TUGAS BESAR I
Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"
IF2211 – Strategi Algoritma
K-03

Disusun oleh
pick_up

Fitrah Ramadhani Nugroho	**13520030**
Tri Sulton Adila	**13520033**
Muhammad Alif Putra Yasa	**13520135**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

DAFTAR ISI

BAB 1
DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Overdrive. Game engine dapat diperoleh pada laman berikut: https://github.com/EntelectChallenge/2020-Overdrive

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan 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 power ups. 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

1.2 Spesifikasi Tugas

Pada tugas besar kali ini, anda diminta untuk membuat sebuah bot untuk bermain permainan Overdrive yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

Download latest release starter pack.zip dari tautan berikut
https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4

1. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut:
   a. Java (minimal Java 8): https://www.oracle.com/java/technologies/downloads/#java8
   b. IntelIiJ IDEA: https://www.jetbrains.com/idea/
   c. NodeJS: https://nodejs.org/en/download/
2. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk Windows dapat buka dengan double-click, Untuk Linux/Mac dapat menjalankan command “make run”).
3. Secara default, permainan akan dilakukan diantara reference bot (default-nya berbahasa Java) dan starter bot (default-nya berbahasa JavaScript) yang disediakan. Untuk mengubah hal tersebut, silahkan edit file “game-runner-config.json”. Anda juga dapat mengubah file “bot.json” dalam direktori “starter-bots” untuk mengatur informasi terkait bot anda.
4. Silahkan bersenang-senang dengan memodifikasi bot yang disediakan di starter-bots. Ingat bahwa bot kalian harus menggunakan bahasa Java dan di-build menggunakan IntelIiJ sebelum menjalankan permainan kembali. Dilarang menggunakan kode program yang sudah ada untuk pemainnya atau kode program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.
5. (Optional) Anda dapat melihat hasil permainan dengan menggunakan visualizer berikut https://github.com/Affuta/overdrive-round-runner
6. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut https://github.com/EntelectChallenge/2020-Overdrive

Strategi greedy yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mencapai garis finish lebih awal atau mencapai garis finish bersamaan tetapi dengan kecepatan lebih besar atau memiliki skor terbesar jika

kedua komponen tersebut masih bernilai imbang. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menggunakan power ups begitu ada untuk mengganggu mobil musuh. Buatlah strategi greedy terbaik, karena setiap bot dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (TBD).

Strategi greedy harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB 2
LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* adalah algoritma yang menyelesaikan permasalahan secara langkah per langkah. Untuk setiap langkahnya, algoritma *greedy* akan mencoba untuk mencari pilihan terbaik yang dapat diperoleh pada saat itu dan mengambilnya tanpa mempedulikan konsekuensi yang akan diperoleh ke depan. Dengan mengambil pilihan yang dirasa terbaik pada setiap langkah (optimum lokal), algoritma ini berharap untuk mendapatkan solusi terbaik secara umum (optimum global). Algoritma *greedy* sering kali tidak memberikan solusi yang paling optimal. Meskipun begitu, algoritma ini dapat memberikan solusi yang mendekati solusi terbaik dalam waktu yang masih wajar. Algoritma *greedy* lebih cocok untuk menyelesaikan permasalahan yang tidak membutuhkan hasil yang benar-benar eksak, tetapi hasil berupa hampiran atau aproksimasi. Beberapa permasalahan populer yang dapat diselesaikan dengan menggunakan pendekatan *greedy*, yaitu *knapsack problem*, *activity selection problem*, *Dijkstra's problem*, mencari *minimum spanning tree* dengan algoritma Prim dan Kruskal, *Huffman coding*, dan *travelling salesman problem*.

Secara umum, algoritma *greedy* terdiri atas beberapa elemen, antara lain:

1. Himpunan Kandidat
   Elemen ini berisikan sekumpulan kandidat yang dapat dipilih untuk setiap langkah.
2. Himpunan Solusi
   Elemen ini berisikan sekumpulan kandidat yang telah dipilih.
3. Fungsi Solusi
   Fungsi solusi merupakan fungsi yang menentukan apakah himpunan solusi sudah berhasil menyelesaikan persoalan.
4. Fungsi Seleksi
   Fungsi seleksi merupakan fungsi yang memilih kandidat berdasarkan strategi tertentu yang bersifat heuristik.
5. Fungsi Kelayakan
   Fungsi kelayakan merupakan fungsi yang memeriksa apakah kandidat yang diambil dapat dimasukkan ke himpunan solusi.
6. Fungsi Objektif
   Fungsi objektif merupakan fungsi yang perlu dimaksimalkan atau diminimalkan.

Berikut merupakan kelebihan dan kekurangan algoritma *greedy*

- Kelebihan
  1. Teknik ini mudah untuk diformulasikan dan diimplementasikan
     Bekerja efisien untuk banyak kasus
  2. Pendekatan *greedy* meminimalkan waktu yang dibutuhkan untuk menghasilkan solusi
- Kekurangan
  1. Pendekatan *greedy* tidak menjamin solusi optimal global karena algoritma ini tidak pernah kembali ke pilihan yang sudah dibuat untuk mencari solusi optimal lokal

2.2 Pemanfaatan dan Cara Kerja Program

2.2.1. Sistem Bot Melakukan Aksinya

Permainan terdiri dari ronde-ronde yang setiap rondenya, kedua mobil akan bergerak sesuai perintah dari bot masing-masing. Terdapat 11 perintah yang dapat dieksekusi bot, antara lain:

1. NOTHING
2. ACCELERATE
3. DECELERATE
4. TURN_LEFT
5. TURN_RIGHT
6. USE_BOOST
7. USE_OIL
8. USE_LIZARD
9. USE_TWEET
10. USE_EMP
11. FIX

Perintah kemudian ditampilkan ke konsol dengan format C;{nomor_ronde};{perintah}. Jika bot mengeluarkan error atau perintah di luar perintah di atas, bot akan diberikan hukuman berupa pengurangan skor dan dianggap melakukan "NOTHING". Pengurangan skor juga bisa terjadi jika mengeluarkan perintah dengan kondisi tertentu, antara lain:

1. Menggunakan power up ketika tidak memiliki power up tersebut.
2. Belok kanan ketika berada di lane paling kanan.
3. Belok kiri ketika berada di lane paling kiri.

Di setiap ronde, akan terdapat file Game State yang akan digunakan bot untuk memutuskan gerakan. File Game State terdiri dari detail mobil sendiri, detail mobil lawan, dan map. Perintah yang diterima dari kedua bot dilaksanakan secara simultan di setiap ronde.

2.2.2. Implementasi Algoritma Greedy ke Dalam Bot

Algoritma greedy merupakan algoritma yang melakukan pengambilan optimum lokal dengan harapan mendapatkan optimum global. Di permainan ini, dapat diibaratkan dengan melakukan perintah paling optimal di setiap ronde dengan harapan melakukan perintah optimal di game tersebut. Info dari ronde disimpan di file Game State. File Game State terdiri atas data mobil sendiri, mobil lawan, dan map. Dengan data dari file game state tersebut, bot akan memilih langkah paling optimal sesuai dengan tujuan programmer. Langkah paling optimal tersebut kemudian dikeluarkan ke konsol. Output tersebut akan digunakan game sebagai perintah bot.

2.2.3. Deskripsi Sistem *Game Engine*

Program menggunakan *game engine* bahasa Scala dengan basis Java yang dibuat dengan menggunakan IntelliJ Idea. Tema permainan dari program ini yaitu balapan mobil. Selain menggunakan java sebagai *game engine,* program juga menyediakan *game runner* untuk mempersiapkan sirkuit balapan dan peraturan terhadap *bot* sesuai spesifikasi dan aturan yang berlaku. Adanya *game runner* berfungsi sebagai fasilitator permainan antara kedua *bot* mobil dengan bertukar informasi antara kedua *bot* mobil dengan *game engine*. Saat awal permainan

game runner akan memberikan informasi dari *game engine* seperti kondisi *lane* dan kondisi *terrain* pada *block* kepada kedua bot mobil. Sebagai gantinya, bot mobil akan memproses informasi yang diterima dan memberikan informasi kepada *game runner* seperti perintah yang dilakukan oleh bot, kecepatan mobil bot saat itu, dan posisi mobil bot. Informasi ini akan diberikan kepada *game engine* sehingga *game engine* bisa mengeksekusi perintah dari *bot*. Setelah itu, *game engine* memberikan informasi yang baru kepada *game runner* dan seterusnya hingga permainan selesai.

2.2.4. Cara Menjalankan Game Engine

Proses menjalankan *game engine* dilakukan dengan mengklik run.bat pada sistem operasi Windows. Informasi yang ada pada *game engine* berada di file game-config.json dan game-runner-config.json. File game-config.json berisi jumlah baris atau *lane* pada lintasan, panjang lintasan, maksimal ronde yang ada, jumlah *score* dan *health* saat awal permainan, *command-command* yang bisa dipakai, *state-state* yang bisa dialami oleh *bot*, berapa peluang kemungkinan tiap *block* muncul, dan informasi dari *power ups* dan rintangan yang ada. Sedangkan file game-runner-config.json berisi lokasi hasil pertandingan, lokasi *game engine*, lokasi kedua *bot* mobil, dan pengaturan buat turnamen. Kedua file *game* ini dapat diubah sesuai keinginan pengguna. Namun untuk spesifikasi tugas besar, parameter yang bisa diubah yaitu :

1. round-state-output-location : Untuk menentukan direktori hasil dari pertandingan yang sudah disimulasikan oleh game engine, berguna jika menggunakan visualizer dengan memindahkan direktori pada visualizer
2. player-a : Untuk menentukan lokasi direktori *bot* pertama
3. player-b : untuk menentukan lokasi direktori *bot* kedua.

Simulasi permainan berjalan ketika file run.bat dijalankan. Setelah dijalankan akan muncul window terminal yang akan menunjukkan hasil simulasi tiap ronde dengan mengeluarkan *map* yang berisi lokasi kedua *bot* mobil dan rintangan yang terlihat saat itu. Selain itu hasil simulasi juga menunjukkan keadaan dua mobil seperti kecepatan mobil dan *command* yang diambil oleh bot. Pemain dapat melihat isi terminal untuk menganalisis permainan berdasarkan keluaran yang ada atau juga bisa menggunakan *visualizer* dari *game overdrive* pada tautan berikut : https://entelect-replay.raezor.co.za/. Cara menggunakan visualizer tersebut cukup dengan men-*zip* file hasil permainan pada direktori *match-log* kemudian mengunggah file .zip ke tautan tersebut. Visualizer tidak lama kemudian akan menampilkan hasil balapan tiap ronde yang lebih enak untuk dilihat dan lebih jelas apa yang terjadi dalam ronde tersebut.

BAB 3
APLIKASI STRATEGI GREEDY

3.1 Mapping Elemen Greedy

Berikut merupakan pemetaan algoritma greedy pada permainan Overdrive.

1. Himpunan kandidat : *Command-command* yang diberikan oleh pemain
   Command-command yang diberikan oleh pemain ke *bot* mobil antara lain yaitu maju (accelerate), melambat (deccelerate), belok kiri, belok kanan, memakai *boost*, melompat (memakai *lizard*), menembak EMP, membocorkan oli, men*tweet cybertruck*, memperbaiki mobil, dan tidak melakukan apa apa (*nothing*).
2. Himpunan solusi : urutan *command* yang diberikan ke *bot* mobil
3. Fungsi solusi: memeriksa apakah mobil menang atau kalah
4. Fungsi seleksi : berbagai alternatif solusi *greedy* yang berada di bagian 3.2
5. Fungsi kelayakan : memeriksa apakah *command* yang dipilih oleh *bot* valid atau tidak
   Command yang dipilih oleh *bot* bisa jadi tidak valid. *Command* tidak valid yaitu: Membelokkan mobil ke kiri ketika mobil berada di jalur 1, Membelokkan mobil ke kanan ketika mobil berada di jalur 4 dan menggunakan *power ups* ketika tidak memiliki *power ups*. Apabila *bot* memberikan *command* yang tidak valid ke *game engine*, mobil tidak akan melakukan apa apa sehingga sebaiknya *command* yang akan dipilih diperiksa terlebih dahulu kevalidannya.
6. Fungsi objektif : Mobil menang

3.2 Berbagai Alternatif Solusi Greedy

Permainan Overdrive akan dimenangkan oleh mobil yang mencapai garis finish terlebih dahulu. Apabila kedua mobil melewati garis finish secara bersamaan, mobil dengan kecepatan lebih tinggi yang akan memenangkannya. Apabila kecepatannya juga sama, mobil yang memiliki skor lebih tinggi yang akan menjadi pemenangnya.

Berdasarkan aturan tersebut, alternatif yang dapat menjadi solusi dari permainan ini adalah sebagai berikut.

1. Memaksimalkan kecepatan

   Tujuan dari permainan Overdrive yaitu agar mobil kita memenangkan perlombaan. Salah satu caranya yaitu dengan membuat mobil memiliki kecepatan semaksimum mungkin dengan harapan mobil musuh akan tertinggal. Idealnya kecepatan maksimum dapat dicapai ketika mobil berhasil mencapai *max speed* yang bernilai 9 kecepatan tanpa menabrak rintangan apapun. Selain itu kecepatan maksimum juga bisa mencapai 15 satuan kecepatan jika mobil menggunakan *power ups boost*.

   Berdasarkan kondisi ideal yang ingin dicapai. *Bot* mobil dibuat untuk menghindari segala rintangan yang ada. Jika ada rintangan di depan jalur mobil saat itu maka mobil akan melihat di jalur kiri dan kanan apakah ada rintangan atau tidak. Jika di jalur kiri ada rintangan sedangkan di jalur kanan tidak maka mobil akan belok ke kanan dan begitu pula sebaliknya jika di jalur kanan ada rintangan sedangkan di jalur kiri tidak maka mobil akan belok ke kiri. Jika tidak ada rintangan di kanan kiri maka mobil akan memutuskan secara acak untuk belok kanan atau kiri. Namun, jika di jalur kiri dan kanan terdapat rintangan maka mobil akan mengecek terlebih dahulu apakah mobil memiliki *power ups lizard*. Jika mobil memiliki *power ups lizard* maka akan digunakan untuk melompati rintangan yang ada. Jika tidak, mobil akan menghitung satu persatu

jumlah rintangan di antara jalur kiri, kanan, dan tengah lalu memilih jalur mana yang memiliki rintangan paling sedikit.

Selain menghindari rintangan, mobil berusaha untuk mendapatkan *power ups boost* dan *lizard*. *Power ups boost* akan digunakan ketika jalur yang dilewati tidak ada hambatan sama sekali agar bisa optimal. Sedangkan *power ups lizard* akan digunakan ketika mobil tidak bisa kemana-mana tanpa menabrak rintangan.

2. Memaksimalkan penyerangan mobil musuh

   Dalam aturan permainan Overdrive, mobil yang bertabrakan dengan objek atau penghalang yang berada di map akan terkena damage yang besarnya bergantung pada objek apa yang telah ditabrak. Objek yang ditabrak dapat berupa mud, oil spill, wall, dan cybertruck. Dua dari empat objek tersebut merupakan objek yang dapat dikeluarkan dengan menggunakan power up, yaitu oil spill apabila menggunakan power up oil dan cybertruck apabila menggunakan power up tweet. Mobil yang terkena sejumlah damage akan berkurang nilai kecepatan maksimumnya. Agar kecepatan maksimum mobil dapat bertambah kembali, mobil perlu menggunakan command fix yang membuat mobil tidak bergerak satu round. Selain penggunaan tweet dan oil, mobil juga dapat membuat kecepatan musuh menjadi tiga dengan menggunakan power up EMP. Setiap kali mobil berada di belakang musuh dan lane yang sama, mobil akan mengeluarkan EMP jika mempunyai power up tersebut. Dengan memaksimalkan penggunaan power up oil. tweet, dan EMP, peluang mobil musuh untuk terkena damage semakin besar dan kecepatannya menjadi berkurang sehingga diharapkan performa mobil musuh menjadi turun dan mobil pengguna dapat memanfaatkan kesempatan tersebut untuk sampai ke garis finish terlebih dahulu.

   Agar penyerangan terhadap mobil musuh menjadi maksimal, mobil perlu untuk mencari lane yang memiliki potensial damage terkecil terlebih dahulu karena apabila mobil terkena damage, performa mobil akan berkurang sehingga penggunaan power up menjadi kurang optimal. Apabila potensial damage tiap lane sama, mobil akan mencari lane dengan jumlah power up terbanyak. Hal ini bertujuan untuk mendapatkan power up sebanyak mungkin agar dapat melakukan banyak penyerangan kepada musuh. Mobil akan menggunakan power up oil, EMP, dan tweet apabila kondisi block di depan mobil sedang tidak ada rintangan dan tidak ada power up yang dapat diambil di samping kanan atau kiri. Tentu saja dengan disertai syarat penggunaan jenis power up, misalkan penggunaan EMP hanya ketika mobil berada di belakang mobil pada lane yang sama atau penggunaan OIL apabila posisi block mobil berada di depan mobil musuh. Begitu power up untuk menyerang mobil musuh telah habis, mobil akan menggunakan power up boost jika ada dan accelerate jika tidak ada power up apa pun lagi.

3. Memaksimalkan skor

   Permainan Overdrive memiliki sistem skor. Skor digunakan untuk memutuskan pemenang ketika kedua mobil sampai di finish bersamaan dengan kecepatan yang sama. Skor pemain bertambah ketika mendapatkan atau menggunakan power up dan berkurang ketika bertabrakan dengan obstacle atau memberi input yang invalid. Hal ini menyebabkan mungkinnya pemain memiliki skor negatif.

   Algoritma pertama-tama mengambil blok sebanyak speed mobil di ronde itu. Pengambilan ini dilakukan tiga kali, yaitu untuk lane kiri, lane depan, dan lane kanan. Untuk lane depan, banyak blok yang diambil setara dengan speed yang dimiliki mobil dan untuk lane kanan

atau kiri, banyak blok yang diambil adalah speed mobil dikurangi satu. Kemudian untuk tiap kemungkinan lane akan dihitung potensial skor dan potensial damage yang akan didapatkan ketika melewati lane tersebut. Perubahan potensial skor dan damage mengikuti aturan berikut:

- Blok Power Up: Skor +8, Damage +0
- Blok Oil Spill: Skor -4, Damage +1
- Blok Mud: Skor -3, Damage +1
- Blok Wall : Skor -0, Damage -2

Kemudian lane yang bisa dilewati tanpa rusak (damage > 4) dengan skor tertinggi akan dipilih sebagai lane yang akan dilewati. Jika lane tersebut lane kiri atau kanan, akan me-return perintah belok kiri atau kanan. Jika lane optimum merupakan lane tengah, jika mobil memiliki power up, mobil akan menggunakan power up. Selain itu, jika mobil memiliki damage lebih dari nol, akan mereturn perintah fix. Di luar itu, mobil akan mengeluarkan perintah accelerate.

3.3 Analisis Efisiensi Alternatif Greedy

Berdasarkan ketiga alternatif solusi greedy solusi yang paling efisien adalah solusi nomor 3 diikuti nomor 1 dan terakhir nomor 2. Alternatif nomor 3 paling efisien karena hanya menghitung block yang tersedia di samping kiri, kanan, dan depan tanpa perlu mengecek *power ups* yang dimiliki. Hal ini disebabkan oleh algoritma nomor 3 berusaha mendapatkan *score* sebanyak-banyaknya dengan cara mengambil *power ups* dan mengaktifkannya. Selanjutnya yaitu alternatif nomor 1 yang memiliki efisiensi sedikit lebih lambat. Algoritma nomor 1 berusaha untuk mencapai kecepatan semaksimal mungkin dengan menghindari rintangan dan menggunakan *boost* segera setelah mendapatkan *power ups* tersebut. Algoritma nomor 1 tidak memperdulikan lawan sehingga menghemat perhitungan. Sedangkan, algoritma nomor 2 paling tidak efisien karena tidak hanya mengecek rintangan yang ada di depan mobil juga mengecek jalur di belakang mobil untuk melihat lawan. Sehingga algoritma memiliki dua iterasi yang berbeda yaitu depan dan belakang. Pengecekan lawan berguna untuk menggunakan *power ups* yang dimiliki mobil seperti *emp* atau *oil spill*.

Secara perhitungan kompleksitas notasi big O, ketiga algoritma memiliki kompleksitas dari awal hingga akhir permainan sebesar O(n^2) dengan n adalah jumlah *blocks* lintasan. Tiap rondenya ketiga algoritma memiliki kompleksitas big O sebesar O(n). Namun tiap algoritma memiliki kompleksitas yang berbeda tiap rondenya jika tidak diukur dengan big O. Algoritma pertama memiliki kompleksitas sebesar (6n + 1p) dengan n adalah jalur lintasan didepan mobil dan p adalah *power ups*. Algoritma kedua memiliki kompleksitas sebesar (10n + 3p) dan algoritma ketiga memiliki kompleksitas sebesar (5n). Kompleksitas diatas tidak menghitungkan perbandingan *if else* pada setiap algoritma walaupun secara komputasi perbandingan *if else* dan komputasi lainnya tidak terlalu mempengaruhi efisiensi strategi.

3.4 Analisis Efektifitas Alternatif Greedy

Ketiga alternatif solusi memiliki keunikan masing-masing dan setiap alternatif memiliki kelebihan dan kelemahannya masing-masing. Untuk mendapatkan solusi yang paling efektif dan menilai efektivitas tiap solusi. Kelompok penulis mengimplementasikan semua alternatif solusi dengan cara mempertandingkan tiap pasangan alternatif solusi dan menghitung rasio menang setiap alternatif solusi. Semua pertandingan dilakukan dengan setelan *default game* dengan jumlah ronde maksimum 600 dan jumlah lintasan sebesar 1500 *blocks* dan setiap pasang alternatif solusi dipertandingkan sebanyak sepuluh kali.

Tabel 3.4.1. Rasio Menang terhadap Jumlah Pertandingan

pemain\lawan	Alternatif 1	Alternatif 2	Alternatif 3
Alternatif 1	-	0 %	0%
Alternatif 2	100 %	-	100%
Alternatif 3	100 %	0%	-

Berdasarkan tabel diatas, dapat disimpulkan bahwa urutan alternatif solusi *greedy* yang paling efektif adalah nomor 2, nomor 3, lalu nomor 1.

3.5 Strategi Greedy yang Dipilih

Berdasarkan tabel 3.4.1, kelompok penulis memilih alternatif nomor 2 yaitu memaksimalkan penyerangan mobil musuh sebagai strategi *greedy* di tugas besar Overdrive ini. Strategi nomor 2 dipilih karena memiliki ratio menang paling tinggi di antara strategi lainnya.

Strategi ini memiliki detail tambahan, yaitu :

1. Mengecek lokasi pendaratan mobil jika menggunakan *power ups lizard*, *power ups lizard* digunakan ketika lokasi pendaratan mobil tidak ada rintangan jika di depan mobil hanya ada satu rintangan.
2. Mengecek penggunaan *power ups* EMP, *power ups* EMP digunakan ketika mobil musuh berada di depan bot mobil dan berada pada lane yang terjangkau oleh EMP.
3. Mengecek penggunaan *power ups Tweet, power ups Tweet* digunakan pada tiga kondisi, yaitu
    - Mobil berada minimal satu block di depan musuh
    - Apabila berada di belakang musuh, mobil tidak berada di lane yang sama dengan mobil musuh
    - Apabila berada di belakang musuh dan di lane yang sama dengan musuh, jarak mobil harus terpisah minimal 15 block di belakang musuh
4. Mobil melakukan boosting apabila mobil tidak memiliki damage
5. Apabila mobil tidak berada dalam kecepatan maksimumnya, sebelum melakukan accelerate, mobil akan memeriksa obstacle untuk speed selanjutnya. Apabila pada speed selanjutnya, ternyata mobil malah menabrak obstacle, mobil akan mengeluarkan command *nothing*

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi program dilakukan pada file Bot.java dengan memodifikasi sedikit Terrain.java, PowerUps.java, Lane.java, dan State.java. Selain itu implementasi program dilakukan dengan menambahkan file TweetCommand.java, OilCommand.java dan EmpCommand.java. Method-method yang dimiliki oleh kelas *Bot* yaitu :

- run()
- isMaxSpeed()
- nextSpeed()
- maxSpeed()
- isNgekor()
- hasPowerUp(
- turn_random()
- isInFront()
- isInSameLane()
- isInSameBlock()
- isReachable()
- getBlocksIn(String position, int lane, int block)
- getPotentialDamage(List<Object> listTerrain)
- getPotentialPowerUps(List<Object> listTerrain)
- cekLizard(List<Object> listTerrain, int damage)

Method yang akan diberi pseudocode adalah *method* yang sulit untuk dipahami maknanya.

procedure run()

Kamus

opponent, mycar : Car
Random : Random
GameState : gameState
FIX, OIL, BOOST, EMP, LIZARD, TURN_RIGHT, TURN_LEFT : Command
getPotentialDamageFront, potentialPowerUpsFront, potentialDamageRight,
potentialDamageLeft, potentialPowerUpsRight, potentialPowerUpsLeft,
potentialDamageAtNextSpeed, potentialDamageAtMaxSpeed : int
blocksInRight, blocksInLeft, blocksAtMaxSpeed : List<Object>

Algoritma

```
blocksInFront ← getBlocksIn("front", myCar.position.lane, myCar.position.block)
potentialDamageFront ← getPotentialDamage(blocksInFront)
potentialPowerUpsFront ← getPotentialPowerUps(blocksInFront)
if (kerusakan mobil lebih dari 1)
        return FIX {memperbaiki mobil}
if(kecepatan mobil nol)
        return AccelerateCommand() {mobil menambah kecepatan}
if (mobil berada di jalur 2 atau 3)
```

```
blocksInRight ← getBlocksIn("right", myCar.position.lane, myCar.position.block);
blocksInLeft← getBlocksIn("left", myCar.position.lane, myCar.position.block);
potentialDamageRight ← getPotentialDamage(blocksInRight);
potentialDamageLeft ← getPotentialDamage(blocksInLeft);
potentialPowerUpsRight ← getPotentialPowerUps(blocksInRight);
potentialPowerUpsLeft ← getPotentialPowerUps(blocksInLeft);
if (isNgekor()) {ada mobil musuh tepat di depan mobil}
        if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) = 0)
                return LIZARD {menggunakan powerups lizard}
        if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) != 0)
                return DecelerateCommand() {mobil melambat}
        if(di kiri dan kanan tidak ada rintangan)
                return turn_random() {belok kiri dan kanan secara random}
        if(di kiri tidak ada rintangan dan di kanan ada)
                return TURN_LEFT {mobil belok kiri}
        if(di kanan tidak ada rintangan dan di kiri ada)
                return TURN_RIGHT {mobil belok kanan}
if( kanan dan depan mobil ada rintangan dan di kiri tidak ada rintangan)
        return TURN_LEFT {mobil belok kiri}
if(kiri dan depan mobil ada rintangan dan di kanan tidak ada rintangan)
        return TURN_RIGHT {mobil belok kanan}
if(depan mobil ada rintangan dan di kanan kiri tidak ada rintangan)
        if(jalur kiri lebih banyak power ups dari jalur kanan)
                return TURN_LEFT {mobil belok kiri}
        if(jalur kiri dan jalur kanan memiliki jumlah power ups yang sama)
                return turn_random() {belok kiri dan kanan secara random}
        return TURN_LEFT {mobil belok kiri}
if( jalur depan, kiri dan kanan kosong tidak ada rintangan)
        if(jalur kiri lebih banyak power ups daripada jalur kanan dan depan)
                return TURN_LEFT {mobil belok kiri}
        if(jalur kanan lebih banyak power ups daripada jalur kiri dan depan)
                return TURN_RIGHT {mobil belok kanan}
        if(jalur kanan dan jalur kiri lebih banyak power ups daripada jalur depan)
                return turn_random() {belok kiri dan kanan secara random}
if( semua jalur ada rintangan)
        if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) = 0)
                return LIZARD {menggunakan powerups lizard}
        if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) != 0)
                return DecelerateCommand() {mobil melambat}
        if(jalur kiri lebih sedikit rintangan daripada jalur kanan dan depan)
                return TURN_LEFT {mobil belok kiri}
        if(jalur kanan lebih sedikit rintangan daripada jalur kiri dan depan)
                return TURN_RIGHT {mobil belok kanan}
        if( jalur kiri, kanan memiliki jumlah rintangan lebih sedikit daripada jalur depan)
```

```
                        if(jalur kiri lebih banyak power ups daripada jalur kanan dan depan)
                                return TURN_LEFT {mobil belok kiri}
                        if(jalur kanan lebih banyak power ups daripada jalur kiri dan depan)
                                return TURN_RIGHT {mobil belok kanan}
                        if(jalur kanan dan jalur kiri lebih banyak power ups daripada jalur depan)
                                return turn_random() {belok kiri dan kanan secara random}

if(mobil berada di jalur 1)
        blocksInRight ← getBlocksIn("right", myCar.position.lane, myCar.position.block);
        potentialDamageRight ← getPotentialDamage(blocksInRight);
        potentialPowerUpsRight ← getPotentialPowerUps(blocksInRight);
        if(jalur kanan tidak ada rintangan dan di depan ada rintangan)
                return TURN_RIGHT {mobil belok kanan}
        if(jalur kanan dan depan ada rintangan atau mobil ngekor)
                if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) = 0)
                        return LIZARD {menggunakan powerups lizard}
                if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) != 0)
                        return DecelerateCommand() {mobil melambat}
                if( jalur kanan lebih sedikit rintangan daripada jalur depan)
                        return TURN_RIGHT {mobil belok kanan}
        if(jalur kanan dan depan tidak ada rintangan)
                if(jalur kanan lebih banyak powerups daripada jalur depan)
                        return TURN_RIGHT {mobil belok kanan}
if(mobil berada di jalur 4)
        blocksInLeft ← getBlocksIn("left", myCar.position.lane, myCar.position.block);
        potentialDamageLeft ← getPotentialDamage(blocksInRight);
        potentialPowerUpsLeft ← getPotentialPowerUps(blocksInRight);
        if(jalur kiri tidak ada rintangan dan di depan ada rintangan)
                return TURN_LEFT {mobil belok kiri}
        if(jalur kiri dan depan ada rintangan atau mobil ngekor)
                if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) = 0)
                        return LIZARD {menggunakan powerups lizard}
                if(ada powerups lizard dan cekLizard(blocksInFront,potentialDamageFront) != 0)
                        return DecelerateCommand() {mobil melambat}
                if( jalur kiri lebih sedikit rintangan daripada jalur depan)
                        return TURN_LEFT {mobil belok kiri}
        if(jalur kiri dan depan tidak ada rintangan)
                if(jalur kiri lebih banyak powerups daripada jalur depan)
                        return TURN_LEFT {mobil belok kiri}
if (mobil ada power ups emp dan !isInFront() && !isInSameBlock() && isReachable())
{di depan mobil ada mobil musuh dan mobil musuh ada di jalur depan kiri kanan}
        return EMP {menggunakan power ups EMP}
if (mobil ada power ups oil dan mobil berada di depan mobil musuh)
        return OIL {menggunakan power ups OIL}
```

```
if (mobil ada power ups tweet)
        if (posisi mobil dan mobil musuh lebih dari 1 blocks dan mobil musuh berada di depan mobil dan mobil musuh tidak satu jalur dengan mobil)
                return TweetCommand(opponent.position.lane, opponent.position.block + opponent.speed + 1)
    blocksAtNextSpeed ← getBlocksIn("nextSpeed", myCar.position.lane, myCar.position.block)
    int potentialDamageAtNextSpeed ← getPotentialDamage(blocksAtNextSpeed)
    if (mobil ada power ups boost)
        blocksAtMaxSpeed = getBlocksIn("maxSpeed", myCar.position.lane, myCar.position.block)
        potentialDamageAtMaxSpeed = getPotentialDamage(blocksAtMaxSpeed)
        if ( ada potensi kerusakan mobil ketika menggunakan boost)
            if(mobil belum kecepatan maksimum dan tidak ada potensi kerusakan ketika melaju)
                    return AccelerateCommand() {mobil menambah kecepatan}
            return DoNothingCommand() {mobil tidak ada perintah baru}
         if(kerusakan mobil lebih dari nol)
            return FIX {memperbaiki mobil}
        if(mobil saat ini tidak memakai boost)
            return BOOST {menggunakan power ups boost}
    if( mobil belum kecepatan maksimum dan ada kerusakan potensial mobil jika melaju)
        return DoNothingCommand() {mobil tidak ada perintah baru}
    return AccelerateCommand() {mobil menambah kecepatan}
```

Bot mobil penulis memiliki skala prioritas dalam mengeksekusi dan memilih perintah, yaitu menghindari mobil musuh > menghindari rintangan > mendapatkan power ups > menggunakan power ups> menambah kecepatan mobil > mengurangi kecepatan mobil > do nothing. Pseudocode diatas menunjukkan skala prioritas dari bot mobil penulis.

Method dari getBlocksIn bertujuan untuk memasukkan blocks di depan mobil dalam list object. Pseudocode dari getBlocksIn(String position, int lane, int block) sebagai berikut :

List<Object> getBlocksIn(String position, int lane, int block)

Kamus :

```
Map, laneList : List<Lane[]>
Blocks : List<Object>
startBlock, lowerBound, upperBound : int
```

Algoritma :

```
int startBlock ← map.get(0)[0].position.block
{untuk di depan dari posisi mobil, block dimana mobil berada tidak ikut diperhitungkan}
int lowerBound ← max(block + 1 - startBlock, 0)
int upperBound ← block - startBlock + myCar.speed
if(nilai position adalah “right”) {mengecek jalur di kanan mobil}
    lane  ← lane +1
    lowerBound  ← max(block - startBlock,0)
    upperBound  ← block - startBlock + myCar.speed -1
```

```
else if (nilai position adalah “left”) {mengecek jalur di kiri mobil}
        lane ← lane -1
        lowerBound ← max(block - startBlock,0)
        upperBound ← block - startBlock + myCar.speed -1
else if (nilai position adalah “nextSpeed”) {mengecek jalur di depan mobil jika mobil melaju}
        upperBound ← block - startBlock + nextSpeed
else if (nilai position adalah “maxSpeed”) {mengecek jalur di depan mobil jika mobil menggunakan boost}
        upperBound ← block - startBlock + maxSpeed
laneList ← map.get(lane - 1);
for ( iterasi dengan i awal adalah lowerBound hingga i sama dengan upperBound)
        if(laneList kosong atau sudah mencapai garis finish)
                Break
        if(laneList[i] diisi oleh CyberTruck)
                blocks.add(“Cybertruck”)
        else
                blocks.add(laneList[i].terrain)
return blocks
```

Method dari getPotentialDamage bertujuan menghitung jumlah damage yang akan diterima mobil jika melewati jalur tersebut. Pseudocode dari getPotentialDamage(List<Object> listTerrain) sebagai berikut :

Kamus :

```
damage : int
```

Algoritma:

```
damage ← 0
For (iterasi i dari 0 hingga listTerrain.size()-1)
        if(listTerrain sama dengan mud atau oil spill)
                damage ← damage+1
        else if (listTerrain sama dengan wall atau cybertruck)
                damage ← damage+2
return damage
```

Method dari getPotentialPowerups bertujuan menghitung jumlah powerups yang akan diterima mobil jika melewati jalur tersebut. Pseudocode dari getPotentialPowerUps(List<Object> listTerrain) sebagai berikut :

Kamus :

```
countPowerUp : int
```

Algoritma:

```
countPowerUp ← 0
For (iterasi i dari 0 hingga listTerrain.size()-1)
        if(listTerrain sama dengan oil power, boost, emp, tweet atau lizard)
                countPowerUp ← countPowerUp+1
return countPowerUp
```

Method dari getLizard bertujuan untuk mengecek apakah tempat mendarat mobil jika memakai lizard terdapat rintangan atau tidak. Pseudocode dari getPotentialDamage(List<Object> listTerrain) sebagai berikut :

Kamus :

```
damage,i : int
```

Algoritma:

```
damage ← 0
if(listTerrain[i] sama dengan mud atau oil spill)
        damage ← damage-1
else if (listTerrain[i] sama dengan wall atau cybertruck)
        damage ← damage-2
return damage
```

Untuk method lainnya, hanya akan ditulis spesifikasinya. Detail spesifikasi tiap method lainnya dapat dilihat di tabel di bawah ini:

Tabel 4.1.1 Nama method dan spesifikasinya

Method	Spesifikasi
isMaxSpeed()	mengembalikan true apabila mobil berada di maxSpeed
nextSpeed()	mengembalikan speed selanjutnya apabila di-accelerate
maxSpeed()	mengembalikan nilai maksimal speed apabila mobil sedang berada di maxSpeed
isNgekor()	mengembalikan true apabila mobil tepat berada satu block di belakang musuh
hasPowerUp(PowerUps powerUpToCheck)	Menerima objek PowerUps dan mengembalikan true apabila memiliki power up yang diinginkan
turn_random()	Mengembalikan arah belok random
isInFront()	Mengembalikan true apabila mobil berada di depan musuh
isInSameLane()	Mengembalikan true apabila mobil berada di lane yang sama
isInSameBlock()	Mengembalikan true apabila mobil berada di block yang sama
isReachable()	Mengembalikan true apabila mobil musuh berada

	di jangkauan emp

4.2 Struktur Data

Di luar struktur data primitif, implementasi bot juga memanfaatkan beberapa struktur data buatan, antara lain:

a. Class GameState
b. Class Car
c. Class Bot

4.2.1 Class GameState

Class GameState menyimpan status game di sebuah ronde. Pada implementasi bot, class GameState diinisialisasikan dengan memparse file state.json di folder tiap ronde. Class GameState memiliki beberapa atribut, antara lain:

a. currentRound. Atribut ini bertipe int dan merepresentasikan nomor ronde game state.
b. maxRounds. Atribut ini bertipe int dan merepresentasikan jumlah maksimum ronde game.
c. player. Atribut ini bertipe Car dan menyimpan detail mobil seperti lokasi, kecepatan, dan powerups yang dimiliki oleh mobil pihak kita.
d. opponent. Atribut ini bertipe Car dan menyimpan detail mobil seperti lokasi, kecepatan, dan powerups yang dimiliki oleh mobil pihak lawan.
e. lanes. Atribut ini bertipe List<Lane[]> atau list dari lanes. Atribut ini berukuran empat dan menyimpan keempat lane di map.

4.2.2 Class Car

Class Car menyimpan info mobil di game. Class ini memiliki delapan atribut, yaitu:

a. id. Atribut ini bertipe int dan merepresentasikan id player (1 atau 2).
b. position. Atribut ini bertipe Position dan menyimpan koordinat block dan lane (x dan y) mobil.
c. speed. Atribut ini bertipe int dan menunjukkan besar laju mobil.
d. state. Atribut ini bertipe State dan merepresentasikan status mobil (cth. READY, HIT_OIL, dll.)
e. damage. Atribut ini bertipe int dan menunjukkan damage atau tingkat kerusakan mobil.
f. powerups. Atribut ini bertipe array of Powerup, yang menyimpan semua power up yang dimiliki mobil.
g. boosting. Atribut ini bertipe boolean dan menunjukkan apakah mobil dalam efek power up boost atau tidak.
h. boostCounter. Atribut ini bertipe int dan menunjukkan banyak ronde yang telah dilewati dengan boost.

4.2.3 Class Bot

Class Bot merupakan kelas utama dari bot algoritma greedy. Kelas Bot terdiri dari atribut-atribut dan metode-metode yang digunakan di algortima greedy. Atribut dari kelas Bot antara lain:

a. directionList. Atribut ini bertipe list of integer berukuran 2. Atribut ini berisi arah yang bisa diambil mobil. Elemen -1 menandakan arah kiri dan elemen 1 menandakan arah kanan.
b. gameState. Atribut ini bertipe GameState dan berisi state game di suatu ronde.
c. opponent.. Atribut ini bertipe Car dan merupakan representasi dari mobil lawan.
d. myCar. Atribut ini bertipe Car dan merupakan representasi dari mobil pihak kita.
e. FIX. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk memperbaiki mobil
f. OIL. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk menggunakan power up Oil.
g. BOOST. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk menggunakan power up Boost.
h. EMP. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk menggunakan power up EMP.
i. LIZARD. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk menggunakan power up Lizard.
j. TURN_RIGHT. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk belok ke kanan.
k. TURN_LEFT. Atribut ini bertipe Command. Atribut ini menyimpan perintah untuk belok ke kiri.

4.3 Pengujian

Dalam permainan Overdrive fungsi objektif yang ada yaitu memenangkan permainan. Karena hanya ada *reference Bot* dan bot-bot dari kelompok penulis maka masih banyak perbedaan pergerakan dan faktor lainnya yang menyebabkan hasil yang berbeda. Oleh karena itu, proses analisisnya menjadi lebih sulit karena banyak faktor yang berbeda dan tidak dapat diprediksi.

Bot yang dipilih oleh kelompok penulis memiliki ratio menang sebesar 100 % jika dibandingkan dengan *bot* lainnya yang dibuat kelompok penulis. Hal tersebut menunjukkan bahwa algoritma *greedy* yang digunakan bisa mendapatkan nilai optimal pada sebagian besar kasus seperti menghindar rintangan, menggunakan *power ups*, dan mencari *power ups*. Berdasarkan pengujian yang ada, *bot* mobil cukup ahli dalam berbagai macam kasus. Walaupun begitu bisa saja *bot* mobil belum memadai dalam menghadapi kasus-kasus yang tidak diduga oleh *bot* mobil dari kelompok lainnya.

BAB 5
SIMPULAN DAN SARAN

5.1 Simpulan

Algoritma Greedy merupakan algoritma yang menyelesaikan permasalahan secara langkah per langkah dengan setiap langkahnya berusaha untuk mendapatkan nilai optimum lokal dari permasalahan tersebut. Nilai optimum ini dapat berupa paling kecil, paling besar, paling banyak, paling sedikit, dll. Walaupun begitu, solusi yang didapatkan dari algoritma *greedy* tidak selalu memberikan solusi yang optimum secara global. Solusi optimum yang diperoleh dari algoritma *greedy* bergantung pada permasalahan yang ingin diselesaikan dan pendekatan heuristik yang digunakan. Namun, algoritma *greedy* mampu memberikan solusi optimum dengan waktu yang relatif lebih cepat daripada algoritma *brute force* dan umumnya solusi yang ada mendekati solusi optimum global.

Pada tugas besar ini, pendekatan heuristik dilakukan guna mencapai kemenangan dengan cara melewati garis *finish* terlebih dahulu sebelum musuh. Tujuan sekunder yaitu untuk mendapatkan kecepatan semaksimal mungkin saat melewati musuh, dan tujuan tersier yaitu mendapatkan skor setinggi-tingginya jika mobil melewati garis *finish* bersamaan dengan mobil musuh. Bot mobil memiliki prioritas perintah dengan mempertimbangkan lokasi mobil musuh dan rintangan yang dihadapi bot. Prioritas bot mobil adalah menghindari rintangan, diikuti dengan mencari *power ups,* lalu menggunakan *power ups* jika kondisi memungkinkan, memperlambat mobil musuh, menambah kecepatan mobil, dan terakhir adalah *do nothing*.

5.2 Saran

1. Untuk kelompok penulis, dibutuhkan kesabaran untuk menghadapi kemungkinan yang berbeda
2. Untuk pemberi tugas, pengambilan materi tugas besar dari *entelect challenge* adalah ide yang baik, namun kedepannya bisa memilih *game* yang memiliki *visualizer* yang *up to date*, mudah dilihat dan dimengerti.
3. Untuk seluruh pembaca bisa melakukan pengujian lebih lanjut dengan kemungkinan yang berbeda untuk mengetes keefektifan strategi dari kelompok penulis.

REFERENSI

Munir, Rinaldi. (2022). *Algoritma Greedy (Bagian 1)*. Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

Munir, Rinaldi. (2022). *Algoritma Greedy (Bagian 2)*. Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

Munir, Rinaldi. (2022). *Algoritma Greedy (Bagian 3)*. Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf

LAMPIRAN

Link video demo : https://www.youtube.com/watch?v=_GO7-cDeqdY

Kode sumber implementasi setiap strategi dapat dilihat pada tautan berikut :

1. Strategi utama (menghancurkan mobil musuh) : https://github.com/malifpy/stima-bot-overdrive/tree/main
2. Strategi memaksimalkan kecepatan : https://github.com/malifpy/stima-bot-overdrive/tree/maksimalkan-speed(fitrah)
3. Strategi memaksimalkan skor : https://github.com/malifpy/stima-bot-overdrive/tree/max-score(alif)