

Pemanfaatan Algoritma Greedy pada Bot Permainan Overdrive

**Tugas Besar I Mata Kuliah Strategi Algoritma
IF2221**



Oleh:

Ng Kyle 13520040

Muhammad Risqi Firdaus 13520043

Christopher Jeffrey 13520055

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022**

BAB I

PENDAHULUAN

1.1. Deskripsi Tugas

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

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan strategi greedy untuk memenangkan permainan. Sesuai dengan spesifikasi tugas 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. IntelliJ 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 IntelliJ 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 harus dikaitkan dengan objektif utama permainan, yakni mencapai garis finish lebih awal atau bersamaan tetapi dengan kecepatan atau skor lebih besar.

BAB II

Landasan Teori

2.1. Dasar Teori

Greedy merupakan algoritma yang berjalan dengan prinsip penyelesaian masalah dengan mencari nilai maksimum sementara pada setiap langkahnya. Nilai maksimum sementara dikenal dengan *local maximum*. Meskipun pada kebanyakan kasus tidak selalu menghasilkan solusi yang optimal global, tetapi algoritma greedy biasanya memberikan solusi yang mendekati optimum dengan waktu yang cukup cepat.

2.2. Cara Kerja Program

Program bot Override ini dijalankan berdasarkan file template yang diberikan. Pada program kali ini, kami menggunakan bahasa Java untuk menyelesaikan program kami.

Program java akan di-*compile* menjadi file jar. File jar hasil kompilasi akan dijalankan dengan untuk menjadi bot melalui make file yang terikat pada json.

File json digunakan sebagai pengarah kompilasi bot. Direktori untuk tiap bot ditaruh dalam file konfigurasi game-runner-config.json. Sedangkan, konfigurasi untuk tiap perintah menggunakan file game-config.json.

Hasil atau record permainan yang dijalankan akan disimpan dalam folder match-logs. Tiap permainan akan disimpan dalam folder sendiri, dengan penamaan waktu permainan dijalankan. Dapat digunakan visualizer dengan mengunggah file zip dari folder hasil kompilasi permainan. Untuk konfigurasi tiap bot, disimpan dalam file bot.json. File hasil kompilasi pada target, dan file program ada pada folder src.

Bot melakukan aksinya berdasarkan perintah yang ditulis pada method run, pada file bot.java. Pada tiap round, bot akan mengeksekusi method tersebut. Algoritma greedy untuk tiap turn atau langkah dapat diimplementasikan pada fungsi tersebut.

Pada tiap turn Bot akan mengeksekusi program. Gerakan atau command yang dilakukan Bot merupakan hasil atau nilai kembalian dari method Run. Program ini menggunakan prinsip dasar OOP, yang untuk tiap commandnya telah di-store pada modul. Sehingga kami hanya perlu menyusun program greedy yang mengembalikan nilai sesuai dengan yang sudah tersimpan pada program sebelumnya.

Untuk mendukung algoritma greedy yang dibuat, selain menyediakan object command, bot juga menyediakan object state. Object ini dapat memberikan nilai balikan berupa kondisi serta posisi baik dari player maupun opponent. Keberadaannya sangat berguna dalam implementasi power up, sehingga power up yang digunakan

dapat efektif. Method ini juga dapat digunakan untuk menghindari lawan, sehingga memperkecil probabilitas tabrakan. State lain yang juga penting adalah data kerusakan kendaraan. Perlu diperhatikan, bahwa kendaraan tidak akan berjalan jika damagenya telah mencapai poin lima. Oleh karena itu, dalam algoritma, kita juga perlu memastikan supaya mobil kita tetap dapat berjalan, dengan mengembalikan command FIX, ketika point damage telah mencapai nilai tertentu.

Pada program juga terdapat method block, terrain dan okupansi. Method ini akan mengembalikan nilai berupa kondisi dari blok yang dicari. Pada method terrain, program akan mengembalikan kondisi atau object berupa bentukan block yang diminta. Terrain dapat berupa mud, wall, maupun lizard. Sedangkan okupansi akan mengembalikan boolean okupansi pada block yang dituju, method ini digunakan untuk mengecek apakah ada truck yang menutup block.

Selain halangan, pada permainan, program juga dapat diset untuk mengejar power up sebanyak-banyaknya. Semakin banyak power up pada sebuah lajur maka kemungkinan untuk menang ketika melalui lajur tersebut juga akan semakin tinggi. Ada beberapa power up yang disediakan oleh game, dan methodnya dapat diakses oleh bot, seperti boost, tweet, EMP dan lizard.

BAB III

APLIKASI GREEDY

3.1. Aplikasi

Himpunan kandidat: Langkah-langkah yang menghasilkan gerakan yang dilakukan bot pada setiap turn. Pembobotan dilakukan berdasarkan jumlah hambatan yang dilalui serta *power up* yang didapat. Juga langkah untuk mengeluarkan power up dan boost. Berikut merupakan himpunan kandidat Command yang dapat dilakukan :

ACCELERATE, USE_BOOST, TURN_RIGHT, TURN_LEFT, NOTHING, USE_EMP, FIX, USE_LIZARD, USE_OIL, USE_TWEET

Himpunan solusi: Langkah yang dipilih bot pada setiap turn (command pada himpunan kandidat) hingga mencapai blok finish.

Fungsi solusi: Apakah blok yang dicapai sudah merupakan garis finish. Langkah yang dipilih bot untuk tiap turn atau nilai yang dikembalikan oleh metode run pada program.

Fungsi seleksi: Pilih langkah yang paling efektif dan efisien di antara menggunakan power up atau boost, berpindah jalur, atau mengakselerasi bot. Bot memilih untuk pindah lane atau lurus atau melakukan tindakan agresif. Pilihlah langkah yang menghindari kolisi dengan *obstacle* (lurus, kiri, kanan), jika tidak memungkinkan akan dipilih tindakan agresif yang paling efektif pada ronde tertentu.

Fungsi kelayakan:

Semua langkah yang tidak menyebabkan keluar jalur, serta melawan aturan permainan adalah layak. Dilakukan pemeriksaan untuk setiap command yang sedang diperiksa kelayakannya.

Fungsi objektif:

Mencapai garis finish terlebih dahulu dari lawan atau jika bersamaan memiliki score lebih tinggi dari lawan.

3.2. Alternatif Solusi

a. Lajur teraman dengan pengecekan satu langkah gerakan.

Program memilih langkah-langkah yang menghasilkan jalan lajur tercepat dan terefektif untuk mencapai garis finish. Melalui lajur yang memiliki hambatan terkecil atau *power up* terbesar di antara jalur lainnya. Serta, langkah untuk mengeluarkan power up dan FIX seoptimal dan efektif mungkin, bergantung pada state dari permainan.

Algoritma akan *me-asses* beban atau nilai dari tiap lajur yang akan dilalui, dengan pivot lajur tempat kendaraan berada saat itu. Dalam tiap asses yang dilakukan, hambatan akan dinilai sebagai beban, dan power up akan dinilai sebagai profit. Algoritma memilih

langkah yang memiliki beban terkecil. Perlu diperhatikan, tiap jenis hambatan nilainya berbeda. Tergantung efek keterlambatan yang ditimbulkan akibat menabrak tiap hambatan.

Selain hambatan dan power up, hal yang perlu diperhatikan adalah kecepatan. Tiap belokan yang dilakukan akan mengurangi kecepatan, sedangkan jika tetap lurus, dan kecepatan belum maksimal, bot akan mengakselerasi mobil.

Maka, alternatif algoritma greedy ini dalam beberapa tahap:

1. Memeriksa apakah lurus merupakan jalur optimal (dalam 2 langkah). Jika ya, maka akan dikembalikan ACCELERATE.
2. Memeriksa apakah belok merupakan jalur optimal (dalam 2 langkah). Jika ya, maka akan dikembalikan TURN.
3. Jika 1 dan 2 tidak terpenuhi, maka pilih jalur dengan nilai kolisi terkecil.

b. Lajur teraman dengan pengecekan dua langkah gerakan.

Strategi ini sama seperti halnya pada strategi yang dipaparkan sebelumnya, namun dengan melakukan penilaian berdasarkan 2 langkah ke depan (memeriksa konfigurasi lane yang ada dalam 2 langkah).

Algoritma Greedy ini yang digunakan dalam program yang dibuat dengan pertimbangan merupakan konsep yang paling menggambarkan konsep Greedy pada game ini karena game ini secara inheren memerlukan kita mempertimbangkan gerakan yang dapat kita lakukan mempertimbangkan 20 blok di depan dan 5 blok di depan kita sehingga optimum lokal yang di-asses adalah 2 putaran.

c. Pengecekan 3 lajur utama dengan pencarian lajur teraman dengan pembobotan

Algoritma bot akan mengecek kondisi 3 lajur utama, jika bot ada di pojok (lajur pertama atau terakhir) maka bot hanya mengecek 3 lajur. Tiap lajur dicek dengan perhitungan satu turn permainan. Tiap pengecekan dilakukan perulangan untuk mengecek tiap block dari ketiga lajur yang dicek.

Pembobotan point dilakukan berdasarkan tiga kriteria, collision yang mungkin terjadi, banyak power up serta proyeksi kecepatan jika mengambil turn tersebut. Pada kondisional pertama, program akan mengeksaminasi berapa nilai tabrakan pada tiap lane, program akan cenderung memilih lane yang menyebabkan damage nol. Jika ada lebih dari satu yang nol, maka program akan memilih yang memiliki power up terbanyak, atau dapat menghasilkan kecepatan tertinggi.

Jika tidak ada lajur yang aman dari 3 lajur yang dicek oleh program, maka program akan mengeksaminasi lajur dengan nilai poin tertinggi. Poin tertinggi didapatkan dari akumulasi besar damage serta power ups terbanyak dan menghasilkan kecepatan tertinggi.

d. Pengecekan 3 lajur utama dengan pembobotan

Mirip seperti algoritma sebelumnya, algoritma ini mengevaluasi 3 lajur utama, lajur lurus, serta kanan dan kiri lajur yang sedang dilalui. Perbedaannya terdapat pada dasar pemilihan utama pada keputusan yang diambil bot. Jika pada program kedua program

mengutamakan keputusan yang dapat memberikan damage paling sedikit bagi bot. Sedangkan, program ini mengutamakan pemilihan keputusan yang memberikan power up terbanyak serta kecepatan akselerasi tertinggi.

Algoritma ini secara konsep seperti halnya Greedy pada Knapsack problem. Untuk mengevaluasi tiap-tiap jalur, dapat diperiksa kelayakan dari setiap lane menggunakan pembobotan pada aspek-aspek persoalan. Dalam permainan Overdrive ini terdapat 3 aspek utama : Speed, Powerups, Obstacle. Maka dapat dibuat 3 jenis alternatif Greedy yaitu:

1. Greedy by Speed

Menilai setiap lane dengan pembobotan atas kecepatan yang dapat digunakan dalam lane tersebut. e.g. Banyaknya blok yang dapat ditraverse pada lane tersebut tanpa bertabrakan.

2. Greedy by Powerups

Menilai setiap lane dengan pembobotan berdasarkan banyaknya powerups pada lane tersebut. Secara practice, umumnya Greedy ini dinilai ketika kita memutuskan untuk melakukan selain memilih ACCELERATE sehingga perlu mempertimbangkan apakah lebih baik memilih jalur kanan, kiri, atau tetap lurus,

3. Greedy by Obstacle

Menilai setiap lane dengan pembobotan berdasarkan banyaknya collision paling sedikit. Hal ini dapat memungkinkan bot untuk mengambil jalur baik lurus, kiri, maupun kanan dengan mempertimbangkan banyaknya halangan pada jalur tersebut. Tradeoff dari Greedy ini adalah memungkinkan bot untuk lebih sering memilih belok sehingga perlu diikuti dengan algoritma pemeriksaan lainnya.

e. Strategi Agresif

Selain pergerakan mobil, untuk menambah kemampuan bot, perlu di-asses kemampuan menyerang bot. Truck atau twit harus ditempatkan 1 kotak didepan block musuh ditambah kecepatannya. EMP digunakan ketika bot tertinggal dari musuhnya dan berada dalam 1 lane. Sedangkan oil, digunakan pada daerah sempit, untuk menghambat musuh yang akan datang, sehingga tak memiliki pilihan lajur yang bebas hambatan. Hal ini baik untuk menyulitkan lawan dengan tradeoff kurangnya penambahan nilai positif secara aktif ke bot sendiri (melakukan ACCELERATE, BOOST, etc).

f. Strategi balapan

Algoritma ini memprioritaskan kecepatan. Selama kecepatanku belum maksimum, dilakukan command ACCELERATE. ACCELERATE tidak dilakukan hanya ketika bot memiliki power up BOOST. Jika bot memiliki BOOST, command yang dilakukan adalah BOOST. Dengan hal ini, bot akan mengutamakan untuk terus mempercepat kecepatan dirinya dalam jalur lurus dengan pertimbangan untuk belok sangat minim.

3.3. Analisis Efisiensi

Dalam penghitungan efisiensi digunakan L : Lane, B : Blok, dan S : Speed. L = 4, Blok = 20, S = Variable, serta C untuk konstanta.

Algoritma pertama melakukan evaluasi terhadap setiap lane sepanjang speed dari bot untuk memeriksa baik banyak kolisi pada tahap pertama serta powerups pada tahap kedua. Sehingga secara ringkas kompleksitas algoritma pertama : $O(C * 3 * S) = O(S)$

Algoritma kedua dan ketiga menghasilkan jumlah komputasi yang paling besar di antara solusi algoritma pemilihan langkah lainnya. Hal tersebut terjadi karena pada algoritma pertama terjadi nested computation (iteration) di mana program akan mengevaluasi n kali lebih banyak (kuadrat). Pada Worst Case secara kecepatan algoritma melakukan evaluasi terhadap seluruh lane sebanyak 2 kali dan setiap lane dievaluasi sepanjang map yang diberikan (20 blok), maka efisiensi secara kasar : $O(2 * C * (3S)^2) = O(S * S)$

Algoritma keempat menghasilkan beban komputasi yang sama, karena secara umum komputasi yang dilakukan. Perbedaan algoritma kedua dan ketiga terdapat pada kondisional pemilihan langkah. Beban utama pemilihan algoritma kedua lebih kepada keamanan mobil, sedangkan algoritma ketiga mengutamakan penambahan kemampuan dan kecepatan dengan kecepatan $O(3 * S * S) = O(S * S)$, melakukan pencarian tiap lane dan evaluasi nilai sebanyak S blok.

Keberadaan algoritma kelima dibilang cukup efisien, karena ia berada pada titik pemanfaatan power berdasarkan state game. Algoritma ini hanya perlu memeriksa 1 lane ke depan maka secara efisiensi waktu $O(S)$.

Algoritma balapan memiliki perbandingan ada atau tidaknya powerup boost maka hanya diperlukan 1 kali perbandingan dengan kompleksitas algoritma $O(1)$.

3.4. Analisis Efektivitas

Meskipun algoritma pertama memiliki tingkat efektivitas yang cukup baik dengan mencari lokal optimum pada 1 langkah. Namun hal ini dapat ditingkatkan dengan algoritma kedua yaitu memeriksa kembali langkah kedua.

Meskipun algoritma kedua memiliki beban komputasi terbesar di antara algoritma lainnya. Namun, dapat dipastikan algoritma ini memiliki langkah paling efektif di antara ketiganya. Algoritma ini memungkinkan untuk menghindari jebakan yang mungkin terjadi pada dua turn berikutnya. Hal ini memungkinkan algoritma untuk mendapatkan solusi optimum lokal dalam 2 tahap. Hal ini didapat dengan batasan map yang diberikan 20 blok di depan sedangkan speed maksimum dari bot adalah 9. Maka dalam 2 langkah bot tetap tidak akan melebihi constraint map yang diberikan dan mendapat solusi optimum lokal (terkecuali speed boost dengan speed 15). Namun, hal ini dapat ditingkatkan dengan mempertimbangkan 3 langkah ke depan ketika speed $\leq 20 \text{ div } 3 = 6$. Namun hal ini akan

meningkatkan kompleksitas algoritma dan tingkat kerumitan code yang memungkinkan error pada implementasi yang banyak (adanya kasus yang terlepas).

Jika algoritma ketiga dan keempat, dapat terjebak pada collision yang berjajar (3 atau 2 ada sebuah urutan blok vertikal). Maka hal tersebut masih dapat diatasi oleh algoritma kedua karena evaluasi dua langkah memungkinkan bot untuk berpindah 2 block melalui dua gerakan beruntun. Hal tersebut terjadi akibat evaluasi dua langkah yang dilakukan sistem.

Penerapan algoritma kelima efektif untuk mengganggu lawan ketika di kondisi bot sedang tidak perlu melakukan ACCELERATE, TURN_LEFT, TURN_RIGHT, dan command positif lainnya. Penarahan hambatan pada 1 block di depan blok yang mungkin dihampiri musuh pada turn berikutnya dapat meminimalisasi terjadinya penempatan hambatan yang tak perlu. Selain itu, penggunaan power up secara efektif juga dapat mengurangi probabilitas terjadinya collision akibat kegagalan mobil untuk melompat. Terakhir, penggunaan EMP pada lajur yang tepat serta timing penggunaan boost yang tepat dapat membuat bot mengejar musuh, jika bot pada posisi tertinggal.

3.5.Strategi Greedy yang Dipilih serta Pertimbangan

Berdasarkan analisis efektivitas serta efisiensi yang dilakukan terhadap algoritma yang dirumuskan. Maka dipilih kombinasi algoritma pertama dan keempat untuk diimplementasikan pada program yang dibuat.

Algoritma pertama dipilih karena memiliki nilai efektivitas serta pergerakan paling efektif dan *agile* dibandingkan algoritma lain. Meskipun memiliki beban komputasi terbesar, bebannya masih dalam angka rasional. Berdasarkan perhitungan yang dilakukan nilai big-O pada algoritma pertama pada tiap eksekusi ialah $O(n^2)$, dengan n merupakan banyak block. Oleh sebab itu, dipilih algoritma tersebut karena banyak turn terbilang sedikit (dibawah 1000 turn) maka dipilihlah algoritma pertama.

Kombinasi dengan algoritma keempat tidak akan menyebabkan penambahan beban komputasi secara signifikan. Namun, memberikan efek yang cukup krusial terutama ketika bot pada posisi tertinggal. Mengeluarkan kemampuan power up di saat yang tepat dapat memberikan hambatan berarti bagi musuh sehingga memberikan waktu yang cukup bagi bot untuk mengejar atau menjauh dari musuh.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma

Algoritma utama pada program terdapat pada file bot.java pada metode Command run. Pseudocode untuk metode tersebut dapat dilihat sebagai berikut:

```
Public Command run(GameState gameState)
//Inisiasi update game state
    this.gameState <- gameState;
    this.myCar <- gameState.player;
    this.opponent <- gameState.opponent;

    // Greedy by Boosting to maintain speeed
    If myCar.boostCounter = && myCar.speed = BOOST_SPEED then
        myCar.speed <- MAXIMUM_SPEED
    end if
    If myCar.damage > 2 then
        return FIX;
    end if

    If myCar = 0 then
        If isCanAndShould() then
            If myCar.damage >0 then return FIX
            else return BOOST
        end if
        else
            return ACCELERATE
        end if
    end if
    //evaluasi dua langkah, apakah lurus ada obstacle dan
    belok tidak
    If (shouldIMoveLeftOrRight(myCar.speed)) then
        //cari yang ga berdamage atau damage terendah
        atau score tertinggi
        return bestBetweenLeftAndRight()
```

```

else
    If myCar.speed = BOOST_SPEED then
        //cek apakah lane lurus ada
        halangan
            If haveObstacle(myCar.position.lane -
            1, myCar.position.block + 1, myCar.speed - 1)
                if(hasPowerUp(PowerUps.LIZARD)
            then
                Return LIZARD
            end if
        end if
    //jika lane lurus ada gada halangan atau ga punya
    LIZARD
        return aggressive() //keluarkan power up
    menyerang
    else //punya lizard dan ada obstacle
        If (isPowerUpMoreThanX(PowerUps.LIZARD, 2)
        && haveObstacle(myCar.position.lane - 1,
        myCar.position.block + 1, myCar.speed - 1) then
            return LIZARD
        else if sCanAndShouldBoost() then //punya
        boost dan aman
            If myCar.damage > 0 then return FIX
            Else return BOOST
            end if
        else
            If ((myCar.damage = 0 or myCar.damage
            = 1) and (myCar.speed < MAXIMUM_SPEED)) or
            ((myCar.damage = 2) and (myCar.speed <
            SPEED_STATE_3) ) then
                Return ACCELERATE
            else //mode menyerang (mengeluarkan
            power ups)
                Return aggressive()
            end if
        end if
    end if
end method

```

4.2. Penjelasan Struktur Data

Struktur data yang digunakan didapat dari kode sumber yang diberikan pada <https://github.com/EntelectChallenge/2020-Overdrive> yaitu dengan runutan kumpulan class sebagai berikut:

- Command : Berisi class dengan isi command-command yang diterima game engine yang telah dijelaskan sebelumnya seperti ACCELERATE, TURN_[DIRECTION], USE_[ITEM], etc.
- Car : Class objek car dengan atribut :int id, int position, int speed, int damage, State state, List of Powerups powerups
- Gamestate : Class dengan atribut: int currentRound, int maxRounds, Car player, Car opponent, List of Lanes lanes.
- Lane : Class dengan atribut : Position position, Terrain terrain, int OccupiedByPlayerId, boolean isOccupiedbyCyberTruck
- Position : Class dengan atribut : int lane, int block.
Dan data enums:
 - Direction memiliki atribut: lane, block, dan string label, serta metode: konstruktor, dan getLabel().
 - PowerUps memiliki atribut: serialized: BOOST, OIL, TWEET, LIZARD, dan EMP.
 - State memiliki atribut serialized: ACCELERATING, READY, NOTHING, TURNING_LEFT, TURNING_RIGHT, HIT_MUD, HIT_OIL, DECELERATING, PICKED_UP_POWERUP, USED_BOOST, USED_OIL, USED_LIZARD, HIT_WALL, HIT_CYBER_TRUCK, FINISHED
 - Terrain memiliki serialized : EMPTY, MUD, OIL_SPILL, OIL_POWER, FINISH, BOOST, WALL, LIZARD, TWEET, dan EMP.

Algoritma greedy secara umum tidak memperbolehkan adanya pencatatan global pada tiap langkah. Mengingat ini, struktur data yang dapat kita simpan di dalam algoritma tidaklah mewah, namun kami masih bisa melakukan pencatatan lokal di tiap langkah agar kode lebih rapi, memberikan abstraksi juga terhadap angka-angka dan kata-kata yang tidak deskriptif menjadi deskriptif. Pencatatan kami buat dengan tipe data integer dan boolean. Kedua tipe data tersebut sudah cukup untuk membantu kami membuat code yang lebih mudah untuk dibaca. Beberapa integer ada yang bersifat final. Final pada java bertindak seperti constant, yaitu sebuah data yang tidak bisa dirubah saat runtime. Data final yang kami buat berguna untuk menampung ‘penalty’ dari terrain, ‘score’ dari powerup, dan ‘speed-state’ dari mobil di dalam game.

4.3. Analisis Desain Solusi Algoritma

Berdasarkan pengetesan yang telah kami lakukan, algoritma yang kami buat belum seratus persen optimal. Mengingat fungsi objektif dari algoritma ini adalah mencapai garis finish lebih cepat daripada lawan, bot kami masih bisa kalah dari bot lawan. Hal ini kami simpulkan berdasarkan pengamatan langsung, melihat bot kami bekerja satu langkah demi satu langkah, dan melihat masih ada beberapa langkah yang belum ‘local maxima’, belum merupakan langkah terbaik yang bisa dilakukan berdasarkan informasi yang dia miliki.

Dengan menggunakan website <https://entelect-replay.raezor.co.za/#>, hasil match yang awalnya berupa ASCII di command line atau terminal, dapat kita lihat sebagai GUI.

Round 107
Reset Remove this match

[1072, 1]	[1073, 1]	[1074, 1]	[1075, 1]	[1076, 1]	[1077, 1]	[1078, 1]	[1079, 1]	[1080, 1]	[1081, 1]	[1082, 1]	[1083, 1]	[1084, 1]	[1085, 1]	[1086, 1]	[1087, 1]	[1088, 1]	[1089, 1]	[1090, 1]	[1091, 1]	[1092, 1]	[1093, 1]	[1094, 1]	[1095, 1]	[1096, 1]	[1097, 1]
[1072, 2]	[1073, 2]	[1074, 2]	[1075, 2]	[1076, 2]	[1077, 2] ← WALL→	[1078, 2]	[1079, 2] ← WALL→	[1080, 2]	[1081, 2]	[1082, 2]	[1083, 2]	[1084, 2]	[1085, 2]	[1086, 2]	[1087, 2] ← WALL→	[1088, 2]	[1089, 2] ← WALL→	[1090, 2]	[1091, 2] ← WALL→	[1092, 2]	[1093, 2]	[1094, 2]	[1095, 2]	[1096, 2]	[1097, 2]
[1072, 3]	[1073, 3]	[1074, 3]	[1075, 3]	[1076, 3] ← WALL→	[1077, 3]	[1078, 3]	[1079, 3]	[1080, 3]	[1081, 3]	[1082, 3]	[1083, 3]	[1084, 3]	[1085, 3] ← WALL→	[1086, 3] ← WALL→	[1087, 3] ← WALL→	[1088, 3]	[1089, 3]	[1090, 3]	[1091, 3]	[1092, 3]	[1093, 3]	[1094, 3]	[1095, 3]	[1096, 3]	[1097, 3]
[1072, 4]	[1073, 4]	[1074, 4]	[1075, 4]	[1076, 4]	[1077, 4]	[1078, 4]	[1079, 4]	[1080, 4]	[1081, 4] ← WALL→	[1082, 4]	[1083, 4]	[1084, 4]	[1085, 4]	[1086, 4]	[1087, 4]	[1088, 4]	[1089, 4]	[1090, 4]	[1091, 4]	[1092, 4] ← WALL→	[1093, 4]	[1094, 4]	[1095, 4]	[1096, 4]	[1097, 4]

< First < Prev 107 Next > Last >

(Click the button that displays the round number to quickly switch rounds)

Round Details
Max Rounds: 600
Current Round: 107

A - tea



Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
1 [x: 1077, y: 2]	15	2	1077	1	YES	EMPEMPMPMPMPMPMP MPMPMPMLIZARD,EMP BOOST,LIZARD,EMP,LIZA RD,LIZARD,EMP,BOOST,E MPTWEET,EMP

Bot Command

```
Command: TURN_RIGHT
Execution time: 2ms
Exception: null
```

Berdasarkan analisis di atas, terbukti bahwa, solusi local maxima tak selalu sama dengan global maxima. Kita menemukan bahwa pada hambatan beruntun atau hambatan yang muncul setelah gerakan dan berada di luar ruang evaluasi program tidak mungkin dapat dihindari bot.

Meskipun memiliki kekurangan dalam lingkup ruang evaluasi, tetapi greedy memiliki advantage utama yakni algoritma yang cukup mangkus. Algoritma greedy tidak memakan beban komputasi yang besar dibandingkan algoritma brute force atau DP dan DC.

Round 072																										
Reset Remove this match																										
[604, 1]	[605, 1]	[606, 1]	[607, 1]	[608, 1]	[609, 1]	[610, 1]	[611, 1]	[612, 1]	[613, 1] <GO/STOP>	[614, 1] <GO/STOP>	[615, 1]	[616, 1]	[617, 1]	[618, 1]	[619, 1]	[620, 1] <GO/STOP>	[621, 1]	[622, 1]	[623, 1]	[624, 1]	[625, 1]	[626, 1]	[627, 1]	[628, 1]	[629, 1]	
[604, 2]	[605, 2]	[606, 2]	[607, 2]	[608, 2]	[609, 2]	[610, 2]	[611, 2]	[612, 2]	[613, 2]	[614, 2]	[615, 2]	[616, 2]	[617, 2]	[618, 2]	[619, 2]	[620, 2]	[621, 2]	[622, 2]	[623, 2]	[624, 2]	[625, 2]	[626, 2]	[627, 2]	[628, 2]	[629, 2]	
[604, 3] <GO/STOP>	[605, 3]	[606, 3]	[607, 3]	[608, 3]	[609, 3] <GO/STOP>	[610, 3]	[611, 3]	[612, 3]	[613, 3]	[614, 3] <GO/STOP>	[615, 3]	[616, 3]	[617, 3]	[618, 3]	[619, 3]	[620, 3] <GO/STOP>	[621, 3]	[622, 3]	[623, 3]	[624, 3]	[625, 3]	[626, 3]	[627, 3]	[628, 3] <GO/STOP>	[629, 3]	
[604, 4]	[605, 4]	[606, 4]	[607, 4]	[608, 4]	[609, 4]	[610, 4]	[611, 4]	[612, 4]	[613, 4]	[614, 4] <GO/STOP>	[615, 4]	[616, 4] <GO/STOP>	[617, 4]	[618, 4]	[619, 4]	[620, 4] <GO/STOP>	[621, 4]	[622, 4]	[623, 4]	[624, 4]	[625, 4]	[626, 4] <GO/STOP>	[627, 4]	[628, 4]	[629, 4]	

[< First] [< Prev] **72** [Next >] [Last >]
(Click the button that displays the round number to quickly switch rounds)

Round Details Current Round: 72

A - tea (selected) ▾

Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
2	15	3	609	5	YES	OIL,OIL,OIL,BOOST,LIZAR D,OIL

Bot Command

Command: TURN_LEFT
Execution time: 2ms
Exception: null

Gambarx. Contoh evaluasi satu langkah kedepan

Round 61																											
First Prev 61 Next Last																											
[441, 1]	[442, 1]	[443, 1]	[444, 1]	[445, 1]	[446, 1] <GO/STOP>	[447, 1]	[448, 1]	[449, 1]	[450, 1]	[451, 1]	[452, 1]	[453, 1]	[454, 1]	[455, 1]	[456, 1]	[457, 1]	[458, 1]	[459, 1]	[460, 1]	[461, 1]	[462, 1]	[463, 1]	[464, 1]	[465, 1] <GO/STOP>	[466, 1]		
[441, 2]	[442, 2]	[443, 2]	[444, 2]	[445, 2]	[446, 2]	[447, 2]	[448, 2]	[449, 2]	[450, 2]	[451, 2]	[452, 2]	[453, 2]	[454, 2]	[455, 2]	[456, 2]	[457, 2]	[458, 2]	[459, 2]	[460, 2]	[461, 2] <GO/STOP>	[462, 2]	[463, 2]	[464, 2]	[465, 2]	[466, 2]	[467, 2]	[468, 2]
[441, 3] <GO/STOP>	[442, 3]	[443, 3]	[444, 3]	[445, 3]	[446, 3]	[447, 3]	[448, 3]	[449, 3]	[450, 3]	[451, 3]	[452, 3]	[453, 3]	[454, 3]	[455, 3]	[456, 3]	[457, 3]	[458, 3]	[459, 3]	[460, 3]	[461, 3]	[462, 3]	[463, 3]	[464, 3]	[465, 3]	[466, 3]	[467, 3]	
[441, 4]	[442, 4] <GO/STOP>	[443, 4]	[444, 4]	[445, 4]	[446, 4]	[447, 4]	[448, 4]	[449, 4]	[450, 4]	[451, 4]	[452, 4]	[453, 4]	[454, 4]	[455, 4]	[456, 4]	[457, 4]	[458, 4]	[459, 4]	[460, 4]	[461, 4]	[462, 4]	[463, 4]	[464, 4]	[465, 4]	[466, 4]	[467, 4]	

[< First] [< Prev] **61** [Next >] [Last >]
(Click the button that displays the round number to quickly switch rounds)

Round Details Current Round: 61

A - tea (selected) ▾

Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
2	15	1	486	5	YES	OIL,OIL,OIL,BOOST

Bot Command

Command: TURN_RIGHT
Execution time: 2ms
Exception: null

Round 142

[Reset](#) [Remove this match](#)

[1251, 1]	[1252, 1]	[1253, 1]	[1254, 1]	[1255, 1]	[1256, 1] <small>-0.020s</small>	[1257, 1]	[1258, 1]	[1259, 1]	[1260, 1]	[1261, 1]	[1262, 1]	[1263, 1]	[1264, 1]	[1265, 1]	[1266, 1]	[1267, 1]	[1268, 1]	[1269, 1]	[1270, 1]	[1271, 1]	[1272, 1]	[1273, 1]	[1274, 1]	[1275, 1]	[1276, 1]
[1251, 2]	[1252, 2]	[1253, 2]	[1254, 2]	[1255, 2] <small>-0.020s</small>	[1256, 2]	[1257, 2]	[1258, 2]	[1259, 2]	[1260, 2]	[1261, 2]	[1262, 2]	[1263, 2]	[1264, 2] <small>-0.020s</small>	[1265, 2]	[1266, 2]	[1267, 2]	[1268, 2]	[1269, 2] <small>-0.020s</small>	[1270, 2]	[1271, 2] <small>-0.020s</small>	[1272, 2]	[1273, 2]	[1274, 2]	[1275, 2]	[1276, 2]
[1251, 3]	[1252, 3]	[1253, 3]	[1254, 3]	[1255, 3] <small>-0.020s</small>	[1256, 3]	[1257, 3]	[1258, 3]	[1259, 3]	[1260, 3]	[1261, 3]	[1262, 3] <small>-0.020s</small>	[1263, 3]	[1264, 3]	[1265, 3]	[1266, 3] <small>-0.020s</small>	[1267, 3]	[1268, 3]	[1269, 3]	[1270, 3]	[1271, 3] <small>-0.020s</small>	[1272, 3]	[1273, 3]	[1274, 3]	[1275, 3]	[1276, 3]
[1251, 4]	[1252, 4]	[1253, 4]	[1254, 4] <small>-0.020s</small>	[1255, 4]	[1256, 4]	[1257, 4]	[1258, 4]	[1259, 4]	[1260, 4]	[1261, 4]	[1262, 4]	[1263, 4]	[1264, 4]	[1265, 4]	[1266, 4]	[1267, 4]	[1268, 4] <small>-0.020s</small>	[1269, 4]	[1270, 4]	[1271, 4] <small>-0.020s</small>	[1272, 4]	[1273, 4]	[1274, 4]	[1275, 4]	[1276, 4]

[\[< First\]](#) [\[< Prev\]](#) **142** [\[Next >\]](#) [\[Last >\]](#)
(Click the button that displays the round number to quickly switch rounds)

Round Details

Max Rounds: 600

Current Round: 142

A - tea (selected) ▾

Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
2	9	3	1256	0	No	OIL,OIL,OIL,OIL,OIL,BOO ST,LIZARD,LIZARD,OIL,LI ZARD,LIZARD,LIZARD,LIZ ARD

Bot Command
Command: TURN_LEFT
Execution time: 2ms
Exception: null

Gambarx dan y . Contoh evaluasi dua langkah kedepan

[1258, 1]	[1259, 1]	[1260, 1]	[1261, 1]	[1262, 1]	[1263, 1]	[1264, 1]	[1265, 1]	[1266, 1]	[1267, 1]	[1268, 1]	[1269, 1]	[1270, 1]	[1271, 1]	[1272, 1]	[1273, 1]	[1274, 1]	[1275, 1]	[1276, 1]	[1277, 1]	[1278, 1]	[1279, 1]	[1280, 1]	[1281, 1]	[1282, 1]
[1258, 2]	[1259, 2]	[1260, 2]	[1261, 2]	[1262, 2]	[1263, 2]	[1264, 2]	[1265, 2]	[1266, 2]	[1267, 2]	[1268, 2]	[1269, 2]	[1270, 2]	[1271, 2]	[1272, 2]	[1273, 2]	[1274, 2]	[1275, 2]	[1276, 2]	[1277, 2]	[1278, 2]	[1279, 2]	[1280, 2]	[1281, 2]	[1282, 2]
[1258, 3]	[1259, 3]	[1260, 3]	[1261, 3] <small>-0.020s</small>	[1262, 3] <small>-0.020s</small>	[1263, 3]	[1264, 3] <small>-0.020s</small>	[1265, 3]	[1266, 3]	[1267, 3]	[1268, 3]	[1269, 3]	[1270, 3]	[1271, 3] <small>-0.020s</small>	[1272, 3]	[1273, 3]	[1274, 3]	[1275, 3]	[1276, 3]	[1277, 3]	[1278, 3]	[1279, 3]	[1280, 3]	[1281, 3]	[1282, 3]
[1258, 4]	[1259, 4]	[1260, 4]	[1261, 4]	[1262, 4]	[1263, 4]	[1264, 4]	[1265, 4]	[1266, 4]	[1267, 4]	[1268, 4]	[1269, 4]	[1270, 4]	[1271, 4]	[1272, 4] <small>-0.020s</small>	[1273, 4]	[1274, 4]	[1275, 4]	[1276, 4]	[1277, 4]	[1278, 4]	[1279, 4]	[1280, 4]	[1281, 4]	[1282, 4]

[\[< First\]](#) [\[< Prev\]](#) **134** [\[Next >\]](#) [\[Last >\]](#)
(Click the button that displays the round number to quickly switch rounds)

A - tea (click to select) ▾

B - tea (selected) ▾

Position	Speed	Lane	Distance	Boosts	Boosting	Powerups
1	9	3	1263	0	No	LIZARD,EMP,EMP,LIZARD, EMP,EMPLIZARD,LIZARD, LIZARD,EMP,EMP,EMP,LIZ ARD,EMPLIZARD,EMPEM PEM,EMPLIZARD,LIZAR D,EMP,LIZARD,BOOST,OI L

Bot Command
Command: TURN_RIGHT
Execution time: 2ms

Gambar x. Contoh kesalahan. Pada langkah diatas, dia belok kanan lalu belok kiri. Langkah yang lebih baik adalah lurus lalu lurus. Kedua alternatif tetap menghasilkan tubrukan yang tidak terhindarkan, namun alternatif kedua memberikan bot keuntungan berupa dua langkah blok lebih jauh. Hal ini terjadi karena bot bekerja dengan menghindari tubrukan terdekat sebisa mungkin.

BAB V

Kesimpulan dan Saran

5.1. Kesimpulan

Greedy merupakan algoritma yang cukup baik untuk menyelesaikan permasalahan bot override. Algoritma greedy hampir memberikan solusi yang selalu optimal pada perlombaan empat lajur. Algoritma yang memanfaatkan local maxima ini dapat memilih jalur tercepat kecuali jika terdapat halangan beruntun yang melebihi ruang evaluasinya. Meskipun terdapat kekurangan pada evaluasi jangka panjang, algoritma greedy menghasilkan algoritma yang sangat efisien dengan beban komputasi yang rendah dan lebih mudah dibanding dengan algoritma DP atau DC yang mengevaluasi hingga akhir blok. Oleh karena itu, dapat disimpulkan bahwa algoritma greedy bukan algoritma yang dapat digunakan untuk semua masalah, tetapi pada problem dengan masukan besar dan efek hambatan kecil algoritma ini dapat menjadi sangat efektif.

5.2. Saran

Untuk pengembangan selanjutnya, jika beban kompleksitas memungkinkan evaluasi program dapat ditingkatkan kepada evaluasi semua kemungkinan lane serta pergerakan tiga langkah ke depan. Dengan evaluasi kepada tiap lane, serta pergerakan tiga langkah kedepan, program hampir tidak mungkin dikalahkan, serta dapat menemukan solusi paling optimal untuk tiap turn.

Kemudian, jika bot memiliki kecepatan nol, dan memiliki boost maka utamakan untuk menggunakan boost daripada mengakselerasi. Selain itu, dapat pula dievaluasi block yang ada dibelakangnya, serta posisi musuh. Evaluasi block belakang, dapat digunakan untuk menjatuhkan oli dengan waktu paling optimal, dan menghindari serangan EMP dari musuh.

Daftar Pustaka

[1] Rinaldi Munir, Diktat kuliah IF2251 Strategi Algoritmik, , Teknik Informatika ITB,

Lampiran

Link Video Aplikasi : <https://youtu.be/TZIsVEkUvuQ>

Link Github : <https://github.com/Nk-Kyle/Stima1-CarComp>