LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

Kelompok 47 - Optimum Pride

Disusun oleh:

Hansel Valentino Tanoto	13520046
Ziyad Dhia Rafi	13520064
Muhammad Naufal Satriandana	13520068

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB 1 – DESKRIPSI TUGAS

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

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*
    j. *USE_EMP*
    k. *FIX*
5. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.
6. Bot pemain yang pertama kali mencapai garis *finish* akan memenangkan pertandingan. Jika kedua bot mencapai garis *finish* secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar.

Adapun peraturan yang lebih lengkap dari permainan *Overdrive*, dapat dilihat pada laman:

https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md

BAB 2 – LANDASAN TEORI

A. ALGORTITMA *GREEDY*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan (biasanya persoalan optimiasi / *optimization problem*) secara langkah per langkah sedemikian sehingga, pada setiap langkah

1. Mengambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan
2. Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Algoritma *greedy* memiliki beberapa elemen antara lain:

1. Himpunan kandidat (C), yaitu kandidat yang akan dipilih di setiap Langkah
2. Himpunan solusi (S), yaitu kandidat yang sudah dipilih sejauh ini
3. Fungsi solusi, yaitu fungsi yang akan menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi/penyelesaian
4. Fungsi seleksi, yaitu fungsi yang memilih kandidat berdasarkan strategi *greedy* yang diimplementasikan
5. Fungsi kelayakan, yaitu fungsi yang akan mengecek kelayakan kandidat yang dipilih untuk dimasukkan ke dalam himpunan solusi
6. Fungsi objektif, yaitu fungsi untuk mengoptimisasi (memaksimumkan atau meminimumkan) himpunan solusi

Jadi algoritma *greedy* akan mencari sebuah *subset* S dari himpunan C dengan anggota S memenuhi fungsi-fungsi di atas dan akan dioptimisasi dengan fungsi objektif.

Pada algoritma *greedy*, optimum lokal belum tentu merepresentasikan optimum global karena algoritma *greedy* tidak melakukan pengecekan terhadap semua kemungkinan yang ada (tidak bisa *backtracking*) dan hasilnya akan bergantung dari fungsi seleksi yang digunakan sehingga algoritma ini sering digunakan untuk menghasilkan solusi aproksimasi/hampiran dalam waktu yang cukup cepat. Jadi bisa dikatakan, algoritma *greedy* mengorbankan akurasi untuk mempercepat waktu eksekusi.

Untuk menyatakan solusi dari suatu algortitma *greedy* adalah solusi optimal, akan diperlukan pembuktian matematis dan sebaliknya untuk membuktikan tidak optimal, hanya perlu dicari 1 buah kasus yang menunjukkan solusi tersebut tidak optimal (*counterexample*).

B. CARA KERJA *OVERDRIVE*

Pada *game Overdrive*, terdapat minimal 2 pemain (bot) yang akan bermain. Pada pengaturan *default*-nya, pemain (bot) pertama akan mulai balapan pada jalur (*lane*) 1 sedangkan pemain (bot) kedua akan mulai balapan pada jalur (*lane*) 4. Pemenang akan ditentukan berdasarkan pemain yang sampai garis *finish* paling pertama atau pemain dengan skor lebih tinggi apabila mencapai garis *finish* secara bersamaan atau pemain dengan kecepatan terbesar apabila mencapai garis *finish* secara bersamaan dan memiliki skor yang sama.

Pengaturan identitas bot dapat dilakukan dengan meng-*edit file* `bot.json` dengan *directory* `starter-pack/reference-bot/java/bot.json`. Identitas bot yang dimaksud meliputi nama

bot, nama dan *email author*/pembuat bot, letak *folder* dan nama *file* `.jar` dari bot, serta bahasa pemrograman yang digunakan dalam pembuatan bot. Sedangkan pengaturan konfigurasi *game* seperti konfigurasi peta permainan, perhitungan poin, *game state*, dan sebagainya dapat dilakukan dengan meng-*edit file* `game-config.json` pada *folder* `starter-pack`. Kemudian, terdapat *file* `game-runner-config.json` pada *folder* yang sama untuk menentukan lokasi *file-file* yang diperlukan untuk menjalankan program *Overdrive* termasuk lokasi bot untuk semua pemain.

Alur jalannya program *Overdrive* ini adalah sebagai berikut. Pertama-tama, *file* run.bat akan dijalankan yang diawali dengan pembacaan *file* konfigurasi permainan termasuk penentuan *seed* untuk *map*. Kemudian bot untuk semua pemain akan di-*load* ke program dan *game* dimulai. Untuk setiap ronde, bot akan menjalankan *method* `main` pada *class* `Main` yang berada di *directory* `starter-bots/java/src/main/java/za/co/entelect/challenge/Main.java`. *Method* `main` ini akan membaca *state* pada saat itu lalu memberikannya sebagai *input* untuk bot dalam *file* `Bot.java` (`starter-pack/starter-bots/java/src/main/java/za/co/entelect/challenge/Bot.java`). Kemudian bot akan memproses *input* tersebut menggunakan strategi *greedy* yang diimplementasikan dan *output*-nya yang berupa *command* akan diberikan kepada *method* main kembali untuk kemudian dijalankan dan dicetak hasilnya ke layar terminal dan *file* eksternal pada *folder* `match-logs`.

Implementasi algoritma *greedy* akan dituliskan pada *file* `bot.java`. Secara *default*, *File* ini berisi sebuah *class* `Bot` dengan beberapa data *private* (termasuk pendefinisian makro untuk *command* agar mempermudah pemanggilan *command*) dan *method*. *Method* utama dalam *file* ini adalah *method* `Bot` (konstruktor/*ctor* untuk menginisialisasikan data *private* pada *class* `Bot`) dan `run` (method tempat mengimplementasikan strategi *greedy* berupa percabangan untuk memilih keputusan terbaik saat itu/maksimum lokal). Selain itu dapat didefinisikan beberapa fungsi/*method* tambahan lainnya untuk mendukung *method* run tersebut. Implementasi algoritma *greedy* tersebut umumnya berupa percabangan untuk menentukan prioritas penggunaan *command* pada saat itu sesuai informasi *state* saat itu. Sebelum dapat dijalankan, bot harus di-*install* terlebih dahulu menggunakan `maven` dengan mengklik `Maven > java-starter-bot > Lifecycle > install` sehingga akan tercipta *file* `.jar` untuk bot pada *folder* `target` (`starter-pack/starter-bots/java/target`). Dan saat program dijalankan (*file* `run.bat`), *file* `.jar` ini akan dieksekusi sesuai alur di atas.

BAB 3 – APLIKASI STRATEGI *GREEDY*

A. *MAPPING* PERSOALAN MENJADI ELEMEN ALGORITMA *GREEDY*

Himpunan kandidat pada persoalan *Overdrive* ini berupa semua *command* yang dapat dilakukan oleh pemain (bot) yaitu:

1. *NOTHING* : mobil tetap di *lane* saat itu dan kecepatan tetap.
2. *ACCELERATE* : mobil tetap di *lane* yang sama tetapi menambah kecepatannya ke *speed state* berikutnya (maksimal = 9).
3. *DECELERATE* : mobil tetap di *lane* yang sama tetapi mengurangi kecepatannya ke *speed state* sebelumnya (minimum = 0).
4. *TURN_LEFT* : mobil berpindah 1 *lane* ke arah kiri (nomor *lane* lebih kecil).
5. *TURN_RIGHT* : mobil berpindah 1 *lane* ke arah kanan (nomor *lane* lebih besar).
6. *USE_BOOST* : mobil menggunakan power up *boost* sehingga kecepatan bertambah menjadi 15 (*boost speed*).
7. *USE_OIL*: : mobil meletakkan *oil spill* tepat di posisi mobil saat itu.
8. *USE_LIZARD*: : mobil akan lompat pada *lane* yang sama menghindari semua *obstacle* dan *power up* yang ada sejauh kecepatan mobil saat itu.
9. *USE_TWEET <lane> <block>* : memunculkan sebuah *cyber truck* pada *lane* dan *block* yang dituliskan untuk menghalangi mobil lawan.
10. *USE_EMP*: : menembakkan *EMP blast* ke depan yang dapat membuat mobil lawan berhenti dan *speed*-nya berkurang menjadi 3
11. *FIX*: : memperbaiki mobil dengan menghilangan 2 *damage point*.

Himpunan solusi pada persoalan ini berupa beberapa atau semua *command* yang terpilih dan digunakan dalam permainan *Overdrive* tersebut beserta dengan urutan prioritasnya sesuai strategi *greedy* (fungsi seleksi) yang digunakan.

Fungsi solusi akan memeriksa apakah *command* yang dipilih akan bermanfaat/berguna untuk mencapai kemenangan atau malah sia-sia. Misalnya jika melakukan *command USE_EMP* saat di depan tidak ada kendaraan musuh akan menyebabkan *power up* tersebut terbuang sia-sia. Atau *USE_BOOST* saat di depan masih terdapat *obstacle*.

Fungsi seleksi dari persoalan *Overdrive* dapat bermacam-macam sesuai strategi *greedy* yang diimplementasikan. Beberapa diantaranya yaitu, memilih *command* memprioritaskan pengambilan *power up*, memilih *command* yang memprioritaskan penggunaan *power up* (baik untuk mengoptimasi diri sendiri maupun yang agresif / untuk merugikan lawan), memilih *command* yang memprioritaskan untuk menghindari *obstacle*, atau memilih *command* yang mementingkan / selalu mengoptimasi besarnya kecepatan mobil.

Fungsi kelayakan di sini akan memeriksa apakah *command* yang dipilih valid atau dapat dijalankan pada *state* saat itu. Hal ini diimplementasikan dalam bot dengan percabangan yang mengecek kondisi *state* saat itu apakah memungkinkan untuk menjalankan suatu *command*. Constraint / batasan untuk setiap *command*, yaitu:

- *Command TURN_LEFT* hanya dapat dijalankan apabila mobil berada pada nomor *lane* > 1
- *Command TURN_RIGHT* hanya dapat dijalankan apabila mobil berada pada nomor *lane* < 4

- *Command* untuk menggunakan *power up* (*USE_**)hanya dapat digunakan apabila pemain memilikinya di *inventory*
- *Command FIX* hanya dapat dijalankan apabila *damage* mobil > 0
- *Command ACCELERATE* hanya dapat dijalankan apabila kecepatan mobil < kecepatan maksimum pada state tersebut
- *Command DECELERATE* hanya dapat dijalankan apabila kecepatan mobil > 0

Fungsi objektif pada permainan *Overdrive* ini adalah untuk mencapai kemenangan yaitu dengan cara mencapai garis *finish* lebih dulu atau sampai di garis *finish* secara bersamaan dengan poin lebih tinggi atau sampai di garis *finish* secara bersamaan dan memiliki poin yang sama tetapi dengan kecepatan yang lebih besar.

B. ALTERNATIF SOLUSI *GREEDY*

***Greedy* 1:**

Program akan semaksimal mungkin tidak menabrak *obstacle* (*wall* / *mud* / *oil spill*) atau mencari *lane* dengan *damage obstacle* paling sedikit tanpa memedulikan pengambilan dan penggunaan *powerup*. Urutan prioritas sebagai berikut:

1. *USE_BOOST* jika:
- Tidak ada *obstacle* (*mud*, *wall*, atau *oil spill*) di depan, sejauh 15 blok (*boost speed*)
- Memiliki *power up boost*
- *Speed* setelah *boost* lebih besar dari *accelerate*
2. *ACCELERATE* jika:
- Tidak ada *obstacle* (*mud*, *wall*, atau *oil spill*) di depan, sejauh (*accelerated speed*) blok
- *Speed* setelah *accelerate* lebih cepat dari *speed* saat ini (belum mencapai kecepatan tertinggi), selain itu: *fix* jika *damage* > 1
3. Pilih *lane* terbaik:
- *Stay* di *lane* jika *damage* di tengah paling sedikit (menggunakan *power up* agresif)
- *TURN_RIGHT* jika *damage* di kanan paling sedikit
- *TURN_LEFT* jika *damage* di kiri palling sedikit

***Greedy* 2:**

Program akan sebisa mungkin menghindari kemungkinan tabrakan dengan mobil lawan dan *obstacle* (terutama *wall* karena memberikan efek penurunan kecepatan yang paling parah dibandingkan *obstacle* lainnya), serta menggunakan (*spam*) *power up* yang bersifat agresif yaitu *USE_EMP* dan *USE_TWEET*. Urutan prioritas strategi ini adalah sebagai berikut:

1. Lakukan *command FIX* jika *damage* ≥ 4
2. Pilih *lane* terbaik:
- Ketika melihat adanya wall atau mobil lawan di depan, segera lakukan manuver ke kanan (*TURN_RIGHT*) atau kiri (*TURN_LEFT*) tergantung dengan ada atau tidaknya *wall* di *lane* tersebut dan jumlah *obstacle* yang ada. Sebisa mungkin bot akan memilih *lane* yang tidak ada *wall*-nya atau ada *wall* tetapi dengan jumlah paling sedikit.

- Jika jumlah *wall* masih sama (belum bisa diputuskan), selanjutnya bot akan mempertimbangkan jumlah dan keberadaan *obstacle* lainnya (*mud* dan *oil spill*) yaitu dengan memilih *lane* dengan jumlah *mud* dan *oil spill* paling sedikit.
- Ketika tidak ada *wall* di depan, lakukan *command ACCELERATE.* Namun, jika ada *obstacle* lain pertimbangkan *lane* yang dipilih dengan prioritas seperti poin di atas.

3. Jika memiliki *power up tweet*, gunakan sesegera dan sebanyak mungkin command *USE_TWEET* pada posisi kira-kira sejauh (*speed* mobil lawan+3) blok dari posisi lawan saat itu.
4. Jika memiliki *power up EMP* dan musuh terdeteksi berada di depan, gunakan sesegera dan sebanyak mungkin command *USE_EMP*
4. Gunakan *power up* yang lain dengan urutan prioritas *USE_BOOST* > *USE_LIZARD* > *USE_OIL* jika memilikinya. *USE_OIL* hanya digunakan jika mobil musuh terdeteksi ada di belakang.
5. Jika semua *command* di atas tidak dilakukan, *default command* adalah *ACCELERATE*

Pada strategi ini, pengambilan *power up* bukanlah prioritas sehingga *power up* hanya akan diambil 9sambil menjalankan strategi di atas.

***Greedy* 3:**

Program akan mengutamakan pengambilan *power up*, penggunaan *power up boost* dan juga menghindari *obstacle* yang ada, terutama *wall* dengan alasan *obstacle* tersebut akan langsung mengurangi *speed* kendaraan menjadi 3. Penggunaan power up yang paling diprioritaskan adalah *boost* dengan mencoba mempertahankan status *boosting* pada mobil selama 5 ronde (sesuai jangka waktu maksimal efek *boost*). Urutan prioritas strategi *greedy* ini yaitu:

1. Gunakan *command FIX* jika *damage* mobil ≥ 3
2. ACCELERATE mobil jika kecepatannya sudah mencapai 0 (tidak bisa bergerak)
3. Pemilihan *lane* dengan prioritas:
- Mencari *power up* dengan prioritas *boost* dan *lizard* lebih besar dibanding prioritas *tweet* dan *EMP*.
- Menghindari *obstacle* dengan pertama-tama mencari *lane* yang kosong (tidak ada *obstacle*). Kemudian jika tidak ada *lane* yang kosong, akan digunakan *lizard* jika ada untuk melompat menghindari *obstacle* tersebut. Jika masih belum memenuhi kondisi tersebut, maka akan dicari lane yang tidak mengandung *wall*. Apabila masih belum ditemukan pilihan *lane* yang sesuai, bot akan memeriksa urutan prioritas berikutnya
- Menghindari mobil lawan dengan strategi yang kurang lebih mirip dengan strategi pada poin sebelumnya

4. Menggunakan *command USE_BOOST* jika memilikinya dan tidak ada halangan / *obstacle* untuk 15 blok ke depan, serta tidak sedang menggunakan *boost* (*speed* mobil < *boosting speed* = 15). Selain itu jika terdapat akumulasi *damage* mobil > 0, akan dilakukan *command FIX* terlebih dahulu sehingga *command USE_BOOST* akan digunakan pada kesempatan berikutnya apabila ditemui *state* yang sesuai kondisi di atas lagi.
5. Menggunakan *power up* yang lainnya jika ada, dengan prioritas *USE_EMP* (hanya digunakan jika mobil musuh terdeteksi ada di depan) > *USE_TWEET* > *USE_OIL* (hanya digunakan jika mobil musuh terdeteksi ada di belakang).

C. ANALISIS EFISIENSI ALTERNATIF ALGORITMA *GREEDY*

***Greedy* 1:**

Algoritma ini tidak memiliki perhitungan yang begitu kompleks. Program terdiri dari beberapa *if statement* yang hanya melihat kondisi mobil saat ini. Komputasi paling besar terjadi saat melakukan iterasi terhadap *blocks* yang ada di depan ataupun terhadap *powerup* yang dimiliki. Komputasi tersebut memiliki satu kalang di dalamnya. Dengan demikian, kompleksitas algoritma pada program ini adalah O(n).

***Greedy* 2:**

Sama seperti strategi sebelumnya, mayoritas blok program pada algoritma ini hanya berupa percabangan *if statement* sedangkan untuk *loop* hanya dilakukan saat mendeteksi blok di *lane* saat ini, *lane* kiri, dan *lane* kanan, serta iterasi *inventory power up*. Jadi secara umum program ini memiliki kompleksitas dalam notasi Big-O sebesar O(n) dengan n adalah jumlah blok yang dapat dilihat / dievaluasi oleh mobil pada *state* saat itu (sesuai kecepatan maksimal mobil). Namun, jika *power up* jarang digunakan sehingga menumpuk di *inventory*, maka iterasi *inventory* akan cukup berpengaruh pada kompleksitas algoritma tetapi masih dalam batasan O(n) / kompleksitas linear.

***Greedy* 3:**

Algoritma *greedy* ini juga tidak memiliki banyak kalang/*loop* dan hanya mengandung banyak percabangan sehingga kompleksitasnya juga dapat dinyatakan dalam notasi Big-O sebagai O(n). *Loop* yang terdapat di program ini hanya digunakan untuk mendeteksi blok di depan, kiri, dan kanan mobil serta untuk *searching* pada *inventory power up*. Jadi program ini memiliki kompleksitas yang bersifat linear.

D. ANALISIS EFEKTIVITAS ALTERNATIF ALGORITMA *GREEDY*

***Greedy* 1:**

Algoritma ini memprioritaskan mobil agar tidak terkena *obstacle* sama sekali. Mobil mencari *lane* dengan *damage* paling sedikit sehingga jarang rusak. Mobil hanya akan melakukan *command FIX* jika mobil seharusnya *ACCELERATE* tetapi *speed* tidak dapat bertambah karena memiliki *damage*. Namun, algoritma ini tidak memperhitungkan pengambilan *boost* atau *power up* lain yang sebenarnya cukup berguna untuk memenangkan balapan.

***Greedy* 2:**

Masih mirip dengan strategi sebelumnya, algoritma ini tidak memprioritaskan pengambilan *power up* yang penting dalam mengoptimasi kondisi/*state* mobil. Selain itu penggunaan *command FIX* hanya dilakukan saat *damage* ≥ 4 untuk mengindari mobil terlalu sering berhenti, tetapi efek negatifnya adalah mobil akan sulit mencapai kecepatan optimum karena setiap perbaikan, *damage* yang berkurang hanya 2 poin (tidak pernah memperbaiki sampai *damage* = 0). Namun, dengan prioritas utamanya untuk menghindari *obstacle*, *damage* yang terakumulasi mungkin akan lebih jarang/sulit mencapai nilai yang besar. Kemudian, karena penggunaan beberapa *power up* yang menjadi prioritas akhir, akan ada kemungkinan *power up* menumpuk di *inventory* karena bot cenderung mengeksekusi *command* yang ada di prioritas atas.

***Greedy* 3:**

Secara umum, algoritma *greedy* kali ini lebih baik dibandingkan algoritma 1 dan 2 di atas yang dapat dibuktikan dengan kemenangan yang selalu diraih bot dengan algoritma ini pada saat pengetesan. Dalam permainan *Overdrive* ini, *power up* merupakan salah satu hal yang penting sehingga strategi *greedy power up* ini cukup efektif. Dengan mempertahankan dan memanfaatkan status *boosting* serta menghindari *obstacle* yang ada, bot dengan algoritma ini dapat melaju rata-rata 40 blok selama 5 ronde dan maksimal 75 blok (15 blok x 5 ronde) jika mendapat *lane* yang bebas *obstacle*. Selain itu, perbaikan mobil dilakukan setiap akumulasi *damage* mobil ≥ 3 sehingga dapat dicapai kecepatan maksimal yang optimum (karena *max speed* bergantung akumulasi *damage*). Hal yang kurang dioptimalkan dalam algoritma ini adalah penggunaan *power up* yang lain karena beberapa di antaranya menjadi prioritas akhir.

E. STRATEGI *GREEDY* TERPILIH

Algoritma *greedy* yang akhirnya terpilih adalah algoritma yang memaksimalkan penggunaan power up boost (*greedy* 3). Tentunya sebelum itu algoritma ini juga harus bisa menghindari beberapa rintangan seperti *wall*, *mud*, *oil spill*, dan *player*, dengan urutan prioritas tersebut. Inti dari cara kerja algoritma ini adalah melakukan *FIX* hingga tidak ada *damage* pada giliran bot hendak menggunakan *command USE_BOOST*. Dengan melakukan hal tersebut, penggunaan *boost* menjadi lebih optimal karena mobil akan melaju sejauh 15 blok daripada hanya 9 blok saja dan dapat bertahan selama 5 ronde jika *state map*-nya mendukung. Selain itu program ini juga melakukan berbagai hal agresif seperti menggunakan *power up* jika tidak ada lagi yang bisa dilakukan. Berdasarkan pengujian, algoritma ini juga secara umum lebih unggul dibandingkan algoritma 1 dan 2.

BAB 4 – IMPLEMENTASI DAN PENGUJIAN

A. IMPLEMENTASI ALGORITMA *GREEDY*

Algoritma program utama yang terdapat pada *method* `run` di *Class* `Bot` adalah sebagai berikut:

```
//Minimalisasi damage agar dapat mencapai speed minimal 8
if(damage mobil >= 3) then
    fix mobil
//Jika mobil tidak bergerak maka harus digerakkan
if(speed mobil <= 0) then
    accelerate mobil


//Mobil akan mencari power up jika mungkin dengan mengutamakan
power up boost dan lizard >> power up emp dan tweet
if(mobil tidak di lane paling atas) then
    if(tidak ada mud, wall atau oil spill di lane kiri dan
terdapat boost atau lizard)
        belok kiri
if(mobil tidak di lane paling bawah) then
    if(tidak ada mud, wall atau oil spill di lane kanan dan
terdapat boost atau lizard)
        belok kanan
if(mobil tidak di lane paling atas) then
    if(tidak ada mud, wall atau oil spill di lane kiri dan
terdapat emp atau tweet)
        belok kiri
if(mobil tidak di lane paling bawah) then
    if(tidak ada mud, wall atau oil spill di lane kanan dan
terdapat emp atau tweet)
        belok kanan

//Periksa untuk rintangan di depan (apakah ada obstacle)
if(ada mud, wall, atau oil spill didepan) then
    //Pertama-tama cari lane kosong
    if(mobil tidak di lane paling atas) then
        if(tidak ada mud, wall, dan oil spill di kiri) then
            belok kiri
    if(mobil tidak di lane paling bawah) then
        if(tidak ada mud, wall, dan oil spill di kanan) then
            belok kanan
    //Jika tidak ada lane kosong, jump dengan power up lizard
    if(punya lizard) then
        use lizard
    //Jika semua tidak mungkin, minimal cari lane tanpa wall
    if(mobil tidak di lane paling atas) then
        if(wall di kiri) then
            belok kiri
    if(mobil tidak di lane paling bawah) then
        if(tidak ada wall di kanan) then
            belok kanan
```

```
    //Jika semua tidak terpenuhi, terpaksa menabrakkan diri

//Periksa jika mobil musuh di depan
if(ada musuh di depan) then
    //Pertama-tama cari lane kosong
    if(mobil tidak di lane paling atas) then
        if(tidak ada mud, wall, dan oil spill di kiri) then
            belok kiri
    if(mobil tidak di lane paling bawah) then
        if(tidak ada mud, wall, dan oil spill di kanan) then
            belok kanan
    //Jika tidak ada lane kosong, jump dengan power up lizard
    if(punya lizard) then
        use lizard
     //Jika semua tidak mungkin, pilih belok ke kiri atau kanan
    if(mobil tidak di lane paling atas) then
        belok kiri
    if(mobil tidak di lane paling bawah) then
        belok kanan

//Jika punya boost, pastikan terlebih dahulu speed bisa mencapai
15
if(punya boost dan tidak ada halangan sejauh 15 block di depan
dan speed < 15) then
    //Jika masih ada damage, fix terlebih dulu
    if(damage mobil > 0) then
        fix mobil
    //Jika tidak, langsung boost
    use boost

//Jika punya emp
if(punya emp dan mobil musuh di depan) then
    use emp

//Jika punya tweet
if(punya tweet)
    use tweet di sekitar musuh, yakni di depan musuh sejauh
kecepatan musuh + 3

//Jika punya oil
if(punya oil dan mobil musuh di belakang) then
    use oil
```

Selain algoritma program utama tersebut, terdapat beberapa fungsi tambahan/pendukung yang digunakan yaitu *method* untuk menentukan isi blok di depan, mengecek *inventory power up*, dan menentukan *state* kecepatan berikutnya jika melakukan *ACCELERATE*.

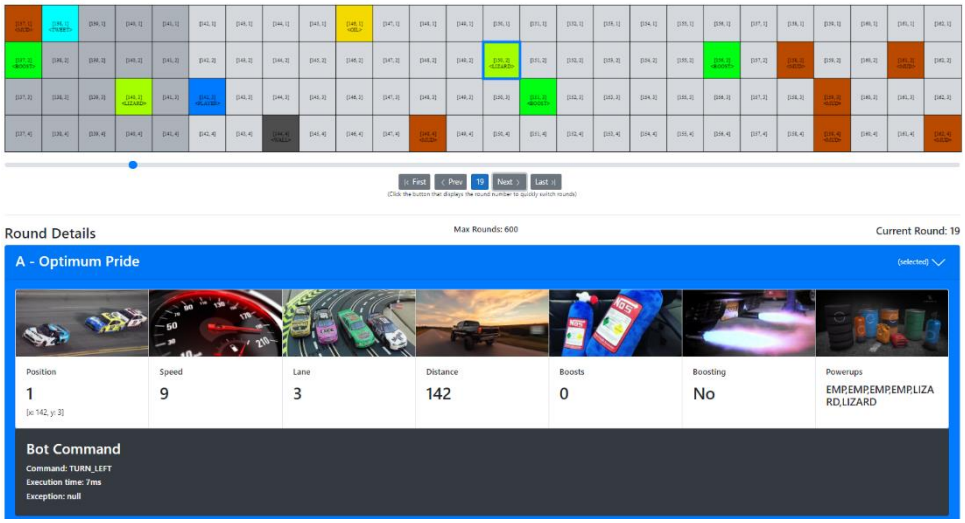
B. STRUKTUR DATA PROGRAM

Karena pemrograman yang dilakukan beorientasi objek, kebanyakan struktur data terdiri dari kelas dan objek, yang dalam kasus ini, adalah kelas bot. Namun ada struktur data tambahan yang

digunakan, yakni *list of object* yang digunakan untuk menampung *block* di depan maupun samping kanan mobil. Dalam program ini, list tersebut bernama `blocks`, `acceleratedBlocks`, `boostBlocks`, `leftBlocks` dan `rightBlocks`.

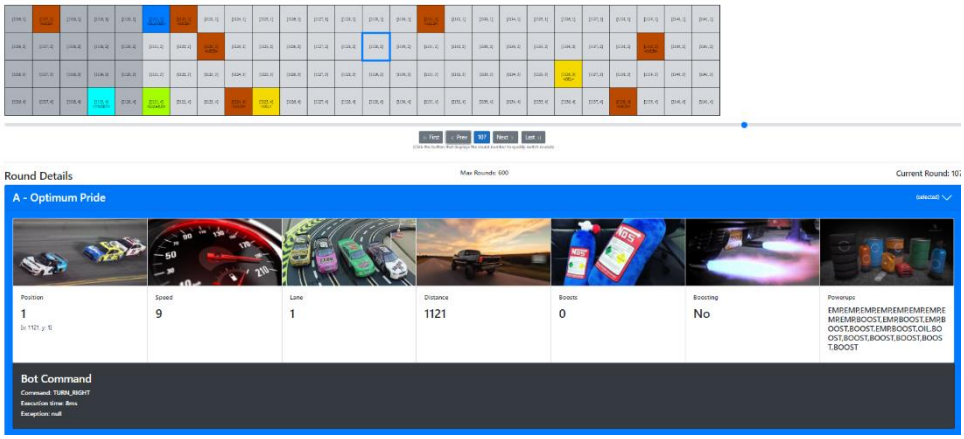
C. ANALISIS DESAIN SOLUSI ALGORITMA *GREEDY*

Dilakukan lomba antara Optimum Pride dan CoffeeReff. Algoritma greedy yang akan ditinjau adalah algoritma *greedy* milik Optimum Pride (biru).



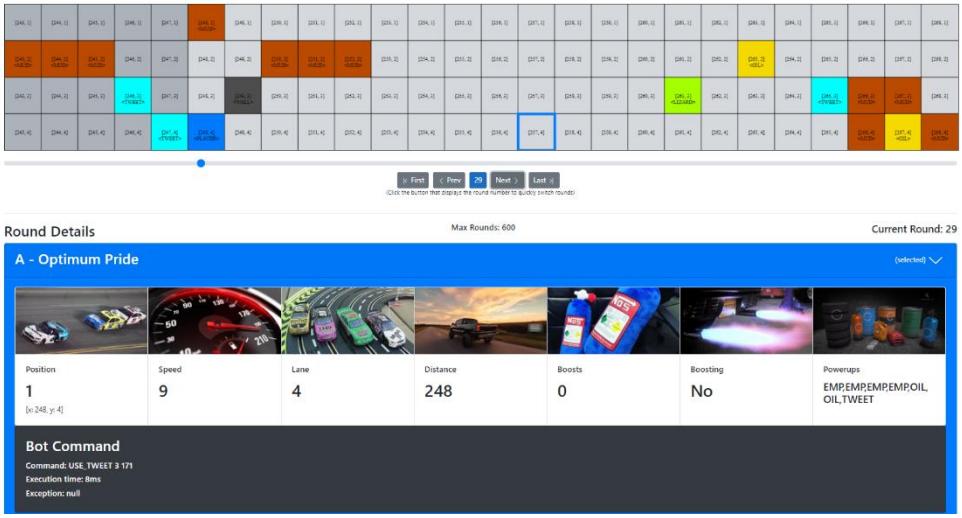
Gambar 4.1 Bot mengincar power up lizard

Pada kasus ini, algoritma *greedy* yang dilakukan adalah mengincar *power up lizard*. Algoritma ini berhasil mendapatkan nilai optimal.



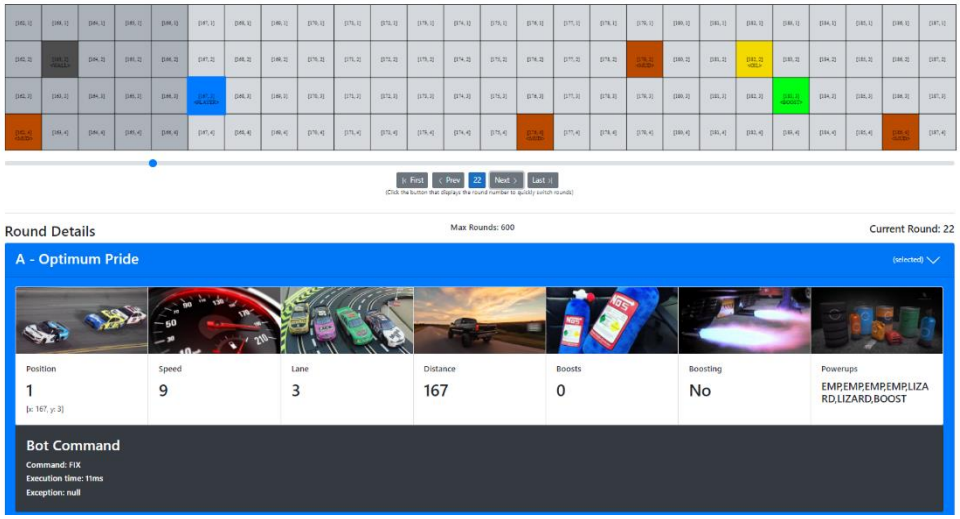
Gambar 4.2 Bot menghindari mud dan megnincar lane tanpa wall

Pada kasus ini, bot tidak memiliki *lizard* dan di depan maupun di kanan ada *mud*. Bot akan belok kanan karena minimal tidak ada *wall*. Dalam kasus ini algoritma sudah optimal, namun dapat ditingkatkan lagi dengan menghitung jumlah *mud* dan jumlah *power up* terlebih dahulu sebelum belok.



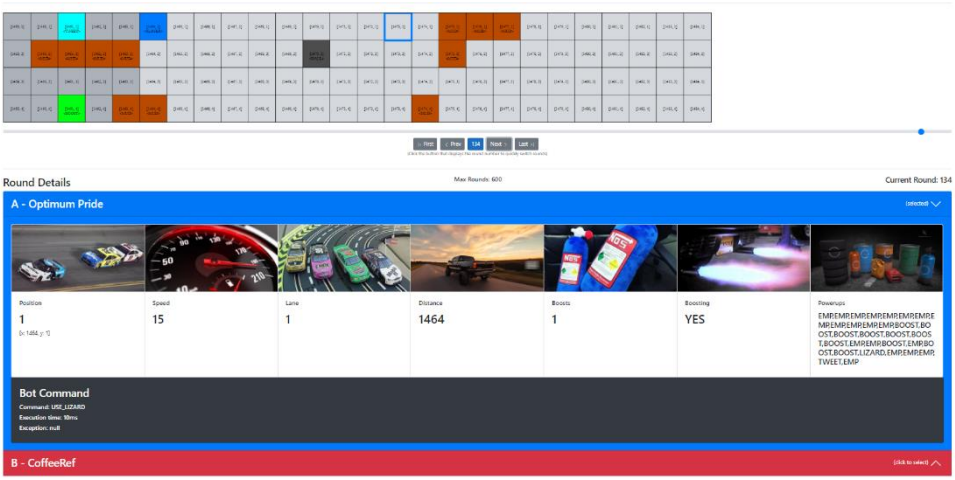
Gambar 4.3 Bot menggunakan tweet karena tidak ada lagi yang bisa dilakukan

Pada kasus ini, bot menggunakan *tweet* karena tidak bisa *accelerate* lagi. Algoritma sudah berhasil mendapatkan nilai optimal.



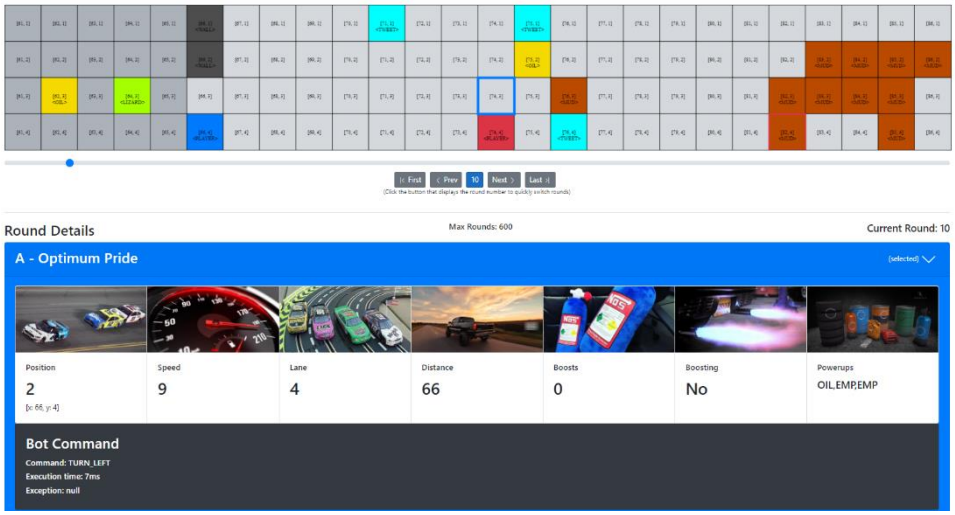
Gambar 4.4 Bot melakukan fix sebelum menggunakan boost

Pada kasus ini, mobil masih melaju dengan kecepatan 9 namun sudah melakaukan *fix*. Hal tersebut dilakukan karena kondisi untuk menggunakan *boost* akan terpenuhi pada giliran berikutnya. Alhasil, bot akan mendapatkan kompensasi kehilangan pergerakan 9 *block* dengan *boost* sejauh 15 *block* sebanyak maksimal 5 kali. Pada kasus ini, algoritma berhasil mendapatkan nilai optimal.



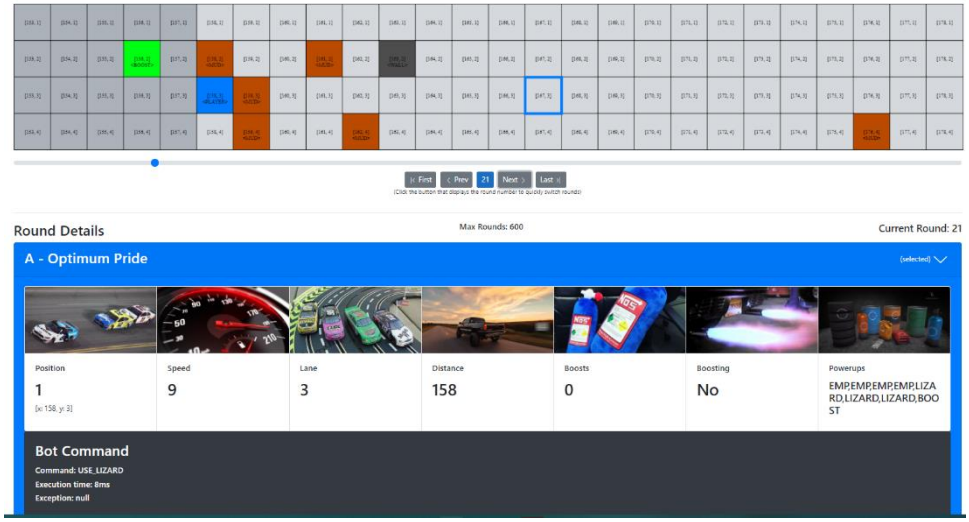
Gambar 4.5 Bot menggunakan lizard karna berasumsi akan menabrak mud

Pada kasus ini mobil tampak menggunakan *lizard* tanpa alasan. Namun sebenarnya hal ini terjadi karena sedikit kekeliruan algoritma untuk melihat *block* di depan maupun samping depan. Yang digunakan untuk melihat di depan dan di samping depan adalah *speed* dari mobil yang dalam kasus ini adalah 15. Seharusnya menggunakan *boost timer* juga untuk menentukan apakah *boost* sudah berakhir atau tidak. Walaupun dalam kasus ini tidak ada masalah, akan menjadi masalah jika sebenarnya di depan ada banyak *power up*, tetapi mobil malah menggunakan *lizard* atau berbelok ke *lane* yang kosong.



Gambar 4.6 Bot berbelok karena ada mobil musuh

Pada kasus ini, bot sudah mendapatkan hasil optimal karena telah menghindari tabrakan, walaupun belum bisa dipastikan apakah akan menabrak.



Gambar 4.7 Bot mengunakan lizard karena tidak ada lane kosong di depan

Pada kasus ini bot sudah mendapatkan algoritma optimal.

BAB 5 – KESIMPULAN DAN SARAN

A. KESIMPULAN

Dalam tugas besar kali ini, kami telah berhasil membuat sebuah bot *Override* yang didasarkan pada algoritma *Greedy*. Algoritma ini mencari hasil terbaik lokal (pada tiap *round*) dengan harapan mencarai hasil terbaik gobal. Bot ini telah berhasil mengalahkan bot lain dengan *finish race* lebih dahulu. Strategi yang kami gunakan berfokus pada menghindari obstacle dan mengambil *powerUp*.

B. SARAN

Dalam pengembangannya, algoritma ini masih dapat dibuat lebih baik lagi. Seperti pada analisis desain di bab 4, program masih dapat menggunakan informasi *boost timer* untuk mengetahui apakah *boost* sudah selesai atau belum. Selain itu, program bisa ditingkatkan menjadi melihat 4 lane sekaligus agar bisa merencanakan sesuatu dalam dua giliran. Selain itu dalam menghindari *obstacle*, dapat juga dipertimbangan untuk mengevaluasi informasi jumlah *obstacle* pada suatu *lane*.

DAFTAR PUSTAKA

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

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

KODE SUMBER

Link Repository GitHub: https://github.com/ziyaddr/Tubes1_Optimum-Pride