

Tugas Besar I IF2211 Strategi Algoritma
Semester II Tahun 2021/2022
Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan
“Overdrive”



Disusun Oleh:

13520110 - Farrel Ahmad
13520143 Muhammad Gerald Akbar Giffera
13520148 - Fikri Ihsan

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2022

BAB 1	
Deskripsi Tugas	3
BAB 2	
Landasan Teori	5
2.1 Algoritma Greedy	5
2.2 Cara Kerja Program	5
BAB 3	
Aplikasi Strategi Greedy	6
3.1 Elemen Algoritma Greedy dalam Game Overdrive	6
3.1.1 Himpunan Kandidat	6
3.1.2 Himpunan Solusi	6
3.1.3 Fungsi Solusi	6
3.1.4 Fungsi Seleksi	6
3.1.5 Fungsi Kelayakan	6
3.1.6 Fungsi Objektif	6
3.2 Eksplorasi Alternatif Solusi	6
3.3 Analisis Efisiensi dan Efektivitas	6
3.4 Strategi yang dipilih	6
BAB 4	
Implementasi dan Pengujian	7
4.1 Implementasi algoritma greedy pada program	7
4.2 Struktur data yang digunakan	7
4.3 Analisis Solusi	7
BAB 5	
Kesimpulan dan Saran	8
5.1 Kesimpulan	8
5.2 Saran	8
DAFTAR PUSTAKA	9

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
- j. USE_EMP
- k. FIX

BAB 2

Landasan Teori

2.1 Algoritma Greedy

Algoritma Greedy adalah algoritma yang menyelesaikan suatu persoalan secara langkah per langkah, dengan setiap langkahnya mengambil pilihan terbaik yang dapat dipilih, tanpa memperhatikan konsekuensi ke depan. Dengan pendekatan ini, diharapkan bahwa solusi yang dihasilkan merupakan solusi yang optimal secara global. Akan tetapi, Algoritma greedy tidak selalu menghasilkan solusi yang paling optimal untuk semua jenis persoalan.

Dalam Algoritma Greedy terdapat beberapa elemen-elemen yang dapat kita definisikan sesuai dengan persoalan yang kita hadapi, untuk membantu menemukan solusi yang optimal. Elemen-elemen tersebut yaitu:

- a. Himpunan Kandidat (C)
Himpunan Kandidat adalah sebuah himpunan yang berisi langkah yang mungkin diambil oleh algoritma untuk setiap tahap.
- b. Himpunan Solusi (S)
Himpunan Solusi adalah himpunan yang berisi elemen dari himpunan kandidat yang terpilih sebagai solusi dari tahap tertentu.
- c. Fungsi Solusi
Fungsi Solusi menentukan apakah himpunan kandidat yang dipilih dapat memberikan solusi dari langkah/tahap tersebut.
- d. Fungsi Seleksi
Fungsi Seleksi merupakan sebuah fungsi yang memilih kandidat sesuai dengan strategi greedy yang diimplementasikan.
- e. Fungsi Kelayakan
Fungsi Kelayakan merupakan sebuah fungsi yang memeriksa apakah sebuah kandidat dapat dimasukkan ke dalam himpunan solusi.
- f. Fungsi Objektif
Fungsi Objektif menyatakan kondisi yang ingin diperoleh dari algoritma, pada konteks algoritma greedy yaitu memaksimalkan atau meminimumkan

2.2 Cara Kerja Program

Game Entelect Challenge 2020 ini bekerja dengan menjalankan game engine yang telah dibuat. Game engine ini akan menjalankan dua buah bot yang kemudian akan diadu balapan satu sama lain. Setelah dijalankan, game engine akan memberikan gamestate untuk masing-masing bot. Bot akan membaca gamestate yang berisi informasi state aktivitas robot sekarang, posisi robot, posisi musuh, dan state lainnya. Berdasarkan informasi state-state ini, bot akan memilih strategi yang tepat dan akan memberikan command ke game engine untuk menggerakkan mobil bot masing-masing. Algoritma greedy dipakai untuk menentukan command. Command yang akan dipilih diimplementasikan pada bot.java sehingga algoritma greedy ada pada file tersebut, sedangkan bot akan mendapatkan gamestate dan dikirim ke bot.java dari main.java yang dijalankan.

BAB 3

Aplikasi Strategi Greedy

3.1 Strategi Algoritma Greedy yang Mungkin Dipilih

3.1.1 Greedy by damage

Greedy by damage adalah strategi pemilihan *command* dengan pertimbangan *damage* yang diterima mobil dan score yang didapat. Pada beberapa *obstacles* seperti *mud* dan *wall* mempengaruhi score yang didapat, dimana score tersebut berguna untuk penentuan pemenang. Sedangkan *damage* yang diterima mempengaruhi batasan kecepatan maksimum mobil.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat : *command FIX*
- Himpunan Solusi : *command FIX* digunakan
- Fungsi Solusi : Memeriksa apakah masih ada kerusakan yang berakibat memperlambat laju mobil.
- Fungsi Seleksi : gunakan *FIX* , jika masih terdapat kerusakan maksimal 5.
- Fungsi Kelayakan : Memeriksa berapakah kecepatan dan kerusakan pada mobil
- Fungsi Objektif : Memaksimalkan kecepatan dengan menghilangkan batas maksimum yang disebabkan kerusakan pada mobil

b. Analisis Efisiensi Solusi

Seleksi yang digunakan dengan memeriksa berapa kecepatan yang dimiliki mobil sekarang dan berapa poin kerusakan yang dimiliki mobil. Bagi kondisi kerusakan dan kecepatan menjadi n kondisi. Pemeriksaan ini memiliki efisiensi waktu algoritma :

$$T(n) = n = O(n)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila :

- Mobil memiliki kerusakan
- Kecepatan mobil tidak tidak meningkat

Strategi ini tidak efektif apabila :

- Mobil tidak ada kerusakan

3.1.2 Greedy by speedv1

Greedy by speed adalah strategi memaksimalkan speed dari mobil. Pada versi ini, hal ini dilakukan dengan menggunakan boost yang diperoleh sepanjang lintasan dan memaksimalkan penggunaannya. Jika pada suatu tahap mobil mempunyai boost, dan sepanjang lintasan tersebut tidak ada *obstacle* yang menghalangi, maka boost akan digunakan untuk memaksimalkan kecepatan mobil.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: *command USE_BOOST*
 - Himpunan Solusi: *command USE_BOOST* digunakan
 - Fungsi Solusi: Memeriksa apakah sepanjang lane dimana mobil berada tidak ada *obstacle* yang akan menghalangi penggunaan *BOOST* secara maksimal.
 - Fungsi Seleksi: gunakan *USE_BOOST*, jika tidak ada *obstacle* yang menghalangi di lane.
 - Fungsi Kelayakan: Memeriksa apakah mobil sudah mempunyai *BOOST*
 - Fungsi Objektif: Memaksimalkan speed dengan memaksimalkan penggunaan *BOOST* selama dan sejauh mungkin.
- b. Analisis Efisiensi Solusi
- Seleksi yang dilakukan adalah dengan memeriksa keadaan didepan mobil sejauh n *block*. Lakukan loop sejauh *visibility* mobil(1), lalu periksa apakah setiap *block i* yang diakses pada loop mengandung salah satu dari 3 *obstacle* yang tersedia. Pemeriksaan ini menggunakan fungsi buatan sendiri dengan efisiensi waktu algoritma:
- $$T(n) = n * 3 = O(n)$$
- c. Analisis Efektivitas Solusi
- Strategi ini efektif apabila:
- Lane mobil tidak mengandung *obstacle* sejauh *visibility* mobil
 - Tidak harus banyak berpindah lane
- Strategi ini tidak efektif apabila:
- Banyak *obstacle* yang dihadapi di lane yang sama maupun di lane yang lain

3.1.3 Greedy by obstacle

Greedy by obstacle merupakan strategi algoritma greedy mempunyai tujuan untuk meminimalisir kontak bot mobil dengan *obstacle-obstacle* yang ada. Maka pada dasarnya, mobil akan memilih lane dengan *obstacle* yang paling sedikit sejauh jarak pandangnya dengan mengubah lane tempat mobil berada.

- a. Mapping Elemen-Elemen Greedy
- Himpunan Kandidat: *command TURN_LEFT, TURN_RIGHT, dan ACCELERATE*
 - Himpunan Solusi: Mobil menggunakan *command* yang meminimalisir adanya tabrakan dengan *obstacle*
 - Fungsi Solusi: Memeriksa lane mana yang tidak ada mempunyai *obstacle* di sepanjang jalan mobil
 - Fungsi Seleksi: *TURN_LEFT* jika di lane sebelah kiri mobil tidak ada halangan, *TURN_RIGHT* jika di lane sebelah kanan mobil tidak ada halangan, *ACCELERATE* jika di lane kanan, kiri dan depan mobil terdapat halangan.
 - Fungsi Kelayakan: Memeriksa apakah *command* yang digunakan dapat menghindari *obstacle*
 - Fungsi Objektif: Meminimalisir kontak dengan *obstacle*
- b. Analisis Efisiensi Solusi
- Terdapat sejumlah n *block* yang harus diperiksa dimana n adalah kecepatan dari bot mobil, dan juga sejumlah 2-3 lane yang harus diperiksa sebanyak $n-1$ *block*. Setiap lane harus diperiksa jika terdapat salah satu dari 3 *obstacle*. Sehingga kompleksitasnya adalah:

$$T(n) = 2 \cdot 3 \cdot (n-1) \cdot 3 + n \cdot 3 = O(n)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Tidak ada *obstacle* yang berdekatan baik di lane tempat mobil berada maupun di lane samping mobil

Strategi ini tidak efektif apabila:

- Terdapat banyak *obstacle* yang jaraknya saling berdekatan.

3.1.4 Greedy by offensive power up

Greedy by power up adalah strategi untuk memutuskan tentang penggunaan *offensive power up* yang dimiliki. Beberapa *power up* digunakan sebagai bentuk penyerangan kepada mobil musuh. Power up tersebut terdiri dari *oil*, *tweet*, dan *emp*. Oil dan tweet digunakan untuk memberikan *obstacle* tambahan kepada musuh, sedangkan emp berguna untuk menghentikan lawan serta mengubah kecepatan lawan menjadi 3.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: *command EMP*, *command OIL*, *command TWEET*
- Himpunan Solusi: mobil menggunakan *offensive power up* untuk memperlambat laju lawan.
- Fungsi Solusi: Mengecek posisi lawan berada
- Fungsi Seleksi: *command EMP* jika lawan berada di depan mobil pemain, *command oil* jika lawan berada di belakang mobil kita dan satu jalur, *command TWEET* jika kita didepan lawan.
- Fungsi Kelayakan: Mengevaluasi apakah *power up* dapat berdampak pada laju musuh.
- Fungsi Objektif: Meminimalisir jarak jika kita di belakang mobil musuh dan Memaksimalkan jarak dan memberikan damage jika kita di depan mobil musuh.

b. Analisis Efisiensi Solusi

Terdapat sejumlah n blok dimana n adalah jarak pandang ke depan dan ke belakang serta terdapat 4 jalur yang berbeda. Terdapat 3 buah *offensive power up* yang harus di cek apakah bot mobil memiliki *offensive power up* tersebut atau tidak. Pada strategi ini juga diharuskan untuk mencari posisi lawan pada jarak pandang tersebut lalu memutuskan *offensive power up* mana yang akan dipakai

$$T(n) = n \cdot 4 \cdot 3 - 1 = O(n)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Mobil musuh berdekatan dengan bot mobil
- Mobil musuh berada pada jarak yang dapat dilihat oleh bot mobil

Strategi ini tidak efektif apabila:

- Posisi mobil musuh jauh dari bot mobil

3.1.5 Greedy by power up pickup

Greedy by power up pickup merupakan strategi algoritma greedy dimana bot mobil akan memilih gerakan baik itu tetap berada di lane yang sama, maupun berpindah berdasarkan powerup yang dapat diperoleh bot mobil.

a. Mapping Elemen-Elemen Greedy

- Himpunan Kandidat: *command TURN_LEFT, TURN_RIGHT dan ACCELERATE*
- Himpunan Solusi: Mobil menggunakan command yang dapat mengambil powerup sebanyak mungkin
- Fungsi Solusi: Memeriksa dimana tempat powerup terbanyak berada di antara lane yang dapat diakses oleh mobil pada posisi tertentu
- Fungsi Seleksi: ACCELERATE jika terdapat power up di lane mobil berada, TURN_LEFT jika ada power-up di lane sebelah kiri mobil, TURN_RIGHT jika terdapat power-up di lane sebelah kanan mobil. Namun jika terdapat beberapa power-up di jangkauan mobil, maka diutamakan untuk mengambil lane yang terdapat power-up *BOOST*.
- Fungsi Kelayakan: *command* dapat membantu mobil mendapatkan power-up sebanyak mungkin
- Fungsi Objektif: Mengambil power-up sebanyak mungkin

b. Analisis Efisiensi Solusi

Terdapat sejumlah *n block*, dimana *n* adalah kecepatan dari bot mobil, di 2-3 lane yang berbeda tergantung pada posisi mobil berada. Pada setiap *block* diperiksa jika pada block tersebut terdapat salah satu dari 5 power-up yang tersedia. Karena strategi ini mengutamakan memperoleh power-up *BOOST*, maka pada worst-case scenario yaitu pada kasus semua lane yang diakses terdapat power up yang dapat diperoleh. Lakukan loop sebanyak *n* di 3 lane berbeda(1), periksa lane mana yang mempunyai BOOST(2), utamakan pindah ke lane yang mempunyai BOOST(3). Sehingga akan diperoleh kompleksitas algoritma:

$$T(n) = 3*5*(n-1)(n-1) + 5*n*n = O(n^2)$$

c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:

- Power-up yang diperoleh oleh mobil juga dapat mempercepat mobil
- Tidak ada obstacle yang menghalangi jalan mobil dalam mengambil power-up

Strategi ini tidak efektif apabila:

- Banyak *obstacle* yang menghalangi jalan mobil
-

3.1.6 Greedy by speedv2

Greedy by speedv2 merupakan algoritma greedy dimana bot akan melaksanakan command accelerate sesering mungkin, sehingga dengan begitu kecepatan dari bot mobil yang di implementasikan akan selalu mencapai ataupun mendekati kecepatan maksimal

- a. Mapping Elemen-Elemen Greedy
 - Himpunan Kandidat: *command ACCELERATE*
 - Himpunan Solusi: *command ACCELERATE digunakan*
 - Fungsi Solusi: Memeriksa apakah tidak ada *command* lain yang dapat dilakukan oleh bot
 - Fungsi Seleksi: *ACCELERATE* jika tidak ada *obstacle*, atau tidak punya power up, atau terjebak diantara *obstacle*.
 - Fungsi Kelayakan: *command* dapat mempercepat mobil sesering dan secepat mungkin
 - Fungsi Objektif: Memaksimalkan kecepatan mobil
- b. Analisis Efisiensi Solusi

Pada strategi ini, bot mobil akan mempercepat kecepatannya sesering mungkin, sehingga bisa mencapai kecepatan maksimum sesering mungkin. Dalam pendekatan ini hanya dilakukan pemeriksaan apakah tidak ada kemungkinan lain yang dapat dilakukan oleh mobil sehingga dapat
- c. Analisis Efektivitas Solusi

Strategi ini efektif apabila:
Strategi ini tidak efektif apabila:

3.2 Solusi Algoritma yang dipilih

Pada game overdrive ini, tentunya banyak strategi algoritma yang bisa diimplementasikan untuk mencapai solusi yang optimal. Pemilihan ini tentunya memerlukan pertimbangan berdasarkan kondisi mobil baik itu *obstacle*, *damage* atau *power-up* yang dimiliki.

Pada implementasi program kami, kami menggunakan semua strategi greedy yang disebutkan sebelumnya, berdasarkan kasus atau skenario yang dihadapi oleh bot mobil. Jika mobil mempunyai damage yang besar, maka akan dilakukan perbaikan dengan command FIX, karena jika tidak maka mobil lama-kelamaan akan tidak bisa bergerak dan mengakibatkannya menjadi sangat merugikan. Untuk memaksimalkan kecepatan mobil, maka bot mobil akan selalu berusaha untuk memaksimalkan penggunaan BOOST sejauh dan selama mungkin. Untuk menangani kasus dimana ada halangan *obstacle* pada lane, maka akan diusahakan untuk menghindari dari *obstacle* yang ada di depan dan berpindah ke lane lain, jika tidak memungkinkan pindah maka kecepatan mobil akan ditambah dengan harapan pengurangan kecepatan yang diperoleh dari menabrak *obstacle* tidak akan terlalu signifikan. Untuk kasus dimana ada powerup yang dapat diperoleh bot mobil, maka pergerakan bot mobil akan didasarkan pada lane tempat powerup tersebut berada. Lalu untuk menangani skenario dimana bot mobil mempunyai powerup yang bersifat offensive (dapat mengganggu pergerakan bot lawan), maka powerup akan digunakan jika sedang berada di dalam satu lane yang sama, dengan harapan bahwa kemungkinan powerup tersebut mengenai mobil lawan lebih besar. Lalu kasus-kasus dimana tidak termasuk dalam kasus yang disebutkan diatas, ataupun

penyelesaiannya tidak diimplementasikan, maka bot akan melakukan *ACCELERATE*, hal ini dilakukan dengan tujuan memaksimalkan kecepatan dari bot.

BAB 4

Implementasi dan Pengujian

4.1 Pseudocode

Function decide(Lane[] laneStraight, Lane[] laneRight, Lane[] laneLeft, int x, int y, Car myCar, Car opponent) → Command

Deklarasi

```
clearStraight, haveTweet, haveOilPow, haveEMP, haveBoost, haveLizar
: Boolean
x_op, y_op : int
clearRight, clearLeft, clearFront : boolean
powerUpRight, powerUpLeft, powerUpFront: boolean
hasBoostFront, hasBoostLeft, hasBoostRight : boolean
```

Algoritma

```
clearStraight ← laneObstacleClear(laneStraight, x, myCar.speed)
haveTweet ← available(PowerUps.TWEET, myCar.powerups)
haveOilPow ← available(PowerUps.OIL, myCar.powerups)
haveEMP ← available(PowerUps.EMP, myCar.powerups)
haveBoost ← available(PowerUps.BOOST, myCar.powerups)
haveLizard ← available(PowerUps.LIZARD, myCar.powerups)
```

```
x_op ← opponent.position.block
y_op ← opponent.position.lane
```

```
// Damage logic
```

```
if (myCar.damage >= 5) then
    → FIX
```

```
else if (myCar.damage = 4 and myCar.speed > 6 ) then
    → FIX
```

```
else if (myCar.damage = 3 and myCar.speed > 8) then
    → FIX
```

```
endif
```

```
// Stuck for accelerate
```

```
Try
```

```
    if (myCar.state.equals(State.NOTHING)) then
        → ACCELERATE
```

```
    Endif
```

```
catch (Exception e)
    → ACCELERATE
```

```

// Boost logic
if (haveBoost and laneObstacleClear(laneStraight, x, 20)) then
    → BOOST

// avoidance logic bot
if (not clearStraight) then
    // periksa lane dimana mobil berada
    if (y = 1) then
        if (clearRight) then
            → TURN_RIGHT
        endif
    endif
    else if (y = 4) then
        if (clearLeft) then
            → TURN_LEFT
        endif
    endif
    else if (y = 2 or y = 3) then
        if (clearRight and powerupRight) then
            → TURN_RIGHT
        endif
        else if (clearLeft and powerupLeft) then
            → TURN_LEFT
        endif
        else if (clearRight) then
            → TURN_RIGHT
        endif
        else if (clearLeft) then
            → TURN_LEFT
        endif
    // jika kondisi tidak memungkinkan untuk berganti lane,
    dan
    // ada obstacle yang menghalangi
    if (haveLizard) then
        → USE_LIZARD
    Endif
endif

// Offensive logic
if ((haveTweet or haveOilPow) and x > x_op) then
    if (y = y_op) then
        if (haveTweet) then
            → new TweetCommand(y, x)

```

```

        Else then
            → OIL
        endif
    endif
Else then
    //tailing car behind
    if (y < y_op) then
        clearRight ← laneObstacleClear(laneRight, x,
        myCar.speed)
        if (clearRight) then
            → TURN_RIGHT
        endif
    Else then
        clearLeft ← laneObstacleClear(laneLeft, x,
        myCar.speed)
        if (clearLeft) then
            → TURN_LEFT
        endif
    Endif
Endif

if (haveEMP and x < x_op) then
    //if aimed correctly
    if (y = y_op) then
        → EMP;
    //else, correct aim
    Else then
        if (y = y_op-1) then
            clearRight ← laneObstacleClear(laneRight, x,
            myCar.speed)
            if (clearRight) then
                → TURN_RIGHT
            endif
        else if (y = y_op+1) then
            clearLeft ← laneObstacleClear(laneLeft, x,
            myCar.speed)
            if (clearLeft) then
                → TURN_LEFT
            endif
        Else then
            → ACCELERATE
        endif
    endif
endif

```

```

// catch power-up logic
if (y = 1) then
    clearRight ← laneObstacleClear(laneRight, x, myCar.speed)
    clearFront ← laneObstacleClear(laneStraight, x,
myCar.speed)
    powerUpFront ← laneHasBoost(laneStraight, x, myCar.speed)
    powerUpRight ← laneHasBoost(laneStraight, x, myCar.speed)

    if (clearFront and powerUpFront) then
        → ACCELERATE
    endif
    else if (clearRight and powerUpRight) then
        → TURN_RIGHT
    Endif
Endif

if (y = 2 or y = 3)
    If (powerUpFront and clearFront) then
        If (hasBoostFront) then
            → ACCELERATE
        endif
        else if (hasBoostLeft and clearLeft and not
hasBoostRight) then
            → TURN_LEFT
        endif
        else if (hasBoostRight and clearRight and not
hasBoostLeft) then
            → TURN_RIGHT
        endif
    endif
    else then
        if (clearRight and powerUpRight) then
            → TURN_RIGHT
        else if (clearLeft and powerUpLeft) then
            → TURN_LEFT
        endif
    endif
Endif

if (y = 4) then
    clearLeft ← laneObstacleClear(laneLeft, x, myCar.speed)
    powerUpLeft ← laneHasPowerUp(laneLeft, x, myCar.speed)
    clearFront ← laneObstacleClear(laneStraight, x, myCar.speed)
    powerUpFront ← laneHasPowerUp(laneStraight, x, myCar.speed)
    if (clearFront and powerUpFront) then

```



```

        → ACCELERATE
    else if (clearLeft and powerUpLeft) then
        → TURN_LEFT
    endif
endif

if (y=2 or y=3) then
    clearLeft ← laneObstacleClear(laneLeft, x, myCar.speed)
    clearRight ← laneObstacleClear(laneRight, x, myCar.speed)
    clearFront ← laneObstacleClear(laneStraight,x,myCar.speed)
    powerUpFront ← laneHasPowerUp(laneStraight,x,myCar.speed)
    powerUpLeft ← laneHasPowerUp(laneLeft, x, myCar.speed)
    powerUpRight ← laneHasPowerUp(laneRight, x, myCar.speed)
    hasBoostFront ← laneHasBoost(laneStraight,x,myCar.speed)
    hasBoostLeft ← laneHasBoost(laneLeft,x,myCar.speed)
    hasBoostRight ← laneHasBoost(laneRight,x,myCar.speed)

    if (powerUpFront and clearFront) then
        // prioritize boost powerup
        if (hasBoostFront) then
            → ACCELERATE
        else if (hasBoostLeft and clearLeft and not
hasBoostRight) then
            → TURN_LEFT
        else if (hasBoostRight and clearRight and not
hasBoostLeft) then
            → TURN_RIGHT
        Else then
            → ACCELERATE
        endif
    endif
    Else then
        if (clearRight and powerUpRight) then
            → TURN_RIGHT
        else if (clearLeft and powerUpLeft) then
            → TURN_LEFT
        endif
    endif
Endif

// DEFAULT LOGIC
→ ACCELERATE

```

4.2 Implementasi algoritma greedy pada program

Link GitHub : <https://github.com/farrel-a/entelect-bot-2020-stima-13>

Algoritma yang diimplementasikan berubah fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif dari tujuan permainan ini yaitu untuk mencapai garis *finish* terlebih dahulu. Himpunan kandidat berupa kumpulan *command* atau perintah yang dapat dijalankan kemudian dipilih menggunakan fungsi solusi untuk menghasilkan himpunan solusi. Fungsi seleksi memilih salah satu command dengan strategi greedy berdasarkan prioritas greedy. Fungsi kelayakan mengecek apakah command yang dipilih tidak melanggar aturan permainan dan dapat dimasukkan ke dalam himpunan solusi. Fungsi solusi mengecek apakah command yang sudah dipilih merupakan solusi yang tidak melanggar aturan dan sesuai dengan prioritas greedy. Fungsi objektif maksimasi/minimasi tergantung prioritas strategi greedy yang terpilih berdasarkan state mobil sekarang.

Algoritma greedy yang diimplementasikan dapat dideskripsikan sebagai berikut,

1. Melihat state pada mobil sekarang dari argumen yang diterima
2. Seleksi command berdasarkan salah satu urutan prioritas greedy yang dipilih berdasarkan state mobil:
 1. Fix pada kondisi damage tertentu (**greedy by damage**)
 2. Boost jika tersedia dan jalanan di depan bebas dari halangan (**greedy by speed**)
 3. Avoid obstacle jika di depan ada halangan dengan belok ke kiri atau ke kanan yang bebas dari obstacle, atau jika tidak bisa belok maka gunakan Lizard, jika tidak ada maka Accelerate dan paksa tabrak obstacle (**greedy by obstacle**)
 4. Pakai offensive power up jika ada dengan memakai tweet atau oil spill jika berada di depan dan pada lane yang sama dengan musuh atau memakai emp jika berada di belakang dan pada lane yang sama dengan musuh (**greedy by offensive power up**)
 5. Mengambil power up yang tersedia dengan prioritas mengambil boost (**greedy by power up pick up**)
 6. Jika tidak memenuhi kelima di atas, Accelerate (tambah kecepatan) (**greedy by speed**)
3. Memilih command berdasarkan salah satu dari enam prioritas greedy yang terpilih
4. Mengecek kelayakan command yang dipilih (tidak melanggar aturan) berdasarkan strategi greedy yang terpilih
5. Mengecek apakah command adalah solusi sesuai dengan tujuan strategi greedy untuk menang dan tidak melanggar aturan
6. Memilih command yang maksimasi/minimasi tergantung strategi greedy yang terpilih
7. Return command yang dipilih untuk dijalankan oleh game engine pada mobil bot
8. Menuliskan command yang dipilih sebagai input untuk command pada game engine.

4.3 Struktur data yang digunakan

Implementasi algoritma greedy terdapat pada file bot.java, file ini berupa class bot yang diinstansiasi pada file main.java. File main akan mendapatkan gamestate dari game engine dan menjalankan method run

pada bot dengan memberikan argumen berupa gamestate untuk dibaca pada objek bot. Bot kemudian melakukan algoritma greedy berdasarkan gamestate yang ada. Terdapat beberapa method fungsi pada bot yang dibuat untuk mengecek kondisi berdasarkan state, yaitu:

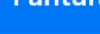
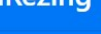





1. `available` : fungsi boolean yang mengecek apakah suatu power up yang dicari dimiliki pada mobil bot
2. `checkObjForward` : fungsi boolean yang mengecek apakah Terrain yang dicari ada di depan mobil sejauh jarak pandang parameter `distance`
3. `laneObstacleClear` : fungsi boolean yang mengecek apakah lane yang dipilih untuk dilihat tidak memiliki obstacle atau bebas dari Mud, Oil Spill, dan Wall
4. `laneHasPowerUp` : fungsi boolean yang mengecek apakah lane yang dipilih untuk dilihat memiliki power up apapun
5. `laneHasBoost` : fungsi boolean yang mengecek apakah lane yang dipilih untuk dilihat memiliki power up Boost

Method `run` pada bot akan mengembalikan sebuah objek yaitu `command` yang akan dijalankan. Objek `command` sudah di define untuk membentuk objek `command` tersebut jika di assign. `Command` yang diberikan sebenarnya merupakan instantiasi objek dari kelas `command` yang tersedia. Objek `command` yang di return akan diterima oleh `main.java`. `Main` kemudian akan menuliskan ke `system` dengan `system.out.println` berupa string `command` yang dijalankan. Game engine akan membaca string tersebut dan mengaplikasikannya pada mobil bot tersebut.

4.4 Analisis Solusi

Solusi yang dipaparkan dapat berfungsi dengan semestinya. Penggunaan method-method yang dibuat untuk bot dapat membantu mengimplementasikan jenis greedy yang dipilih. Contohnya pada strategi greedy by speed, penggunaan method `available` membuat bot mengutamakan penggunaan boost jika mobil memiliki boost. Greedy by speed juga mengutamakan akselerasi jika prioritas lain tidak terpenuhi. Selain itu, penggunaan method `checkObjForward` dan `laneObstacleClear` membantu greedy by obstacle mengecek apakah lane mobil sekarang bebas dari obstacle. Penggunaan `laneHasPowerUp` dan `laneHasBoost` membantu greedy by power up pick up untuk mengambil lane yang memiliki power up dan diutamakan power up adalah boost agar dapat membantu greedy by speed. Penggunaan method `available` juga membantu greedy by offensive yaitu ketika power up tersedia dan berada pada posisi yang diinginkan akan menggunakan power up tersebut untuk menyerang mobil musuh.

Akan tetapi solusi yang diimplementasikan masih ditemukan beberapa kekurangan. Greedy by obstacle yang diimplementasikan terkadang suka tidak akurat dan masih kurang pertimbangan aspek lain seperti pemilihan antara jalur dengan dua *obstacles* dan satu *obstacle*. Hal ini mungkin disebabkan pengecekan yang kurang jauh atau pengecekan yang kurang lincah dalam permainan yang berlaju cepat (*fast paced*). Selain itu penggunaan boost pada permainan seringkali tidak memberikan dampak. Sebagai contoh, ada satu kasus ketika menggunakan boost tapi kecepatan tidak bertambah hingga 15 sesuai dengan aturan permainan. Penggunaan *powerup* seperti *Lizard* terkadang tidak berfungsi dengan baik yang berakibat pengurangan pada kecepatan mobil dan penambahan poin kerusakan pada mobil.

A - PanturaRezing						
(selected) ▼						
						
Position 1 <small>[x: 758, y: 3]</small>	Speed 6	Lane 3	Distance 758	Boosts 5	Boosting YES	Powerups EMP,EMP,E MP,EMP,EM P,EMP,EMP, EMP,EMP,E MP,EMP,EM P

Gambar 4.4.1 Error pada penggunaan *power up boost*

BAB 5

Kesimpulan dan Saran

5.1 Kesimpulan

Program bot yang dibuat dengan pendekatan algoritma greedy berhasil memenangkan permainan terhadap reference bot. Penggunaan prioritas greedy berfungsi dengan semestinya. Akan tetapi masih ada kekurangan yaitu algoritma tidak cukup lincah untuk menentukan command yang terbaik untuk memenangkan pertandingan. Game engine ternyata ditemukan sering bermasalah yaitu ketika menggunakan boost tetapi tidak memberikan kecepatan tinggi yang didokumentasi pada aturan permainan. Penggunaan lizard juga seringkali tidak memberikan efek ketika power up tersebut dinyalakan. Penggunaan greedy dengan penekanan greedy by obstacle, yaitu menghindari halangan mungkin strategi yang lebih tepat karena dengan mengandalkan acceleration dan menghindari obstacle dapat memberikan kecepatan lebih tinggi yang lebih pasti.

5.2 Saran

Berdasarkan kesimpulan di atas dapat diambil saran:

1. Perlunya analisis strategi lebih jauh terhadap algoritma greedy yang didesain dan diimplementasikan
2. Perlunya perbaikan terhadap algoritma yang tidak cukup lincah dalam menentukan command

DAFTAR PUSTAKA

Munir, Rinaldi. 2021. Algoritma *Greedy* (Bagian 1)

[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).

Diakses pada 17 Februari 2022.

Entelect Challenge, 2020, Overdrive, <https://github.com/EntelectChallenge/2020-Overdrive>. Diakses pada

15 Februari 2022.