LAPORAN TUGAS BESAR
IF2211 STRATEGI ALGORITMA

Overdrive Car-Bot

HADEH

Disusun oleh :

Kelompok 07 Kelas 02

Suryanto	13520059
Vieri Mansyl	13520092
Brianaldo Phandiarta	13520113

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

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.

Gambar 1. Ilustrasi permainan Overdrive

Pada tugas besar kali ini, kami diminta untuk membuat sebuah *bot* untuk bermain permainan Overdrive yang telah dijelaskan sebelumnya. Untuk memulai, anda dapat mengikuti panduan singkat sebagai berikut.

1. Download latest release starter pack.zip dari tautan berikut https://github.com/EntelectChallenge/2020-Overdrive/releases/tag/2020.3.4
2. Untuk menjalankan permainan, kalian butuh beberapa requirement dasar sebagai berikut.
    a. Java (minimal Java 8): https://www.oracle.com/java/technologies/downloads/#java8
    b. IntelIiJ IDEA: https://www.jetbrains.com/idea/
    c. NodeJS: https://nodejs.org/en/download/
3. Untuk menjalankan permainan, kalian dapat membuka file “run.bat” (Untuk Windows dapat buka dengan double-click, Untuk Linux/Mac dapat menjalankan command “make run”).
4. Secara default, permainan akan dilakukan diantara *reference bot* (default-nya berbahasa Java) dan starter *bot* (default-nya berbahasa JavaScript) yang disediakan. Untuk mengubah hal tersebut, silahkan edit file “game-runner-config.json”. Anda juga dapat mengubah file “*bot.json*” dalam direktori “starter-bots” untuk mengatur informasi terkait *bot* anda.
5. Silahkan bersenang-senang dengan memodifikasi *bot* yang disediakan di starter-bots. Ingat bahwa *bot* kalian harus menggunakan bahasa Java dan di-build menggunakan IntelIiJ sebelum menjalankan permainan kembali. Dilarang menggunakan kode program yang sudah ada untuk pemainnya atau kode

program lain yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, tetapi belajar dari program yang sudah ada tidak dilarang.

6. (Optional) Anda dapat melihat hasil permainan dengan menggunakan visualizer berikut https://github.com/Affuta/overdrive-round-runner
7. Untuk referensi lebih lanjut, silahkan eksplorasi di tautan berikut.

Strategi greedy yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mencapai garis finish lebih awal atau mencapai garis finish bersamaan tetapi dengan kecepatan lebih besar atau memiliki skor terbesar jika kedua komponen tersebut masih bernilai imbang. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menggunakan powerups begitu ada untuk mengganggu mobil musuh. Buatlah strategi greedy terbaik, karena setiap *bot* dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 1 (TBD).

Strategi greedy harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas mereka dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

BAB II
LANDASAN TEORI

2.1. Algoritma *Greedy*

Algoritma *Greedy* merupakan algoritma yang selalu mengambil langkah optimum pada setiap perjalanan dalam menyelesaikan suatu permasalahan dengan harapan langkah optimum yang diambil merupakan langkah terbaik untuk mencapai hasil optimum. Akibatnya, pada setiap pengambilan langkah, diperlukan analisis terhadap setiap kasus sehingga dapat dipilih langkah yang paling optimum. Faktanya, algoritma *greedy* tidak selalu memberikan hasil optimum dari suatu permasalahan, tetapi memberikan hasil yang mendekati 'hasil terbaik' dengan memakan waktu yang terhitung cepat. Hal ini yang menjadikan algoritma *greedy* menjadi salah satu algoritma yang masih sering dipakai sampai saat ini.

Terdapat elemen-elemen yang perlu ditentukan dalam algoritma greedy, yaitu :

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)
5. memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
6. Fungsi kelayakan (feasible)
   memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
7. Fungsi objektif
   memaksimumkan atau meminimumkan

Secara umum, skema dari algoritma *Greedy* (disusun berdasarkan elemen-elemennya) dapat direpresentasikan dalam pseudocode sebagai berikut.

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
  x : kandidat
  S : himpunan_solusi
Algoritma
  S ← {} { inisialisasi S dengan kosong }
  while (not SOLUSI(S)) and (C != {} ) do
    x ← SELEKSI(C) { pilih sebuah kandidat dari C}
    C ← C - {x} { buang x dari C karena sudah dipilih }
    if LAYAK(S U {x}) then
    {x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
       S ← S U {x} { masukkan x ke dalam himpunan solusi }
    Endif
  Endwhile   {SOLUSI(S) or C = {} }
  if SOLUSI(S) then { solusi sudah lengkap }
     return S
  else
     write('tidak ada solusi')
  endif
```

2.2. Cara Kerja Program Overdrive

Permainan Overdrive merupakan permainan balapan antar dua mobil, yang mana kedua mobil merupakan *bot* hasil rancangan pengembang (*developer*) yang diadu. Permainan dinyatakan berakhir apabila salah satu dari *bot* mobil melewati garis akhir dibanding mobil lainnya. Terdapat beberapa kasus yang menjadikan permainan berakhir, yaitu:

*Asumsi terdapat mobil A dan mobil B yang sedang bertanding balapan.

- Mobil A melewati mobil B terlebih dahulu – mobil A dinyatakan sebagai pemenang
- Mobil A dan mobil B melewati garis akhir bersamaan – mobil dengan kecepatan tercepat dinyatakan sebagai pemenang
- Mobil A dan mobil B melewati garis akhir bersamaan dan berakhir dengan kecepatan yang sama – mobil dengan skor terbanyak dinyatakan sebagai pemenang

2.2.1 Pengaturan *Game Engine*

Sebelum menjalankan *game engine Overdrive*, kita harus mempersiapkan dua *bot* untuk bertarung satu sama lain. Dalam mempersiapkan *bot*, *bot* dapat di *build* menggunakan Maven Project. Maven Project akan mem-*build bot* dengan menggunakan *source code* beserta *dependencies*-nya. Setelah *bot* berhasil di-*build*, akan terbentuk *folder* baru bernama *target* yang berisi .jar dari *bot*. Terakhir, kita perlu menyesuaikan informasi terkait *bot* pada *file* bot.json.

Untuk menjalankan *game engine Overdrive*, pastikan game-config.json dan game-runner-confing.json sudah sesuai dengan kebutuhkan kita. Pada *operating system* Windows, *game engine Overdrive* dapat dijalankan dengan menjalankan run.bat. Pada MacOS dan Unix, *game engine Overdrive* dapat dijalankan melalui terminal pada direktori terluar dengan make run.

2.2.2 Implementasi Algoritma *Greedy* dalam Game *Overdrive*

Algoritma *greedy* diimplementasikan di dalam file bot.java yang terdapat di dalam folder *starter-bots* pada kelas *run()*. Struktur dari folder bot di bawah menunjukkan keberadaan dari algoritma *greedy* pada *game engine overdrive* kami.

```
<<<DIREKTORI BOT>>>
→Starter-bots
  →Java
    bot.json  {file untuk men-setting identitas dari bot}
    →Src
      →Main
        →Java
            →Za
              →Co
                →Entelect
                  →Challenge
                      Bot.java {Algoritma Greedy terdapat di dalam method run().}
                      Main.java {program utama}
    →target
      java-starter-bot-jar-with-dependencies.jar{ hasil compile dari program bot
      beserta dependencies-nya pada library yang digunakan}
      →classes
        →Za
```

```
→Co
  →Entelect
    →Challenge
      Bot.class
      Main.class
```

File bot.java akan mencari pilihan terbaik dari *n* buah *command* yang dapat dilakukan. Dengan cara berpikir seperti itu, program akan melakukan perbandingan hasil dari tiap 'simulasi permainan' pada seluruh langkah yang dapat diambil. 'Simulasi' yang dioperasikan oleh program ialah sebagai berikut.

1. nothing() : kondisi *bot* apabila menggunakan command DO_NOTHING
2. accelerate() : kondisi *bot* apabila menggunakan command ACCELERATE
3. decelerate() : kondisi *bot* apabila menggunakan command DECELERATE
4. turnLeft() : kondisi *bot* apabila menggunakan command TURN_LEFT
5. turnRight() : kondisi *bot* apabila menggunakan command TURN_RIGHT
6. useBoost() : kondisi *bot* apabila menggunakan command USE_BOOST
7. useLizard() : kondisi *bot* apabila menggunakan command USE_LIZARD
8. useTweet() : kondisi *bot* apabila menggunakan command USE_TWEET
9. useEmp() : kondisi *bot* apabila menggunakan command USE_EMP

Hasil dari tiap 'simulasi' tersebut akan disortir lalu diambil 'simulasi' pada kondisi yang menghasilkan langkah paling optimum. File bot.java yang telah berisi implementasi *algoritma greedy* akan di-*build* dengan menggunakan IDE IntelliJ. Program permainan dapat di-*run* dengan menggunakan file *run.bat*. *Bot* yang digunakan dalam permainan menggunakan *jar.file* yang telah berhasil di-*built* sebelumnya.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Elemen Algoritma *Greedy* pada Permainan

1. Himpunan Kandidat
   Himpunan kandidat pada permainan Overdrive berupa himpunan command yang dapat digunakan oleh *bot*, yaitu FIX, DO_NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET, dan USE_LIZARD, USE_EMP.
2. Himpunan Solusi
   Himpunan solusi pada permainan Overdrive berupa himpunan *command* yang dieksekusi pada setiap *round*-nya.
3. Fungsi Solusi
   Tidak ada fungsi solusi
4. Fungsi Seleksi
   Pilih command yang memberikan speed dan poin tertinggi pada akhir round.
5. Fungsi Kelayakan
   Fungsi kelayakan berupa pemeriksaan *command* yang digunakan oleh *bot* pada setiap *round*. Pemeriksaan kelayakan dilakukan terhadap *command* TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_TWEET, USE_LIZARD, dan USE_EMP.
6. Fungsi Objektif
   Memenangkan permainan dengan mencapai garis *finish* secepat mungkin.

3.2 Alternatif Solusi *Greedy*

Terdapat beberapa alternatif strategi *greedy* yang dapat dipakai untuk mencari solusi pada permainan ini. Beberapa diantaranya adalah sebagai berikut :

1. *Greedy by disturbing Opponent*

   Pada dasarnya, untuk memenangkan permainan *Overdrive*, *bot* tidak harus mencapai garis *finish* secepat mungkin, *bot* hanya perlu mencapai garis *finish* lebih cepat dari pada *opponent*. Oleh karena itu, kelompok kami merancang algoritma *greedy by* disturb *opponent*. Dalam teknis permainan, terdapat beberapa *power up* yang dapat digunakan dengan tujuan mengganggu *opponent*, yaitu Oil Item, Tweet, dan EMP. Penggunaan *power up* tersebut dapat digunakan dengan menggunakan *command* USE_OIL, USE_TWEET, dan USE_EMP. Pertama, *bot* akan mencari *path* yang memiliki *power ups* yang terbanyak, khususnya *power ups* yang dapat mengganggu *opponent*. Kemudian, ketika *bot* memiliki *power ups*, dalam validasinya dapat menggunakan fungsi kelayakan, *bot* akan terus memakai *power up* yang paling sesuai untuk mengganggu musuh, dengan prioritas USE_EMP, USE_TWEET, dan USE_OIL. Ada kelemahan dalam alogritma *greedy* ini, yaitu karena *bot* memprioritaskan untuk mengganggu *opponent*, *bot* akan lebih sering menabrak *obstacle*. Hal ini dikarenakan saat penggunaan *power up*, *bot* tidak dapat memilih untuk menghindari *obstacle*. Selain itu, *opponent* juga dapat dengan mudah menghindar *obstacle power up*, kecuali Emp. Oleh karena itu, kami menilai bahwa strategi ini tidak efektif. Kompleksitas algoritma strategi ini adalah *O(n)* dengan *n* adalah himpunan kandidat dan sudah terurut berdasarkan prioritasnya, yaitu USE_EMP, USE_TWEET, USE_OIL, dan *command* sisanya.

2. *Greedy by Damage*
   Dalam permainan *Overdrive*, untuk mencapai garis *finish*, *speed* dari suatu mobil sangatlah berpengaruh. Selain itu, *maximum speed* dari suatu mobil berbanding lurus dari *damage* yang diterima oleh mobil. Oleh karena itu, program dirancang dengan algoritma *greedy by damage*. Secara garis besar, strategi ini akan mencari solusi sehingga *command* yang dieksekusi oleh *bot* akan menghasilkan *damage* yang terkecil terhadap *bot* pada *round* tersebut. Dalam penerapannya, setiap *command* akan diprediksi menggunakan fungsi prediksi yang telah dibuat.

Fungsi prediksi tersebut akan mengembalikan prediksi *damage* yang diterima saat menjalankan suatu *command*. Dari hasil prediksi, akan diambil *command* dengan *damage* terkecil yang diterima oleh *bot*. Strategi ini berhasil menghindari *obstacle* dan mencari *path* yang paling aman untuk mobil. Akan tetapi, strategi ini kurang optimal. Hal ini dikarenakan *bot* akan jarang menggunakan seluruh *power up* yang telah didapatkannya, kecuali Lizard. Kompleksitas algoritma strategi ini adalah *O(n)* dengan *n* adalah himpunan kandidat dan *O(m)* dengan *m* adalah banyaknya pengecekan *terrain* pada saat memprediksi *damage* akhir saat menjalankan suatu *command*.

3. *Greedy by Score*
   Dalam permainan *Overdrive*, terdapat parameter *score* yang akan berubah seiiring jalannya permainan. *Score* dari *bot* dapat bertambah 4 ketika mengambil ataupun menggunakan *power up*. Selain itu, *score* dari *bot* juga dapat berkuring sebanyak 3 ketika menabrak mud, 4 ketika menabrak oil spill, dan 5 ketika menggunakan *command* yang tidak valid. Sehingga, dengan menggunakan parameter *score*, strategi dapat memprediksi *command* yang terbaik dari kombinasi *damage*, *speed*, dan *power up*. Sama seperti strategi *greedy by damage*, program akan memprediksi *score* akhir dari setiap *command* menggunakan fungsi prediksi yang telah dibuat. Kemudian, *command* dengan *score* tertinggi akan digunakan. Kelemahan dari strategi ini adalah ketika pada suatu *lane* terdapat *obstacle* dan *power up*, *score* tetap dapat menjadi yang tertinggi sehingga digunakan pilihan *command* tersebut. Hal ini dapat mencegah untuk mencapai solusi optimal pada akhir permainan. . Kompleksitas algoritma strategi ini adalah *O(n)* dengan *n* adalah himpunan kandidat dan *O(m)* dengan *m* adalah banyaknya pengecekan *terrain* pada saat memprediksi *score* akhir saat menjalankan suatu *command*.

4. *Greedy by Speed*
   Dalam permainan *Overdrive*, untuk mencapai garis *finish*, *speed* dari suatu mobil sangatlah berpengaruh. Oleh karena itu, program dirancang dengan algoritma *greedy by speed*. Secara garis besar, strategi ini akan mencari solusi sehingga *command* yang dieksekusi oleh *bot* akan mencapai *speed* tercepat pada *round* tersebut. Dalam penerapannya, setiap *command* akan diprediksi menggunakan fungsi prediksi yang telah dibuat. Fungsi prediksi tersebut akan mengembalikan prediksi *damage* yang diterima saat menjalankan suatu *command*. Dari hasil prediksi, akan diambil *speed* dengan *speed* terbesar yang diterima oleh *bot* pada akhir *round*. Strategi ini berhasil menghindari *obstacle* dan mencari *path* yang menghasilkan *speed* tercepat pada akhir *round*. Kompleksitas algoritma strategi ini adalah *O(n)* dengan *n* adalah himpunan kandidat dan *O(m)* dengan *m* adalah banyaknya pengecekan *terrain* pada saat memprediksi *speed* akhir saat menjalankan suatu *command*.

5. *Greedy by Point*
   Menggunakan konsep yang hampir sama dengan strategi *greedy by score*, algoritma *greedy by point*, akan membandingkan *point* hasil prediksi dari fungsi prediksi yang dibuat. *Point* merupakan parameter baru yang dibuat agar dapat memanipulasi bobot *obstacles* dan *power up*. Dengan menggunakan parameter baru ini, program dapat memberikan bobot yang sesuai dengan manfaat atau kerugian dari *power up* ataupun *obstacle*. Kompleksitas algoritma strategi ini adalah *O(n)* dengan *n* adalah himpunan kandidat dan *O(m)* dengan *m* adalah banyaknya pengecekan *terrain* pada saat memprediksi *score* akhir saat menjalankan suatu *command*.

3.3 Strategi *Greedy* yang digunakan

Strategi yang diimplementasikan pada program ini merupakan gabungan dari *greedy by speed* dan *greedy by point*. Dalam pengimplementasiannya, akan diurutkan terlebih dahulu kandidat-kandidat yang memenuhi syarat berserta prioritas. Himpunan kandidat diurutkan sebagai berikut (prioritas terbaik menuju terburuk): USE_TWEET→USE_EMP→NOTHING→ACCELERATE→DECELERATE→USE_BOOST→USE_LIZARD→TURN_LEFT→TURN_RIGHT→USE_OIL. Jika diperhatikan, urutan himpunan kandidat diawali dengan penggunaan *power up* pengganggu *opponent*. Selain itu, dilakukan juga uji kelayakan. Sebagai contoh, uji kelayakan kandidat USE_TWEET dengan cara memeriksa apakah bot memiliki *power up* Tweet.

Selanjutnya, dibuat suatu fungsi prediksi yang akan mengembalikan parameter-parameter akhir yang akan digunakan untuk perbandingan setiap *command*-nya. Dari hasil yang didapat, himpunan hasil prediksi akan di urutkan terlebih dahulu terhadap *speed*-nya. Hal ini berhubungan dengan konvensi kami, yang mana parameter *speed* menjadi hal penentu utama agar objektif dari permainan dapat tercapai. Setelah pengurutan dilakukan terhadap *speed*, pengurutan kembali dilakukan ketika terdapat *speed* akhir yang sama dari *command-command*. Pengurutan ini dilakukan terhadap *point*. *Greedy by point* ini digunakan untuk menjadi pembanding sekunder ketika terdapat hasil prediksi yang sama setelah perbandingan *speed*. Hal ini dikarenakan perbandingan menggunakan *point* diharapkan dapat menjadi pembanding *damage* dan pengambilan *power up* sekaligus. *Point* Pada strategi ini dinilai berdasarkan hasil konvensi kami dan dinilai berdasarkan kondisi-kondisi yang tertera di bawah ini.

1. Pick up Boost mendapatkan 5 point.
2. Pick up Tweet mendapatkan 4 point.
3. Pick up EMP mendapatkan 3 point.
4. Pick up Lizard mendapatkan 2 point.
5. Pick up Oil mendapatkan 1 point.
6. Menabrak Mud mengurangkan 2 point.
7. Menabrak Oil Spill mengurangkan 2 point.
8. Menabrak Wall mengurangkan 5 point.

Setelah dilakukan pengurutan terhadap *point*, akan diambil kandidat solusi, yaitu kandidat pertama dari himpunan kandidat yang telah melewati fungsi kelayakan, seleksi, dan objektif. Selain itu, terdapat penambahan fungsi heuristik yang memanggil *command* FIX ketika *damage* dari mobil lebih dari 2 atau ketika *bot* memiliki *power up* Boost dan damage lebih dari 0.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Hasil program yang telah kami rancang dapat diakses melalui repository berikut.

https://github.com/SurTan02/TUBES-STIMA-1

Alur dari program ini ditunjukkan oleh *pseudocode* program di bawah ini.

4.1.1 Hit Obstacles

Terdapat 4 jenis rintangan pada permainan ini, yaitu *Mud* , *Oil*, *Wall*, dan *Truck*. Fungsi *hitObs* ini akan menunjukkan kondisi bot mobil ketika melewati rintangan-rintangan tersebut. Hasil kondisi bot mobil setelah melewati rintangan-rintangan tersebut akan menjadi parameter dalam penentuan langkah optimum nantinya. Realisasi dari fungsi *hitObs* dinyatakan dibawah berikut.

```
{Pada kode aslinya, terdapat 4 method yaitu hitMud, hitOil, hitWall, dan hitTruck.
Namun, keempat  method ini memiliki algoritma yang sama sehingga akan dikelompokkan
menjadi satu.}

{CarPred adalah kelas yang memiliki karakteristik mirip dengan Class Car}

CarPred hitObs( input Car :  CarPred) → CarPred

KAMUS LOKAL
res : CarPred

ALGORITMA
  If (res.damage > 5) then
      res.damage ← 2
  endif
  {Menentukan nilai prediksi speed dengan menurunkan speed sebesar 1 state}
  {Penurunan speed state juga memiliki berbagai aturan tergantung jenis obstacles
yang ditemui}
  if (res.speed = SPEED_STATE_BEFORE) then
     res.speed ←SPEED_STATE_AFTER
  endif
  {Menentukan nilai prediksi speed dengan membatasinya dengan nilai MaxSpeed}
  if (getMaxSpeed < res.speed) then
     res.speed ← getMaxSpeed
  endif
  return res
```

4.1.2 Check Terrain

Fungsi *checkTerrain* menghitung ‘poin’ dari *lane* yang dilewati bot mobil. ‘Poin’ akan bertambah apabila melewati *power up* dan berkurang apabila melewati *obstacle*. Realisasi dari fungsi *checkTerrain* dinyatakan dibawah berikut.

```
CarPred checkTerrain( input CarPred :  Car , CarLane : Lane) → CarPred

KAMUS LOKAL
res : CarPred

ALGORITMA
  res ← Car
  {memprediksi poin berdasarkan obstacle/power up}
  {Obstacles akan memberikan kerugian, sehingga akan program akan melakukan
prediksi kecepatan dan poin mobil menjadi menurun}
  {PowerUps akan menambah poin prediksi}

  If (carLane.terrain = obstacles) then
```

```
    res.point ← res.point – POINT
  {POINT berupa int tergantung jenis seperti MUD, TRUCK, OIL_SPILL, TRUCK}
  else(carLane.terrain = PowerUp) then
    res.point ← res.point + POINT
  {POINT berupa int tergantung jenis seperti Boost,tweet, emp, lizard, oil}
  endif
  return res
```

4.1.3 Use Command

Fungsi *useCommand* memberikan prediksi kondisi bot mobil terhadap speed dan poinnya apabila mengeksekusi suatu command. Realisasi dari fungsi useCommand dinyatakan dibawah berikut.

```
CarPred useCommand( input CarPred :  Car) → CarPred

KAMUS LOKAL
res           : CarPred
i             : int
initialBlock  : int        {berupa informasi koordinat x mobil}
dest          : int
laneList      : lane

ALGORITMA
  res ← Car
  Assign seluruh informasi Car ke res {Seperti speed, damage, dll}.
  dest ← res.speed
  for i ← initialBlock to initialBlock + dest   {Pengecekan tiap lane}
    if (laneList[i] = NULL or laneList[i] = Finish) then
      break
   endif
    res ← checkTerrain(res, laneList[i])
  endfor

  return res
```

4.1.4 getOpFinalPosition

Fungsi *getOpFinalPosition* mengembalikan ‘prediksi’ posisi bot mobil lawan. Realisasi dari fungsi *getOpFinalPosition* dinyatakan dibawah berikut.

```
int getOpFinalPosition() → int
{Method untuk memprediksi move lawan}

KAMUS LOKAL
temp                     : Position
maxSpeed,accelerateSpeed : int

ALGORITMA
  Temp ← opponent.position

  Depends On opponent.damage :
              (opponent.damage = 5) : maxSpeed ← DMG_MAX_SPEED_5
              (opponent.damage = 4) : maxSpeed ← DMG_MAX_SPEED_4
              (opponent.damage = 3) : maxSpeed ← DMG_MAX_SPEED_3
              (opponent.damage = 2) : maxSpeed ← DMG_MAX_SPEED_2
              (opponent.damage = 1) : maxSpeed ← DMG_MAX_SPEED_1
              (opponent.damage = 0) : maxSpeed ← DMG_MAX_SPEED_0

  Depends On opponent.speed:
              (opponent. speed = MINIMUM_SPEED) : accelerateSpeed ← SPEED_STATE_1
              (opponent. speed = SPEED_STATE_1) : accelerateSpeed ← INITIAL_STATE
              (opponent. speed = INITIAL_SPEED) : accelerateSpeed ← SPEED_STATE_2
              (opponent. speed = SPEED_STATE_2) : accelerateSpeed ← SPEED_STATE_3
              (opponent. speed = SPEED_STATE_3) : accelerateSpeed ← MAXIMUM_SPEED
```

```
              (opponent. speed = BOOST_SPEED) : accelerateSpeed ← BOOST_SPEED

  if (maxSpeed < accelerateSpeed) then
            accelateSpeed ← maxSpeed
  endif

  return temp.block + accelerateSpeed + 1
```

4.1.5 Main Program

Kode utama dari program bot direalisasikan pada *method run()*. Pada *method* ini, terimplementasikan algoritma *Greedy*, yang mana akan dipilih pilihan teroptimum berdasarkan kondisi bot mobil setelah menjalani 'simulasi' berdasarkan setiap jenis *command* yang dijalani.

```
Command run() → Command
KAMUS LOKAL
  carPred : List of CarPred
  myCar : car

ALGORITMA
if hasPowerUp( PowerUps.Tweet, myCar.powerups) then
  UseTweet (myCar)      {Prediksi UseTweet}
Endif

if hasPowerUp( PowerUps.EMP, myCar.powerups) then
  if (opponent.lane = myCar.lane) || opponent.lane berada di kanan/kiri myCar) then
     UseEMP (myCar)            {Prediksi UseEmp}
  endif
endif

{Prediksi 3 command yaitu nothing, accelerate, decelerate}
Nothing(myCar)
accelarate(myCar)
decelarate(myCar)

if hasPowerUp( PowerUps.Boost, myCar.powerups) and (not boosting) then
   UseBoost (myCar)            {Prediksi UseBoost}
Endif

if hasPowerUp( PowerUps.Lizard, myCar.powerups) then
  UseLizard (myCar)            {Prediksi UseLizard}
Endif

If (myCar.position != 1) then  {Batasan agar tidak menabrak dinding samping}
  turnLeft(myCar)              {prediksi turnLeft}
endIf

If (myCar.position != 4) then  {Batasan agar tidak menabrak dinding samping}
  turnRight(myCar)             {prediksi turnRight}
endIf

sort(carPred) by Speed
sort(carPred) by Point
if  myCar.damage  >=  2  or  myCar.damage  =1  and  hasPowerUp(  PowerUps.Boost,
myCar.powerups then
   return fixCommand( )
else
   DEPENDS ON carPred[0]
       carPred[0] == ACCELERATE   : return AccelerateCommand( )
       carPred[0] == DECELERATE   : return DecelerateCommand( )
       carPred[0] == TURN_LEFT    : return ChangeLaneCommand(0)
       carPred[0] == TURN_RIGHT   : return ChangeLaneCommand(1)
       carPred[0] == USE_LIZARD   : return LizardCommand( )
       carPred[0] == USE_BOOST    : return BoostCommand( )
       carPred[0] == USE_EMP      : return EMPCommand( )
```

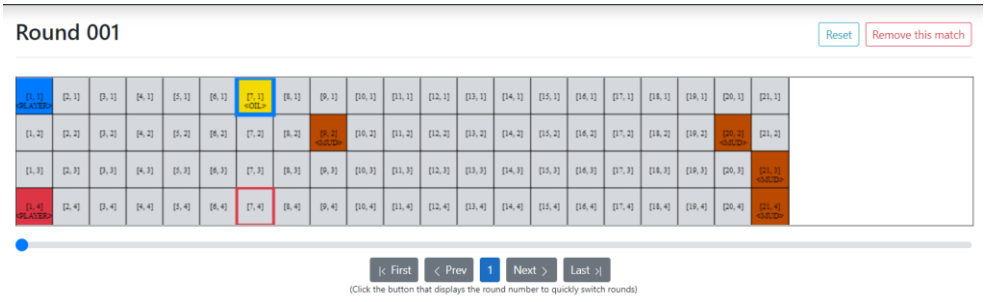
```
        carPred[0] == USE_TWEET     : return TweetCommand( )
           return doNothingCommand();
endif
```

4.2 Struktur Data

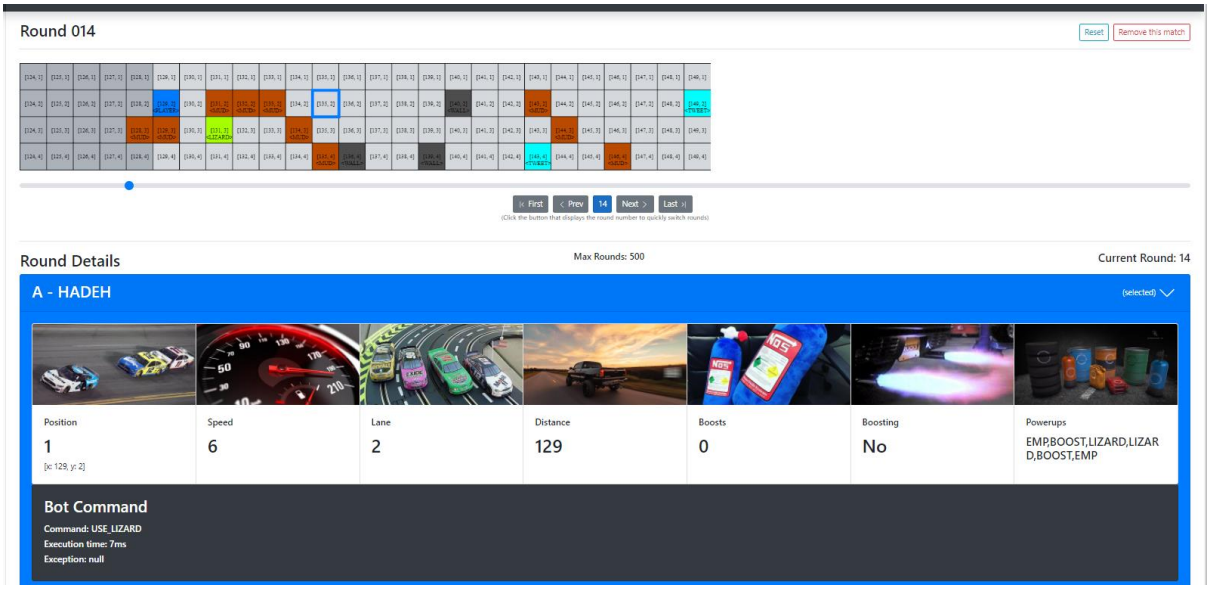
Struktur data yang kami implementasikan di dalam program *overdrive bot* dan berlandaskan atas algoritma *greedy* dipetakan sebagai berikut.

1. **Command** menyimpan kelas-kelas yang berhubungan dengan prosedur dalam *game engine*, terdiri dari :
   a. FixCommand : Berfungsi untuk memperbaiki mobil dengan mengurangi damage sebesar dua satuan.
   b. DoNothingCommand : Tidak Melakukan apa-apa
   c. AccelerateCommand : Mobil akan menambah kecepatan ke state selanjutnya
   d. DecelerateCommand : Mobil akan mengurangi kecepatan ke state sebelumnya
   e. ChangeLaneCommand : Mobil akan berpindah lane, ke kanan atau ke kiri.
   f. LizardCommand : Menggunakan *power up* lizard untuk menghindari *obstacles*
   g. BoostCommand : Menggunakan *power up* Boost untuk menambah kecepatan
   h. EMPCommand : Menggunakan *power up* EMP untuk menyerang musuh
   i. TweetCommand : Menggunakan *power up* Tweet untuk menjatuhkan *cyber truck* di lokasi tertentu
   j. OilCommand : Menggunakan *power up* Oil untuk menjatuhkan oil di lokasi tertentu.
2. **Entities** yang menyimpan kelas-kelas yang berhubungan dengan objek dalam *game engine*, terdiri dari :
   a. Car : Pada main program, myCar didefinisikan sebagai objek dari kelas Car yang memiliki *speed*, damage, position, state, powerups, boosting, dan boostcounter.
   b. CarPred : Class yang merupakan salinan dari Class Car. Digunakan untuk memprediksi kondisi bot mobil saat menjalankan suatu *command* pada suatu round.
   c. GameState : Class yang berisi kondisi global dari permainan.
   d. Lane : Class yang berisi kondisi lane pada permainan. Class ini berfungsi untuk mengecek objek yang berada di atas lane, posisi player, dan informasi tentang *cyber truck.*
   e. Position : Class yang mendeskripsikan lebih jelas terkait posisi mobil. Terdapat atribut *lane* (sumbu y) dan *block* (sumbu x)
3. **Enums** menyimpan nilai iterasi untuk memudahkan dalam pencocokan setiap kemungkinan yang ada , terdiri dari :
   a. Direction : Berisi arah untuk pergerakan mobil
   f. PowerUps : Berisi daftar *power ups* yang tersedia pada game.
   g. State : Berisi daftar *state* yang mungkin dialami oleh player.
   h. Terrain : Berisi daftar objek yang ada pada setiap block di map.
4. **Bot.java** sebagai realisasi file yang berisi kode program bot yang dibuat oleh pemain
5. **Main Java** berisi mekanisme pembacaan state dan operasi eksekusi state

4.3 Pengujian dan Analisis Desain Algoritma Greedy



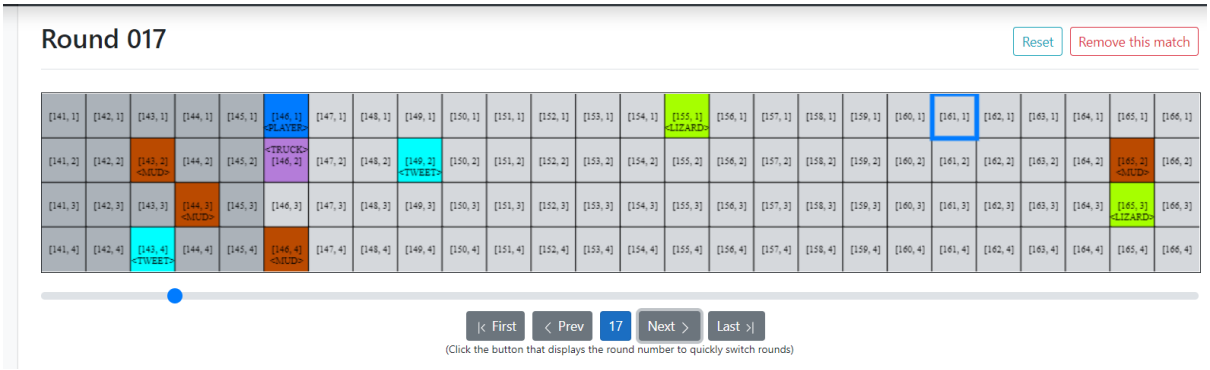
Gambar 4.3.1 Awal Permainan



Gambar 4.3.2 tangkap layar *bot* mobil menggunakan *lizard*

Pada gambar 4.3.2 dapat dilihat bahwa bot memutuskan untuk menggunakan *power up* lizard. Hal ini dilakukan karena *command* ini akan menghasilkan jarak terjauh tanpa mengurangi *speed* dari mobil.

Gambar 4.3.3 tangkap layar *bot* mobil menghindari *mud*

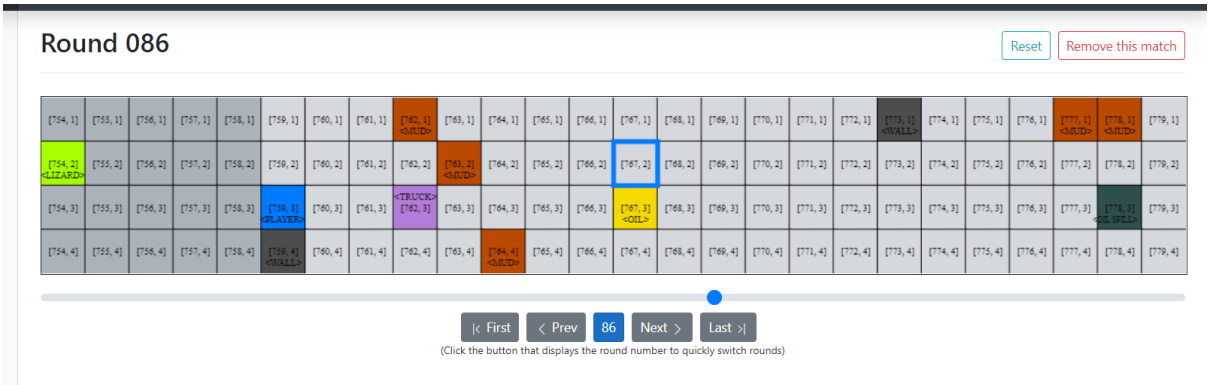


Gambar 4.3.4 tangkap layar *bot* mobil menggunakan *boost*

Pada gambar 4.3.3 bot memutuskan untuk melakukan pergerakan ke kiri (atas). Pergerakan ini efektif karena pergerakan ke depan (baik accelerate maupun nothing) akan menabrak *cyber truck.* Sedangkan pergerakan ke kanan akan mengenai *mud*. Selanjutnya, pada gambar 4.3.4 diperlihatkan keadaan saat bot menggunakan

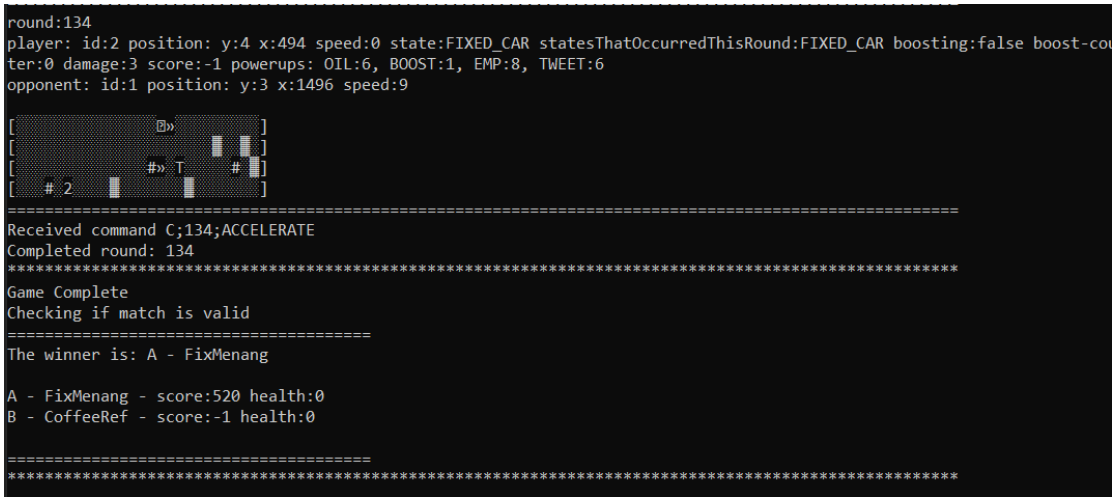
boostCommand. BoostCommand dirancang untuk digunakan apabila 15 block kedepan tidak memiliki *obstacles* yang dapat merusak mobil.

Gambar 4.3.5 tangkap layar *bot* mobil menghindari *oil spill*



Gambar 4.3.6 tangkap layar bot mobil menghindari *cybertruck*

Dua gambar di atas, yaitu gambar 4.3.5 dan 4.3.6 memilih untuk melakukan pergerakan ke kiri untuk menghindari lane terburuk. Pada gambar 4.3.6, mobil tetap terkena *mud*, tetapi pergerakan ke kiri atas merupakan gerakan terbaik karena hanya terkena satu obstacle, ditambah efek yang diberikan *mud* tidak seburuk *wall dan truck.*



Gambar 4.3.7 Hasil dari permainan

Gambar 4.3.7 menunjukkan end state dari permainan ini. Bot berhasil memenangkan permainan melawan *reference-bot* setelah 134 *round*.

BAB V
KESIMPULAN DAN SARAN

5.1. KESIMPULAN

Melalui Tugas Besar ini, kami berhasil merancang bot dalam permainan *Overdrive* dengan mengimplementasikan algoritma *greedy* ke dalam program bot tersebut. Strategi yang kami ambil ialah strategi *greedy by speed* diikuti dengan *greedy by poin* dengan lebih memprioritaskan kecepatan (*speed*) bot mobil diikuti dengan 'poin' yang terakumulasi berdasarkan hasil konvensi kami. Strategi *greedy by speed* mengacu pada bot mobil untuk mendapatkan kecepatan secepat-cepatnya sehingga selalu mendekati garis akhir dilanding bot mobil lawan. Pada kasus tertentu, strategi *greedy by poin* akan menjadi penunjang terhadap strategi *greedy by speed* sehingga bot mobil dapat memilah *track lane* yang paling memberikan keuntungan bagi bot sehingga tidak berpaku keras pada kecepatan mobil saja.

5.2. SARAN

Program bot *overdrive* masih dapat dikembangkan, terutama dalam penggunaan *power up* dan dalam penentuan *lane* secara optimal. Penggunaan *Power up* – terutama *power up* yang membutuhkan presisi tinggi seperti Tweet – dapat dikembangkan terhadap momen yang tepat untuk menggunakan *power up* tersebut sehingga bot lawan dapat diganggu dan , akhirnya, membalikkan keadaan permainan. Penentuan *lane* yang tepat akan memberikan keuntungan besar bagi bot sedemikian sehingga penggunaan *power up* yang bergantung pada kondisi *lane* – seperti Boost dan Oil – dapat dimaksimalkan. Keberadaan *power up* Oil dapat menjadi salah satu kunci utama dalam memenangkan permainan, sehingga perlu pengembangan lebih lanjut guna menggunakan Oil lebih baik.

BAB VI
DAFTAR PUSTAKA

1. Algoritma Greedy (Bagian 1)
   https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
2. Entelect Challenge 2020 – Overdrive
   https://github.com/EntelectChallenge/2020-Overdrive
3. Visualizer entelect – Overdrive
   https://entelect-replay.raezor.co.za/#