(boolean isBooster, boolean isNextBooster) → int
```

```
{ mengembalikan integer sebagai skala prioritas command}

Deklarasi

      -

Algoritma

      if (myCar.position.lane - 1 > 0){

         if (isNabrakpowerUp_turning(myCar.position.lane-1, myCar.position.block,
      isBooster) || isNabrakpowerUp_turning(myCar.position.lane, myCar.position.block,
      isBooster) || isNabrakPowerUpInfront_atCurrentLane(myCar.position.lane-1,
      myCar.position.block, isBooster, isNextBooster)){

            if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
      isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
      myCar.position.block, isBooster) &&
      !isNabrakObstacleInfront_atCurrentLane(myCar.position.lane-1,
      myCar.position.block, isBooster, isNextBooster)){

               return 5;

            }

            else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
      isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
      myCar.position.block, isBooster) &&
      isNabrakObstacleInfront_atCurrentLane(myCar.position.lane-1,
      myCar.position.block, isBooster, isNextBooster)){

               return 4;

            }

            else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
      isBooster) && isNabrakObstacle_turning(myCar.position.lane-1,
      myCar.position.block, isBooster)){

               return 1;

            }

            else if (!isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
      isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
      myCar.position.block, isBooster)){
```

```
            return 2;

        }

        else{

            return 0;

        }

    }

    else{

        if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
myCar.position.block, isBooster) &&
!isNabrakObstacleInfront_atCurrentLane(myCar.position.lane-1,
myCar.position.block, isBooster, isNextBooster)){

            return 4;

        }

        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
myCar.position.block, isBooster) &&
isNabrakObstacleInfront_atCurrentLane(myCar.position.lane-1,
myCar.position.block, isBooster, isNextBooster)){

            return 3;

        }

        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && isNabrakObstacle_turning(myCar.position.lane-1,
myCar.position.block, isBooster)){

            return 0;

        }

        else if (!isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane-1,
myCar.position.block, isBooster)){

            return 1;
```

```
            }
            else{
                return 0;
            }
        }
    }
    else{
        return 0;
    }
```

UseTurn_Right

```
function UseTurn_Right(boolean isBooster, boolean isNextBooster) → Int
{ mengembalikan integer sebagai skala prioritas command}
Deklarasi
    -
Algoritma
    if (myCar.position.lane - 1 < 3){
        if (isNabrakpowerUp_turning(myCar.position.lane+1, myCar.position.block,
    isBooster) || isNabrakpowerUp_turning(myCar.position.lane, myCar.position.block,
    isBooster) || isNabrakPowerUpInfront_atCurrentLane(myCar.position.lane+1,
    myCar.position.block, isBooster, isNextBooster)){
            if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
    isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
    myCar.position.block, isBooster) &&
    !isNabrakObstacleInfront_atCurrentLane(myCar.position.lane+1,
    myCar.position.block, isBooster, isNextBooster)){
                return 5;
```

```
        }

        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster) &&
isNabrakObstacleInfront_atCurrentLane(myCar.position.lane+1,
myCar.position.block, isBooster, isNextBooster)){

            return 4;

        }

        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster)){

            return 1;

        }

        else if (!isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster)){

            return 2;

        }

        else{

            return 0;

        }

    }

    else{

        if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster) &&
!isNabrakObstacleInfront_atCurrentLane(myCar.position.lane+1,
myCar.position.block, isBooster, isNextBooster)){

            return 4;

        }
```

```
        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster) &&
isNabrakObstacleInfront_atCurrentLane(myCar.position.lane+1,
myCar.position.block, isBooster, isNextBooster)){

            return 3;

        }

        else if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster)){

            return 0;

        }

        else if (!isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
isBooster) && !isNabrakObstacle_turning(myCar.position.lane+1,
myCar.position.block, isBooster)){

            return 1;

        }

        else{

            return 0;

        }

    }

}

else{

    return 0;

}
```

UseFix

```
function UseFix() → int

{ mengembalikan integer sebagai skala prioritas command}

Deklarasi

      -

Algoritma

      if (myCar.damage >= 3 || (myCar.damage >= 2 &&
      hasPowerUp(PowerUps.BOOST)) && getMaxPos()>myCar.speed){

         return 5;

      }

      else if (myCar.damage >= 2){

         return 4;

      }

      else if (myCar.damage >= 1){

         return 3;

      }

      else {

         return 0;

      }
```

UseDo_Nothing

```
function UseDo_Nothing() → int

{ mengembalikan integer sebagai skala prioritas command}

Deklarasi
```

```
      -

Algoritma

      if (isNabrakObstacle_turning(myCar.position.lane, myCar.position.block,
      myCar.boosting) || hasPowerUp(PowerUps.BOOST) ||
      hasPowerUp(PowerUps.OIL) || hasPowerUp(PowerUps.TWEET) ||
      hasPowerUp(PowerUps.LIZARD) || hasPowerUp(PowerUps.EMP)){

         return 0;

      }

      else if (!hasPowerUp(PowerUps.BOOST) && !hasPowerUp(PowerUps.OIL) &&
      !hasPowerUp(PowerUps.TWEET) && !hasPowerUp(PowerUps.LIZARD) &&
      !hasPowerUp(PowerUps.EMP)){

         return 1;

      }

      else {

         return 2;

      }
```

UseBoost

```
function UseBoost() → Int

{ mengembalikan integer sebagai skala prioritas command}

Deklarasi

      curLane : Data sebuah lane

      cntWallCyber : Jumlah Wall dan Cyber Truck yang ada di depan

      cntMudOil : Jumlah Mud dan Oil Truck yang ada di depan

      cntBoost : Jumlah powerup boost yang ada di depan

      startBlock    : Indeks blok awal
```

```
Algoritma

      Lane[] curLane=gameState.lanes.get(myCar.position.lane-1);

      int cntWallCyber=0;

      int cntMudOil=0;

      int cntBoost=0;

      int startBlock=gameState.lanes.get(0)[0].position.block;

      // Menghitung jumlah obstacle dan powerup yang ada

      for(int i=max(myCar.position.block-
      startBlock+1,0);i<=min(getMaxPos(),myCar.position.block-
      startBlock+getCurSpeed(0, true));i++){

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

                      break;

                  }else if(curLane[i].terrain==Terrain.MUD ||
      curLane[i].terrain==Terrain.OIL_SPILL){

                      cntMudOil++;

                  }else if(curLane[i].terrain==Terrain.WALL ||
      curLane[i].isOccupiedByCyberTruck){

                      cntWallCyber++;

                  }else if(curLane[i].terrain==Terrain.BOOST){

                      cntBoost++;

                  }

              }

              if(!hasPowerUp(PowerUps.BOOST) || cntWallCyber>0){

                  return 0;

              }else if(cntMudOil>0 && cntBoost==0){

                  return 1;

              }else if(myCar.speed==getMaxSpeedByDamage() || (cntBoost==1
```

```
      && cntMudOil>0)){
                  return 2;
              }else if(cntBoost>1 && cntMudOil>0){
                  return 3;
              }else if(myCar.damage==0){
                  return 5;
              }else{
                  return 4;
              }
```

UseAccelerate

```
function UseAccelerate() → Int
{ mengembalikan integer sebagai skala prioritas command}
Deklarasi
      curLane : Data sebuah lane
      cntWallCyber : Jumlah Wall dan Cyber Truck yang ada di depan
      cntMudOil : Jumlah Mud dan Oil Truck yang ada di depan
      cntBoost : Jumlah powerup boost yang ada di depan
      startBlock    : Indeks blok awal
Algoritma
      Lane[] curLane=gameState.lanes.get(myCar.position.lane-1);
      int cntWallCyber=0;
      int cntMudOil=0;
      int cntBoost=0;
```

```
int startBlock=gameState.lanes.get(0)[0].position.block;

// Menghitung jumlah obstacle dan powerup yang ada

for(int i=max(myCar.position.block-
startBlock+1,0);i<=min(getMaxPos(),myCar.position.block-
startBlock+getCurSpeed(1, false));i++){

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

                break;

            }else if(curLane[i].terrain==Terrain.MUD ||
curLane[i].terrain==Terrain.OIL_SPILL){

                cntMudOil++;

            }else if(curLane[i].terrain==Terrain.WALL ||
curLane[i].isOccupiedByCyberTruck){

                cntWallCyber++;

            }else if(curLane[i].terrain==Terrain.BOOST){

                cntBoost++;

            }

        }

        if(myCar.speed==0){

            return 5;

        }else if(cntWallCyber>0){

            return 0;

        }else if(cntMudOil > 0 && cntBoost == 0){

            return 1;

        }else if(myCar.speed >= getCurSpeed(6,false)){

            return 2;

        }else if(cntBoost == 1 && cntMudOil>0){
```

```
                return 4;
            }else{
                return 5;
            }
```

UseDecelerate

```
function UseDecelerate() → Int
{ mengembalikan integer sebagai skala prioritas command}
Deklarasi
      curLane : Data sebuah lane
      flagBoost : Apakah terdapat powerup boost di depan
      startBlock    : Indeks blok awal
Algoritma
      Lane[] curLane=gameState.lanes.get(myCar.position.lane-1);
      boolean flagBoost=false;;
      int startBlock=gameState.lanes.get(0)[0].position.block;
      // Mencari apakah terdapat powerup booster
      for (int i = max(myCar.position.block-startBlock+1,0); i <=
      min(getMaxPos(),myCar.position.block-startBlock+getCurSpeed(-1,
      false));i++){
                if (curLane[i] == null || curLane[i].terrain ==
      Terrain.FINISH) {
                    break;
                }
                if(curLane[i].terrain==Terrain.BOOST){
                    flagBoost=true;
```

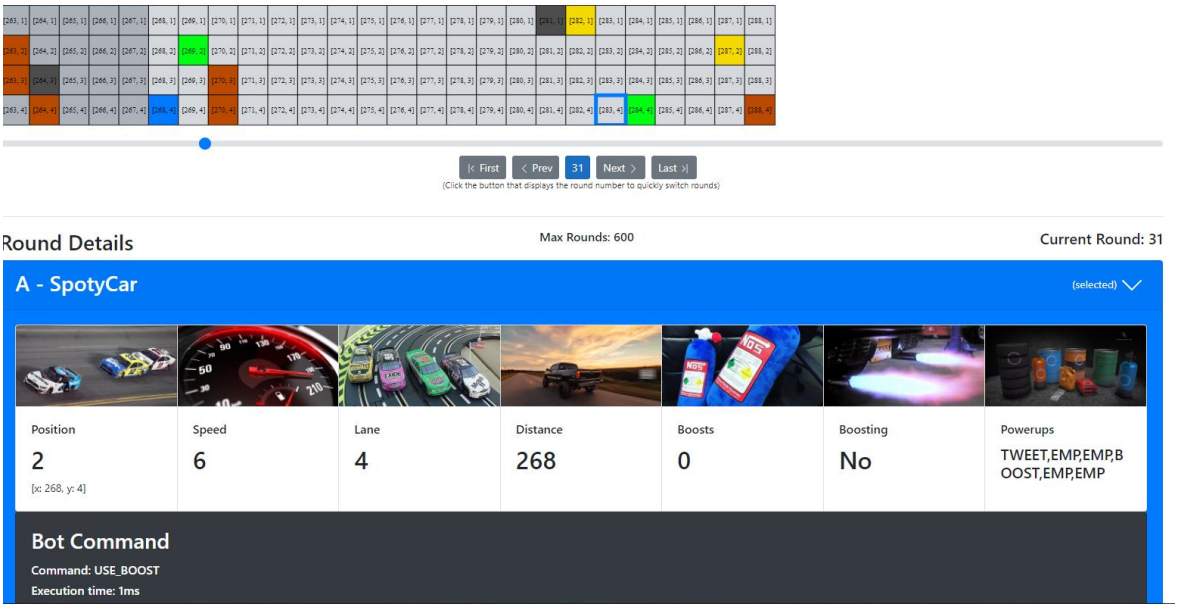
```
                    break;
                }
            }
            if(flagBoost && UseAccelerate()<4){
                return 4;
            }else{
                return 1;
            }
```

4.2 Penjelasan Struktur Data

Pada program Bot, kami menggunakan dua jenis struktur data untuk menyimpan *state* program. Pertama adalah struktur data matriks yang fungsinya untuk menyimpan keseluruhan informasi dari setiap blok yang berada dalam jarak pandang pemain. Matriks ini akan diinisialisasi ketika konstruktor class dipanggil. Selanjutnya kami juga menggunakan struktur data array statis untuk menyimpan keseluruhan informasi blok pada jalur tertentu. Fungsi dari struktur data ini utamanya adalah untuk mengecek apakah terdapat *obstacle* yang menghalangi di depan *car* atau apakah terdapat *power up* yang bisa diambil pengguna pada jalur tertentu.

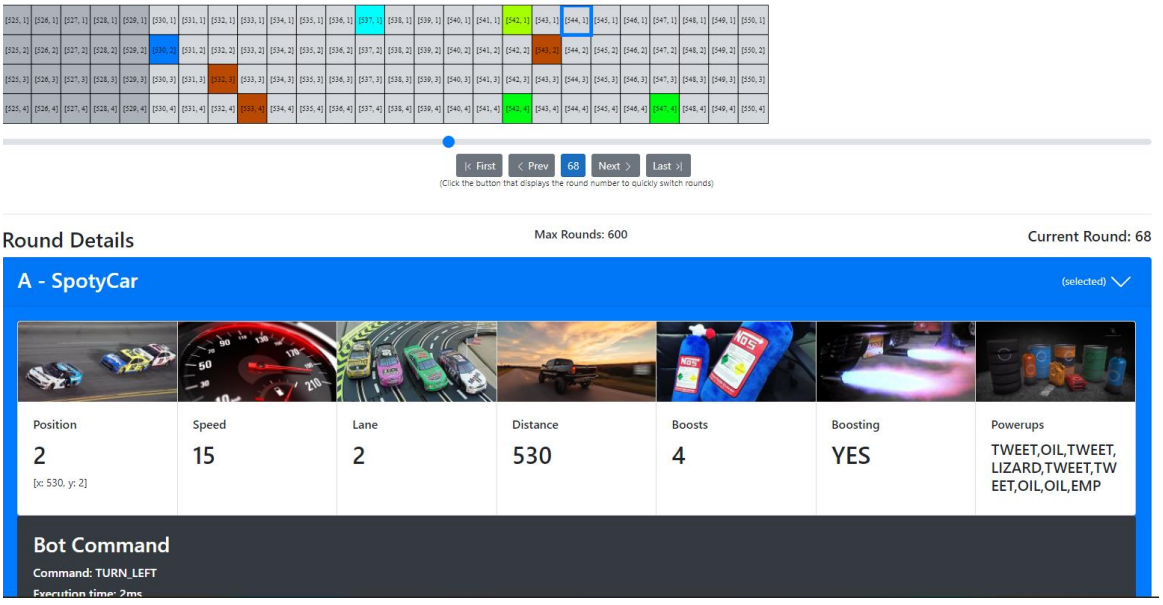
4.3 Analisis Desain Solusi Algoritma

Sesuai strategi yang kami buat, bot kami didesain sedemikian sehingga agar dapat memaksimalkan penggunaan powerup dan kecepatan. Namun, disisi lain car kami masih sering menabrak obstacle di depannya karena yang menjadi fokus dari car kami adalah powerup dan kecepatan, seperti contoh kasus berikut.



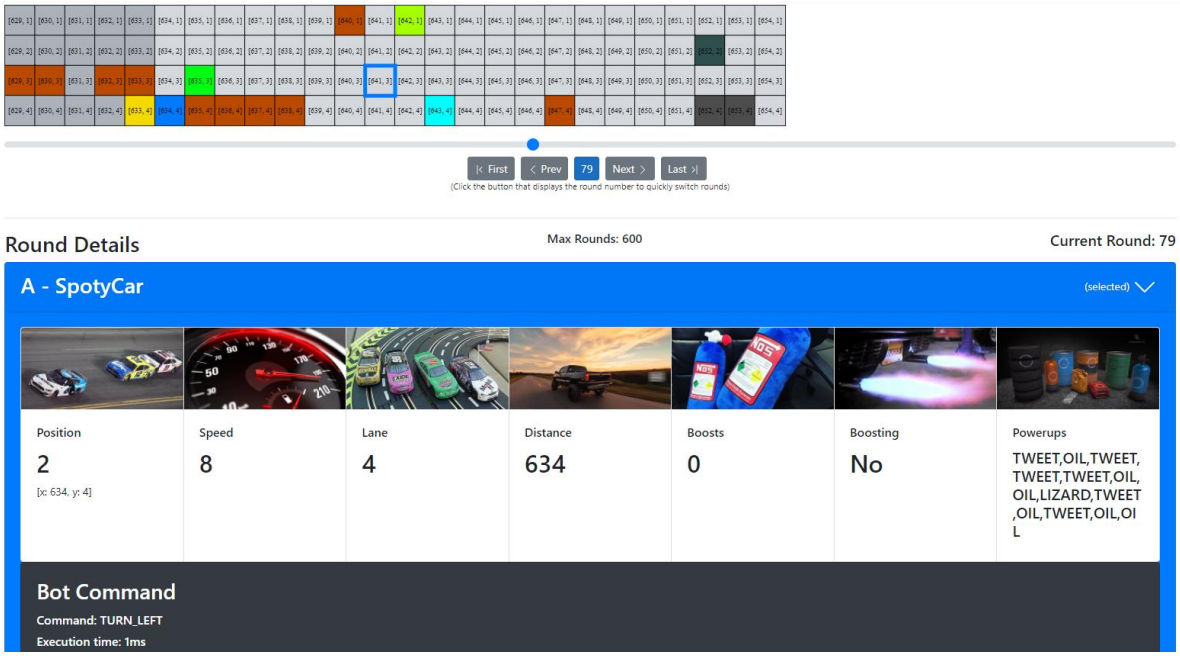
Gambar 4.3.1 Visualisasi Permainan Overdrive

Selain itu, bot kami juga didesain untuk berusaha mendapatkan powerup selagi bisa dilakukan. Karena mendapatkan powerup akan meningkatkan score dari car kita. Hal tersebut seperti pada kasus pada gambar dibawah ini. Saat kasus pada gambar di bawah ini car berusaha mencari tempat yang memiliki powerup yang banyak. Dimana car kami bisa mengecek lane sebelah kirinya, lane di tempat dia, dan lane sebalah kanannya.



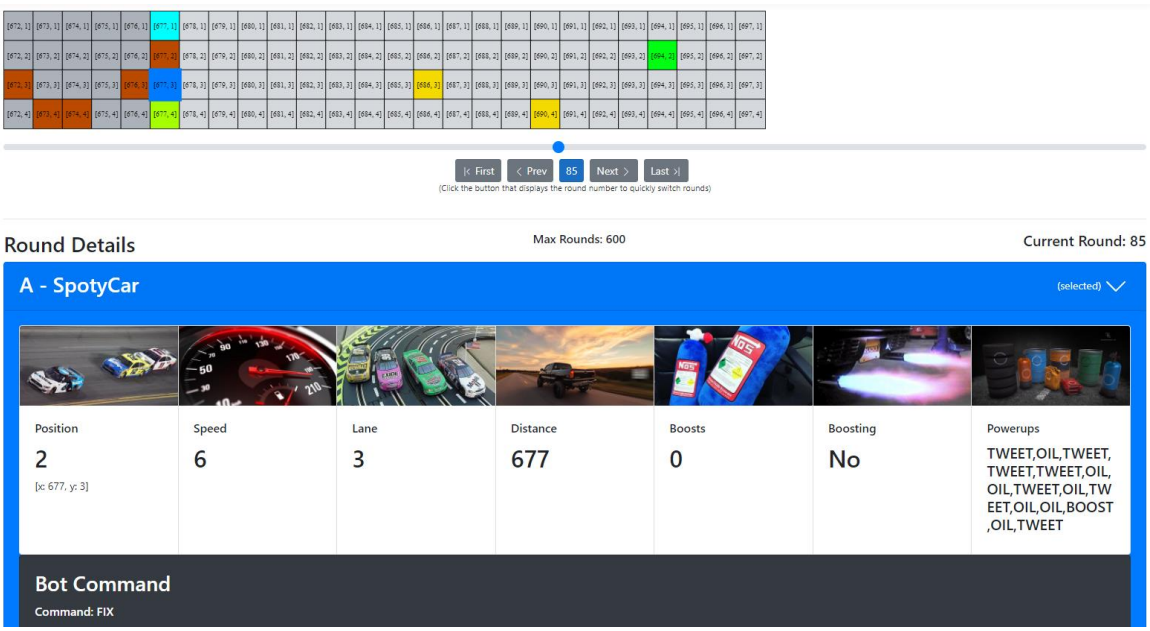
Gambar 4.3.2 Visualisasi Permainan Overdrive

Selain didesain untuk memprioritaskan penggunaan obstacle dan pemaksimalan kecepatan, Car kami juga didesain sedemikian rupa sehingga agar tidak terlalu banyak terkena damage dari segala yang bisa menambah damage pada car kami. Contoh seperti kasus pada gambar di bawah ini, dimana Car kami pindah dari lane yang banyak obstacle menuju lane yang tidak ada obstacle.



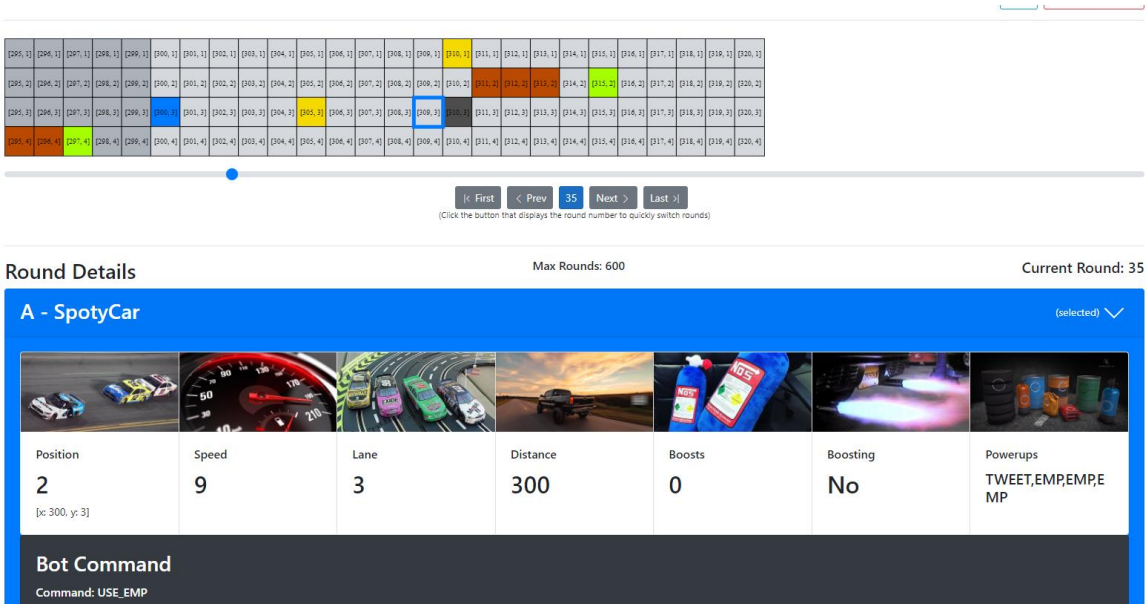
Gambar 4.3.3 Visualisasi Permainan Overdrive

Karena Fix diletakkan pada prioritas teratas, jadi apabila Car kami tiba-tiba terkena damage, maka Fungsi Seleksi kami akan langsung memilih Command Fix untuk meng-healing Car kami agar damage yang diterima dapat diminimumkan. Hal tersebut dapat dilihat dari kasus pada gambar di bawah ini.



Gambar 4.3.4 Visualisasi Permainan Overdrive

Perhatikan bahwa selain berfokus pada diri sendiri, Car kami juga dapat menyerang Car musuh lain. Salah satu cara yang dapat di lakukan oleh Car kami adalah dengan menyerang menggunakan EMP. Hal tersebut dapat dilihat dari kasus pada gambar di bawah ini.



Gambar 4.3.5 Visualisasi Permainan Overdrive

BAB 5
KESIMPULAN DAN SARAN

5.1 Kesimpulan dan Saran

Dalam menyelesaikan persoalan optimasi, algoritma greedy merupakan salah satu alternatif solusi yang cukup efektif untuk digunakan. Salah satu kelebihan dari algoritma greedy yaitu algoritma ini cenderung mudah untuk diimplementasikan dan dipahami cara kerjanya. Meskipun begitu, dalam beberapa persoalan algoritma greedy tidak dapat mencapai solusi optimum global melainkan hanya optimum lokal.

Dalam menyelesaikan persoalan permainan Overdrive ini, secara umum algoritma greedy yang kami implementasikan adalah dengan terus menambah kecepatan atau melakukan boost selama tidak terdapat obstacle, pindah atau lompat jalur jika terdapat obstacle, dan memberikan gangguan kepada lawan bila tidak diperlukan penambahan kecepatan maupun perpindahan jalur. Kelebihan dari algoritma greedy yang kami implementasikan yaitu algoritma kami bisa memanfaatkan powerup secara maksimal terutama powerup boost dan emp. Selain itu, program bot yang kami buat juga dapat memaksimalkan pengambilan powerup dan menghindari obstacle. Sedangkan kelemahan dari program bot yang kami buat yakni bot cenderung lambat di awal permainan karena masih kurangnya powerup dan cenderung baru bisa mengejar di pertengahan permainan. Selain itu, dalam beberapa kasus program bot kami tidak dapat menghindari obstacle disebabkan sifat dari algoritma greedy itu sendiri yang hanya mencari solusi optimum lokal.

DAFTAR PUSTAKA

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

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

https://github.com/EntelectChallenge/2020-Overdrive/blob/master/game-engine/game-rules.md#visibility

LAMPIRAN

Github repository:
https://github.com/primahafiz/tubes-stima-1

Link video Youtube :
https://youtu.be/9PpxbQVzgpo