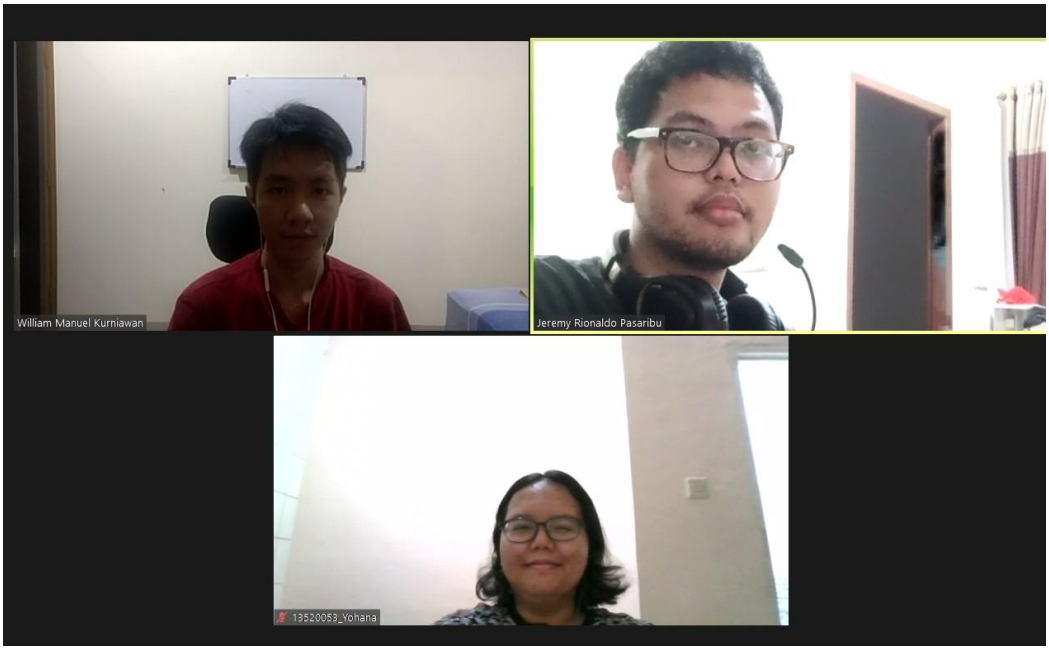
Laporan Tugas Besar 1 IF2211 Strategi Algoritma
Semester II tahun 2021/2022

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"



Dipersiapkan oleh:

Kelompok 40: Kar

13520020	William Manuel Kurniawan
13520053	Yohana Golkaria Nainggolan
13520082	Jeremy Rionaldo Pasaribu

Sekolah Teknik Elektro dan Informatika

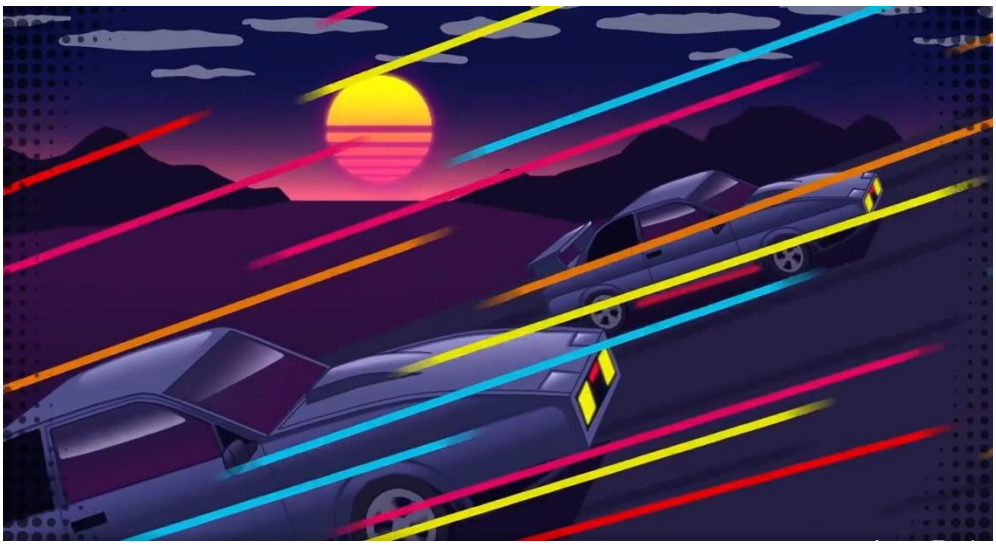
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

Daftar Isi

BAB 1: Deskripsi Masalah

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.



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 blok yang saling berurutan, panjang peta terdiri atas 1500 blok. Terdapat 5 tipe blok, yaitu Empty, Mud, Oil Spill, Flimsy Wall, dan Finish Line yang masing-masing karakteristik dan efek berbeda. Blok dapat memuat powerups yang bisa diambil oleh mobil yang melewati blok 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 blok 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 blok untuk setiap round. Game state akan memberikan jarak pandang hingga 20 blok di depan dan 5 blok 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
   i. USE_EMP
   j. FIX

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

2.1. Algoritma Greedy

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah tersebut akan berakhir dengan optimum global.

Berikut merupakan elemen-elemen algoritma *greedy*:

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

2.2. Permainan Overdrive

File yang diunduh untuk menjalankan permainan Overdrive adalah sebuah file .zip bernama starter-pack yang didapatkan pada link yang tertera pada halaman sebelumnya. Setelah file .zip diekstrak, akan dihasilkan folder starter-pack yang berisi game engine, konfigurasi permainan, serta contoh kode bot sederhana. Untuk menjalankan permainan, jalankan file Windows Batch File bernama run. Setelah file run dijalankan, terminal baru akan dibuka dan menjalankan permainan sampai selesai dan menghasilkan folder baru bernama match-logs yang berisi folder hasil dari permainan tersebut. Untuk metode visualisasi lain, folder hasil dari permainan dapat diubah menjadi file .zip dan dapat divisualisasikan menggunakan link https://entelect-replay.raezor.co.za/.

2.3. Pengaturan Permainan Overdrive

Permainan Overdrive diatur menggunakan file game-runner.json dan game-runner-config.json. File game-runner.json berisi peraturan permainan seperti jumlah ronde dan panjang lintasan yang sudah diatur sehingga tidak perlu diubah. File game-runner-config.json berisi pengaturan file yang digunakan seperti folder bot yang digunakan, nama file game engine, nama folder output hasil permainan, dan lain-lain. Berdasarkan pengaturan awal, game engine akan mengambil starter bot yang berada pada folder

javascript dan reference bot yang berada pada folder java. Oleh karena itu, game-runner-config.json harus diubah agar game engine menggunakan bot yang berada di folder java.

2.4. Bot Permainan Overdrive

Permainan Overdrive dapat menggunakan bot yang dibuat dari beberapa bahasa pemrograman yang berbeda tetapi diputuskan untuk dibuat menggunakan bahasa pemrograman Java. Dalam folder starter bot, diberikan file Bot.java dan Main.java yang digunakan sebagai contoh bot, beserta folder enums, folder entities, dan folder command untuk menyimpan sebagian jenis perintah, status, dan terrain. Berdasarkan pengaturan awal, starter bot pada folder java belum bisa digunakan karena harus melakukan build terlebih dahulu di IntelliJ untuk membuat file .jar yang akan digunakan game engine. Dari pengaturan awal, bot yang digunakan dalam folder reference bot menggunakan bahasa pemrograman Java dan dapat digunakan sebagai referensi jika bot pada folder starter bot ingin diubah.

2.5. Cara Kerja Bot

Dalam permainan overdrive, setiap pemain akan diberi kesempatan untuk melakukan 1 perintah setiap ronde di saat yang bersamaan. Ada beberapa perintah yang digunakan untuk mengubah pergerakan bot, menggunakan powerup, dan lain-lain. Jika perintah yang digunakan dianggap tidak sah oleh game engine, maka game engine akan menganggap bahwa pemain tidak memasukkan perintah apa pun saat ronde itu. Dalam file Bot.java, perintah akan dikirimkan ke game engine dari fungsi run yang akan menghasilkan atribut perintah yang disimpan dalam folder command. Untuk mengubah prioritas perintah dalam bot, fungsi run harus diubah sehingga menghasilkan perintah yang diinginkan. Fungsi lain dapat dibuat dalam file dan dimasukkan ke dalam fungsi run untuk membantu jalan program. File Main.java akan digunakan untuk membantu menjalankan file Bot.java dan tidak perlu diubah.

BAB 3: Aplikasi Strategi *Greedy*

3.1. Pemetaan Persoalan *Overdrive* Menjadi Elemen-Elemen Algoritma *Greedy*

3.1.1. Himpunan Kandidat

Perintah-perintah yang mungkin dijalankan serta *block* jalur dalam permainan *Overdrive*. Contoh perintah seperti *NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP,* dan *FIX*

3.1.2. Himpunan Solusi

Perintah-perintah serta *block* jalur dari himpunan kandidat yang terpilih sesuai dengan strategi yang digunakan untuk setiap ronde permainan selama permainan berlangsung.

3.1.3. Fungsi Solusi

Memeriksa apakah perintah yang dieksekusi terdefinisi atau tidak dalam permainan *Overdrive*. Jika tidak terdefinisi maka perintah tidak valid untuk dijalankan oleh *game engine*.

3.1.4. Fungsi Seleksi (*Selection Function*)

Memilih perintah terbaik sesuai dengan prioritas strategi yang digunakan *bot* pada setiap ronde selama permainan berlangsung.

3.1.5. Fungsi Kelayakan (*Feasibility*)

Memeriksa apakah perintah yang dieksekusi sesuai dengan kondisi strategi serta mengikuti peraturan dalam permainan *Overdrive*. Validasi dilakukan terhadap jumlah *power up*, jumlah blok, serta jalur mobil yang digunakan ketika ingin mengeksekusi suatu perintah.

3.1.6. Fungsi obyektif

Memaksimalkan kesempatan mobil untuk memenangkan permainan *Overdrive* baik dengan cara mencapai garis *finish* sebelum bot lawan maupun mencapai garis *finish* secara bersamaan dengan *bot* lawan tetapi memiliki skor tertinggi terhadap *bot* lawan.

3.2. Eksplorasi Alternatif Strategi *Greedy* dalam Permainan *Overdrive*

Berikut adalah beberapa alternatif strategi *greedy* yang digunakan *bot* untuk memenangkan permainan *Overdrive*:

3.2.1. Alternatif Strategi 1

Mobil hanya akan fokus dalam menghindari rintangan serta menambah kecepatan mobil tanpa menggunakan *PowerUp* untuk menghambat lawan. Mobil akan selalu berusaha mengeksekusi perintah *FIX* ketika mobil terkena *damage*. Jika mobil tidak memiliki *damage*, mobil akan berusaha untuk menghindari rintangan jika blok yang akan dilalui oleh mobil memiliki karakteristik untuk memperlambat atau menghalangi mobil seperti *Mud, Oil Spill* atau *Flimsy Wall*. Perintah yang akan dieksekusi untuk menghindari rintangan yaitu *TURN_LEFT*, *TURN_RIGHT*, *ACCELERATE*, atau *USE_LIZARD*. Jika tidak terdapat rintangan di depan mobil, mobil akan berusaha secara *greedy* mengeksekusi perintah *USE_BOOST* untuk menambah kecepatan mobil sesuai dengan kondisi yang diperlukan. Kondisi *USE_BOOST* akan terpenuhi ketika mobil memiliki kecepatan di bawah kecepatan *boost* (*speed* < 15) serta mobil memiliki *PowerUp BOOST*. Jika kondisi *USE_BOOST* tidak terpenuhi, maka mobil akan mengeksekusi perintah *ACCELERATE*.

3.2.2. Alternatif Strategi 2

Mobil akan berusaha mengeksekusi perintah *FIX* ketika mobil terkena *damage*. Jika mobil tidak memiliki *damage*, mobil akan berusaha untuk menghindari rintangan jika blok yang akan dilalui oleh mobil memiliki karakteristik untuk memperlambat atau menghalangi mobil seperti *Mud, Oil Spill* atau *Flimsy Wall*. Perintah yang digunakan untuk menghindari rintangan yaitu *TURN_LEFT*, *TURN_RIGHT*, *ACCELERATE*, atau *USE_LIZARD*. Jika tidak ada rintangan di depan mobil, mobil akan berusaha secara *greedy* mengeksekusi perintah *USE_BOOST* untuk menambah kecepatan mobil sesuai dengan kondisi yang diperlukan. Kondisi *USE_BOOST* akan terpenuhi ketika mobil memiliki kecepatan di bawah kecepatan *boost* (*speed* < 15) serta mobil memiliki *PowerUp BOOST*. Jika kondisi perintah *USE_BOOST* tidak terpenuhi, mobil akan berusaha menghambat pergerakan lawan sesuai dengan kondisi yang diperlukan. Kondisi akan terpenuhi jika mobil memiliki *PowerUp* untuk menghambat lawan seperti *EMP*, *OIL*, serta *TWEET*. Prioritas perintah yang digunakan untuk menghambat lawan yaitu *USE_EMP*, *USE_TWEET*, kemudian *USE_OIL* secara terurut. Jika semua kondisi untuk menghambat lawan tidak terpenuhi, mobil akan mengeksekusi perintah *ACCELERATE*.

3.2.3. Alternatif Strategi 3

Mobil akan berusaha untuk mengeksekusi perintah *FIX* ketika mobil memiliki *damage*. Jika mobil tidak memiliki *damage*, mobil akan mencoba untuk menghindari rintangan jika blok yang akan dilalui oleh mobil memiliki karakteristik untuk memperlambat atau menghalangi mobil seperti *Mud, Oil Spill* atau *Flimsy Wall*.

Perintah yang digunakan untuk menghindar yaitu *TURN_LEFT*, *TURN_RIGHT*, *ACCELERATE*, atau *USE_LIZARD*. Jika tidak terdapat rintangan di depan mobil, mobil akan berusaha mengeksekusi perintah secara *greedy* untuk menghambat pergerakan lawan sesuai dengan kondisi yang diperlukan. Kondisi terpenuhi jika mobil memiliki *PowerUp* untuk menghambat lawan seperti *EMP*, *OIL*, serta *TWEET*. Prioritas perintah yang digunakan untuk menghambat lawan yaitu *USE_EMP*, *USE_TWEET*, kemudian *USE_OIL* secara terurut. Jika kondisi untuk menghambat lawan tidak terpenuhi, mobil akan mengeksekusi perintah USE_BOOST sesuai dengan kondisi yang diperlukan. Kondisi *USE_BOOST* akan terpenuhi ketika mobil memiliki kecepatan di bawah kecepatan *boost* (*speed* < 15) serta mobil memiliki *PowerUp BOOST*. Jika kondisi *USE_BOOST* tidak terpenuhi, mobil akan mengeksekusi perintah ACCELERATE jika tidak terdapat rintangan di depan mobil. Jika terdapat rintangan di depan mobil, maka mobil akan mengeksekusi perintah *NOTHING*.

3.3. Efisiensi Strategi *Greedy*

3.3.1. Alternatif Strategi 1

Implementasi alternatif strategi ini lebih mudah dibandingkan alternatif strategi lainnya. Untuk memeriksa rintangan dalam suatu blok, program harus memeriksa jumlah blok tertentu di depan mobil dan menyimpan data pemeriksaan tersebut untuk digunakan. Pengecekan hanya dilakukan sekali sebanyak kecepatan mmobil (*myCar*.*speed*). Pengecekan tersebut memiliki kompleksitas algoritma sebesar O(n). Untuk menentukan jalur terbaik yang dilewati mobil, setiap jalur yang dapat dilewati akan diberi skor tertentu dan pemberian skor tersebut harus menelusuri blok di jalur tersebut sehingga menambah kompleksitas sebesar O(n) untuk jalur kiri, jalur kanan, serta jalur tengah mobil. Penambahan kompleksitas tersebut menyebabkan kompleksitas algoritma untuk pergerakan mobil menjadi $O(n^2)$. Penggunaan *PowerUp LIZARD* juga akan memiliki kompleksitas $O(n^2)$ karena menggunakan sistem skor yang sama.

Untuk penggunaan *PowerUp BOOST*, pengecekan blok di depan mobil memiliki kompleksitas algoritma yang sama sebesar O(n). *PowerUp EMP*, *TWEET*, serta *OIL_POWER* tidak digunakan dalam alternatif strategi 1 sehingga tidak menambah kompleksitas algoritma. Untuk perintah *FIX*, program akan berjalan berdasarkan kondisi tertentu sehingga memiliki kompleksitas algoritma O(1). Perintah *NOTHING* dan *DECELERATE* juga tidak digunakan sehingga tidak menambah kompleksitas algoritma. Kompleksitas akhir program untuk alternatif strategi 1 adalah $O(n^2)$ untuk setiap ronde dalam permainan *Overdrive*.

3.3.2. Alternatif Strategi 2

Implementasi alternatif strategi ini lebih kompleks daripada alternatif strategi 1. Untuk memeriksa rintangan dalam suatu blok, program harus memeriksa jumlah blok tertentu di depan mobil dan menyimpan data tersebut untuk digunakan seperti pada alternatif strategi 1. Oleh karena itu, pemeriksaan awal keberadaan rintangan pada jalur

mobil memiliki kompleksitas O(n). Pemeriksaan blok di depan mobil akan digunakan untuk menentukan pergerakan mobil antara perintah *TURN_LEFT*, *TURN_RIGHT*, atau *ACCELERATE*. Karena program ingin menentukan jalur terbaik yang dilalui mobil, maka jalur-jalur akan diberi skor tertentu dan pemberian skor tersebut harus menelusuri kembali blok yang akan dilewati oleh mobil sehingga menambah kompleksitas sebesar O(n). Penambahan kompleksitas tersebut menyebabkan kompleksitas algoritma untuk pergerakan mobil menjadi $O(n^2)$. Penggunaan powerup *LIZARD* juga akan memiliki kompleksitas $O(n^2)$ karena menggunakan sistem skor yang sama dengan pergerakan mobil.

Ketika menggunakan powerup *BOOST*, program akan memeriksa 15 blok ke depan pada jalur mobil untuk memastikan mobil tidak menabrak. Karena melakukkan pengecekan, powerup *BOOST* juga memiliki kompleksitas O(n). Penggunaan powerup *USE_TWEET*, *USE_OIL*, dan *USE_EMP* memiliki kompleksitas O(1) karena bekerja dengan menerima kondisi tertentu yang tidak bergantung pada pengecekan blok di depan mobil tetapi bergantung pada pengecekan posisi mobil lawan terhadap mobil pemain yang dapat dianggap sebagai konstanta. Penggunaan perintah *FIX* juga memiliki kompleksitas O(1) karena bekerja berdasarkan kondisi *damage* yang diterima oleh mobil. Perintah *NOTHING* dan *DECELERATE* tidak digunakan sehingga tidak menambah kompleksitas apapun ke dalam program. Kompleksitas akhir program untuk alternatif strategi 2 adalah $O(n^2)$ untuk setiap ronde dalam permainan *Overdrive*

3.3.3. Alternatif Strategi 3

Implementasi alternatif strategi ini mirip seperti alternatif strategi 2, tetapi urutan prioritas penggunaan *PowerUp* untuk alternatif strategi 3 berbeda dengan alternatif strategi 2. Kompleksitas algoritma alternatif strategi 3 mirip seperti kompleksitas algoritma alternatif strategi 2 dengan algoritma pergerakan mobil memiliki kompleksitas $O(n^2)$ karena harus mengecek blok di depan mobil dan memberikan skor pada jalur di sekitar mobil. Perbedaan kompleksitas algoritma antara alternatif strategi 2 dan strategi 3 yaitu program dapat mengeksekusi perintah *NOTHING* sebagai prioritas terakhir mobil ketika seluruh kondisi sebelum prioritas perintah *NOTHING* tidak terpenuhi.

Penggunaan *PowerUp LIZARD* memiliki kompleksitas sebesar $O(n^2)$ karena mengecek blok di depan mobil untuk dilompati. *PowerUp BOOST* memiliki kompleksitas O(n) mirip seperti alternatif strategi 2 serta *PowerUp* lainnya dengan perintah *FIX* memiliki kompleksitas O(1). Perintah *DECELERATE* tidak digunakan dalam program dan tidak menambah kompleksitas program seperti pada solusi 1. Kompleksitas akhir program untuk alternatif strategi 3 adalah $O(n^2)$ untuk setiap ronde dalam permainan *Overdrive*.

3.4. Analisis Efektivitas

3.4.1. Alternatif Strategi 1

Alternatif strategi 1 paling tidak efektif dari alternatif strategi lainnya karena mobil tidak menggunakan beberapa *PowerUp* yang tersedia dalam permainan *Overdrive* seperti *USE_EMP*, *USE_OIL*, serta *USE_TWEET*. Penggunaan *PowerUp* tersebut memiliki manfaat untuk menghampat pergerakan lawan sehingga mobil tidak dapat melewati mobil lawan jika mobil lawan selalu menggunakan *PowerUp BOOST* dan *PowerUP* yang dapat menghambat pergerakan mobil.

3.4.2. Alternatif Strategi 2

Alternatif strategi ini cukup efektif dalam memenangkan permainan *Overdrive* karena kondisi yang seimbang dalam menghindari rintangan serta menggunakan *PowerUp* untuk menambah kecepatan mobil serta menghambat lawan. Kekurangan dari alternatif ini yaitu penggunaan *USE_TWEET* memiliki kemungkinan untuk menabrak mobil sendiri ketika lawan berada dekat dengan posisi mobil baik di depan maupun di belakang, sehingga *truck* dari *PowerUp TWEET* dapat menghambat mobil sendiri. Kekurangan lainnya yaitu mobil dapat menabrak rintangan ketika mengeksekusi perintah *ACCELERATE* karena sejumlah blok yang dicek hanya sebesar nilai kecepatan mobil, bukan kecepatan selanjutnya akibat perintah *ACCELERATE* (Contohnya *myCar.speed* = 6 menjadi *myCar.speed* = 8 ketika mobil mengeksekusi perintah *ACCELERATE* sehingga blok yang dapat dilalui oleh kecepatan mobil 8 mungkin untuk memiliki rintangan yang belum dicek oleh program).

3.4.3. Alternatif Strategi 3

Alternatif strategi ini paling efektif dalam memenangkan permainan *Overdrive* dari alternatif strategi lainnya karena mobil menggunakan perintah *NOTHING* untuk menghindari penggunaan perintah *ACCELERATE* ketika terdapat rintangan yang dapat dilalui oleh kecepatan mobil selanjutnya. Alternatif strategi ini juga memiliki kondisi yang seimbang dalam menghindari rintangan serta menggunakan *PowerUp* untuk menambah kecepatan mobil serta menghambat lawan. Kekurangan dari alternatif strategi ini mirip seperti alternatif strategi 2 yaitu penggunaan *USE_TWEET* memiliki kemungkinan untuk menabrak mobil sendiri ketika lawan berada dekat dengan posisi mobil baik di depan maupun di belakang lawan, sehingga *truck* dari *PowerUp TWEET* dapat menghambat mobil sendiri.

3.5. Strategi *Greedy* yang Dipilih

Pada akhirnya, penulis memilih desain alternatif strategi 3 sebagai implementasi utama algoritma *greedy* dengan penambahan beberapa kondisi heuristik. Secara garis besar, mobil akan menngeksekusi perintah *FIX* ketika mobil memiliki *damage* lebih dari 2 poin. Ketika memiliki *damage* di bawah 2 poin, program akan mengecek rintangan di depan mobil dengan membuat 2 *list* blok yaitu blok kecepatan mobil

sekarang (*list block*) serta blok kecepatan mobil selanjutnya (*list accelerateBlock*). Jika terdapat rintangan dalam *list block*, mobil akan mencoba menghindar dengan mengeksekusi perintah *USE_LIZARD*, *ACCELERATE*, *TURN_LEFT*, atau *TURN_RIGHT*. *USE_LIZARD* akan dieksekusi ketika terdapat rintangan pada jalur kiri, jalur kanan, serta jalur tengah mobil pada *list block* yang tidak dapat dihindari serta mobil memiliki *PowerUp LIZARD*. *ACCELERATE*, *TURN_LEFT*, serta *TURN_RIGHT* akan dieksekusi ketika jalur yang dipilih mobil memiliki skor yang tertinggi tergantung jumlah rintangan serta blok *PowerUp* yang ada. Jika tidak terdapat rintangan, mobil akan meprioritaskan untuk mengeksekusi perintah *USE_EMP*, *USE_TWEET*, atau *USE_OIL* dengan prioritas *USE_EMP*, *USE_TWEET*, atau *USE_OIL* secara terurut. *USE_EMP* akan dieksekusi ketika lawan berada di depan mobil serta berada dalam zona serangan *EMP*. Perintah *USE_TWEET* akan dieksekusi dengan posisi awal lawan ditambah kecepatan lawan dan 2 blok di depan mobil lawan. Prioritas *USE_OIL* digunakan jika mobil pemain sudah berada di depan lawan dan lawan berjarak 5 blok atau lebih dari mobil sendiri. Jika semua kondisi untuk menghambat lawan tidak terpenuhi, mobil akan mengeksekusi perintah *USE_BOOST*. Kondisi *USE_BOOST* akan terpenuhi jika kecepatan mobil berada di bawah 15 poin, tidak terdapat rintangan 15 blok di depan mobil, serta memiliki *PowerUp BOOST*. Jika kondisi *USE_BOOST* tidak terpenuhi, mobil akan mengeksekusi perintah *ACCELERATE* jika blok dalam *list accelerateBlock* tidak terdapat rintangan. Jika terdapat rintangan dalam *list accelerateBlock*, maka mobil akan mengeksekusi perintah *NOTHING* tanpa menambah kecepatan mobil.

Pertimbangan dari pemilihan alternatif strategi ini yaitu efisiensi serta efektivitas strategi. Secara efektivitas, strategi ini cukup efektif dalam memenangkan permainan dari hasil *trial and error* menggunakan efektivitas alternatif strategi 2 serta penambahan heuristik. Mobil tidak perlu selalu memperbaiki mobil ketika terkena *damage* serta mobil dapat menghindari rintangan menggunakan *list block* serta *list accelerateBlock.* Skor yang didapatkan oleh mobil juga maksimal karena penggunaan *PowerUp* secara keseluruhan serta menghindar dari rintangan yang cukup merugikan. Secara efisiensi, strategi ini ketika dibandingkan dengan ketiga alternatif lainnya memiliki kompleksitas algoritma yang sama yaitu $O(n^2)$ karena kondisi heuristik yang digunakan berupa konstanta.

BAB 4: Implementasi Dan Pengujian

4.1. Implementasi Algoritma *Greedy*

Implementasi algoritma *greedy* pada program bot dalam *game* engine akan dipaparkan menggunakan *pseudocode* serta komentar untuk memperjelas proses program:

4.1.1. public Command run

```
function run() → Command
{Mengembalikan command/perintah yang akan digunakan oleh bot
berdasarkan strategi greedy yang digunakan}

KAMUS LOKAL
middleScore: integer
lizardScore: integer
leftScore : integer
rightScore : integer
{Nilai/score untuk jalur kiri, kanan, serta tengah mobil}

blocks : Array of Object
{Menyimpan sejumlah block di depan mobil}

acceleratedBlocks: Array of Object
{Menyimpan sejumlah block di depan mobil ketika mobil
mengeksekusi perintah ACCELERATE}

boostBlocks: Array of Object
{Menyimpan sejumlah block di depan mobil ketika mobil akan
melakukan perintah USE_BOOST}

speedList: Array[0..8] of integer = 0
{List untuk menyimpan speed state mobil}

ALGORITMA
{Menyimpan kecepatan mobil selanjutnya (next speed state) dengan
menggunakan kecepatan awal mobil (myCar.speed) sebagai index
array}
speedList[0] ← 3
speedList[3] ← 5
speedList[5] ← 6
speedList[6] ← 8
speedList[8] ← 9
```

```
{Mencoba untuk melakukan perintah FIX ketika mobil memiliki
damage ≥ 2}
if (Mycar.damage ≥ 2) then
     → FIX

{Menyimpan block di jalur tengah depan mobil sejumlah nilai yang
sama dengan kecepatan mobil selanjutnya
(speedtList[myCar.speed]). Jika kecepatan mobil ≥ 9, blok yang
diambil sebesar kecepatan mobil itu juga}
if myCar.speed < 9 then
     accelerateBlocks ← getBlocksInFront(myCar.position.lane,
     myCar.position.block + 1, speedList[myCar.speed] - 1)

else

     accelerateBlocks ← getBlocksInFront(myCar.position.lane,
     myCar.position.block + 1, myCar.speed - 1)

{Menyimpan block di jalur tengah depan mobil sejumlah nilai yang
sama dengan kecepatan mobil sekarang (myCar.speed)}
blocks ← getBlocksInFront(myCar.position.lane,
          myCar.position.block + 1, myCar.speed - 1)


{Mengecek apakah terdapat rintangan dalam list blocks}

if (isNotClear(blocks)) then
{Mengambil skor dari accelerateBlock (jalur tengah next speed),
block (jalur tengah speed sekarang), jalur kiri, serta jalur
kanan. Jika bot berada di pojok kiri/kanan maka skor jalur
kiri/kanan tidak perlu dihitung}

     middleScore ← getScore(accelerateBlocks)
     lizardScore ← getScore(blocks)
     {skor untuk melakukan PowerUp Lizard}

     if (myCar.position.lane ≠ 1) then
          leftScore ← getScore(getBlocksInFront
                      (myCar.position.lane - 1,
                      myCar.position.block, myCar.speed-1))

     if (myCar.position.lane ≠ 4) then
```

```
            rightScore ← getScore(getBlocksInFront
                          (myCar.position.lane + 1,
                          myCar.position.block, myCar.speed-1))

      depend on (rightScore, lizardScore, leftScore)
      {Jika jalur kiri, jalur kanan, serta jalur tengah  speed
      sekarang memiliki skor negatif (terdapat rintangan) dan bot
      memiliki PowerUp Lizard, bot eksekusi perintah USE_LIZARD}

      rightScore < 0 AND lizardScore < 0 AND leftScore < 0 AND
                  hasPowerUp(PowerUps.LIZARD) : → LIZARD

      {Eksekusi perintah gerakan menuju jalur kiri, jalur kanan,
      atau jalur tengah next speed yang memiliki skor tertinggi}

      middleScore ≥ leftScore AND middleScore ≥ rightScore : →
                                                ACCELERATE

      leftScore ≥ middleScore AND leftScore ≥ rightScore AND
                     myCar.position.lane ≠ 1 : → TURN_LEFT

      rightScore ≥ middleScore AND rightScore ≥ leftScore AND
                     myCar.position.lane ≠ 4 : → TURN_RIGHT

{Mengecek jika musuh ada di depan pemain dan berada di baris
dekat pemain. Jika musuh ditemukan dan memiliki powerup EMP, maka
EMP digunakan}

if ((opponent.position.lane = myCar.position.lane OR
opponent.position.lane = myCar.position.lane - 1 OR
opponent.position.lane = myCar.position.lane + 1) AND
opponent.position.block > myCar.position.block) then
if (hasPowerUp(PowerUps.EMP)) then
      → EMP

{Mencoba menggunakan PowerUp TWEET sesuai dengan kondisi}
if hasPowerUp(PowerUps.TWEET) then
      → new TweetCommand(opponent.position.lane,
opponent.position.block + opponent.speed + 2)

{Eksekusi PowerUp OIL sesuai dengan kondisi}
if (opponent.position.lane = myCar.position.lane AND
opponent.position.block + 5 < myCar.position.block) then
```

```
     if hasPowerUp(PowerUps.OIL) then
          → OIL

{Menyimpan block jalur tengah sebanyak 15 block di depan mobil}
boostBlocks ← getBlocksInFront(myCar.position.lane,
myCar.position.block + 1, 14)

{Mencoba mengeksekusi PowerUp BOOST, jika mobil memiliki damage,
maka mobil diperbaiki terlebih dahulu}
if (hasPowerUp(PowerUps.BOOST) AND myCar.speed ≤ 15 AND
isClearForBoost()) then
     if (myCar.damage ≥ 1) then
          → FIX
     else
          → BOOST
else
{Mobil tidak melakukan apa-apa jika perintah ACCELERATE
mengakibatkan mobil menabrak rintangan}
     if (isNotClear(accelerateBlocks)) then
          → NOTHING
     else
          → ACCELERATE
```

4.1.2. private int getScore

```
function getScore (blocks: Array of Object) → integer
{Mengembalikan hasil total score dari seluruh block terhadap
Terrain}

KAMUS LOKAL
score: integer {total score}
i: integer {index traversal}

ALGRORITMA
i traversal[0..blocks.size()]
     {Menambahkan nilai skor berdasarkan karakteristik jenis
     block. Mobil akan mengutamakan mengambil jalur yang memiliki
     PowerUp serta sedikit atau tidak ada rintangan}

     depend on (blocks[i])
          blocks[i] = Terrain.WALL      : score ← score – 1000
          blocks[i] = Terrain.MUD       : score ← score – 100
          blocks[i] = Terrain.BOOST     : score ← score + 5
```

```
          blocks[i] = Terrain.OIL_SPILL : score ← score – 100
          blocks[i] = Terrain.LIZARD    : score ← score + 1
          blocks[i] = Terrain.EMP       : score ← score + 5
          blocks[i] = Terrain.TWEET     : score ← score + 2
     else
          score ← score + 1
→ score
```

4.1.3. private Boolean hasPowerUp

```
function hasPowerUp(powerUpToCheck: PowerUps) → boolean
{Mengecek apakah terdapat suatu power up yang tersedia dalam bot
myCar}

KAMUS LOKAL
i: integer {index traversal}

ALGORITMA
i traversal [0..myCar.powerups.size()]
     if powerUp[i] = powerUpToCheck then
          → true
→ false
```

4.1.4. private List<Object> getBlocksInFront

```
function getBlocksInFront(lane: integer, block: integer, speed:
integer) → Array of Object
{Mengembalikan list suatu block yang dapat dihinggapi oleh
kecepatan masukan terhadap parameter lane dengan posisi awal
block berasal dari parameter block}

KAMUS LOKAL
map : Array of Lane
blocks: Array of Object
startBlock: integer
laneList: Lane
i: integer {untuk traversal}

ALGORITMA
map ← gameState.lanes
startBlock ← map[0].position.block
laneList ← map[lane – 1]
i traversal[max(block-startBlock, 0)..block-startBlock+speed)]
```

```
     blocks.add(laneList[i].terrain)
→ blocks
```

4.1.5. private Boolean isNotClear

```
function isNotClear(blocks: Array of Object) → boolean
{Mengecek apakah terdapat rintangan berupa Mud, Oil Spill, atau
Wall di dalam parameter list block}

KAMUS LOKAL
-

ALGORITMA
→ blocks.contains(Terrain.MUD) OR blocks.contains(Terrain.WALL)
   OR blocks.contains(Terrain.OIL_SPILL)
```

4.2. Struktur Data Program

Permainan Overdrive menggunakan struktur data berbentuk *class*/kelas. Berikut adalah file berisi kelas yang digunakan untuk bot overdrive:

4.2.1. Bagian Command

Kelas-kelas pada bagian ini adalah kelas yang berguna untuk menyimpan perintah gerakan serta penggunaan *PowerUp* untuk *bot* mobil

a) **AccelerateCommand**
   Kelas yang digunakan untuk menyimpan perintah *ACCELERATE*
b) **BoostCommand**
   Kelas yang digunakan untuk menyimpan perintah penggunaa powerup *BOOST* / *USE_BOOST*
c) **ChangeLaneCommand**
   Kelas yang digunakan untuk menyimpan perintah belok yaitu *TURN_LEFT* dan *TURN_RIGHT*
d) **Command**
   Kelas yang digunakan untuk memproses perintah
e) **DecelerateCommand**
   Kelas yang digunakan untuk menyimpan perintah *DECELERATE*
f) **DoNothingCommand**
   Kelas yang digunakan untuk menyimpan perintah *NOTHING*.
g) **EmpCommand**
   Kelas yang digunakan untuk menyimpan perintah penggunaan powerup *EMP* / *USE_EMP*.
h) **FixCommand**
   Kelas yang digunakan untuk menyimpan perintah untuk memperbaiki mobil / *FIX*

i) **LizardCommand**
Kelas yang digunakan untuk menyimpan perintah penggunaan powerup *LIZARD* / *USE_LIZARD*.

j) **OilCommand**
Kelas yang digunakan untuk menyimpan perintah penggunaan powerup *OIL* / *USE_OIL*.

k) **TweetCommand**
Kelas yang digunakan untuk menyimpan perintah penggunaan powerup *TWEET* / TweetCommand(lane, block) dengan lane dan block menunjukkan posisi powerup *TWEET* ingin digunakan.

4.2.2. Folder Entitites

Bagian ini berisi kelas-kelas untuk menyimpan data *entity* dari permainan.

a) **Car**
Kelas untuk menyimpan atribut-atribut terkait kondisi mobil, seperti kecepatan, posisi, dll.

b) **GameState**
Kelas untuk menyimpan data-data permainan seperti data-data mobil pemain, ronde yang sedang berjalan, ronde maksimal, data lawan, dsb.

c) **Lane**
Kelas untuk menyimpan atribut-atribut seperti posisi dan terrain.

d) **Position**
Kelas untuk membentuk atribut koordinat x dan y.

4.2.3. Folder Enums

Bagian ini berisi kelas-kelas untuk menyimpan data-data seperti *direction* pada peta, *state bot*, serta *PowerUp* yang tersedia pada permainan.

a) **Direction**
Kelas yang digunakan untuk menyimpan atribut-atribut arah yaitu FORWARD, BACKWARD, LEFT, dan RIGHT.

b) **PowerUps**
Kelas yang digunakan untuk menyimpan powerup yaitu BOOST, OIL, EMP, TWEET, dan LIZARD.

c) **State**
Kelas yang digunakan untuk menyimpan atribut-atribut terkait keadaan kendaraan. Berikut adalah atribut keadaan kendaraan :

- ACCELERATING
- READY
- NOTHING
- TURNING_LEFT
- TURNING_RIGHT
- HIT_MUD

- HIT_OIL
- DECELERATING
- PICKED_UP_POWERUP
- USED_BOOST
- USED_OIL
- USED_LIZARD
- USED_TWEET
- HIT_WALL
- HIT_CYBER_TRUCK
- FINISHED

d) Terrain

Kelas yang digunakan untuk menyimpan atribut-atribut terkait sifat jalan yang dilalui mobil.

4.2.4. Kelas Bot

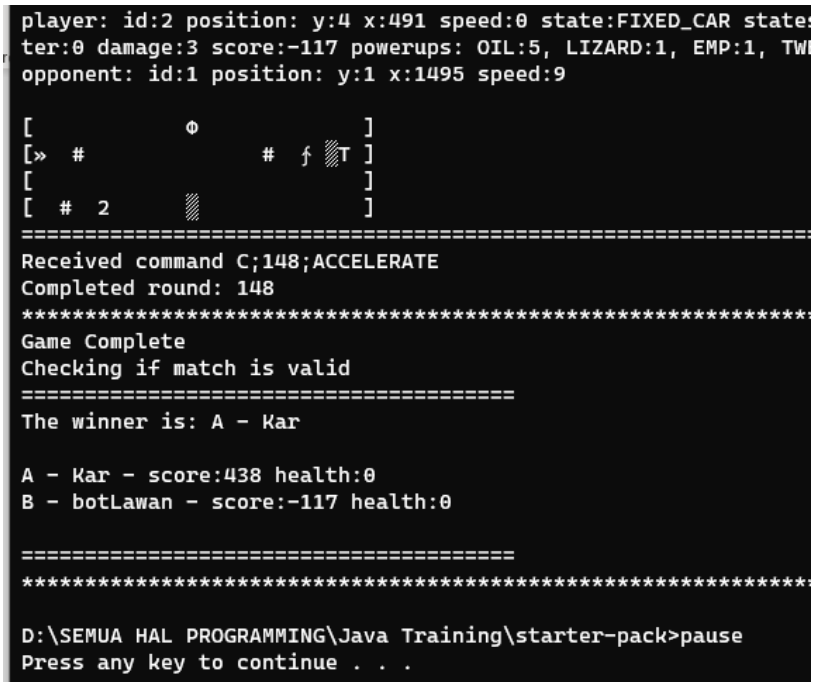
Kelas ini berisi pengembangan dari implementasi algoritma *greedy* yang kami rancang dan gunakan untuk memenangkan permainan "Overdrive", memanfaatkan kelas-kelas yang disebutkan sebelumnya.

4.2.5. Kelas Main

Kelas yang digunakan untuk memanggil bot dan melaksanakan program dengan menjalankan seluruh algoritma dan *command-command* yang sudah dibentuk untuk memulai permainan.

4.3. Analisis Pengujian Algoritma *Greedy*

Hasil percobaan untuk analisis pengujian algoritma *greedy* ini didapatkan dengan cara menjalankan `run.bat` dalam folder `starter-pack` (kedua *bot* yang akan ditandingkan harus sudah di-*build* sebelumnya) untuk memulai permainan "Overdrive".



Gambar 2. Hasil Permainan Overdrive

"Player A" merupakan *bot* yang menggunakan strategi *greedy* yang sudah dipilih sedangkan "Player B" merupakan *bot* lawan. Dengan menggunakan referensi *bot* sebagai *bot* lawan, *bot* dengan strategi *greedy* mampu memangkan permainan sebanyak 10 dari 10 permainan yang dicoba. Hal ini disebabkan referensi *bot* memiliki algoritma yang sederhana sehinga tolak ukur untuk pengujian algoritma *greedy* ini kurang baik.

Dengan menggunakan alternatif strategi 1 sebagai *bot* lawan, bot dengan strategi *greedy* ini mampu memenangkan permainan sebanyak 10 dari 10 permainan. Kemenangan strategi *greedy* ini disebabkan oleh penggunaan *PowerUp* untuk menghambat *bot* lawan secara terus menerus serta *bot* lawan tidak dapat menyerang balik karena strategi yang ada di dalam *bot* tersebut.

Dengan menggunakan alternatif strategi 2 sebagai *bot* lawan, bot dengan strategi *greedy* ini mampu memenangkan permainan sebanyak 7 dari 10 permainan. Kekalahan strategi *greedy* ini disebabkan oleh penempatan rintangan yang merugikan *bot greedy* tetapi tidak merugikan *bot* lawan. Contohnya seperti penempatan rintangan *Mud* yang banyak di depan *bot greedy* tetapi *bot* lawan tidak terdapat rintangan sama sekali. Kekalahan strategi ini juga disebabkan oleh *bot* lawan mendapatkan banyak *PowerUp* sehingga *bot* lawan menyerang secara terus menerus terutama penggunaan *EMP* dan *TWEET* yang merugikan *bot greedy*. Kemudian kemenangan strategi *greedy* ini dapat disebabkan oleh kelebihan *bot*

dalam menggunakan perintah *NOTHING* ketika terdapat rintangan jika menggunakan perintah *ACCELERATE*.

Dengan menggunakan alternatif strategi 3 sebagai *bot* lawan, bot dengan strategi *greedy* ini mampu memenangkan permainan sebanyak 4 dari 10 permainan. Kekalahan strategi ini karena alternatif strategi 3 merupakan desain utama dari strategi *greedy* yang dipilih sehingga algoritma yang digunakan tidak jauh beda, hanya saja terdapat beberapa penggunaan heuristik untuk strategi *greedy* yang dipilih. Penyebab kekalahan *bot greedy* sama seperti yang dijelaskan ketika melawan *bot* alternatif strategi 3 akibat penempatan rintangan serta penggunaan *PowerUp*.

Kesimpulan yang didapatkan dari kekalahan algoritma *greedy* ini disebabkan oleh penempatan rintangan pada peta yang merugikan *bot greedy* sehingga bot menjadi kurang efisien dalam menghindari rintangan. Kelemahan algoritma *greedy* ini juga terdapat ketika mobil mencoba melompat menggunakan *USE_LIZARD* lalu mendarat pada rintangan yang menghambat mpbo; meskipun tidak terdapat rintangan yang perlu dilompat. Hal ini karena strategi *greedy* yang perlu meminimalisir kerusakan karena ketiga jalur terdapat rintangan yang menghambat (Meskipun rintangan hanya terdapat pada blok pendaratan mobil). Penggunaan *PowerUp* secara terus menerus dari *bot* lawan juga menghambat bot *greedy* untuk menang ketika *bot* lawan mendapatkan banyak *PowerUp*. Kemudian kemenangan algoritma *greedy* ini bisa disimpulkan karena kondisi strategi bot yang sudah efektif (*fix*, menghindar, *PowerUp*) serta penggunaan perintah *NOTHING* untuk meminimalisir menabrak rintangan ketika *ACCELERATE*. Kemenangan *bot greedy* juga disebabkan oleh kelebih algoritma *greedy* yang telah dijelaskan sebelumnya.

BAB 5: Kesimpulan dan Saran

5.1. Kesimpulan

Kami berhasil mengimplementasikan permainan "Overdrive" menggunakan algoritma *Greedy* yang bisa mencapai tujuan objektif dari permainan ini, yaitu memenangkan permainan ini baik dengan cara mencapai garis *finish* terlebih dahulu, memiliki kecepatan tercepat jika *finish* bersamaan maupun memiliki skor terbesar bila mencapai garis *finish* bersamaan dan memiliki kecepatan yang sama. Selain itu, kami juga menerapkan beberapa pendekataan strategi *greedy* dalam prosesnya untuk memenangkan permainan, seperti *command* untuk memperbaiki mobil jika *damage*-nya lebih dari 2, memperlambat lawan, ataupun menghindari maupun menggunakan *powerup* jika terdapat rintangan.

Namun, berdasarkan pengujian dengan *reference bot*, bot yang kami buat tidak selalu menang. Dari hal ini, kami menyadari bahwa algoritma *greedy* yang dibuat tidak selalu menghasilkan solusi yang paling optimal pada implementasi permainan "Overdrive" ini.

5.2. Saran

Pengaplikasian algoritma greedy pada permainan "Overdrive" ini masih dapat menghasilkan strategi yang lebih optimal lagi sehingga saran kami pada tugas besar kali ini adalah agar pengembangan permainan "Overdrive" ini mungkin dapat dikembangkan kembali pada masa yang mendatang. Program dapat dikembangkan lebih jauh menggunakan pendekatan-pendekatan greedy yang lebih efisien dan efektif dari strategi yang telah kami implementasikan. Untuk waktu jangka panjangnya, prinsip Intelegensi Buatan (Artificial Intelligence) mungkin dapat diterapkan pada permainan ini sehingga bot yang dimainkan dapat "belajar" setiap pengujiannya dan beradaptasi pada pengujian selanjutnya.

Daftar Pustaka

Entelect Challenge. 2020. Entelect Challenge 2020 - Overdrive. Diakses pada 18 Februari 2022 dari https://github.com/EntelectChallenge/2020-Overdrive

Franché van den Berg. 2020. Entelect Challenge - Community Visualizers. Diakses pada 18 Februari 2022 dari https://entelect-replay.raezor.co.za/

Munir, R. 2021. Algoritma Greedy Bagian 1. Institut Teknologi Bandung. Diakses pada 18 Februari 2022 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

Munir, R. 2021. Algoritma Greedy Bagian 2. Institut Teknologi Bandung. Diakses pada 18 Februari 2022 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

Munir, R. 2021. Algoritma Greedy Bagian 3. Institut Teknologi Bandung. Diakses pada 18 Februari 2022 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

Lampiran

Link *Source Code* (Github):

https://github.com/JeremyRio/StimaTubes1

Link Video Penjelasan Algoritma (*Youtube*):

https://youtu.be/bc6TsTZazrM