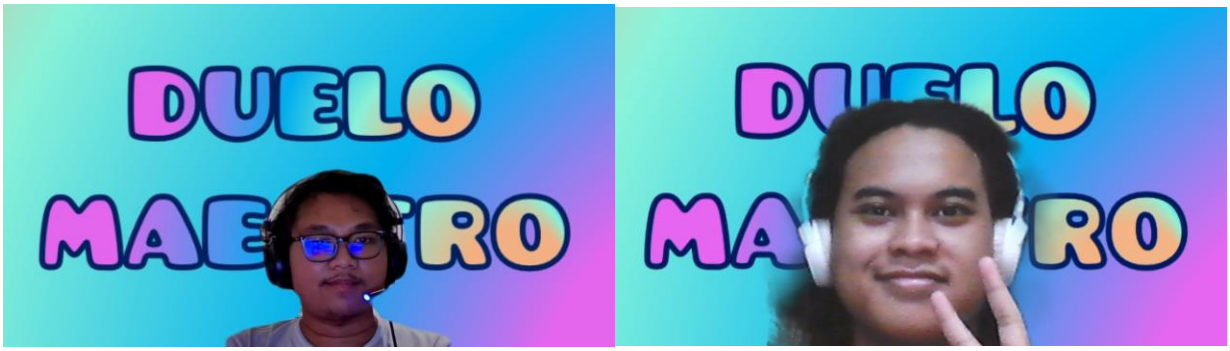
Laporan Tugas Besar 1
Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"
IF2211 Strategi Algoritma



Disusun Oleh:
Kelompok 28 - Duelo Maestro
Farnas Rozaan Iraqee (13520067)
Raden Rifqi Rahman (13520067)

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

2021/2022

DAFTAR ISI

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, digunakan sebuah game engine yang mengimplementasikan permainan Overdrive. Tugas kami adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan strategi greedy untuk memenangkan permainan.

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.

BAB II

LANDASAN TEORI

A. Dasar Teori Algoritma *Greedy*

Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. *Greedy* sendiri diambil dari Bahasa Inggris yang artinya rakus, tamak atau serakah .Prinsip algoritma *greedy* adalah: "*take what you can get now!*".

Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*). Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Algoritma ini memecahkan persoalan optimasi (*optimization problems*), yaitu persoalan yang menuntut pencarian solusi optimum. Persoalan optimasi sendiri ada dua macam: maksimasi (*maximization*) dan minimasi (*minimization*). Persoalan optimasi memiliki beberapa elemen yang penting untuk diperhatikan. Elemen-elemen tersebut adalah sebagai berikut:

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

Namun, pada sebagian persoalan, algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal, namun sub-optimal.

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

Contoh-contoh persoalan yang diselesaikan dengan algoritma *greedy*:

1. Persoalan penukaran uang (*coin exchange problem*)
2. Persoalan memilih aktivitas (*activity selection problem*)
3. Minimisasi waktu di dalam sistem
4. Persoalan knapsack (*knapsack problem*)

5. Penjadwalan Job dengan tenggat waktu (*job schedulling with deadlines*)
6. Pohon merentang minimum (*minimum spanning tree*)
7. Lintasan terpendek (*shortest path*)
8. Kode Huffman (*Huffman code*)
9. Pecahan Mesir (*Egyptian fraction*)

B. Cara Kerja Program

Pada permainan *Overdrive*, permainan dijalankan dengan mengeksekusi *game runner* permainan. *Game runner* dijalankan dengan mengeksekusi *file* arsip Java `game-runner-jar-with-dependencies.jar`. Selanjutnya, *game runner* akan membaca *file* konfigurasi *runner* yang memuat data direktori lokasi *game engine*, direktori bot kedua pemain, dsb. Setelah *file* konfigurasi dibaca, *runner* akan menjalankan permainan dengan mempertandingkan kedua bot pemain dalam *engine* permainan. Dalam permainan *engine* juga membaca *file* konfigurasi permainan yang memuat data konfigurasi permainan, seperti banyaknya lane, panjangnya jarak *finish line*, dsb.

Di dalam permainan, *engine* akan menulis sebuah *file* sementara '*temporary*' berisi keadaan permainan '*game state*' pada setiap ronde. Kemudian *engine* akan menjalankan program bot kedua pemain. Pada setiap ronde, *engine* berekspektasi pada kedua bot untuk mengembalikan sebuah perintah '*command*' kepada *engine* yang akan dieksekusi oleh *engine* untuk memperbarui keadaan permainan '*update game state*' untuk ronde selanjutnya.

Saat *engine* menjalankan program bot pemain, setiap bot dapat membaca *file game state* sementara yang ditulis oleh *engine*. Untuk bermain, bot harus mencoba memberikan perintah terbaik yang mungkin untuk mengalahkan lawan dan memenangkan permainan. Dalam program bot yang dibuat, bot bekerja dengan cara menguraikan '*parsing*' *game state* menjadi beberapa komponen. Bot kemudian menjalankan algoritma *greedy* yang akan menghasilkan perintah untuk dikembalikan kepada *engine*.

Algoritma *greedy* yang diimplementasi ke dalam bot bekerja dengan memetakan komponen-komponen permainan ke dalam elemen-elemen algoritma *greedy*. Algoritma kemudian mengevaluasi pilihan-pilihan aksi mulai dari aksi dengan prioritas tertinggi. Pilihan aksi dievaluasi untuk menentukan kelayakan aksi untuk dieksekusi. Evaluasi tersebut dilakukan berdasarkan komponen-komponen *game state* pada ronde saat itu. Ketika ditemukan aksi dengan prioritas tertinggi yang layak untuk dieksekusi, bot akan mengeksekusi aksi tersebut untuk mengembalikan perintah kepada *engine*.

BAB III

APLIKASI STRATEGI GREEDY

A. Pemetaan Komponen Permainan *Overdrive* ke Elemen Algoritma *Greedy*

Pada dasarnya, permainan *Overdrive* dilakukan dengan mengirimkan perintah '*command*' dari bot kepada *game engine* pada setiap ronde. Dengan demikian, himpunan perintah-perintah atau *command* yang tersedia adalah himpunan kandidat dari algoritma *greedy*. Selanjutnya, tujuan dari permainan tersebut adalah mencapai garis *finish* pada jarak 1500 secepat mungkin. Akibatnya, himpunan solusi dari algoritma *greedy* pada permainan ini adalah urutan perintah yang dieksekusi atau diberikan oleh bot untuk mencapai garis *finish*. Demikian pula dengan fungsi solusi. Fungsi solusi dari algoritma *greedy* ini menentukan apakah perintah-perintah yang diberikan telah menuntun mobil pemain mencapai garis *finish*. Fungsi seleksi pada algoritma *greedy* ini memilih perintah yang paling masuk akal '*reasonable*' untuk dieksekusi untuk mencapai jarak 1500 secepat-cepatnya. Fungsi kelayakan menentukan kelayakan perintah, yaitu apakah perintah akan merugikan pemain jika dieksekusi atau bahkan tidak dapat dieksekusi sama sekali, atau tidak. Fungsi objektif pada algoritma *greedy* ini meminimumkan banyaknya perintah yang dieksekusi untuk mencapai garis *finish*.

Secara umum, elemen algoritma *greedy* pada permainan *Overdrive* adalah sebagai berikut.

1. Himpunan kandidat adalah himpunan perintah yang tersedia.
2. Himpunan solusi adalah urutan perintah yang dieksekusi untuk mencapai garis *finish*.
3. Fungsi solusi menentukan apakah pemain telah mencapai garis *finish*.
4. Fungsi seleksi memilih perintah yang paling masuk akal '*reasonable*' untuk dieksekusi.
5. Fungsi kelayakan menentukan kelayakan perintah untuk dieksekusi.
6. Fungsi objektif meminimumkan banyaknya perintah yang dieksekusi.

Akan tetapi, terdapat informasi keadaan (*state*) permainan yang dapat diketahui oleh bot pada setiap rondenya. Proses seleksi yang dilakukan oleh fungsi seleksi akan sangat kompleks dengan mempertimbangkan *state* yang ada. Oleh karena itu, pada bot yang diimplementasi, dilakukan abstraksi terhadap elemen-elemen algoritma *greedy* di atas.

Pada implementasinya, bot tidak melakukan seleksi terhadap perintah untuk mendapatkan perintah yang masuk akal '*reasonable*', melainkan bot melakukan seleksi terhadap aksi yang mungkin dilakukan oleh strategi *greedy* yang digunakan oleh bot. Maksudnya, pada implementasi didefinisikan kumpulan aksi yang mungkin dilakukan oleh bot. Sebuah aksi dapat dipetakan ke dalam satu atau lebih perintah, tergantung kelayakannya pada keadaan '*state*' permainan saat itu. Akibatnya, fungsi seleksi lebih mudah dieksekusi dengan menyeleksi aksi—yang jauh lebih banyak dan beragam dari perintah—dibandingkan dengan menyeleksi perintah secara langsung. Dengan demikian, berikut adalah elemen algoritma *greedy* pada permainan *Overdrive* setelah dilakukan abstraksi.

1. Himpunan kandidat adalah himpunan *aksi* yang *didefinisikan*.
2. Himpunan solusi adalah urutan *aksi* yang dieksekusi untuk mencapai garis *finish*.
3. Fungsi solusi menentukan apakah pemain telah mencapai garis *finish*.
4. Fungsi seleksi memilih *aksi* dengan prioritas tertinggi yang layak dieksekusi.
5. Fungsi kelayakan menentukan kelayakan *aksi* untuk dieksekusi.
6. Fungsi objektif meminimumkan banyaknya *aksi* yang dieksekusi.

B. Alternatif Solusi *Greedy*

Terdapat sangat banyak alternatif solusi *greedy* yang dapat diimplementasi pada bot. Dari banyaknya alternatif solusi yang mungkin, 10 alternatif solusi diimplementasi dan diuji pada tugas besar ini.

1. Strategi *greedy Safe*

Strategi *Safe* mengambil dan menggunakan *power up* dengan bijak. Strategi ini lebih mengutamakan menggunakan *power up* dibandingkan dengan mengambilnya, dan lebih mengutamakan mengambil *power up* dibandingkan dengan akselerasi atau pindah jalur secara acak '*random*' tanpa mencari *power up* di sekitar.

2. Strategi *greedy Berserk*

Strategi *Berserk* mengumpulkan *power up* sebanyak-banyaknya dan menggunakannya secara berulang-ulang pada seperlima terakhir pertandingan—yaitu pada jarak 1200 atau lebih—yang disebut sebagai aksi *berserk.*

3. Strategi *greedy Rage*

Strategi *Rage* merusak mobil pemain lawan sesering mungkin. Strategi ini lebih mengutamakan mengambil *EMP* dan *TWEET* dibandingkan dengan yang lain. Strategi ini juga menggunakan *power up* lain secara bijak.

4. Strategi *greedy Boost Maniac*

Strategi *Boost Maniac* mirip seperti strategi *Safe*, tetapi menggunakan dan mengambil *BOOST* sesering mungkin.

5. Strategi *greedy Power Up Maniac*

Strategi *Power Up Maniac* menggunakan dan mengambil *power up* sesering mungkin.

6. Strategi *greedy Destroyer*

Strategi *Destroyer* mengumpulkan *EMP* sebanyak mungkin kemudian menggunakannya berturut-turut untuk menghancurkan mobil pemain lawan dan membatasinya pindah jalur.

7. Strategi *greedy Skipper*

Strategi *Skipper* mirip seperti strategi *Safe*, tetapi lebih suka menggunakan *LIZARD* dibandingkan dengan *power up* yang lain.

8. Strategi *greedy Ultra Berserk*

Strategi *Ultra Berserk* mirip seperti strategi *Berserk*, tetapi lebih sering mengambil *power up* dan melakukan aksi *berserk* pada setengah terakhir pertandingan dibandingkan dengan seperlima terakhir.

9. **Strategi *greedy Haste Berserk***

   Strategi *Haste Berserk* mirip seperti strategi *Berserk*, tetapi menggunakan *BOOST* saat mendapatkannya dari pada menunggu seperlima terakhir pertandingan.

10. **Strategi *greedy Haste Skipper***

    Strategi *Haste Skipper* mirip seperti strategi *Skipper*, tetapi lebih memilih menggunakan *BOOST* daripada *LIZARD*.

C. Analisis Efisiensi Alternatif Solusi *Greedy*

Pada dasarnya, setiap alternatif solusi strategi *greedy* di atas tersusun atas banyak aksi yang sama dengan hanya sedikit perbedaan. Beberapa aksi yang didefinisikan dikelompokkan ke dalam beberapa kategori, yaitu aksi melaju '*go action*', aksi memperbaiki '*fix action*', aksi mengambil '*pick up action*', aksi menggunakan '*use action*', dan aksi final '*final action*'. Akan tetapi, untuk strategi *Berserk, Rage, Ultra Berserk, Haste Berserk,* dan *Destroyer*; terdapat perilaku atau aksi-aksi yang tidak dapat dikategorikan ke dalam kategori-kategori aksi yang telah disebutkan. Dengan demikian, terdapat kategori aksi tambahan yaitu aksi spesial '*special action*' yang memuat aksi-aksi yang tidak dapat dikelompokkan ke dalam kategori lainnya khusus untuk strategi-strategi tersebut.

Berdasarkan kategori aksi yang dilakukan oleh setiap strategi, strategi *Berserk, Rage, Ultra Berserk, Haste Berserk,* dan *Destroyer* memiliki aksi tambahan. Akibatnya, alternatif solusi-solusi tersebut dapat dipandang lebih tidak efisien dibandingkan alternatif solusi lainnya. Di sisi lain, strategi *Safe, Boost Maniac, Power Up Maniac, Skipper,* dan *Haste Skipper* dapat dipandang memiliki efisiensi yang relatif sama, tetapi lebih efisien dari 5 strategi lainnya. Oleh karena itu, strategi yang paling efisien dari alternatif strategi yang diusulkan adalah strategi *Safe, Boost Maniac, Power Up Maniac, Skipper,* dan *Haste Skipper*.

D. Analisis Efektivitas Alternatif Solusi *Greedy*

Efektivitas alternatif-alternatif solusi *greedy* yang diusulkan bervariasi. Hal ini dikarenakan efektivitas setiap strategi sangat bergantung pada keberuntungan '*luck*' dari peta permainan. Sebagai contoh, strategi *Power Up Maniac, Berserk, Haste Berserk*, dan *Ultra Berserk* hanya akan efektif jika bot dapat menemukan dan mengambil banyak *power up* yang tersebar dalam peta. Lebih spesifik lagi, strategi *Rage* hanya akan efektif jika bot dapat mengambil *EMP* dan *TWEET* sebanyak mungkin; strategi *Boost Maniac* hanya akan efektif jika bot dapat mengambil *BOOST* sebanyak mungkin; dan streategi *Destroyer* hanya akan efektif jika bot dapat mengambil *EMP* sebanyak mungkin. Meskipun demikian, ada beberapa strategi yang cukup "seimbang", yaitu tidak terlalu bergantung pada tata letak peta permainan.

Strategi-strategi tersebut adalah strategi *Safe*, *Skipper*, dan *Haste Skipper*. Ketiga strategi memuat aksi-aksi yang sama dan hanya berbeda pada prioritas *power up* yang digunakan. Dengan demikian, ketiga strategi relatif cukup efektif pada berbagai tata letak peta permainan.

E. Solusi *Greedy* yang Dipilih

Berdasarkan analisis efisiensi dan analisis efektivitas, terdapat 3 strategi yang cukup unggul dari strategi lainnya. Strategi-strategi tersebut yaitu strategi *Safe*, *Skipper*, dan *Haste Skipper*. Dari ketiga strategi ini, strategi *Skipper* dipilih sebagai strategi utama pada bot. Alasan pemilihan strategi *Skipper* dibandingkan 2 strategi lainnya dibahas lebih lanjut pada Bab IV.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algoritma *Greedy*

Berikut adalah implementasi algoritma *greedy* dengan strategi *Skipper* dalam bentuk *pseudocode*.

```
procedure Skipper()

Declare
  type Action : <...> /* enumerated type */
  type Command: <...> /* defined type */
  actions: array of Action
  action : Action
  command: Command

Algorithm
actions <- getActions()
for each action in actions
  if isFeasible(action) then
    command <- execute action
    break
output(command)
```

Gambar 1. *Pseudocode* program utama bot

```
function getActions() -> array of Action
/* Get Skipper strategy actions, sorted by priority */

Declare

Algorithm
  -> [
       FinalAction_Accelerate,
       FinalAction_Normal,
       FinalAction_TurnLeft,
       FinalAction_TurnRight,
       FixAction_Urgent,
       UseAction_Lizard,
       UseAction_EMP,
       UseAction_CyberTruck,
```

```
      UseAction_Boost,
      UseAction_Oil,
      FixAction_SemiUrgent,
      PickUpAction_Lizard,
      PickUpAction_EMP,
      PickUpAction_CyberTruck,
      PickUpAction_Boost,
      PickUpAction_Oil,
      GoAction_Accelerate,
      GoAction_Normal,
      GoAction_TurnLeft,
      GoAction_TurnRight,
      GoAction_Decelerate,
      FixAction_Normal
    ]
```

Gambar 2. *Pseudocode* daftar aksi dalam strategi *Skipper*

```
function isFeasible(action: Action) -> boolean

Declare

Algorithm
if (action = FinalAction_Accelerate) then
  -> player will reach finish line if accelerating

if (action = FinalAction_Normal) then
  -> player will reach finish line if not doing anything

if (action = FinalAction_TurnLeft) then
  -> player will reach finish line if turning left

if (action = FinalAction_TurnRight) then
  -> player will reach finish line if turning right

if (action = FixAction_Urgent) then
  -> player's car has at least 5 damage

if (action = UseAction_Lizard) then
  -> player has LIZARD power up
     and there are obstacles ahead of player
```

```
     and player will not land on an obstacle after using LIZARD

if (action = UseAction_EMP) then
  -> player has EMP power up
     and player will not hit hard obstacle
     and player is behind opponent
     and opponent is within EMP range

if (action = UseAction_CyberTruck) then
  -> player has TWEET power up
     and player will not hit hard obstacle

if (action = UseAction_Boost) then
  -> player has BOOST power up
     and player will not hit hard obstacle when boosted
     and player is not currently boosting

if (action = UseAction_Oil) then
  -> player has OIL power up
     and player will not hit hard obstacle when boosted
     and player is ahead of opponent
     and opponent is far enough to not be able to bypass the oil

if (action = FixAction_SemiUrgent) then
  -> player's car has at least 3 damage

if (action = PickUpAction_Lizard) then
  -> player sees LIZARD reachable power up that is not blocked by
any obstacle

if (action = PickUpAction_EMP) then
  -> player sees EMP reachable power up that is not blocked by any
obstacle

if (action = PickUpAction_CyberTruck) then
  -> player sees TWEET reachable power up that is not blocked by
     any obstacle

if (action = PickUpAction_Boost) then
  -> player sees BOOST reachable power up that is not blocked by
     any obstacle
```

```
if (action = PickUpAction_Oil) then
  -> player sees OIL reachable power up that is not blocked by any
     obstacle

if (action = GoAction_Accelerate) then
  if (player's car speed = 0 and player's car can gain more speed)
    then
    -> true
  if (player will bump hard obstacles regardless of the lane
      they're at and player's car can gain more speed) then
    -> true
  if (player will bump hard obstacles when accelerated but not
      when not accelerated) then
    -> false
  if (player will bump any obstacles regardless of the lane
      they're at and player's car can gain more speed) then
    -> true
  if (player will bump any obstacles when accelerated but not
      when not accelerated) then
    -> false
  -> player will not bump any obstacles

if (action = GoAction_Normal) then
  if (player will bump hard obstacles regardless of the lane
      they're at) then
    -> true
  if (player will bump any obstacles regardless of the lane
      they're at) then
    -> true
  -> player will not bump any obstacles

if (action = GoAction_TurnLeft) then
  if (player cannot turn left) then
    -> false
  if (player will bump any obstacles regardless of the lane
      they're at) then
    -> player will not bump hard obstacles when turning left
  -> player will not bump any obstacles
     and there is no hard obstacle in player's car left side
     and player is not hit by
```

```
    EMP

if (action = GoAction_TurnRight) then
  if (player cannot turn right) then
    -> false
  if (player will bump any obstacles regardless of the lane
      they're at) then
    -> player will not bump hard obstacles when turning right
  -> player will not bump any obstacles
     and there is no hard obstacle in player's car right side
     and player is not hit by EMP

if (action = GoAction_Decelerate) then
  -> player will not bump any obstacles when decelerated

if (action = FixAction_Normal) then
  -> true
```

Gambar 3. *Pseudocode* fungsi kelayakan aksi

B. Struktur Data

Dalam program bot yang dibuat, terdapat beberapa struktur data yang digunakan, termasuk struktur data dalam permainan *Overdrive* dan struktur data tambahan, sebagai berikut.

1. Direction

Class Direction mengenumerasi kemungkinan arah belokan yang dapat dilakukan oleh *bot* mobil, yaitu arah kiri dan kanan.

2. PowerUp

Class PowerUp mengenumerasi jenis-jenis *power up* yang tersedia dan dapat diambil oleh *bot* mobil, yaitu *boost, oil, tweet, lizard,* dan *EMP*.

3. State

Class State mengenumerasi kemungkinan kondisi *bot* mobil pada saat tertentu, yaitu sebagai berikut:

a. *ACCELERATING* : mobil sedang menambah kecepatannya
b. *DECELERATING* : kecepatan mobil terkurangi
c. *FINISHED* : mobil sudah mencapai garis *finish*
d. *HIT_CYBER_TRUCK* : mobil menabrak *cyber truck*
e. *HIT_EMP* : mobil terkena tembakan *EMP*
f. *HIT_MUD* : mobil melewati *mud block*
g. *HIT_OIL* : mobil menginjak *oil*
h. *HIT_WALL* : mobil menabrak *wall*

i. *NOTHING* : tidak ada yang terjadi pada mobil
j. *PICKED_UP_POWERUP* : mobil mengambil *power up*
k. *READY* : mobil siap untuk melaju
l. *TURNING_LEFT* : mobil belok kiri
m. *TURNING_RIGHT* : mobil belok kanan
n. *USED_BOOST* : mobil menggunakan *power up boost*
o. *USED_EMP* : mobil menggunakan *power up EMP*
p. *USED_LIZARD* : mobil menggunakan *power up lizard*
q. *USED_TWEET* : mobil menggunakan *power up tweet*
r. *USED_OIL* : mobil menggunakan *power up oil*

4. **Terrain**

Class Terrain mengenumerasi kemungkinan kondisi *block*, yaitu sebagai berikut:

a. *BOOST* : terdapat *power up boost* pada *block*
b. *EMP* : terdapat *power up EMP* pada *block*
c. *EMPTY* : *block* kosong
d. *FINISH* : *finish line*
e. *LIZARD* : terdapat *power up lizard* pada *block*
f. *MUD* : terdapat *mud* pada *block*
g. *OIL_POWER* : terdapat *power up oil* pada *block*
h. *OIL_SPILL* : terdapat *oil spill* pada *block*
i. *TWEET* : terdapat *power up tweet* pada *block*
j. *WALL* : terdapat *wall* pada *block*

5. **Position**

Class Position ini mencatat posisi dari mobil, yaitu terletak di *lane* dan *block* sebelah mana.

6. **Block**

Class Block ini mencatat posisi, kondisi *block* (*terrain*), ID *player* yang menempati, dan apakah block tersebut terdapat *cyber truck*.

7. **Car**

Class Car ini mencatat kondisi mobil pada saat tertentu. Atribut yang dicatat adalah ID, posisi, kecepatan, *state* (sesuai dengan *class* State), *state* yang sudah dialami mobil di *round* tersebut, *damage* yang diterima, *powerups* yang dimiliki, apakah mobil dalam keadaan *boosting* (menggunakan *item boost*), dan jumlah *item boost* yang telah digunakan mobil.

8. **GameState**

Class GameState ini mencatat kondisi dari *game*, yaitu sekarang ronde ke berapa, maksimal *game* sampai ronde berapa, informasi mobil *player* dan musuh, serta *layout world map*.

9. **Command**

Command merupakan *interface* yang dipakai ketika ingin menuliskan *command* tertentu. Terdapat beberapa *class* yang memanfaatkan *interface* Command ini. *Class* yang dimaksud adalah sebagai berikut:

a. *AccelerateCommand* menuliskan "ACCELERATE"
b. *BoostCommand* menuliskan "USE_BOOST"
c. *DecelerateCommand* menuliskan "DECELERATE"
d. *DoNothingCommand* menuliskan "NOTHING"
e. *EmpCommand* menuliskan "USE_EMP"
f. *FixCommand* menuliskan "FIX"
g. *LizardCommand* menuliskan "USE_LIZARD"
h. *OilCommand* menuliskan "USE_OIL"
i. *TurnCommand* menuliskan "TURN_LEFT" atau "TURN_RIGHT"
j. *TweetCommand* menuliskan "USE_TWEET <*lane*> <*block*>"

10. LocalMap

Class LocalMap merepresentasikan sebuah map dari *game* pada *state* tertentu.

11. ObstacleType

Class ObstacleType mengenumerasikan jenis-jenis rintangan berdasarkan *damage* yang dihasilkannya, yaitu *soft, hard,* atau keduanya.

12. DetectionParams

Class DetectionParams berisi semua parameter yang dibutuhkan untuk melakukan iterasi terhadap *blocks* yang tersedia di LocalMap. Hal itu dilakukan sebagai sebuah pertimbangan untuk melakukan GoAction apa pada *state* tersebut.

13. PowerUpSearchResult

Class PowerUpSearchResult menyimpan informasi ketersediaan *power up* setelah dilakukan pencarian dan *action* yang akan dilakukan setelah dilakukan pencarian tersebut.

14. RelativePosition

Class RelativePosition merepresentasikan posisi relatif dari sebuah mobil terhadap mobil lainnya berdasarkan *lane* mobil tersebut. Ada enam kemungkinan posisi relatif, yaitu sebagai berikut:

a. *ACROSS* : posisi kedua mobil berseberangan
b. *LEFT* : posisi mobil lain berada di sebelah kiri
c. *RIGHT* : posisi mobil lain berada di sebelah kanan
d. *FAR_LEFT* : posisi mobil lain berada jauh di sebelah kiri
e. *FAR_RIGHT* : posisi mobil lain berada jauh di sebelah kanan
f. *PAR* : posisi kedua mobil berada di *lane* yang sama

15. Action

Action merupakan *interface* yang dipakai ketika ingin mengeksekusi aksi tertentu. Terdapat beberapa *class* yang memanfaatkan *interface* Action ini. *Class* yang dimaksud adalah sebagai berikut:

a. *FinalAction* terdiri dari aksi-aksi yang *feasible* jika aksi tersebut merupakan aksi terakhir sebelum mencapai garis *finish*
b. *FixAction* terdiri dari aksi-aksi untuk memperbaiki mobil bergantung pada damage yang diterima mobil
c. *GoAction* terdiri dari aksi-aksi dasar dalam *game*
d. *PickUpAction* terdiri dari aksi-aksi untuk mengambil *power up* tertentu
e. *SpecialAction* terdiri dari aksi-aksi khusus yang belum didefinisikan di kategori *action* yang lain dan mempunyai syarat kelayakan yang lebih spesifik sebelum aksi dieksekusi
f. *UseAction* terdiri dari aksi-aksi untuk menggunakan *power up* yang dimiliki mobil

16. **Strategy**

Class Strategy berisi beberapa strategi *greedy* beserta *action* yang dieksekusi masing-masing strategi tersebut (keputusan terakhir menggunakan strategi *Skipper*).

17. **Detector**

Detector terbagi menjadi tiga *class*, yaitu sebagai berikut:

a. *ObstacleDetector* mendeteksi rintangan yang terletak di sekitar mobil
b. *PositionDetector* mendeteksi dan menghitung proses lojik terkait kedua mobil
c. *PowerUpDetector* mendeteksi *power up* yang terletak di sekitar mobil

18. **Checker**

Checker berfungsi untuk mengecek kelayakan suatu aksi. Checker tersebut dibagi menjadi enam kategori. Berikut adalah keenam kategori tersebut:

a. *FinalActionChecker* mengecek kelayakan *FinalAction.*
b. *FixActionChecker* mengecek kelayakan *FixAction.*
c. *GoActionChecker* mengecek kelayakan *GoAction.*
d. *PickUpActionChecker* mengecek kelayakan *PickUpAction.*
e. *SpecialActionChecker* mengecek kelayakan *SpecialAction.*
f. *UseActionChecker* mengecek kelayakan *UseAction.*

19. **Finder**

Class Finder berfungsi untuk mencari *terrai* atau *block* tertentu di map. Kemudian terdapat juga FinishLineFinder yang berfungsi mencari *finish line* dan ObstacleFinder yang berfungsi untuk mencari rintangan.

20. **Bot**

Class Bot berfungsi untuk membuat bot baru dari *game state* tertentu. Pada *class* ini terdapat *method* useStrategy yang berfungsi untuk menentukan strategi apa yang akan digunakan dalam *game* (dipilih strategi *Skipper*) dan run yang berfungsi untuk

menganalisis *game state* dan menghitung aksi yang terbaik dan layak untuk dilakukan di *state* tertentu berdasarkan *list* Action dari sebuah strategi.

C. Analisis Desain Solusi Algoritma *Greedy*

Performa solusi algoritma *greedy* yang dipilih dapat dianalisis dengan melakukan pengujian. Pada tugas besar ini, pengujian dilakukan dengan melakukan pertandingan antara bot referensi dengan setiap strategi yang diusulkan. Selanjutnya beberapa alternatif strategi terbaik diambil untuk ditandingkan dengan satu sama lain.

Pada setiap pertandingan dalam pengujian, diambil beberapa data hasil pertandingan sebagai parameter pengujian. Parameter-parameter pengujian tersebut antara lain total kemenangan '*wins*' (W) baik sebagai pemain A '*wins as A*' (WA) ataupun sebagai pemain B '*wins as B*' (WB), *margin* kemenangan rata-rata '*average winning margin*' (AWM) dan tertinggi '*highest winning margin*' (HWM), skor rata-rata '*average score*' (AS) dan tertinggi '*highscore*' (HS), banyak ronde kemenangan tercepat '*fastest winning rounds*' (FWR) dan ronde rata-rata pertandingan '*average rounds*' (AR). Untuk suatu pertandingan, performa sebuah alternatif solusi semakin baik jika solusi tersebut semakin banyak menang (W, WA, dan WB tinggi), menang dengan *margin* yang semakin tinggi (HWM dan AWM tinggi), memperoleh skor semakin besar (AS dan HS tinggi), dan semakin cepat dalam memenangkan pertandingan (FWR dan AR rendah).

Untuk setiap pengujian, pertandingan dilakukan sebanyak 20 kali, yaitu 10 pertandingan sebagai pemain A dan 10 pertandingan sebagai pemain B untuk masing-masing bot. Berikut adalah data hasil pengujian alternatif strategi yang diusulkan terhadap bot referensi.

Tabel 1. Data hasil pengujian terhadap bot referensi

Strategi	*Safe*	*Berserk*	*Rage*	*Boost Maniac*	*Power Up Maniac*
W	20	20	20	20	20
WA	10	10	10	10	10
WB	10	10	10	10	10
AWM	+721.65	+582.05	+669.55	+703	+650.1
HWM	+822	+732	+759	+769	+735
AS	280.95	174.45	199.8	249.95	171.15
HS	393	250	284	387	253

FWR	205	227	202	200	210
AR	213.1	251.45	220.95	218.9	231.15

Strategi	*Destroyer*	*Skipper*	*Ultra Berserk*	*Haste*
W	20	20	20	
WA	10	10	10	
WB	10	10	10	
AWM	+694.95	+732.1	+485.4	+5
HWM	+777	+856	+687	+
AS	261.65	300.3	376.1	1
HS	358	392	516	
FWR	203	191	256	
AR	220.35	210.15	286.25	2

Berdasarkan tabel data hasil pengujian di atas, sesuai dengan analisis efisiensi dan efektivitas, terlihat bahwa strategi *Safe*, *Skipper*, dan *Haste Skipper* adalah strategi terbaik di antara strategi yang diusulkan. Selanjutnya, ketiga usulan strategi ini ditandingkan satu sama lain. Berikut adalah data hasil pengujian ketiga strategi.

Tabel 2. Data hasil pengujian startegi *Safe* vs *Skipper* vs *Haste Skipper*

Strategi	*Safe*	*Skipper*	*Haste Skipper*
W	20	21	19
WA	10	11	8
WB	10	10	11
AWM	-4.875	+1.4	+3.475
HWM	+225	+179	+190

AS	246.7	242.775	250.125
HS	362	378	420
FWR	191	198	195
AR	215.15	215.625	213.225

Berdasarkan hasil pengujian terhadap 3 alternatif solusi terbaik, terlihat bahwa strategi *Skipper* memperoleh kemenangan terbanyak di antara 2 strategi lainnya. Hal ini mendasari pemilihan strategi ini seperti telah disebutkan sebelumnya pada Bab III.

Setelah dilakukan pengujian antar usulan strategi dengan bot referensi ataupun dengan satu sama lain, dilakukan analisis juga terhadap pertandingan yang telah selesai. Berdasarkan hasil pengamatan dan analisis terhadap riwayat permainan, ditemukan bahwa usulan strategi *greedy* belum optimal dalam memenangkan permainan. Karena kemiripan antara beberapa usulan strategi, hasil yang tidak optimal karena alasan serupa juga ditemukan pada hampir semua usulan strategi. Berdasarkan hasil analisis, usulan strategi memiliki beberapa kecenderungan yang menyebabkan hasil pertandingan tidak optimal, yaitu sebagai berikut.

1. Usulan strategi cenderung mengutamakan penggunaan *power up* daripada meningkatkan kecepatan. Dengan demikian, ketika bot memiliki banyak *power up* padahal tidak perlu menggunakannya—misalnya pemain telah unggul telak dari lawan dalam segi jarak—bot akan tetap menggunakan *power up* yang dapat digunakan. Akibatnya, kecepatan pemain menjadi tidak maksimal dan kemenangan diperoleh dengan lebih lambat.
2. Usulan strategi belum mempertimbangkan ketertinggalan. Karena strategi cenderung menggunakan *power up* daripada meningkatkan kecepatan, strategi yang diusulkan akan mudah kalah jika telah tertinggal oleh lawan dengan jarak yang cukup jauh. Strategi yang diusulkan tidak membedakan aksi yang dilakukan ketika memimpin dan tertinggal. Akibatnya, pemain tidak dapat mengejar ketertinggalan dari lawan ketika telah tertinggal cukup jauh.
3. Usulan strategi cenderung mengutamakan ketahanan mobil daripada kecepatan. Setiap usulan strategi memiliki kecenderungan untuk mempertahankan mobil agar tidak rusak dibandingkan dengan mempercepat mobil. Akibatnya, usulan strategi jarang memberikan kecepatan yang maksimal untuk mobil pemain sehingga hasil permainan tidak optimal.

D. Alamat *Repository Source Code* Program

Source code program bot yang telah diimplementasi dapat diakses dalam *repository* GitHub melalui tautan ini.

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan bot yang telah diprogram dan diimplementasi serta hasil pengujian yang telah dianalisis, maka dapat ditarik kesimpulan sebagai berikut.

1. Bot permainan *Overdrive* dapat dibuat menggunakan strategi algoritma *greedy* dengan terlebih dahulu memetakan komponen-komponen permainan ke dalam elemen-elemen algoritma *greedy*.
2. Usulan strategi *greedy* bervariasi dalam hal efisiensi, efektivitas, dan performa dengan usulan strategi *Skipper* memberikan performa yang terbaik.
3. Usulan strategi *greedy* yang diberikan masih belum optimal karena cenderung mengutamakan penggunaan *power up*, tidak membedakan prioritas aksi yang dieksekusi saat memimpin atau tertinggal, dan lebih mengutamakan ketahanan mobil daripada kecepatan.

B. Saran

Setelah perancangan, pengimplementasian, dan pengujian bot dengan strategi algoritma *greedy* ini selesai, untuk pembuatan bot yang selanjutnya disarankan sebagai berikut.

1. Bot yang dibuat sebaiknya efisien, efektif, dan *performant*, baik dalam hal aksi yang dieksekusi maupun hasil yang diberikan.
2. Bot yang dibuat seharusnya mengutamakan kecepatan dan dapat membedakan aksi yang dieksekusi ketika memimpin dan tertinggal, dengan tujuan agar diperoleh sesedikit mungkin aksi untuk mencapai hasil yang optimal.

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf