LAPORAN TUGAS BESAR

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"

Laporan dibuat untuk memenuhi salah satu tugas besar mata kuliah

IF2211 Strategi Algoritma

Disusun oleh:

I Gede Arya Raditya P.	**13520036**
Arik Rayi Arkananta	**13520048**
Ubaidillah Ariq Prathama	**13520085**

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

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. 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

BAB 2
Landasan Teori

A. Algoritma Greedy

Algoritma greedy merupakan jenis algoritma yang menggunakan pendekatan penyelesaian masalah dengan mencari nilai maksimum sementara pada setiap langkahnya. Nilai maksimum sementara ini dikenal dengan istilah *local maximum*. Pada kebanyakan kasus, algoritma greedy tidak akan menghasilkan solusi paling optimal, begitupun algoritma greedy biasanya memberikan solusi yang mendekati nilai optimum dalam waktu yang cukup cepat.

Kelebihan algoritma greedy adalah kompleksitasnya. Jika suatu persoalan dikerjakan dengan algoritma brute force memerlukan kompleksitas $O(n!)$ atau $O(2^n)$, biasanya algoritma greedy hanya membutuhkan kompleksitas $O(n)$ hingga $O(n^2)$ dengan syarat data yang dipakai pada algoritma tersebut sudah sorted. Contoh permasalahan klasik yang dapat diselesaikan dengan permasalahan greedy antara lain : coin exchange problem, activity selection problem, knapsack, minimum spanning tree, dan lain-lain. Walaupun algoritma greedy menghasilkan solusi yang belum tentu optimal tetapi beberapa algoritma greedy memiliki pembuktian matematis yang membuktikan bahwa algoritma tersebut pasti menghasilkan solusi optimal.

B. Cara Kerja Program

Pada dasarnya untuk menjalankan program kita cukup menjalankan file run.bat yang ada dalam folder starter-pack. File run.bat ini akan menjalankan game engine dan dua bot yang akan dipakai dengan melihat file game-runner-config.json. Kita dapat mengubah bot mana yang akan diuji dengan mengubah target file pada game-runner-config.json. Pada tugas besar ini, bot yang akan kita implementasi berada pada folder starter-bot/java. File yang dipakai oleh game engine bukanlah source code dalam bentuk .java melainkan dalam bentuk .jar. Oleh karena itu, kita harus build program kita terlebih dahulu menggunakan pom.xml dan maven project. Jika build berhasil akan menghasilkan sebuah folder target yang juga berisi file .jar yang akan dipakai.

Tugas kita adalah membuat bot yang efektif dan efisien menggunakan algoritma greedy. Permainan overdrive ini dilakukan dalam beberapa ronde. Setiap ronde bot harus memilih command yang akan dilakukan. Pemilihan command inilah yang merupakan algoritma greedy yang perlu kita implementasikan. Algoritma greedy yang diambil bisa saja membiarkan bot untuk mengambil powerup sebanyak mungkin tanpa menghiraukan obstacle atau sebaliknya. Strategi greedy yang kita ambil akan memengaruhi performa bot.

BAB 3
Aplikasi Strategi Greedy

A. Mapping Persoalan Overdrive ke Dalam Elemen Greedy

Pada dasarnya, algoritma greedy dapat dibagi menjadi enam bagian fungsi dan himpunan yang mendefiniskan algoritma tersebut, yaitu :

1. Himpunan Kandidat

   Himpunan semua command yang mungkin dilakukan oleh bot yaitu :

   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

2. Himpunan Solusi

   Himpunan command-command yang sudah terpilih.

3. Fungsi Solusi

   Fungsi yang mengecek apakah sudah sampai finish line.

4. Fungsi Seleksi

   Fungsi dan algoritma yang digunakan untuk memilih command yang dipilih, akan dibahas pada poin berikutnya.

5. Fungsi Kelayakan

   Fungsi yang digunakan untuk mengecek apakah command yang digunakan valid.
   Sebelum menggunakan akan dicek, apakah kita memiliki power up tersebut.
   Sebelum belok kiri atau kanan akan dicek lane kita berada, apakah memungkinkan.

6. Fungsi Objektif

Mencapai finish line sebelum musuh dengan round yang minimum.

B. Alternatif Solusi Algoritma Greedy

Pada bagian ini, supaya analisis efisiensi lebih terlihat akan digunakan kompleksitas waktu. Agar perbedaan antar algoritma juga terlihat lebih jelas akan digunakan variabel sebagai berikut.

- Banyaknya command yang mungkin : N (11)
- Speed saat ini : S
- Panjang lintasan map dalam satu round : P (20)
- Lebar lintasan map : L (4)

1. Greedy by Speed

   Strategi greedy pada alternatif ini adalah memilih command sehingga selisih speed kita selalu lebih besar atau sama dengan speed lawan. Pada strategi ini, kita selalu mempertimbangkan speed lawan pada setiap langkah yang kita ambil. Kita akan menghitung selisih speed akhir kita dengan lawan jika kita melakukan ACCELERATE, EMP, BOOST, dan lain-lain. Command yang menghasilkan speed akhir paling tinggi akan kita pilih.

   Analisis Efisiensi :

   Kita harus mengecek speed kita jika dilakukan 11 command yang ada dan mencatatnya. Kita juga harus mengecek efek yang dilakukan oleh command kita terhadap speed lawan. Oleh karena itu kompleksitasnya adalah $O(2 * N) = O(N)$

   Analisis Efektivitas :

   Pada dasarnya jika speed kita selalu lebih besar dari speed musuh, kita pasti memenangkan pertandingan. Akan tetapi, kita tidak selalu dapat memprediksi lawan dengan benar karena kita tidak bisa tau kondisi map lawan. Selain itu, jika kita menyerang lawan, belum tentu semua serangan kita akan mengenainya. Akibatnya, prediksi selisih yang kita gunakan belum tentu benar.

2. Greedy by Offensive

Strategi greedy pada alternatif ini adalah memilih command offensive jika kita memiliki command offensive (USE_OIL, USE_TWEET, USE_EMP). Inti dari strategi ini adalah membuat musuh tidak bisa accelerate dikarenakan efek serangan kita dan mobil mereka akan terus mendapatkan damage. Jika kita berada di belakang musuh, utamakan penggunaan EMP jika ada. Jika tidak ada, kita bisa meletakkan Cyber Truck tidak jauh di depan musuh. Kita dapat menghitung koordinatnya dengan opponent.block + opponent.speed. Jika kita berada di depan musuh, kita juga akan menggunakan Cyber Truck jika kita milikki dengan logic yang sama. Jika tidak ada, kita dapat meletakkan oil. Terakhir, jika kita tidak memiliki powerup offensive sama sekali, barulah kita menggunakan command lainnya.

Analisis Efisiensi :

Pada algoritma ini kita hanya perlu mengecek apakah kita menggunakan EMP dengan cara mengecek apakah kita berada di belakang musuh dan apakah kita berada di lane yang dekat dengan musuh. Hal ini dapat dicek dalam $O(1)$. Untuk mengecek apakah kita dapat menggunakan TWEET, kita hanya perlu mengecek koordinat dari musuh sehingga dapat dilakukan dalam $O(1)$. Untuk mengecek apakah kita dapat menggunakan OIL cukup dicek apakah kita berada di depan lawan dan dapat dilakukan dalam $O(1)$. Oleh karena itu, kompleksitas algoritma ini adalah $O(1)$.

Analisis Efektivitas :

Sisi positif dari strategi ini adalah lawan tidak bisa memiliki speed yang tinggi. Akan tetapi, kita juga tidak bisa memiliki speed tinggi karena kita tidak fokus untuk menaikkan kecepatan. Selain itu, serangan kita belum tentu mengenai lawan karena bisa saja didodge oleh lawan. Sebenarnya strategi ini bisa saja bagus jika logic dari bot dikombinasikan dengan accelerate.

3. Greedy by Weighted Direction

Strategi greedy pada alternatif ini adalah pada setiap langkah melihat block mana saja yang akan dilewati. Terdapat beberapa kasus untuk konfigurasi block yang akan dilewati yaitu ketika melakukan TURN_LEFT, TURN_RIGHT, USE_LIZARD, ACCELERATE, DECCELERATE, USE_BOOST, dan command sisanya yang akan membuat mobil maju dengan speed sekarang. Jadi terdapat 7 kemungkinan kombinasi

block yang dilewati dalam 1 turn. Kombinasi block ini akan diberi weight setiap blocknya. Parameter yang diperhatikan antara lain adalah kondisi mobil setelah melewati block-block tersebut seperti berapakah speed akhir mobil, damage akhir mobil, jumlah block maju, dan apakah terdapat boost pada block tersebut. Pemilihan command didasarkan pada yang memiliki weight terbaik, jika weight lurus (tanpa ACCELERATE) yang paling baik baru pilih command menyerang terbaik.

Analisis Efisiensi :

Pada algoritma ini akan dicek block sejauh speed mobil pada round ini. Jumlah block yang dicek adalah sesuai speed karena mobil akan maju sejauh speed kecuali mobil menggunakan LIZARD karena hanya perlu dicek 1 block. Oleh karena itu, kompleksitasnya adalah $O(5 * S + 1) = O(S)$. Jika weight lurus (tanpa ACCELERATE) yang paling baik dapat digunakan startegi offensive yang sama dengan poin nomor 2 yang memiliki kompleksitas $O(1)$. Oleh karena itu, kompleksitas dari algoritma ini adalah $O(S)$.

Analisis Efektifivitas :

Algoritma ini sangat baik untuk tetap konsisten mempertahankan kecepatan tanpa mengenai obstacle dan mengurangi damage mobil. Algoritma ini fokus untuk menyelesaikan race secepat mungkin tanpa mementingkan mobil lain. Menyerang lawan merupakan opsi terakhir yang artinya mobil lawan juga bisa melaju tanpa gangguan yang berarti yang merupakan kelemahan algoritma ini. Pada dasarnya algoritma ini akan berjalan dengan baik jika lawan tidak fokus untuk menyelesaikan race dengan cepat.

C. Solusi Yang Dipilih

Jika dilihat dari efisiensi algoritma, setiap algoritma memiliki kompleksitas waktu yang berbeda-beda. Akan tetapi, nilai variabel yang ada cukup kecil sehingga tidak terlalu memengaruhi runtime. Oleh karena itu, bagian efisiensi tidak menjadi pertimabangan utama kami.

Solusi yang kami pilih adalah alternatif ketiga yaitu greedy by weighted direction. Solusi ini kami pilih karena efektivitasnya. Alternatif pertama dan kedua sebenarnya cukup baik, tetapi terdapat satu permasalahan utama yang dimiliki kedua alternatif

tersebut. Masalah tersebut adalah prioritas utama bot mobil bukan untuk menyelesaikan race secepat mungkin ataupun mempertahankan mobil di kecepatan maksimal. Menurut kami, bot seharusnya lebih fokus pada menghindar dan menaikkan kecepatan karena serangan yang dilakukan oleh bot berpotensi cukup besar untuk didodge oleh lawan sehingga sulit untuk diprediksi. Pada alternatif ketiga, faktor yang menentukan kemenangan murni hanya perbandingan kemampuan bot kita dan bot lawan dalam menyelesaikan race secepat mungkin.

Algoritma ini lebih detailnya dibagi menjadi beberapa fungsi. Fungsi run yang merupakan built in dari game ini kami implementasikan dengan cukup sederhana. Jika mobil sudah memiliki damage 5, harus digunakan command FIX terlebih dahulu jika tidak mobil akan stuck dan tidak bisa maju. Jika kecepatan mobil 0 juga harus melakukan ACCELERATE. Jika tidak keduanya maka akan dipanggil fungsi move yang menerima masukkan gameState dan mengeluarkan output berupa command.

Pada fungsi move, kami mempertimbangkan arah manakah yang paling baik untuk dipilih oleh mobil. Seperti yang sudah dijelaskan di atas terdapat 7 kombinasi block yang akan kita lewati yaitu TURN_LEFT, TURN_RIGHT, USE_LIZARD, ACCELERATE, DECCELERATE, USE_BOOST, dan command sisanya. Prioritas urutan yang kami gunakan adalah :

- ACCELERATE/USE_BOOST
- LIZARD
- ACCELERATE/USE_BOOST
- OFFENSIVE (LURUS)
- TURN_LEFT/TURN_RIGHT
- DECELERATE

Prioritas ini akan dipakai pada beberapa kasus khusus ketika weight dari kombinasi block yang ada sama. Selebihnya prioritas kita tentukan menggunakan fungsi bernama countWeight fungsi ini menghitung kombinasi manakah yang terbaik (semakin kecil nilainya semakin baik).

Fungsi countWeight menerima masukan berupa posisi awal mobil berupa lane dan block, speed mobil pada round berikutnya, dan gameState. Speed mobil akan dihitung pada fungsi move sebelum masuk ke fungsi countWeight. Fungsi countWeight akan

melakukan looping pada map yang kita dapatkan dari gameState untuk mengecek block mana saja yang akan kita lewati. Weight yang kami pakai adalah :

- weightPosition = 10
- weightSpeed = 24
- weightDamage = -76
- weightBoost = 102
- weightLizard = 40
- weightEMP = 41
- weightTweet = 20

Weight ini kami dapatkan dari trial and error dan diambil performa terbaik. Jika block yang kita lewati terdapat boost maka akan menambahkan weight sebesar 102 begitu pula dengan yang lainnya. WeightSpeed akan dihitung dengan mengalikan speed akhir. WeightPosition akan dikalikan dengan berapa block bot maju sehingga kami dapat mempertimbangkan untuk kasus menabrak Cyber Truck dan juga collision dengan lawan. WeightDamage juga akan dikalikan dengan damage yang didapat pada round tersebut untuk mempertimbangkan jika ada obstacle di lane tersebut.

Fungsi accelerate akan digunakan ketika kami memilih antara ACCELERATE dan USE_BOOST. Pada intinya USE_BOOST selalu diutamakan jika kami memilikinya dan damage dari mobil kita 0 dan tidak sedang dalam durasi BOOST. Jika kami memiliki BOOST tetapi damage mobil kami tidak 0, akan dilakukan FIX. Jika tidak mempunyai BOOST, akan dipilih ACCELERATE jika maxSpeed masih bisa naik. Jika tidak akan dilakukan FIX.

Bot baru akan melakukan offensive ketika arah yang dipilih lebih baik lurus tanpa melakukan ACCELERATE ataupun USE_BOOST. Pemilihan command ini dilakukan dalam fungsi offensive. Offensive di sini dibagi ketika kita di depan lawan dan di belakang lawan. Ketika di belakang lawan dan selisih lane dengan lawan kurang sama dengan satu, akan digunakan EMP. Jika kita di depan lawan akan diprioritaskan menggunakan TWEET jika ada. TWEET diletakkan dengan memprediksi speed lawan selanjutnya, lalu diletakkan pada block awal lawan + prediksi speed lawan + 1. Jika tidak ada TWEET baru digunakan OIL.

BAB 4
Implementasi dan Pengujian

A. Pseudocode

Pada subbab ini akan diberikan pseudocode dari fungsi-fungsi utama pada file `Bot.Java` beserta komentarnya. Fungsi selain jalan kerja utama bot (fungsi perhitungan dan sebagainya) tidak kami masukkan karena akan sangat panjang. Untuk kode lengkapnya bisa dilihat di *source code*.

```
function run(gameState : GameState) → Command
Kamus Lokal
       myCar : Car
       opponent : Car
Algoritma
       // Persiapan data yang dibutuhkan
       myCar ← gameState.player
       opponent ← gameState.opponent

       // Jika damage dari mobil sudah 5 atau lebih besar
       // maka akan langsung me-return command FIX
       if (myCar.damage ≥ 5) then
             → FIX
       // Jika speed dari mobil sudah 0 atau lebih kecil
       // maka akan langsung masuk ke fungsi accelerate yang akan
       // me-return tipe data command
       if(myCar.speed ≤ 0) then
            → accelerate(gameState)

       // Jika posisi pemain ada di belakang musuh akan dipilih antara
       // masuk ke fungsi move yang akan me-return tipe data command
       → move(gameState)
```

```
function accelerate(gameState : GameState)→ Command
Kamus Lokal
       myCar : Car
       maxSpeed : array of int
Algoritma
       // Persiapan data yang dibutuhkan
       myCar ← gameState.player
       maxSpeeed = [15, 9, 8, 6, 3, 0]

       // Jika kecepatan pemain dan musuh sama, maka akan dipilih antara
       // memanggil fungsi offensive yang akan me-return tipe data Command,
       // me-return Command BOOST, atau me-return Command FIX tergantung dari
       // damage dari pemain dan ketersediaan powerUps BOOST
       if (myCar.speed = maxSpeeed[myCar.damage]) then
             if (myCar.damage = 1 and
              not hasPowerUp(PowerUps.BOOST, myCar.powerups)) then
                   → offensive(gameState)
            else if (myCar.damage = 0 and
                hasPowerUp(PowerUps.BOOST, myCar.powerups) and
                myCar.boostCounter = 0) then
                   → BOOST
```

```
            else if(myCar.damage = 0
                    → offensive(gameState)
            else then
                    → FIX;
       // Jika kecepatan pemain dan musuh tidak sama, maka akan dipilih
       // antara memanggil fungsi offensive yang me-return tipe data Command
       // , me-return Command BOOST, atau me-return Command ACCELERATE
       // tergantung dari damage pemain, ketersediaan powerUps BOOST, dan
       // status powerUps BOOST
       else then
            if (myCar.damage = 0 and
               hasPowerUp(PowerUps.BOOST, myCar.powerups) and
               myCar.boostCounter = 0) then
                    → BOOST
            else if (myCar.damage = 0 and
               hasPowerUp(PowerUps.BOOST, myCar.powerups) and
               myCar.boostCounter != 0) then
                    → offensive(gameState)
            else if (myCar.damage = 0 and
               not hasPowerUp(PowerUps.BOOST, myCar.powerups) and
               myCar.speed = 9) then
                    → offensive(gameState)
            else then
                    → ACCELERATE
```

```
function move(gameState : GameState)→ Command
Kamus Lokal
       myCar : car
       sepid : int
       sepidEkseleret : int
       weightKiri : double
       weightKanan : double
       weightLurus : double
       weightUjungTengah : double
       weightAccelerate : double
       weightDecelerate : double

Algoritma
       // Persiapan data yang dibutuhkan
       myCar ← gameState.player
       sepid ← myCar.speed
      sepidEkseleret ← myCar.speed

       // Pengubahan variabel sepid dan sepidEkseleret untuk kondisi tertentu
       // untuk penggunaan powerUps Boost
      if (myCar.speed = 15 and myCar.boostCounter = 1) then
            sepid ← 9
       if ((myCar.speed = 9 or myCar.speed = 8) and myCar.damage = 0) then
            sepidEkseleret ← 15
       else then
            sepidEkseleret ← nextSpeed[sepid]

       // Perhitungan weight semua kemungkinan gerak menggunakan fungsi
       // countWeight dan countWeightLizard
       weightKiri ← countWeight(myCar.position.lane - 1, myCar.position.block
- 1, sepid, gameState)
      weightLurus ← countWeight(myCar.position.lane, myCar.position.block,
sepid, gameState)
       weightKanan ← countWeight(myCar.position.lane + 1, myCar.position.block
- 1, sepid, gameState)
```

```
      weightUjungTengah ← countWeightLizard(myCar.position.lane,
myCar.position.block, sepid, gameState)
      weightAccelerate ← countWeight(myCar.position.lane,
myCar.position.block, sepidEkseleret, gameState
      weightDecelerate ← countWeight(myCar.position.lane,
myCar.position.block, prevSpeed[sepid], gameState)

      // Jika weightAccelerate paling besar maka akan memanggil fungsi
accelerate
      // yang akan me-return tipe data Command
     if (weightAccelerate ≤ weightKiri and
        weightAccelerate ≤ weightKanan and
        weightAccelerate ≤ weightUjungTengah and
       weightAccelerate ≤ weightLurus and
        weightAccelerate ≤ weightDecelerate) then
           → accelerate(gameState)
      // Jika weightUjungTengah paling besar dan mempunyai powerUps Lizard
      // maka akan me-return Command LIZARD untuk melompat
      else if (weightUjungTengah ≤ weightKanan and
        weightUjungTengah ≤ weightKiri and
        weightUjungTengah < weightLurus
        weightUjungTengah ≤ weightDecelerate and
        hasPowerUp(PowerUps.LIZARD, myCar.powerups)) then
           → LIZARD
      // Jika weightUjungTengah tetap paling besar namun tidak memiliki
      // powerUps Lizard maka akan memanggil fungsi accelerate yang
      // akan me-return tipe data Command
      else if (weightAccelerate ≤ weightKiri and
        weightAccelerate ≤ weightKanan and weightAccelerate ≤ weightLurus and
       weightAccelerate ≤ weightDecelerate) then
           → accelerate(gameState)
      // Jika weightLurus paling besar maka akan memanggil fungsi offensive
      // yang akan me-return tipe data Command
     else if (weightLurus ≤ weightKanan and
        weightLurus ≤ weightKiri and
        weightLurus ≤ weightDecelerate) then
           → offensive(gameState)
      // Jika weightKanan dan weightKiri sama besar maka akan dipilih yang
      // paling mendekati lane tengah
      else if (weightKanan = weightKiri and weightKiri ≤ weightDecelerate)
then
           if (myCar.position.lane ≤ 2) then
                  → TURN_RIGHT
           else then
                  → TURN_LEFT
      // Jika weightKanan paling besar maka akan me-return Command TURN_RIGHT
     else if (weightKanan < weightKiri and weightKanan ≤ weightDecelerate)
then
           → TURN_RIGHT
      // Jika weightKiri paling besar maka akan me-return Command TURN_LEFT
     else if (weightKiri < weightKanan and weightKiri ≤ weightDecelerate)
then
           → TURN_LEFT
      // Jika weightDecelerate paling besar maka akan me-return Command
DECELERATE
      else then
            → DECELERATE
```

```
function offensive(gameState : GameState)→ Command
```

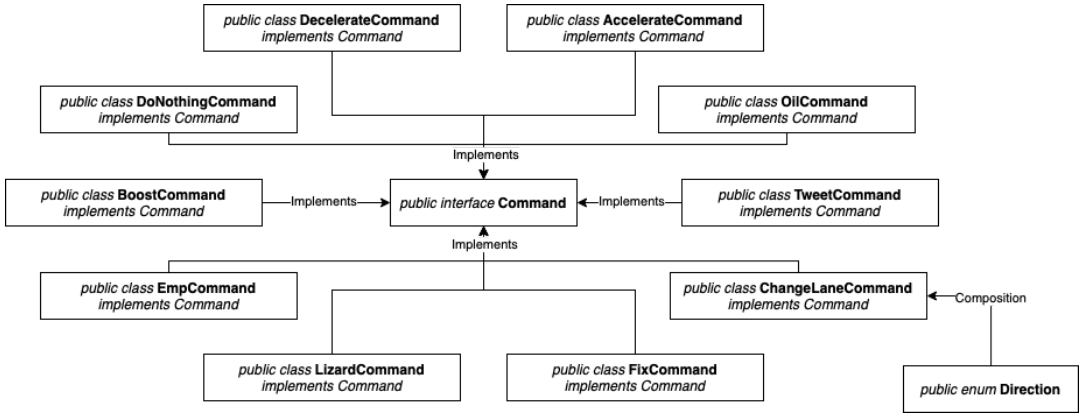
```
Kamus Lokal
       myCar : car
       opponent : car
Algoritma
       // Persiapan data yang dibutuhkan
       myCar ← gameState.player
       opponent ← gameState.opponent

       // Jika pemain ada di belakang musuh akan dipilih Command EMP jika
       // pemain memiliki powerUps EMP dan ada di lane dengan jarak maksimal 1
       if (myCar.position.block < opponent.position.block) then
              if (hasPowerUp(PowerUps.EMP, myCar.powerups)) then
                     if(absolute(myCar.position.lane - opponent.position.lane)
                     ≤ 1)
                       then
                            → EMP
              → DO_NOTHING
       // Jika posisi pemain sama dengan musuh maka tidak akan dilakukan
       // apa-apa
       else if (myCar.position.block = opponent.position.block) then
              → DO_NOTHING
       else then
              // Jika pemain memiliki powerUps TWEET musuh ada jauh dibelakang
              // pemain, fungsi akan me-return Command TWEET
              if (hasPowerUp(PowerUps.TWEET, myCar.powerups) and
                opponent.position.block + nextSpeed[opponent.speed] + 1 ≤
                myCar.position.block) then
                     → TweetCommand(opponent.position.lane,
                       opponent.position.block + nextSpeed[opponent.speed] + 1)
               // Jika pemain ada di depan musuh namun tidak terlalu jauh,
               // fungsi akan me-return Command OIL
              if (hasPowerUp(PowerUps.OIL, myCar.powerups)) then
                     → OIL
               // Jika tidak ada kondisi yang memenuhi, program akan me-return
               // Command DO_NOTHING
              → DO_NOTHING
```

B. Struktur Data

Terdapat dua struktur data pada game *overdrive* yang berisikan atribut-atribut informasi dari satu ronde game dan juga yang akan digunakan untuk *command* atau perintah yang akan digunakan untuk ronde selanjutnya. Berikut adalah rincian struktur data tersebut:

1. Commands



Setiap putaran, ada sepuluh kelas perintah bawaan yang akan digunakan untuk mengirimkan perintah ke permainan. Metode *render* akan diberikan dengan perintah, yang harus di-*override*.

Perintah pertama, AccelerateCommand berguna untuk menambahkan *speed* kita satu tingkatan, namun ketika kita sudah di posisi *max speed* perintah ini tidak akan berjalan.

Perintah kedua, DecelerateCommand berguna untuk menurunkan *speed* kita satu tingkatan sampai mencapai *speed minimum* lalu perintah ini tidak bisa lagi digunakan atau tidak akan berfungsi.

Perintah ketiga, BoostCommand berguna untuk mengubah speed kita menjadi *max speed* dalam satu perintah, keunggulan dari perintah ini adalah kita bisa melebih batas maksimum *non-boost speed*, yaitu 15.

Perintah keempat, FixCommand berguna untuk mengurangi damage kita sebesar 2 hingga 0. Perintah ini sangat berguna karena jika tidak memperbaiki mobil maka mobil akan berjalan sangat lambat atau bahkan bisa mencapai 0.

Perintah kelima, DoNothingCommand berarti kita tidak akan melakukan apa apa selain maju sesuai keadaan speed sekarang.

Perintah keenam, EmpCommand berguna untuk menembak sebuah tembakan sejauh tak terhingga dan ketika tembakan ini mengenai musuh, maka kecepatan musuh akan otomatis berubah menjadi kecepatan minimum yaitu 3.

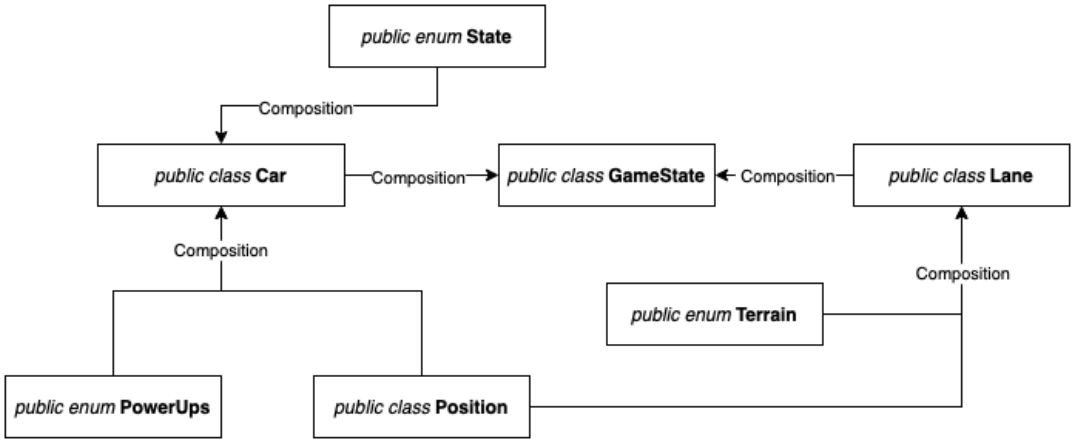
Perintah ketujuh, TweetCommand berguna untuk menempatkan sebuah *obstacle* berbentuk truk yang jika ditabrak oleh musuh akan menambahkan damage musuh sebesar 2, mengurangi *speed* musuh menjadi 3, dan musuh akan terdiam satu *round* pada satu block dibelakang truk.

Perintah kedelapan, OilCommand berguna untuk menumpahkan *obstacle* berbentuk oli di *block* mobil *player* saat ini dan jika oil tersebut dilewati oleh musuh akan memiliki efek seperti *obstacle mud* yaitu damage sebesar 1 dan mengurangi *speed* sebesar satu tingkatan.

Perintah kesembilan, LizardCommand berguna untuk meloncati seluruh *block* sebanyak *speed-1* kedepan lalu berhenti pada *block* ujung sesuai *speed*. Perintah ini berguna untuk menghindar dari segala *obstacle* yang ada di depan kita.

Perintah kesepuluh, ChangeLandCommand berguna untuk berpindah *lane* sebesar satu ke kiri atau ke kanan. Perintah ini memanfaatkan *Class* lain pada *game* ini, yaitu *class Direction*. Perintah ini biasanya digunakan jika player tidak memilih untuk di *lane* yang sama karena ada *obstacle* yang harus dihindari.

2. Entities

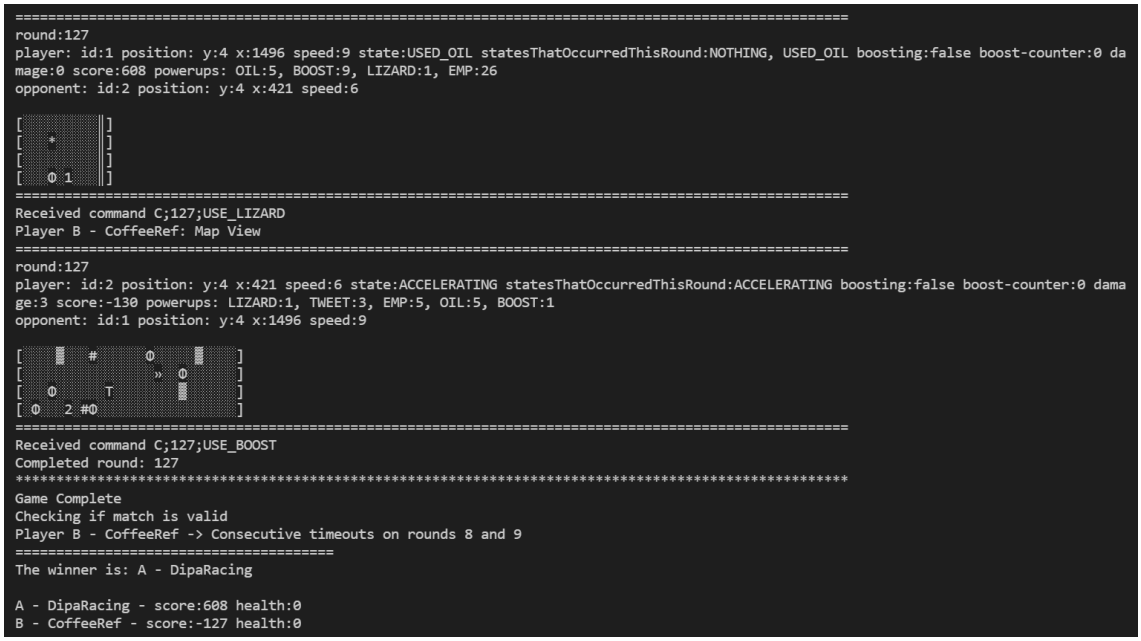


Entities merupakan class yang berisikan objek-objek yang dapat di instansiasi pada game overdrive. Class paling utama dalam game adalah *GameState*. Pada *GameState,* player dapat mengetahui *position* player dan lintasan disekitarnya, yaitu 5 *block* ke belakang dan 20 *block* ke depan. Setiap *block* pada lintasan dinyatakan sebagai class Lane yang terdiri dari position, terrain, isOccupiedByCyberTruck, dan occupiedByPlayerId. Dari GameState kita bisa mengetahui data player kita dan musuh yang berbentuk *Car* juga peta pada game. Dari data *Car* tersebut kita bisa mengetahui data mobil kita maupun mobil musuh. Data mobil yang dimiliki oleh musuh tidak bisa semuanya kita ketahui seperti *PowerUps*, *Damage*, dan status *Boosting*. *PowerUps* pada permainan ini terdiri dari Boost, Oil, Tweet, Lizard, dan EMP yang sudah dijelaskan pada struktur data sebelum ini.

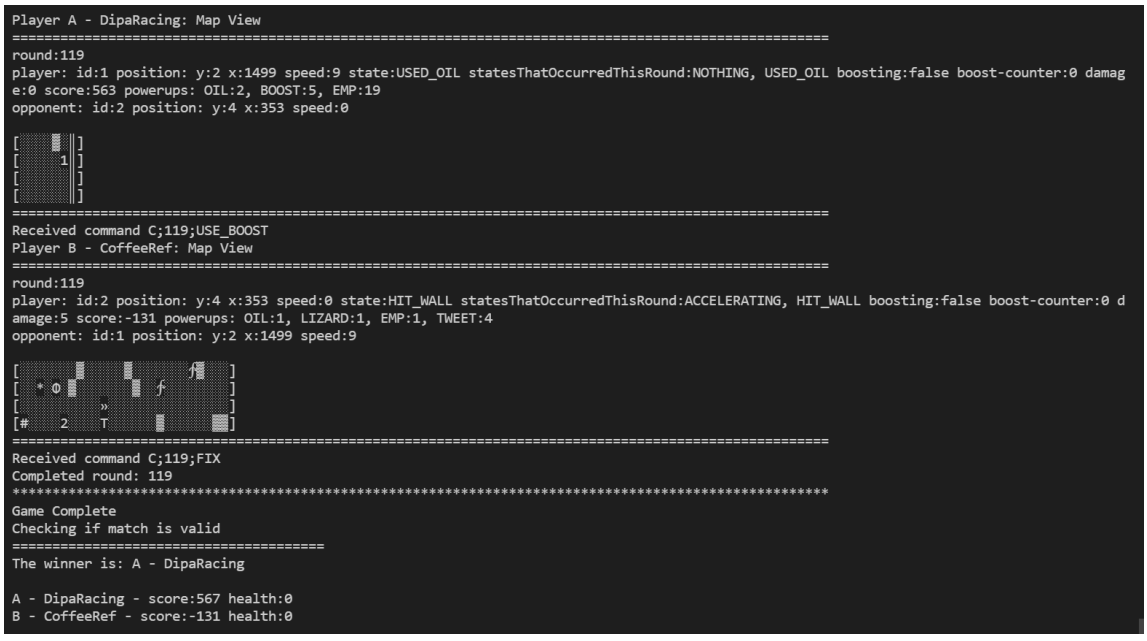
Class terrain terdiri dari EMPTY, MUD, OIL_SPILL, OIL_POWER, FINISH, BOOST, WALL, LIZARD, TWEET, dan EMP. Class position terdiri dari lane dan block. Sedangkan class isOccupiedByCyberTruck memiliki nilai boolean true jika block terisi oleh *Cyber Truck* dan false jika tidak terisi oleh *Cyber Truck.* Dan terakhir occupiedByPlayerId bernilai 1 jika diisi oleh player dengan id 1, bernilai 2 jika diisi oleh player dengan id 2, dan bernilai 0 jika tidak diisi oleh player manapun.

C. Analisis dan Pengujian

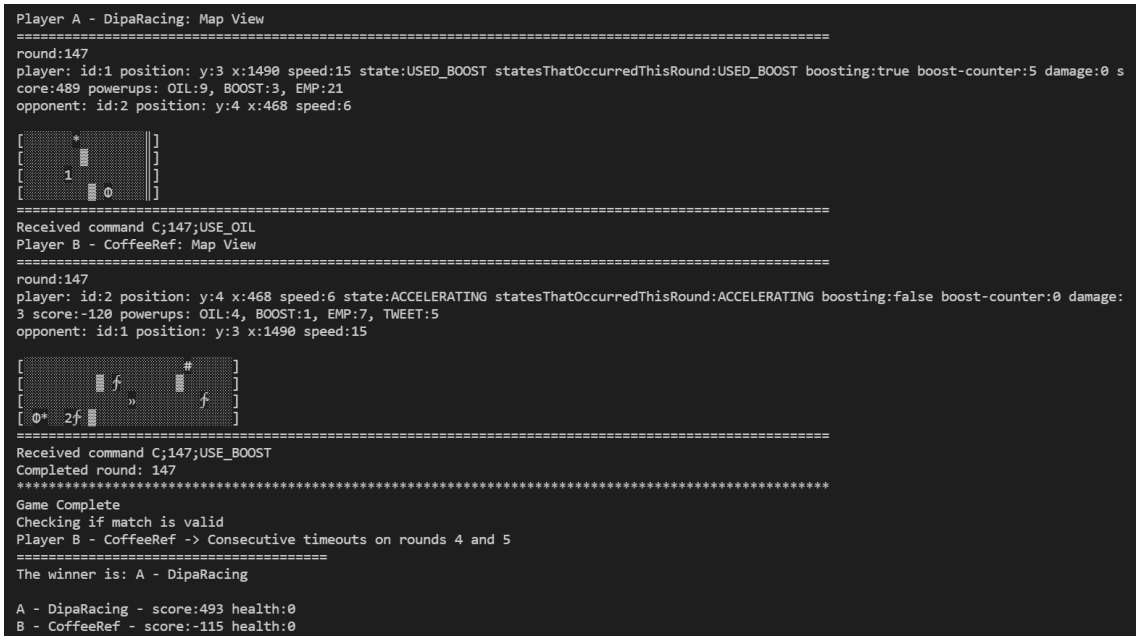
1. Pengujian 1



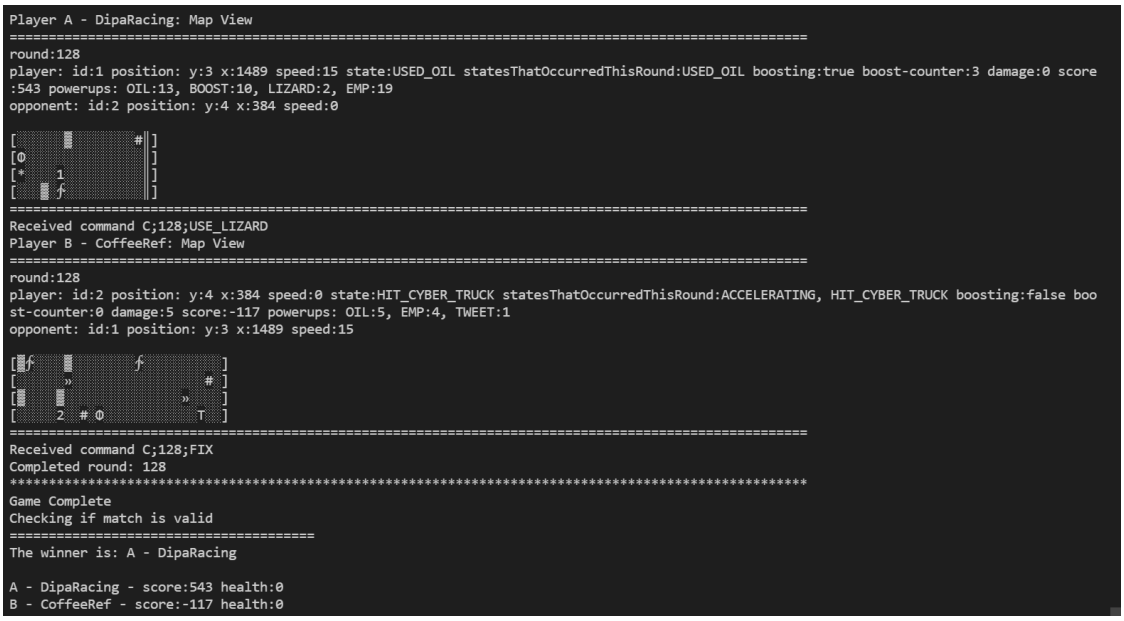
2. Pengujian 2



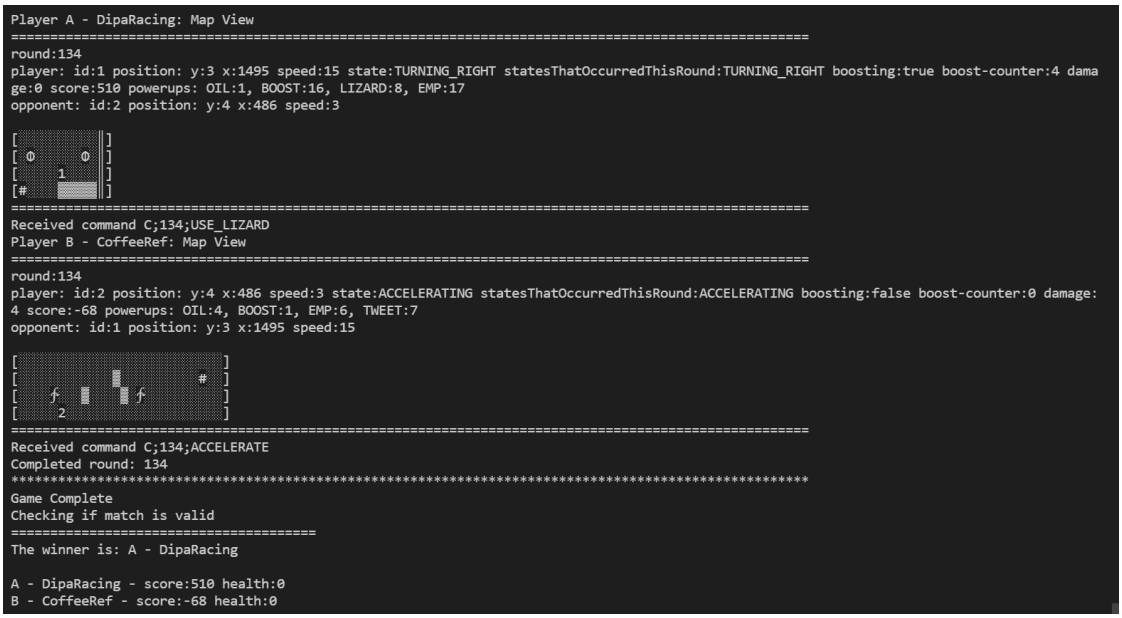
3. Pengujian 3



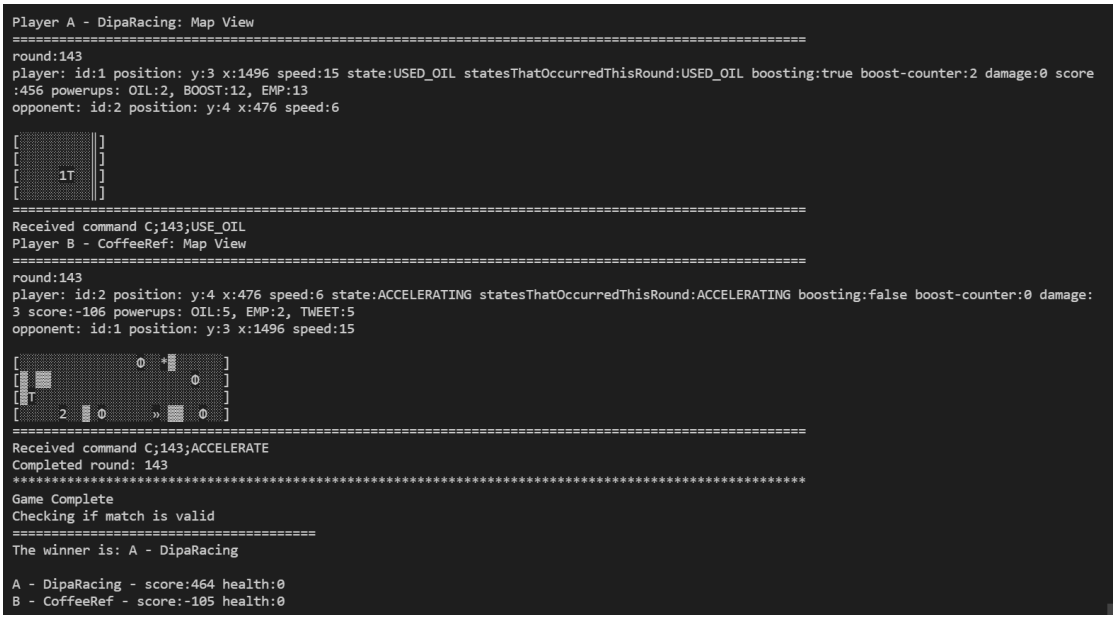
4. Pengujian 4



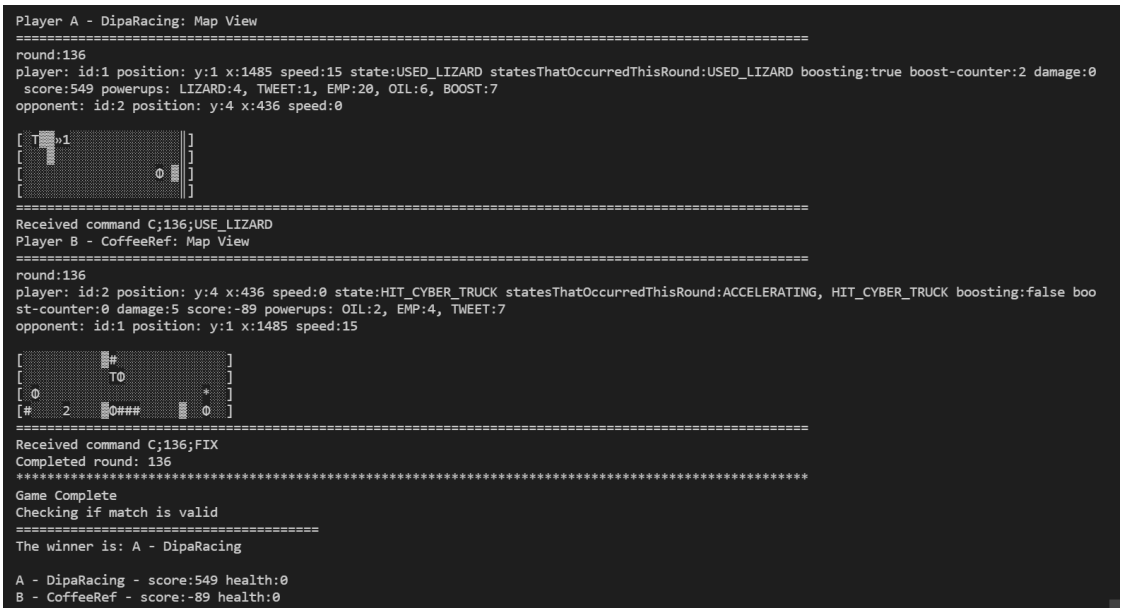
5. Pengujian 5



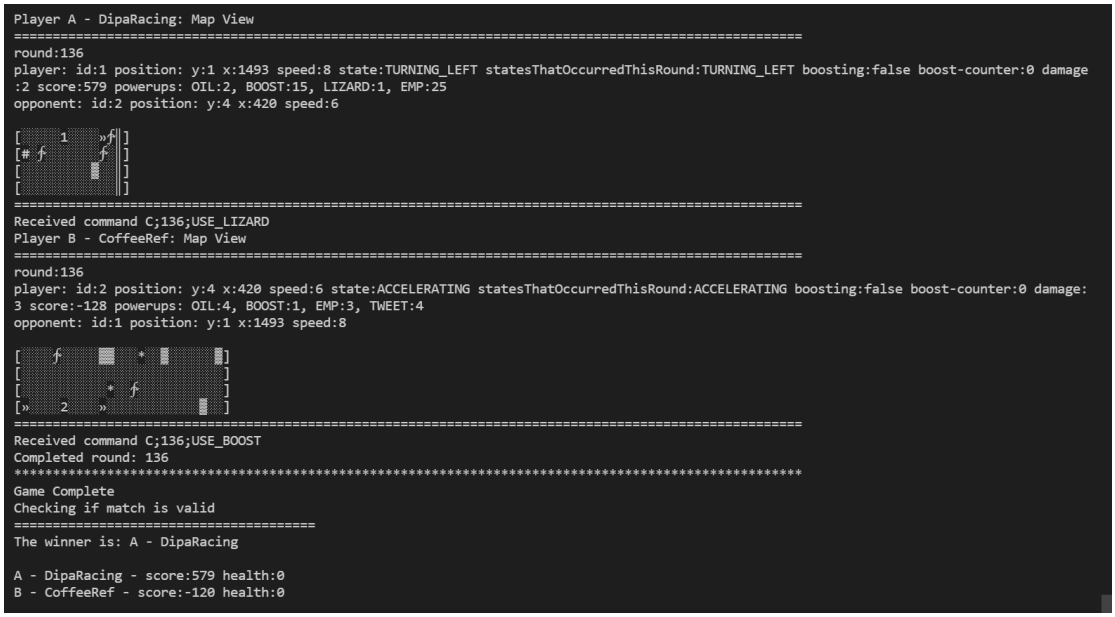
6. Pengujian 6



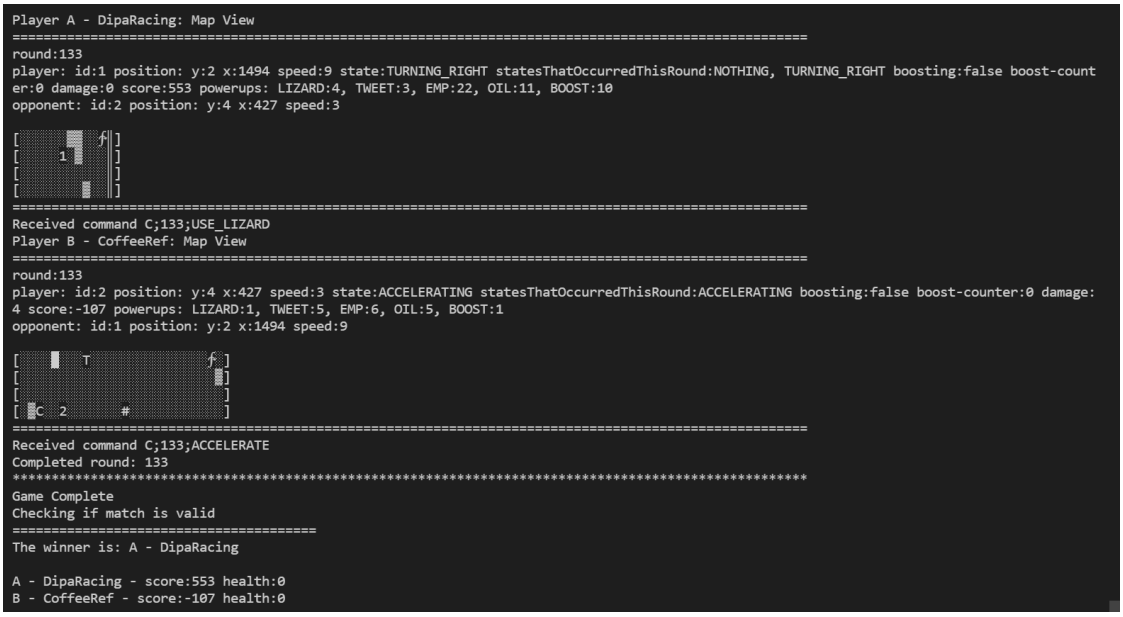
7. Pengujian 7



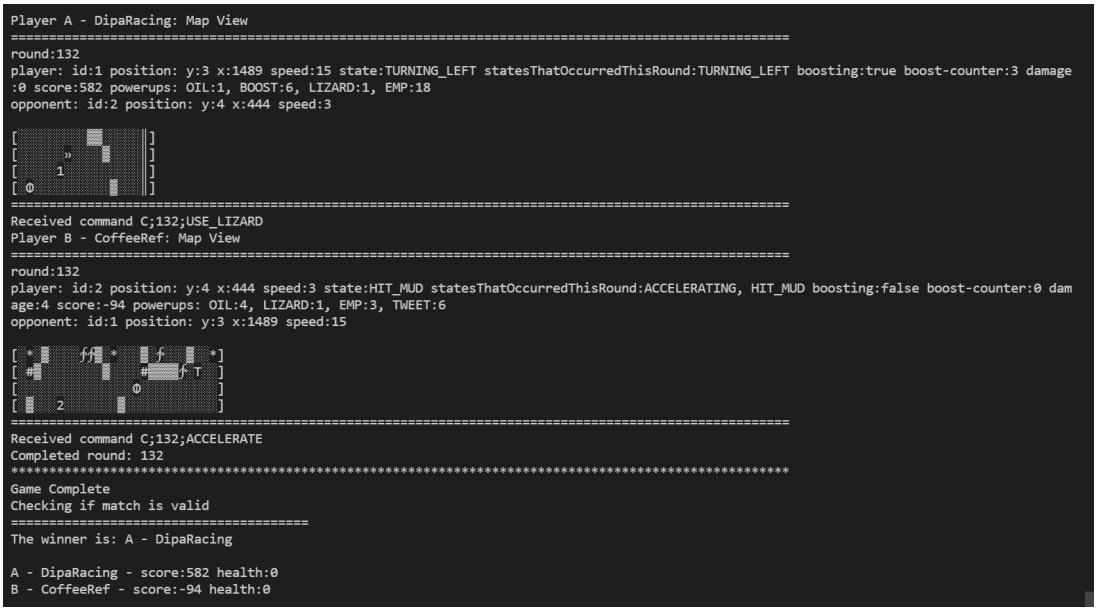
8. Pengujian 8



9. Pengujian 9



10. Pengujian 10



Berdasarkan pengujian yang telah kami laksanakan sebanyak 10 kali melawan bot *coffeeref* yang merupakan bot bawaan. Bot kami menang sebanyak 10 kali dari 10 percobaan. Sehingga, bisa dikatakan bahwa strategi *greedy* yang kami jalankan cukup optimal. Namun, jika dibandingkan dengan bot-bot buatan peserta lomba Entelect, akan tidak seoptimal bot mereka karena mereka menggunakan *machine learning* yang menggunakan komputer untuk mendapatkan nilai optimal yang sesungguhnya.

BAB 5
Kesimpulan dan Saran

A. Kesimpulan

Secara keseluruhan, bot yang diprogram dapat memenangkan 10 dari 10 permainan yang diuji dalam sekitar 120-150 ronde dan selisih sekitar 1000 block melawan bot referensi. Persentase kemenangan bernilai 100%, bot telah menunjukan kemampuan untuk menang melawan bot referensi. Selain itu, implementasi algoritma greedy dalam bot yang digunakan juga sudah berjalan. Hal ini juga membuktikan bahwa algoritma greedy dapat memiliki tingkat efektivitas yang tinggi dan tingkat efisiensi yang baik.

B. Saran

Secara umum, tugas sudah berjalan dengan baik. Walau begitu, beberapa hal untuk meningkatkan performa yang sudah ada masih dapat dilakukan. Untuk kedepannya, algoritma untuk countWeight masih dapat terus diperbaiki nilainya dengan melakukan testing yang lebih banyak lagi dan melihat alternatif yang terbaik. Selain itu, penggunaan komentar yang lebih baik mendeskripsikan setiap proses dapat ditulis oleh pemrogram. Mungkin juga, untuk kedepannya, bahasa pemrograman yang digunakan dapat bebas dipilih mengingat banyaknya variasi bahasa untuk bot pada program ini. Secara umum, tim ini sudah mengeluarkan usaha paling maksimal untuk keberjalanan program dan tugas dikerjakan dengan baik. Atas perhatian pembaca dalam membaca laporan ini, penulis mengucapkan terima kasih.

Daftar Pustaka

EntelectChallenge, B. (2020). Entelectchallenge/2020-Overdrive. https://github.com/EntelectChallenge/2020-Overdrive. (Entellect Github)

Munir, Rinaldi (2022). Algoritma Greedy Bagian 1, 2, dan 3. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/stima21-22.htm#SlideKuliah.

Lampiran

Link Github : https://github.com/gedearyarp/DipaRacing-Entellect-Challenge-2020

Link Youtube : https://youtu.be/hViNhhYMC8A