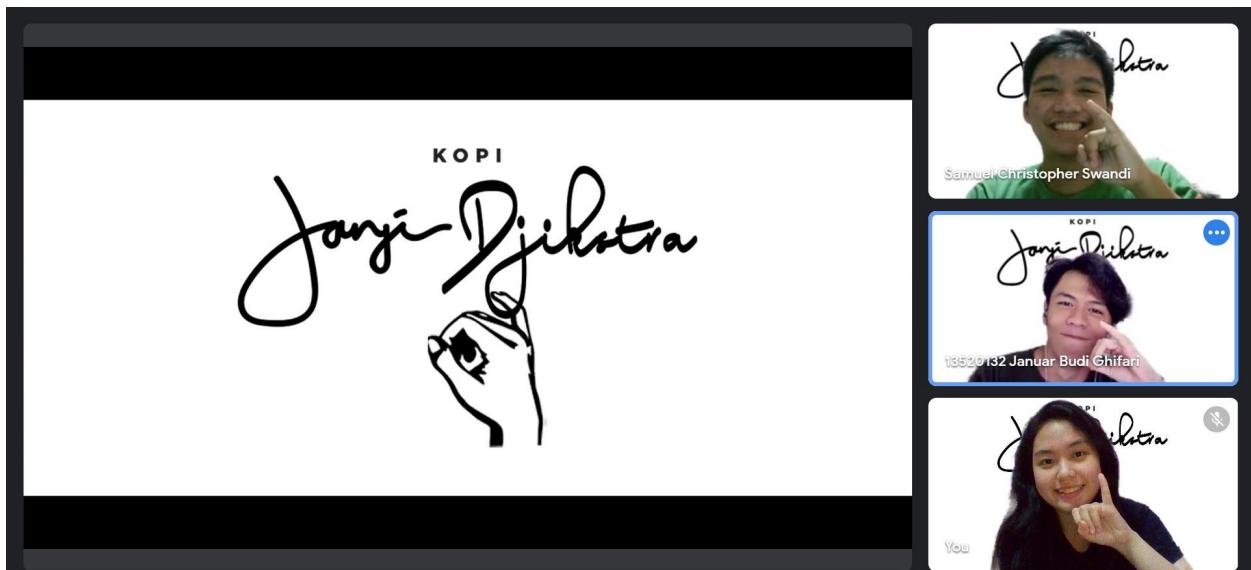


Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Overdrive”

Laporan Tugas Besar

IF2211 Strategi Algoritma

Semester II Tahun 2021/2022



Disusun oleh:

Samuel Christopher Swandi 13520075

Grace Claudia 13520078

Januar Budi Ghifari 13520132

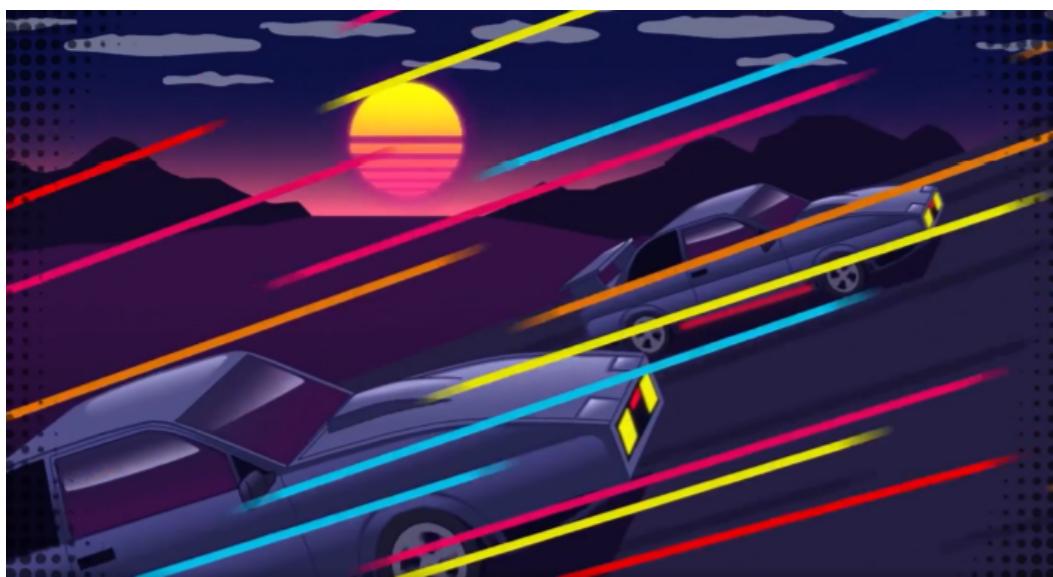
DAFTAR ISI

BAB 1 Deskripsi Tugas	2
BAB 2 Landasan Teori	5
BAB 3 Aplikasi Strategi Greedy	10
BAB 4 Implementasi dan pengujian	16
BAB 5 Kesimpulan dan Saran	26
Daftar Pustaka	27

BAB I

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/20203.4>

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
 - 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

Pada Bab 2, akan dijelaskan mengenai dasar teori algoritma greedy dan penggunaan game engine untuk menjalankan game overdrive.

2.1 Algoritma Greedy

Menurut Cormen, Leiserson, Rivest & Stein (2009:414-428), algoritma greedy merupakan algoritma dengan selalu mengambil solusi lokal yang optimal dengan harapan pilihan itu mengarah pada solusi optimal global. Algoritma ini menggunakan pendekatan penyelesaian masalah dengan memilih pilihan yang nampak memberikan hasil yang terbaik atau yang disebut juga **optimum lokal**, “*take what you can get now!*” pada setiap langkahnya dengan harapan bahwa akan mengarah ke solusi **optimum global**.

Algoritma *greedy* biasanya digunakan dalam permasalahan optimasi yang dapat berupa maksimasi atau minimasi. Maksimasi merupakan pencarian solusi terbesar sedangkan minimasi merupakan pencarian solusi terkecil. Algoritma ini tidak selalu berhasil memberikan solusi optimal. Tetapi, algoritma ini hampir selalu memberikan solusi yang mendekati nilai optimal. Menurut Setiadi (2008:95-97), prinsip-prinsip metode greedy sebagai berikut:

1. Metode greedy mengabaikan perhitungan lengkap dalam mencari solusi
2. Metode greedy mencari solusi dengan cepat
3. Metode greedy berguna untuk pencarian solusi yang memakan waktu terlalu lama dengan menggunakan komputer

Algoritma greedy disusun oleh elemen-elemen antara lain:

- a. Himpunan kandidat (C)
himpunan yang berisi elemen-elemen pembentuk solusi
- b. Himpunan solusi (S)
himpunan yang berisi kandidat-kandidat yang sudah terpilih sebagai solusi dari suatu persoalan
- c. Fungsi seleksi (*Selection Function*)
memilih kandidat yang paling memungkinkan untuk mencapai solusi yang optimal.
- d. Fungsi kelayakan (*Feasibility*)
memeriksa apakah suatu kandidat yang telah dipilih memberikan solusi yang

layak, yaitu kandidat tersebut bersama-sama dengan himpunan solusi tidak melanggar *constraints* yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang

e. Fungsi objektif

Fungsi yang digunakan untuk memaksimumkan atau meminimumkan solusi yang dimiliki

Algoritma greedy dapat kita temukan dalam kehidupan sehari-hari:

- *Travelling saleperson problem* dengan algoritma Kruskal.
- Mencari *minimum spanning tree* dengan algoritma Prim
- Menentukan waktu minimal untuk sistem *scheduling*

2.2 Game Engine Permainan *Overdrive*

Pada bab ini akan dijelaskan mengenai *game engine* yang digunakan untuk permainan *overdrive* secara keseluruhan yaitu bagaimana bot melakukan aksinya, bagaimana mengimplementasikan algoritma greedy ke dalam bot, bagaimana menjalankan game engine, dan lain sebagainya.

2.2.1 Prerekuisit

Menjalankan game engine di lokal dengan meng-*clone* repo pada laman berikut: <https://github.com/EntelectChallenge/2020-Overdrive>.

Sebagai tambahan, kita juga perlu menginstall java (minimal java 8), IntelliJ IDEA atau maven, dan NodeJS.

2.2.2 Configuration pada game-runner-config.json

Konfigurasi yang terdapat pada *game-runner-config.json* adalah sebagai berikut

```
1  {
2      "round-state-output-location": "./match-logs",
3      "game-config-file-location": "game-config.json",
4      "game-engine-jar": "game-engine.jar",
5      "verbose-mode": true,
6      "max-runtime-ms": 1000,
7      "player-b": "./srp-bot/java",
8      "player-a": "./chi-bot",
9      "max-request-retries": 10,
10     "request-timeout-ms": 5000,
11     "is-tournament-mode": false,
12     "tournament": {
13         "connection-string": "",
14         "bots-container": "",
15         "match-logs-container": "",
16         "game-engine-container": "",
17         "api-endpoint": "http://localhost"
18     }
19 }
20
```

round-state-output-location	Untuk tempat menyimpan hasil pertandingan tiap round
game-config-file-location	Tempat konfigurasi untuk keberjalan game
game-engine.jar	Untuk tempat eksekusi file
player-a	Untuk player pertama yang akan bertanding
player-b	Untuk player kedua yang akan bertanding
is-tournament-mode	Untuk mengecek apakah merupakan turnamen atau bukan
tournament	Jika merupakan turnamen, maka akan diberikan konfigurasi berupa connection ke server tempat turnamen dijalankan

Untuk konfigurasi yang lain seperti verbose-mode, max-runtime-ms, dan lain-lain merupakan konfigurasi default dari entelect untuk kita melakukan testing sehingga kami tidak tahu lebih lanjut tentang konfigurasi tersebut.

2.2.3 Bot.java

Bot.java merupakan tempat keberadaan seluruh logic dari *bot* yang akan bermain. Di dalamnya terdapat algoritma yang menunjang pengambilan keputusan oleh sebuah *bot*. Untuk menjalankan bot, kita harus meng-build *bot* tersebut menjadi file *.jar* yang nantinya dapat dijalankan oleh *game-engine*.

2.2.4 Pengimplementasian algoritma greedy ke dalam bot

Algoritma greedy yang diterapkan pada game ini dapat memiliki beberapa kemungkinan diantaranya yaitu greedy dengan memaksimalkan penggunaan *power ups*, memaksimalkan *speed*, memaksimalkan *score*, dan meminimalkan *damage*. Penjelasan mendetail terkait algoritma greedy pada permainan ini terdapat dalam bab selanjutnya yaitu bab 3 laporan ini.

2.2.5 Menjalankan *game engine*

Untuk menjalankan *game engine*, kita pertama harus mengatur beberapa konfigurasi pada *game-runner-config.json* untuk menentukan player yang akan bertanding. Lalu tinggal menggunakan command “*make run*” pada command prompt, lalu *game* akan berjalan.

BAB 3

APLIKASI STRATEGI GREEDY

3.1 Mapping persoalan *overdrive* menjadi elemen-elemen algoritma Greedy

Algoritma Greedy yang digunakan dalam persoalan *overdrive* kami dapat dibagi menjadi 2 unsur yaitu movement dan powerups.

Untuk movement, ada 4 bagian berdasarkan prioritasnya yaitu greedy untuk memaksimalkan *speed*, greedy untuk memaksimalkan *score*, greedy untuk meminimalkan *damage*, dan greedy untuk mendapatkan powerups lebih banyak.

Movement

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang mungkin dijalankan oleh mobil di dalam permainan. Perintah yang dimaksud dapat merupakan perintah untuk bergerak ke kiri, bergerak ke kanan, atau diam di jalur sekarang.
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan kondisi setiap turn permainan selama permainan berlangsung.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan.
Fungsi Seleksi (Selection Function)	Pemilihan perintah yang sesuai dengan prioritas strategi yang digunakan dalam bot yang telah dibangun dalam implementasi permainan Overdrive. Pembangunan prioritas dalam bahasa pemrograman bisa dilakukan dengan

	membuat fungsi penyeleksian yang terurut sesuai dengan prioritas strategi yang sudah ditentukan.
Fungsi Kelayakan (Feasibility)	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap komponen-komponen permainan, misalnya apakah terdapat obstacle di depan, kecepatan mobil player dan musuh sekarang, manuver mana yang akan menghasilkan speed terbesar, manuver mana yang akan membuat mobil mendapatkan powerups lebih banyak, jumlah damage sekarang, dan manuver mana yang akan menghasilkan damage terkecil, jumlah score sekarang, dan manuver mana yang akan menghasilkan score terbesar.
Fungsi Objektif	Menangkan permainan overdrive, dengan memperhatikan speed, damage, dan score yang dimiliki.

Penggunaan Powerups

Elemen Algoritma Greedy	Pemetaan pada Permainan Overdrive
Himpunan Kandidat (C)	Perintah-perintah yang mungkin dijalankan oleh mobil di dalam permainan. Perintah yang dimaksud adalah menggunakan powerups yang dimiliki atau tidak menggunakan powerups sama sekali.
Himpunan Solusi (S)	Perintah terbaik yang dipilih sesuai dengan kondisi setiap turn permainan selama permainan berlangsung.
Fungsi Solusi	Memeriksa bahwa perintah yang dipilih adalah perintah yang terdefinisi atau tidak di dalam permainan. Jika tidak, tentunya perintah ini bukan solusi yang valid untuk dijalankan.
Fungsi Seleksi (Selection Function)	Pemilihan perintah yang sesuai dengan kondisi mobil player dan musuh yang akan dicek kelayakannya setiap turn.
Fungsi Kelayakan (Feasibility)	Memeriksa bahwa semua kondisi yang diperlukan untuk mengeksekusi perintah yang dipilih sudah terpenuhi atau belum. Validasi dilakukan terhadap komponen-komponen permainan, misalnya object apakah yang ada di depan, speed mobil player dan musuh, kondisi boost player dan musuh, dan juga damage mobil player.
Fungsi Objektif	Menangkan permainan overdrive, dengan mengandalkan penggunaan powerups yang digunakan selama balapan berlangsung. Penerapan greedy

	di sini adalah akan selalu menggunakan powerups yang tersedia dan layak sesuai kondisi di setiap turn berlangsung.
--	--

3.2 Eksplorasi alternatif solusi greedy

Berdasarkan bot yang telah kami buat, Kami dapat menciptakan 4 strategi alternatif yang diperoleh berdasarkan objektif dari permainan *overdrive* ini, yaitu menang dengan memaksimalkan speed, menang dengan meminimalisasi damage, menang dengan penggunaan powerups, dan menang dengan score maksimal.

a. Menang dengan memaksimalkan speed

Dalam strategi ini mobil akan diarahkan untuk selalu bergerak dengan tujuan memperoleh speed maksimal agar dapat memenangkan permainan dengan mencapai garis finish terlebih dahulu. Pergerakan mobil akan selalu mengutamakan speed, mobil akan meminimalisasi pindah lane karena manuver itu akan mengurangi speed. Mobil akan pindah lane apabila terdapat object di depan yang akan mengurangi speed. Mobil juga akan mengutamakan mendapatkan boost dan akan langsung menggunakan boost untuk mencapai keadaan BOOST sesering mungkin.

Strategi ini secara kasar merupakan strategi yang paling masuk akal dalam game balapan yang tujuannya mencapai finish terlebih dahulu. Akan tetapi, dalam game *overdrive* ini ada damage yang harus diperhatikan juga karena apabila mobil masuk kedalam keadaan yang mengharuskan melakukan FIX, maka speed pun akan berkurang.

b. Menang dengan meminimalisasi damage

Dalam strategi ini mobil akan diarahkan untuk selalu bergerak dengan tujuan memperoleh damage seminimal mungkin agar dapat memenangkan permainan dengan memanfaatkan sesedikit mungkin melakukan FIX. Pergerakan mobil akan selalu mengutamakan minimalsasi damage, Mobil akan melakukan manuver pindah lane ke lane yang lebih sedikit memberikan damage pada mobil. Mobil juga akan mengutamakan menghindari obstacle agar tidak perlu melakukan FIX.

Strategi ini dapat dibilang lebih baik daripada memaksimalkan speed karena apabila mobil jarang melakukan FIX, maka akan semakin sedikit juga mobil berhenti dan speed pun dapat secara maksimal dimanfaatkan disetiap turn.

c. Menang dengan penggunaan powerups

Strategi penggunaan powerups akan memaksimalkan pergerakan mobil untuk mendapatkan powerups sebanyak-banyaknya dan menggunakannya secara langsung saat kondisi yang ditentukan sebagai syarat penggunaan powerup sudah tercapai. Strategi ini memasukan pergerakan lawan sebagai faktor dari kemenangan karena beberapa powerups dapat memberikan *disadvantage* kepada mobil lawan, seperti mengurangi speed, dan memberikan damage pada lawan. Strategi ini juga dapat menjadi strategi yang paling menguntungkan karena powerups merupakan bagian penting dalam game *overdrive* karena speed dan damage dapat dengan mudah didapatkan saat menggunakan powerups.

d. Menang dengan meminimalkan pengurangan score.

Strategi meminimalkan pengurangan score akan menjadi prioritas terakhir dari strategi kami. Dengan strategi ini, kita dapat memilih lane mana yang paling sedikit mendapatkan pengurangan score jika tertabrak *obstacle-obstacle* tertentu. Strategi ini bekerja dengan menghitung terlebih dahulu minus score yang diperoleh di lane tertentu yang nantinya akan dibandingkan dan yang minusnya terendah akan dipilih.

3.3 Analisis efisiensi dari kumpulan alternatif solusi greedy yang dirumuskan

Sebenarnya dalam game ini, kompleksitas waktu tidak dipentingkan sebab *game* telah didesain untuk menerima perintah dari dua *player* dan menjalankan secara bersamaan. Meskipun demikian, efisiensi dari algoritma ini perlu dianalisis sehingga akan menghasilkan algoritma yang lebih mangkus dan lebih baik lagi.

Untuk strategi *movement*, kita harus mengecek beberapa alternatif jalur yang harus dipilih oleh bot. Dengan demikian, jalur yang akan dipilih oleh bot merupakan jalur dengan keuntungan terbesar sesuai dengan kriteria tertentu. Dalam setiap iterasi, bot akan menganalisis maksimal 3 lane sebanyak 4 iterasi sesuai dengan *speed*, *damage*, *score*, dan *powerups* secara berturut. Maka kompleksitasnya adalah $O(12n) = O(n)$ dengan n adalah banyak block yang dianalisis oleh bot (dengan maksimal block yang dapat dianalisis adalah 15 block kedepan). Dengan begitu, strategi analisis *movement* sudah lumayan mangkus karena hanya mengecek maksimal 180 kasus iterasi dalam kasus terburuk.

Untuk strategi *powerups*, kita harus mengecek setiap *powerups* dan menentukan apakah *powerups* perlu digunakan atau tidak. Algoritma ini memiliki perbedaan untuk setiap *powerups*. Misalnya untuk fungsi *shouldCarUseLizard()*, yang akan mengecek apakah bot harus menggunakan *powerups lizard* atau tidak. Kompleksitas untuk setiap algoritma ini adalah $O(n)$, dengan n adalah banyak block yang dianalisis oleh bot (dengan maksimal block 15).

3.4 Analisis efektivitas dari kumpulan alternatif solusi greedy yang dirumuskan

Sebenarnya dalam game ini, kompleksitas waktu tidak dipentingkan sebab *game* telah didesain untuk menerima perintah dari dua *player* dan menjalankan secara bersamaan. Meskipun demikian, efisiensi dari algoritma ini perlu dianalisis sehingga akan menghasilkan algoritma yang lebih mangkus dan lebih baik lagi.

Kombinasi strategi yang kami gunakan adalah penggabungan antara greedy *movement* dan *powerups*. Kami merasa bahwa kombinasi ini merupakan kombinasi yang cukup baik. Dengan strategi *movement*, bot dapat memilih jalur terbaik yang harus dilalui oleh bot, dan dengan strategi *powerups*, bot dapat memilih *powerups* apa yang harus dijalankan jika sudah berada pada jalur terbaik. Buktinya, strategi tersebut sudah berhasil mengalahkan *reference bot* yang telah disediakan oleh Entelect. Hal ini dikarenakan bot sudah didesain untuk berjalan di jalur terbaik (tanpa halangan) dan penggunaan *powerups* terbaik dalam tiap keadaan.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi algoritma greedy pada program bot dalam game engine

```
● ● ●
1 function Run(GameState gameState) -> Command
2 { Menjalankan public/method yang mengembilkan command langkah yang dipilih berdasarkan algoritma yang telah didesain }
3
4 KAMUS LOKAL
5   gameState = GameState { Berisi keadaan game saat round tertentu }
6   ACCELERATE = AccelerateCommand() { Command untuk menjalankan akselerasi }
7   LIZARD = LizardCommand() { Command untuk menjalankan lizard }
8   OIL = OilCommand() { Command untuk menjalankan oil }
9   BOOST = BoostCommand() { Command untuk menjalankan boost }
10  EMP = EMPCommand() { Command untuk menjalankan EMP }
11  FIX = FixCommand() { Command untuk menjalankan FIX }
12
13 ALGORITMA
14   { assign player ke variable myCar }
15   myCar <- gameState.player;
16
17   { assign opponent ke variable opponent }
18   opponent <- gameState.opponent;
19
20   { assign list of blocks di depan myCar }
21   blocks <- getBlocksInFront(gameState, myCar.position.lane, myCar.position.block)
22
23   { jika damage dari myCar lebih dari 2, dan di depan myCar tidak ada wall, mud, dan oil spill maka akan FIX }
24   if (myCar.damage >= 2 && !blocks.contains(WALL) && !blocks.contains(MUD) && !blocks.contains(OIL_SPILL) then
25     return FIX
26
27   { jika damage dari myCar lebih dari 4, maka akan return FIX }
28   if (myCar.damage >= 4) then
29     return FIX
30
31   { jika speed dari myCar 0, maka harus dijalankan ACCELERATE untuk menjalankan mobil kembali }
32   if (myCar.speed == 0) then
33     return ACCELERATE
34
35   { untuk mengambil lane yang paling menguntungkan }
36   lane <- checkBestPosition(gameState, myCar)
37
38   { jika lane paling menguntungkan bukan merupakan lane bot saat ini }
39   { maka bot akan berubah arah menjadi lane yang paling menguntungkan }
40   if (Lane != myCar.lane) then
41     return ChangeLaneCommand(lane)
42
43   if (shouldCarUseLizard(gameState, myCar) then
44     return LIZARD
45
46   if (shouldCarUseEMP(gameState, myCar) then
47     return EMP
48
49   if (shouldCarUseBoost(gameState, myCar) then
50     return BOOST
51
52   if (shouldCarUseTweet(gameState, myCar) then
53     return TweetCommand(opponent.lane, opponent.block + opponent.speed + 2)
54
55   if (shouldCarUseOil(gameState, myCar) then
56     return OIL
57
58   return ACCELERATE
```

1. CheckBestPosition()

```
● ● ●

1 function CheckBestPosition(GameState gameState, Car myCar) -> integer
2 { fungsi untuk menentukan lane terbaik yang akan dipilih oleh bot berdasarkan konsiderasi tertentu }
3
4 KAMUS LOKAL
5     priorities = Array of string
6     blocksInFront = Array of Object
7     blocksInLeft = Array of Object
8     blocksInRight = Array of Object
9     left = integer
10    right = integer
11    front = integer
12
13 ALGORITMA
14     priorities <- ["damage", "speed", "score", "powerups"]
15     blocksInFront <- getBlocksInFront(gameState, myCar.lane, myCar.blocks)
16
17     { looping untuk setiap elemen pada array dan menamai tiap elemen = priority }
18     for (Object priority : priorities) do
19         if (priority == "speed") then
20
21             { jika bot ada di lane 2 atau 4, yang artinya bot dapat bergerak ke kanan dan kiri }
22             if (myCar.lane == 2 || myCar.lane == 4) then
23
24                 { untuk mendapatkan object blocks di kanan dan kiri bot }
25                 blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
26                 blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
27
28                 { menghitung nilai pengurangan speed untuk lane kiri, kanan, dan depan }
29                 front <- countSpeedDecrement(gameState, myCar, blocksInFront)
30                 left <- countSpeedDecrement(gameState, myCar, blocksInLeft)
31                 right <- countSpeedDecrement(gameState, myCar, blocksInRight)
32
33                 { perbandingan antara lane kiri, kanan, dan depan }
34                 { untuk menentukan lane terbaik }
35                 if (left < front && left < right) then
36                     return myCar.lane - 1
37
38                 if (front < left && front < right) then
39                     return myCar.lane
40
41                 if (right < left && right < front) then
42                     return myCar.lane
43
44             else if (myCar.lane == 1) then
45                 blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
46                 right <- countSpeedDecrement(gameState, myCar, blocksInRight)
47                 front <- countSpeedDecrement(gameState, myCar, blocksInFront)
48
49                 if (right < front) then
50                     return myCar.lane + 1
51                 else
52                     return myCar.lane
53
54             else
55                 blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
56                 front <- countSpeedDecrement(gameState, myCar, blocksInFront)
57                 left <- countSpeedDecrement(gameState, myCar, blocksInLeft)
58
59                 if (left < front) then
60                     return myCar.lane - 1
61                 else
62                     return myCar.lane
63
```

```
● ● ●

1     if (priority == "score")
2
3     { jika bot ada di lane 2 atau 4, yang artinya bot dapat bergerak ke kanan dan kiri }
4     if (myCar.lane == 2 || myCar.lane == 4) then
5
6         { untuk mendapatkan object blocks di kanan dan kiri bot }
7         blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
8         blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
9
10    { menghitung nilai pengurangan score untuk lane kiri, kanan, dan depan }
11    front <- countScoreDecrement(gameState, myCar, blocksInFront)
12    left <- countScoreDecrement(gameState, myCar, blocksInLeft)
13    right <- countScoreDecrement(gameState, myCar, blocksInRight)
14
15    { perbandingan antara lane kiri, kanan, dan depan }
16    { untuk menentukan lane terbaik }
17    if (left < front && left < right) then
18        return myCar.lane - 1
19
20    if (front < left && front < right) then
21        return myCar.lane
22
23    if (right < left && right < front) then
24        return myCar.lane
25
26    else if (myCar.lane == 1) then
27        blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
28        right <- countScoreDecrement(gameState, myCar, blocksInRight)
29        front <- countScoreDecrement(gameState, myCar, blocksInFront)
30
31        if (right < front) then
32            return myCar.lane + 1
33        else
34            return myCar.lane
35
36    else
37        blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
38        front <- countScoreDecrement(gameState, myCar, blocksInFront)
39        left <- countScoreDecrement(gameState, myCar, blocksInLeft)
40
41        if (left < front) then
42            return myCar.lane - 1
43        else
44            return myCar.lane
```

Dapat dilihat dari gambar di atas, bot sudah berhasil mendeteksi cybertruck di depannya, dan menghindar ke jalur terbaik (dapat dilihat juga bahwa jalur di atas terdapat obstacle mud).

```
● ● ●
1     if (priority == "damage")
2
3         { jika bot ada di lane 2 atau 4, yang artinya bot dapat bergerak ke kanan dan kiri }
4         if (myCar.lane == 2 || myCar.lane == 4) then
5
6             { untuk mendapatkan object blocks di kanan dan kiri bot }
7             blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
8             blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
9
10            { menghitung nilai pengurangan damage untuk lane kiri, kanan, dan depan }
11            front <- countDamageDecrement(gameState, myCar, blocksInFront)
12            left <- countDamageDecrement(gameState, myCar, blocksInLeft)
13            right <- countDamageDecrement(gameState, myCar, blocksInRight)
14
15            { perbandingan antara lane kiri, kanan, dan depan }
16            { untuk menentukan lane terbaik }
17            if (left < front && left < right) then
18                return myCar.lane - 1
19
20            if (front < left && front < right) then
21                return myCar.lane
22
23            if (right < left && right < front) then
24                return myCar.lane
25
26        else if (myCar.lane == 1) then
27            blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
28            right <- countDamageDecrement(gameState, myCar, blocksInRight)
29            front <- countDamageDecrement(gameState, myCar, blocksInFront)
30
31            if (right < front) then
32                return myCar.lane + 1
33            else
34                return myCar.lane
35
36        else
37            blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
38            front <- countDamageDecrement(gameState, myCar, blocksInFront)
39            left <- countDamageDecrement(gameState, myCar, blocksInLeft)
40
41            if (left < front) then
42                return myCar.lane - 1
43            else
44                return myCar.lane
```

```
1      if (priority == "powerups")
2
3          { jika bot ada di lane 2 atau 4, yang artinya bot dapat bergerak ke kanan dan kiri }
4          if (myCar.lane == 2 || myCar.lane == 4) then
5
6              { untuk mendapatkan object blocks di kanan dan kiri bot }
7              blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
8              blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
9
10         { menghitung nilai penambahan powerups untuk lane kiri, kanan, dan depan }
11         front <- countGetPowerUps(gameState, myCar, blocksInFront)
12         left <- countGetPowerUps(gameState, myCar, blocksInLeft)
13         right <- countGetPowerUps(gameState, myCar, blocksInRight)
14
15         { perbandingan antara lane kiri, kanan, dan depan }
16         { untuk menentukan lane terbaik }
17         if (left > front && left > right) then
18             return myCar.lane - 1
19
20         if (front > left && front > right) then
21             return myCar.lane
22
23         if (right > left && right > front) then
24             return myCar.lane
25
26     else if (myCar.lane == 1) then
27         blocksInRight <- getBlocksInFront(gameState, myCar.lane+1, myCar.block-1)
28         right <- countGetPowerUps(gameState, myCar, blocksInRight)
29         front <- countGetPowerUps(gameState, myCar, blocksInFront)
30
31         if (right > front) then
32             return myCar.lane + 1
33         else
34             return myCar.lane
35
36     else
37         blocksInLeft <- getBlocksInFront(gameState, myCar.lane-1, myCar.block-1)
38         front <- countGetPowerUps(gameState, myCar, blocksInFront)
39         left <- countGetPowerUps(gameState, myCar, blocksInLeft)
40
41         if (left > front) then
42             return myCar.lane - 1
43         else
44             return myCar.lane
45
46 return myCar.lane
```

2. ShouldCarUseLizard()

```
● ● ●
1 function shouldCarUseLizard(GameState gameState, Car myCar) -> boolean
2 { function untuk menentukan apakah bot akan menggunakan command LIZARD atau tidak }
3
4 KAMUS LOKAL
5     blocks = Array of object
6     cyberTruckInFront = boolean
7
8 ALGORITMA
9     { assign block di depan bot }
10    blocks <- getBlocksInFront(gameState, myCar.lane, myCar.blocks)
11
12    { mengecek apakah ada cybertruck di depan bot}
13    cyberTruckInFront <- getCyberTruckInFront(gameState, myCar.lane, myCar.blocks)
14
15    { mengecek apakah myCar memiliki LIZARD POWERUP }
16    if (hasPowerUp(LIZARD, myCar.powerups)) then
17
18        { jika kecepatannya 15 maka akan mengembalikan command LIZARD berdasarkan }
19        { jika tidak ada wall, mud, oil_spill, dan cybertruck}
20        if (myCar.speed == 15) then
21            return isWallInFrontOf(myCar) || blocks.contains(MUD) || blocks.contains(OIL_SPILL) || cyberTruckInFront
22        else
23            { pakai LIZARD jika ada wall dan jumlah mud di depan myCar lebih dari 3 }
24        return isWallInFrontOf(myCar)|| countMudInFrontOf(myCar) >= 3
```

3. ShouldCarUseEMP()

```
● ● ●
1 function shouldCarUseEMP(GameState gameState, Car myCar) -> boolean
2 { function untuk menentukan apakah bot akan menggunakan command EMP atau tidak }
3
4 ALGORITMA
5     { mengecek apakah myCar memiliki EMP POWERUP }
6     if (hasPowerUp(EMP, myCar.powerups)) then
7
8         { jika tidak ada wall dan kecepatan musuh lebih dari 9, maka akan langsung menggunakan EMP }
9         return !isWallInFrontOf(myCar) && opponent.speed >= 9
10
11    return false
12
```

4. ShouldCarUseBoost()

```
1 function shouldCarUseSpeedBoost(GameState gameState, Car myCar) -> boolean
2 { function untuk menentukan apakah bot akan menggunakan command SPEED_BOOST atau tidak }
3
4 ALGORITMA
5     { mengecek apakah myCar memiliki SPEED_BOOST POWERUP }
6     if (hasPowerUp(SPEED_BOOST, myCar.powerups)) then
7
8         { untuk instant boost }
9         { jika speed lebih dari 3, dan tidak ada wall di depan mobil }
10        { fungsi isWallInFrontOf(myCar) juga mengecek apakah ada cybertruck di depan car }
11        { maka akan menggunakan SPEED_BOOST }
12        if (myCar.speed >= 3 && !isWallInFrontOf(myCar)) then
13
14            { jika damage car lebih dari 2 }
15            { maka akan return FIX }
16            { karena jika tidak, maka akan boost sampai maksimal 9 speed }
17            if (myCar.damage >= 2) then
18                return FIX
19
20            { jika mud di depan car lebih dari 3 }
21            { maka akan return FIX }
22            { karena jika tidak, maka akan boost lalu akan berkurang maximal speednya menjadi 8 }
23            if (countMudInFrontOf(myCar) <= 3) then
24                return BOOST
25
26        return BOOST
27
28    return NOTHING
29
30    return false
```

5. ShouldCarUseTweet()

```
1 function shouldCarUseTweet(GameState gameState, Car myCar) -> boolean
2 { function untuk menentukan apakah bot akan menggunakan command TWEET atau tidak }
3
4 ALGORITMA
5     { mengecek apakah myCar memiliki TWEET POWERUP }
6     { dan apakah tidak ada wall dan mud yang lebih banyak dari 3 di depan car }
7     { untuk menghindari adanya tabrakan selagi menggunakan TWEET }
8     if (hasPowerUp(TWEET, myCar) && !isWallInFrontOf(myCar) && countMudInFrontOf(myCar) <= 3) then
9
10        { menggunakan TWEET berdasarkan kecepatan musuh }
11        { jika kecepatan musuh lebih dari 8, maka akan menggunakan TWEET }
12        return opponent.speed >= 8
13
14    return false
```

6. ShouldCarUseOil()

```
● ● ●
1 function shouldCarUseOil(GameState gameState, Car myCar) -> boolean
2 { function untuk menentukan apakah bot akan menggunakan command OIL atau tidak }
3
4 ALGORITMA
5     { mengecek apakah myCar memiliki OIL POWERUP }
6     { dan apakah tidak ada wall dan mud yang lebih banyak dari 3 di depan car }
7     { untuk menghindari adanya tabrakan selagi menggunakan OIL }
8     if (hasPowerUp(OIL, myCar) then
9
10        { mengecek apakah opponent di lane yang sama dan di depan tidak ada wall }
11        if (isOpponentOnTheSameLane(gameState, myCar) && !isWallInFrontOf(myCar) then
12            if (myCar.lane == 1) then
13                { Mengecek sebelah kanan dari lane myCar }
14                blocksInRight <- blocksInFrontOf(myCar.right.lane)
15
16                { Return true jika di sebelah kanan terdapat oil_spill, mud, wall, maupun cybertruck }
17                return blocksInRight.contains(OIL_SPILL) || blocksInRight.contains(MUD) || blocksInRight.contains(WALL)
18
19            if (myCar.lane == 4) then
20                { Mengecek sebelah kiri dari lane myCar }
21                blocksInLeft <- blocksInFrontOf(myCar.left.lane)
22
23                { Return true jika di sebelah kanan terdapat oil_spill, mud, wall, maupun cybertruck }
24                return blocksInLeft.contains(OIL_SPILL) || blocksInLeft.contains(MUD) || blocksInLeft.contains(WALL)
25
26        return false
```

4.2 Penjelasan struktur data dan struktur tambahan

Untuk algoritma strategi yang kami gunakan, tidak ada struktur data tambahan yang kami tambahkan untuk menunjang keberlangsungan strategi ini. Penggunaan struktur data yang kami manfaatkan hanyalah struktur data array yang bertujuan untuk menyimpan blocks.

4.3 Analisis desain solusi algoritma yang diimplementasikan pada setiap pengujian

Percobaan pengujian pada *game* ini, dilakukan dengan hanya dengan menjalankan ‘make run’ pada command line. Berikut beberapa pengujian yang sudah kami lakukan.

a. Pengujian I

Pada pengujian pertama kami melakukan pengujian untuk simple logic yang telah kami coba seperti fix logic. Kami memastikan bahwa *bot* akan selalu fix pada keadaan di mana damage *bot* lebih dari 3.

b. Pengujian II

Pada pengujian ke dua, kami sudah membuat seluruh strategi yang akan digunakan, namun masih ada beberapa hal yang perlu ditingkatkan seperti analisis keberadaan cybertruck di depan *bot*, yang menghasilkan keadaan *bot* yang terus tertabrak oleh cybertruck. Berikut bot kami yang selalu menabrak cybertruck jika ada cybertruck di depannya.

[5, 1]	[566, 1]	[567, 1]	[568, 1]	[569, 1]	[570, 1]	[571, 1]
[5, 2]	[566, 2]	[567, 2]	[568, 2]	[569, 2]	[570, 2]	[571, 2]
[5, 3]	[566, 3]	[567, 3] <PLAYER> <BOOST>	[568, 3]	<TRUCK> [569, 3]	[570, 3]	[571, 3]
[5, 4] <EET>	[566, 4]	[567, 4]	[568, 4]	[569, 4]	[570, 4]	[571, 4]

Selain analisis keberadaan cybertruck, ada juga keadaan lain seperti saat bot menabrak wall dua kali, dan setelah menabrak, keadaan speed bot adalah 0. Karena kami mengutamakan strategi movement, maka bot akan selalu *TURN_LEFT* (dikarenakan ada wall di depannya), tanpa melihat bahwa speed yang dia punya sekarang ada 0. Maka hal tersebut menyebabkan bot tidak akan melaju sampai bot lainnya menang.

c. Pengujian II

Pada pengujian ketiga, kami sudah meningkatkan performa dan memperbaiki kesalahan pada pengujian ke dua. Dihasilkan *bot* sebagai berikut.

[856, 1]	[857, 1]	[858, 1] <LIZARD>	[859, 1]	[860, 1]	[861, 1] <MUD>	[862, 1]	[863, 1]
[856, 2]	[857, 2] <PLAYER>	[858, 2]	[859, 2]	<TRUCK> [860, 2] <MUD>	[861, 2]	[862, 2]	[863, 2]
[856, 3]	[857, 3]	[858, 3]	[859, 3]	[860, 3]	[861, 3]	[862, 3]	[863, 3]
[856, 4]	[857, 4]	[858, 4]	[859, 4]	[860, 4]	[861, 4]	[862, 4]	[863, 4]

Dapat dilihat dari gambar di atas, *bot* sudah berhasil mendeteksi cybertruck di depannya, dan menghindar ke jalur terbaik (dapat dilihat juga bahwa jalur di atas terdapat obstacle mud).

Selain *improvement* pada cybertruck, kami juga mengubah beberapa prioritas pada *bot* kami sehingga dihasilkan prioritas yaitu mengutamakan pertambahan speed (sehingga kami harus menghitung jalur mana yang memiliki pengurangan kecepatan paling banyak untuk dihindari). Lalu setelah pertambahan speed, kami mengutamakan pengurangan *health* pada *bot* sehingga *bot* tidak harus menerima damage yang terlalu banyak dan akan berakhir pada keadaan FIX yang di mana harus mengorbankan satu round untuk melakukan command tersebut. Setelah speed dan damage, kami mengutamakan penambahan score dan pengambilan powerups.

Setelah strategi movement selesai, prioritas kami adalah penggunaan dari powerups. Untuk setiap powerups, kami membaginya menjadi fungsi masing-masing dengan kriteria masing-masing. Misalnya untuk mengecek apakah *bot* harus menggunakan powerup *TWEET* maka diberikan fungsi sendiri yaitu *ShouldCarUseTweet()* yang di dalamnya terdapat beberapa kondisi sehingga *bot* dapat menjalankan powerups tersebut. Dapat dilihat dari gambar di atas, bot sudah berhasil mendeteksi cybertruck di depannya, dan menghindar ke jalur terbaik (dapat dilihat juga bahwa jalur di atas terdapat obstacle mud).

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma yang mengangkat tema *game* balap yaitu “*overdrive*”, kami berhasil membuat bot yang memanfaatkan algoritma greedy sebagai strategi kami. Disamping algoritma greedy, untuk strategi yang terbaik, kami juga mengkombinasikan algoritma lain dalam penyusunan bot ini.

5.2 Saran

Pengaplikasian algoritma greedy pada permainan *overdrive* ini sudah dapat dikategorikan cukup berhasil. Tetapi, masih ada beberapa hal yang dapat dikembangkan sebagai ruang pembelajaran kami, salah satunya adalah adanya peluang untuk mengefisiensikan kode dari program karena untuk beberapa fungsi ada yang masih dapat dioptimasi. Selain itu, penggunaan perkembangan AI untuk pengimplementasian bot sehingga bot dapat mempelajari dari setiap pengujian yang ada demi pengambilan keputusan yang terbaik. Yang terakhir, penyertaan pseudocode yang nampaknya tidak terlalu diperlukan karena program sudah dibuat dengan *comment* dan *readability* yang cukup.

Daftar Pustaka

Entelect Forum. Entelect Worms 2019 Challenge. Diakses pada 4 Februari 2020 pukul 18.21 WIB melalui <https://forum.entelect.co.za/>

Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). When the greedy algorithm fails. *Discrete optimization*, 1(2), 121-127

GeeksforGeeks. 2022. *Greedy Algorithms* - GeeksforGeeks. [online]. Diakses pada 10 Februari 2022 pukul 11.32 WIB melalui <https://www.geeksforgeeks.org/greedy-algorithms/>

Munir, R. (2020). Algoritma Greedy Bagian 1[PDF]. Institut Teknologi Bandung. Diakses pada 8 Februari 2022 pukul 14.00 WIB melalui [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf).

Munir, R. (2020). Algoritma Greedy Bagian 2[PDF]. Institut Teknologi Bandung. Diakses pada 8 Februari 2022 pukul 14.20 WIB melalui [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf).

Munir, R. (2020). Algoritma Greedy Bagian 3[PDF]. Institut Teknologi Bandung. Diakses pada 8 Februari 2022 pukul 14.37 WIB melalui [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf)

LINK GITHUB

<https://github.com/samuelswandi/MobilBalap>