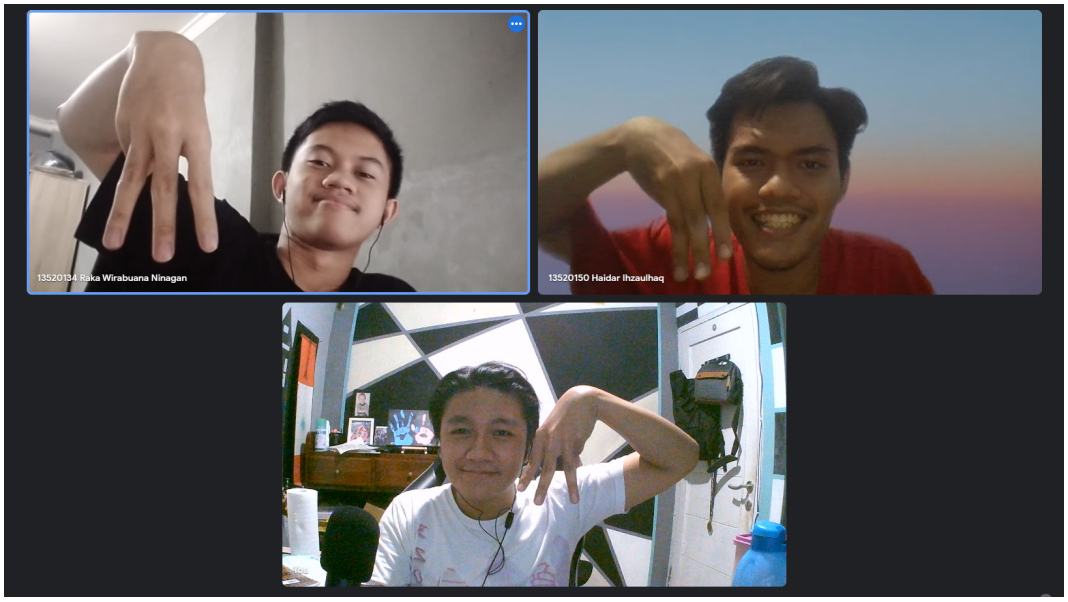
LAPORAN TUGAS BESAR I

Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Overdrive”

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2021/2022



Disusun Oleh :

Ilham Bintang Nurmansyah	13520102
Raka Wirabuana Ninagan	13520134
Haidar Ihzaulhaq	13520150

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

Bab 1

Deskripsi Tugas

Overdrive adalah sebuah game yang mempertandingan 2 bot mobil dalam sebuah ajang balapan. Setiap pemain akan memiliki sebuah bot mobil dan masing-masing bot akan saling bertanding untuk mencapai garis finish dan memenangkan pertandingan. Agar dapat memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu untuk dapat mengalahkan lawannya.

Tugas mahasiswa adalah mengimplementasikan bot mobil dalam permainan Overdrive dengan menggunakan strategi *greedy* untuk memenangkan permainan. 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.

Bab 2

Landasan Teori

2.1 Algoritma Greedy

Algoritma greedy adalah algoritma yang memecahkan sebuah persoalan secara langkah per langkah sedemikian sehingga pada setiap langkah, dilakukan pengambilan pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhitungkan apa yang akan terjadi kedepannya dan diharapkan bahwa dengan memilih pilihan terbaik di setiap langkahnya (optimum lokal) akan memberikan hasil terbaik (optimum global). Sederhananya, pilihan yang diambil di setiap kesempatan pengambilan pilihan selalu yang terbaik di kesempatan tersebut. Cara ini bisa menghasilkan hasil terbaik secara keseluruhan, tetapi ada kemungkinan juga salah (karena tidak diperhitungkan secara menyeluruh). Meskipun bukan hasil terbaik yang sebenarnya, solusi penggunaan algoritma ini akan mendekati hasil terbaik.

2.2 Game Overdrive

Spesifikasi permainan yang digunakan sesuai dengan spesifikasi yang disediakan oleh game engine Overdrive. Ada beberapa aturan umum yang juga menjadi cara kerja *game* ini, yaitu:

1. Peta permainan berbentuk array 2 dimensi yang memiliki 4 jalur lurus dengan setiap jalur dibentuk oleh block dengan panjang peta 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 power ups yang tersedia adalah:
    a. Oil item, dapat menumpahkan oli di bawah mobil kita berada.
    b. Boost, dapat mempercepat kecepatan mobil anda secara drastis.
    c. Lizard, berguna untuk menghindari *obstacle* yang mengganggu jalannya mobil.
    d. Tweet, dapat menjatuhkan truk di block spesifik yang diinginkan.
    e. EMP, dapat menembakkan EMP ke depan jalur dari mobil kita 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 dapat berubah sesuai tingkatan, seperti berikut:
    a. MINIMUM_SPEED = 0
    b. SPEED_STATE_1 = 3
    c. INITIAL_SPEED = 5
    d. SPEED_STATE_2 = 6
    e. SPEED_STATE_3 = 8
    f. MAXIMUM_SPEED = 9
    g. BOOST_SPEED = 15

Kecepatan n berarti bot mobil akan bergerak sebesar n *block* di setiap *round*-nya.

4. Game state akan memberikan jarak pandang hingga 20 block di depan dan 5 block di belakang bot.
5. Terdapat beberapa command yang pada setiap round dapat digunakan oleh masing-masing pemain. Berikut jenis-jenis command yang ada pada permainan:
    a. NOTHING = tidak melakukan apa-apa.
    b. ACCELERATE = meningkatkan kecepatan bot.
    c. DECELERATE = mengurangi kecepatan bot.
    d. TURN_LEFT = belok kiri.
    e. TURN_RIGHT = belok kanan.
    f. USE_BOOST = menggunakan boost jika memilikinya.
    g. USE_OIL = menggunakan *oil* jika memilikinya.
    h. USE_LIZARD = menggunakan *lizard* jika memilikinya.
    i. USE_TWEET = melemparkan *truck* ke posisi yang diinginkan jika memilikinya.
    j. USE_EMP = menggunakan EMP jika memilikinya.
    k. FIX = mengurangi damage bot sebesar 2.
6. 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.
7. 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.

2.3 Run Game Engine Overdrive

Untuk menjalankan bot pada permainan Overdrive ini, ada beberapa langkah yang dapat dilakukan, yaitu:

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, dibutuhkan beberapa *requirement* dasar sebagai berikut.
    a. Java (minimal Java 8)
    b. IntelIiJ IDEA
    c. NodeJS
3. Untuk menjalankan permainan, dapat dibuka 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, edit file “game-runner-config.json”. Informasi tentang bot yang dibuat juga dapat diubah pada file “bot.json” dalam direktori “starter-bots”

5. Untuk penampilan yang lebih menarik, dapat digunakan visualizer berikut https://github.com/Affuta/overdrive-round-runner

2.4 Implementasi Bot

Implementasi bot dapat dilakukan dengan cara berikut:

1. Buka folder starter-bots yang ada pada folder game tersebut.
2. Pilih bahasa yang ingin digunakan, dalam hal laporan ini, bahasa yang akan digunakan adalah bahasa Java.
3. Pada bahasa Java, buka Folder src dengan menggunakan IntelIiJ IDEA dan cari file bernama Main.java dan Bot.java
4. Implementasi bot dapat dilakukan pada file Bot.java atau dapat membuat file baru pada folder tersebut.
5. Setelah implementasi selesai, maka folder java dapat dibuat jar.
6. File .jar itulah yang nantinya akan diproses oleh game engine yang ada (run.bat).

Bab 3

Aplikasi Strategi *Greedy*

3.1 Elemen Algoritma Greedy

- Himpunan Kandidat : Perintah tindakan yang dilakukan mobil pemain dengan representasi
    a. NOTHING
    b. ACCELERATE
    c. DECELERATE
    d. TURN_LEFT
    e. TURN_RIGHT
    f. USE_BOOST
    g. USE_OIL
    h. USE_TWEET lane block
    i. USE_LIZARD
- Himpunan Solusi : *command-command* yang terpilih
- Fungsi Solusi : Memeriksa apakah mobil sampai ke garis finish lebih cepat
- Fungsi Seleksi : Memilih *command* yang mengarahkan mobil untuk bergerak lebih cepat
- Fungsi Kelayakan : Memeriksa apakah jalur yang dipilih sesuai dengan orientasi dari algoritma yang dipilih
- Fungsi Objektif : Round yang dilalui oleh mobil pemain minimum dan menghasilkan kemenangan

3.2 Eksplorasi Alternatif Solusi Greedy

- **Greedy 1 Lane** : Algoritma yang mengandalkan *command* ACCELERATE, USE_BOOST, USE_LIZARD, dan FIX di satu jalur (hajar terus pantang mundur)
- **Greedy High HP** : Algoritma yang mengutamakan tidak mengenai *obstacle* sama sekali dengan memanfaatkan DECELERATE sehingga *obstacle* lebih mungkin dihindari
- **Greedy Speedrun** : Algoritma yang mengutamakan *damage* ke mobil pemain seminimum mungkin dengan cara melalui jalur tanpa *obstacle* atau jalur dengan *obstacle* yang sangat minimum.
- **Greedy PowerUps Winner** : Algoritma greedy yang memanfaatkan semua powerups yang dimiliki dengan tujuan mempercepat mobil diri sendiri dan memperlambat mobil lawan.

3.3 Analisis Efisiensi dan Efektivitas tiap Alternatif Solusi Greedy

Asumsi bahwa analisis efisiensi berdasarkan algoritma yang dijalankan dalam satu ronde.

Alternatif Solusi Greedy	Efisiensi	Efektivitas
Greedy 1 Lane	Best Case & Worst Case : O(1) Alasannya adalah algoritma greedy ini hanya fokus pada dirinya sendiri. Ketika mobil rusak, mobil melakukan FIX. Ketika mobil aman-aman saja, lakukan ACCELERATE atau BOOST jika punya *powerups* BOOST.	Dengan fokus di satu jalur, algoritma ini akan mengoptimalkan pergerakan mobil untuk selalu maju. Tujuan utamanya adalah agar mobil tidak perlu melakukan pergerakan ke kanan ataupun ke kiri, sehingga dalam kondisi kosong, jarak jangkau selalu yang terjauh. Kekurangan dari algoritma ini adalah ketika jalur mobil tersebut memiliki banyak sekali *obstacle* sehingga penggunaan FIX semakin sering dilakukan. Secara logika, penggunaan FIX berkali-kali menunjukkan bahwa mobil tersebut terlalu sering melakukan pemberhentian, sehingga menghambat tujuan utama dari balapan.
Greedy High HP	- Best Case : O(1) Ketika tidak ada *obstacle* apapun di jalur mobilnya berada, sehingga melakukan ACCELERATE. - Worst Case : O(E) Semua jalan yang memungkinkan terdapat *obstacle*. E untuk banyaknya jalan yang memungkinkan.	Algoritma ini berfokus untuk menghindari damage semaksimal mungkin, yaitu dengan memanfaatkan DECELERATE (tetap memanfaatkan ACCELERATE ketika maju). Kelebihan dari algoritma ini adalah DECELERATE memberikan kesempatan bagi mobil untuk menghindari *obstacle* dengan lebih baik karena kecepatannya yang lebih lambat ketika mendekati *obstacle*. Kekurangan dari algoritma ini adalah terlalu banyak angka kecepatan yang terbuang akibat DECELERATE sehingga laju memiliki rata-rata yang lebih lambat.
Greedy Speedrun	- Best Case : O(1) Ketika tidak ada *obstacle* apapun di jalur mobilnya berada, sehingga melakukan ACCELERATE atau melakukan BOOST. - Worst Case : O(E) Semua jalan yang memungkinkan terdapat	Orientasi utama dari algoritma ini adalah meminimalisasi *damage* tanpa memperlambat gerakan mobil. *Command-command* yang digunakan hampir semuanya berkarakter menggerakan mobil maju. Akibatnya, penggunaan powerups yang berkarakter pengganggu lawan pun diminimalisasi. Kelebihan dari algoritma ini adalah setiap kesempatan diutamakan kepada gerakan untuk maju sembari mengutamakan tujuan meminimasi damage, sehingga jarak jangkau lebih jauh dibandingkan algoritma lainnya.

	obstacle. E untuk banyaknya jalan yang memungkinkan.	Kekurangan dari algoritma ini adalah kurangnya gangguan kepada lawan sehingga bisa saja kekalahan terjadi bukan karena kalah cepat (secara normal), tetapi karena diganggu oleh lawan.
Greedy PowerUps Winner	- Best Case : O(1) Ketika lawan memenuhi kondisi untuk diserang dan powerups yang dibutuhkan memenuhi / jalur dimana mobil berada tidak terdapat *obstacle* - Worst Case : O(E) Semua jalan yang memungkinkan terdapat *obstacle*. E untuk banyaknya jalan yang memungkinkan.	Sesuai namanya, greedy ini memanfaatkan semua powerups yang dimiliki dengan tujuan mempercepat mobil diri sendiri dan memperlambat mobil lawan. Pergerakan dari mobil sendiri berfokus pada menambah powerups dan memanfaatkannya untuk meraih garis finish secepat mungkin dan mencegah lawan mencapai garis finish lebih dulu.

Berdasarkan uraian dan analisis di atas, tim kami memutuskan untuk memilih algoritma greedy speedrun dengan alasan kekurangan yang hanya disebabkan oleh tidak terukurnya gangguan lawan sehingga mobil pemain sebenarnya berada mendekati kondisi paling idealnya.

Bab 4

Implementasi dan Pengujian

4.1 Pseudocode

```
function getBlocksInFront(lane : integer, block : integer, gameState : GameState, speed : integer)
-> List
    map : List<Lane[]>
    map = gameState.lanes
    blocks : List<Object>
    startBlock : integer
        startBlock = map[0][0].position.block

    laneList : List
        laneList = map[lane - 1]
    for i <- max(block-startBlock,0) to (block-startBlock+speed)
      if (laneList[i] == null || laneList[i].terrain == Terrain.FINISH) then
        break
      endif
      blocks.add(laneList[i].terrain);
    endfor
    return blocks
```

```
function cntObstacleInFront(currBlock : List) -> int
    count : integer
    count = 0
    for i <- 0 to currBlock.size()
      if ((currBlock.get(i) == Terrain.MUD) ||
          (currBlock.get(i) == Terrain.WALL) ||
          (currBlock.get(i) == Terrain.OIL_SPILL))
        count += 1
      endif
    endfor
    return count
```

```
function lurus(block: List, myCar:Car Object, opponentCar:Objct Car):
        Object List currBlock = blocks;
        int maxSpeed = 9

        if mycar.damage == 0 then
                maxSpeed=15
```

```
        else if mycar.damage == 1 then
                maxSpeed = 9
        else if mycar.damage == 2 then
                maxSpeed = 7
        else if mycar.damage == 3 then
                maxSpeed = 6
        else if mycar.damage == 4 then
                maxSpeed = 3
        else if mycar.damage == 5 then
                maxSpeed = 0
        endif

        if !currBlock.contains(Terrain.MUD) && !currBlock.contains(Terrain.WALL) &&
           !currBlock.contains(Terrain.OIL_SPILL) then
                if mycar.speed < maxSpeed && !hasPowerUp(PowerUps.BOOST,
                   mycar.powerups) then
                        return ACCELERATE
                else
                        if hasPowerUp(PowerUps.BOOST, mycar.powerups) then
                                return BOOST;
                        if mycar.speed >= maxSpeed then
                                if hasPowerUp(PowerUps.EMP, mycar.powerups) then
                               if opponentCar.position.lane == mycar.position.lane &&
                               opponentCar.position.block > mycar.position.block then
                                 return EMP
                                if hasPowerUp(PowerUps.TWEET, mycar.powerups)
                        return new TweetCommand(opponentCar.position.lane,
opponentCar.position.block + 1)
                                if hasPowerUp(PowerUps.OIL, mycar.powerups) then
                               return OIL
                                else
                                        return ACCELERATE
                            endif
                        else
                                return ACCELERATE
                    endif
                endif
        else
                if hasPowerUp(PowerUps.LIZARD, mycar.powerups) then
                        return LIZARD
                else
                        return DO_NOTHING
```

```
            endif
        endif
```

```
function canRightandLeft(leftBlock : List, rightBlock : List, currentLane : integer) -> Command
        boolean isTurnLeft    =    !leftBlock.contains(Terrain.MUD) &&
                                !leftBlock.contains(Terrain.WALL) &&
                                    !leftBlock.contains(Terrain.OIL_SPILL)
        boolean isTurnRight   = !rightBlock.contains(Terrain.MUD) &&
                                !rightBlock.contains(Terrain.WALL) &&
                                !rightBlock.contains(Terrain.OIL_SPILL)
        if (isTurnLeft && isTurnRight) then
       // Kiri dan kanan bisa. Apabila di line index 1, ambil TURN_RIGHT, kalo line index 2
        Ambil TURN_RIGHT
            if (currentLane == 2) then
                return TURN_RIGHT
            endif
            else
             return TURN_LEFT
              endif
        endif
        else
              if (isTurnLeft == true && isTurnRight == false) then
             // Gabagus ke kanan
                    return TURN_LEFT
            endif
              else if (isTurnLeft == false && isTurnRight == true) then
               // Gabagus ke kiri
                     return TURN_RIGHT
            endif
            else
           // dua-duanya false
           // Return do nothing biar tar di luar aja dibandingin kualitas keputusannya sm
            yang lurus
                     return DO_NOTHING
            endif
    endif
```

```
function OnlyRight(rightBlock : List) -> Command
// Logic nya mirip seperti canRightAndLeft
        boolean isTurnRight = !rightBlock.contains(Terrain.MUD) &&
                              !rightBlock.contains(Terrain.WALL) &&
```

```
                                !rightBlock.contains(Terrain.OIL_SPILL)
    if (isTurnRight) then
      return TURN_RIGHT
    endif
    else
      return DO_NOTHING
    endif
```

```
function OnlyLeft(leftBlock : List) -> Command
// Logic nya mirip seperti CekRightAndLeft
    List<Object> leftBlock = blocksKiri
    boolean isTurnLeft   =      !leftBlock.contains(Terrain.MUD) &&
                                !leftBlock.contains(Terrain.WALL) &&
                                !leftBlock.contains(Terrain.OIL_SPILL)
    if (isTurnLeft) then
      return TURN_LEFT
    endif
    else
      return DO_NOTHING
    endif
```

```
function run(gameState : GameState) -> Command
    myCar : Car
    opponent : Car
    blocksLurus, blocksKanan, blocksKiri : List<Object>
        currentLane, jmlObstacleKiri, jmlObstacleKanan, jmlObstacleLurus : integer
        option1, option2 : Command
        opponent = gameState.opponent
        myCar = gameState.player
      jmlObstacleKiri = 999
      jmlObstacleKanan = 999
      jmlObstacleLurus = 999
      blocksLurus = getBlocksInFront(myCar.position.lane, myCar.position.block,
                    gameState, myCar.speed)
      option1 = lurus(blocksLurus, myCar, opponent)
      jmlObstacleLurus = cntObstacleInFront(blocksLurus)
     currentLane = myCar.position.lane

    if (currentLane == 1) then
      blocksKanan = getBlocksInFront(myCar.position.lane + 1, myCar.position.block,
                    gameState, myCar.speed)
      jmlObstacleKanan = cntObstacleInFront(blocksKanan)
```

```
    option2 = OnlyRight(blocksKanan)
endif
else if (currentLane == 4) then
    blocksKiri = getBlocksInFront(myCar.position.lane - 1, myCar.position.block,
                gameState, myCar.speed)
    jmlObstacleKiri = cntObstacleInFront(blocksKiri)
    option2 = OnlyLeft(blocksKiri)
endif
else
    blocksKanan = getBlocksInFront(myCar.position.lane + 1, myCar.position.block,
                    gameState, myCar.speed);
    blocksKiri = getBlocksInFront(myCar.position.lane - 1, myCar.position.block,
                gameState, myCar.speed);
    jmlObstacleKanan = cntObstacleInFront(blocksKanan);
    jmlObstacleKiri = cntObstacleInFront(blocksKiri);
    option2 = canRightAndLeft(blocksKiri, blocksKanan, currentLane);
endif
     //Masuk perbandingan opsi Command yang akan dipilih
if (myCar.damage >= 3) then
    return FIX
endif
if (myCar.speed == 0) then
    return ACCELERATE
endif
if (option1 != DO_NOTHING) then
    // Tidak ada Obstacle di lane lurus
    return option1
endif
else if (option2 != DO_NOTHING) then
    // Ada obstacle di lane lurus tetapi kanan / kiri kosong
    return option2
endif
else
    // Ada obstacle di lurus, kanan, dan kiri
    //Bandingin jumlah obstacle kanan, kiri, dan tengah
    prioritasLane : integer
        prioritasLane = min(jmlObstacleKanan,min(jmlObstacleKiri,jmlObstacleLurus))
    if (prioritasLane == jmlObstacleLurus) then
        return ACCELERATE
    endif
    else
        if (jmlObstacleKanan == jmlObstacleKiri) then
```

```
            if (currentLane == 2) then
              return TURN_RIGHT
            endif
            else
              return TURN_LEFT
            endif
          endif
          else if (prioritasLane == jmlObstacleKanan) then
            return TURN_RIGHT
          endif
          else
            return TURN_LEFT
          endif
        endif
      endif
```

```
procedure main(args: Array String):

    Scanner sc = new Scanner(System.in);
    Gson gson = new Gson();
    Bot bot = new Bot();
        while true :
       try :
          int roundNumber = sc.nextInt()
          String statePath = String.format("./%s/%d/%s", ROUNDS_DIRECTORY, roundNumber,
STATE_FILE_NAME)

          String state = new String(Files.readAllBytes(Paths.get(statePath)))

          GameState gameState = gson.fromJson(state, GameState.class)

          Command command = bot.run(gameState)


          System.out.println(String.format("C;%d;%s", roundNumber, command.render()))
        catch (Exception e) :
          e.printStackTrace()
```

4.2 Struktur Data

Dalam program ini kami menggunakan beberapa struktur dalam untuk membantu algoritma. Struktur data yang kami gunakan adalah sebagai berikut:

1. Objek Bot
2. Objek Car
3. Objek Command
4. Objek GameState
5. List Objek Block

Objek bot digunakan untuk membuat *instance* dari sebuah kelas Bot. Kelas Bot memiliki atribut berupa instan dari kelas Car dan Command, dan juga memiliki atribut bertipe primitif seperti integer. Objek ini dipanggil pada Main.java sebagai bagian dari game engine. Objek Car digunakan untuk mendapatkan data data mengenai bot mobil kita seperti *speed, damage,* dan *ability.* Objek Command digunakan untuk memberikan bot kita command yang akan di eksekusi di *round* sekarang. Beberapa contoh Command seperti ACCELERATE, LIZARD, BOOST , dan FIX. Objek GameState digunakan untuk mengetahui state atau keadaan di *round* sekarang. State yang bisa diketahui seperti nomor ronde, keadaan bot mobil musuh, keadaan lanes atau jalur ,dan keadaan bot mobil kita. List Objek Block digunakan untuk mengetahui isi dari block yang ada pada jalur didepan kita. List ini berguna sekali untuk menentukan algoritma *greedy* yang digunakan. List Objek Block bisa berisi *obstacle* atau *ability*

4.3 Analisis Algoritma *Greedy*

Solusi algoritma *greedy* yang kelompok kami implementasikan pada permainan kali ini cukup optimal pada beberapa kasus. Algoritma kami mampu mendeteksi ada berapa banyak obstacle di setiap *lane* yang mungkin untuk di jelajahi, dan mampu membuat keputusan terbaik berdasarkan jumlah obstacle yang ada pada setiap *lane*. Misalnya pada lane 1 terdapat *n obstacle* sedangkan pada *lane* 0 ataupun 2 terdapat jumlah *obstacle* < *n*. Maka bot kami akan berpindah pada *lane* yang memiliki jumlah *obstacle* lebih sedikit. Lalu bot kami juga memprioritaskan kecepatan, sehingga bila mobil memiliki *damage* > 3 maka bot kami akan langsung memperbaikinya agar memiliki *max speed* yang optimal. Lalu bila bot kami memiliki *ability boost* , maka ia akan menggunakannya bila tidak ada obstacle yang terlihat sejauh jarak kecepatan mobil. Namun bot kami tidak optimal pada kasus kasus tertentu yang mungkin memiliki best move DECELERATE, karena kami tidak mengimplementasikan *command* tersebut di dalam bot kami. Terakhir, bila sudah mencapai kecepatan maksimal dan tidak ada yang bisa dilakukan lagi selain melakukan command NOTHING, algoritma kami akan memulai aksinya menggunakan *ability* yang bersifat menyerang seperti *oil, tweet,* dan *EMP.* Dalam penggunaan *ability* ini pun, kami menggunakan prioritas terhadap *ability* yang akan digunakan. Ability *EMP* mendapat prioritas paling tinggi karena persentase untuk mengenai lawan cukup besar dibanding *ability* lain. Lalu yang kedua adalah *ability tweet* karena jika dibandingkan dengan *oil*, *ability tweet* memiliki kemungkinan untuk mengganggu lawan lebih besar, karena

bisa disimpan dimana saja oleh kita, sedangkan *oil* berada di prioritas terakhir karena hanya bisa disimpan di belakang kita dan juga bisa dideteksi oleh musuh.

Bab 5

Kesimpulan dan Saran

5.1 Kesimpulan

Pada tugas besar kali ini kami dibuat berpikir untuk menciptakan algoritma greedy terbaik yang bisa diimplementasikan pada permainan Overdrive. Tujuan utama dari algoritma greedy kami adalah mencapai finish secepat-cepatnya, bukan menyerang musuh sebanyak-banyaknya. Langkah yang diambil oleh program kami setiap ronde nya berusaha untuk menghasilkan optimum lokal yaitu mengurangi jarak dengan finish sebesar besarnya. Hasilnya algoritma kami bisa mencapai finish ketika melawan bot referensi dengan kecepatan rata rata 8.5 setiap match nya, atau sekitar 176 round setiap match nya (hasil kecepatan ini berubah ubah bergantung pada kondisi dari arena balap). Hasil ini bisa dibilang mendekati maksimum karena bila algoritma sangat baik, maka setiap round bisa diselesaikan dengan kecepatan antara 9 hingga 15, atau sekitar 150 round setiap match.

5.2 Saran

Pahami dulu gambaran besar dari hal yang ingin dibuat algoritma greedynya, dalam konteks tugas ini adalah gambaran besar dari game Overdrive, karena tanpa pemahaman utama dari permainannya itu sendiri, akan sulit mengimplementasikan ide kasar yang terpikirkan. Selain itu, akan lebih baik jika algoritma greedy yang dibuat memiliki pembanding yang sepadan sehingga lebih paham dimana harus melakukan perbaikan dan bagaimana cara perbaikannya.

5.3 Link Repository dan Video Demo

Link Repository : https://github.com/Hambinn/Tubes1_Stima_Mengresing.git
Link Video Demo : https://www.youtube.com/watch?v=5XwxZjjPXl0

Daftar Pustaka

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

https://github.com/EntelectChallenge/2020-Overdrive/blob/master/game-engine/game-rules.md#the-game

https://www.jetbrains.com/help/idea/compiling-applications.html#package_into_jar

https://stackoverflow.com/questions/30204884/creating-a-jar-from-a-maven-project-in-intellij/30210471

https://github.com/Affuta/overdrive-round-runner

https://drive.google.com/file/d/1q-Jd_EhGubuQFlbevyC_UCcGkF_Y4ruV/view