LAPORAN TUGAS BESAR I

IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"

Disusun oleh:

13520023 Ahmad Alfani Handoyo

13520066 Putri Nurhaliza

13520084 Adelline Kania Setiyawan

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2021/2022

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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

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.

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

A. Algoritma *Greedy*

Algoritma *greedy* adalah sebuah teknik desain yang umum yang dapat diterapkan dalam mengoptimasi pemecahan suatu masalah. Algoritma ini mencoba mendapatkan solusi dari masalahnya dengan sebuah urutan langkah, yang tiap langkahnya menjelajahi pilihan yang ada dari solusi parsialnya, hingga pada akhirnya tercapai solusi penuhnya. Yang menjadi ciri dan kunci utama dari algoritma ini adalah pada setiap langkahnya pilihan yang dipilih harus layak, optimal secara lokal, dan tidak dapat dibatalkan. Layak dalam arti pilihan yang dipilih harus memenuhi pembatas-pembatas pada masalah yang diberikan. Optimal secara lokal dalam arti bahwa setiap pilihan adalah pilihan terbaik secara lokal yang tersedia di antara pilihan-pilihan yang mungkin pada langkah itu. Tidak dapat dibatalkan artinya bahwa ketika suatu pilihan telah dilakukan maka pilihan itu tidak dapat diubah pada langkah-langkah berikutnya di algoritmanya.

Seperti terindikasi pada namanya, *greedy* yang berarti rakus, algoritma mengambil pilihan yang "serakus" mungkin pada setiap langkahnya agar harapannya urutan langkah yang optimal ini akan menghasilkan suatu solusi optimal yang global untuk masalah yang terungkit. Nyatanya, karena algoritma ini hanya memperhatikan pengoptimalan pada setiap langkah yang spesifik dan bukan secara menyeluruh, maka algoritma *greedy* belum tentu akan selalu memberikan solusi yang optimal, namun hanya sub-optimal. Ini dikarenakan pilihan yang diambil pada suatu langkah tertentu tidak diperhatikan apakah akan berkonsekuensi ke depan, hanya optimal pada saat itu saja.

Terdapat beberapa elemen yang membantu menentukan suatu algoritma *greedy*:

1. Himpunan kandidat, C: berisi kandidat yang akan dipilih pada setiap langkah. Misalnya, simpul/sisi di dalam graf, *job*, *task*, koin, benda, karakter, dsb.
2. Himpunan solusi, S: berisi kandidat yang sudah dipilih
3. Fungsi solusi: untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi

4. Fungi seleksi/*selection function*: memilih kandidat berdasarkan suatu strategi *greedy* tertentu. Strategi ini bersifat heuristik
5. Fungsi kelayakan/*feasibility function*: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi, apakah suatu kandidat layak atau tidak
6. Fungsi objektif: memaksimumkan atau meminimumkan

Secara umum, algoritma *greedy* mempunyai skema sebagai berikut:

```
function greedy(himpunan_kandidat C) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
kandidat X
himpunan_solusi S

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 U {x}) then
    { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
        S ← S U {x} { masukkan x ke dalam himpunan solusi }
{SOLUSI(S) or C = {} }
if SOLUSI(S) then { solusi sudah lengkap }
    → S
else
    output('Tidak ada solusi')
```

B. Cara Kerja Program

Pada *game engine*, setiap *round*-nya player dapat mengeksekusikan sebuah *command* untuk mobilnya masing-masing. Setiap *command* ini unik dalam arti akan melakukan suatu aksi yang spesifik untuk mobil pada *round* itu. Terdapat 11 *command* yang ada yang dapat dilakukan oleh bot:

1. NOTHING: mobil tidak akan melakukan apa-apa untuk *round* itu. Kecepatan mobil tidak akan berubah dan mobil akan tetap pada *lane* yang sama.

2. ACCELERATE: kecepatan mobil akan bertambah ke *state* kecepatan berikutnya yang tersedia. Mobil akan tetap pada *lane* yang sama.
3. DECELERATE: kecepatan mobil akan berkurang ke *state* kecepatan mobil yang lebih rendah. Mobil akan tetap pada *lane* yang sama.
4. TURN_LEFT: mobil akan bergerak satu blok ke kiri, dan melaju sisa jatah kecepatan pada *lane* yang baru itu. Kecepatan mobil akan tetap dan ini berarti mobil akan bergerak sebesar *speed*-1 ke depan.
5. TURN_RIGHT: mobil akan bergerak satu blok ke kanan, dan melaju sisa jatah kecepatan pada *lane* yang baru itu. Kecepatan mobil akan tetap dan ini berarti mobil akan bergerak sebesar *speed*-1 ke depan.
6. USE_BOOST: mobil akan menggunakan *powerup boost* yang telah diambil sebelumnya. *Boost* akan mempercepat kecepatan ke *BOOST_SPEED* (yaitu 15) untuk 5 *round* ke depan.
7. USE_OIL: menaruh *oil* tepat di bawah mobil yang bila dilewati oleh mobil lain akan mengurangi kecepatannya.
8. USE_TWEET <lane> <block>: memunculkan *cybertruck* pada koordinat *lane* dan *block* yang diinginkan. Bila suatu mobil bertabrakan dengan *cybertruck* maka mobil akan berhenti di belakang *cybertruck* untuk *round* itu, dan kecepatan mobil akan dikurangi ke 3.
9. USE_LIZARD: menggunakan *lizard* akan membuat mobil loncat untuk *round* itu sehingga mobil tidak akan terkena pada segala *obstacle* yang terkena pada *round* tersebut kecuali pada blok dimana mobil mendarat.
10. USE_EMP: menembak EMP dari mobil yang melaju ke depan pada *lane* mobil dan juga kiri-kanannya hingga akhir *map*. Mobil musuh yang terkena EMP akan terhenti untuk *round* itu dan kecepatannya akan terkurang menjadi 3.
11. FIX: mengurangi *damage* sebanyak 2 dari mobil. Mobil akan berhenti untuk *round* tersebut.

Command-command bot ini dapat diimplementasikan sebagai algoritma *greedy* pada beberapa bahasa yang tersedia seperti C++, C#, Go, Java, JavaScript, Lisp, Python 3, Rust, dan bahasa lain yang diimplementasi oleh komunitas dibalik *game*. Namun, demi kompatibilitas dengan *game engine* yang ditulis pada Java, maka bot yang juga

diimplementasikan menggunakan Java. Bot sendiri dibangun dengan membuat folder Bot yang mempunyai struktur sebagai berikut:

```
Bot
– java
  – src/main/java/za/co/entelect/challenge
        – command
        – entities
        – enums
        – Bot.java
        – Main.java
  – target
        – java-starter-bot-jar-with-dependencies.jar
        – java-starter-bot.jar
  – bot.json
  – pom.xml
```

Agoritma *greedy* untuk bot diimplementasikan pada *file* Bot.java, tepatnya pada bagian *public Command run(GameState gameState).* Bagian *run()* nantinya akan mengembalikan sebuah *command* dari 11 yang tercantum sebelumnya. Algoritma *greedy*-nya sendiri ditentukan dari *command* apa yang dikembalikan oleh fungsi *run()* yang dapat dikendalikan dengan menggunakan percabangan sesuai kondisi-kondisi yang diinginkan oleh pembangun bot. Isi dari percabangan yaitu *command* yang ingin dijalankan bila memenuhi kondisi percabangan itu. Karena fungsi dijalankan secara prosedural, maka *command* yang dikembalikan bergantung pada percabangan mana yang pertama kali memenuhi segala kondisinya agar mengembalikan *command* untuk cabang tersebut.

Identitas bot dapat diatur pada *file* bot.json sebagai berikut:

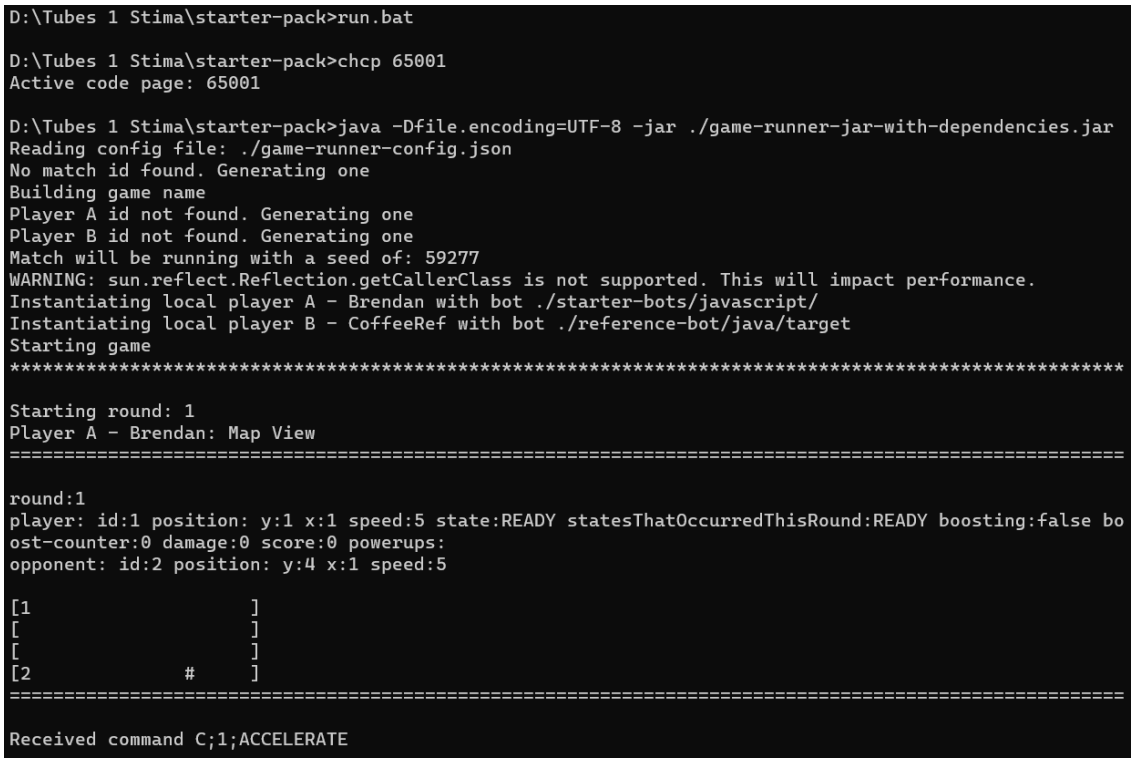
```
{
  "author": < Nama pembuat bot >,
  "email": < Email pembuat bot >,
  "nickName": < Nama bot >,
  "botLocation": < Lokasi build bot >,
  "botFileName": < Nama build .jar bot >,
  "botLanguage": < Bahasa pemrograman untuk bot >
}
```

Untuk botFileName, *default*-nya untuk bot kami adalah java-starter-bot-jar-with-dependencies.jar karena bot terbangun dari folder *starter-bots*. Ketika bot sudah selesai dibuat, maka bot perlu di-*build* terlebih dahulu menggunakan Maven beserta dengan *dependancies*-nya melalui bot.json dan pom.xml.

Setelah bot berhasil di-*build*, jangan lupa untuk mengatur game-runner-config.json sehingga "player-a" dan "player-b" merujuk pada kedua bot yang ingin ditandingkan. Sebagai contoh bila bot pertama terletak pada *directory* ./starter-bots/java dan bot kedua terletak pada *directory* ./reference-bot/java maka pada game-runner-config.json:

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

Bila game-runner-config.json sudah terkonfigurasi, maka *game engine* dapat dijalankan dengan menjalankan *run.bat* pada folder *root*.



BAB III
APLIKASI STRATEGI GREEDY

A. Mapping Persoalan *Overdrive* Menjadi Elemen-Elemen Algoritma *Greedy*

Meskipun skor menjadi salah satu faktor penentu kemenangan dari dua bot yang berlawan, namun kemungkinan skor akan mempengaruhi kemenangan kecil karena hanya akan dipertimbangkan bila kedua bot melewati garis *finish* pada waktu yang sama. Maka dari itu, dalam penentuan strategi, skor tidak menjadi prioritas utama dimasukkan dalam penentuan *command*. Dengan cacatan itu, skor tetap menjadi pertimbangan dengan tetap menjaga kerasionalan *command* yang dipilih agar tidak terjadi pengurangan skor yang drastis.

Dalam *mapping* persoalan *Overdrive*, diuraikan elemen-elemen algoritma *greedy*-nya:

1. Himpunan kandidat:
   { NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET <lane> <block>, USE_EMP, FIX }
2. Himpunan solusi:
   Command yang valid untuk setiap *round*-nya yang akan dijalankan oleh mobil
3. Fungsi solusi:
   Memeriksa apakah urutan *command* yang dipilih membawakan mobil ke garis *finish*
4. Fungsi seleksi:
   Pilihlah *command* yang paling memperlambat dan memberi *damage* terbesar ke musuh atau mempercepat dan memberi *damage* terkecil ke diri sendiri
5. Fungsi kelayakan:
   Command termasuk *command* yang valid. *Command* yang dilakukan adalah *command* yang valid dilakukan pada posisi itu pula. Bila berbelok maka tidak boleh berbelok ke tembok *lane*, misal pada *lane* paling kiri maka tidak berbelok ke kiri. Bila menggunakan *powerup*, *powerup* tersebut harus ada terlebih dahulu.
6. Fungsi objektif:
   Memenangkan *game* dengan menjadi pemain pertama yang melewati garis *finish*.

B. Eksplorasi Alternatif Solusi *Greedy*

Berdasarkan *command* serta *state* yang terdapat pada permainan Overdrive ini, terdapat beberapa eksplorasi alternatif sebagai solusi dari strategi *greedy* untuk memenangkan permainan ini, yaitu:

1. Strategi *greedy* memaksimalkan kecepatan mobil

   Strategi *greedy* ini akan selalu menaikkan kecepatan mobil hingga ke kecepatan maksimal yang ditentukan berdasarkan *damage* mobil. Strategi *greedy* ini bertujuan untuk membuat mobil sampai ke garis *finish* secepat mungkin karena memang tujuan utama untuk memenangkan permainan Overdrive ini adalah menjadi yang pertama sampai di garis *finish*.

   Strategi *greedy* ini dapat diimplementasikan dengan selalu mengutamakan me-*return command* ACCELERATE agar kecepatan mobil terus bertambah hingga kecepatan maksimalnya. Strategi *greedy* ini juga dapat dikombinasikan dengan mengutamakan me-*return command* FIX. Hal ini bertujuan agar mobil dapat diperbaiki terlebih dahulu ketika *damage* mobil sudah mencapai 5. Apabila *command* FIX tidak diperhatikan, mobil memiliki kemungkinan besar untuk tidak bisa berjalan karena *damage* mobil yang tinggi menyebabkan mobil tidak bisa bergerak sebelum mobil diperbaiki.
2. Strategi *greedy* menghindari *obstacle*

   Seperti yang sudah dijelaskan sebeluumnya, pada permainan Overdrive ini, terdapat 4 *obstacle*, yaitu *mud, wall, oil spill,* dan *cyber truck*. Penempatan *obstacle-obstacle* pada permainan juga sangat sering ditemukan. Padahal, *obstacle-obstacle* ini sangat memengaruhi penurunan performa mobil, baik *damage* maupun kecepatannya. Sehingga strategi *greedy* untuk menghindari *obstacle* yang ada sangat membantu mobil menghindari *obstacle-obstacle* yang ada di depan atau samping mobil sehingga performa mobil tidak turun dan bisa mencapai garis *finish* dengan cepat.

   Algoritma dari strategi *greedy* ini adalah dengan mengecek kondisi *lane* yang dapat dijangkau mobil di depan, samping kanan, dan samping kiri mobil. Mobil akan memilih *lane* yang tidak mengandung *obstacle*. Apabila di semua *lane* jangkauan mobil terdapat *obstacle*, mobil akan memilih *lane* mana yang memberikan *damage* terendah kepada mobil. Hal ini berdasarkan dengan *damage* yang dihasilkan oleh setiap *obstacle*. Apabila diurutkan, urutan *obstacle* dengan *damage* tertinggi ke terendah adalah *obstacle cyber*

truck, wall, oil spill, dan *mud*. Oleh karena itu, mobil akan memprioritaskan untuk menghindari *obstacle* dengan *damage* tertinggi berdasarkan urutan tersebut.

Untuk menghindari *obstacle-obstacle* tersebut, mobil dapat berpindah *lane* atau menggunakan *Powerups* LIZARD apabila mobil memiliki *Powerups* tersebut. Apabila di depan mobil tidak terdapat *obstacle* apapun, mobil akan me-*return command* ACCELERATE. Sehingga *command* yang akan diprioritaskan untuk di-*return* pada strategi ini adalah *command* TURN_LEFT dan TURN_RIGHT untuk mengindari *obstacle* dengan berpindah *lane*, *command* USE_LIZARD untuk menghindari *obstacle* dengan menggunakan LIZARD, dan *command* ACCELERATE apabila tidak terdapat *obstacle* apapun di depan mobil.

3. Strategi *greedy* memaksimalkan kecepatan mobil dan menghindari *obstacle*

   Strategi *greedy* ini merupakan kombinasi dari strategi *greedy* pada poin 1 dan 2. Pada strategi ini, mobil tidak hanya mempercepat kecepatan mobil hingga kecepatan maksimal, tetapi juga berusaha untuk menghindari *obstacle*. Hal ini bertujuan agar performa mobil tidak cepat menurun karena mengenai *obstacle*, tetapi tetap bisa berada di kecepatan maksimal mobil seharusnya.

   Algoritma dari startegi *greedy* ini dapat diimplementasikan dengan mengutamakan untuk mengecek *obstacle* yang ada di sekitar jangkauan mobil terlebih dahulu. Apabila terdapat *obstacle* pada *lane* yang akan mobil lewati, *command-command* untuk menghindari *obstacle*, seperti yang sudah dijelaskan pada poin 2, akan di-*return* terlebih dahulu agar mobil dapat menghindari *obstacle-obstacle* tersebut. Apabila tidak terdapat *obstacle* apapun pada lane yang akan dilewati, mobil akan me-*return command* ACCELERATE agar kecepatan mobil bisa selalu maksimal sehingga mobil akan cepat mencapai garis *finisih* dan memenangkan permainan. Namun, perlu dilakukan validasi terlebih dahulu terkait *damage* mobil. Hal ini bertujuan untuk memperbaiki mobil dengan *command* FIX apabila *damage* mobil sudah terlalu tinggi sehingga mobil bisa terus berjalan dan mempercepat kecepatannya.

4. Strategi *greedy* memanfaatkan *Powerups*

   Berbeda dengan strategi-strategi sebelumnya, pada strategi *greedy* ini, fokus utama mobil adalah mengumpulkan sebanyak-banyaknya *Powerups* dan memanfaatkan *Powerups* yang dipunya, seperti dengan melakukan serangan kepada mobil lawan dengan

Powerups yang dimiliki mobil. Hal ini bertujuan untuk memperlambat dan memberikan *damage* kepada mobil lawan agar performa mobil lawan akan terus menurun dan membuat lawan sibuk untuk melakukan perbaikan kepada mobilnya sehingga mobil lawan akan selalu terhambat untuk mencapai garis *finish* secepat-cepatnya.

Algoritma untuk mengimplementasikan strategi ini adalah dengan terlebih dahulu mobil akan selalu memprioritaskan untuk mengambil *Powerups* yang tersedia. Setelah itu, untuk menggunakan *Powerups* tersebut, mobil akan mengecek apa saja *Powerups* yang dimiliki oleh mobil lalu menggunakan *command-command* yang mengeluarkan *Powerups* yang dapat mengganggu performa lawan, seperti USE_OIL, USE_EMP, USE_TWEET. Penggunaan *Powerups* dapat diprioritaskan berdasarkan *damage* yang diberikan kepada mobil lawan dan kemudahan *Powerups* untuk menyerang mobil lawan.

Berdasarkan hal tersebut, penggunaan *Powerups tweet* akan lebih diprioritaskan untuk digunakan karena *damage* yang diberikan ke mobil lawan cukup besar dan cukup akurat dalam menyerang mobil lawan. Prioritas selanjutnya bisa menggunakan *Powerups EMP* yang menembakkan bom ke lawan dan berpengaruh cukup besar ke *damage* mobil lawan. Setelah itu, *Powerups oil* bisa digunakan untuk mencipatakn jalanan licin yang juga berpengaruh pada *damage* mobil lawan apabila mobil lawan mengenai *oil* yang kita tumpahkan. Namun, perlu diperhatikan bahwa diperlukan validasi-validasi lainnya sebelum menggunakan *Powerups* tersebut, seperti validasi mengecek posisi lawan.

Selain *Powerups* yang bertujuan untuk menyerang mobil lawan, terdapat juga *Powerups* yang berguna untuk meningkatkan performa mobil, yaitu *Powerups boost* dan *Powerups* yang berguna untuk menghindari *obstacle* yang ada, yaitu *Powerups lizard*. *Powerups* ini juga sangat berguna agar performa mobil kita bisa meningkat sehingga akan lebih cepat mencapai garis *finish*. Sama halnya dengan *Powerups* yang bersifat menyerang lawan, diperlukan validasi-validasi lainnya sebelum menggunakan *Powerups* tersebut, seperti validasi apakah ada *obstacle* di jangkauan mobil atau validasi apakah kondisi *damage* mobil sudah memenuhi aturan untuk menggunakan *command* USE_BOOST. Validasi-validasi ini sangat penting untuk dilakukan agar nantinya mobil tidak me-*return command* NO_COMMAND.

C. Analisis Efisiensi

Berdasarkan alternatif-alternatif strategi *greedy* yang sudah dijelaskan pada bagian sebelumnya, diperlukan analisis efisiensi terhadap strategi-strategi *greedy* tersebut. Analisis efisiensi yang akan dijabarkan berfokus pada apakah strategi *greedy* yang dijadikan alternatif sudah cukup efisien dalam penggunaan *command*-*command* yang ada. Berikut ini adalah analisis efisiensi dari alternatif strategi *greedy* tersebut:

1. Strategi *greedy* memaksimalkan kecepatan mobil

   Pada strategi *greedy* ini, *command* ACCELERATE akan menjadi *command* utama yang akan di-*return*. Dengan pemanggilan *command* ACCELERATE, mobil bisa dengan cepat sampai pada garis *finish* terlebih dahulu. Walaupun demikian, penggunaan *command* ACCELERATE tanpa melakukan pengecekan apakah *lane* yang akan ditempuh tidak mengandung *obstcale* apapun dapat membuat performa mobil turun dengan drastis. Hal ini menyebabkan *damage* mobil terus meningkat sehingga diperlukan melakukan *command* FIX yang cukup banyak. Hal ini menyebabkan pemanggilan *command* tidak efisien karena mobil akan sangat sering memanggil *command* FIX yang diakibatkan oleh pemanggilan *command* ACCELERATE tanpa melakukan pengecekan terlebih dahulu.
2. Strategi *greedy* menghindari *obstacle*

   Pada strategi *greedy* ini, penggunaan *command* yang di-*return* lebih efisien dibandingkan penggunaan strategi *greedy* yang memaksimalkan kecepatan mobil. Hal ini disebabkan mobil akan selalu berusaha menghindari *obstacle* yang ada sehingga *damage* mobil tidak cepat naik dan pemanggilan *command* FIX yang sebenarnya menurut kelompok kami kurang efisien tidak terlalu banyak. Dengan menggunakan strategi *greedy*, mayoritas *command*-*command* yang di-*return* selama permainan berlangsung akan selalu membuat mobil bergerak maju kedepan. Hal ini jauh lebih efisien dibandingkan strategi *greedy* sebelumnya yang mayoritas *command*-nya hanya membuat mobil diam statis di tempat.

   Akan tetapi, dalam sisi efisiensi, walaupun *command*-*command* yang di-*return* hampir semuanya membuat mobil maju, namun jarak tempuh mobil maju bisa hanya sedikit-sedikit. Hal ini menyebabkan strategi ini juga kurang efisien karena mobil tidak bisa langsung maju cukup jauh. Hal ini disebabkan oleh mobil yang tidak menjadikan *command* ACCELERATE sebagai salah satu prioritasnya. Sehingga, walaupun mobil akan

maju terus, mobil akan cukup lama untuk mencapai garis *finish*. Pada akhirnya, kemungkinan tingkat efisiensi strategi *greedy* memaksimalkan kecepatan mobil dan menghindari *obstacle* tidak akan jauh berbeda.

3. Strategi *greedy* memaksimalkan kecepatan mobil dan menghindari *obstacle*

   Walaupun strategi *greedy* memaksimalkan kecepatan mobil dan strategi *greedy* menghindari *obstacle* kurang efisien, namun apabila kedua strategi *greedy* ini dikombinasikan akan menghasilkan strategi *greedy* yang akan jauh lebih efisien dibandingkan kedua strategi *greedy* ini diimplementasikan sendiri-sendiri. Dengan menggabung kedua strategi *greedy* menjadi satu, *command* yang dianggap kurang efisien karena menyebabkan mobil diam di tempat jadi lebih jarang digunakan, seperti *command* FIX yang lebih jarang digunakan karena mobil akan berusaha menghindari *obstacle* sehingga mobil tidak akan cepat rusak. Namun, walaupun strategi *greedy* ini cukup efisien, masih terdapat banyak *command* yang tidak digunakan pada strategi *greedy* ini. Hal ini menyebabkan strategi *greedy* ini menjadi kurang efisien karena tidak memanfaatkan *command-command* yang mungkin bisa membuat performa atau permainan mobil menjadi jauh lebih efisien.

4. Strategi *greedy* memanfaatkan *Powerups*

   Strategi *greedy* dengan memanfaatkan *Powerups* sangat efisien digunakan dalam menyerang lawan. Namun, fokus utama dari permainan dengan menggunakan strategi *greedy* ini menjadi lebih ke arah menyerang lawan. Padahal, untuk memenangkan permainan ini bukanlah untuk menyerang mobil lawan sebanyak-banyaknya, melainkan untuk mencapai garis *finish* terlebih dahulu. Oleh karena itu, strategi *greedy* ini sangat kurang efisien dalam memaksimalkan performa mobil untuk bisa mencapai garis *finish* terlebih dahulu. Walaupun terdapat *Powerups* yang dapat digunakan untuk meningkatkan performa mobil, seperti *Powerups lizard* dan *boost*, namun penggunaan strategi ini masih kurang efisien karena masih terdapat banyak *command-command* yang mungkin bisa membuat performa atau permainan mobil menjadi jauh lebih efisien tetapi tidak digunakan atau dimanfaatkan.

D. Analisis Efektivitas

Selain analisis efisien yang telah dilakukan pada subbab sebelumnya, diperluka juga analisis efektivitas terhadap alternatif-alternatif strategi *greedy* yang ada agar bisa ditentukan

strategi *greedy* terbaik untuk diimplementasikan pada program bot yang akan dibuat. Analisis efektivitas yang dilakukan akan melihat apakah strategi *greedy* dapat membuat bot mobil bisa sampai di garis *finish* secepat mungkin agar dapat memenangkan permainan Overdrive. Berikut ini adalah analisis efisiensi dari alternatif strategi *greedy* tersebut:

1. Strategi *greedy* memaksimalkan kecepatan mobil

   Pada strategi *greedy* ini, kecepatan mobil akan selalu ditingkatkan hingga mencapai kecepatan maksimal agar bisa cepat mencapai garis *finish*. Hal ini bersesuaian dengan tujuan untuk memenangkan permainan, yaitu mencapai garis *finish* lebih dahulu dibandingkan lawan. Apabila kecepatan mobil bisa selalu dinaikkan, strategi *greedy* ini akan cukup efektif untuk permainan ini. Namun, dalam permainan, terdapat banyak sekali *obstacle* yang bukan hanya dapat menurunkan kecepatan mobil, melainkan juga menyebabkan *damage* mobil hingga mencapai 5 sehingga diperlukan perbaikan agar kecepatan mobil bisa naik lagi. Dengan kecepatan yang selalu diusahakan maksimal, mobil juga akan sering tertabrak *obstacle* sehingga kecepatan mobil akan sulit untuk maksimal. Hal ini menyebabkan strategi *greedy* yang memaksimalkan kecepatan mobil tidak efektif untuk membuat mobil cepat mencapai garis *finish.*
2. Strategi *greedy* menghindari *obstacle*

   Pada strategi *greedy* ini, mobil akan selalu berusaha menghindari *obstacle* yang ada di sekitar jangkauan mobil dengan tujuan agar kecepatan mobil tidak cepat turun dan *damage* mobil tidak terlalu tinggi sehingga bisa mencapai garis *finish* lebih dulu. Strategi *greedy* ini mungkin akan cukup efektif apabila mobil bisa sepenuhnya tidak mengenai *obstacle* apapun. Namun, pada permainan *overdrive*, *obstacle* yang ada cukup banyak dan sangat memungkinkan dimana mobil sudah tidak bisa lagi menghindari *obstacle* dan pada akhirnya menyebabkan kecepatan mobil berkurang. Hal ini menyebabkan kemungkinan bahwa kecepatan mobil akan mencapai 0 dan mobil tidak akan bisa mencapai garis *finsih* dengan efektif. Walaupun mobil pasti akan diperbaiki, namun kecepatan mobil tidak ditingkatkan terus sehingga kecepatan mobil akan terlalu rendah dan lambat untuk bisa mencapai garis *finish* lebih dulu dibandingkan lawan. Pada akhirnya, strategi *greedy* menghindari *obstacle* masih kurang efektif untuk memenangkan permainan Overdrive dengan mencapai garis *finish*.
3. Strategi *greedy* memaksimalkan kecepatan mobil dan menghindari *obstacle*

Sama dengan efisiensi sebelumnya, efektivitas strategi *greedy* yang merupakan kombinasi memaksimalkan kecepatan mobil dan mengindari *obstacle* akan lebih efektif dibandingkan efektivitas strategi *greedy* tersebut yang tidak dikombinasikan. Hal ini disebabkan oleh mobil akan selalu berusaha untuk mengindari *obstacle* dulu sebelum akhirnya memaksimalkan kecepatan mobil sehingga performa mobil tidak akan cepat menurun, baik kecepatannya yang lambat maupun *damage* yang terlalu tinggi, sehingga mobil dapat lebih cepat mencapai garis *finish*. Mobil akan selalu diusahakan bergerak dengan kecepatan maksimal sehingga akan lebih cepat untuk mencapai garis *finish.* Oleh karena itu, strategi ini sudah cukup efektif untuk membuat mobil memenangkan permainan dengan mencapai garis *finish* lebih dulu daripada lawan.

4. Strategi *greedy* memanfaatkan *Powerups*

   Berbeda dengan strategi *greedy* yang lain, strategi *greedy* memenfaatkan *Powerups* tidak terlalu berfokus untuk mencapai garis *finish* terlebih dahulu, melainkan membuat performa mobil lawan menurun sehingga mobil lawan bisa lebih lama mencapai garis *finish*. Strategi ini cukup efektif untuk dilakukan apabila lawan tidak memiliki mekanisme untung menghindar dari serangan. Namun, apabila lawan memiliki mekanisme tersebut, seperti bisa menghindari *oil, cyber truck,* atau bahkan, tembakan EMP, strategi ini akan menjadi sangat tidak efektif. Hal ini disebabkan oleh serangan yang kita lakukan tidak berpengaruh apa-apa pada mobil lawan, sedangkan mobil kita akan lama mencapai garis *finish* karena lebih berfokus untuk menyerang lawan. Oleh karena itu, apabila ingin menggunakan strategi ini, akan lebih efektif untuk dikombinasikan dengan strategi *greedy* yang lain agar tetap bisa menyerang mobil lawan dan juga berusaha untuk mencapai garis *finish* lebih dulu dibandingkan mobil lawan.

E. Strategi *Greedy* yang dipilih

Strategi *Greedy* yang dipilih untuk memenangkan permainan Overdrive adalah menghindari *obstacle-obstacle* yang ada namun tetap memaksimalkan kecepatan mobil dengan *command Accelerate* serta mengambil dan memanfaatkan *Powerups* yang tersedia sebanyak-banyaknya dengan tujuan untuk memperlambat mobil lawan. Strategi greedy yang dipilih ini merupakan kombinasi dari alternatif solusi *greedy* pada poin tiga dan empat. Berikut ini adalah implementasi yang menunjukkan prioritas-prioritas pengambilan *command* untuk mewujudkan strategi *Greedy* yang diinginkan:

1. Keputusan pertama yang akan dilakukan adalah memperbaiki mobil jika *damage* lebih dari atau sama dengan 2. Walaupun itu artinya mobil tidak akan ada pergerakan maju pada *round* tersebut, hal ini penting untuk diutamakan sebab jika *damage* mencapai 5 maka akan lebih berbahaya bagi mobil karena kecepatannya akan menjadi 0. Sekali penggunaan *FIX* akan mengurangi *damage* hingga 2 poin. Sehingga, saat dilakukan Accelerate setelahnya, maka kecepatan mobil dapat berangsur naik dan mencegah ketertinggalan. Mobil terlebih dahulu harus memvalidasi *damage* yang ia punya, jika valid (≥2) maka mobil akan melakukan *command* FIX.
2. Jika mobil tidak memenuhi kondisi di atas, maka selanjutnya perlu diperiksa kecepatan mobil pada saat itu. *Accelerate* akan dilakukan jika kecepatan mobil kurang dari atau sama dengan 3. Hal ini ditujukan agar mobil tidak berjalan terlampau lamban atau pun berhenti total.
3. Mobil dapat mengetahui kondisi *block lane* di depannya. Apabila terdapat *obstacle* (*mud, oil spill, wall, cyber truck*) pada *lane* tempat mobil melaju, maka keputusan terbaik saat itu adalah menghindarinya. Sebab *obstacle* tersebut dapat memberikan *damage* yang tentunya akan mengurangi kecepatan mobil. Menghindari *obstacle* dapat dilakukan dengan:
   a. Menggunakan *Powerups LIZARD* yang dapat membuat mobil melewati *block* tersebut tanpa masalah, bahkan beberapa *block* sekaligus. Perlu dilakukan validasi terlebih dahulu apakah mobil memiliki *LIZARD* atau tidak. Penggunaan *LIZARD* dilakukan dengan *command* USE_LIZARD.
   b. Cara lain adalah dengan pindah ke *lane* kanan atau kiri yang memungkinkan. Namun, perlu divalidasi terlebih dahulu apakah *lane* tersebut aman dari *obstacle* atau tidak pada jangkauan gerak mobil saat itu.
4. Selanjutnya, akan diperiksa apakah *block-block* yang dapat dilalui mobil pada saat itu memiliki *Powerups* atau tidak. Jika ya, akan dicek
   - Jika terdapat Powerups LIZARD dan yang dimiliki mobil kurang dari 5
   - Jika terdapat Powerups lain dan yang dimiliki mobil kurang dari 3

   maka mobil akan diarahkan untuk melalui *block* tersebut untuk mengambil *Powerups* yang tersedia yang kemudian dapat digunakan untuk menambah kekuatan/kemampuan, maupun untuk menyerang lawan. Penggunaan *Powerups* lebih lanjut akan dijelaskan pada poin 6.

5. Selanjutnya perlu diperiksa kembali kecepatan mobil saat ini. Jika kurang dari atau sama dengan 6, maka perlu dilakukan *Accelerate*. Peningkatan kecepatan mobil ini dilakukan karena akan memberikan dampak yang lebih signifikan dibanding pilihan pada poin-poin di bawah. Sebab, hal pertama yang menentukan pemenang dalam *game* ini adalah siapa yang pertama mencapai garis *finish*, sehingga kecepatan mobil perlu diperhatikan jika tidak memenuhi kondisi sebelumnya di atas.
6. Setelah pada poin 4 divalidasi bahwa tidak ada *Powerups* yang bisa diambil, pada poin ke-6 ini mobil akan memprioritaskan untuk menggunakan *Powerups* yang dimiliki. Berikut ini adalah urutan prioritas *Powerups* yang akan digunakan:
   a. *Powerups TWEET*

      Penggunaan *Powerups TWEET* memiliki *damage* yang sama dengan *Powerups EMP*, yaitu membuat kecepatan mobil lawan apabila tertabrak oleh *cyber truck* yang kita taruh turun drastis menjadi 3. Bedanya, *Powerups TWEET* dapat diluncurkan ke koordinat mana saja, bahkan satu blok persis di depan musuh. Oleh karena itu, kami menaruh *Powerups TWEET* pada prioritas utama. *Powerups TWEET* ini akan langsung digunakan apabila mobil memiliki *Powerups TWEET*. Pada penggunaan *Powerups TWEET*, *command* USE_TWEET akan meletakkan *cyber truck* pada posisi *lane* lawan sekarang dan *block* lawan sekarang ditambah dengan kecepatan mobil lawan. Hal ini bertujuan untuk memperbesar kemungkinan mobil lawan tertabrak oleh *cyber truck* yang kita letakkan.
   b. *Powerups EMP*

      Penggunaan *Powerups EMP* kami jadikan prioritas kedua adalah karena *damage* yang dihasilkan ketika menyerang lawan cukup tinggi serta cakupan tembakannya juga cukup luas (tidak hanya pada satu *lane* saja, tetapi juga *lane* samping kanan dan kirinya). Kecepatan mobil lawan akan langsung turun drastis hingga kecepatan 3. Namun, sebelum menggunakan *Powerups EMP* ini, program akan melakukan beberapa validasi terlebih dahulu, yaitu:
      - Mobil memiliki *Powerups EMP*
      - Kecepatan mobil lawan harus lebih besar daripada 5. Hal ini perlu divalidasi agar tembakan yang kita berikan berpengaruh besar kepada lawan.

- Posisi mobil berada di belakang mobil lawan. Juga, mobil lawan berada di radius serangan *Powerups EMP* ini, yaitu berada di satu *lane* atau *lane* sebelah kanan atau kiri mobil. Hal ini bertujuan agar serangan *Powerups EMP* yang diberikan bisa tepat mengenai mobil lawan.

Apabila kedua kondisi itu terpenuhi, maka mobil akan melakukan *command USE_EMP* sehingga *Powerups EMP* dapat menyerang mobil lawan dan kecepatan mobil lawan berkurang hingga 3. Namun, apabila kedua kondisi tersebut tidak terpenuhi, mobil tidak akan menggunakan *Powerups EMP*.

Walaupun *Powerups EMP* ini tidak akan digunakan pada suatu *round* karena syarat pada poin ke-2 tidak terpenuhi, mobil akan berusaha untuk berada di satu *lane* dengan lawan. Hal ini akan dijelaskan lebih lanjut pada poin ke 7.

c. *Powerups BOOST*

Priortitas *Powerups* selanjutnya yang akan digunakan adalah *Powerups BOOST*. Hal ini bertujuan untuk mempercepat kecepatan mobil hingga kecepatan maksimum sehingga mobil akan cepat sampai ke garis *finish*. Walaupun semakin cepat mobil akan semakin cepat juga mobil mencapai garis *finish*, namun hal ini berarti semakin besar juga kemungkinan mobil untuk terkena *obstacle* yang ada. Hal ini karena jarak maju mobil yang cukup besar sehingga akan sulit untuk menghindari *obstacle*. Oleh karena itu, kami menempatkan *Powerups BOOST* pada prioritas ke-3. Untuk menggunakan *Powerups BOOST* terdapat beberapa validasi yang harus terpenuhi, yaitu:

- Mobil memiliki *Powerups BOOST*
- *Damage* mobil harus sama dengan 0 karena *Powerups BOOST* hanya dapat bekerja apabila *damage* mobil sama dengan 0. Apabila validasi ini tidak dilakukan, mobil akan sia-sia melakukan suatu *command* yang tidak memberikan dampak apapun pada mobil dan hanya membuang *Powerups* yang dimiliki.
- Tidak terdapat *obstacle* apapun pada 15 *block* di satu *lane*. Namun apabila terdapat *obstacle,* program akan mengecek apakah mobil memiliki *Powerups LIZARD.* Validasi ini penting agar setelah mobil berhasil menggunakan *Powerups BOOST*, *Powerups BOOST* ini tidak langsung hilang karena mobil langsung menabrak *obstacle* yang ada di jangkauannya.

Apabila ketiga kondisi ini terpenuhi, maka mobil akan melakukan *command* USE_BOOST yang membuat kecepatan mobil akan langsung naik menjadi 15.

d. *Powerups OIL*

Penggunaan *Powerups OIL* dijadikan prioritas terakhir karena *damage* yang dihasilkan kepada mobil lawan tidak terlalu besar dan lawan dengan mudah dapat menghindarinya, terlebih apabila lawan memiliki *Powerups LIZARD*. Oleh karena itu, penggunaan *command* USE_OIL baru digunakan apabila *Powerups* lainnya tidak ada yang bisa digunakan, mobil memiliki *Powerups OIL*, serta posisi mobil lawan ada dibelakang posisi mobil kita sekarang.

7. Jika posisi mobil berada di belakang mobil lawan, maka akan diusahakan agar lawan masuk ke jangkauan serangan EMP. Untuk itu, mobil perlu berpindah ke *lane* 2 atau 3 jika *empty*, bergantung posisi mobil dan posisi lawan pada saat itu. Sebaliknya, jika posisi mobil berada di depan mobil lawan, maka mobil perlu sebisa mungkin menghindari jangakauan serangan EMP lawan dengan belok ke *lane* tepi terdekat. Karena, apa bila mobil berada di salah satu *lane* tengah, maka mobil lebih gampang memasuki jangkauan serangan *Powerups EMP* lawan.
8. Secara *default*, opsi terakhir yang dapat dilakukan oleh mobil adalah untuk melakukan *Accelerate* untuk tetap memastikan bahwa mobil melaju pada kecepatan tercepat.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma Greedy

1. Memperbaiki mobil

```
if (myCar.damage ≥ 2) then
{ Mengecek apakah damage mobil lebih dari sama dengan 2 }
        → FIX
```

2. Mempercepat mobil

```
if (myCar.speed ≤ 3) then
{ Mengecek apakah kecepatan mobil kurang dari sama dengan 3 }
        → ACCELERATE
```

3. Menghindari Obstacle

```
if !(blocksAreEmpty(lane, block, gameState, speed)) then
{ Mengecek apakah blocks kedepan yang akan dicapai memiliki obstacle}
        if (hasPowerUp(LIZARD, myCar.powerups)) then
        { Mengecek apakah memiliki LIZARD untuk digunakan}
                → LIZARD
        else
        { Tidak memiliki LIZARD untuk digunakan, maka belok jika mungkin. Speed
        dianggap berkurang 1 karena untuk pindah lane membutuhkan 1 speed }
                if (myCar.position.lane = 1) then
                        if (blocksAreEmpty(2, block, gameState, speed-1)) then
                        { Mengecek apakah lane 2 empty, jika ya belok kanan }
                                → TURN_RIGHT
                else if (myCar.position.lane = 4) {
                        if (blocksAreEmpty(3, block, gameState, speed-1)) then
                        { Mengecek apakah lane 3 empty, jika ya belok kiri }
                                → TURN_LEFT
                else if (myCar.position.lane = 2) then
                        if (blocksAreEmpty(1, block, gameState, speed-1)) then
                        { Mengecek apakah lane 1 empty, jika ya belok kiri }
                                → TURN_LEFT
                        else if (blocksAreEmpty(3, block, gameState, speed-1))
                                then
                        { Mengecek apakah lane 3 empty, jika ya belok kanan }
```

```
                    → TURN_RIGHT
            else if (myCar.position.lane = 3) then
                if (blocksAreEmpty(4, block, gameState, speed-1)) then
                { Mengecek apakah lane 4 empty, jika ya belok kanan }
                    → TURN_RIGHT
                else if (blocksAreEmpty(2, block, gameState, speed-1))
                    then
                { Mengecek apakah lane 2 empty, jika ya belok kiri }
                    → TURN_LEFT
```

4. Pengambilan *Powerups*

   Untuk generalisasi perintah, dibuat fungsi untuk mengembalikan *command* untuk mengambil *powerups:*

```
function TakePowerUp(Terrain terrain, PowerUps powerup, int lane, int block,
GameState gameState, int speed) → Command
{ Menentukan command yang dilakukan berdasarkan posisi powerup. Dikembalikan
Command dummy bernama NOTHING agar tidak mengembalikan Command apa pun bila
tidak memenuhi salah satu dari kondisi fungsi. }
Deklarasi
List<Object> blocks, leftBlocks, rightBlocks

Algoritma
blocks ← getBlocksInFront(lane, block, gameState, speed)
{ Mendapatkan block pada lane mobil }
if (blocksAreEmpty(lane, block, gameState, speed) and
blocks.contains(terrain)) then
{ Mengecek apakah block pada lane mobil tidak ada obstacle dan mengandung
powerup yang diinginkan }
    → ACCELERATE
else if (lane > 1) then
{ Lane tidak paling kiri }
    leftBlocks ← getBlocksInFront(lane - 1, block, gameState, speed - 1)
    { Mendapatkan block pada lane kiri dari mobil }
    if (blocksAreEmpty(lane - 1, block, gameState, speed - 1) and
    leftBlocks.contains(terrain)) then
    { Mengecek apakah block pada lane kiri dari mobil tidak ada obstacle dan
    mengandung powerup yang diinginkan }
```

```
                → TURN_LEFT
        else
                → NOTHING
else if (lane < 4) then
{ Lane tidak paling kanan }
        rightBlocks ← getBlocksInFront(lane + 1, block, gameState, speed - 1)
        { Mendapatkan block pada lane kanan dari mobil }
        if (blocksAreEmpty(lane + 1, block, gameState, speed - 1) and
        rightBlocks.contains(terrain)) then
        { Mengecek apakah block pada lane kanan dari mobil tidak ada obstacle
        dan mengandung powerup yang diinginkan }
                → TURN_RIGHT
        else
                → NOTHING
else
→ NOTHING
```

Fungsi tersebut diimplementasikan pada algoritma sebagai berikut:

```
if (countPowerUp(PowerUps.LIZARD, myCar.powerups) < 5) then
{ Hanya cari powerup lizard bila jumlahnya kurang dari 5}
    Command powaction ← TakePowerUp(Terrain.LIZARD, PowerUps.LIZARD,
    myCar.position.lane, myCar.position.block, gameState, myCar.speed)
    if (powaction ≠ NOTHING) then
        → powaction
else if (countPowerUp(PowerUps.TWEET, myCar.powerups) < 3) then
{ Hanya cari powerup tweet bila jumlahnya kurang dari 3}
    Command powaction ← TakePowerUp(Terrain.TWEET, PowerUps.TWEET,
    myCar.position.lane, myCar.position.block, gameState, myCar.speed)
    if (powaction ≠ NOTHING) then
        → powaction
else if (countPowerUp(PowerUps.EMP, myCar.powerups) < 3) then
{ Hanya cari powerup EMP bila jumlahnya kurang dari 3}
    Command powaction ← TakePowerUp(Terrain.EMP, PowerUps.EMP,
    myCar.position.lane, myCar.position.block, gameState, myCar.speed)
    if (powaction ≠ NOTHING) then
        → powaction
```

```
else if (countPowerUp(PowerUps.BOOST, myCar.powerups) < 3) then
{ Hanya cari powerup boost bila jumlahnya kurang dari 3}
    Command powaction ← TakePowerUp(Terrain.BOOST, PowerUps.BOOST,
    myCar.position.lane, myCar.position.block, gameState, myCar.speed)
    if (powaction ≠ NOTHING) then
        → powaction
else if (countPowerUp(PowerUps.OIL, myCar.powerups) < 3) then
{ Hanya cari powerup oil bila jumlahnya kurang dari 3}
    Command powaction ← TakePowerUp(Terrain.OIL_POWER, PowerUps.OIL,
    myCar.position.lane, myCar.position.block, gameState, myCar.speed)
    if (powaction ≠ NOTHING) then
        → powaction
```

5. Mempercepat mobil

```
if (myCar.speed ≤ 6) then
{ Mengecek apakah kecepatan mobil kurang dari sama dengan 6 }
      → ACCELERATE
```

6. Penggunaan *Powerups*

```
if hasPowerUp(PowerUps.TWEET, myCar.powerups) then
{ Mengecek apakah mobil memiliki powerups tweet }
      if opponent.speed > 5 then
      { Mengecek apakah kecepatan lawan > 5, agar damage tweet maksimal }
            int yOp ← opponent.position.lane
            int xOp ← opponent.position.block
            int speedOp ← opponent.speed
            { Cyber truck akan diletakkan pada posisi mobil lawan + 1 }
            → new TweetCommand(yOp, xOp + speedOp + 1)


if hasPowerUp(PowerUps.EMP, myCar.powerups) then
{ Mengecek apakah mobil memiliki powerups EMP }
      if opponent.speed > 5 then
      { Memastikan kecepatan lawan > 5 agar damage EMP maksimal }
            if myCar.position.block > opponent.position.block then
            { Memastikan posisi lawan di depan posisi mobil }
                  int yOp ← opponent.positon.lane
                  int yMy ← myCar.position.lane
                  if yMy = yOp or yMy = yOp + 1 or yMy = yOp + 1 then
```

```
                    { Memastikan posisi lane lawan berada di jangkauan }
                        → EMP

    if hasPowerUp(PowerUps.BOOST, myCar.powerups) then
    { Mengecek terlebih dahulu apakah mobil memiliki Powerups Boost }
        if myCar.damage = 0 then
        { Mengecek apakah damage mobil sudah 0. Hal ini bertujuan agar tidak
          mengembalikan command NO_COMMAND }
            if blocksAreEmpty(getBlocksInFront(myCar.Position.lane,
                myCar.position.block, gameState, 15)) then
            { Mengecek apakah 15 block di depan lane mobil tidak terdapat
              obstacle apapun }
                → BOOST

    if hasPowerUp(PowerUps.OIL, myCar.powerups) and
       myCar.position.block > opponent.position.block then
        { Mengecek apakah mobil memiliki powerups oil dan posisi mobil di depan
          posisi mobil lawan }
        → OIL
```

7. Jika berada di belakang lawan, usahakan range EMP kita mengenai lawan

```
    if (opponent.position.block > myCar.position.block) then
    { Mengecek apakah mobil lawan di depan mobil kita }
        if (myCar.position.lane = 1) then
        { Jika berada di lane 1, belok kanan agar masuk ke tengah }
            if (blocksAreEmpty(2, block, gameState, speed-1)) then
                → TURN_RIGHT
        else if (myCar.position.lane = 2 and opponent.position.lane = 4) then
        { Jika berada di lane 2 dan lawan di lane 4, belok kanan agar lawan
        masuk range }
            if (blocksAreEmpty(3, block, gameState, speed-1)) then
                → TURN_RIGHT
        else if (myCar.position.lane = 3 and opponent.position.lane = 1) then
        { Jika berada di lane 3 dan lawan di lane 1, belok kiri agar lawan masuk
        range }
```

```
            if (blocksAreEmpty(2, block, gameState, speed-1)) then
                  → TURN_LEFT
      else if (myCar.position.lane = 4) then
      { Jika berada di lane 4, belok kiri agar masuk ke tengah }
            if (blocksAreEmpty(3, block, gameState, speed-1)) then
                  → TURN_LEFT
```

Jika berada di depan lawan, usahakan menghindar dari range emp lawan, peluang terbesar adalah ke tepi

```
if (opponent.position.block < myCar.position.block) then
{ Mengecek apakah kita di depan mobil lawan }
      if (myCar.position.lane = 2 and opponent.position.lane ≠ 4) then
      { Jika berada di lane 2 dan lawan tidak di lane 4, belok kiri }
            if (blocksAreEmpty(1, block, gameState, speed-1)) then
                  → TURN_LEFT
      else if (myCar.position.lane = 3 and opponent.position.lane ≠ 1) then
      { Jika berada di lane 3 dan lawan tidak di lane 1, belok kanan }
            if (blocksAreEmpty(4, block, gameState, speed-1)) then
                  → TURN_RIGHT
```

8. Opsi terakhir untuk mempercepat mobil

```
→ ACCELERATE
```

B. Struktur Data yang Digunakan Pada Bot Overdrive

Secara garis besar, struktur data yang digunakan diimplementasikan kedalam 3 (tiga) kelompok, yaitu *command*, *entities*, dan *enums* yang dimanfaatkan pada Main.java dan Bot.java. Bot.java sendiri memiliki struktur data tambahan untuk mendukung algoritma didalamnya.

1. Command
   Data pada command merupakan implementasi perintah atas aksi apa yang harus dilakukan bot pada suatu kondisi.

Atribut/Method	**Keterangan**
Command ACCELERATE	Perintah untuk menggunakan *accelerate* (meningkatkan *speed*)

Command LIZARD	Perintah untuk menggunakan *powerups* LIZARD (melewati *obstacle* tanpa masalah)
Command OIL	Perintah untuk menggunakan *powerups* OIL (memberikan *damage* pada yang melewatinya)
Command BOOST	Perintah untuk menggunakan *powerups* BOOST (membuat speed hingga maksimal)
Command EMP	Perintah untuk menggunakan *powerups* EMP (menembak lawan)
Command TweetCommand	Perintah untuk menggunakan *powerups* TWEET (menjatuhkan *cybertruck* ke posisi tertentu)
Command TURN_LEFT	Perintah untuk belok kiri
Command TURN_RIGHT	Perintah untuk belok kanan
Command NOTHING	Tidak melakukan apa-apa (akan bergerak lurus tergantung *speed* saat itu)

2. Entities
   a. Class Car
      menyimpan

Atribut/Method	**Keterangan**
int id	id untuk mengidentifikasi pemain
Position position	lokasi mobil pada saat tersebut
int speed	kecepatan mobil (0-15)
int damage	kerusakan mobil (0-5)
State state	keadaan terakhir mobil
Powerups[] powerups	*powerups* yang dimiliki mobil
Boolean boosting	*true* jika sedang menggunakan BOOST
int boostCounter	banyaknya BOOST yang digunakan

   b. Class GameState

Atribut/Method	**Keterangan**
int CurrentRound	*round game* pada saat tersebut
int maxRounds	maksimal *round* yang akan dilakukan dalam permainan
Car player	mobil pemain
Car opponent	mobil lawan
List<lane[]> lanes	*lane* yang ada pada permainan

   c. Class Lane

Atribut/Method	**Keterangan**
Position position	untuk mengidentifikasi posisi saat ini
Terrain terrain	untuk mengidentifikasi medan pada *block*
int occupiedByPlayerId	untuk mengidentifikasi id pemain yang menempati *lane* tersebut
Boolean cyberTruck	*true* jika terdapat cybertruck

d. Class Position

Atribut/Method	**Keterangan**
int lane	posisi pada *lane*/sb-Y (1-4)
int block	posisi pada *block*/sb-X (0-Finish)

3. Enums
   a. Direction
      Berfungsi untuk menentukan perpindahan mobil dengan mengubah data *block* dan *lane* berdasarkan label arah (forward, backward, left, right)
   b. PowerUps
      Powerups berfungsi untuk memberikan kemampuan tambahan pada player untuk menyerang lawan ataupun kekuatan untuk bertahan. Terdapat 5 *powerups* yang ada, yaitu boost, oil, tweet, dan lizard.
   c. State
      Data yang menyatakan kondisi mobil player pada *round* tersebut.
   d. Terrain
      Data yang menyatakan kondisi *block*, dapat diisi oleh *powerups*, *obstacle*, maupun *empty*.

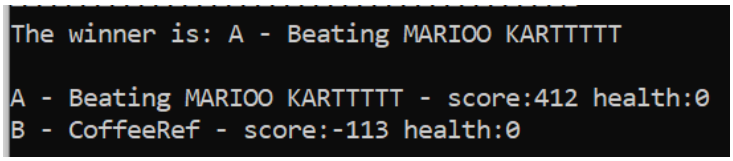
4. Class Bot

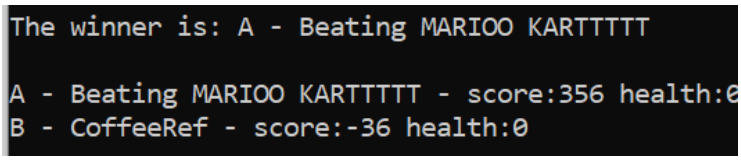
Atribut/Method	**Keterangan**
Command run()	Berisi percabangan *If* pada algoritma *greedy* beserta struktur data lokal yang diperlukan
Command TakePowerUp()	Me-*return command* untuk mengambil *powerups* (jika kondisi valid). Method ini dipanggil di dalam run() dengan parameter *powerups* yang ada.
boolean hasPowerUp()	Me-*return true* jika player memiliki *powerup* yang dicek, *false* jika tidak.
int countPowerUp()	Me-*return* banyaknya suatu *powerup* yang dimiliki player.
boolean blocsAreEmpty()	Me-*return true* jika semua *blocks* didepan yang akan dicapai adalah *Empty* (tidak terdapat *obstacle* apapun), *false* jika terdapat minimal 1 *obstacle*.
List<object> getBlocksInFront()	Me-*return* list block yang dapat dicapai player berdasarkan posisi dan kecepatannya.
Boolean checkCyberTruck	Me-*return true* jika terdapat *cyberTruck* pada *blocks* yang dicek, *false* jika tidak.

5. Class main
   Berisi 1 (satu) *method* utama yang me-*loop* pemanggilan bot.run di sepanjang jalannya permainan.

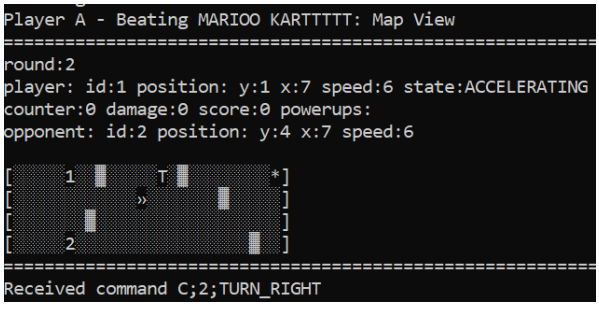
C. Analisis Hasil Implementasi Solusi Greedy Pada Setiap Pengujian

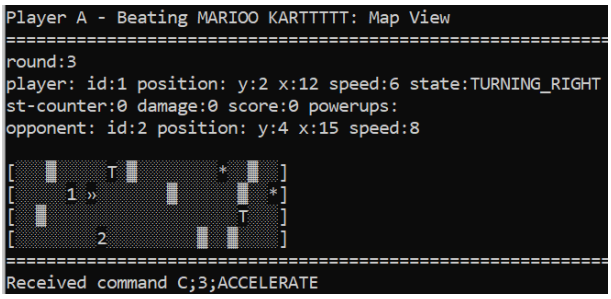
Berdasarkan pengujian yang telah dilakukan, solusi *Greedy* yang ditawarkan berhasil berjalan secara optimal. Bot yang diimplementasikan dengan solusi ini selalu dapat mengalahkan bot referensi secara mutlak. Dari setiap pengujian yang dilakukan melawan bot referensi, mobil pemain memiliki peforma yang baik hingga akhir. Sebab mobil tidak hanya diharapkan menjadi yang pertama mencapai finish, namun juga memiliki skor semaksimal mungkin. Tidak hanya melawan bot referensi, telah dilakukan pengujian melawan bot-bot yang lainnya. Pada beberapa pengujian, bot ini berhasil memenangkan permainan dengan baik. Namun, pada laporan ini tidak dapat disertakan data yang cukup merepresentasikan sebab terdapat banyak bot lain yang belum dicoba untuk ditandingkan. Berikut adalah beberapa hasil eksekusi melawan bot referensi.





Jika ditelusuri lebih dalam, bot berhasil membuat keputusan yang baik secara lokal berdasarkan algoritma greedy yang diterapkan. Sebagai contoh, pada *round* 2 di bawah jika mobil tetap bergerak lurus maka ia akan melewati *mud* yang akan memberikan *damage*. Dengan strategi poin 3, yaitu menghindari *obstacle*, mobil berhasil melewati *round* tersebut tanpa masalah dengan cara pindah ke *lane* di kanannya yang *empty*.





Pada pengujian yang telah dilakukan, kadang kala bot tidak melakukan *command* khusus apapun dalam strategi greedy yang diprioritaskan. Artinya, kondisi yang terjadi pada saat itu tidak memenuhi syarat pada strategi *Greedy*. Terdapat 2 (dua) kemungkinan yang dapat menyebabkan hal ini. Yang pertama, memang tidak ada hal khusus yang perlu dilakukan mobil pada *state* tersebut (misal ketika *blocks* di depan *empty* dan *speed* serta *damage* mobil aman, maka tidak diperlukan hal khusus). Yang kedua, terdapat detail kondisi-kondisi yang belum cukup ditangani sehingga bagian tersebut terlewati.

BAB V
KESIMPULAN, SARAN, DAN REFLEKSI

Salah satu algoritma yang populer dalam penyelesaian masalah terkait optimasi adalah algoritma *Greedy*. Pada Tugas Besar I Strategi Algoritma 2022 ini, algoritma Greedy diimplementasikan pada perancangan Bot dalam permainan *Overdrive*. Tujuan utama dari strategi ini adalah membuat keputusan terbaik pada tiap *round* nya dengan harapan Bot akan berhasil menang secara global. Algoritma *Greedy* dirancang dengan mempertimbangkan pilihan yang patut diprioritaskan sebab pemain dapat menghadapi berbagai kondisi di sepanjang permainan. Solusi yang kami rancang pada laporan ini berhasil memberikan hasil yang cukup baik, terbukti dari hasil eksperimen melawan bot referensi maupun bot yang lain.

Dalam pengerjaan tugas besar ini, kami menyadari segala kendala dan keterbatasan pada program yang kami buat. Untuk itu, kami mengharapkan kritik dan saran dari pihak-pihak yang turut mengevaluasi program maupun laporan ini. Tentunya, Bot ini kedepannya dapat dikembangkan untuk mendapatkan hasil yang lebih optimal menggunakan algoritma Greedy dengan alternatif strategi yang lebih detail. Tak sebatas itu, Bot ini juga dapat dikembangkan menggunakan algoritma lain yang lebih kompleks.

LAMPIRAN

Repository GitHub:

https://github.com/adellinekania/Tubes1_BeatingMarioKart

Video Kelompok:

https://youtu.be/NsWoe6UrwMA

DAFTAR PUSTAKA

EntelectChalleng (2020). *Entelect Challenge 2020 – Overdrive*. Diakses pada 7 Februari 2021, dari https://github.com/EntelectChallenge/2020-Overdrive

Levitin, Anany (2012). *Introduction to the Design and Analysis of Algorithms, 3rd Ed*. United States of America: Pearson Education Inc.

Munir, Rinaldi (2021). *Algoritma Greedy (2021) Bag. 1.* Diakses pada 14 Februari 2021, dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf