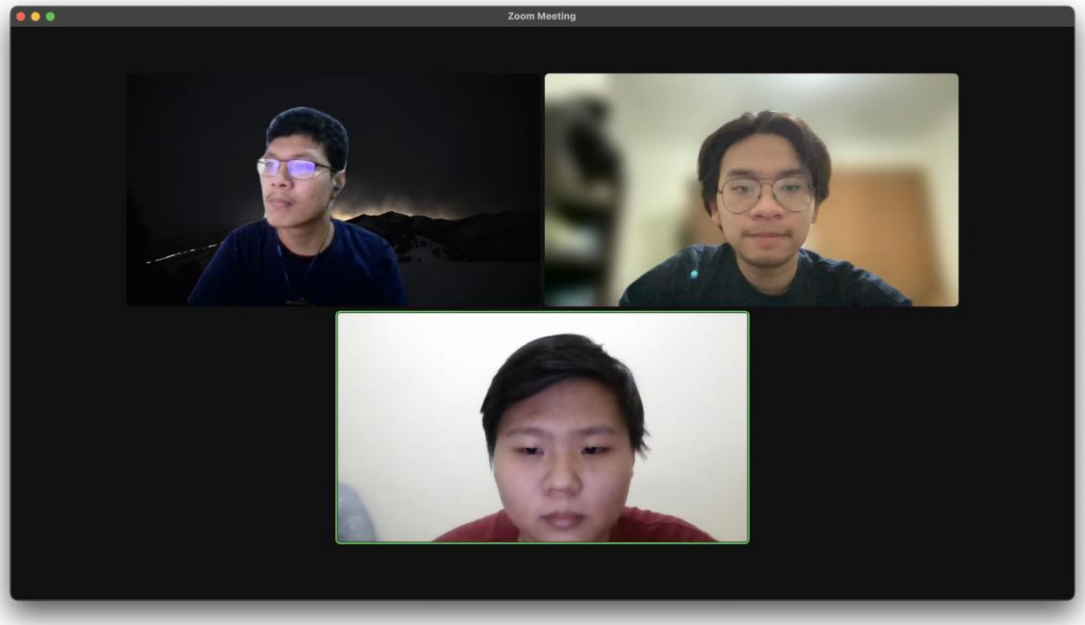
Tugas Besar 1 IF2211 Strategi Algoritma

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"

Semester II Tahun 2021/2022



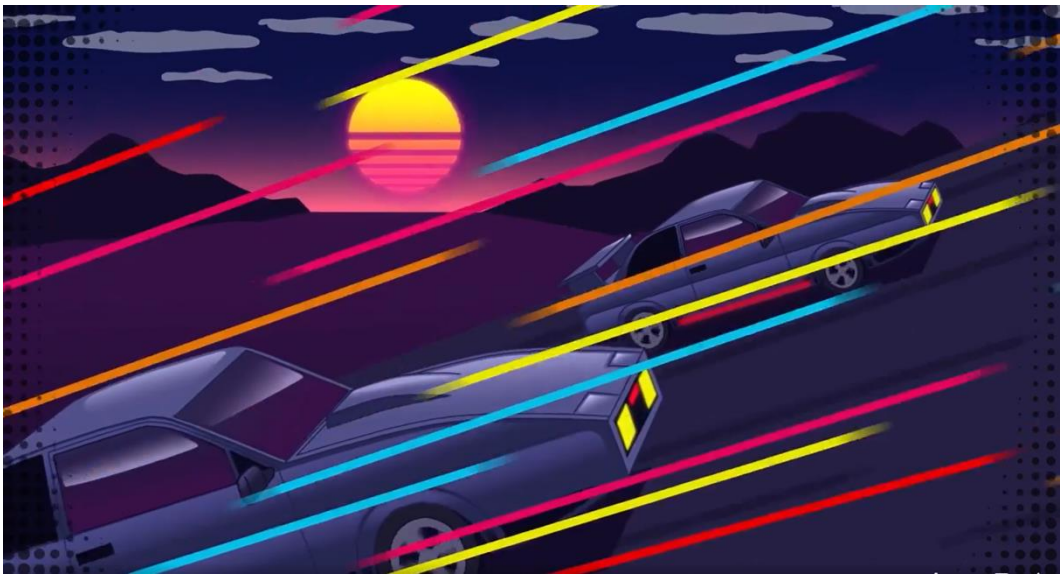
KELOMPOK sabeb_wkwk

13520099	Vincent Prasetiya Atmadja
13520107	Azka Syauqy Irsyad
13520138	Gerald Abraham Sianturi

BAB 1

DESKRIPSI TUGAS

Overdrive adalah sebuah *game* yang mempertandingan dua 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:

    1. *Oil item, dapat menumpahkan oli di bawah mobil anda berada.*
    2. *Boost, dapat mempercepat kecepatan mobil anda secara drastis.*
    3. *Lizard, berguna untuk menghindari lizard yang mengganggu jalan mobil anda.*
    4. *Tweet, dapat menjatuhkan truk di block spesifik yang anda inginkan.*
    5. *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:

    1. *NOTHING*
    2. *ACCELERATE*
    3. *DECELERATE*
    4. *TURN_LEFT*

5. *TURN_RIGHT*
6. *USE_BOOST*
7. *USE_OIL*
8. *USE_LIZARD*
9. *USE_TWEET <lane> <block>*
10. *USE_EMP*
11. *FIX*

5. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. Jika command tidak valid, bot mobil tidak akan melakukan apa-apa dan akan mendapatkan pengurangan skor.
6. Bot pemain yang pertama kali mencapai garis *finish* akan memenangkan pertandingan. Jika kedua bot mencapai garis *finish* secara bersamaan, bot yang akan memenangkan pertandingan adalah yang memiliki kecepatan tercepat, dan jika kecepatannya sama, bot yang memenangkan pertandingan adalah yang memiliki skor terbesar.

Adapun peraturan yang lebih lengkap dari permainan *Overdrive*, dapat dilihat pada laman:

https://github.com/EntelectChallenge/2020-Overdrive/blob/develop/game-engine/game-rules.md

BAB 2

LANDASAN TEORI

2.1. Algoritma *Greedy*

Algoritma *greedy* merupakan suatu algoritma yang paling popular dan sederhana untuk menyelesaikan persoalan untuk melakukan optimasi. *Greedy* sendiri memiliki arti tamak, rakus, atau serakah yang diambil dari bahasa Inggris. Prinsip dari algoritma ini adalah *take what you can get now*. Contoh masalah sehari-hari yang dapat menggunakan prinsip ini adalah seperti memilih jenis investasi, bermain kartu remi, dan mencari jalur tersingkat dari suatu daerah ke daerah lainnya.

Algoritma ini membentuk solusi langkah per langkah, dimana dengan begitu perlu dilakukan eksplorasi dan evaluasi terhadap keputusan yang akan diambil di tiap langkahnya agar terbentuk *output* yang paling maksimal serta keputussan yang telah diambil tidak dapat diubaj lagi pada langkah selanjutnya. Dengan begitu, pada setiap langkahnya kita memilih *local optimum* dengan harapan akan mencapai *global optimum* di akhir eksekusi program.

Ada beberapa elemen yang terdapat di dalam algoritma *greedy*, yaitu sebagai berikut.

1. Himpunan kandidat, *C*: berisi kandidat yang akan dipilih pada tiap langkah;
2. Himpunan solusi, *S*: berisi kandidat yang sudah dipilh;
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi;
4. Fungsi seleksi: memilih kandidat berdasarkan strategi *greedy* tertentu, dimana strategi ini bersifat heuristik;
5. Fungsi kelayakan: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (apakah layak atau tidak); serta
6. Fungsi objektif: memaksimumkan atau meminumkan.

Perlu diperhatikan juga, dalam pelaksanaan strategi *greedy*, optimum global yang didapat belum tentu merupakan solusi yang paling optimal. Bisa jadi ada solusi lain yang merupakan solusi yang paling optimalnya.

2.2. Cara Kerja Program

Untuk membangun program, pertama perlu melakukan *clone* terhadap repository Github yang telah disediakan. Pastikan juga kita mengikuti arahan, pedoman, maupun *requirements* yang dicantumkan dalam file readme program tersebut. Untuk melakukan run pada program, kita dapat langsung mengklik file run.bat yang tersedia atau pun dengan menggunakan *command* make run, bergantung pada system operasi yang digunakan.

Proses pengimplementasian bot yang akan dilakukan berlangsung di dalam folder *starter-bots*, lebih tepatnya di subfolder bagian /src/main/java/za. Pada folder tersebut, sudah ada file-file yang berisi *command* dasar yang bisa langsung kita manfaatkan dalam pembuatan algoritma *greedy* pada bot kita. Kita juga dapat menambah beberapa command tambahan yang sekiranya diperlukan dalam keberjalanan program yang akan kita implementasikan nantinya.

Pada folder ini, kami membuat beberapa file tambahan baru, yaitu file untuk melengkapi *command tweet* (TweetCommand.java) serta *lizard* (LizardCommand.java), file Value.java yang berisi kelas *value* guna memasukkan nilai ke dalam *command*, file Speed.java berupa kelas *Speed* yang berisi tentang deklarasi serta hal-hal yang berkaitan dengan kecepatan mobil, serta file Weight.java yang merupakan kelas dimana di dalamnya berlangsung pembobotan *command* yang dapat dilakukan di tiap rondenya yang berfungsi untuk menunjukan *command* yang memberikan nilai terbaik di *round* tersebut.

Dari semua *command*, baik yang sudah tersedia maupun yang dibuat manual, yang diperlukan dalam pengaplikasian algoritma *greedy* ini, hal-hal tersebut dipanggil dalam suatu file utama yang nantinya akan menjadi urutan pengambilan keputusan oleh bot yang kita kerjakan, berada di file Bot.java. Bot akan melakukan aksi sesuai dengan urutan dan hasil keluaran dari tiap-tiap *method* yang ada, dimana dalam melakukan aksinya didasari atas hasil pembobotan yang telah dilakukan pada ronde tersebut.

Source code pada folder ini kemudian akan dibuild menggunakan package pada lifecycle maven, Hasil dari package ini akan berupa 3 fil jar, salah satunya dengan dependencies. File jar inilah yang nantinya akan dimaikan dengan menggunakan starter pack

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1 *Mapping* Elemen Algoritma *Greedy*

1. Himpunan kandidat, berisi setiap *command* atau *action* yang dapat dilakukan oleh bot *car*. Isi dari himpunan tersebut dibatasi oleh ketentuan *game*, di antaranya, NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP, FIX
2. Himpunan solusi, yaitu daftar *command* yang terpilih berdasarkan kondisi tertentu.
3. Fungsi solusi, yang memeriksa apakah bobot *command* yang telah diambil merupakan solusi terbaik.
4. Fungsi seleksi, dimana memilih bobot *command* yang paling menguntungkan.
5. Fungsi kelayakan, memeriksa apakah *command* yang diambil sesuai dengan kondisi saat itu dan pemakaiannya dirasa efisien.
6. Fungsi objektif, dimana memilih *command* dengan nilai yang paling maksimal berdasarkan kondisi-kondisi yang ada.

3.2 Alternatif Pertimbangan Solusi

1. Strategi *Greedy by Avoiding Obstacle*

   Strategi ini merupakan pemilihan keputusan yang berdasarkan pada *lane* dengan mempertimbangkan jumlah poin *obstacle* yang paling kecil nilai total bobotnya. Banyaknya *block* yang diperiksa bergantung pada kecepatan yang ditempuh saat ini. Nilai dari tiap *obstacle* adalah sebagai berikut.

   - *Mud = 2*
   - *Oil = 2*
   - *Wall = 3*
   - *Truck = 6*

   Solusi ini akan efektif jika tujuan utama kita hanya pada kecepatan, tidak memedulikan *power up*, dan tidak terkena *power up* lawan seperti EMP. Namun kondisi ini membuat kita harus rela kehilangan kemungkinan mendapat *power up* yang dapat

didapat pada *lane* tersebut. Efisiensi dari strategi ini karena hanya menggunakan satu kali loop, yaitu loop ke depan sejauh kecepatan kita, maka hanya memerlukan notasi O(n).

2. Strategi *Greedy by Picking Power Up*

   Strategi ini merupakan pemilihan keputusan yang berdasarkan pada kemungkinan terambilnya *power up* pada *lane* yang bersangkutan, dan mengambil pada *lane* mana yang menghasilkan nilai tertinggi. Banyaknya *block* yang perlu di cek adalah sejauh kecepatan bot saat ini. Nilai dari *power up* adalah sebagai berikut.

   - *EMP = 7*
   - *Boost = 6*
   - *Lizard = 5*
   - *Tweet = 5*
   - *Oil = 3*

   Strategi ini efektif jika tujuan kita adalah memanfaatkan keuntungan dari adanya *power up* dan ingin mengumpulkan poin. Namun, kondisi ini tidak baik untuk kecepatan bot dan *damage* yang didapat akan melimpah. *State* utama menang dalam permainan ini juga tidak ter*cover* dengan strategi ini. Untuk efisiensi program, karena hanya butuh satu kali iterasi sebanyak n *step*, maka bernotasi O(n).

3. Strategi *Greedy by Lane*

   Strategi ini menggabungkan dua strategi sebelumnya. Dengan begitu, kita memperhitungkan segala hal yang ada di dalam *lane* yang memungkinkan, dan pada akhirnya mengambil bobot yang paling besar. Banyaknya *block* yang perlu dicek juga sama seperti sebelumnya, yaitu sebanyak kecepatan pada saat ini. Besar bobot dari tiap itemnya adalah sebagai berikut.

   - *EMP = 7*
   - *Boost = 6*
   - *Lizard = 5*
   - *Tweet = 5*
   - *Oil (Power Up) = 3*

- *Mud = -2*
- *Oil (Obstacle) = -2*
- *Wall = -3*
- *Truck = -6*

Kondisi ini efektif jika bot masih berada di belakang lawan, namun kurang menguntungkan jika berada di depan dan lawan memiliki *EMP*. Efisiensi dari strategi ini sama dengan dua strategi sebelumnya, yaitu sebesar O(n).

4. Strategi *Greedy by Command*

   Strategi ini dilakukan dengan mencari *command* yang berlaku dan memberikan dampak terbaik dalam ronde. Setiap *command* akan dicoba dan dihitung nilai efektifnya pada ronde tersebut dan akan dipilih jika di rasa paling menguntungkan. Kondisi-kondisi yang diperhitungkan sesuai dengan prioritas dari strategi ini adalah besar pergeseran bot dan pengaruh terhadap kondisi lawan. Pengecekan bergantung pada kecepatan yang berlaku berdasarkan *command* yang dilakukan.

   Strategi ini efektif jika mempunyai memilih memperhitungkan segala kemungkinan dari ronde yang ada, tidak berpaku pada satu tujuan saja. Namun, penentuan prioritasnya memungkinkan memberikan solusi yang kurang optimal karena prioritas ditentukan berdasarkan keinginan pembuat. Efisiensi dari strategi ini adalah dengan iterasinya yang dapat mencapai *state* kecepatan n+6 (kondisi *boosting*), dalam notasi big O tetap O(n).

5. Strategi *Greedy by Position*

   Strategi ini berfokus pada posisi dari lawan, baik itu dari *lane* maupun *block*. Dengan begitu, ketika posisi tertinggal, bot akan memprioritaskan agar menyusul lawan dan memanfaatkan *tweet* serta *emp* untuk memengaruhi kondisi lawan. Ketika di depan, bot akan berusaha menjauh sejauh mungkin dan mencari *oil* untuk memengaruhi kondisi lawan pula.

   Strategi ini efektif jika tujuan kita adalah untuk mengganggu keadaan lawan, baik ketika di depan maupun ketika masih di belakang. Namun, dengan begitu maka focus kita dalam pertandingan menjadi buyar karena yang menjadi patokan utama adalah posisi lawan. Efisiensi dari strategi ini juga sama dengan sebelumnya, yaitu bernotasi O(n).

6. Strategi

3.3. Mekanisme Strategi *Greedy* yang Dipilih

Dalam pengaplikasian strategi *greedy* ke dalam bot ini, pertama yang kami lakukan adalah me-*list* semua daftar *command* yang dapat digunakan selama permainan berlangsung. Setelah itu, kami membagi daftar *command* tersebut ke dalam dua bagian, yaitu efektif jika digunakan ketika posisi bot sedang *leading*, dan yang efektif digunakan jika posisi bot tertinggal. Daftar *command* yang dapat digunakan akan dimasukkan ke dalam suatu *array* yang tiap elemennya berisi nama *command* serta *value* yang merupakan nilai bobot *command* yang bersangkutan. Dari daftar *command* yang dapat digunakan tersebut, selanjutnya akan digunakan skema pembobotan terhadap efek pengambilan *command* tersebut. Pembobotan yang dilakukan dalam strategi kami adalah sebagai berikut.

1. Pergeseran maju mobil (pergeseran ke kanan)

   Dari setiap *command* yang berlaku, akan dilakukan perhitungan jumlah pergeseran maju yang akan dilakukan mobil terhadap *command* yang sedang dicek. Jumlah pergeseran mobil ini nantinya akan dikali dengan pembobotan untuk pergeseran mobil, yaitu sebesar 22. Dengan demikian, semakin besar nilai pergeseran yang akan dilalui mobil, maka bobot dari *command* tersebut juga akan semakin besar.
2. Perubahan speed mobil

   Dari setiap *command* yang berlaku, akan dilakukan perhitungan besarnya perubahan kecepatan di ronde tersebut berdasarkan *command* yang sedang dicek. Perubahan *speed* ini dimaksud ketika *speed* mengalami kenaikan ataupun mengalami penurunan diakibatkan terkena *obstacles*. Hasil dari perubahan *speed* ini nantinya akan diakumulasi serta dikalikan dengan pembobotan sebesar 25.
3. *Power up* yang dapat didapatkan

   Dari setiap *command* yang berlaku, akan dilakukan perhitungan ada *power up* apa saja yang dapat didapatkan jika kita menggunakan *command* yang sedang dicek. Pengecekan dilakukan di seluruh *block* yang akan dilalui ketika *command* yang sedang dicek dijalankan, dan tiap *power up* memiliki nilai yang berbeda-beda, dengan rincian *EMP* bernilai 7, *Boost* bernilai 6, *Tweet* dan *Lizard* bernilai 5, serta

Oil bernilai 3. Tiap *power up* yang didapatkan akan diakumulasikan poinnya, yang kemudian akan dikalikan dengan dengan pembobotan untuk *power up* sebesar 20.

4. Kondisi penggunaan *command* dan *power up* sebagai bonus point

   Dari tiap *command* yang berlaku, akan dilakukan perhitungan terhadap kondisi mengenai kapan waktu yang tepat untuk menggunakan *power up* ataupun *command*. Berikut penjelasan lebih lanjut mengenai pembobotan kondisi tersebut.

   a. *Nothing, Accelerate, Turn Right, Turn Left* selalu tidak menambahkan poin bobot.
   b. *Decelerate* akan memberikan nilai bobot -1000 jika pada saat yang bersamaan bot sedang menggunakan *Boost*.
   c. *Use Boost* akan memberikan nilai bobot 10 jika pada saat itu bot tidak menderita *damage* sama sekali, sedangkan jika menderita *damage* sebanyak satu maka akan memberikan nilai bobot sebesar 9. Jika kecepatan bot sudah maksimal, maka penggunaannya akan bernilai -100.
   d. *Use Lizard* akan memberikan nilai bobot sebesar 7 jika lane yang akan dilaluinya memiliki *obstacle* ataupun bot lawan dengan total akumulasi lebih dari 4.
   e. *Use Oil* akan menghitung jumlah *lane* yang terdapat *obstacle* serta menghitung banyak *obstacle*. Dari kedua hal tersebut, jika jumlah *lane* yang dapat dijangkau berjumlah 3 dan semuanya memiliki *obstacles* lebih dari 5, maka nilai bobotnya adalah 6 dan jika kurang maka bernilai 4; jika jumlah *lane* yang dapat dijangkau hanya kanan atau kiri dan jumlah *obstacles* lebih dari 3 maka nilai bobotnya adalah 4 dan sebaliknya adalah 2.
   f. *Use EMP* akan memberikan nilai bobot sebesar 9 jika l*ane* kita sama dengan *lane* bot lawan, dan bernilai 7 jika bot lawan ada di kiri atau kanan *lane* kita.

   Dari nilai pembobotan di atas, akan dikalikan lagi dengan nilai bobot kondisi ini sebesar 22.

Dari hasil pembobotan di atas, akan dicari nilai maksimum dari *value* tiap *command* yang ada dalam *array*, yang kemudian menjadi kandidat *command* yang akan diambil. Pada eksekusinya, daftar prioritas dari *command* yang dipilih dari tiap rondenya adalah sebagai berikut.

1. Jika posisi kita sudah berada di *block* 1485, maka:
    a. Jika kecepatan kita adalah 0, maka lakukan *fix*.
    b. Kondisi lainnya selalu melakukan *accelerate*.
2. Jika *damage* mobil kita lebih besar dari atau sama dengan 2, maka lakukan *fix*.
3. Kondisi lainnya mengambil dari *command* terbaik dari pembobotan yang telah dilakukan dalam array *command* yang berlaku. Jika *nothing*, maka akan menggunakan *tweet* jika terdapat *tweet power up* dan jika *accelerate* namun *speed* sudah maksimum juga akan mengeluarkan *tweet* jika ada.

Proses pengambilan keputusan tiap rondenya akan berlangsung seperti prioritas di atas hingga permainan mencapai akhir. Strategi ini diambil dengan mengambil konsep dari *greedy by command, position,* dan *lane* sesuai dengan penjelasan yang sudah dijelaskan pada bagian alternatif di atas. Alasan dari pengambilan strategi ini adalah agar tiap ronde yang dilalui, hasil *command* keluaran merupakan yang paling baik, setidaknya menurut preferensi kami. Kami juga memprioritaskan posisi ketercapaian di tiap rondenya agar mendapat hasil yang paling maksimal, sehingga dilakukan pembobotan seperti di atas dan menaruh *fix* di prioritas yang tinggi serta selalu dilakukan tiap mencapai *damage* sebesar 2 agar *maximum speed* selalu berkisar di atas 8. Dengan begitu, tiap round diharapkan selalu memberikan ketercapaian *move* ke depan yang besar dan memberikan solusi yang optimal.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Deskripsi singkat implementasi

Implementasi dikerjakan menggunakan bahasa java. Kakas lain yang turut digunakan, di antaranya <isi library>

4.2 *Pseudocode*

Dalam algoritma greedy kami, kami membaginya menjadi 2 bagian yang terletak pada Bot.java dan Weight.java. Pada Bot.java, algoritma berpusat pada setup untuk perhitungan dan return command berdasarkan algoritma. Pada Weight.java, algoritma berpusat pada pemberian nilai untuk setiap gerakan berdasarkan berbagai parameter kemudian akan mengembalikan gerakan dengan nilai tertinggi.

Pembuatan Pseudocode ini juga akan kami bagi menjadi 2 bagian, untuk Bot.java dan Weight.java. Akan tetapi, untuk menghemat ruang dan keefektifan membaca, hanya fungsi-fungsi utama yang akan kami masukkan. Selengkapnya dapat dilihat di source code kami.

Bot.java

Beberapa variable kami sudah kami inisiasi melalui fungsi OOP yaitu konstruktor. Variabel-variabel tersebut adalah myCar, opponent, speed, dan gameState

```
function run() -> Command
Deklarasi
      myCar, opponent : Car
      gameState : GameState
      speed : Speed
      weightList : list of Value
      available : list of <list of Value>
      tobetested : Weight
      bestCommand : String
      block : integer
Body
      //Pengecekan apabila sudah dekat dengan garis finish
      if(myCar.position.block >= 1485) then
            if(myCar.speed = 0) then
                  ⟶ FixCommand()
            else
                  → AccelerateCommand()
            endif
      endif

      //Kalau mobil berdamage sekurang-kurangnya 2, maka mobil akan
diperbaiki
      if(myCar.damage >= 2) then
            → FixCommand()
      endif
      //Setup beberapa variabel
      available ← getAvailableBlock(myCar.position.block,
mycar.powerups, gameState)
      if(isLeading()) then
            WeightList ← createWeightList(1)
      else
            WeighList ← createWeightList(2)
      endif
      tobetested ← Weight(WeightList)
      bestCommand ← tobetestesd.bestCommand(myCar, opponent, available,
gameState.lanes.get(0)[0].position.block, speed)

      depend on (bestCommand)
            case "Nothing" :
                  if(checkPowerUps(Powerups.TWEET, myCar.powerups)) then
                        block ← getBestBlock(opponent)
                        → TweetCommand(opponent.position.lane, block)
                  endif
                  → TweetCommand(opponent.position.lane, block)
```

```
                  → DoNothingCommand()
            case "Accelerate":
                  if(checkPowerUps(Powerups.TWEET, myCar.powerups) and
(myCar.speed = speed.getMaxSpeed() or myCar.speed = 9 ) then
                        block ← getBestBlock(opponent)
                        → TweetCommand(opponent.position.lane, block)
                  endif
                  → AccelerateCommand()
            case "Decelerate":
                  → DecelerateCommand()
            case "Turn_Right":
                  → ChangeLaneCommand(1)
            case "Turn_Left":
                  → ChangeLaneCommand(-1)
            case "Use_Boost"):
                  → BoostCommand()
            case "Use_Lizard":
                  → LizardCommand()
            case "Use_Oil":
                  → OilCommand()
            case "Use_EMP":
                  → EmpCommand()
            default:
                  → AccelerateCommand()

function getBestBlock(opponent : Car) → Integer
//Fungsi untuk mencari block terbaik untuk fungsi Tweet
Deklarasi
      multiply, bestblock : integer
Body
      multiply ← 15
      bestblock ← opponent.possiton.block + 2 * multiply;

      if(bestblock >= 1500) then
            bestblock ← 1499.
```

```
      endif
      → bestblock

function isLeading() -> boolean
Deklarasi
Body
      if(myCar.position.block > opponent.position.block)then
            → true
      → false
      endif


function checkTurnValid(direction : integer, lane : int) → Boolean
Deklarasi
Body
      if(direction = -1 and lane = 1) then
            → false
      else if(direction = 1 and lane = 4){
            → false
      else
            → true
      endif
```

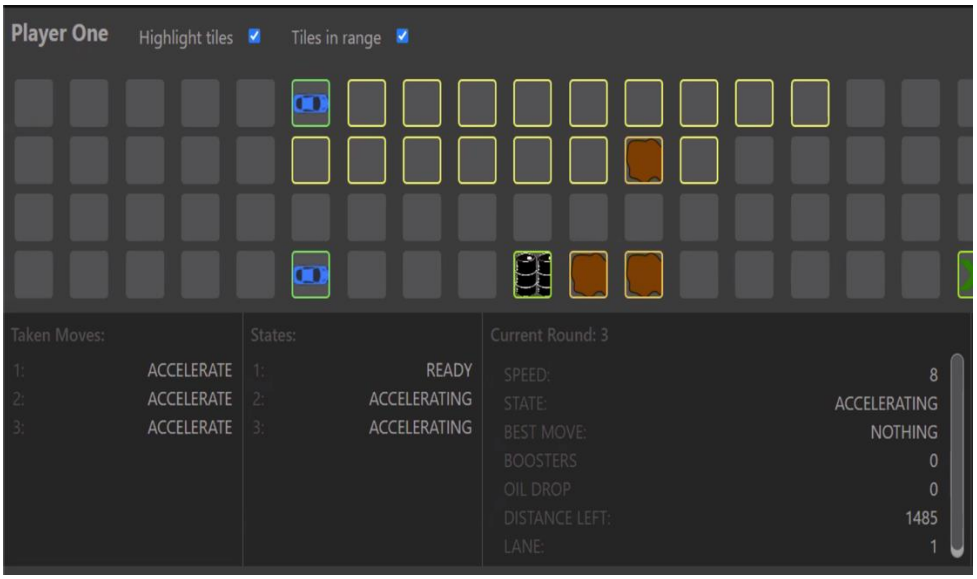
```
procedure shiftColumn(speed, myCar, available)
Deklarasi:
      speed: Speed
      myCar: Car
      available: Matrix of Lane
      temp: String
      AllComand: Array of Value
      currentSpeed: Integer
      ShiftColumn: Integer { bobot parameter shiftColumn }
Body:
i traversal [0..AllCommand.size]
      temp ← getCommand()
      depend on (temp):
            case "Nothing", "Use_Lizard", "Use_Oil", "Use_EMP":
                  addValue(currentSpeed * ShiftColumn)
            case "Accelerate":
                  addValue(currentSpeed.getAccelerate() * ShiftColumn)
            case "Decelerate":
                  addValue(currentSpeed.getDecelerate() * ShiftColumn)
            case "Turn_Right":
                  addValue((currentSpeed - 1) * ShiftColumn)
            case "Turn_Left":
                  addValue((currentSpeed - 1) * ShiftColumn)
            case "Use_Boost":
                  addValue(currentSpeed.getBoost() * ShiftColumn)
end traversal
procedure speedChange(speed, myCar, available)
Deklarasi:
      speed: Speed
      myCar: Car
      indexLane, speedTempStraight, speedTempRight, speedTempBoost,
speedTempAccelerate, speedTempDecelerate, currentSpeed, speedChange:
Integer
      available: Matrix of Lane
Body:
      indexLane ← myCar.position.lane - 1
      currentSpeed ← speed.getCurrentSpeed()
      depend on (temp):
            case "Nothing", "Use_Oil", "Use_EMP":
                  addValue((speedTempStraight - currentSpeed) *
speedChange)
            case "Accelerate":
                  addValue((speedTempAccelerate - currentSpeed)*
speedChange)
            case "Decelerate":
                  addValue((speedTempDecelerate - currentSpeed)*
speedChange)
            case "Turn_Right":
```

```
                  addValue((speedTempRight – currentSpeed)* speedChange)
            case "Turn_Left":
                  addValue((speedTempLeft – currentSpeed) * speedChange)
            b "Use_Boost":
                  addValue((speedTempBoost – currentSpeed)* speedChange)
            case "Use_Lizard":
                  if terrain after jump is Wall then
                        addValue((3 – currentSpeed) * speedChange)
                  else if terrain after jump is Mud or Oil_spill
                        addValue                addValue((speedTempRight
– currentSpeed)* speedChange)
                  else then
                        addValue((speedTempStraight – currentSpeed) *
speedChange)
                  endif

procedure powerUp(myCar, available, speed)
Deklarasi:
      speed: Speed
      myCar: Car
      available: Matrix of Lane
      AllCommand: Array of Lane
      countStraight, countRight, countLeft, countAcc, countDec,
countBoost, countLizard, powerUp: Integer
Body:
      i traversal [0 .. AllCommand.size]
            count ← 0
            depend on (AllCommand.get(i).getCommand())
                  case "Nothing", "Use_Oil", "Use_EMP":
                        addValue(countStraight * PowerUp)
                  case "Turn_Right":
                        addValue(countRight * PowerUp)
                  case "Turn_Left":
                        addValue(countLeft * PowerUp)
                  case "Accelerate":
                        addValue(countAcc * PowerUp)
                  case "Decelerate"
                        addValue(countDec * PowerUp)
                  case "Use_Boost":
                        addValue(countBoost * PowerUp)
                  case "Use_Lizard":
                        addValue(countLizard * PowerUp)
      end traversal
```

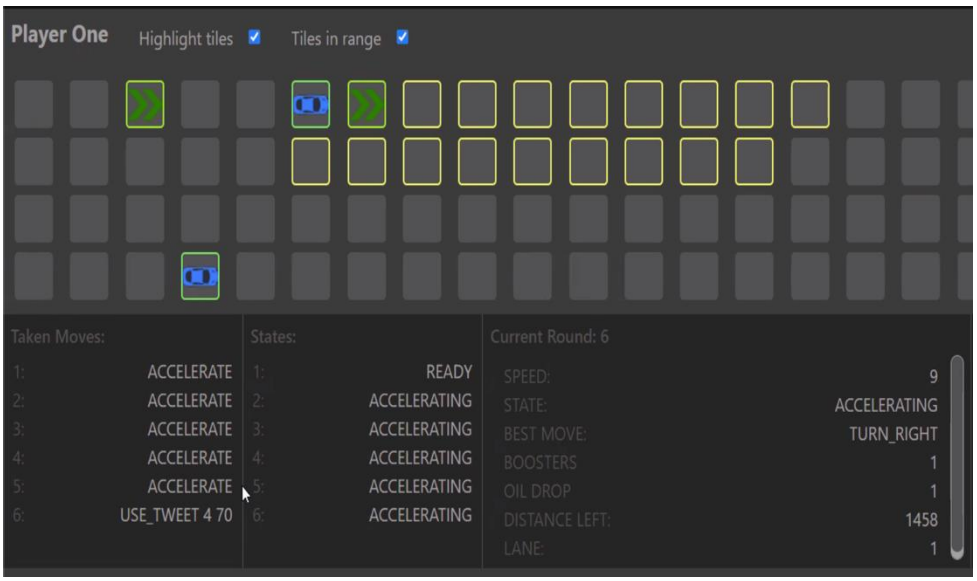
```
procedure bonusPoint(input myCar : Car, input opponent : Car, input
available : List of <List of Lane>, input speed : Speed, output
AllCommand : List of Value)

Deklarasi
      start, many, lookOpp, loc, numLane: integer
      temp : String

Body
start ← startAll
i traversal [0..AllCommand.size]
      temp ← AllCommand.get(i).getCommand()
      depend on (temp):
            case "Nothing":
                  AllCommand.get(i).addValue(0)
            case "Accelerate":
                  AllCommand.get(i).addValue(0)
            case "Decelerate":
                  if (temp is "Use_Boost")then
                        AllCommand.get(i).addValue(BonusScore * (-1000))
            case "Turn_Right":
                  AllCommand.get(i).addValue(0)
            case "Turn_Left":
                  AllCommand.get(i).addValue(0)
            case "Use_Boost":
                  if myCar.speed is speed.getMaxSpeed() then
                        AllCommand.get(i).AddValue(BonusScore * (-1000)
                  else then
                        if myCar.damage is 0 then
                              AllCommand.get(i).addValue(BonusScore *
(10)
                        if myCar.damage is 1 then
                              AllCommand.get(i).addValue(BonusScore * 9)
            case "Use_Lizard":
                  many ← 0
                  i traversal [start .. myCar.position.lane-1.size()
                        if terrain is Mud or Oil_Spill or Wall or
Opponent Car
                        many ← many + 1
                  loc ← available.get(myCar.position.lane-1.size()
                  if(many >= 4 and terrain is Empty or Oil_Power or
Boost or Lizard or EMP or Tweet
                        AllCommand.get(i).addValue(BonusScore * 7)
                  end traversal
            case "Use_Oil":
                  if myCar.position.lane-1 is not 0 then
                        i traversal [0 .. myCar.position.lane-1.size()]
                              if terrain is Mud or Oil_Spill or Wall
                                    obstacle ← obstacle + 1
                        end traversal
                        numLane ← numLane + 2
```

```
if myCar.position.lane + 1 is not 5
      i traversal [0 .. myCar.position.lane-1.size()
            if terrain is Mud or Oil_Spill or Wall
                  obstacle ← obstacle + 1
            endif
      end traversal
      numLane ← numLane + 2
i traversal [0 .. myCar.position.lane-1.size()]
      if terrain is Mud or Oil_Spill or Wall
            obstacle ← obstacle + 1
      endif
end traversal
if numLane is 4 then
      if obstacle > 5
            AllCommand.get(i).addValue(BonusScore * 6)
      else
            AllCommand.get(i).addValue(BonusScore * 4)
      u
endif
if numLane is 2 then
      if obstacle > 3
            AllCommand.get(i).addValue(BonusScore * 4)
      else
            AllCommand.get(i).addValue(BonusScore * 2)
else
      AllCommand.get(i).addValue(0)
```

4.3 Pengujian



Pada kondisi ini, karena baru memulai permainan dan belum ada *power up*, maka berdasarkan strategi yang telah diaplikasikan akan memilih *command accelerate*



Pada kondisi ini, karena sudah mencapai kecepatan maksimum pada kecepatan biasa, karena ada *power up tweet* maka akan digunakan

Pada kondisi ini, karena sudah dipakai *boost*, maka selanjutnya akan digunakan *tweet*.

4.4 Struktur data yang digunakan

Dalam program ini, kami menggunakan beberapa tipe data, baik yang kami buat sendiri maupun telah disediakan oleh pihak *Entellect*. Tipe data tersebut dapat dikelompokkan sebagai berikut

A. Command

Tipe data Comman adalah tipe data yang disediakan pihak *Entellect* untuk mengirimkan Command. Command akan dikirimkan dengan memanfaatkan salah satu sifat OOP, yaitu *override*. Method yang akan di*override* adalah method render yang terdapat di Command.java. Secara keseluruhan, ada 11 file pada Command yaitu

1. AccelerateCommand.java
2. BoostCommand.java

3. ChangeLaneCommand.java
4. Command.java
5. DoNothingCommand.java
6. EmpCommand.java
7. FixCommand.java
8. OilCommand.java
9. TweetCommand.java

B. Entities

Entities adalah tipe data yang juga disediakan oleh pihak Entelect dan berguna sebagai entitas dalam game. Tipe data ini digunakan untuk mendapatkan informasi dari game. Entities sendiri dibangun oleh tipe data Enum yang menyimpan hal-hal dasar. Ada 4 tipe data Enum yaitu

1. Direction.java
2. PowerUps.java
3. State.java
4. Terrain.java

Tipe data Entities ada 4 dan akan dijelaskan di bawah ini beserta dengan data yang ada di dalamnya

1. Car.java

Terdiri dari

a. id : integer
b. position: Position
c. speed: id
d. state: State
e. damage: integer
f. powerups : **List of** Powerups
g. boosting: Boolean

2. GameState.java

Terdiri dari

a. currentRound : integer
b. maxRound : integer
c. player : Car
d. opponent : Car
e. slanes : **List of** Lane

3. Lane.java

Terdiri dari

a. position: Position
b. terrain : Terrain
c. occupiedByPlayerId: int
d. Position.java

5. Position.java

Terdiri dari
a. lane : integer
b. block : integer

C. Tipe Data Lainnya

Selain tipe data di atas, kelompok kami juga membuat tipe data lain untuk mempermudah perhitungan. Tipe data yang kami buat memanfaatkan beberap konsep OOP di Java dan selengkapnya dapat dilihat pada source code. Pada laporan ini, kami hanya akan memasukkan atribut yang terdapat pada tipe data dan manfaat tipe data tersebut

1. Speed.java

   Tipe data ini berguna untuk mendapatkan state kecepatan maksimal, kecepatan mobil Sekarang, kecepatan setelah akselerasi dan setelah deselerasi disesuaikan dengan damage mobil. Atribut yang ada di dalamnya adalah sebagai berikut.

   a. damage : integer
   b. currentSpeed : integer
   c. maxSpeed : integer

2. Value.java

   Tipe data ini berguna untuk memberikan nilai kepada setiap Command, yang diwakili oleh sebuah string, dan sebuah nilai. Tipe data ini sebenarnya adalah penerapan dari tipe data Dictionary atau Pair pada C++. Atribut yang ada di dalamnya adalah sebagai berikut.

a. command : String
b. score : integer

BAB 5

KESIMPULAN DAN SARAN

5.1 KESIMPULAN

Berdasarkan hasil strategi yang telah kelompok kamu rumuskan, notasi big O untuk pengeksekusian pencarian dengan *greedy* kurang lebih sama antara satu dengan yang lainnya. Sebetulnya, masih ada banyak strategi *greedy* lain yang lebih optimal dari strategi yang kami buat. Hasil *output* dari strategi yang kami buat juga belum tentu merupakan hasil yang paling optimal.

5.2 SARAN

Untuk kedepannya mungkin jangka waktu pengerjaan dapat lebih diperpanjang, agar proses eksplorasi dapat terjadi lebih luas. Selain itu, tingkat kerumitan pembuatan laporan dapat jauh dikurangi karena cukup membuat kewalahan, terutama di bagian pseudocodenya.

DAFTAR REFERENSI

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

[2] https://github.com/EntelectChallenge/2020-Overdrive

Link-Github: https://github.com/geraldabrhm/Stima01