LAPORAN TUGAS BESAR I

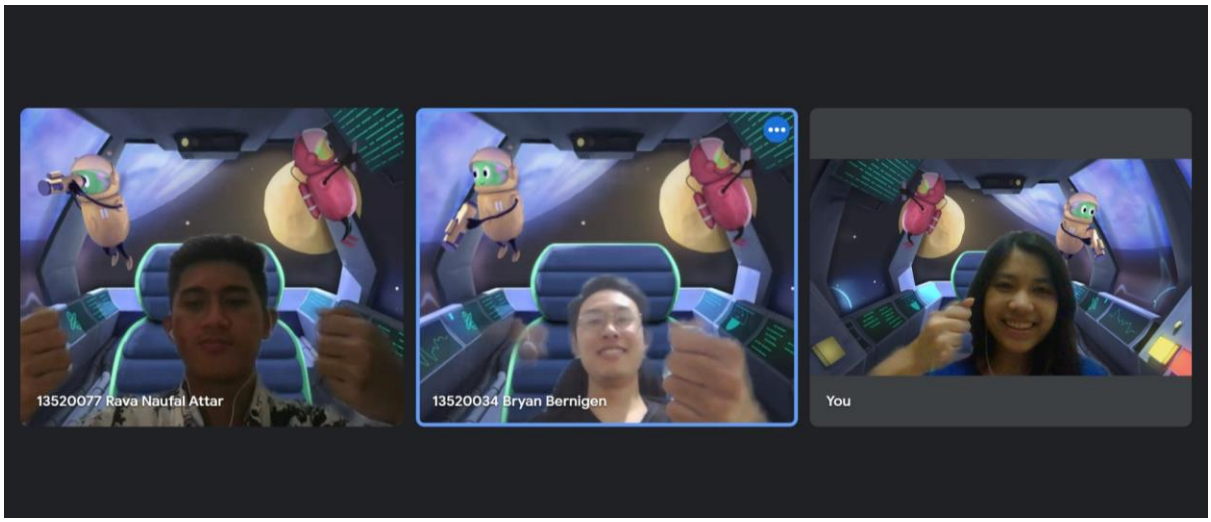
Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan

Mata Kuliah Strategi Algoritma (IF2211)

KELAS 02

Dosen : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.



DISUSUN OLEH:

Kelompok 54

"Ngeeeng"

Anggota:

Bryan Bernigen (13520034)

Felicia Sutandijo (13520050)

Rava Naufal Attar (13520077)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2021/2022

Daftar Isi

BAB I
Deskripsi Tugas

Overdrive adalah sebuah *game* yang mempertandingkan 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 II
Landasan Teori

2.1. Dasar Teori

Algoritma *Greedy* merupakan algoritma yang digunakan untuk memecahkan persoalan dengan memilih pilihan optimum lokal pada setiap langkahnya. *Greedy* sendiri merupakan kata Bahasa Inggris yang berarti ‘rakus’, ‘tamak’, atau ‘loba’. Pada mayoritas kasus, algoritma *Greedy* tidak menghasilkan solusi yang terbaik, namun dapat menghasilkan solusi optimum lokal yang harapannya mengarah kepada solusi optimum global.

Elemen-elemen dari algoritma Greedy adalah sebagai berikut.

1. **Himpunan kandidat, C** : berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
2. **Himpunan solusi, S** : berisi kandidat yang sudah dipilih
3. **Fungsi solusi** : menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. **Fungsi seleksi s(*selection function*)** : memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
5. **Fungsi kelayakan (*feasible*)** : memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. **Fungsi objektif** : memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi objektif.

Algoritma *Greedy* secara umum dapat ditulis dalam *pseudocode* sebagai berikut.

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
    x : kandidat
    S : himpunan_solusi

Algoritma:
    S ← {} { inisialisasi S dengan kosong }
    while (not SOLUSI(S)) and (C ≠ {}) do
        x ← SELEKSI(C) { pilih sebuah kandidat dari C}
        C ← C - {x} { buang x dari C karena sudah dipilih }
        if LAYAK(S ∪ {x}) then { x memenuhi kelayakan untuk dimasukkan
                                 ke dalam himpunan solusi }
            S ← S ∪ {x} { masukkan x ke dalam himpunan solusi }
        endif
    endwhile
    { SOLUSI(S) or C = {} }

    if SOLUSI(S) then { solusi sudah lengkap }
        return S
    else
        write("tidak ada solusi")
    endif
```

*Greed*y memiliki dua property ini:

1. *Greedy choice property*

   Algoritma *Greedy* merupakan algoritma yang efektif pada persoalan-persoalan yang dapat diselesaikan dengan membuat pilihan demi pilihan yang optimum pada saat itu saja, tanpa mempertimbangkan ‘masa lalu’ ataupun ‘masa depan’.

2. Substruktur yang optimal

   Sebuah permasalahan yang memiliki solusi optimum global yang mengandung solusi optimum lokal merupakan permasalahan yang memiliki substruktur yang optimal.

2.2. Cara Kerja Program

Sebelum merancang program, beberapa *prerequisites* yang diperlukan sebagai berikut:

1. Java (minimal Java 8) : *https://www.oracle.com/java/technologies/downloads/#java8*
2. Intellij IDEA : *https://www.jetbrains.com/idea/*
3. NodeJS : *https://www.jetbrains.com/idea/*

Pada program, bot melakukan aksinya yaitu dengan mengembalikan suatu *command* yang akan dilakukan disetiap ronde permainan. Pada bot tersebut yang terdapat di file `Bot.java`, terdapat strategi tersendiri untuk memutuskan *command* apa yang akan dipilih oleh bot untuk mendapatkan kondisi yang optimal.

Untuk mengimplementasikan algoritma *greedy* ke dalam bot, dapat dilakukan dengan membuat perubahan dan strategi terkait pada file `Bot.java`. Bot akan melakukan aksinya dengan menggunakan fungsi `public Command run(Gamestate gamestate)` di file `Bot.java`. Fungsi ini yang akan mengembalikan *command* pada bot untuk melakukan aksinya disetiap ronde permainan.

Untuk melakukan perubahan keterangan seperti *author*, *email*, dan *nickName* dapat membuat konfigurasi sendiri pada file `bot.json` yang terletak di *path* `src/java`.

Untuk melakukan *build* program, dapat dilakukan langkah-langkah sebagai berikut:

1. Buka Intellij IDEA
2. Buka folder `src/java`
3. Klik `maven` pada bagian kanan Intellij IDEA
4. Klik `java-starter-bot`
5. Klik `Lifecycle`
6. Klik *install* dua kali (atau dapat menekan tombol *play button* pada bagian *install*)
7. Program akan berhasil di *build* pada folder target di `src/java`

Untuk menjalankan *game engine* dapat dilakukan langkah-langkah sebagai berikut:

1. *Download* `starter-pack.zip` yang terdapat pada tautan: *https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4*
2. Ekstrak folder `.zip` tersebut pada *path* yang diinginkan
3. Buka folder strater-pack yang telah diekstrak
4. Edit file `game-runner-config.json` dengan mengubah folder *path* pada “player-a” atau “player-b” menjadi *path* `src/java` di tempat hasil rancangan `Bot.java` sebelumnya.

   Sebagai contoh bila percanganan bot dilakukan pada file `C:`, maka tampilan konfigurasi sebelum dan sesudah akan seperti gambar berikut

```
"player-a": "./starter-bots/java",
"player-b": "./reference-bot/java",
```

Gambar 2.2.1. sebelum

```
"player-a": "C:/Tubes1_Ngeeeng/src/java",
"player-b": "./reference-bot/java",
```

Gambar 2.2.2. sesudah

5. Pada *path* folder yang sama dengan file `game-runner-config.json`, *double-click* file `run.bat`
6. *Game* berhasil dijalankan

BAB III
Aplikasi Strategi *Greedy*

3.1. Proses *Mapping* Persoalan

Himpunan Kandidat

Seluruh *Command* yang ada pada permainan *overdrive*, mencangkup*: NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET <lane> <block>, USE_EMP, FIX*.

Himpunan Solusi

Berisi *command-command* yang dipilih dari *command-command* yang ada pada himpunan kandidat sesuai dengan fungsi seleksi.

Fungsi Solusi

Mengecek apakah posisi pemain sudah sampai di garis *finish.*

Fungsi Seleksi (*selection function*)

Memilih salah satu *command* yang valid diantara himpunan kandidat yang ada. Jika fungsi objektif adalah *greedy by speed*, maka *Command* yang dipilih adalah *command* yang menghasilkan *speed* terbesar dan *damage* terkecil. Jika kecepatan mobil sudah maksimal, maka fungsi seleksi akan memilih *command powerup* yang ada.

Fungsi Kelayakan (*feasible*)

Mengecek apabila *command* yang dihasilkan valid. *Command* yang selalu valid adalah *NOTHING, ACCELERATE, DECELERATE, FIX. Command TURN_LEFT* valid jika *lane* mobil bukan 1. *Command TURN_RIGHT* valid jika *lane* mobil bukan 4. *Command USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET <lane> <block>, USE_EMP* akan valid jika *player* memiliki *powerup* yang sesuai.

Fungsi Objektif

Jika menerapkan *greedy by speed*, maka objektif yang dimaksimalkan adalah kecepatan.

3.2. Alternatif Solusi *Greedy*

Pada tugas besar ini, beberapa alternatif solusi *Greedy* dieksplorasi untuk menentukan solusi yang terbaik. Solusi-solusi *Greedy* tersebut antara lain:

1. **Strategi *Greedy* berdasarkan *speed***

   Strategi *Greedy* berdasarkan *speed* merupakan strategi *Greedy* yang mengutamakan *speed* atau kecepatan mobil pada suatu waktu. Pada permainan *Overdrive* ini, kecepatan sebuah mobil dapat mencapai kecepatan maksimum yang bergantung pada *damage* yang telah diterima mobil. Dengan kata lain, semakin kecil *damage* mobil, semakin besar pula kecepatan maksimum mobil. Kemudian,

kecepatan maksimum ini dapat dicapai dengan terus-menerus melakukan akselerasi hingga mobil mencapai kecepatan maksimum.

Pada setiap ronde, algoritma akan menilai mana *command* yang akan menghasilkan kecepatan tercepat di ronde berikutnya.

2. **Strategi *Greedy* berdasarkan *power-up***

   Strategi *Greedy* berdasarkan *power-up* mengutamakan akuisisi *power-up* terbanyak serta penggunaannya untuk menyerang dan menghambat lawan. *Power-up* yang terdapat pada permainan *Overdrive* antara lain yaitu *BOOST*, *OIL*, *TWEET*, *LIZARD*, dan *EMP*.

   Pada setiap ronde, algoritma akan menggunakan *power-up* untuk menghambat lawan bila memiliki *power-up*, serta menilai mana *command* yang akan mendapatkan *power-up* paling banyak.

3. **Strategi *Greedy* berdasarkan jarak**

   Mirip dengan strategi *Greedy* berdasarkan kecepatan, strategi ini juga mengutamakan jalannya mobil. Strategi *Greedy* berdasarkan jarak mengutamakan command yang dapat menghasilkan jarak terjauh yang dicapai.

   Pada setiap ronde, algoritma akan menilai mana *command* yang menghasilkan jarak terjauh (x) yang dapat dicapai.

4. **Strategi *Greedy* berdasarkan *damage***

   Berbeda dengan strategi-strategi *Greedy* lainnya yang mencari solusi maksimum, strategi *Greedy* berdasarkan *damage* merupakan strategi *Greedy* yang meminimasi. *Damage* yang diterima mobil pada permainan akan berdampak pada kecepatan maksimum yang dapat dicapai mobil. Oleh karena itu, meminimasi *damage* yang diterima mobil bermanfaat untuk menjaga laju mobil tetap stabil tanpa harus melakukan *fix* berulang-kali.

   Pada setiap ronde, algoritma akan menilai mana *command* yang menghasilkan *damage* paling kecil terhadap mobil dan memilih *command* tersebut.

5. **Strategi *Greedy* berdasarkan nilai**

   Salah satu kriteria kemenangan dalam permainan *Overdrive* adalah nilai atau *score* yang akan diperhitungkan apabila kedua pemain bersamaan sampai pada garis *finish*. Karena itu, strategi memaksimalkan nilai merupakan salah satu strategi yang *valid* untuk digunakan pada permainan ini.

   Pada setiap ronde, algoritma akan menilai mana *command* yang menghasilkan nilai paling besar.

3.3. Analisis Efisiensi

Untuk seluruh strategi *Greedy* yang dirumuskan, berikut analisis efisiensinya per satu ronde atau per satu kali melakukan pemilihan *command.*

1. **Strategi *Greedy* berdasarkan *speed***
   Efisiensi waktu: O(n), dengan n kecepatan maksimum mobil pada ronde tersebut.
   Penjelasan:
   Strategi ini mempertimbangkan kecepatan maksimal yang dapat tercapai di setiap rondenya. *Command* utama yang menjadi pertimbangan untuk mencapai kecepatan maksimum ialah USE_BOOST, ACCELERATE, TURN_LEFT, dan TURN_RIGHT, sehingga algoritma akan memeriksa dan membandingkan kecepatan mobil setelah melakukan keempat *command* tersebut. Ketika memprediksi *command* ACCELERATE, TURN_LEFT, dan TURN_RIGHT, algoritma perlu memeriksa blok-blok di depan mobil untuk mencari halangan yang dapat menurunkan kecepatan. Hal ini dicapai dengan melakukan iterasi blok-blok di depan mobil sejauh kecepatan maksimum mobil sehingga menghasilkan efisiensi waktu O(n).

2. **Strategi *Greedy* berdasarkan *power-up***
   Efisiensi waktu: O(1).
   Penjelasan:
   Strategi ini hanya mempertimbangkan keberadaan *power-up* yang disimpan dalam sebuah *array*, sehingga bila pemain memiliki *power-up, power-up* tersebut akan langsung digunakan. Bila pemain tidak memiliki *power-up,* pemain bebas menjalankan *command* apapun. Dengan demikian, efisiensi waktu dari algoritma *Greedy by power-up* adalah O(1) untuk melakukan pembacaan pada elemen pertama *array.*

3. **Strategi *Greedy* berdasarkan jarak**
   Efisiensi waktu: O(n), dengan n kecepatan maksimum mobil pada ronde tersebut.
   Penjelasan:
   Sangat mirip dengan strategi *Greedy by speed,* strategi ini mempertimbangkan *command-command* utama USE_BOOST, ACCELERATE, TURN_LEFT, dan TURN_RIGHT untuk mencari command yang menghasilkan jarak mobil paling jauh. Karena itu, seperti pada algoritma *Greedy by speed,* ketika memprediksi *command* ACCELERATE, TURN_LEFT, dan TURN_RIGHT, algoritma perlu memeriksa blok-blok di depan mobil untuk mencari halangan yang dapat menurunkan kecepatan. Hal ini dicapai dengan melakukan iterasi blok-blok di depan mobil sejauh kecepatan maksimum mobil sehingga menghasilkan efisiensi waktu O(n).

4. **Strategi *Greedy* berdasarkan *damage***
   Efisiensi waktu: O(n), dengan n kecepatan maksimum mobil pada ronde tersebut.
   Penjelasan:
   Strategi ini juga menerapkan pengecekan ketiga *lane* yang dapat ditempuh mobil untuk mencari mana *lane* yang memiliki hambatan paling sedikit, sehingga efisiensi waktu pun menjadi O(n).

5. **Strategi *Greedy* berdasarkan nilai**
   Efisiensi waktu: O(n), dengan n kecepatan maksimum mobil pada ronde tersebut.
   Penjelasan:
   Nilai dapat didapatkan bila mobil mengambil *power up* ataupun memakainya, dan dapat berkurang bila mobil menabrak suatu halangan ataupun melakukan kesalahan pada pemberian perintah. Oleh sebab itu, selain langsung menggunakan *power*-up bila ada (dengan efisiensi O(1)), algoritma ini juga perlu melakukan pemeriksaan lajur-lajur yang dapat dijalani mobil bila tidak memiliki *power-up* sehingga membutuhkan efisiensi O(n).

3.4. Analisis Efektivitas

1. **Strategi *Greedy* berdasarkan *speed***
   Strategi ini efektif apabila:
   Map yang di-*generate* oleh system tidak mengandung terlalu banyak halangan, sehingga mobil tidak perlu terlalu banyak diperbaiki.

   Strategi ini tidak efektif apabila:
   Bot lawan menggunakan strategi *Greedy* by *power-up* sehingga mobil akan sering menerima *damage* dan terlalu sering memerlukan perbaikan.

2. **Strategi *Greedy* berdasarkan *power-up***
   Strategi ini efektif apabila:
   Bot lawan sudah berada jauh di depan bot ini, sehingga strategi ini berfungsi untuk menghambat bot lawan dan menaikkan kesempatan menang. Karena itu, strategi ini juga dapat dibilang efektif bila bot lawan menggunakan strategi *Greedy by speed* atau *Greedy by distance*. Dengan memanfaatkan strategi ini, lawan jadi kesulitan mencapai kecepatan maksimumnya.
   Strategi ini juga sangat efektif bila lawan tidak menggunakan *command* FIX sama sekali, yang akan menyebabkan mobil lawan rusak total sehingga tidak dapat bergerak.

   Strategi ini tidak efektif apabila:
   Map yang di-*generate* memiliki sedikit *power-up* sehingga tidak cukup untuk dipakai terus-menerus untuk menghambat lawan. Strategi ini juga kurang efektif bila

bot lawan berada jauh di belakang karena akan membuang kesempatan untuk mempercepat diri.

3. **Strategi *Greedy* berdasarkan jarak**

   Strategi ini efektif apabila:

   Map yang di-*generate* tidak memiliki banyak halangan sehingga dapat memaksimalkan jarak pada setiap rondenya tanpa terlalu peduli dengan *damage.*

   Strategi ini tidak efektif apabila:

   Map yang di-*generate* banyak memiliki halangan yang dapat memberikan *damage* sehingga berpotensi menurunkan kecepatan mobil, bahkan menghentikannya, bila tidak diperhatikan.

4. **Strategi *Greedy* berdasarkan *damage***

   Strategi ini efektif apabila:

   Map yang di-*generate* oleh sistem memiliki banyak halangan sehingga lebih menguntungkan untuk menhindarinya daripada menerobos dan memperbaiki setelahnya.

   Strategi ini tidak efektif apabila:

   Lawan menggunakan strategi yang mengutamakan jarak atau kecepatan sehingga strategi yang cenderung ‘berhati-hati’ ini menjadi kalah cepat untuk mencapai garis *finish.*

5. **Strategi *Greedy* berdasarkan nilai**

   Strategi ini efektif apabila:

   Kedua pemain sampai pada saat yang bersamaan pada garis *finish.*

   Strategi ini tidak efektif apabila:

   Kedua pemain sampai pada waktu yang berbeda pada garis *finish.*

3.5. Strategi *Greedy* Program

Pada program diterapkan *greedy by speed*, sehingga di setiap *round*-nya bot akan melakukan berbagai cara untuk mendapatkan kecepatan yang maksimal. Untuk penggunaan *powerup* seperti *USE_TWEET <lane> <block>*, *USE_OIL*, *USE_EMP*, hanya digunakan apabila mobil telah mencapai kecepatan maksimal atau keputusan untuk *ACCELERATE, TURN_LEFT*, dan *TURN_RIGHT* akan menghasilkan kecepatan dan dampak *damage* yang sama. Oleh karena itu, penggunaan *powerup* tidak akan mengganggu kecepatan mobil yang sedang berlangsung.

Strategi ini memprioritaskan kecepatan mobil terlebih dahulu (dengan menimbang *damage*) dan penggunaan *powerup* untuk prioritas berikutnya. Strategi ini menimbang bahwasanya pemenang ditentukan dari pencapai garis *finish* pertama, status *speed* tertinggi (apabila mencapai garis *finish* bersamaan), dan jumlah *score* terbanyak

(apabila status *speed* masih bernilai sama). Oleh karena itu, bot diekspektasikan dapat mencapai garis *finish* secepatnya dengan status *speed* yang tinggi dan memiliki jumlah *score* yang besar pula (hasil dari penggunaan *powerup*).

BAB IV
Implementasi dan Pengujian

4.1. Implementasi

Berikut adalah implementasi dari algoritma *greedy* yang digunakan. Algoritma yang ditampilkan tidak menjabarkan seluruh fungsi yang ada dan tidak pula sama seutuhnya seperti kode program yang ada pada source code. Hanya mencakup bagian-bagian penting program agar pembaca dapat memahami kode yang tersedia.

```
function run(GameState gamestate) -> Command
{ Implementasi algoritma greedy by speed.
    Prioritas Bot adalah sebagai berikut: speed terbesar >
    damage terkecil > menyerang musuh > paling mendekati garis finish }

DEKLARASI
myCar, opponentCar  : Car
{ Objek dengan 4 atribut private yakni integer lane untuk menandakan koordinat
  y mobil, integer block untuk koordinat x mobil, integer speed, dan integer
  damage }
maju, kiri, kanan   : speedAndDamage
{ Objek dengan 2 atribut private yakni integer speed dan integer damage }

ALGORITMA
if (myCar.damage >= 2) then
    -> FIX
if (punyaBoost) then
    if (myCar.damage = 0 and myCar.speed < 15) then
        booster <- projectedMove(myCar.lane, myCar.block, 15)
        if (booster.speed = 15) then
            -> USE_BOOST
maju <- projectedMove(myCar.lane, myCar.block, nextSpeed(myCar.speed,
                        myCar.damage))
{ maju digunakan untuk memproyeksi speed dan damage
  ketika Command ACCELERATE digunakan pada mobil }
if (maju.speed = nextSpeed(myCar.speed, myCar.damage)) then
    { jika tidak ada MUD/WALL didepan mobil }
    if (myCar.speed = maju.speed) then
        { kecepatan mobil sudah maksimal }
        -> attackConsideration(myCar, opponentCar)
    { jika ada MUD/WALL didepan mobil }
    kiri <- projectedMove(myCar.lane - 1, myCar.block, myCar.speed)
    kanan <- projectedMove(myCar.lane + 1, myCar.block, myCar.speed)
    -> bestMove(maju, kiri, kanan)
```

```
function projectedMove (lane, block, speed : integer ) -> speedAndDamage
{ megembalikan kecepatan final dan damage yang diterima mobil jika melalui
  sekian block pada sebuah lane dengan jumlah sekian block sebanyak speed }

DEKLARASI
arrayOfBlocks    : array [0..speed-1] of string
i                : integer
mobil            : speedAndDamage

ALGORITMA
mobil.speed <- speed
mobil.damage <- 0
i <- 0
repeat speed times
    if (arrayOfBlocks[i] = MUD) then
        mobil.speed <- prevSpeed(mobil.speed)
        { prevSpeed mengembalikan kecepatan mobil jika state kecepatan mobil
          berkurang 1 }
        mobil.damage <- mobil.damage + 1
    if (arrayOfBlocks[i] = OIL_SPILL) then
        mobil.speed <- prevSpeed(mobil.speed)
        mobil.damage <- mobil.damage + 1
    if (arrayOfBlocks[i] = WALL) then
        mobil.speed <- 3
        mobil.damage <- mobil.damage + 2
    if (arrayOfBlocks[i] = isOccupiedByCyberTruck) then
        mobil.speed <- 3
        mobil.damage <- mobil.damage + 2

-> mobil
```

```
function bestMove(maju, kiri, kanan : speedAndDamage, myCar, opponentCar :
                  Car) -> Command
{ mengembalikan Command terbaik yang bisa dilakukan oleh mobil (myCar)
  Command terbaik dilihat dari prioritas bot yakni speed tercepat >
  damage terkecil > agresi > block paling dekat dengan finish line }

DEKLARASI
lizard  : integer

ALGORITMA
if (kiri.speed < myCar.speed and kanan.speed < myCar.speed) then
    lizard <- isWorth_useLizard(myCar.lane, myCar.block, myCar.speed)
    { function isWorth_useLizard akan mengembalikan kecepatan mobil jika mobil
      melewati (xx - 1) block didepannya dan mendarat pada xx block didepannya
      dengan xx sebesar myCar.speed. Jika tidak ada
      MUD/WALL/OIL_SPILL/CYBERTRUCK yang dilewati, maka return speed akan
      sebesar -1 }
    if (lizard > maju.speed) then
    -> USE_LIZARD
if (maju.speed = kiri.speed and maju.speed = kanan.speed) then
    if (maju.damage = kiri.damage and maju.damage = kanan.damage) then
        -> attackConsideration(myCar, opponentCar)
if (maju.speed >= kanan.speed and maju.speed >= kiri.speed) then
    if (maju.damage <= kiri.damage and maju.damage <= kanan.damage) then
        if (maju.speed = nextSpeed(myCar.speed)) then
            -> attackConsideration(myCar, opponentCar)
        else
            -> ACCELERATE
if (myCar.position.lane = 3) then
    { jika mobil berada di lane 3 maka prioritas akan belok kiri terlebih
      dahulu sebelum kanan jika memiliki speed dan damage yang sama. Karena
      jika belok kanan, maka pada turn selanjutnya pilihan belok hanya ada
      belok kiri }
    if (kiri.speed >= maju.speed and kiri.speed >= kanan.speed) then
        if (kiri.damage <= maju.damage and kiri.damage <= kanan.damage) then
            -> TURN_LEFT
    -> TURN_RIGHT
else
    if (kanan.speed >= maju.speed and kanan.speed >= kiri.speed) then
        if (kanan.damage <= maju.damage and kanan.damage <= kiri.damage) then
            -> TURN_RIGHT
    -> TURN_LEFT
```

```
function attackConsideration(myCar, opponentCar : Car) -> Command
{ mengembalikan Command serangan terbaik yang dapat dilakukan }

DEKLARASI
-

ALGORITMA
if (opponentCar.block < myCar.block) then
    { jika musuh dibelakang kita maka prioritas adalah USE_TWEET > USE_OIL >
      USE_EMP }
    if (punyaTweet) then
        -> TweetCommand(opponentCar.lane, opponentCar.block)
    if (punyaOil) then
        -> USE_OIL
    if (punyaEMP) then
        -> USE_EMP
else
    { jika musuh didepan kita maka prioritas adalah USE_EMP > USE_TWEET >
      USE_OIL }
    if (punyaEMP) then
        -> USE_EMP
    if (punyaTweet) then
        -> TweetCommand(opponentCar.lane, opponentCar.block)
    if (punyaOil) then
        -> USE_OIL
-> ACCELERATE
```

Implementasi Fungsi

Nama Fungsi	Keterangan
`int getOpponentLane (Car opponent);`	Untuk mendapatkan posisi *lane* mobil musuh
`int getOpponentBlock (Car opponent);`	Untuk mendapatkan posisi *block* mobil musuh
`int getMyCarLane (Car myCar);`	Untuk mendapatkan posisi *lane* mobil *author*
`int getMyCarBlock (Car myCar);`	Untuk mendapatkan posisi *block* mobil *author*
`int nextSpeed(int input_speed, int damage);`	Untuk *return speed state* berikutnya dari *speed state* saat ini
`int prevSpeed(int input_speed, int damage);`	Untuk *return speed state* sebelumnya dari *speed state* saat ini

`Boolean isOpponentBehind (Car myCar, Car opponent);`	Untuk mengecek apakah posisi mobil musuh berada di belakang mobil *author*
`int isWorth_useLizard(int lane, int block, int speed, GameState gameState);`	Untuk menentukan apakah *powerup lizard worth* untuk dipakai
`Boolean hasPowerUp(PowerUps powerUpToCheck, PowerUps[] powerUps);`	Untuk mengecek keberadaan suatu *powerup*
`projectedCar projectedMove(int lane, int block, int speed, int damage, int kirikanan, GameState gameState);`	Untuk memproyeksi perkiraan kecepatan dan *damage* mobil pada *round* selanjutnya jika mobil melewati *lane* dan *block* tertentu sesuai dengan *lane* dan *block* pada parameter input
`int bestMove(projectedCar goForward, projectedCar turnLeft, projectedCar turnRight, Car myCar, GameState gameState);`	Untuk mengembalikan kode dari *Command* terbaik yang dapat dilakukan mobil. *Command* terbaik dilihat dari hasil *speed* terbesar dan *damage* terkecil yang diproyeksi menggunakan **projectedMove**. **BestMove** akan mengembalikan kode untuk menyerang musuh apabila hasil proyeksi tidak memperoleh kecepatan dan *damage* yang cukup baik
`Command attackConsideration (Car myCar, Car opponent);`	Untuk mempertimbangkan prioritas penggunaan *powerup TWEET*, *OIL*, atau *EMP*
`Bot(Random random, GameState gameState);`	Konstruktor dari class Bot
`Command run(GameState gameState);`	Untuk mengembalikan *Command* yang dihasilkan oleh kalkulasi bot setiap *round*-nya

4.2. Struktur Data

Struktur Data dan Kelas	Penjelasan Singkat
`Main.java`	Program utama yang akan dijalankan. Kelas main akan meminta data command dari kelas bot untuk menggerakan mobil.
`Bot.java`	Kelas yang akan menjalankan strategi greedy yang dibuat oleh user. Kelas bot akan mengambil data dari kelas-kelas lainnya untuk mengomputasi strategi greedy yang telah di tetapkan di fungsi run() dan akan mengembalikan command yang akan di jalankan ke main.java.

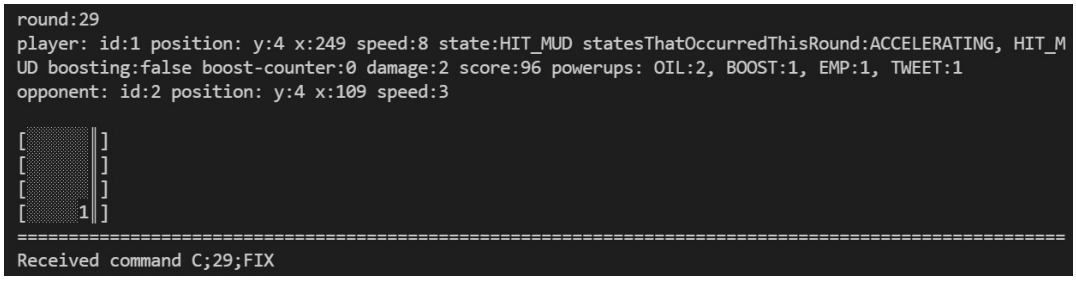
Package Command	
AccelerateCommand.java	Kelas yang akan mempercepat speed mobil.
BoostCommand.java	Kelas yang akan membuat mobil menggunakan power-up boost dari daftar powerup yang dimiliki.
ChangeLaneCommand.java	Kelas yang akan membuat mobil berpindah jalur
Command.java	Kelas yang menjalankan command yang di return oleh kelas lainnya pada package command menjadi suatu aksi nyata
DecelerateCommand.java	Kelas yang akan memperlabat laju mobil
DoNothingCommand.java	Kelas yang membuat mobil tidak melakukan apa-apa.
EmpCommand.java	Kelas yang akan membuat mobil menggunakan power-up EMP dari daftar powerup yang dimiliki.
FixCommand.java	Kelas yang akan mengurangi damage mobil sebanyak 2 satuan.
LizardCommand.java	Kelas yang akan membuat mobil menggunakan power-up Lizard dari daftar powerup yang dimiliki.
OilCommand.java	Kelas yang akan membuat mobil menggunakan power-up oil dari daftar powerup yang dimiliki.
TweetCommand.java	Kelas yang akan membuat mobil menggunakan power-up tweet dari daftar powerup yang dimiliki.
Package Entities	
Car.java	Kelas yang merepresentasikan mobil pengguna maupunmobil musuh. Kelas Car memiliki: - int id → membedakan player 1 atau 2 - Position position → menunjukkan lokasi player pada peta - int speed → menunjukkan kecepatan mobil - State state → menunjukkan status pemain pada setiap ronde - int damage → menunjukkan damage mobil setiap ronde - PowerUps[] powerups → list yang menyimpan seluruh powerups yang diambil oleh player - Boolean boosting → mengetahui apabila mobil sedang menggunakn powerup boost

	- int boostCounter → mengetahui berapa ronde powerup boost sebuah mobil aktif
GameState.java	Kelas yang menunjukkan data-data permainan secara menyeluruh. Kelas GameState memiliki: - int currentRound → menghitung jumlah ronde sekaligus menampilkan saat ini ronde ke berapa - int maxRounds → ronde maksimal - Car player → data mobil pemain - Car opponent → data mobil musuh - List<Lane[]> lanes → data tiap tiles pada peta
Lane.java	Kelas yang menunjukkan kondisi sebuah blok pada peta. Kelas Lane berisi: - Position position → kordinat sebuah blok - Terrain terrain → kondisi blok tersebut - int occupiedByPlayerId → data untuk mengetahu jika blok tersebut ada player tertentu - int isOccupiedByCyberTruck → data jika ada player yang menggunakan powerup tweet pada blok tersebut
Positions.java	Kelas yang menandakan kordinat sebuah blok atau mobil. Kelas Position memiliki: - Int lane → kordinat y dari 1-4 - Int block → kordinat x dari 0-finish
Package Enums	
Directions.java	Deklarasi Arah pergerakan mobil
PowerUps.java	Deklarasi nama-namsa powerup yang dapat digunakan player
State.java	Deklarasi nama-nama state yang dapat terjadi pada mobil player
Terrain.java	Deklarasi nama-nama medan yang dapat muncul pada block

4.3. Pengujian

1. Strategi selalu *FIX* ketika *damage* ≥ 2 , agar dapat melakukan *boosting*

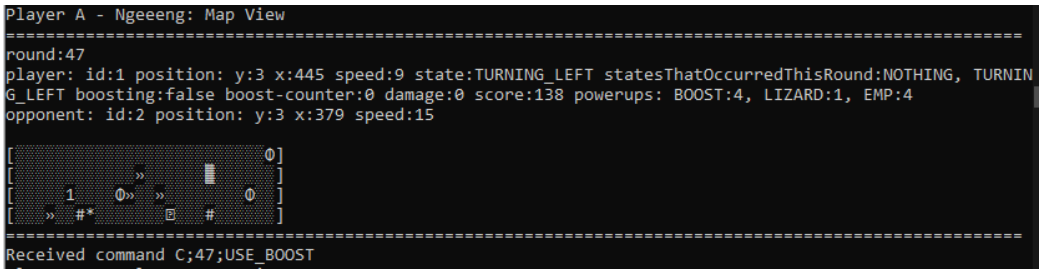
Optimal	Ketika mobil baru saja menabrak dinding/*cybertruck*/*emp*, kecepatan mobil akan menjadi 3 dan *damage* menjadi 2. Jika bot langsung melakukan *fix* mobil jika *damage* ≥ 2, pada *turn* selanjutnya kecepatan mobil akan menjadi 15. Hal tersebut lebih cepat dibandingkan dengan bot melakukan dua kali *accelerate* (5+8 < 15)
Tidak Optimal	Jika sudah dekat garis *finish* namun mobil berhenti untuk *FIX*, sehingga memungkinkan musuh untuk mendahului mobil *author*



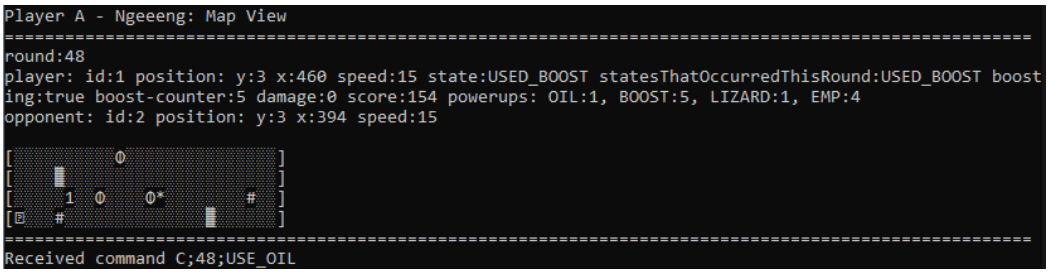
Gambar 4.3.1.

2. *Boost* hanya dipakai jika 15 *block* di depan mobil tidak ada halangan dan *damage* mobil = 0

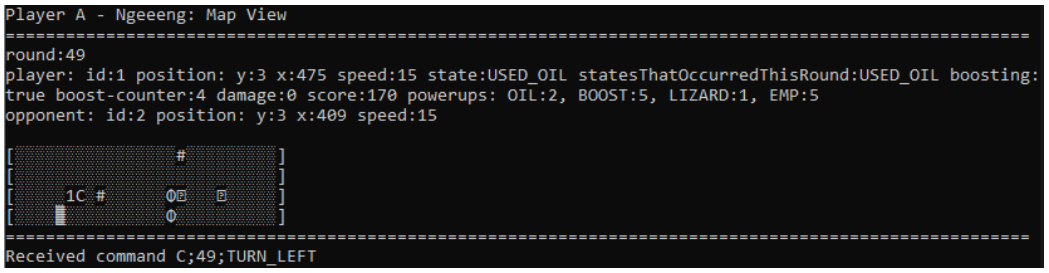
Optimal	*Boost* dapat bertahan lebih dari satu *round* sehingga minimal *block* yang dilalui dengan menggunakan satu *powerup boost* adalah 30 *block*



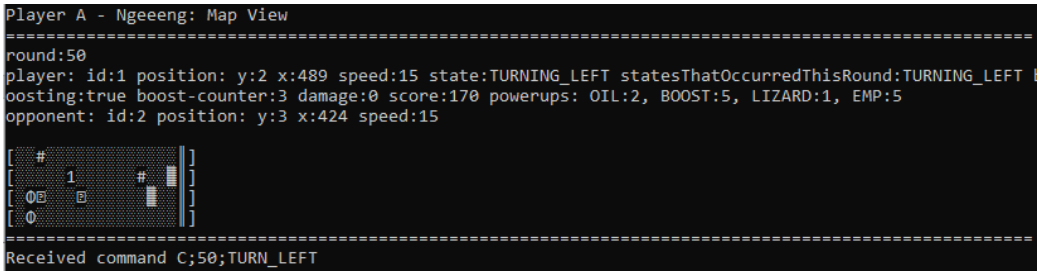
Gambar 4.3.2.



Gambar 4.3.3.

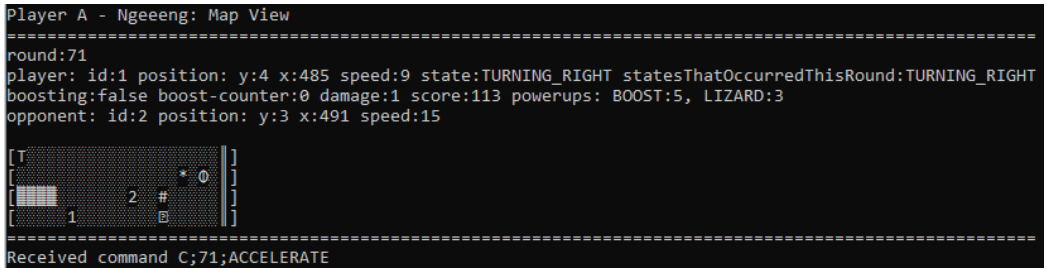


Gambar 4.3.4.



Gambar 4.3.5.

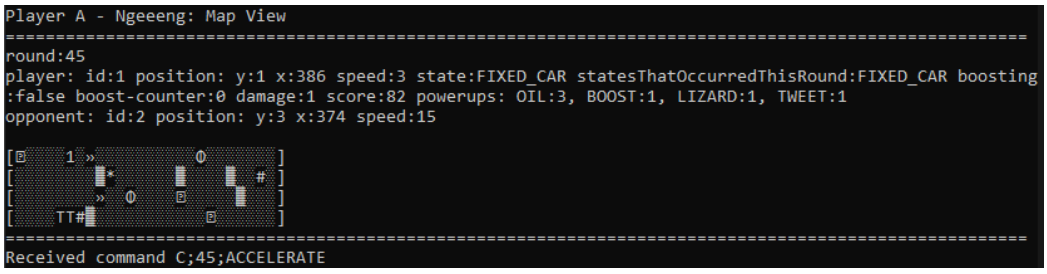
Tidak Optimal	*Boost* disimpan sampai akhir dan akhirnya tidak terpakai sama sekali. Pada contoh dibawah, bot memiliki 5 *boost* namun tidak digunakan sama sekali walau seudah mendekati garis *finish*. Jika bot memilih untuk menggunakan *boost*, maka kedua bot akan *finish* pada waktu yang bersamaan



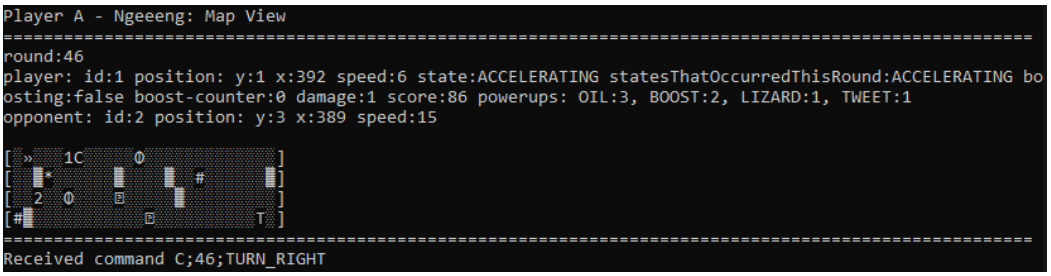
Gambar 4.3.6.

3. Memprioritaskan kecepatan dibandingkan hal lainnya

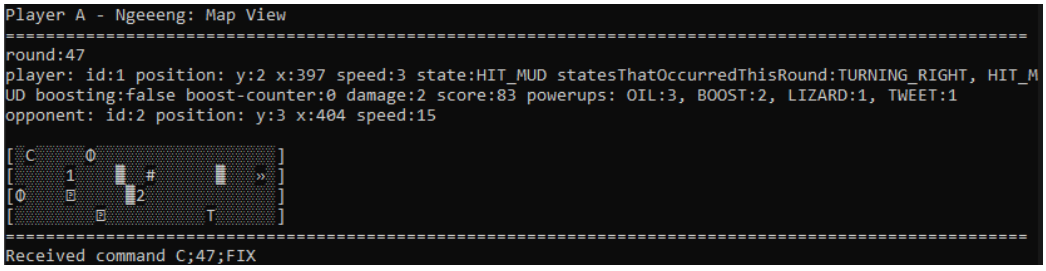
Optimal	Semakin tinggi kecepatan, semakin jauh mobil melaju sehingga semakin dekat mobil dengan garis *finish*
Tidak Optimal	Bot terlalu fokus untuk mempercepat diri sendiri sehingga lupa untuk memperhatikan musuh. Bot tidak akan mengganggu musuh yang sedang melaju dengan kecepatan *boosting*. Dengan demikian musuh dapat melaju dengan kecepatan *boosting* secara bebas selama bot mempercepat diri sampai memperoleh kecepatan maksimal. Bot akan mengganggu mobil lain jika kecepatan diri sendiri sudah maksimal. Padahal pada contoh dibawah ini, jika bot menyerang musuh pada *round* ke-45, maka perbedaan jarak antara bot dengan musuh tidak akan sejauh ini



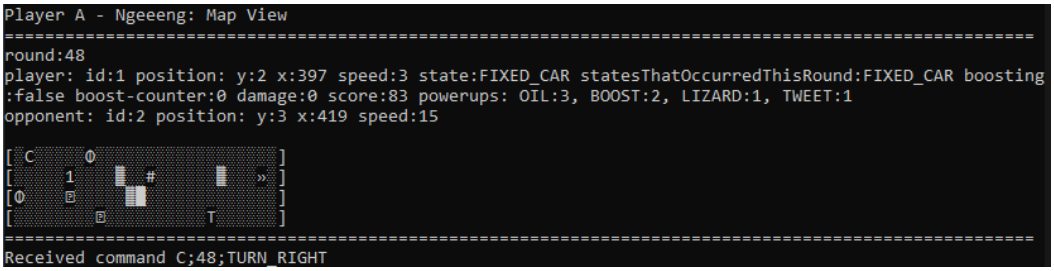
Gambar 4.3.7.



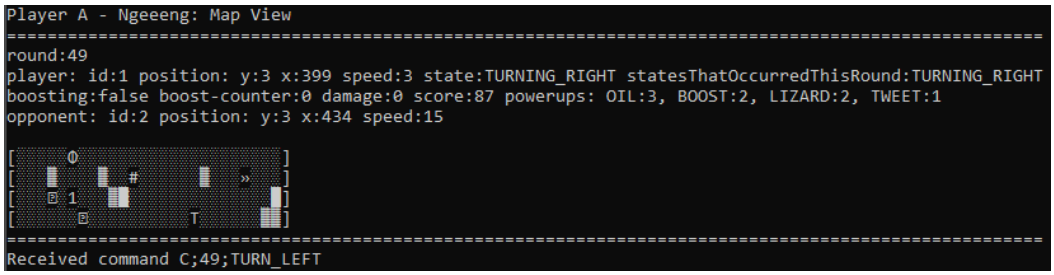
Gambar 4.3.8.



Gambar 4.3.9.



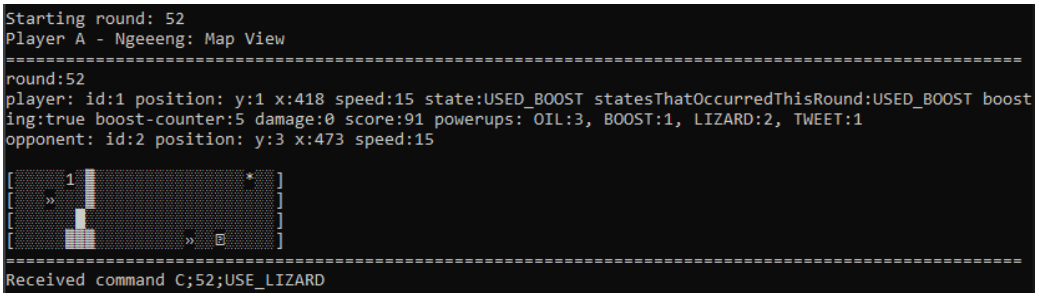
Gambar 4.3.10.



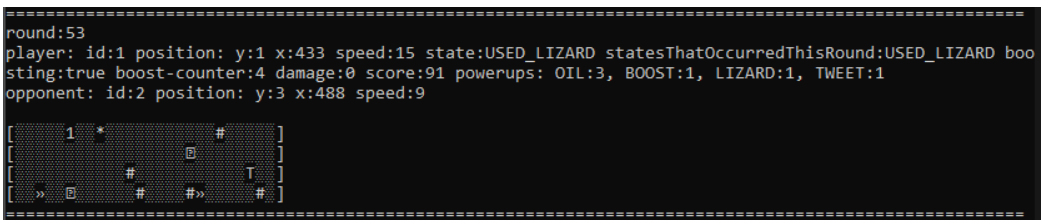
Gambar 4.3.11.

4. Menggunakan *Lizard* hanya jika ada rintangan yang dilewati dan tidak ada rintangan pada saat mendarat

Optimal	*Powerup lizard* disimpan sampai ada kondisi seperti penjelasan sehingga pada saat seperti ini *powerup lizard* tidak habis dipakai pada ronde-ronde sebelumnya

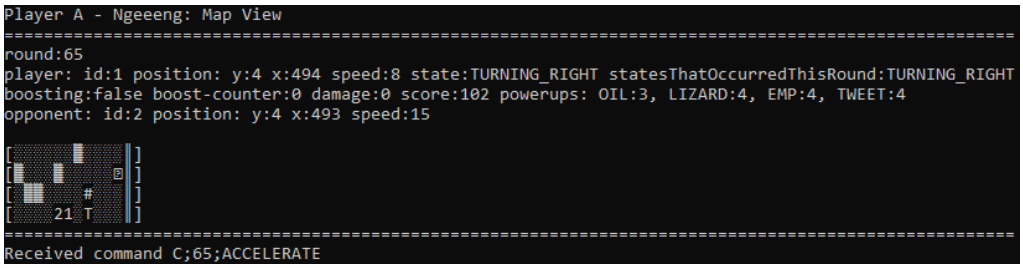


Gambar 4.3.12.



Gambar 4.3.13.

Tidak Optimal	*Powerup lizard* disimpan sampai akhir sehingga menumpuk dan tidak terpakai sama sekali sampai selesai



Gambar 4.3.14.

BAB V
Kesimpulan dan saran

5.1. Kesimpulan

Dari Tugas Besar I IF2211 Strategi Algoritma Semester II 2021/2022 berjudul *Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"*, kami mendapati bahwa untuk membangun bot dalam permainan *overdrive* ini dapat dilakukan dengan pendekatan strategi *greedy*. Dan alternatif strategi *greedy* yang ada tidak hanya satu, melainkan terdapat *greedy by speed, greedy by powerup, greedy by distance, greedy by damage,* dan *greedy by score.*

Tidak hanya itu, dari setiap alternatif strategi *greedy* juga pasti memiliki kelebihan dan kekurangannya masing-masing. Penyelesaian dengan strategi *greedy* yaitu dengan mengoptimalkan situasi pada setiap lingkup lokal, dengan harapan pada lingkup global tercapai pula kondisi yang optimal. Akan tetapi, ekspektasi tersebut tidaklah selalu berhasil dengan menggunakan strategi *greedy*. Pasti ada kalanya strategi mencapai kondisi yang optimal, dan stidak mentup kemungkinan pula strategi tidak mencapai kondisi optimal tersebut.

5.2. Saran

Saran-saran yang dapat kami berikan untuk Tugas Besar I IF2211 Strategi Algoritma Semester II 2021/2022 adalah:

1. Algoritma greedy yang digunakan pada Tugas Besar kali ini tentu masih terdapat kekurangan sehingga masih dapat dikembangkan untuk kepentingan efisiensi program
2. Kode program dapat dibuat lebih modular dan tersegmentasi dengan baik, sehingga mempermudah programmer untuk maintenance program yang tersedia
3. Menimbang agar sebaiknya program ini dapat dipublikasikan setelah dikembangkan lebih lanjut. Sehingga program ini dapat menjadi referensi publik dan memiliki manfaat yang lebih luas

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

Lampiran

Link github : *https://github.com/sivaren/Tubes1_Ngeeeng*
Link presentasi : *https://youtu.be/aVTYzwBRoP8*