Tugas Besar 1

IF2211 Strategi Algoritma

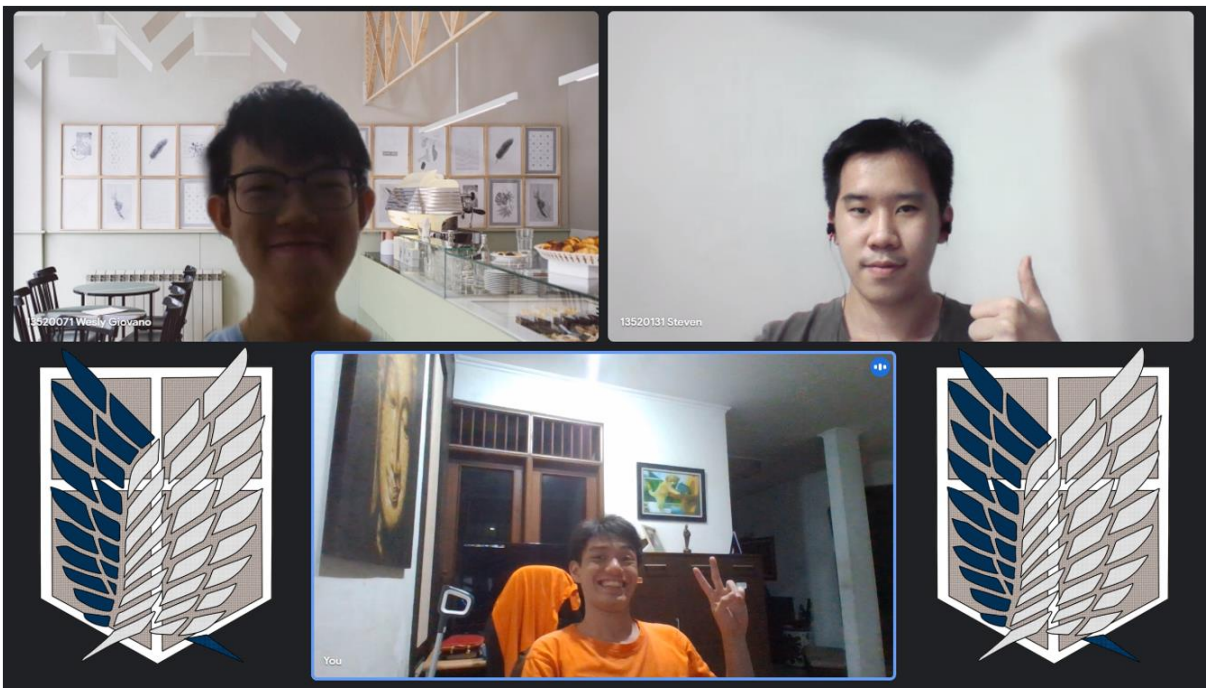
Aplikasi Greedy Algorithm dalam Permainan Overdrive

Semester II Tahun 2021/2022

Disusun oleh:

Kelompok Sasageyo

Wesly Giovano	13520071
Steven	13520131
Aldwin Hardi Swastia	13520167



Institut Teknologi Bandung

Sekolah Teknik Elektro dan Informatika

2022

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

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.

Gambar 1. Ilustrasi permainan *Overdrive*

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah *game engine* yang mengimplementasikan permainan *Overdrive*. *Game engine* dapat diperoleh pada laman berikut: https://github.com/EntelectChallenge/2020-Overdrive.

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

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

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

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

b. *Boost*, dapat mempercepat kecepatan mobil anda secara drastis.

c. *Lizard*, berguna untuk menghindari *lizard* yang mengganggu jalan mobil anda.

d. *Tweet*, dapat menjatuhkan truk di *block* spesifik yang anda inginkan.

e. *EMP*, dapat menembakkan *EMP* ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 *lane* yang sama) akan terus berada di *lane* yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.

3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 *block* untuk setiap *round*. *Game state* akan memberikan jarak pandang hingga 20 *block* di depan dan 5 *block* di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.

4. Terdapat *command* yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan *powerups*. Pada setiap *round*, masing-masing pemain dapat memberikan satu buah *command* untuk mobil mereka. Berikut jenis-jenis *command* yang ada pada permainan:

   a. NOTHING
   b. ACCELERATE
   c. DECELERATE
   d. TURN_LEFT
   e. TURN_RIGHT
   f. USE_BOOST
   g. USE_OIL
   h. USE_LIZARD
   i. USE_TWEET <lane> <block>
   j. USE_EMP
   k. FIX

5. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.

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

2.1 Dasar Teori

Algoritma greedy merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi (maksimasi atau minimasi). Dari pengertiannya sendiri, greedy berarti tamak, rakus ataupun serakah. Oleh karena itu, algoritma greedy memiliki prinsip "Take what you can get now!" dengan harapan setiap langkah yang merupakan optimum lokal dapat berakhir dengan hasil optimum global.

Terdapat enam buah elemen algoritma greedy, yaitu

1. Himpunan kandidat, berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, berisi kandidat yang sudah dipilih.
3. Fungsi solusi, untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi, untuk memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy bersifat heuristik.
5. Fungsi kelayakan, untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi objektif, yaitu fungsi yang digunakan untuk melakukan optimasi terhadap strategi algoritma, umumnya berupa memaksimumkan atau meminimumkan sesuatu.

Karena algoritma greedy hanya mencari langkah terbaik pada setiap langkahnya, maka algoritma greedy memiliki kompleksitas yang jauh lebih rendah daripada algoritma brute force.

2.2 Cara Kerja Program Secara Umum

Program Bot mobil secara umum bekerja sebagai fungsi yang mengembalikan beberapa aksi, seperti yang telah disebutkan pada BAB I. Bot mobil akan bekerja mencari aksi terbaik yang dapat dilakukan mobil dengan memperhatikan prioritas yang ada. Algoritma *greedy* diimplementasikan dengan mencari jalur terbaik yang mungkin dilalui oleh mobil dengan memberi poin pada setiap *obstacle* ataupun *power-ups*, sehingga jalur yang dilalui adalah jalur yang memiliki poin tertinggi.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Elemen Algoritma Greedy

Himpunan kandidat: himpunan *command* tak terhingga yang dapat dipilih, yaitu NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP, FIX.

Himpunan solusi: *command-command* yang terpilih.

Fungsi solusi: memeriksa apakah himpunan solusi mengakibatkan mobil mencapai *finish line*.

Fungsi seleksi: memilih *command* yang paling optimal dengan mengkalkulasi poin dari tiap jalur dengan setiap jenis *block* memiliki bobot poinnya masing-masing, dengan beberapa prioritas ketika kondisi tertentu terjadi.

Fungsi kelayakan: tidak ada, karena setiap kandidat yang terseleksi pastilah layak untuk menjadi elemen dari himpunan solusi.

Fungsi objektif: meminimumkan jumlah langkah yang dibutuhkan untuk mencapai *finish line*.

3.2 Alternatif Solusi Algoritma Greedy

Tiga buah alternatif ide solusi untuk permainan ini diajukan, yaitu sebagai berikut.

1. Mobil memiliki prioritas untuk menghindari semua jenis *obstacle* pada *track* dengan tujuan *speed* tidak akan turun. Dengan *speed* setiap *round* tetap terjaga dan mungkin naik, maka harapannya *speed* mobil secara keseluruhan akan tetap tinggi, sehingga lebih cepat sampai pada *finish line*.
2. Mobil memiliki prioritas untuk memilih jalur dengan poin tertinggi, dengan poin di-*assign* terhadap setiap objek yang terdapat pada block, baik *power-up* maupun *obstacle*. Dengan jalur yang dipilih selalu yang “terbaik”, harapannya pilihan tersebut dapat membawa keuntungan yang besar terhadap mobil, yaitu *power-up* yang banyak dan berkualitas serta *obstacle* yang dihindari.
3. Mobil memiliki prioritas untuk memiliki kecepatan tinggi dengan menggunakan *boost* ketika tidak ada *obstacle* di depannya, kemudian memilih jalur dengan poin tertinggi. Dengan *speed* tetap tinggi dan jalur yang dipilih selalu yang “terbaik”, harapannya pilihan tersebut membawa mobil memiliki *speed* yang tinggi dan memiliki *power-up* yang banyak.

3.3 Analisis Efisiensi dari Alternatif Solusi

Ide solusi pertama memiliki fokus pada pengecekan obstacle yang terdapat pada jalur yang mungkin dilalui, sehingga untuk setiap *block* yang mungkin dilalui, akan dilakukan

pengecekan apakah *block* mengandung *obstacle* atau tidak. Jadi, ide solusi ini memiliki efisiensi *O(n)* dengan *n* menunjukkan *block* yang mungkin dilalui.

Ide solusi kedua memiliki fokus pada pengecekan jalur dengan poin tertinggi pada semua jalur yang mungkin dilalui, sehingga untuk setiap *block* yang mungkin dilalui, akan dilakukan pengecekan nilai poin dari *block* tersebut, kemudian dijumlahkan semua poinnya. Jadi, ide solusi ini memiliki efisiensi *O(n)* dengan *n* menunjukkan *block* yang mungkin dilalui.

Ide solusi ketiga memiliki fokus pada penggunaan *power-up boost*, sehingga diperlukan pengecekan apakah terdapat *boost* dalam daftar *power-up* yang dimiliki, dan juga pengecekan jalur dengan poin tertinggi pada semua jalur yang mungkin dilalui. Jadi, ide solusi ini memiliki efisiensi *O(m+n)* dengan *m* menunjukkan *power-up* yang dimiliki dan *n* menunjukkan *block* yang mungkin dilalui.

3.4 Analisis Efektivitas dari Alternatif Solusi

Ide solusi pertama memprioritaskan penghindaran *obstacle* agar *speed* tetap terjaga. Dari *initial speed*, mobil akan berusaha *accelerate* sebisa mungkin dan membelok ke kiri atau kanan ketika terdapat *obstacle* pada jalur depan. Ide ini memiliki efektivitas yang cukup baik karena *speed* suatu saat akan mencapai pada puncaknya dan akan jarang turun ke *speed* rendah. Namun, hal ini hanya berlaku jika jumlah *obstacle* pada *race track* sedikit, yang tidak dapat diprediksi dengan keacakan *race track*. Akibatnya, ide solusi ini hanya mengakibatkan kondisi perulangan: "*speed* rendah → *speed* tinggi → gagal menghindari *obstacle* → *speed* rendah" dengan jarang sekali penggunaan *power-up* karena (1) mobil jarang mengoleksi *power-up* dan (2) mobil memiliki prioritas untuk bergerak daripada menggunakan *power-up*.

Ide solusi kedua memperhatikan setiap blok yang mungkin dilalui dengan memberikan poin pada *power-ups* dan *obstacle*. Setelah itu, mobil akan memilih jalan yang memiliki poin tertinggi dan menggunakan *power-ups* yang ada. Kelebihan dari solusi ini terletak pada efektivitasnya yang memperhatikan segala blok yang mungkin menguntungkan bagi mobil. Hal ini membuat mobil dapat selalu memilih jalan yang terbaik. Akan tetapi, ide solusi ini memiliki kelemahan, yaitu kurang utilisasi dari *accelerate* sehingga kecepatan mobil tidak terjaga, mobil terlalu fokus itu melakukan perpindahan jalur dan menggunakan *power-ups* yang ada.

Ide solusi ketiga memprioritaskan kecepatan tinggi dengan menggunakan *boost* ketika tidak ada *obstacle* di depannya, kemudian jalur dengan poin tertinggi sebagai prioritas komplemen. Ketika tidak ada *boost*, maka mobil akan memilih jalur dengan poin tertinggi, dengan *boost* adalah elemen dengan poin tertinggi. Sehingga ketika didapatkan *boost*, mobil dapat langsung mencapai kecepatan maksimum tanpa harus melalui beberapa *speed state*. Ketika *boost*, mobil juga dapat secara aktif berpindah lane untuk menghindari *obstacle* agar *speed* tetap terjaga. Alhasil, mobil dapat secara cepat mencapai kecepatan maksimum, berpindah jalur menghindari *obstacle*, mengoleksi *power-up*, dan menggunakan *power-up* dengan baik.

3.5 Strategi Greedy yang Dipilih

Ketiga ide solusi memiliki efisiensi dalam notasi waktu asimptotik yang mirip, tetapi ide ketiga memiliki efektivitas yang lebih tinggi. Jadi, untuk strategi algoritma untuk *bot* dalam permainan Overdrive, dipilih ide solusi ketiga untuk diimplementasi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma

Berikut adalah algoritma yang diimplementasikan pada fungsi utama.

```
{
Note:
speedMap(0) => speedMobil = 0
speedMap(1) => speedMobil = 3
speedMap(2) => speedMobil = 6
speedMap(3) => speedMobil = 8
speedMap(4) => speedMobil = 9
speedMap(5) => speedMobil = 15
 }

{ Memperbaiki mobil apabila terjadi kerusakan supaya kecepatan mobil tidak terpengaruh secara signitfikan }
if (kerusakanMobil >= 2) then
   return FIX

{ Menggunakan boost apabila tidak ada obstacle di jalur mobil atau saat kecepatan mobil sudah terlalu rendah. Akan tetapi, sebelum di boost, dilakukan fix terlebih dahulu apabila terdapat damage pada mobil sehingga kecepatan mobil maksimal }
if (adaBoost AND carBelumDiBoost) then
   if (!obstacleDiJalurTengah AND kerusakanMobil = 1) then
      return FIX
   if (!obstacleDiJalurTengah AND speedMap(3)) then
      return BOOST

{ Apabila tidak memiliki boost, maka akan dilakukan akselerasi }
if (kecepatanMobil = speedMap(0) then
   return ACCELERATE

{ Menghitung poin dari tiap lajur dengan menggunakan algoritma greedy yang telah ditetapkan }
poinKiri = totalPoin(jalurKiri)
poinTengah = totalPoin(jalurTengah) + 0.15 { Bonus poin di jalur tengah karena jalur tengah mendapatkan bonus 1 block }
poinKanan = totalPoin(jalurKanan)

{ Apabila kecepatan mobil sudah tinggi, akan dilakukan penggunaan powerups guna menjaga kecepatan dari mobil ataupun untuk merusak/menghambat mobil lawan }
if kecepatanMobil >= speedMap(4) then
   if (adaLizard) then
      { Dilakukan pengecekan obstacle supaya lizard tidak digunakan secara sia-sia }
      if (!obstacleDiJalurTengah) then
         if (adaBoost AND kecepatanMobil != speedMap(5)) then
```

```
            return BOOST

        { Langsung belok kiri apabila jalur kiri merupakan jalur yang terbaik }
        if (!obstacleDiJalurKiri AND poinKiri>poinTengah>poinKanan) then
            return TURN_LEFT

        { Langsung belok kanan apabila jalur kanan merupakan jalur yang terbaik }
        if (!obstacleDiJalurKanan AND poinKanan>poinTengah>poinKiri) then
            return TURN_RIGHT

        { Apabila tidak ada obstacle pada jalur tengah, maka akan digunakan powerups untuk
merusak/menghambar mobil lawan }
        if (!obstacleDiJalurTengah) then
            if (adaTweet) then
                return taruhCyberTruck(diJalurMobilLain)
            if (adaMinyak)
                return OIL
            return ACCELERATE

        { Apabila tidak ada obstacle di jalur kiri maka belok kiri }
        if (!obstacleDiJalurKiri) then
            return TURN_LEFT

        { Apabila tidak ada obstacle di jalur kanan maka belok kanan }
         if (!obstacleDiJalurKanan) then
            return TURN_RIGHT

        { Apabila tidak ada yang menuhi syarat di atas, maka akan digunakan lizard untuk
menjaga kecepatan mobil }
        return LIZARD

{ Mengganggu mobil lawan apabila kecepatan mobil sedang rendah dan kecepatan mobil
lawan sedang tinggi dan mobil lawan sedang dalam keadaan unggul }
if ( (kecepatanMobil <= speedMap(2) AND kecepatanMobilLawan >= speedMap(3))
        AND posisiMobil < posisiMobilLawan ) then

    { Mengganggu dengan EMP }
    if (adaEmp) then
        return EMP

    { Pastikan jalur mobil sendiri aman terlebih dahulu }
    if (poinTengah >= 0.1) then
        { Mengejar ketertinggalan jarak dengan lawan }
        if (adaBoost) then
            return BOOST

        { Mengganggu dengan CyberTruck }
        if (adaTweet) then
            return taruhCyberTruck(diJalurMobilLain)
```

```
{ Apabila tersangkut tepat di belakang mobil lawan, langsung mengganti jalur mobil }
if ( (posisiMobil + 1 = posisiMobilLawan) AND jalurMobil = jalurMobilLawan then
   if (poinKiri >= poinKanan) then { Lakukan pengecekan poin terlebih dahulu }
      return TURN_LEFT
   else
      return TURN_RIGHT

{ Apabila ketiga jalur tidak menguntungkan sama sekali, akan dilakukan peloncatan
obstacle apabila powerup lizard tersedia }
if (poinKiri < 0 AND poinTengah < 0 AND poinKanan < 0) then
   if (adaLizard) then
      return LIZARD

{ Mengecek jalur tarbaik untuk diambil sesuai dengan poin }
if (poinKiri > poinTengah AND poinKiri > poinKanan) then
   return TURN_LEFT
if (poinKanan >= poinKiri AND poinKanan > poinTengah) then
   return TURN_RIGHT

{ Menggunakan powerups yang dapat menggangu lawan }
if (adaEmp) then
   if (
         (jalurMobil = jalurMobilLawan OR
         jalurMobil = jalurMobilLawan+1 OR
         jalurMobil = jalurMobilLawan-1)
         AND posisiMobil < posisiMobilLawan
   ) then
      return EMP

if (adaOil AND kecepatanMobil >= speedMap(4)) then
   return OIL

if (adaBoost AND kecepatanMobil <= speedMap(2)) then
   return BOOST

{ Pilihan terakhir apabila syarat-syarat yang ditentukan tidak ada terpenuhi sama sekali }
return ACCELERATE
```

Untuk mendukung fungsi totalPoin() pada algoritma di atas, berikut adalah implementasi dari fungsi tersebut.

```
{ Apabila jalur tidak dapat dicapai maka tidak mungkin dijalani }
if (jalur = null) then
   return –1000

poin = 0
obstacleCount = < 0, 0, 0, 0 > {membentuk tuple}
{ Isi tuple => CyberTruck, Wall, OilSpill, Mud }

{ Lakukan iterasi pada jalur untuk mengecek apa isi block yang terkandung pada jalur }
for (block in jalur) do
```

```
    { Lakukan penambahan poin apabila block merupakan powerups }
    { Nilai yang diberikan ditentukan dengan pertimbangan poin dan kalkulasi yang telah
dilakukan }
    if (block = OIL_POWER) then poin += 0.2
    else if (block = BOOST) then poin += 2.15
    else if (block = LIZARD) then poin += 1.2
    else if (block = TWEET) then poin += 1.1
    else if (block = EMP)
        { Apabila sedang ketinggalan maka akan diprioritaskan emp untuk mengejar lawan }
        if (posisiMobil < posisiMobilLawan) then poin += 1.47
        { Apabila sedang tidak ketinggalan maka tidak begitu perlu untuk mengambil emp }
        else poin += 1.03

    { Lakukan penambahan obstacleCount untuk block obstacle }
    else
        if (block = MUD) then obstacleCount[3] += 1
        else if (block = OIL_SPILL) then obstacleCount[2] += 1
        else if (block = WALL) then obstacleCount[1] += 1
        else if (block = CYBERTRUCK) then obstacleCount[0] += 1

damage = 0
{ Lakukan looping untuk menghitung damage yang diterima mobil dan pengurangan poin
mobil dengan constrain damage 5 karena sesuai dengan peraturan di mana damage =
min(damageJalur, 5) sehingga apabila damage telah lebih dari 5 poin jalur tersebut tidak
dikurangi lagi }
{ Singkatnya ambil saja jalur dengan powerup terbanyak apabila semua jalur akan merusak
mobil sampai damagenya mencapai lebih dari 5 }

for (i=0, i<4, i++) do
    while (obstacleCount[i] > 0 AND damage<5) do
        obstacleCount[i] -= 1
        if (i = 0) then
            point −= 100 { Kelompok kami memutuskan untuk tidak menabrak cybertruck }
            damage += 2
        else if (i = 1) then
            point −= 2
            damage += 2
        else if (i = 2) then
            point −= 0.82
            damage += 1
        else
            point −= 0.8
            damage += 1

{ Mengembalikan poin setelah selesai dikalkulasi sesuai dengan algoritma greedy point
based yang dibuat oleh kelompok kami }
return poin
```

4.2 Struktur Data

Permainan Overdrive menggunakan struktur data berjenis tuple dan enumeration, ditandai dengan “Tuple” dan “Enum”, dengan daftar sebagai berikut.

1. Tuple Car:
   - int id
   - Position position
   - int speed
   - int damage
   - State state
   - PowerUps powerups
   - Boolean boosting
   - int boostCounter

2. Tuple GameState:
   - int currentRound
   - int maxRounds
   - Car player
   - Car opponent
   - List<Lane[]> lanes

3. Tuple Lane:
   - Position position
   - Terrain terrain
   - int occupiedByPlayerId

4. Tuple Position
   - int lane
   - int block

5. Enum Direction = {“FORWARD”, “BACKWARD”, “LEFT”, “RIGHT”}

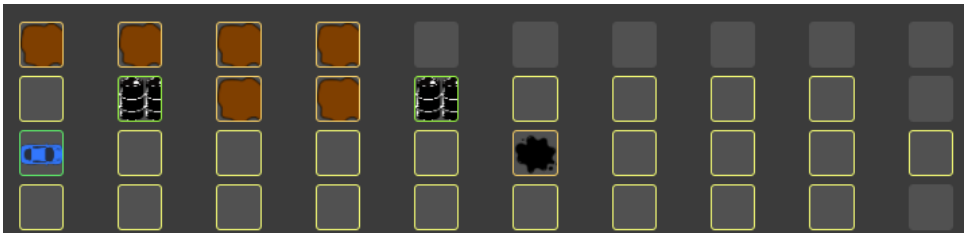
6. Enum PowerUps = {“BOOST”, “OIL”, “TWEET”, “LIZARD”, “EMP”}

7. Enum State = {“ACCELETATING”, “READY”, “NOTHING”, “TURNING_RIGHT”, “TURNING_LEFT”, “HIT_MUD”, “HIT_OIL”, “DECELERATING”, “PICKED_UP_POWERUP”, “USED_BOOST”, “USED_OIL”, “USED_LIZARD”, “USED_TWEET”, “HIT_WALL”, “HIT_CYCBER_TRUCK”, “FINISHED”}

8. Enum Terrain = {“EMPTY”, “MUD”, “OIL_SPILL”, “OIL_POWER”, “FINISH”, “BOOST”, “WALL”, “LIZARD”, “TWEET”, “EMP”, “CYCBERTRUCK”}

4.3 Analisis dari Desain Solusi

4.3.1 Analisis Pertama

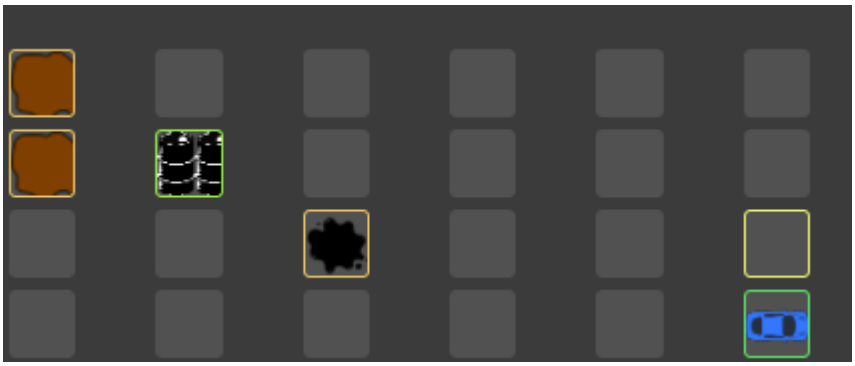


Jalur kiri memiliki 2 buah mud dan 2 buah oil powerups yang dimana 2 buah mud dapat merusak mobil dan mengurangi kecepatan mobil dan juga 2 buah oil powerups tidak begitu berharga untuk dikumpulkan mengingat mobil harus diperbaiki dulu akibat kerusakan yang disebabkan oleh mud.

Jalur tengah memiliki 1 buah oil spill tanpa powerups yang dimana oil spill tidak hanya dapat merusak mobil, tetapi juga dapat menurunkan skor akhir.

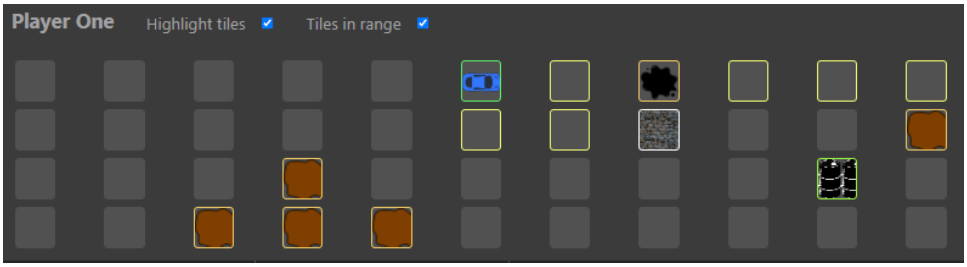
Jalur kanan tidak memiliki hambatan apa apa.

Oleh karena itu, dapat disimpulkan bahwa jalur terbaik merupakan jalur kanan.



Best Move: TURN_RIGHT
Taken Move: TURN_RIGHT
Kesimpulan: Bot berhasil mengambil keputusan yang paling tepat sesuai analisis yang dilakukan.

4.3.2 Analisis Kedua

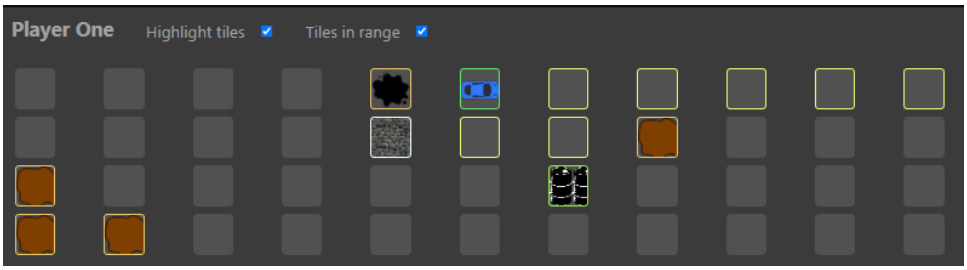


Jalur kiri tidak dapat diakses dikarenakan bot berada pada jalur paling atas.

Jalur tengah memiliki sebuah oil spill yang di mana oil spill tersebut dapat merusak mobil dan mengurangi skor akhir.

Jalur kanan memiliki sebuah wall yang di mana wall tersebut dapat membuat kecepatan mobil menurun drastis dan dapat mengurangi skor akhir.

Dikarenakan pengurangan poin akibat wall lebih tinggi daripada oil spill, maka dapat disimpulkan bahwa move yang paling tepat adalah maju menerobos oil spill.



Best Move: ACCELERATE
Taken Move: ACCELERATE
Kesimpulan: Bot berhasil mengambil keputusan yang paling tepat sesuai analisis yang dilakukan.

BAB V

KESIMPULAN DAN SARAN

5.1 Simpulan

Algoritma *Greedy* dapat diimplementasikan dalam membuat bot dalam permainan Overdrive. Pada permainan Overdrive, strategi *greedy* diaplikasikan untuk mencari setiap aksi terbaik yang dapat dilakukan oleh mobil di setiap roundnya. Strategi *greedy* yang digunakan diimplementasikan menggunakan sistem poin dengan poin plus untuk *power-ups* dan poin minus untuk *obstacle*.

5.2 Saran

Kedepannya, program dapat dioptimasi dengan meningkatkan efektivitas algoritma *greedy*. Efektivitas dapat ditingkatkan dengan melakukan analisis lebih dalam terhadap prioritas setiap *power-ups* dan *obstacle* yang akan ditelusuri. Dengan analisis tersebut, program dapat menghasilkan aksi yang lebih efektif tiap roundnya.

DAFTAR PUSTAKA

R. Munir, School of Electrical Engineering and Informatics, Bandung Institute of Technology, lecture slide: Algoritma Greedy, 2021.

https://github.com/EntelectChallenge/2020-Overdrive (diakses 06 Februari 2022).

LINK REPOSITORY DAN VIDEO

Video demo : https://youtu.be/bSfO8lVtIqA
Repository git : https://github.com/aldwinhs/TugasBesarStima1-13520071