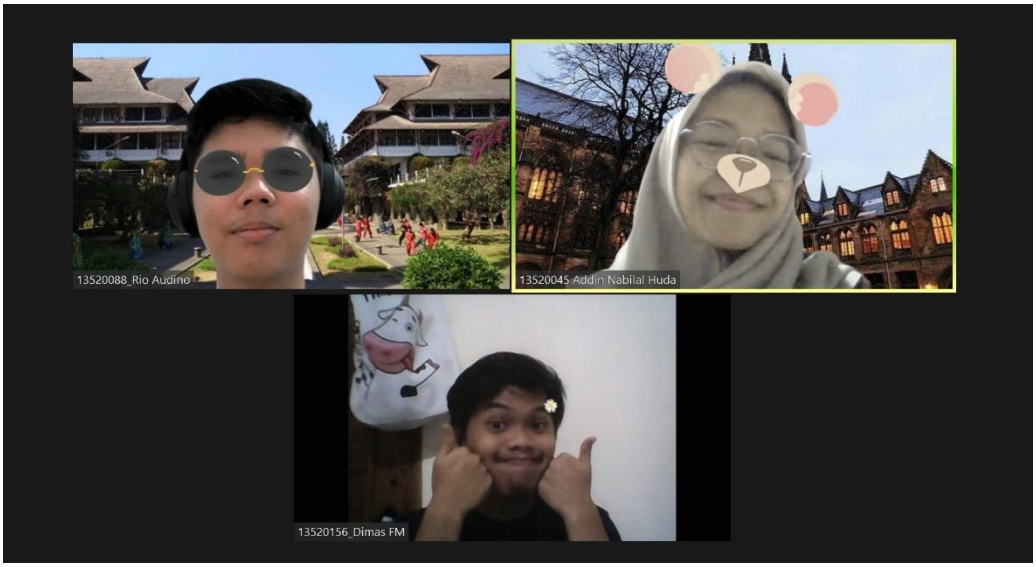
LAPORAN TUGAS BESAR

PEMANFAATAN ALGORITMA *GREEDY* DALAM APLIKASI PERMAINAN "OVERDRIVE"



Disusun oleh:

13520045 Addin Nabilal Huda

13520088 Rio Alexander Audino

1320156 Dimas Faidh Muzaki

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

Daftar Isi

Daftar Isi 1
BAB 1 DESKRIPSI TUGAS 2
BAB 2 LANDASAN TEORI 4
2.1 Algoritma Greedy 4
2.2 Alur Program 4
BAB 3 APLIKASI STRATEGI GREEDY 6
3.1 Mapping Persoalan 6
3.2 Alternatif Solusi Greedy 7
3.3 Analisis Efisiensi & Efektivitas dari Alternatif Solusi *Greedy* yang Dirumuskan 9
3.4 Strategi yang Digunakan 10
BAB 4 IMPLEMENTASI DAN PENGUJIAN 14
4.1 Implementasi Algoritma Greedy 14
4.2 Pseudo Code 16
4.3 Struktur Data 25
4.4 Studi Analisis Implementasi Algoritma Greedy 27
BAB 5 KESIMPULAN DAN SARAN 28
DAFTAR PUSTAKA 29
LAMPIRAN 30

BAB 1
DESKRIPSI TUGAS

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

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Overdrive*. *Game engine* dapat diperoleh pada laman berikut:

https://github.com/EntelectChallenge/2020-Overdrive.

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan **strategi greedy** untuk memenangkan permainan. Untuk mengimplementasikan bot tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter-bots* di dalam *starter-pack* pada laman berikut ini:

https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh *game engine Overdrive* pada tautan di atas. Beberapa aturan umum adalah sebagai berikut.

1. Peta permainan memiliki bentuk array 2 dimensi yang memiliki 4 jalur lurus. Setiap jalur dibentuk oleh *block* yang saling berurutan, panjang peta terdiri atas 1500 *block*. Terdapat 5 tipe *block*, yaitu *Empty*, *Mud*, *Oil Spill*, *Flimsy Wall*, dan *Finish Line* yang masing-masing karakteristik dan efek berbeda. *Block* dapat memuat *powerups* yang bisa diambil oleh mobil yang melewati *block* tersebut.
2. Beberapa *powerups* yang tersedia adalah:
    a. *Oil item*, dapat menumpahkan oli di bawah mobil anda berada.
    b. *Boost*, dapat mempercepat kecepatan mobil anda secara drastis.

    c. *Lizard*, berguna untuk menghindari *lizard* yang mengganggu jalan mobil anda.
    d. *Tweet*, dapat menjatuhkan truk di *block* spesifik yang anda inginkan.
    e. *EMP*, dapat menembakkan *EMP* ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 *lane* yang sama) akan terus berada di *lane* yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.
3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 *block* untuk setiap *round*. *Game state* akan memberikan jarak pandang hingga 20 *block* di depan dan 5 *block* di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.
4. Terdapat *command* yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan *powerups*. Pada setiap *round*, masing-masing pemain dapat memberikan satu buah *command* untuk mobil mereka. Berikut jenis-jenis *command* yang ada pada permainan:
    a. *NOTHING*
    b. *ACCELERATE*
    c. *DECELERATE*
    d. *TURN_LEFT*
    e. *TURN_RIGHT*
    f. *USE_BOOST*
    g. *USE_OIL*
    h. *USE_LIZARD*
    i. *USE_TWEET <lane> <block>*
    j. *USE_EMP*
    k. *FIX*
5. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.
6. Bot pemain yang pertama kali mencapai garis *finish* akan memenangkan pertandingan. Jika kedua bot mencapai garis *finish* secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar.

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* merupakan metode yang popular dan sederhana untuk memecahkan persoalan optimasi [1]. Algoritma *greedy* menyelesaikan masalah ke dalam bentuk langkah-langkah. Pada tiap langkah, algoritma *greedy* memilih satu pilihan yang dianggap paling baik untuk langkah tersebut sesuai dengan strategi yang digunakan. Tiap pilihan yang diambil dianggap sebagai optimum lokal dengan harapan akan membentuk solusi optimum global.

Dalam algoritma *greedy*, terdapat elemen-elemen sebagai berikut:

1. Himpunan kandidat, (C)
   Himpunan kandidat C berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, (S)
   Himpunan ini berisi kandidat yang sudah dipilih.
3. Fungsi solusi.
   Fungsi solusi berguna untuk menentukan apakah himpunan kandidat yang dipilih sudah membentuk solusi.
4. Fungsi seleksi (*selection function*)
   Fungsi ini memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
5. Fungsi kelayakan (*feasible*).
   Kandidat yang dipilih akan diuji oleh fungsi kelayakan sebelum dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif.
   Fungsi obyektif merupakan tujuan dari algoritma *greedy*, yaitu memaksimumkan atau meminimumkan.

Berdasarkan elemen elemen di atas, algoritma *greedy* dapat disimpulkan bahwa, algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

2.2 Alur Program

Untuk menjalankan permainan, kita cukup mengeksekusi perintah run.bat yang sudah tersedia di dalam starter-pack. Namun sebelum itu, kita perlu menyesuaikan file game-config.json dan game-runner-config.json apabila ingin mengubah konfigurasi permainan yang dijalankan. Dengan file game-config.json, kita dapat mengatur panjang lintasan balapan dan jumlah lane yang digunakan. Akan tetapi, di game-runner-config.json, kita dapat menentukan bot yang akan ditandingkan. Hal tersebut dapat dilakukan dengan cara menuliskan path folder bot terletak.

Folder bot sekurang-kurangnya berisi sebuah file bernama bot.json. bot.json memberi tahu game engine beberapa informasi penting seperti nama file bot, bahasa pemrograman bot, lokasi file bot, dan nickname yang digunakan bot di dalam permainan. Untuk bot berbahasa java, file bot yang diterima oleh game engine adalah file jar hasil kompilasi kode java.

Selama permainan, game engine berkomunikasi dengan kedua bot. Komunikasi dilakukan untuk setiap ronde selama balapan belum berakhir. Bot menerima informasi dari game engine berupa state round. Sedangkan game engine menerima masukan dari bot berupa command yang dipilih untuk melanjutkan balapan. Command yang diterima game engine berupa pesan yang ditulis ke konsol sesuai dengan format yang tersedia. Balapan akan dinyatakan selesai apabila salah satu bot telah sampai di garis *finish*.

Seperti yang disebutkan sebelumnya, bot akan diberikan state round pada setiap rondenya. Dan untuk setiap ronde itu juga, bot kami akan mengubah state round yang berupa file json menjadi objek yang lebih mudah diproses. Bot memiliki sebuah kelas bernama “bot” yang didalamnya terdapat kumpulan *method* yang menjadi logika robot dalam memilih *command* yang akan digunakan. Tak lupa, di logika tersebutlah algoritma greedy diimplementasikan. Algoritma *greedy* diimplementasikan dengan cara membobotkan setiap *command* sesuai dengan keadaan ronde. *Command* dengan bobot poin tertinggi pada akhirnya akan digunakan untuk ronde tersebut.

BAB 3
APLIKASI STRATEGI GREEDY

3.1 Mapping Persoalan

Elemen Algoritma *Greedy*	Pemetaan pada Permainan *Overdrive*
Himpunan Kandidat (C)	Perintah-perintah yang disediakan oleh permainan. Perintah-perintah tersebut mencakup *NOTHING* untuk tidak melakukan apa-apa, *ACCELERATE* untuk menambah kecepatan, *DECELERATE* untuk mengurangi kecepatan, *TURN_LEFT* untuk berpindah *lane* ke kiri, *TURN_RIGHT* untuk berpindah *lane* kanan, *USE_BOOST* untuk memakai *powerup boost*, *USE_OIL* untuk memakai *powerup oil*, *USE_LIZARD* untuk memakai *powerup lizard*, *USE_TWEET* <*lane*> <*block*> untuk memakai *powerup tweet*, *USE_EMP* untuk memakai *powerup emp*, dan *FIX* untuk memperbaiki mobil. Namun, masing-masing *command* akan diboboti sesuai dengan kondisi ronde saat permainan agar kompatibel dengan strategi *greedy* yang dipakai. Pembobotan dibagi menjadi 2 jenis, yaitu pembobotan *lane* dan pembobotan *powerup*.
Himpunan Solusi (S)	Secara eksplisit, himpunan solusi tidak terdapat dalam program. Secara nalar, himpunan solusi, S, merupakan himpunan yang berisi semua *command* yang dijalani dari awal hingga game selesai. Himpunan tersebut berisikan *command-command* yang mengantarkan *bot* dari titik *start* sampai melewati garish *finish.*
Fungsi Solusi	Fungsi solusi tidak diterapkan oleh *bot*, melainkan diterapkan oleh *game engine.* Pada setiap pergantian ronde, game akan mengecek keadaan kedua mobil. Hasil fungsi solusi bernilai benar apabila salah satu atau kedua mobil telah sampai di garis finish. Hal ini berarti, permainan sudah selesai dan dapat ditentukan pemenangnya.
Fungsi Seleksi (*Selection Function)*	Memilih *command* berdasarkan hasil perhitungan poin yang sudah dilakukan. Pada *bot* kami, fungsi seleksi diterapkan pada fungsi *choosingLane.* Fungsi tersebut akan memilih *lane* yang paling baik untuk dilalui oleh mobil. Alhasil, keluaran dari fungsi seleksi ini adalah keputusan mobil untuk tetap di jalur yang sedang dilalui atau

	pindah jalur dengan cara *TURN_LEFT* atau *TURN_RIGHT*. Tak sampai disitu, *bot* memiliki strategi tambahan apabila pada tahap sebelumnya memilih untuk tetap di jalur yang sama. *Bot* akan menyeleksi *command-command* yang mungkin dilakukan apabila tidak berpindah jalur, seperti memakai *powerup* dan *ACCELERATE*. Selain berdasarkan poin, kami juga memiliki seleksi berdasarkan kondisi mobil. Kami cenderung akan memperbaiki mobil apabila kondisi mobil sudah cukup rusak. Hal ini dilakukan demi menjaga speed maksimum yang dapat dicapai.
Fungsi Kelayakan (*Feasibility)*	Fungsi kelayakan akan memeriksa kesanggupan *command* untuk dilakukan. Sebagai contoh, *command powerup* tertentu tidak akan bisa dipakai apabila keadaan bot pada saat itu tidak memiliki *powerup* yang dibutuhkan. Fungsi kelayakan juga diterapkan untuk keputusan pindah *lane.* Mobil tidak akan berbelok apabila sedang berada di pingir *map* karena tidak mungkin untuk dilakukan dan bisa berakibat pada pengurangan *point*.
Fungsi Objektif	Memenangkan permainan *overdrive*, baik dengan cara menjadi yang pertama dalam mencapai garis finish, memaksimalkan kecepatan, maupun dengan mengumpulkan *point* sebanyak-banyaknya dengan cara menghindari penalti.

3.2 Alternatif Solusi Greedy

Terdapat beberapa alternatif untuk mengimplementasikan strategi *greedy* pada permainan *Overdrive* untuk mencapai objektif dari permainan ini, yaitu untuk mencapai garis *finish* lebih awal, mencapai garis *finish* secara bersamaan, tetapi dengan kecepatan lebih besar atau, atau memiliki poin skor terbesar bila kedua komponen tersebut berimbang. Berikut merupakan penjabaran alternatif-alternatif yang bisa diimplementasikan:

a. *Greedy* pada kecepatan

Pergerakan mobil pada permainan ini sangat ditentukan oleh kecepatan mobil yang berubah-ubah sesuai *command* dan kondisi *block* yang dilewati. Dengan memanfaatkan c*ommand-command* yang tersedia, mobil dapat dipercepat, diperlambat, atau menggunakan *powerups.*

Untuk mencapai garis *finish* terlebih dahulu, salah satu strategi yang dapat digunakan adalah menjaga kecepatan mobil tetap dalam MAX_SPEED di angka 9 atau BOOST_SPEED di angka 15 sehingga dapat melewati jarak maksimal pada setiap *round*. Hal ini dapat dilakukan dengan memanfaatkan *command* USE_BOOST dan

ACCELERATE setiap kali memenuhi kondisi yang diperlukan.

Karena keadaan *damage* pada mobil memengaruhi kecepatan maksimum MAX_SPEED, maka bila *damage* mobil >0, *command* yang diprioritaskan adalah *command* FIX. *Command* tersebut akan dijalankan hingga *damage* mobil kembali di angka 0.

Jika dipastikan *damage* mobil sudah 0, maka mobil dapat berjalan hingga batas maksimum MAX_SPEED di angka 9. Maka, prioritas selanjutnya adalah menggunakan *command* USE_BOOST bila mobil memiliki *powerup* BOOST. Bila mobil tidak memiliki *powerup* BOOST dan mobil tidak sedang berada pada kecepatan MAX_SPEED, *command* yang menjadi prioritas adalah ACCELERATE sehingga mobil dapat terjaga pada kecepatan maksimum.

b. *Greedy* paada pengambilan dan penggunaan *powerups*

Pada game *Overdrive* ini, terdapat beberapa powerup yang dapat digunakan untuk memberikan efek tertentu pada mobil dan mobil lawan. Powerup dapat digunakan bila mobil memiliki powerup yang otomatis terambil dari *block-block* yang dilewatinya.

Salah satu strategi yang dapat digunakan adalah untuk mengambil dan memanfaatkan powerup seoptimal mungkin. Pada awal ronde, akan dihitung jumlah powerup yang tersimpan padaa setiap *lane* yang mungkin dipilih. *Lane* yang dapat dipilih adalah *lane* sebelah kanan dari *current lane*, *lane* sebelah kiri dari *current lane*, dan *current lane* itu sendiri. Karena terdapat kemungkinan pemain tidak dapat melakukan TURN_LEFT atau TURN_RIGHT (ketika mobil sudah berada di *lane* paling kiri atau paling kanan), maka kondisi indeks ketiga lane tersebut perlu dicek terlebih dahulu untuk menghindari akses indeks *lane* yang *out of bound.* Setelah dilakukan pengecekan indeks *lane*, akan dihitung jumlah *powerup* yang tersedia pada *lane* yang dapat diakses. Jumlah *powerup* per *lane* akan disimpan dalam *list of integer* dengan indeks *list* sekaligus menggambarkan *indeks* lane. Kemudian, dilakukan pengecekan *lane* dengan jumlah *powerup* terbanyak dengan cara membandingkan nilai-nilai pada *list of integer*.

Sebagai *improvement*, dapat diterapkan sistem poin untuk me-*ranking* setiap *powerup* yang dapat diambil. Bila diterapkan *improvement* ini, *lane* yang dipilih adalah *lane* dengan poin terbesar yang ditinjau dari segi manfaat dan jumlah *powerup.*

Setelah didapatkan *lane* dengan prioritas terbesar, akan ditentukan *command* yang akan digunakan mobil. Bila *lane* dengan prioritas terbesar terdapat pada *lane* kiri atau kanan, maka *command* yang dipilih adalah TURN_LEFT atau TURN_RIGHT. Bila *current lane* merupakan *lane* dengan prioritas pertama, maka kemungkinan *command-command* yang digunakan adalah *command* untuk menggunakan *powerup*, *command* ACCELERATE, *command* DECCELERATE, dan *command* DO NOTHING. Karena kita akan menerapkan strategi *Greedy* pada *powerup*, maka *command* untuk menggunakan *powerup* akan menjadi prioritas utama. Untuk memilih *powerup* yang akan digunakan dari daftar *powerup* yang tersedia, dapat diterapkan sistem poin untuk me-*ranking* setiap *powerup* berdasarkan besar manfaatnya. *Powerup* dengan prioritas terbesar(bila tersedia) akan dipilih terlebih dahulu.

c. *Greedy* paada penghindaran *obstacles*

Obstacle merupakan objek pengganggu yang dapat menurunkan kecepatan mobil dan meningkatkan *damage* pada mobil. Adanya *damage* pada mobil dapat berpengaruh pada penurunan kecepatan maksimal yang dapat dicapai mobil. Akibatnya, mobil akan semakin lambat dalam mencapai garis *finish* bila terganggu *obstacles*.

Berangkat dari hal tersebut, strategi yang dapat digunakan adalah strategi memilih

command untuk dapat menghindari *obstacle* sebaik mungkin. Pada awal ronde, akan dihitung jumlah *obstacle* yang tersimpan padaa setiap *lane* yang mungkin dipilih. *Lane* yang dapat dipilih adalah *lane* sebelah kanan dari *current lane*, *lane* sebelah kiri dari *current lane*, dan *current lane* itu sendiri. Karena terdapat kemungkinan pemain tidak dapat melakukan TURN_LEFT atau TURN_RIGHT (ketika mobil sudah berada di *lane* paling kiri atau paling kanan), maka kondisi indeks ketiga lane tersebut perlu dicek terlebih dahulu untuk menghindari akses indeks *lane* yang *out of bound.* Setelah dilakukan pengecekan indeks *lane*, akan dihitung jumlah *powerup* yang tersedia pada *lane* yang dapat diakses. Jumlah *obstacle* per *lane* akan disimpan dalam *list of integer* dengan indeks *list* sekaligus menggambarkan *indeks* lane. Kemudian, dilakukan pengecekan *lane* dengan jumlah *obstacle* paling sedikit dengan cara membandingkan nilai-nilai pada *list of integer*.

Sebagai *improvement*, dapat diterapkan sistem poin untuk me-*ranking* setiap *obstacle* yang dapat mengganggu. *Obstacle* yang mengakibatkan kerusakan dan pengurangan kecepatan yang lebih besar akan memiliki prioritas penghindaran yang lebih besar pula. Bila diterapkan *improvement* ini, *lane* yang dipilih adalah *lane* dengan poin terbesar yang ditinjau dari segi prioritas penghindaran *obstacle.*

Setelah didapatkan *lane* dengan prioritas terbesar, akan ditentukan *command* yang akan digunakan mobil. Bila *lane* dengan prioritas terbesar terdapat pada *lane* kiri atau kanan, maka *command* yang dipilih adalah TURN_LEFT atau TURN_RIGHT. Bila *current lane* merupakan *lane* dengan prioritas pertama, maka terdapat kemungkinan *command* yang digunakan adalah *command* untuk menggunakan *powerup*, *command* ACCELERATE, *command* DECCELERATE, dan *command* DO NOTHING. Karena strategi penghindaran *obstacle* ini juga mencakup strategi untuk membuat mobil lawan sebanyak mungkin terkena *obstacle*, maka prioritas *command* yang selanjutnya dipilih adalah *command* untuk menggunakan *powerup* yang mengganggu mobil lawan. Dapat diterapkan sistem poin untuk menentukan prioritas penggunaan *command* dengan poin terbesar merupakan poin untuk *command* yang memberikan kerugian paling besar pada mobil lawan.

3.3 Analisis Efisiensi & Efektivitas dari Alternatif Solusi *Greedy* yang Dirumuskan

No	Nama Strategi	Efisiensi	Efektivitas
1	Greedy pada Kecepatan	- *Best case:* O(1), terjadi ketika damage mobil > 0 yang membuat program langsung memilih command FIX untuk mengurangi *damage* mobil - *Worst case*: O(n), terjadi ketika semua kondisi tidak ada yang terpenuhi. Mobil tidak memiliki prioritas gerakan lain dan pada akhirrnya menjalaankan DO NOTHING	Strategi ini tidak efektif untuk digunakan pada keadaan *map* yang kosong tanpa ada *obstacle* maupun *powerup*. Bila terdapat *obstacle* di *current lane* dan mobil memilih BOOST/ACCELERATE, kecepatan mobil akan berkurang dan *damage* meningkat sehingga percepatan tidak dapat dilakukan terus menerus. Sedangkan, bila terdapat *powerup* yang lebih banyak di

			lane kanan/kiri, *powerup* tersebut tidak dapat terambil secara optimal. *Powerup* selain BOOST yang dimiliki juga tidak bisa terpakai dengan baik karena memiliki prioritas rendah.
2	Greedy pada Powerup (tanpa improvement)	- *Best case:* O(n), terjadi ketika jumlah *powerup* terbanyak ada pada *lane* kiri atau kanan, sehingga program tidak perlu memeriksa kemungkinan penggunaan *command* untuk menggunakan *powerup* - *Worst case:* O(n), terjadi ketika program perlu memeriksa kemungkinan penggunaan *command* yang digunakan untuk menggunakan *powerup*,yaitu ketika *current lane* menyimpan *powerups* terbanyak	Strategi ini tidak efektif untuk digunakan pada keadaan *map* dengan banyak *obstacle* karena tidak ada strategi penghindaran *obstacle* yang diimplementasikan. Akibatnya, bila *lane* yang memiliki banyak *powerups* dipilih, tetapi *lane* tersebut mengandung banyak *obstacle,* kecepatan mobil akan berkurang dan *damage* akan meningkat. Bahkan, bila tidak diterpakan strategi perbaikan mobil, kecepatan mobil bisa turun ke angka 0 bila *damage* mobil sudah berada di angka 5.
3	Greedy pada Obstacle	- *Best case:* O(n), terjadi ketika jumlah *obstacle* tersedikit ada pada *lane* kiri atau kanan, sehingga program tidak perlu memeriksa kemungkinan penggunaan *command* untuk menggunakan *powerup (*bila jumlah *obstacle* tersedikit ada pada *current lane)* - *Worst case:* O(n), terjadi ketika program perlu memeriksa kemungkinan penggunaan *command* yang digunakan untuk menggunakan *powerup,* yaitu ketika *current lane* menyimpan jumlah *obstacle* paling sedikit	Strategi ini efektif digunakan pada keadaan *map* yang memiliki banyak *obstacle* karena dapat menjaga kestabilan kecepatan mobil. Namun, strategi ini tidak memanfaatkan *powerup* dengan baik. Maka, besar kemungkinan mobil akan kalah bila lawan dapat memanfaatkan *powerup* dengan baik.

3.4 Strategi yang Digunakan

Berdasarkan analisis dari berbagai alternatif solusi *greedy* yang dapat digunakan, kami mencoba untuk mengombinasikan ketiga strategi yang sudah dijabarkan sebelumnya untuk memaksimalkan kelebihan dan meminimalkan kekurangan dari setiap strategi. Algoritma *greedy* yang kami pilih adalah algoritma yang dapat memilih *command* dengan

mempertimbangkan kecepatan, *damage* mobil, *powerup* yang dapat digunakan, *powerups* yang dapat diambil, serta *obstacle* yang dapat merugikan.

Untuk dapat mencapai objektif tersebut, kami menggunakan sistem poin. Sistem poin tersebut dapat me-*ranking* setiap *command* untuk pada akhirnya dapat dipilih mobil untuk digunakan secara optimal. Penentuan prioritas dengan menggunakan sistem poin yang kami gunakan tetap mempertimbangkan keadaan mobil dan *blocks* di depan mobil saat ini.

Desain strategi ini dimulai dengan mengecek *damage* mobil. Pengecekan *damage* diprioritaskan karena *damage* memengaruhi kecepatan maksimal mobil sehingga untuk menjaga kecepatan mobil tetap tinggi, *damage* mobil harus diperbaiki terlebih dahulu dengan menggunakan *command* FIX.

Bila kondisi yang mengharuskan penggunaan *command* FIX pada program tidak terpenuhi, program akan lanjut melakukan pengecekan terhadap kecepatan mobil. Bila kecepatan mobil 0, *command* yang dipilih adalah ACCELERATE untuk membuat mobil kembali bergerak. Selanjutnya, penentuan *command* dilakukan menggunakan sistem poin.

Sistem pertama kali akan mengecek poin berdasarkan powerup yang dimiliki mobil. Pengecekan ini *dihandle* oleh *method* total_point_using_powerups yang akan mengecek total poin berdasarkan powerup tertentu. Setiap powerup dapat memberikan poin yang berbeda bila memenuhi kondisi tertentu. Sistem poin yang diimplementasikan pada bagian ini adalah sebagai berikut:

Powerup yang Akan Dicek	Poin	Kondisi yang Harus Dipenuhi
EMP	8	Kecepatan lawan>3, lawan berada di lane yang sama dengan pemain atau di lane kanan/kiri sebelah lane pemain, pemain berada di depan pemain
BOOST	15	Tidak ada obstacle di *current lane* pada ronde saat ini dan *damage* mobil 0
OIL	2	Lawan berada di *lane* yang sama dengan mobil dan lawan berada di belakang mobil
TWEET	5	Kecepatan lawan lebih dari 6 dan kecepatan mobil lebih dari 8
LIZARD	{}	Poin bergantung pada jumlah obstacle dan powerup di depan, adanya obstacle menambah poin dan adanya powerup mengurangi poin. Sistem ini digunakan karena dengan menggunakan LIZARD, kita akan rugi karena tidak bisa mengambil powerup dan untung karena bisa menghindarinya

Selanjutnya, program akan mengecek jumlah *obstacles* dan *powerups* di setiap lane (depan, kanan, kiri) dan menghitung poin yang didapat dengan kondisi tersebut. Sistem poin yang diterapkan mempertimbangkan jumlah powerup yang dimiliki untuk mencegah pengambilan powerup yang terlalu banyak dan *redundant.* Berikut merupakan sistem poin pada bagian ini:

Obstacle/Powerup	Poin	Kondisi yang Harus Dipenuhi
OIL POWER	1	Bila mobil belum memiliki powerup ini sebelumnya, adanya powerup OIL POWER yang dapat diambil akan menambah poin
BOOST	15	Bila mobil memiliki powerup BOOST<5, adanya powerup BOOST yang dapat diambil akan menambah poin
LIZARD	2	Bila mobil memiliki powerup LIZARD<3, adanya powerup LIZARD yang dapat diambil akan menambah poin
TWEET	3	Bila mobil memiliki powerup TWEET<3, adanya powerup TWEET yang dapat diambil akan menambah poin
EMP	3	Bila mobil memiliki powerup EMP<2, adanya powerup EMP yang dapat diambil akan menambah poin
MUD	-9	-
OIL SPILL	-9	-
WALL	-14	-

Selain itu, adanya musuh di suatu block dapat mengurangi poin. Hal ini dilakukan sebagai strategi menghindari tabrakan. Setiap EMPTY BLOCK juga akan dihitung 2 poin. Ini merupakan strategi untuk menghindari cyber truck.

Setelah powerup dan obstacle dari setiap lane sudah dicek, poin-poin tersebut akan diset sebagai poin setiap lane. Selanjutnya, untuk setiap lane akan dicek kembali kondisi-kondisi yang dapat menambah/mengurangi poin yang sudah dihitung sebelumnya. Penerapan sitem poin pada bagian ini adalah sebagai berikut:

Lane	Keadaan yang Mungkin Digunakan	Kondisi	Kontribusi terhadap Poin Sebelumnya
Current	Menggunakan powerup	Memiliki powerup	Bergantung pada poin yang didapat dari powerup yang akan digunakan (perhitungan sudah dilakukan pada step sebelumnya), ditambah 3
	Boosting	Sedang dalam kondisi boosting dan tidak ada obstacle di depan	Ditambah 20
	Accelerating	Tidak ada obstacle di depan	Ditambah 3

Kiri, kanan	Turn left, Turn right	Sedang dalam kondisi boosting dan tidak ada obstacle di depan	Ditambah 20

Setelah diketahui poin setiap lane dengan keadaan yang mungkin digunakan, program akan mengembalikan command dengan poin tertinggi berdasarkan keadaan yang memberikan poin tertinggi.

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

Pada bot yang kami kembangkan, implementasi algoritma *greedy* dapat dibagi ke dalam beberapa langkah utama, yaitu main, mengecek *damage*, mengecek *powerups*, mengecek *lanes*, dan memilih *lane*. Seperti yang telah dijelaskan sebelumnya, implementasi *greedy* yang kami lakukan menitikberatkan pada optimalisasi poin tiap aksi. Optimalisasi poin terjadi pada fungsi mengecek *powerups*, mengecek *lanes*, dan memilih *lane*. Berikut akan kami jelaskan tiap langkah utama dan fungsi terkaitnya.

a. main : *run*

main adalah proses utama yang digunakan oleh bot. Proses main terjadi pada fungsi *run*. Semua fungsi implementasi *greedy* akan disatukan di dalam fungsi utama *run*. Fungsi *run* akan me-*return* sebuah *command* yang menjalankan bot. Di dalam fungsi *run* kita dapat mengatur prioritas/urutan fungsi-fungsi lainnya. Berikut proses yang dikerjakan fungsi *run*,

- Menginisiasi object player(*myCar*) dan opponent(*opponent*)
- Mengecek damage
- Mengecek speed
- Mengecek total poin dari *powerups* player
- Mengecek total poin dari tiap *lanes* dan *powerups*
- Mencari pilihan *Command*(*lanes*/*powerups*) terbaik dari jumlah point
- Memanggil *Command* terbaik dari hasil kalkulasi poin sebelumnya

fungsi di dalam *source-code :* *run*

b. Mengecek *damage*

Mengecek *damage* adalah proses yang menganalisis kerusakan mobil. Proses mengecek damage terjadi di dalam fungsi *damage_check*. Konsep *greedy* yang kita gunakan adalah untuk meminimalisir damage yang terjadi sesuai dengan kecepatan maksimumnya. Berikut proses yang dikerjakan fungsi damage_check

- Mengecek kecepatan maksimum mobil berdasarkan damagenya
- Memebandingkan kecepatan sekarang dan kecepatan maksimum yang mungkin, serta mengecek kepemilikan *powerups* boost.
- Mengembalikan true jika mobil perlu dilakukan perbaikan

fungsi di dalam *source-code* : *damage_check*

c. Mengecek *powerups*

Mengecek *powerups* adalah proses mencari *command powerups* terbaik yang harus diambil berdasarkan pemetaan poin yang dilakukan. Proses mengecek *powerups* terjadi pada fungsi *get_total_points_using_powerups*. Konsep greedy yang kita gunakan adalah memilih *Command Powerups* dengan total poin tertinggi. Total poin yang dimaksud adalah total perolehan

poin dari tiap *obstacle* yang akan dilewati atau jarak yang akan bisa ditempuh. Pemetaan poin terdapat pada bab … . Fungsi ini akan mengembalikan jenis command beserta total poin yang diperoleh. Berikut proses yang dikerjakan fungsi *get_total_points_using_powerups*,

- Menghitung poin tiap jenis command (nol jika tidak memiliki powerups terkait)
- Mencari command dengan poin tertinggi
- Mengembalikan command serta poin yang diperolehnya (*null* jika tidak memiliki *powerups*)

fungsi di dalam *source-code :* *get_total_points_using_powerups*

d. Mengecek *lanes*

Mengecek *lanes* adalah proses menghitung total peroleh point dari tiap lane yang dipilih (kiri/kanan/*lane* sekarang/menggunakan *powerups*). Proses mengecek lanes terjadi pada fungsi *getPointsOfAllLane*. Konsep greedy yang kita terapkan adalah konfigurasi poin pada tiap object. Sama seperti proses mengecek powerups, perolehan poin dihitung berdasarkan jumlah obstacle, jarak, dan pemetaan poinnya. Fungsi ini akan mengembalikan *array of integer* yang berisikan total point tiap lane yang ada. Berikut proses yang dikerjakan fungsi *getPointsOfAllLane*,

- Menghitung poin ketika belok kiri
- Menghitung poin ketika belok kanan
- Menghitung poin jika tetap di *lane* sekarang dan menggunakan *powerup*
- Menghitung poin jika tetap di *lane* sekarang dan menggunakan *accelerate*
- Mengembalikan *array of integer* berisikan total poin tiap *lane*

fungsi di dalam *source-code : getPointsOfAllLane*

e. Memilih *lane*

Memilih *lane* adalah proses yang dilakukan untuk mencari *lane*/*command*/*powerups* terbaik dari ketiga *lane* yang ada. Proses memilih *lane* terjadi di dalam fungsi *choosingLane*. Konsep greedy yang kita gunakan adalah memilih *lane* dengan perolehan poin tertinggi. Fungsi ini akan mengembalikan *command* terbaik ke fungsi utama. Berikut proses ang diikerjakan fungsi *choosingLane*,

- Menghitung total poin *lane* dengna memanggil fungsi *getPointOfAllLane*
- Memilih lane dengan perolehan poin tertinggi
- Mengembalikan *command* sesuai *lane* yang dipilih

fungsi di dalam *source-code : choosingLane*

4.2 Pseudo Code

a. run(input Gamestate gameState) -> Command

```
function run(input GameState gameState) -> Command

{Fungsi utama program yang mengandungu seluruh proses algoritma. Fungsi akan
mengembalikan command yang akan dijalankan}

KAMUS LOKAL

        myCar, opponent : Car

        power_ups_points, lane_points : array of Integer

        COMMAND : Command

ALGORITMA

        {Instatiate Player and Opponent}

        myCar <- gameState.player

        oppopnent <- gameState.player


        {Check player's car damage}

        {avoid fixing 0 damage car, maintain damage and speed at sweet point}

        if (myCar.damage>0 and (damage_check(gameState) and myCar.speed < 15) then

                -> FIX


        {Check speed, avoid 0 speed bug}

        if (myCar.speed == 0) then

                -> ACCELERATE


        {Calculate points each powerups and lanes}

        power_ups_points <- get_total_points_using_powerups(gameState)

        lane_points <- getPointsOfAllLane(power_ups_points, gameState)

        COMMAND <- choosingLane(lane_points, powerups_points, gameState)


        {return choosed COMMAND from points calculations}

        -> COMMAND
```

b. damage_check(input GameState gameState) -> Boolean

```
function damage_check(input GameState) -> Boolean

{Fungsi yang mengecek tingkat kerusakan mobil, mengembalikan true jika mobil perlu
dibetulkan}

KAMUS LOKAL

        myCar : Car

        maxSpeed, numOfBoost : Integer

        numOfPowerups : array of Integer

ALGORITMA

        {Instantiate player}
```

```
        myCar <- gameState.player


        {Check max speed based on damage}

        maxSpeed <- max_speed_checker(myCar)


        {Count Boost Powerups}

        numOfPowerups <- getNumOfPowerups(gameState)

        numOfBoost <- numOfPowerups[1]


        {To maximize efficiency, we need to keep speed up to their possible max speed.
        The max speed is based on damage (according to the rule). We also need to
        consider number of boost we have}

        {From our calcultions, 8 is a sweet spot for speed ^~^}

        -> (myCar->speed == maxSpeed and (maxSpeed < 8 or numOfBoost > 0))
```

c. getRandomNumber(input int min, int max) -> Integer

```
function getRandomNumber(input integer min, integer max) -> Integer

{Fungsi ini akan mengembalikan sebuah integer secara acak, antara min dan maxnya}

KAMUS LOKAL

ALGORITMA

        -> (Math->random() * (max-min)) + min
```

d. total_points_using_powerups(input PowerUps powerUpToCheck, input GameState gameState) -> Int

```
function total_points_using_powerups(input Powerups powerUpToCheck, input GameState
gameState) -> Integer

{Fungsi ini akan mencari total poin setiap powerups yang dimiliki player. Fungsi akan
mengembalikan sebuah array of integer yang berisikan jumlah poin tiap powerups
(bernilai 0 jika player tidak memiliki powerup tersebut)

KAMUS LOKAL

        myCar, opponent : Car

        point, diff : Integer

ALGORITMA

        {Instantiate player and opponent)

        myCar <- gameState.player

        opponent <- gameState.opponent


        {Instantiate point}

        point <- 0
```

```
{Checking each powerups}

{Creating conditional based on powerUpToCheck. The conditional is created with
switch-case because the program is made with java}

switch(powerUpToCheck}:

        case EMP:

                {Valid if opponent in the EMP Range and in front of player}

                diff <- abs(opponent.position.lane – myCar.posisition.lane}

                if ({opponent is in range and not slow}) then

                        point <- point + 8 //point for using EMP

        case BOOST:

                {Valid if there is no obstacle ahead of player and player have 0
                damage}

                if ({no obstacle ahead and player car damage == 0}) then

                        point <- point + 15 // point for using BOOST

        case OIL:

                {Only valid the opponent is in the same lane as the player and
                the player ahead near the oppopnent}

                if ({opponent in the OIL range}) then

                        point <- point + 2 // point for using OIL

        case TWEET:

                {Only valid if player at speed and oppent is not slow}

                {This decision is to avoid using multiple TWEET in a row}

                if (myCar.speed > 8 and opponent.speed > 6) then

                        point <- point + 5 // point for using TWEET

        case LIZARD:

                {Only valid if player want to jump over obstacles/opponent and
                the landing block is not occupied by obstacles}

                if ({landing block is clearr of obstacles}) then

                        {If the landing block is clear, we need to calculate the
                        profit of this powerups. Each obstacles we avoid have 3
                        extra points, each powerups we missed have 1 deduction
                        point}

                        point<- point + {bonus point from avoiding obstacles}

                        point<- point – {deduction point from skipping powerups}


{end of switch case}

{Validating player's powerups, zero point if player doesn't have the respective
powerups}

if ({player has powerups}) then

        -> points

else

        >   0
```

e. contains_obstacle(input Array of Object blocks) -> Boolean

```
function contains_obstacle(input Arrary of Object blocks) -> Boolean
{Fungsi ini akan mengembalikan true jika blocks mengandung obstacle}
KAMUS LOKAL
ALGORITMA
        -> blocks.contains(Terrain.MUD) or blocks.contains(Terrain.OIL_SPILL) or
        blocks.contains(Terrain.WALL)
```

f. get_total_points_using_powerups(input GameState gameState) -> Array of Integer

```
function get_total_points_using_powerups(input GameState gameState)-> Array of Integer
{Fungsi ini akan mengembalikan true jika blocks mengandung obstacle}

KAMUS LOKAL
        points_using_powerups, powerups_to_user : array of Integer
        command, max_points : integer
ALGORITMA
        {Instantiate list of integer}
        points_using_powerups = [0, 0, 0, 0, 0]

        {Calculating scores of each powerups and store it inside array}
        points_using_powerups[0]<-total_point_using_powerups(PowerUps.EMP,gameState)
        points_using_powerups[1]<-total_point_using_powerups(PowerUps.BOOST,gameState)
        points_using_powerups[2]<-total_point_using_powerups(PowerUps.OIL,gameState)
        points_using_powerups[3]<-total_point_using_powerups(PowerUps.LIZARD,gameState)
        points_using_powerups[4]= total_point_using_powerups(PowerUps.TWEET,gameState)

        {Check the best powerups based on points, by comparing each points in the array
        (skipped the code, has no specific implementation}}
        command <- {Command for the best powerups}
        max_points <- {Points from the best powerups}

        {Return command with their points, null if there is no choosen command}
        powerups_to_use = [0, 0]
        if ({command valid & avaible}) then
                powerups_to_use[0] <- max_points //points
                powerups_to_use[1] <- command //command
        else
                powerups_to_use[0] <- NULL
```

```
        powerups_to_use[1] <- NULL
    {Returning powerups_to_use}
    -> powerups_to_use
```

g. hasPowerUp(input PowerUps powerUpToCheck, input PowerUps[] avaible) -> Boolean

```
function hasPowerUp(input PowerUps powerUpToCheck, input array of PowerUps avaible) -> Boolean
{Fungsi ini akan mengembalikan true jika powerUpToCheck ada di dalam list PowerUps}
KAMUS LOKAL
ALGORITMA
    {Iterating through list of powerups and comparing with powerUpToCheck}
    for (PowerUps powerUp in avaible) do
        if (powerUp == powerUpToCheck) then
            -> true
    {End of loop}
    -> false
```

h. use_powerups(input int commandNum) -> Command

```
function user_powerups(input integer commandNum) -> Command
{Fungsi ini akan mengembalikan command yang sesuai dengan nomor command }
KAMUS LOKAL
ALGORITMA
    {Checking the commandNum and return the respective command}
    {Creating conditional based on commandNum. The conditional is created with
    switch-case because the program is made with java}
    switch(commandNum) :
        case 1 :
            -> EMP
        case 2 :
            -> BOOST
        case 3 :
            -> OIL
        case 4 :
            -> LIZARD
        case 5 :
            -> TWEET_COMMAND
        default :
            -> ACCELERATE
```

```
        {End of switch case}
```

i. getNumofPowerups(input GameState gameState) -> Array of Integer

```
function getNumofPowerups(input GameState gameState) -> Array of Integer

{Fungsi ini akan mengembalikan array of integer yang berisikan jumlah tiap jenis
powerups}

KAMUS LOKAL

        myCar : Car

        NumOfPowerUps : array of integer

ALGORITMA

        {Instatiate player}

        myCar <- gameState.player


        {Instantiate array}

        NumOfPowerUps = [0, 0, 0, 0, 0]


        {Count each powerups the player have}

        for (Object powerup in myCar.powerups) do

                if (powerup == OIL) then

                        NumOfPowerUps[0] <- NumOfPowerUps[0] + 1

                if (powerup == BOOST) then

                        NumOfPowerUps[1] <- NumOfPowerUps[1] + 1

                if (powerup == LIZARD) then

                        NumOfPowerUps[2] <- NumOfPowerUps[2] + 1

                if (powerup == TWEETT) then

                        NumOfPowerUps[3] <- NumOfPowerUps[3] + 1

                if (powerup == EMP) then

                        NumOfPowerUps[4] <- NumOfPowerUps[4] + 1

        {End Of Loop, returning number of each powerups}

        -> NumOfPowerUps
```

j. max_speed_check(input Car myCar) -> Integer

```
function max_speed_check(input Car myCar) -> Integer

{Fungsi ini akan mengembalikan kecepatan maksimum yang mungkil dari mobil player
berdasarkan damagenya}

KAMUS LOKAL

        maxSpeed : integer

ALGORITMA

        {Creating conditional based on player's damage. The conditional is created with
        switch-case because the program is made with java}
```

```
switch(myCar.damage):
        case 5 :
                maxSpeed <- 0
        case 4 :
                maxSpeed <- 3
        case 3 :
                maxSpeed <- 6
        case 2 :
                maxSpeed <- 8
        case 1 :
                maxSpeed <- 9
        case 0 :
                maxSpeed <- 15
        default :
                maxSpeed < 5
-> maxSpeed
```

k. current_speed_if(input Car myCar, input Command command) -> Integer

```
function current_speed_if(input Car myCar, input Command command) -> Integer
{Fungsi ini berguna untuk memprediksi kecepatan mobil bot pada 1 ronde kedepan
apabila pada ronde sekarang bot memilih command tertentu}

KAMUS LOKAL
    currSpeed : integer { merupakan kondisi speed mobil nanti}
    maxSpeed : integer { kecepatan maksimum yang dapat dicapai oleh mobil untuk
kondisi Kesehatan mobil saat ini }

ALGORITMA
    currSpeed, maxSpeed <- max_speed_check(myCar)
    if (myCar.speed = 15) then
            {15 adalah kecepatan maksimum yang diizinkan oleh game}
            currSpeed <- 15
    else if (command = ACCELERATE) then
            {currSpeed akan berubah ke tahapan speed selanjutnya selama currSpeed
            kurang dari kecepatan maksimum, ada 7 tahapan speed yaitu 0, 3, 5, 6,
            8, 9, 15}
    elsei if (jika memilih berbelok) then
            currSpeed <- currSpeed – 1

    ->      currSpeed
```

l. getPointsFromList(input Array of Integer pointsPerLane, input Array of Integer numOfPowerups) -> Array of Integer

```
function getPointsFromList(input Array of Integer pointsPerLane, input Array of
```

```
Integer numOfPowerips) -> Array of Integer
{Fungsi ini menghitung poin dari dari apa saja yang terdapat di jalur,
pointsPerLane berisi jumlah hal hal yang ada di jalur sedanggkan numOfPowerUps
merupakan powerup yang dimiliki  }
KAMUS
    Point: integer
    Obstacle: integer
ALGORITMA
    {pada bagian ini akan dikonversi keberadaan powerup menjadi poin. Namun, kami
    menetapkan angka angka yang menjadi batas powerup agar tidak mengambil powerup
    terlalu banyal}

    If (punya oil < 0) then
            {kami tidak menggunakan oil karena tidak terlalu beneficial}
    If (punya boost < 5) then
            {15 adalah poin untuk powerup boost}
            Point <- point + <jumlah boost di lane> * 15
    If (punya lizard < 3) then
            {2 adalah poin untuk powerup boost}
            Point <- point + <jumlah boost di lane> * 2
    If (punya tweet < 3) then
            {3 adalah poin untuk powerup boost}
            Point <- point + <jumlah boost di lane> * 3
    If (punya boost < 2) then
            {3 adalah poin untuk powerup boost}
            Point <- point + <jumlah boost di lane> * 3

    {selanjutnya adalah pengurangan point apabila terdapat rintangan}
    Point <- Point + jumlah mud di lane * -9
    Point <- Point + jumlah oil spill di lane * -9
    Point <- Point + jumlah wall di lane * -14
    Point <- Point + jumlah enemy di lane * -12
    Point <- Point + jumlah block kosong di lane * 2

    Obstacle <- Obstacle + jumlah mud di lane * -7
    Obstacle <- Obstacle + jumlah oil spill di lane * -7
    Obstacle <- Obstacle + jumlah wall di lane * -12
    Obstacle <- Obstacle + jumlah enemy di lane * -12

    Points <- [point, obstacle]
    -> points;
```

m. getPointsOfAllLane(input Array of Integer power_ups_points, GameState gameState) -> Array of Integer

```
function getPointsOfAllLane(input Array of Integer power_ups_points, GameState
gameState) -> Array of Integer
{Fungsi ini mencari poin untuk semua lane}
KAMUS LOKAL
    myCar : Car {kondisi mobil saat ini}
    currLane : integer {posisi baris mobil}
```

```
    currBlock : integer {posisi absis mobil}
    speedIf : integer
    TURNING_POINT_REDUCTION : integer = 0
    ACCELERATE_POINT_BONUS : integer = 3 {point tambahan apabila ACCELERATE}
    BOOSTING_POINT_BONUS : integer = 20 {point tambahan apabila memakai boost}
    lanePoints : [0,0,0,0] Array of Integer
ALGORITMA
    if(mobil bisa belok kiri) then
            bonus_point <- 0
            {memeriksa kecepatan mobil apabila belok kiri}
            speedIf <- current_speed_if(…)

            {melihat block yang ada di lane sebelah kiri}
            pointsPerLane <- getNumOfBlockInFront(…)

            {menerapkan pengurangan poin karena berbelok}
            bonus_point <- bonus_point + TURNING_POINT_BONUS
            lanePoints[1] <- point untuk lane di sebelah kiri

    {langkah sebelumnya dilakukan kembali untuk belok kanan dan diset ke dalam
    lanePoints[2]}

    {lalu diulangi untuk lane tempat sekarang mobil berada dan memakai powerup lalu
    disimpan ke lanePoinst[3]}

    {lalu diulangi untuk lane tempat sekarang mobil berada dan accelerate lalu
    disimpan ke lanePoinst[0]}

    -> lanePoints
```

n. choosingLane(Array of Integer lane_points, Array of Integer power_ups_points, GameState gameState) -> Command

```
function choosingLane(input Array of Integer lane_points, input Array of Integer
power_ups_points, input GameState gameState) -> Command
{fungsi ini berguna untuk menentukan command yang akan digunakan berdasarkan point
lanes, point powerup, dan state}
KAMUS LOKAL
    maxPoint: integer
    lane: integer
ALGORITMA
    Maxpoint <- max(lane_points) {mencari point tertinggi}
    Lane <- lane_points.indexOf(Maxpoint) {mencari lane dengan point tertinggi}

    if (lane = kiri} then -> TURN_LEFT
    if (lane = kanan} then -> TURN_RIGHT
    if (lane = lurus} then -> ACCELERATE
    if (kecepatan mobil < 15) then -> use_powerups(power_ups_points.get(1))

    -> ACCELERATE
```

o. getNumOfBlockInFront(int pos_lane, int pos_block, int currSpeed, GameState gameState) -> Array of Integer

```
function getNumOfBlockInFront(input integer pos_lane, input integer pos_block,
input integer currSpeed, GameState gameState) -> Array of Integer
{fungsi ini berguna untuk mencatat jumlah block berdasarkan jenisnya yang ada di
sebuah lane}
KAMUS LOKAL
    myCar: Car
    opponent: Car
    NumOfBlockInFront: array of integer [0..11]
    Blocks: array of object
ALGORITMA
    Blocks <- getBlocksInFront(pos_lane, pos_block, gameState, currSpeed)
    If (jika kita tepat di belakang musuh) then
           NumOfBlockInFront[9] <- 1; {index 9 adalah musuh}
    i traversal in blocks do :
           if(i adalah MUD) then
                  NumOfBlockInFront[0] <- NumOfBlockInFront[0] + 1
           if(i adalah OIL_SPILL) then
                  NumOfBlockInFront[1] <- NumOfBlockInFront[1] + 1
           if(i adalah OIL_POWER) then
                  NumOfBlockInFront[2] <- NumOfBlockInFront[2] + 1

           if(i adalah FINISH) then
                  NumOfBlockInFront[3] <- NumOfBlockInFront[3] + 1
           if(i adalah BOOS) then
                  NumOfBlockInFront[4] <- NumOfBlockInFront[4] + 1

           if(i adalah WALL) then
                  NumOfBlockInFront[5] <- NumOfBlockInFront[5] + 1

           if(i adalah LIZARD) then
                  NumOfBlockInFront[6] <- NumOfBlockInFront[6] + 1
           if(i adalah TWEET) then
                  NumOfBlockInFront[7] <- NumOfBlockInFront[7] + 1
           if(i adalah EMP) then
                  NumOfBlockInFront[8] <- NumOfBlockInFront[8] + 1

           if(i adalah EMPTY) then
                  NumOfBlockInFront[9] <- NumOfBlockInFront[9] + 1
    -> NumOfBlockInFront
```

4.3 Struktur Data

Karena bot dikembangkan dengan menggunakan bahasa Java yang merupakan bahasa berbasis objek, struktur data didefinisikan sebagai kelas (*class*). Kelas utama yang didefinisikan adalah **Bot.java** yang di dalamnya berisi *method* run yang akan mengembalikan *command* yang dipilih. Adapun kelas-kelas pendukung dimuat dalam folder “optimalization”. Berikut adalah penjelasan kelas-kelas yang digunakan dalam program:

No	Nama Kelas	Penjelasan
1	ConditionChecker	Kelas ini berisi data-data kemungkinan besar kecepatan mobil yang disimpan dalam List SPEEDS dan juga instance-instance dari *command.* Terdapat method-method yang digunakan untuk mengecek kondisi mobil seperti hasPoweUp yang mengembalikan true bila mobil memiliki powerup tertentu, max_speed_check yang berfungsi untuk mengecek kecepatan maksimal berdasarkan *damage* mobil, *current_speed_if* yang berfungsi untuk menentukan kecepatan mobil bila menggunakan powerup tertentu, dan
2	BlockChecker	Kelas ini berisi method-method yang digunakan untuk mengecek keadaan *blocks* di depan pemain. Terdapat method `getNumOfBlockInFront` yang berfungsi untuk menghitung jumlah *obstacle* dan *powerup* yang terdapat pada sebuah lane bila mobil berjalan dengan kecepatan currSpeed. Method lainnya, `getBlocksInFront` berfungsi untuk menentukan *blocks-blocks* yang terdapat di depan pemain bila pemain berada di posisi (lane,block) dengan kecepatan currSpeed.
3	DamageChecker	Kelas ini berisi method yang digunakan untuk mengecek *damage* mobil berdasarkan kondisi-kondisi tertentu.
4	BestPowerupToUseChecker	Kelas ini berisi method `get_total_points_using_powerups` yang digunakan untuk menentukan *powerup* terbaik yang dapat digunakan. Method ini mengembalikan array 2 dimensi yang berisi poin dan *command* terbaik yang dapat digunakan.
5	NumOfPowerupChecker	Kelas ini berisi method `getNumOfPowerUps` untuk menghitung *powerups* yang dimiliki oleh mobil
6	PointsFromList	Kelas ini berisi method `getPointsFromList` yang berfungsi menghitung total poin dari sebuah array yang merupakan representasi kondisi obstacles dan powerups di sebuah lane
7	PointsOfAllLaneChecker	Kelas ini berisi method yang berfungsi untuk menghitung poin-poin dari setiap lane dengan *constraints* tertentu
8	TotalPointsUsingPowerupChecker	Kelas ini berfungsi menghitung total poin dengan menggunakan *powerup* tertentu pada kondisi tertentu.

9	LaneChooser	Kelas ini berisi *method* `choosingLane` untuk memilih lane dengan poin tertinggi dan `use_powerup` untuk mereturn command using powerup

4.4 Studi Analisis Implementasi Algoritma Greedy

Program hasil implementasi algoritma greedy yang kita kembangkan sangatlah bergantung pada pemetaan point yang kita buat. Dari hasil pemetaan point bot bisa seolah-olah membuat prioritasnya sendiri. Misal kita meningkatkan point obstacle, nantinya bot juga akan memprioritaskan obstacle. Oleh karena itu sangat diperlukan optimasi point yang sangat baik, agar dihasilkan bot yang optimal.

Iterasi pertama konfigurasi poin yang kami buat sangatlah menitikberatkan pada pengambilan powerups dan penggunaan boost, tanpa jumlah maksimal pengambilan. Pemetaan bonus point untuk pengambilan powerups jauh lebih besar dibanndingkan pengurangan poin akibat obstacle. Pada awal ronde konfigurasi ini bekerja dengan baik, tetapi semakin ke akhir ronde malah mengurangi performa mobil. Mobil malah sibuk mengambil powerup dan melupakan tujuan utamanya, yakni mencapai garis finish.

Iterasi kedua konfigurasi poin yang kami buat masih menitikberatkan pada pengambilan powerups, tetapi memiliki jumlah maksimal powerup yang seharusnya diambil. Konfigurasi ini cukup efektif dalam mengatasi permasalahan sebelumnya. Akan tetapi terlihat masalah baru, mobil seperti sulit sekali menjaga speednya. Mobil sering kali menabrak obstacle untuk powerups dan mengakibatkan penurunan kecepatan.

Iterasi ketiga konfigurasi poin yang kami buat mulai menitikberatkan pada obstatcles dan speed. Pemetaan poin obstacles sekarang jauh lebih besar dibandingkan powerups dan jumlah powerups yang diambil dibuat seminim mungkin. Konfigurasi ini terbilang efek mengatasi permasalahan sebelumnya. Kecepatan dan damage bot terjaga, sesuai dengan tujuan utamanya – mencapai garis finish. Kalaupun ditandingkan dengan dua konfigurasi lainnya, bot ini jauh lebih unggul. Kami merasa iterasi ini sudah cukup efektif dan efisien dalam menjalankan bot. Oleh karena itu, kami memutukan untuk menggunakan konfigurasi poin ketiga.

BAB 5
KESIMPULAN DAN SARAN

Algoritma *greedy* merupakan metode popular untuk permasalahan optimasi. Algoritma ini dapat diterapkan pada permasalahan yang penyelesainnya dapat dibagi menjadi langkah-langkah menuju solusi. Lebih dari itu, ia juga dapat dipakai dalam permainan. Algoritma *greedy* dapat diterapkan pada *bot* permainan yang memiliki sistem ronde seperti pada permainan *overdrive.* Kelompok kami sukses dalam merancang *bot* untuk dimainkan pada game *overdrive.* Kesuksesan diukur dari kemampuan *bot* untuk menyelesaikan permainan. Strategi *greedy* yang kami gunakan membuat *bot* kami kompetitif dengan *reference-bot*, bahkan *bot* kelompok lain. Strategi *greedy* juga memenuhi elemen-elemen dari algoritma *greedy.* Namun, tidak bisa dipungkiri bahwa ada masalah yang dialami karena strategi kami. Pembobotan poin pada komponen-komponen permainan membuat munculnya variable-variabel bebas yang mempengaruhi tindakan *bot.* Variabel bebas yang banyak dapat menyulitkan karena sulitnya ditemukan kombinasi terbaik. Kami menyarankan untuk dilakukan analisis lebih lanjut untuk menemukan korelasi antara bobot poin yang diberikan dengan tingkat kemenangan *bot.* Hal itu penting untuk dilakukan agar terciptanya *bot* yang memiliki tingkat kemenangan tinggi.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2021. Slide Bahan Kuliah: Algoritma Greedy (Bagian 1). Institut Teknologi Bandung: Bandung.

LAMPIRAN

Link repository

https://github.com/addinnabilal/Entelect-Challenge-2020

Link video demo

https://youtu.be/Ry4M_1vWX48