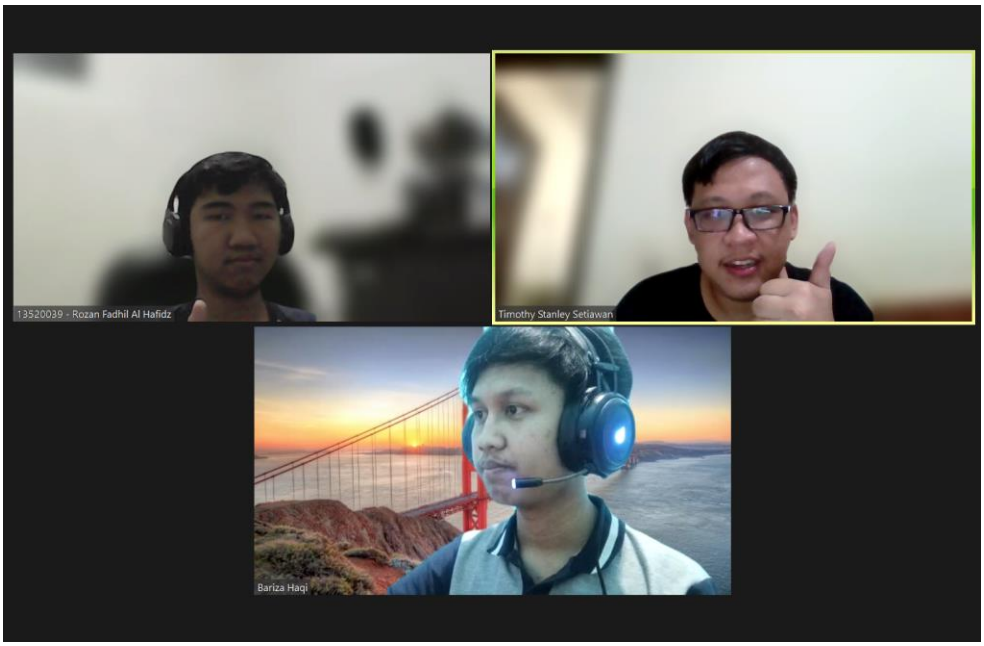
LAPORAN TUGAS BESAR

IF2211/Strategi Algoritma



Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Overdrive"

Dipersiapkan oleh:

Kelompok 41

Bariza Haqi	13520018
Timothy Stanley Setiawan	13520028
Rozan Fadhil Al Hafidz	13520039

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

Daftar Isi

Daftar Tabel

Daftar Gambar

Bab 1 : Deskripsi Masalah

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

1. Download latest release starter pack.zip dari tautan berikut https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4
2. 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/
3. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk Windows dapat buka dengan double-click, Untuk Linux/Mac dapat menjalankan command “make run”).
4. 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.
5. 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.
6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan visualizer berikut https://github.com/Affuta/overdrive-round-runner
7. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut.

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 powerups 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 merupakan metode yang paling populer dan sedehana untuk memecahkan persoalan optimasi. Persoalan optimasi (optimization problems) adalah persoalan mencari solusi optimal. Ada dua macam persoalan optimasi:, yaitu maksimasi dan minimasi.

Kata greedy sendiri dalam Bahasa Inggris memiliki arti rakus atau tamak. Prinsip algortima greedy adalah “take what you can take now!”. Algoritma greedy membentuk solusi langkah per langkah. Kemudian pada setiap langkah, terdapat banyak pilihan yang perlu dievaluasi. Dengan demikian, pada setiap langkah harus dibuat keputusan dalam menentukan pilihan yang terbaik. Tidak bisa kembali ke langkah sebelumnya. Jadi pada setiap langkah, kita memilih optimum lokal dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global.

Definsi Algortima greedy adalah algortima yang memecahkan masalah secara langkah per Langkah sehingga pada setiap langkahnya :

1) Mengambil pilihan terbaik yang dapat diambil saat itu juga tanpa memperhatikan kosenkuensi ke depan (sesuai dengan prinsip greedy)
2) Dengan memilih optimum lokal untuk tiap langkahnya, diharapkan akan mengarah kepada optimum global.

Elemen-elemen yang biasa muncul dalam algorima greedy:

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

Walaupun algortima greedy diandalkan untuk penyelesaian masalah optimasi, algotima greedy belum tentu bisa menghasilkan hasil yang paling optimal (optimum global). Ada dua hal yang mengakibatkan hal ini terjadi

1) Algoritma greedy tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada.
2) Terdapat beberapa fungsi seleksi yang berbeda sehingga kita harus memilih fungsi yang tepat agar masalah ini dapat menghasilkan solusi yang optimal.

2.2 Algoritma Greedy

Permaian overdrive ini pada dasarnya akan meminta masukan command yang akan dilakukan mobil untuk setiap rondenya hingga mobil kita atau musuh mencapai garis finish. Pada tugas besar ini, peran si pemberi command itu diambil alih oleh bot yang akan dibuat. Bot yang dibuat nantinya akan memilih command yang terbaik (optimum) untuk ronde yang sedang berlangsung. Karena menggunakan algoritma greedy, bot tidak perlu tau kejadian sebelum dan akan terjadi di ronde berikutnya. Hal yang menjadi dasar pertimbangan bot untuk mengambil keputusan adalah kondisi mobil kita dan/ atau musuh pada ronde yang sedang berlangsung (posisi, *damage*, *speed*, *power up* yang dimiliki), *obstacle* yang bertebaran di jalan, dan *power up* yang bertebaran dijalan. Untuk mendapatkan informasi-informasi tersebut, perlu akses terhadap *gamestate* yang sedang berlangsung dengan cara mengimpor *src* yang berisi *gamestate* yang sedang berlangsung (*command, entities, enum*).

Dengan informasi-informasi tersebut, bot diberikan sekumpulan solusi yang harus diambil bot apabila terjadi kondisi tertentu. Sekumpulan solusi itu lah yang akan ditelaah oleh kami, bukan dilakukan oleh bot (bukan *machine learning*). Kemudian, setelah sekumpulan solusi yang sudah cantumkan pada bot diambil berdasarkan prioritas tertentu yang sudah dibuat, bot akan mengembalikan nilai command yang nantinya akan menjadi nilai masukan command pada game. Prinsip greedy yang diambil dalam pembuatan bot ini adalah dengan sekumpulan solusi yang sudah dibuat, solusi apakah yang terbaik dilakukan untuk menghadapi kondisi yang terjadi pada tiap ronde-ronde yang terjadi (optimum lokal) dengan harapan akan mengarah kepada kemenangan melawan bot musuh (optimum global).

Bab 3 : Pemanfaatan Strategi Greedy

3.1 Pemetaan Algoritma Greedy pada Permainan Overdrive

3.1.1 Pemetaan Umum Permasalahan Permainan

Berikut ini merupakan pemetaan permainan Overdrive secara umum menjadi elemen-elemen penyusun algoritma Greedy.

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang mungkin dijalankan oleh pemain, yaitu NOTHING, TURN_LEFT, TURN_RIGHT, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET, dan USE_LIZARD.
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan kondisi setiap ronde permainan selama permainan berlangsung.
Fungsi Solusi	Memeriksa kondisi yang sedang dialami mobil dan keadaan sekitarnya untuk mendapatkan solusi yang paling optimal
Fungsi Seleksi (Selection Function)	Menentukan prioritas dari solusi berdasarkan keadaan yang paling menguntungkan dan tidak menyebabkan kerugian sekecil apapun. Dalam permainan Overdrive, prioritas utama adalah menghindari jebakan, kemudian mempercepat mobil, setelah itu menyerang mobil musuh.
Fungsi Kelayakan (Feasibility)	Menentukan kelayakan dari perintah yang akan dieksekusi, yaitu mengecek blok di depan sebelum melakukan percepatan mobil dan

	mengecek ketersediaan power up sebelum memerintahkan command power up
Fungsi Objektif	Fungsi objektif pada permainan Overdrive adalah membuat mobil yang kita mainkan melewati garis finish paling awal sebagai prioritas pertama dan mendapat skor yang tinggi sebagai prioritas kedua.

Table 1 Pemetaan Umum Permasalahan Permainan

3.1.2 Pemetaan Perbaikan Mobil, Percepat Mobil, dan Penggunaan Boost

Perbaikan mobil merupakan salah satu hal yang paling penting pada permainan ini. Semakin banyak kerusakan yang dialami oleh mobil, semakin rendah pula kecepatan minimumnya. Saat mobil tidak rusak sama sekali (damage = 0), mobil tersebut dapat mencapai kecepatan 15 dengan menggunakan *power up boost*. Oleh karena itu, perbaikan mobil harus dilakukan di saat yang tepat agar pergerakannya menjadi efisien.

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang terkait adalah FIX, ACCELERATE, dan USE_BOOST
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan kondisi kecepatan mobil saat ini dan jumlah *power up boost* yang dimiliki.
Fungsi Solusi	Memeriksa apakah mobil memiliki kerusakan yang cukup parah. Selain itu, memeriksa apakah memiliki jumlah boost yang cukup banyak. Jika iya, lakukan perbaikan pada mobil. Jika mobil masih bisa dipercepat, lakukan percepat mobil.
Fungsi Seleksi (Selection Function)	Jika kerusakan yang dialami tidak terlalu parah dan mobil tidak punya banyak *power up boost*,

	jangan lakukan perbaikan karena akan membuat mobil ketinggalan giliran (*turn*).
Fungsi Kelayakan (Feasibility)	Menentukan kelayakan dari perintah yang akan dieksekusi, yaitu mengecek damage yang diterima mobil. Selain itu, cek blok di depan mobil sebelum mempercepat mobil atau menggunakan *boost*.
Fungsi Objektif	Fungsi objektifnya adalah membuat mobil yang kita mainkan bisa mencapai kecepatan sekencang mungkin agar dapat mencapai garis finish dengan cepat.

Table 2 Pemetaan Perbaikan Mobil, Percepatan Mobil, dan Penggunaan Boost

3.1.3 Pemetaan Pergerakan Mobil dan Pengambilan *Power Up*

Pergerakan mobil juga merupakan hal yang penting di dalam permainan ini karena di dalam permainan ini terdapat jebakan yang sebisa mungkin dihindari dan berbagai *power up* yang dapat diambil.

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang terkait adalah USE_LIZARD, TURN_LEFT dan TURN_RIGHT
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan mempertimbangkan kondisi daerah sekitar mobil dan kecepatan mobil
Fungsi Solusi	Memeriksa apakah terdapat jebakan di depan mobil. Kemudian memeriksa apakah terdapat jalan buntu di depannya lagi. Selain itu, periksa apakah ada *power up* yang bisa

	diambil. Setelah itu, pilih lajur yang paling baik. Jika tidak ada, gunakan *lizard*.
Fungsi Seleksi (Selection Function)	Prioritas utama adalah menghindari jebakan. Jika tidak ada jebakan, barulah mengecek lajur terjangkau yang memiliki *power up.*
Fungsi Kelayakan (Feasibility)	Menentukan kelayakan dari perintah yang akan dieksekusi, yaitu mengecek apakah gerakan pindah lajur yang dilakukan akan membawa mobil ke jalan buntu. Jika tidak, maka lakukan gerakan tersebut layak untuk dilakukan.
Fungsi Objektif	Fungsi objektifnya adalah untuk menghindari jebakan sebisa mungkin agar terhindar dari *damage* dan mendapat *power up* sebanyak mungkin.

Table 3 Pemetaam Pergerakan Mobil dan Pengambilan Power Up

3.1.4 Pemetaan Penyerangan Mobil Lawan

Penyerangan merupakan salah satu bagian yang penting untuk mencegah mobil musuh melewati garis *finish* lebih dahulu. Menyerang mobil musuh bisa dilakukan dengan menggunakan *power up*.

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang terkait adalah USE_OIL, USE_TWEET, dan USE_EMP
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan posisi mobil musuh terhadap mobil sendiri dan lingkungan sekitar.
Fungsi Solusi	Jika mobil musuh berada di belakang, gunakan power up *oil*. Jika mobil musuh berada di depan, gunakan dan terjangkau oleh EMP,

	gunakan EMP. Jika tidak ada, gunakan TWEET.
Fungsi Seleksi (Selection Function)	Pemilihan perintah dilakukan berdasarkan posisi mobil lawan terhadap mobil sendiri. Selain itu, lingkungan sekitar juga menjadi pertimbangan. Misalnya ada 3 jebakan pada satu kolom, maka tambah jebakan di baris yang kosong agar menjadi 4 jebakan dalam satu kolom.
Fungsi Kelayakan (Feasibility)	Menentukan kelayakan dari perintah yang akan dieksekusi, yaitu mengecek apakah di depan mobil pemain terdapat jebakan sehingga mobil bisa menggunakan *power up* dengan aman
Fungsi Objektif	Fungsi objektifnya adalah untuk mencegah mobil lawan melewati garis finish lebih dulu tanpa merugikan mobil sendiri.

Table 4 Pemetaan Penyerangan Mobil Lawan

3.2 Eksplorasi Alternatif Strategi *Greedy* dalam Permainan Overdrive

Mekanisme permainan Overdrive cukup kompleks sehingga ada sangat banyak strategi yang mungkin diimplementasikan pada permainan ini. Oleh karena itu, eksplorasi strategi ini akan dibagi ke dalam kelompok-kelompok strategi untuk memudahkan klasifikasi strategi. Berikut ini merupakan alternatif-alternatif solusinya.

3.2.1 Strategi Perbaikan Mobil, Percepat Mobil, dan Penggunaan Boost

Kelompok strategi ini memperkirakan saat yang tepat untuk melakukan perbaikan mobil, melakukan percepatan mobil, dan melakukan *boost*. Hal yang menjadi pertimbangan adalah kerusakan yang dialami mobil, kecepatan mobil, dan jumlah *boost* yang dimiliki.

Strategi pertama adalah melakukan perbaikan mobil setiap mobil menerima damage. Strategi ini membuat mobil selalu bisa dipercepat hingga kecepatan 15.

Strategi kedua adalah melakukan perbaikan mobil hanya saat mobil tidak bisa bergerak. Strategi ini membuat mobil lebih sering berjalan dan jarang kehilangan giliran (*turn*).

Strategi ketiga adalah melakukan perbaikan setiap memiliki *boost*. Strategi ini menjadikan mobil tidak terlalu sering kehilangan giliran (*turn*) namun tetap dapat mempercepat dirinya hingga kecepatan 15.

Strategi keempat adalah gabungan ketiga strategi sebelumnya, yaitu, melakukan perbaikan mobil disaat kecepatan mobil sudah relative maksimum terhadap kerusakan yang diterima mobil, lakukan perbaikan mobil. Selain itu, jika mobil memiliki satu atau lebih *boost*, lakukan perbaikan mobil agar mobil dapat mencapai kecepatan 15 saat menggunakan *boost*.

3.2.2 Strategi Pergerakan Mobil dan Pengambilan *Power Up*

Kelompok strategi ini mengatur kapan mobil harus berbelok untuk menghindari jebakan dan mengambil power up. Hal yang menjadi pertimbangan pada strategi ini adalah kondisi daerah sekitar mobil dan kecepatan mobil.

Strategi pertama adalah dengan menghindari jebakan dan lajur tepi. Jika terdapat jebakan di depan mobil, maka belok ke lajur yang tidak terdapat jebakan. Jika tidak terdapat jebakan, cek apakah mobil berada di lajur pinggir (lajur 1 atau 4). Jika iya, pindah ke lajur yang lebih tengah.

Strategi kedua adalah menghindari jebakan dan jalan buntu. Jika terdapat jebakan di depan mobil, maka belok ke lajur yang tidak terdapat jebakan. Jika tidak terdapat jebakan, cek blok di depan mobil apakah menuju ke jalan buntu (jalan yang tertutup jebakan). Jika iya, belok ke lajur yang tidak menuju jalan buntu.

Strategi ketiga adalah dengan mengambil power up dan melompati blok jebakan dengan menggunakan *power up lizard*. Caranya adalah mengecek apakah ada lajur terjangkau yang mengandung power up. Jika ada, belok ke lajur tersebut. Urutan prioritas power up adalah boost > lizard > emp > tweet > oil. Jika ada jebakan, gunakan *power up lizard* untuk menghindari jebakan.

Strategi keempat adalah menggabungkan ketiga strategi di atas, yaitu menghindari jebakan, menghindari jalan buntu, menghindari lajur pinggir, dan mengambil power up jika terjangkau. Jika tidak terdapat lajur yang aman, gunakan *power up lizard* untuk menghindari jebakan.

3.2.3 Strategi Penyerangan Mobil Lawan

Kelompok strategi ini mengatur kapan mobil melakukan penyerangan kepada mobil lawan menggunakan power up. Hal-hal yang menjadi pertimbangan adalah posisi mobil lawan dan jumlah power up yang dimiliki.

Strategi pertama adalah jika mobil lawan ada di belakang dan terdapat tiga blok jebakan dalam satu kolom, maka tambah jebakan pada baris yang masih kosong menjadi empat jebakan dalam satu kolom, bisa menggunakan *oil* maupun *tweet*.

Strategi kedua adalah jika mobil sendiri sudah cukup cepat dan mobil lawan berada di belakang, langsung keluarkan *oil* di blok mobil sendiri atau gunakan *tweet* di blok depan mobil lawan jika mobil memiliki *power up* tersebut.

Strategi ketiga adalah jika musuh ada di depan, gunakan EMP setiap kali musuh berada di dalam jangkauan EMP (maksimal berjarak satu lane) jika mobil memiliki *power up* EMP.

Strategi keempat adalah gabungan dari tiga strategi sebelumnya, dengan prioritas adalah strategi pertama > strategi kedua. Jika mobil lawan berada di depan, gunakan strategi ketiga.

3.3 Analisis Efisiensi Strategi

Untuk strategi perbaikan mobil, algoritma yang digunakan hanyalah mengecek kerusakan yang sedang diterima mobil. Oleh karena itu, kompleksitasnya adalah O(1). Untuk mempercepat mobil atau menggunakan boost, algortima yang digunakan adalah mengecek blok di depan mobil dengan tujuan mobil tidak menabrak jebakan yang ada di depannya. Oleh karena itu, kompleksitas algoritmanya sebanding dengan jumlah blok yang dapat dijangkau mobil jika melakukan percepatan, yaitu O(n). Untuk menggunakan *boost*, hanya terdapat tambahan pengecekan apakah mobil memiliki *power up boost* atau tidak dengan kompleksitas O(n). Jadi, kelompok strategi perbaikan mobil, percepat mobil, dan penggunaan boost ini memiliki total kompleksitas O(n).

Untuk strategi menghindari jebakan di depan, algoritma yang digunakan adalah mengecek blok yang ada di depan mobil. Kompleksitasnya sebanding dengan jumlah blok yang dicek, yaitu O(n). Jika terdapat jebakan, jalankan algoritma untuk mengecek lajur terjangkau yang paling baik. Maksud dari lajur terjangkau adalah lajur yang bisa dipilih mobil hanya dengan satu kali perintah (belok kiri atau kanan). Kompleksitasnya juga sebanding dengan jumlah blok yang dicek, yaitu O(n). Untuk mengecek lajur yang menuju jalan buntu, terdapat algoritma yang mengecek di seluruh lajur di depan blok yang dapat dijangkau. Kompleksitasnya juga sebanding dengan jumlah blok yang dicek, yaitu O(n). Terdapat juga algoritma untuk mengecek lajur terjangkau yang mengandung *power up*. Kompleksitasnya juga sebanding dengan jumlah blok yang dicek, yaitu O(n). Jika seluruh algoritma kelompok strategi pergerakan mobil dan pengambilan *power up* digabungkan, kompleksitasnya adalah O(n).

Untuk strategi menggunakan *power up oil*, algoritma yang digunakan adalah mengecek posisi mobil lawan terhadap mobil sendiri. Kompleksitasnya O(1). Untuk membuat jalan buntu yang menghalangi mobil lawan (membuat 4 jebakan berjejer di kolom) menggunakan *oil* atau *tweet*, algoritma tambahan yang digunakan adalah mengecek apakah ada kolom yang mengandung 3 jebakan berjejer. Kompleksitasnya sebanding dengan jumlah blok yang di cek, yaitu O(n). Untuk menggunakan *power up* EMP, algoritma yang digunakan adalah mengecek posisi mobil lawan terhadap mobil sendiri. Kompleksitasnya adalah O(1). Jika seluruh algoritma kelompok strategi penyerangan mobil lawan digabungkan, kompleksitasnya adalah O(n).

3.4 Analisis Efektivitas

Pada bagian ini, kami akan menjelaskan efektivitas algoritma yang kami buat. Analisis ini dilakukan untuk mencari algoritma yang paling baik untuk memainkan permainan Overdrive.

Perbaikan mobil merupakan hal yang harus dipertimbangkan di permainan ini karena kerusakan mobil akan membatasi kecepatan maksimum yang dapat dilakukan mobil. Perbaikan mobil akan mengurangi *damage* mobil sebanyak 2 satuan. Oleh karena itu, perbaikan mobil akan efektif untuk dilakukan saat *damage* yang dialami mobil lebih besar atau sama dengan 2. Namun, mobil harus tidak rusak sama sekali (*damage*=0). Oleh karena itu, kita membuat kondisi baru. Jika mobil memiliki kerusakan lebih dari atau sama dengan dua (*damage* >=2) dan mobil memiliki boost lebih dari satu, maka lakukan perbaikan mobil. Kami menggunakan parameter *boost* lebih dari satu untuk mengantisipasi musuh menggunakan EMP sesaat setalah mobil sendiri melakukan *boost* sehingga jika terdapat *boost* lebih dari satu, mobil tersebut bisa langsung menggunakan *boost* lagi.

Menghindari jebakan dan mengambil *power up* juga tak kalah pentingnya. Oleh karena itu, selain menghindari jebakan yang berada di depan mobil, kami juga mengecek blok di depannya lagi apakah ada jalan yang buntu. Hal ini dilakukan untuk meminimumkan *damage* yang diterima mobil. Semakin jarang mobil terkena *damage,* semakin jarang mobil berhenti untuk memperbaiki *damage*. Selain itu, *power up* juga sangat efektif untuk melaju dengan cepat dan menghadang mobil lawan. Oleh karena itu, mengumpulkan *power up* sambal menghindari jebakan merupakan strategi yang sangat efektif.

Menyerang mobil musuh juga penting untuk mencegah lawan melewati garis *finish* lebih dulu. Menggunakan *power up* untuk menyerang lawan akan sangat efektif jika sedang tidak ada jebakan yang menghalangi mobil. *Power up* yang menurut kami paling efektif adalah EMP. EMP dapat menyerang mobil lawan yang berada di depan dengan jangkauan vertikal 3 lajur dan jangkauan horizontal tak terhingga. Oleh karena itu, kami memerintahkan mobil kami untuk langsung menggunakan EMP saat mobil lawan sedang di depan dan tidak ada jebakan yang menghalangi. Selain itu, menggunakan *oil* juga efektif saat mobil musuh berada di belakang. Jika tidak memiliki *oil* dan EMP, menggunakan *tweet* juga efektif untuk menghalangi musuh karena

jika mobil musuh menabrak cybertruck, maka mobil tersebut akan berhenti sesaat dan terkena *damage*.

3.5 Pemilihan Algoritma Greedy

Pada akhirnya, kami memilih gabungan-gabungan dari alternatif solusi yang kami jabarkan pada bagian sebelumnya. Kami menyesuaikan seluruh algoritma yang ada agar dapat digunakan secara bersamaan. Agar algoritma yang kami gabungkan berjalan dengan baik, kami menentukan skala prioritas pada algoritma yang kami buat.

Prioritas pertama adalah melakukan perbaikan mobil jika mobil tidak berjalan sama sekali. Kemudian adalah percepat mobil jika sudah diperbaiki namun mobil belum berjalan. Setelah itu, lakukan perbaikan mobil jika mobil mengalami kerusakan cukup parah dan kecepatannya sudah mencapai batas. Setelah itu, hindari jebakan dan jalan buntu yang menghalangi mobil. Pakai *power up lizard* jika diperlukan. Menghindari jebakan dapat dilakukan sambil mengambil *power up* jika menemukannya.

Prioritas selanjutnya adalah menggunakan *power up* EMP jika mobil lawan berada di depan dan masuk ke jangkauan EMP. Hal ini dilakukan untuk membuat mobil lawan berhenti sementara dan mengurangi kecepatan mobil lawan. Kemudian, lakukan perbaikan mobil jika mobil memiliki *boost* lebih dari satu. Hal ini dilakukan agar mobil bisa mencapai kecepatan 15. Setelah itu, ambil *power up* jika menemukannya di blok yang dapat dijangkau.

Prioritas selanjutnya adalah menggunakan power up-power up lain, yaitu *oil* dan *tweet* sesuai keadaan. Power up tersebut bertujuan untuk menghalangi mobil lawan melewati garis finish lebih dulu. Setelah itu, cek apakah mobil lawan menghalangi mobil sendiri. Jika iya, belok ke lajur yang tidak mengandung jebakan.

Bab 4 : Implementasi dan Pengujian

4.1 Implementasi Algoritma *Greedy*

Implementasi algoritma greedy pada program kami terdapat pada bagian file bot.java. Berikut adalah Source Code-nya.

```
public class Bot {

    private List<Command> directionList = new ArrayList<>();

    private final Random random;
    private final GameState gameState;
    private final Car opponent;
    private final Car myCar;
    private boolean dontTurnLeft; // true jika mobil tidak boleh belok kiri
    private boolean dontTurnRight; // true jika mobil tidak boleh belok kanan

    private final static Command TURN_RIGHT = new ChangeLaneCommand(1);
    private final static Command TURN_LEFT = new ChangeLaneCommand(-1);
```

4.1.1 Konstruktor class bot

```
    // Inisialisasi objek atau variable pada konstruktor
    public Bot(Random random, GameState gameState) {
        this.random = random;
        this.gameState = gameState;
        this.myCar = gameState.player;
        this.opponent = gameState.opponent;
        this.dontTurnLeft = false;
        this.dontTurnRight = false;

        directionList.add(TURN_LEFT);
        directionList.add(TURN_RIGHT);
    }
```

4.1.2 Fungsi Run

```
    // prosedur untuk menentukan strategi mobil yang akan dipertandingkan agar bisa memenangkan
pertandingan. Urutan kode akan menentukan urutan langkah yang akan dilakukan pada mobil (dari atas ke
bawah).
public Command run() {
        // Variabel yang menyimpan blok terjangkau di depan
        List<Object> blocks = getBlocksInFront(this.myCar.position.lane, this.myCar.position.block + 1);
```

```
        // Variabel yang menyimpan blok terjangkau di kiri
        List<Object> leftBlocks = getBlocksInFront(this.myCar.position.lane - 1,
this.myCar.position.block);
        // Variabel yang menyimpan blok terjangkau di kanan
        List<Object> rightBlocks = getBlocksInFront(this.myCar.position.lane + 1,
this.myCar.position.block);
        // Variabel yang menyimpan blok terjangkau yang bisa dicapai saat pindah tempat untuk boost di
kanan
        List<Object> extendedLaneBoostRight = getBlocksInFront(this.myCar.position.lane + 1,
this.myCar.position.block + this.myCar.speed + 15);
        // Variabel yang menyimpan blok terjangkau yang bisa dicapai saat pindah tempat untuk boost di
kiri mobil
        List<Object> extendedLaneBoostLeft = getBlocksInFront(this.myCar.position.lane - 1,
this.myCar.position.block + this.myCar.speed + 15);
        // Variabel yang menyimpan blok terjangkau di depan saat accelerate
        List<Object> extendedBlocksAcc = getBlocksInFrontToAcc(this.myCar.position.lane,
this.myCar.position.block + 1, nextSpeedIfAcc());
        // Variabel yang menyimpan blok terjangkau di depan saat boost
        List<Object> extendedBlocksBoost = getBlocksInFrontToAcc(this.myCar.position.lane,
this.myCar.position.block + 1, 15);

        // Lakukan fix jika mobil menerima damage maksimum
        if (this.myCar.damage == 5) {
            return new FixCommand();
        }

        // Lakukan accelerate jika mobil tidak berjalan
        if (this.myCar.speed == 0) {
            return new AccelerateCommand();
        }

        // Lakukan fix jika kondisi cocok
        if (this.myCar.speed == this.maxSpeed()) {
            if (this.myCar.damage >= 2 && this.opponent.position.block < this.myCar.position.block)
                return new FixCommand();
            else if (this.myCar.damage >= 3) {
                return new FixCommand();
            }
        }
        // Cek block di depan jangkauan
        Command extendedCommand = checkExtendedLane();
        if (extendedCommand != null) {
            return extendedCommand;
        }

        // Hindari jebakan jika tepat didepan mobil ada jebakan
```

```
        if (blocks != null) {
            if (!laneNotContainObstacle(blocks)) {
                if (leftBlocks != null && rightBlocks != null) {// Jika terdapat lane kiri dan kanan mobil
                    if (laneNotContainObstacle(leftBlocks) && laneNotContainObstacle(rightBlocks)) {//
                        if (laneContainBoost(leftBlocks) && !this.dontTurnLeft) {
                            return TURN_LEFT;
                        } else if (laneContainBoost(rightBlocks) && !this.dontTurnRight) {
                            return TURN_RIGHT;
                        } else if ((laneContainLizard(leftBlocks) || laneContainEMP(leftBlocks))
&& !this.dontTurnLeft) {
                            return TURN_LEFT;
                        } else if ((laneContainLizard(rightBlocks) || laneContainEMP(rightBlocks))
&& !this.dontTurnRight) {
                            return TURN_RIGHT;
                        } else if ((laneContainTweet(leftBlocks) || laneContainOilpower(leftBlocks))
&& !this.dontTurnLeft) {
                            return TURN_LEFT;
                        } else if (!this.dontTurnRight){
                            return TURN_RIGHT;
                        }
                        else if (!this.dontTurnLeft){
                            return TURN_LEFT;
                        }
                        else if (this.myCar.position.lane==3){
                            return TURN_LEFT;
                        }
                        else {
                            return TURN_RIGHT;
                        }
                    }
                    else if (laneNotContainObstacle(rightBlocks)) {
                        return TURN_RIGHT;
                    }
                    else if (laneNotContainObstacle(leftBlocks)) {
                        return TURN_LEFT;
                    }

                    else {
                        if (hasPowerUp(PowerUps.LIZARD, this.myCar.powerups) &&
blockNotContainObstacle(blocks,blocks.size())) {// Jika punya powerup lizard dan saat melompat tidak
mendarat dijebakan
                            return new LizardCommand();
                        }
                        else {
                            if (minObstacle(blocks, rightBlocks, leftBlocks) == 1) {
                                return new AccelerateCommand();
```

```
                            } else if (minObstacle(blocks, rightBlocks, leftBlocks) == 2) {
                                return TURN_RIGHT;
                            } else if (minObstacle(blocks, rightBlocks, leftBlocks) == 3) {
                                return TURN_LEFT;
                            } else {
                                if (!blocks.contains(Terrain.WALL)
|| !blocks.contains((Terrain.CYBERTRUCK))) {
                                    return new AccelerateCommand();
                                } else if (!rightBlocks.contains(Terrain.WALL)
|| !rightBlocks.contains((Terrain.CYBERTRUCK))) {
                                    return TURN_RIGHT;
                                } else if (!leftBlocks.contains(Terrain.WALL)
|| !leftBlocks.contains((Terrain.CYBERTRUCK))) {
                                    return TURN_LEFT;
                                } else {
                                    if (this.myCar.speed >= 8) {
                                        return new DecelerateCommand();
                                    } else {
                                        return new AccelerateCommand();
                                    }
                                }
                            }
                        }
                    }
                }
                if (leftBlocks != null) {// Jika hanya terdapat lane kiri
                    if (laneNotContainObstacle(leftBlocks)) {
                        return TURN_LEFT;
                    }
                    else {
                        if (hasPowerUp(PowerUps.LIZARD, this.myCar.powerups) &&
blockNotContainObstacle(blocks,blocks.size())) { // Jika punya powerup lizard dan saat melompat tidak
mendarat dijebakan
                            return new LizardCommand();
                        }
                        else {
                            if (minObstacle(blocks, null, leftBlocks) == 1) {
                                return new AccelerateCommand();
                            }
                            else if (minObstacle(blocks, null, leftBlocks) == 3) {
                                return TURN_LEFT;
                            } else {
                                if (!blocks.contains(Terrain.WALL)
|| !blocks.contains((Terrain.CYBERTRUCK))) {
                                    return new AccelerateCommand();
                                }
```

```
                                else if (!leftBlocks.contains(Terrain.WALL)
|| !leftBlocks.contains((Terrain.CYBERTRUCK))) {
                                    return TURN_LEFT;
                                }
                                else {
                                    if (this.myCar.speed >= 8) {
                                        return new DecelerateCommand();
                                    } else {
                                        return new AccelerateCommand();
                                    }
                                }
                            }
                        }
                    }
                }
                if (rightBlocks!=null){
                    if (laneNotContainObstacle(rightBlocks)) {// Jika hanya terdapat lane kanan
                        return TURN_RIGHT;
                    }
                    else {
                        if (hasPowerUp(PowerUps.LIZARD, this.myCar.powerups) &&
blockNotContainObstacle(blocks,blocks.size())) { // Jika punya powerup lizard dan saat melompat tidak
mendarat dijebakan
                            return new LizardCommand();
                        }
                        else {
                            if (minObstacle(blocks, rightBlocks, null) == 1) {
                                return new AccelerateCommand();
                            }
                            else if (minObstacle(blocks, rightBlocks, null) == 2) {
                                return TURN_RIGHT;
                            } else {
                                if (!blocks.contains(Terrain.WALL)
|| !blocks.contains((Terrain.CYBERTRUCK))) {
                                    return new AccelerateCommand();
                                }
                                else if (!rightBlocks.contains(Terrain.WALL)
|| !rightBlocks.contains((Terrain.CYBERTRUCK))) {
                                    return TURN_RIGHT;
                                }
                                else {
                                    if (this.myCar.speed >= 8) {
                                        return new DecelerateCommand();
                                    } else {
                                        return new AccelerateCommand();
                                    }
```

```
                                }
                            }
                        }
                    }
                }
            }
        }

        // pakai emp jika terdapat mobil musuh di depan dan kecepatan mobil musuh tidak pada kecepatan
awal
        if (hasPowerUp(PowerUps.EMP,this.myCar.powerups) && this.opponent.position.block >
this.myCar.position.block && (this.myCar.position.lane == this.opponent.position.lane ||
this.myCar.position.lane+1 == this.opponent.position.lane || this.myCar.position.lane -1 ==
this.opponent.position.lane) && this.opponent.speed!=3) {
            return new EmpCommand();
        }

        // fix jika mobil terkena damage dan memiliki boost dan tidak terkena jebakan saat melakukan boost
        if (this.myCar.damage > 0) {
            if ((extendedBlocksBoost)!=null) {
                if (countPowerUps(PowerUps.BOOST, this.myCar.powerups) >= 1 &&
laneNotContainObstacle(extendedBlocksBoost)) {
                    return new FixCommand();
                }
            }
        }

        // Lakukan boost jika keadaan memungkinkan
        if (this.myCar.damage == 0 && this.myCar.speed < 15) {
            if (hasPowerUp(PowerUps.BOOST, this.myCar.powerups)) {
                if (extendedBlocksBoost != null) {
                    if (laneNotContainObstacle(extendedBlocksBoost)  && this.myCar.state !=
State.HIT_EMP){ // block depan saat boost tidak ada jebakan dan status mobil tidak sedang dalam keadaan
tertembak EMP
                        return new BoostCommand();
                    }
                }
            }
        }

        // Cek apakah ada lane terjangkau yang mengandung power up boost
        if (laneContainBoost(blocks)) {
            return goStraight(extendedBlocksAcc);
        }
        if (laneContainBoost(leftBlocks)) {
            if (!this.dontTurnLeft) {
```

```
            return TURN_LEFT;
        }
    }
    if (laneContainBoost(rightBlocks)) {
        if (!this.dontTurnRight) {
            return TURN_RIGHT;
        }
    }

    // Cek apakah ada lane terjangkau yang mengandung power up lizard
    if (laneContainLizard(blocks)) {
        return goStraight(extendedBlocksAcc);
    }
    if (laneContainLizard(leftBlocks)) {
        if (!this.dontTurnLeft) {
            return TURN_LEFT;
        }
    }
    if (laneContainLizard(rightBlocks)) {
        if (!this.dontTurnRight) {
            return TURN_RIGHT;
        }
    }

    // pakai tweet jika terdapat susunan jebakan sehingga membuat musuh tidak bisa menghindar
    if (hasPowerUp(PowerUps.TWEET, this.myCar.powerups)) {
        if (placeTweet()!=null) {
            return placeTweet();
        }
    }
    // pakai oil jika terdapat jebakan di sisi kanan dan kiri mobil
    if (this.opponent.position.block +10 < this.myCar.position.block && canPutOil()) {
        return new OilCommand();
    }

    // Cek apakah ada sisi kiri dan kanan kosong sehingga bisa melakukan boost
    if (hasPowerUp(PowerUps.BOOST, this.myCar.powerups)) {
        if (extendedLaneBoostRight!=null) {
            if (laneNotContainObstacle(extendedLaneBoostRight)) {
                return TURN_RIGHT;
            }
        }
        if (extendedLaneBoostLeft!=null) {
            if (laneNotContainObstacle(extendedLaneBoostLeft)) {
                return TURN_LEFT;
            }
```

```
        }
    }

    // Cek apakah ada lane terjangkau yang mengandung power up EMP
    if (laneContainEMP(leftBlocks)) {
        if (!this.dontTurnLeft) {
            return TURN_LEFT;
        }
    }
    if (laneContainEMP(rightBlocks)) {
        if (!this.dontTurnRight) {
            return TURN_RIGHT;
        }
    }

    // Naikkan kecepatan jika kecepatan kurang dari max speed
    if (this.myCar.speed<this.maxSpeed()) {
        return goStraight(extendedBlocksAcc);
    }

    // Cek apakah ada lane terjangkau yang mengandung power up tweet
    if (laneContainTweet(leftBlocks)) {
        if (!this.dontTurnLeft) {
            return TURN_LEFT;
        }
    }
    if (laneContainTweet(rightBlocks)) {
        if (!this.dontTurnRight) {
            return TURN_RIGHT;
        }
    }

    // Cek apakah ada lane terjangkau yang mengandung power up oil
    if (this.opponent.position.block<this.myCar.position.block) {
        if (laneContainOilpower(leftBlocks)) {
            if (!this.dontTurnLeft) {
                return TURN_LEFT;
            }
        }
        if (laneContainOilpower(rightBlocks)) {
            if (!this.dontTurnRight) {
                return TURN_RIGHT;
            }
        }
    }
```

```
        // Cek apakah ada musuh tepat didepan menghalangi
        if (this.myCar.position.lane==this.opponent.position.lane &&
this.opponent.position.block==this.myCar.position.block+1) {
            if (leftBlocks!=null) {
                if (!dontTurnLeft && laneNotContainObstacle(leftBlocks)) {
                    return TURN_LEFT;
                }
            }
            if (rightBlocks!=null) {
                if (!dontTurnRight && laneNotContainObstacle(rightBlocks)) {
                    return TURN_RIGHT;
                }
            }
        }

        // pakai tweet jika ada
        if (hasPowerUp(PowerUps.TWEET, this.myCar.powerups)){
            return new TweetCommand(this.opponent.position.lane, this.opponent.position.block +
this.opponent.speed + 4);
        }

        return new DoNothingCommand(); // tidak melakukan apapun
    }
```

4.1.3 Fungsi ExtendedLane

```
    /* Cek lane di depan jangkauan agar mobil tidak terjebak*/
    private Command checkExtendedLane() {
        List<Object> extendedLane1;
        List<Object> extendedLane2;
        List<Object> extendedLane3;
        List<Object> extendedLane4;
        List<Object> Lane2;
        List<Object> Lane3;

        int speed;
        int speedAcc;
        // biar kejangkau jebakannya saat kecepatan kecil
        if (this.myCar.speed<6) {
            speed=12;
            speedAcc=8;
        }
        else {
            speed=this.myCar.speed;
            speedAcc=nextSpeedIfAcc();
```

```
        }
        switch (this.myCar.position.lane) {
            case 1:
                extendedLane1 = getBlocksInFrontToAcc(1, this.myCar.position.block + speedAcc + 1,
speedAcc);
                extendedLane2 = getBlocksInFront(2, this.myCar.position.block + speed);
                Lane2 = getBlocksInFront(2, this.myCar.position.block);
                if (extendedLane1 != null && extendedLane2 != null) {
                    if (!laneNotContainObstacle(extendedLane1) && !laneNotContainObstacle(extendedLane2))
{
                        if (laneNotContainObstacle(Lane2)) {
                            return TURN_RIGHT;
                        }
                    }
                }
                break;
            case 2:
                extendedLane1 = getBlocksInFront(1, this.myCar.position.block + speed);
                extendedLane2 = getBlocksInFrontToAcc(2, this.myCar.position.block + speedAcc + 1,
speedAcc);
                extendedLane3 = getBlocksInFront(3, this.myCar.position.block + speed);
                extendedLane4 = getBlocksInFront(4, this.myCar.position.block + speed - 1);
                Lane3 = getBlocksInFront(3, this.myCar.position.block);
                if (extendedLane1 != null && extendedLane2 != null && extendedLane3 != null) {
                    if (!laneNotContainObstacle(extendedLane1) && !laneNotContainObstacle(extendedLane2)
&& !laneNotContainObstacle((extendedLane3))) {
                        if (laneNotContainObstacle(Lane3)) {
                            return TURN_RIGHT;
                        }
                    }
                }
                if (extendedLane2 != null && extendedLane3 != null && extendedLane4 != null) {
                    if (!laneNotContainObstacle(extendedLane2) && !laneNotContainObstacle(extendedLane3)
&& !laneNotContainObstacle((extendedLane4))) {
                        this.dontTurnRight = true;
                    }
                }
                if (extendedLane1 != null && extendedLane2 != null) {
                    if (!laneNotContainObstacle(extendedLane1) && !laneNotContainObstacle(extendedLane2))
{
                        this.dontTurnLeft = true;
                    }
                }
                break;
            case 3:
                extendedLane1 = getBlocksInFront(1, this.myCar.position.block + speed - 1);
```

```
                extendedLane2 = getBlocksInFront(2, this.myCar.position.block + speed);
                extendedLane3 = getBlocksInFrontToAcc(3, this.myCar.position.block + speedAcc + 1,
speedAcc);
                extendedLane4 = getBlocksInFront(4, this.myCar.position.block + speed);
                Lane2 = getBlocksInFront(2, this.myCar.position.block);

                if (extendedLane2 != null && extendedLane3 != null && extendedLane4 != null) {
                    if (!laneNotContainObstacle(extendedLane2) && !laneNotContainObstacle(extendedLane3)
&& !laneNotContainObstacle((extendedLane4))) {
                        if (laneNotContainObstacle(Lane2)) {
                            return TURN_LEFT;
                        }
                    }
                }
                if (extendedLane1 != null && extendedLane2 != null && extendedLane3 != null){
                    if (!laneNotContainObstacle(extendedLane1) && !laneNotContainObstacle(extendedLane2)
&& !laneNotContainObstacle((extendedLane3))) {
                        this.dontTurnLeft = true;
                    }
                }
                if (extendedLane3 != null && extendedLane4 != null) {
                    if (!laneNotContainObstacle(extendedLane3)
&& !laneNotContainObstacle((extendedLane4))) {
                        this.dontTurnRight = true;
                    }
                }
                break;
            case 4:
                extendedLane3 = getBlocksInFront(3, this.myCar.position.block + speed);
                extendedLane4 = getBlocksInFrontToAcc(4, this.myCar.position.block + speedAcc + 1,
speedAcc);
                Lane3 = getBlocksInFront(3, this.myCar.position.block);
                if (extendedLane3 != null && extendedLane4 != null) {
                    if (!laneNotContainObstacle(extendedLane3)
&& !laneNotContainObstacle((extendedLane4))) {
                        if (laneNotContainObstacle(Lane3)) {
                            return TURN_LEFT;
                        }
                    }
                }
                break;
        }
        return null;
    }
```

4.1.4 Fungsi PlaceTweet

```
    // memasang tweet agar musuh terjebak jika terdapat jebakan di beberapa bagian lane
    private Command placeTweet() {
        List<Object> backline1 = getBlocksInBack(1);
        List<Object> backline2 =getBlocksInBack(2);
        List<Object> backline3 =getBlocksInBack(3);
        List<Object> backline4 =getBlocksInBack(4);
        if (this.myCar.position.block-this.opponent.position.block>this.opponent.speed+5) { // jika musuh
berada di belakang mobil
            if (!laneNotContainObstacle(backline1) && !laneNotContainObstacle(backline2)
&& !laneNotContainObstacle(backline3)) {
                return new TweetCommand(4, this.myCar.position.block - 1);
            } else if ((!laneNotContainObstacle(backline1) && !laneNotContainObstacle(backline2)
&& !laneNotContainObstacle(backline4))) {
                return new TweetCommand(3, this.myCar.position.block - 1);
            } else if (!laneNotContainObstacle(backline1) && !laneNotContainObstacle(backline3)
&& !laneNotContainObstacle(backline4)) {
                return new TweetCommand(2, this.myCar.position.block - 1);
            } else if (!laneNotContainObstacle(backline2) && !laneNotContainObstacle(backline3)
&& !laneNotContainObstacle(backline4)) {
                return new TweetCommand(1, this.myCar.position.block - 1);
            }
            else if (countPowerUps(PowerUps.TWEET,this.myCar.powerups)>5) {
                if (!laneNotContainObstacle(backline1) && !laneNotContainObstacle(backline2)) {
                    return new TweetCommand(3, this.myCar.position.block - 1);
                } else if (!laneNotContainObstacle(backline1) && !laneNotContainObstacle(backline3)) {
                    return new TweetCommand(2, this.myCar.position.block - 1);
                } else if (!laneNotContainObstacle(backline2) && !laneNotContainObstacle(backline4)) {
                    return new TweetCommand(3, this.myCar.position.block - 1);
                } else if (!laneNotContainObstacle(backline3) && !laneNotContainObstacle(backline4)) {
                    return new TweetCommand(2, this.myCar.position.block - 1);
                }
            }

        }
        return null;
    }
```

4.1.5 Fungsi maxSpeed

```
    /* Mengembalikan nilai maximum speed sesuai keadaan mobil saat ini */
    private int maxSpeed() {
        if (this.myCar.damage == 0){
            return 15;
        }
```

```
    if (this.myCar.damage == 1){
        return 9;
    }
    if (this.myCar.damage == 2){
        return 8;
    }
    if (this.myCar.damage == 3){
        return 6;
    }
    if (this.myCar.damage == 4){
        return 3;
    }
    return 0;
}
```

4.1.6 Fungsi nextSpeedIfAcc

```
/* Mengembalikan nilai kecepatan jika melakukan accelerate */
private int nextSpeedIfAcc() {
    return this.myCar.speed < 3 ? 3 : this.myCar.speed < 6 ? 6 : this.myCar.speed < 8 ? 8 : 9;
}
```

4.1.7 Fungsi laneContainEMP

```
/* Mengembalikan nilai true jika blocks mengandung EMP */
private boolean laneContainEMP(List<Object> blocks) {
    if (blocks != null) {
        if (laneNotContainObstacle(blocks)) {
            return blocks.contains(Terrain.EMP);
        }
    }
    return false;
}
```

4.1.8 Fungsi laneContainTweet

```
/* Mengembalikan nilai true jika blocks mengandung Tweet */
private boolean laneContainTweet(List<Object> blocks) {
    if (blocks != null) {
        if (laneNotContainObstacle(blocks)) {
            return blocks.contains(Terrain.TWEET);
        }
    }
```

```
        return false;
    }
```

4.1.9 Fungsi laneContainLizard

```
    /* Mengembalikan nilai true jika blocks mengandung lizard */
    private boolean laneContainLizard(List<Object> blocks) {
        if (blocks != null) {
            if (laneNotContainObstacle(blocks)) {
                return blocks.contains(Terrain.LIZARD);
            }
        }
        return false;
    }
```

4.1.10 Fungsi laneContainBosst

```
    /* Mengembalikan nilai true jika blocks mengandung boost */
    private boolean laneContainBoost(List<Object> blocks) {
        if (blocks != null) {
            if (laneNotContainObstacle(blocks)) {
                return blocks.contains(Terrain.BOOST);
            }
        }
        return false;
    }
```

4.1.11 Fungsi laneContainOilpower

```
    /* Mengembalikan nilai true jika blocks mengandung oilpower */
    private boolean laneContainOilpower(List<Object> blocks) {
        if (blocks != null) {
            if (laneNotContainObstacle(blocks)) {
                return blocks.contains(Terrain.OIL_POWER);
            }
        }
        return false;
    }
```

4.1.12 Fungsi laneNotContainObstacle

```
    /* Mengembalikan nilai true jika blocks tidak mengandung jebakan */
    private boolean laneNotContainObstacle(List<Object> blocks) {
```

```
        return !blocks.contains(Terrain.MUD) && !blocks.contains(Terrain.OIL_SPILL)
&& !blocks.contains(Terrain.WALL) && !blocks.contains(Terrain.CYBERTRUCK);
    }
```

4.1.13 Fungsi blockNotContainObstacle

```
    // mengembalikan nilai true jika block di posisi tertentu tidak berisi trap
    private boolean blockNotContainObstacle(List<Object> blocks, int position) {
        return blocks.get(position - 1) != Terrain.MUD && blocks.get(position - 1) != Terrain.WALL &&
blocks.get(position - 1) != Terrain.OIL_SPILL && blocks.get(position - 1) != (Terrain.CYBERTRUCK);
    }
```

4.1.14 Fungsi goStraight

```
    // Cek apakah tidak menabrak saat melakukan accelerate
    private Command goStraight(List<Object> blocks) {
        if (laneNotContainObstacle(blocks)) {
            return new AccelerateCommand();
        }
        return new DoNothingCommand();
    }
```

4.1.15 Fungsi canPutOil

```
    // cek apakah terdapat jebakan disisi kanan dan kiri mobil untuk memasang oil
    private boolean canPutOil() {
        List<Object> backlineRight =getBlocksInBack(this.myCar.position.lane+1);
        List<Object> backlineLeft =getBlocksInBack(this.myCar.position.lane-1);
        if (hasPowerUp(PowerUps.OIL,this.myCar.powerups)) {
            if(backlineRight!=null && backlineLeft!=null) {
                return !laneNotContainObstacle(backlineRight) && !laneNotContainObstacle(backlineLeft);
            }
            else if (backlineLeft!=null) {
                return !laneNotContainObstacle(backlineLeft);
            }
            else { //backlineRight!=null
                return !laneNotContainObstacle(backlineRight);

            }
        }
        return false;
    }
```

4.1.16 Fungsi getBlocksInFront

```
// mengembalikan block dari map berisi objek dari lane yang dipilih. Isi block memiliki jumlah sesuai
dengan kecepatan mobil saat ini
private List<Object> getBlocksInFront(int lane, int block) {
        if (lane < 1 || lane > 4) {
            return null;
        }
        List<Lane[]> map = gameState.lanes;
        List<Object> blocks = new ArrayList<>();
        int startBlock = map.get(0)[0].position.block;
        Lane[] laneList = map.get(lane - 1);
        if (laneList != null) {
            int reachableBlock = block - startBlock + this.myCar.speed - 1;
            try {
                for (int i = max(block - startBlock, 0); i <= reachableBlock && i < laneList.length; i++)
{
                    if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) {
                        break;
                    }
                    if (laneList[i].isOccupiedByCyberTruck) blocks.add(Terrain.CYBERTRUCK); // Jika
terdapat cybertruck, isi dengan cybertruck
                    else blocks.add(laneList[i].terrain);

                }
            }
            catch (Exception e){
                return null;
            }
        }
        return blocks.isEmpty() ? null : blocks;
    }
```

4.1.17 Fungsi getBlocksInFrontToAcc

```
// mengembalikan block dari map berisi objek dari lane yang dipilih. Isi block memiliki jumlah sesuai
dengan kecepatan mobil saat accelerate
    private List<Object> getBlocksInFrontToAcc(int lane, int block, int range) {
        if (lane < 1 || lane > 4) {
            return null;
        }
        List<Lane[]> map = gameState.lanes;
        List<Object> blocks = new ArrayList<>();
        int startBlock = map.get(0)[0].position.block;
        Lane[] laneList = map.get(lane - 1);
```

```
        int reachableBlock = block - startBlock + range - 1;
        for (int i = max(block - startBlock, 0); i <= reachableBlock && i<laneList.length; i++) {
            if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) {
                break;
            }
            if (laneList[i].isOccupiedByCyberTruck) blocks.add(Terrain.CYBERTRUCK);
            else blocks.add(laneList[i].terrain);
        }
        return blocks;
    }
```

4.1.18 Fungsi getBlocksInBack

```
    // mengembalikan block dari map berisi objek dari lane yang dipilih. Isi block diambil dari belakang
mobil yang bisa didapat dan 4 blok di depan mobil
    private List<Object> getBlocksInBack(int lane) {
        if (lane < 1 || lane > 4) {
            return null;
        }

        List<Lane[]> map = gameState.lanes;
        int startBlock = map.get(0)[0].position.block;
        List<Object> blocks = new ArrayList<>();
        Lane[] laneList = map.get(lane - 1);
        for (int i = 0; i < max(this.myCar.position.block - startBlock, 0) + 4; i++) {
            if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) {
                break;
            }
            if (laneList[i].isOccupiedByCyberTruck) blocks.add(Terrain.CYBERTRUCK);
            else blocks.add(laneList[i].terrain);
        }
        return blocks;
    }
```

4.1.19 Fungsi hasPowerUp

```
    /* Fungsi untuk mengecek apakah mobil memiliki suatu power up */
    private Boolean hasPowerUp(PowerUps powerUpToCheck, PowerUps[] available){
        for (PowerUps powerUp: available) {
            if (powerUp != null){
                if (powerUp.equals(powerUpToCheck)) {
                    return true;
                }
            }
        }
```

```
        return false;
    }
```

4.1.20 Fungsi countPowerUps

```
    /* Fungsi untuk menghitung suatu power up yang dimiliki */
    private int countPowerUps(PowerUps powerUpToCheck, PowerUps[] available){
        int powerUpCount = 0;
        for (PowerUps powerUp: available) {
            if (powerUp != null){
                if (powerUp.equals(powerUpToCheck)) {
                    powerUpCount++;
                }
            }
        }
        return powerUpCount;

    }
```

4.1.21 Fungsi minObstacle

```
// Mengembalikan lane dengan jebakan paling sedikit. Me-return 1 jika lane yang dipakai mobil paling
sedikit, 2 jika di kanan mobil paling sedikit, 3 jika di kiri mobil paling sedikit, 4 jika terdapat 2 atau
lebih lane yang memiliki jumlah jebakan sama
    private int minObstacle(List<Object> blocks, List<Object> Rightblocks, List<Object> Leftblocks) {
        int count1=0,count2=0,count3=0;
        if (Rightblocks!=null && Leftblocks!=null) {
            int min1;
            for (Object block : blocks) {
                if (block.equals(Terrain.MUD) || block.equals(Terrain.WALL) ||
block.equals(Terrain.OIL_SPILL)) {
                    count1++;
                }
            }
            for (Object rightblock : Rightblocks) {
                if (rightblock.equals(Terrain.MUD) || rightblock.equals(Terrain.WALL) ||
rightblock.equals(Terrain.OIL_SPILL)) {
                    count2++;
                }
            }
            for (Object leftblock : Leftblocks) {
                if (leftblock.equals(Terrain.MUD) || leftblock.equals(Terrain.WALL) ||
leftblock.equals(Terrain.OIL_SPILL)) {
                    count3++;
                }
```

```
            }
            min1=min(count1,count2);
            if(min1==count1 && min1<count3) {
                return 1;
            }
            else if (min1==count2 && min1<count3) {
                return 2;
            }
            else if (min1>count3){
                return 3;
            }
            else { //min1=count3
                return 4;
            }
        }
        else if (Leftblocks!=null) {
            for (Object block : blocks) {
                if (block.equals(Terrain.MUD) || block.equals(Terrain.WALL) ||
block.equals(Terrain.OIL_SPILL)) {
                    count1++;
                }
            }
            for (Object leftblock : Leftblocks) {
                if (leftblock.equals(Terrain.MUD) || leftblock.equals(Terrain.WALL) ||
leftblock.equals(Terrain.OIL_SPILL)) {
                    count3++;
                }
            }
            if(count1<count3) {
                return 1;
            }
            else if (count1>count3) {
                return 3;
            }
            else { //count1=count3
                return 4;
            }
        }
        else { //Rightblock!=null
            for (Object block : blocks) {
                if (block.equals(Terrain.MUD) || block.equals(Terrain.WALL) ||
block.equals(Terrain.OIL_SPILL)) {
                    count1++;
                }
            }
            for (Object rightblock : Rightblocks) {
```

```
                if (rightblock.equals(Terrain.MUD) || rightblock.equals(Terrain.WALL) ||
rightblock.equals(Terrain.OIL_SPILL)) {
                    count2++;
                }
            }
            if(count1<count2) {
                return 1;
            }
            else if (count1>count2) {
                return 2;
            }
            else { //count1=count2
                return 4;
            }
        }
    }
}
```

4.2 Struktur Data Program

Struktur Data pada Program ini berbasis pada kelas yang terbagi menjadi lima bagian, yaitu : Command, Entities, Enums, Bot, Main.

a. Bagian Command

Kelas-kelas pada bagian ini berisi kelas untuk melakukan aksi yang akan dilakukan oleh mobil.

1) AccelerateCommand

Kelas untuk menghasilkan new Command “ACCELERATE” untuk menaikkan state kecepatan mobil.

2) BoostCommand

Kelas untuk menghasilkan new Command “USE_BOOST” untuk menggunakan power up boost.

3) ChangeLancCommand

Kelas untuk menghasilkan new Command untuk mengubah lane mobil yaitu “LEFT” ke kiri dan “RIGHT” ke kanan.

4) Command
   Berisi Interface Command

5) DecelerateCommand
   Kelas untuk menghasilkan new Command “DECELERATE” untuk menurunkan state kecepatan mobil.

6) DoNothingCommand
   Kelas untuk menghasilkan new Command “NOTHING” yaitu tidak melakukan apa-apa.

7) EmpCommand
   Kelas untuk menghasilkan new Command “USE_EMP” untuk menggunakan power up emp.

8) FixCommand
   Kelas untuk menghasilkan new Command “FIX” untuk menurunkan atau menghilangkan damage mobil

9) LizardCommand
   Kelas untuk menghasilkan new Command “USE_LIZARD” untuk menggunakan power up lizard.

10) OilCommand
    Kelas untuk menghasilkan new Command “USE_OIL” untuk menggunakan power up oil.

11) TweetCommand

Kelas untuk menghasilkan new Command “USE TWEET y x“ untuk menggunakan power up tweet dengan memasang CyberTruck pada lane y dan block x.

b. Bagian Entities

Kelas-kelas pada bagian ini berisi kelas untuk menyimpan data entity pada permainan.

1) Car

Kelas untuk menyimpan atribut-atribut yang dimiliki mobil.

2) GameState

Kelas untuk menyimpan data peta, mobil musuh, mobil player, ronde maksimal dan ronde saat ini.

3) Lane

Kelas untuk menyimpan posisi, medan area balap, tempat yang ditempati player dan CyberTruck

4) Position

Kelas untuk menyimpan posisi “y” yaitu lane dan posisi “x” yaitu block.

c. Bagian Enums

Kelas-kelas pada bagian ini berisi data-data tipe blok pada peta, arah-arah pada permainan. status keadaan mobil dan power up yang tersedia pada permainan

1) Direction

Kelas untuk menyimpan enumerasi arah-arah pada permainan seperti “FORWARD” dan sebagainya.

2) PowerUps

Kelas untuk menyimpan enumerasi power up pada permainan seperti “BOOST” dan sebagainya.

3) State

Kelas untuk menyimpan enumerasi state mobil seperti “ACCELERATING” dan sebagainya.

4) Terrain

Kelas untuk menyimpan enumerasi tipe blok pada arena balap seperti “MUD” dan sebagainya.

d. Bot

Kelas ini berisi pengembangan dari implementasi algoritma Greedy yang dirancang untuk memenangkan permainan balap mobil, memanfaatkan dari seluruh kelas.

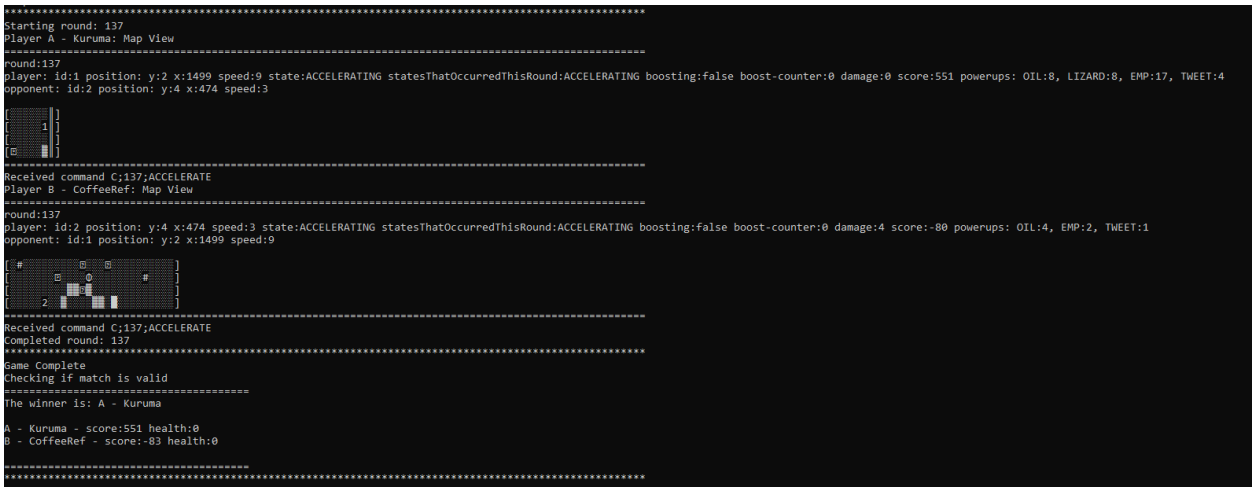
e. Main

Kelas ini untuk menjalankan program secara keseluruhan dari bot yang telah dibuat termasuk command-command yang sudah dibentuk untuk memulai permainan.

4.3 Analisis Pengujian Algoritma *Greedy*

Untuk menjalankan program cukup mengetik ‘run’ pada folder utama permainan dengan terdapat 2 folder bot untuk dipertandingkan. Untuk pengujiannya sendiri terdiri dari tiga kali pengujian terhadap bot yang telah kami rancang.

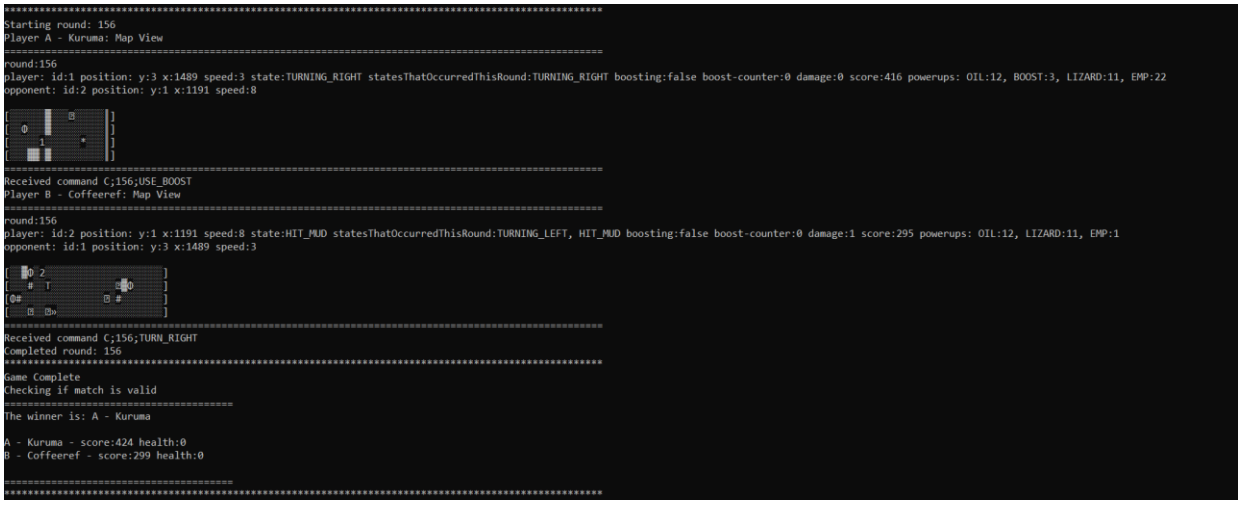
a) Pengujian 1



Gambar 1 Hasil Pengujian 1

Pada pengujian pertama, bot yang kami buat berhasil mengalahkan *reference bot* yang disediakan oleh Entelect hanya dalam 137 ronde dengan jarak yang sangat jauh yaitu 1025 blok. Skor yang didapatkan oleh bot kami pun sangat besar yaitu 551 sedangkan *reference bot* mendapatkan skor -83. Banyak sekali kekurangan pada *reference bot* seperti damage yang bisa di fix hanya bisa sampai 3, kecepatan maksimum yang didapat saat accelerate hanya bisa sampai 6, tidak bisa menggunakan power up tweet, dan tidak bisa menghindar dari oil dan CyberTruck.

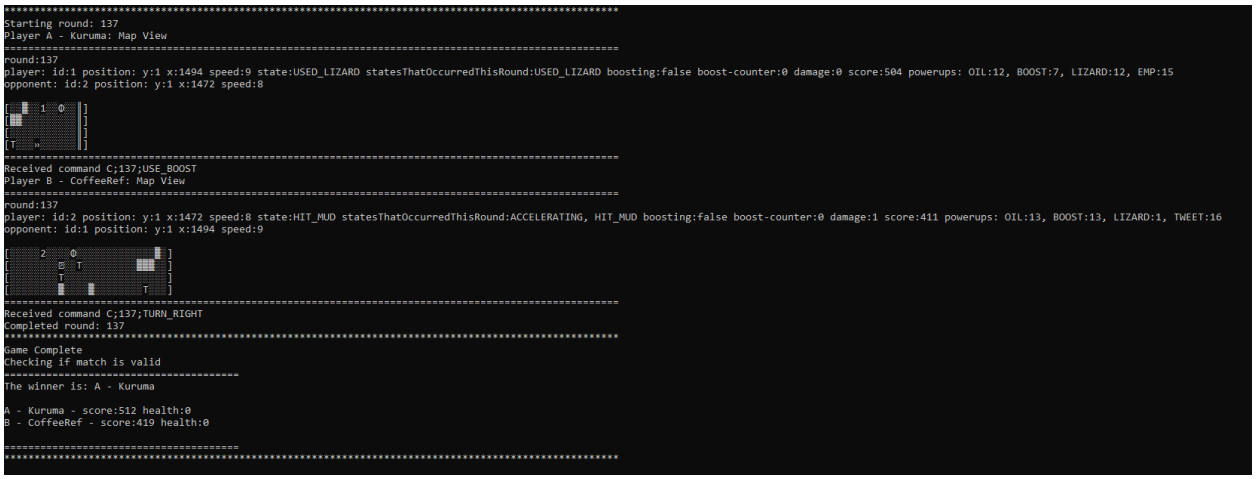
b) Pengujian 2



Gambar 2 Hasil Pengujian 2

Pada pengujian kedua, bot yang kami buat berhasil mengalahkan bot milik kami yang lain dalam 156 ronde dengan jarak yang lumayan jauh yaitu 298 blok. Skor yang didapatkan oleh bot kami pun memiliki selisih 125 poin. hasil ini menentukan bot terbaik yang akan kami pakai.

c) Pengujian 3



Gambar 3 Hasil Pengujian 3

Pada pengujian pertama, bot yang kami buat berhasil mengalahkan bot milik teman kami dalam 137 ronde dengan jarak yang tipis yaitu 22 blok. Skor yang didapatkan oleh bot kami pun memiliki selisih 93 poin. Saat dilihat dalam visualizer, bot kami terus berada di posisi pertama dari awal ronde sampai finish, pada saat ronde 50 bot teman kami tertinggal jauh sebanyak lebih dari 150 blok namun sempat mengejar setelahnya karena bot kami tertembak emp terus sehingga saat finish hanya berbeda 22 blok.

Bab 5 : Kesimpulan dan Saran

5.1 Kesimpulan

1. Hal yang dicapai dalam tugas besar IF2211 Startegi Algoritma yaitu membuat program bot untuk game *overdrive* dengan memanfaatkan algoritma greedy untuk pengambilan keputusan *command*.
2. Bahasa yang digunakan untuk mengembangkan bot adalah Java.
3. Total ada tiga strategi yang menjadi alternatif solusi dalam pengembangan bot ini.
4. Kombinasi dari ketiga strategi yang sudah dibuat menjadi pilihan yang diambil sebagai pengembangan final dari bot.
5. Berdasarkan hasil pengujian melawan bot teman kami, dari total pengujian 20 kali pertandingan total kemenangan yang digapai oleh bot kami adalah 16 kali kemenangan.

5.2 Saran

Saran yang bisa kami berikan untuk mengembangkan bot yang sudah dibuat adalah. Pertama, masih ada beberapa informasi *gamestate* yang belum kami manfaatkan untuk menjadi dasar dalam pengambilan keputusan bot karena kami masih belum menemukan solusi yang tepat untuk memanfaatkan informasi tersebut. Kedua, bot yang dibuat terkadang tidak menghasilkan kemenangan apabila kondisi jalanan yang dilalui kurang mendukung (sulit menghindar *obstacle*, *power up* yang didapat kurang efektif, dsb).

Link GitHub dan Video Kelompok

Link GitHub : https://github.com/Stanley77-web/TubesStima1.git

Link Youtube : https://youtu.be/CJ9ieANZGBE