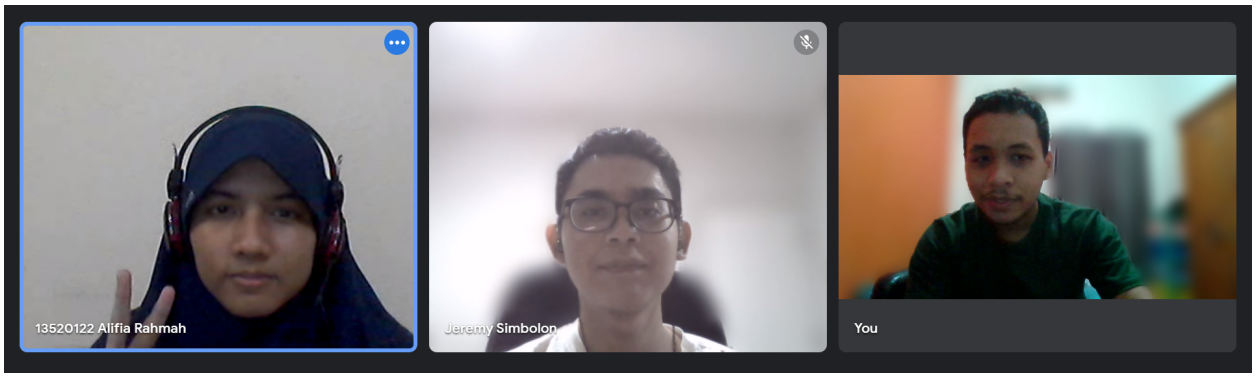
Laporan Tugas Besar 1
Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "Overdrive"
IF2211 Strategi Algoritma



Disusun Oleh:
Kelompok 26 - にげろう

Jeremy S.O.N. Simbolon	13520042
Fawwaz Anugrah Wiradhika Dharmasatya	13520086
Alifia Rahmah	13520122

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021/2022

DAFTAR ISI

BAB I: Deskripsi Tugas

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

Gambar 1.1 Ilustrasi permainan Overdrive

Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine yang mengimplementasikan permainan Overdrive. Game engine dapat diperoleh pada laman berikut:

https://github.com/EntelectChallenge/2020-Overdrive

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

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

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

1. Peta permainan memiliki bentuk array 2 dimensi yang memiliki 4 jalur lurus. Setiap jalur dibentuk oleh block yang saling berurutan, panjang peta terdiri atas 1500 block. Terdapat 5 tipe block, yaitu Empty, Mud, Oil Spill, Flimsy Wall, dan Finish Line yang masing-masing karakteristik dan efek berbeda. Block dapat memuat powerups yang bisa diambil oleh mobil yang melewati block tersebut.
2. Beberapa powerups yang tersedia adalah:
   a. Oil item, dapat menumpahkan oli di bawah mobil anda berada.
   b. Boost, dapat mempercepat kecepatan mobil anda secara drastis.
   c. Lizard, berguna untuk menghindari lizard yang mengganggu jalan mobil anda.
   d. Tweet, dapat menjatuhkan truk di block spesifik yang anda inginkan.
   e. EMP, dapat menembakkan EMP ke depan jalur dari mobil anda dan membuat mobil musuh (jika sedang dalam 1 lane yang sama) akan terus berada di lane yang sama sampai akhir pertandingan. Kecepatan mobil musuh juga dikurangi 3.
3. Bot mobil akan memiliki kecepatan awal sebesar 5 dan akan maju sebanyak 5 block untuk setiap round. Game state akan memberikan jarak pandang hingga 20 block di depan dan 5 block di belakang bot sehingga setiap bot dapat mengetahui kondisi peta permainan pada jarak pandang tersebut.
4. Terdapat command yang memungkinkan bot mobil untuk mengubah jalur, mempercepat, memperlambat, serta menggunakan powerups. Pada setiap round, masing-masing pemain dapat memberikan satu buah command untuk mobil mereka. Berikut jenis-jenis command ang ada pada permainan:
   a. NOTHING
   b. ACCELERATE
   c. DECELERATE
   d. TURN_LEFT
   e. TURN_RIGHT
   f. USE_BOOST
   g. USE_OIL
   h. USE_LIZARD
   i. USE_TWEET <lane> <block>
   j. USE_EMP
   k. FIX

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

Pada tugas besar kali ini, anda 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

Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang populer dan sederhana untuk memecahkan persoalan untuk mencari solusi optimal dalam suatu permasalahan. Terdapat dua macam persoalan optimasi, yaitu untuk maksimasi dan minimasi. Cara kerja algoritma *greedy* adalah dengan memecahkan persoalan secara langkah per langkah sedemikian rupa sehingga terambil pilihan yang terbaik yang dapat diperoleh pada setiap langkah, dengan harapan agar didapatkan solusi terbaik untuk keseluruhan proses.

Cara Kerja Program

Bot

Pada tugas besar ini, bot mensimulasikan sebuah mobil yang nantinya dapat dilombakan dengan bot lain dalam suatu pertandingan. Bot dapat dibuat dalam berbagai bahasa, namun pada tugas besar ini bot dibuat dalam bahasa Java. Bot mobil ini memiliki beberapa perintah untuk menambah kecepatan, mengurangi kecepatan, memperbaiki mobil jika terkena *damage*, dan menggunakan *powerup*. Bot memenangkan pertandingan jika mencapai blok *finish* duluan. Bot memiliki kapasitas damage tertentu. Jika Bot mendapat terlalu banyak damage, Bot tidak bisa bergerak, sehingga harus diperbaiki saat itu juga.
Bot dapat mengambil *powerup* yang tersedia pada lintasan dan menggunakannya. Beberapa powerups yang tersedia adalah:

a. *Oil item*, dapat menumpahkan oli di bawah mobil berada.
b. *Boost*, dapat mempercepat kecepatan mobil secara drastis.
c. *Lizard*, berguna untuk menghindari lizard yang mengganggu jalan mobil.
d. *Tweet*, dapat menjatuhkan truk di block spesifik.
e. *EMP*, dapat menembakkan EMP ke depan jalur dari mobil 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.

Implementasi Algoritma

Terdapat berbagai macam implementasi algoritma *greedy* pada Bot, tergantung apa yang ingin dioptimasi pada pertandingan. Terdapat beberapa hal yang dapat dioptimasi, di antaranya maksimasi kecepatan, maksimasi pemanfaatan powerup, dan meminimasi damage yang didapat.

Game Engine

Game engine untuk Overdrive 2020 dibuat dalam bahasa Java. Binary dari game engine ini terdapat pada file game-engine.jar dan game-runner-jar-with-dependencies.jar. Konfigurasi untuk pertandingan terdapat pada game-runner-config.json.

BAB III: Aplikasi Strategi *Greedy*

Mapping Persoalan *Overdrive* ke Dalam Komponen Algoritma *Greedy*

Untuk membantu kamu dalam melakukan formulasi algoritma, kami mendekomposisi permainan *Overdrive* ke dalam komponen-komponen umum algoritma *greedy* sebagai berikut:

- **Himpunan Kandidat**

  Berupa *command* yang dapat dipilih pemain pada setiap *round*. Mencakup *NOTHING, ACCELERATE, DECELERATE, TURN_LEFT, TURN_RIGHT, USE_BOOST, USE_OIL, USE_LIZARD, USE_TWEET, USE_EMP,* dan *FIX*.

- **Himpunan Solusi**

  Berupa himpunan *command* yang terpilih dari himpunan kandidat dengan anggota sebanyak jumlah *round* yang dijalankan.

- **Fungsi Solusi**

  Berupa fungsi yang memeriksa apakah mobil pemain telah mencapai *FINISH*.

- **Fungsi Seleksi**

  Berupa fungsi yang digunakan untuk menentukan *command* yang dipilih dari himpunan kandidat pada setiap *round*. Natur dari fungsi ini bergantung erat terhadap pendekatan strategi yang dipilih.

- **Fungsi Kelayakan**

  Berupa fungsi yang digunakan untuk memeriksa kelayakan kandidat *command* sebelum dimasukkan ke dalam himpunan solusi. Secara umum, fungsi ini mencakup logika pencegah tabrakan (*collision avoidance logic*) dan syarat kepemilikan power-up yang hendak digunakan. Natur spesifik dari fungsi ini bergantung terhadap pendekatan strategi yang dipilih.

- **Fungsi Objektif**

  Berupa tujuan yang hendak ditempuh oleh algoritma *greedy*. Algoritma yang akan diajukan bertujuan untuk menempuh jarak (*blocks*) sejauh-jauhnya hingga mencapai *FINISH*. Apabila pemain dan lawan melewati *FINISH* di waktu bersamaan, algoritma yang akan diajukan bertujuan untuk memaksimalkan kecepatan mobil ketika melewati *FINISH*. Apabila pemain dan lawan memiliki kecepatan akhir yang sama, algoritma yang

akan diajukan bertujuan untuk memperoleh skor yang maksimal. Detail lebih spesifik dari natur fungsi objektif bergantung terhadap pendekatan strategi yang dipilih.

Alternatif Solusi

Dalam proses eksplorasi strategi, kami mendalami lebih jauh empat alternatif strategi sebagai berikut.

1. Algoritma *greedy speed*
   Algoritma *greedy speed* memprioritaskan kecepatan semaksimal mungkin pada fungsi seleksinya. Algoritma ini akan selalu melakukan akselerasi bila kecepatan mobil di bawah 4. Fungsi kelayakan pada algoritma ini menekankan pada prospek penstabilan kecepatan terutama dalam menghindari tabrakan dan mengincar lane yang memiliki *powerup boost*. Fungsi objektif dari strategi ini bertujuan mencapai kecepatan setinggi-tingginya dan berusaha mempertahankan kecepatan tersebut dengan menghindari rintangan sebisa mungkin.

2. Algoritma *greedy boost*
   Algoritma *greedy boost* memprioritaskan pengambilan *boost* pada fungsi seleksinya. Algoritma akan memilih jalur (*lane*) yang memiliki prospek power-up *boost* ketika perlu melakukan perpindahan jalur. Fungsi kelayakan pada algoritma ini menekankan prospek *boost* ketika menjalankan *collision avoidance logic* serta menekankan penggunaan *boost* oleh mobil bila memungkinkan. Fungsi objektif dari strategi ini bertujuan untuk menggunakan *boost* sebanyak mungkin dengan harapan mencapai *FINISH* lebih cepat dari lawan.

3. Algoritma *greedy offensive*
   Algoritma *greedy offensive* memprioritaskan pengambilan *power-up* yang bersifat intrusif terhadap lawan pada fungsi seleksinya. *Power-up* yang dimaksud mencakup *OIL, TWEET,* dan *EMP*. Algoritma ini akan memilih jalur (*lane*) yang memiliki prospek *power-up* intrusif lebih tinggi ketika melakukan perpindahan jalur. Fungsi kelayakan pada algoritma ini menekankan prospek adanya *power-up* intrusif ketika menjalankan *collision avoidance logic* serta menekankan penggunaan *power-up* intrusif di waktu yang tepat bila memungkinkan. Fungsi objektif dari strategi ini bertujuan untuk mendisrupsi lawan sebanyak-banyaknya sehingga lawan tidak dapat menuju *FINISH* secara efektif.

4. Algoritma *greedy conservative*

Algoritma *greedy conservative* memprioritaskan menghindari kerusakan karena kerusakan berpengaruh terhadap kecepatan maksimum mobil. Algoritma ini memiliki fungsi sendiri dalam menentukan *lane* mana yang menjadi tujuan belok. Fungsi kelayakan dalam strategi ini adalah semua command yang membantu menghindari kerusakan seminimal mungkin sambil berusaha menaikkan kecepatan bila keadaan aman. Fungsi objektif dari strategi ini adalah memprioritaskan menghindari collision semaksimal mungkin, atau menggunakan *powerup lizard* jika memiliki *powerup* tersebut bila tidak ada *lane* yang aman, lalu memperbaiki kendaraan bila terjadi kerusakan. Bila *lane* di depan aman, maka mobil akan memfokuskan untuk melakukan akselerasi dan *boosting* bila memiliki *powerup boost*.

Analisis Efisiensi Alternatif Solusi

Berikut analisis efisiensi dari keempat alternatif solusi yang kami eksplorasi sebelumnya.

1. Algoritma *greedy speed*
   Dalam kondisi *best case*, algoritma *greedy speed* memiliki efisiensi O(s), dengan s merupakan kecepatan awal mobil, yang terjadi ketika mobil mengalami kerusakan lebih dari 3 atau mobil memiliki kecepatan kurang dari 4. Kompleksitasi O(s0 ini terjadi karena bot akan megumpulkan data jenis terrain sejauh kecepatan mobil ke depan sebelum menentukan *command* apa yang akan dilakukan. Pada kondisi *worst case*, algoritma ini memiliki efisiensi O(s+p) dengan p merupakan jumlah *powerup* di inventori. Kondisi ini terjadi jika keadaan di depan lancar dan mobil tidak memiliki powerup *boost* yang bisa digunakan.

2. Algoritma *greedy boost*
   Dalam kondisi *best case*, algoritma *greedy boost* memiliki efisiensi $O(1)$ yang terjadi ketika mobil hendak menggunakan *power-up* yang terletak di posisi pertama dari *array power-up* milik mobil dan posisi mobil terletak satu *block* dari *FINISH*. Dalam kondisi *worst case*, algoritma *greedy boost* memiliki efisiensi $O(n)$ yang terjadi ketika mobil tidak menemukan *power-up* yang hendak digunakan atau posisi mobil terletak di posisi selain blok sebelum *FINISH*.

3. Algoritma *greedy offensive*
   Dalam kondisi *best case*, algoritma *greedy offensive* memiliki efisiensi $O(1)$ yang terjadi ketika mobil hendak menggunakan *power-up* yang terletak di posisi pertama dari *array power-up* milik mobil dan posisi mobil terletak satu *block* dari *FINISH*.

Dalam kondisi *worst case*, algoritma *greedy offensive* memiliki efisiensi $O(n)$ yang terjadi ketika mobil tidak menemukan *power-up* yang hendak digunakan atau posisi mobil terletak di posisi selain blok sebelum *FINISH*.

4. Algoritma *greedy conservative*
   Dalam kondisi *best case*, algoritma *greedy conservative* memiliki efisiensi O(s), dengan s merupakan kecepatan awal mobil, yang terjadi ketika mobil mengalami kerusakan lebih dari 4 atau mobil memiliki kecepatan 0. Dalam kondisi *worst case*, algoritma ini memiliki efisiensi O(s+p) dengan p merupakan jumlah *powerup* di inventori. Kondisi ini terjadi ketika keadaan di depan benar benar aman, kecepatan mobil di atas 0, kerusakan mobil 0, serta mobil tidak memiliki *powerup boost*.

Analisis Efektivitas Alternatif Solusi

Setelah melakukan eksperimen menggunakan keempat alternatif solusi di atas, berikut kami paparkan hasil temuan kami.

1. Algoritma *greedy speed*
   Algoritma *greedy speed* dapat digunakan untuk memenangkan pertandingan dengan bantuan perintah *accelerate* dan *boost*, tetapi cenderung kurang optimal dalam menggunakan *boost* sehingga masih kalah dengan algoritma yang banyak memanfaatkan power-up seperti *boost*.

2. Algoritma *greedy boost*
   Algoritma *greedy boost* efektif untuk menempuh *block* terbanyak di setiap *round*. Tetapi, algoritma ini bersifat pasif dalam menginisiasi dan merespons serangan. Akibatnya, algoritma ini cenderung menjadi target serangan lawan tanpa berusaha menginterupsi mobil lawan

3. Algoritma *greedy offensive*
   Algoritma *greedy offensive* efektif untuk menghambat progres lawan, terutama ketika mobil berada dalam kondisi tertinggal. Lawan pun sibuk mempertahankan posisinya ketika mobil algoritma ini dapat fokus melaju dan menghindari rintangan jalan. Algoritma ini pun cenderung memenangkan pertandingan dari percobaan kami.

4. Algoritma *greedy conservative*

Algoritma *greedy conservative* cukup efektif dalam menghindari *collision* dan serangan lawan kecuali untuk emp dan tweet serta cukup efektif dalam mempertahankan kecepatan. Namun algoritma ini kurang efektif dalam melawan musuh yang memfokuskan dalam mencapai kecepatan semaksimal mungkin serta karena pada algoritma ini mobil cukup memprioritaskan memperbaiki kendaraan meski hanya 1 damage, algoritma ini membuat mobil memiliki fase fase stagnan untuk memperbaiki yang relatif sering sehingga peluang untuk dibalap musuh pun menjadi lebih besar.

Strategi *Greedy* yang Dipilih

Berdasarkan analisis yang telah kami paparkan di atas, kami memutuskan untuk memilih strategi *greedy offensive* karena pendekatannya efektif dalam mengatasi strategi lawan.

BAB IV: Implementasi dan Pengujian

Implementasi Algoritma *Greedy*

Berikut *pseudocode* dari potongan algoritma *greedy offensive* yang kami pilih.

```
define public method run() as
    myLane = my car's current lane
    myBlock = my car's current block
    opponentLane = my opponent's current lane
    opponentBlock = my opponent's current block

    blocks = call getBlocksInFront(myLane, myBlock)

    // Take note of possible turn(s) the car can make
    if myLane equals 1 do
        add left turn as not possible
        add right turn as possible
    else if myLane equals 4 do
        add left turn as possible
        add right turn as not possible
    else do
        add left turn as possible
        add right turn as possible

    // Fix logic
    if my car's damage is more than 2 and my car isn't boosting do
        return command FIX

    // Restart logic
    if my car's speed equals 0 do
        return command ACCELERATE

    // Collision avoidance logic
    if blocks in front contains MUD or OIL_SPILL or WALL do
        create ArrayList leftBlocks
        create ArrayList rightBlocks

        if my car can turn left do
            leftBlocks = call getBlocksInFront(myLane - 1, myBlock)

        if my car can turn right do
            rightBlocks = call getBlocksInFront(myLane + 1, myBlock)


        if my car can turn left and can turn right do
            if (leftBlocks contains MUD or OIL_SPILL or WALL) and (rightBlocks contains MUD or
OIL_SPILL or WALL) and my car has LIZARD power-up do
                return command LIZARD
            else if leftBlocks contains MUD or OIL_SPILL or WALL do
```

```
                return command TURN_RIGHT
            else if rightBlocks contains MUD or OIL_SPILL or WALL do
                return command TURN_LEFT
            else if leftBlocks contains BOOST or OIL_POWER or EMP or TWEET or LIZARD do
                return command TURN_LEFT
            else if leftBlocks contains BOOST or OIL_POWER or EMP or TWEET or LIZARD do
                return command TURN_RIGHT
            else do
                return random command from directionList
        else if my car can turn left do
            if (leftBlocks contains MUD or OIL_SPILL or WALL) and my car has LIZARD power-up do
                return command LIZARD
            else do
                return command TURN_LEFT
        else if my car can turn right do
            if (rightBlocks contains MUD or OIL_SPILL or WALL) and my car has LIZARD power-up do
                return command LIZARD
            else do
                return command TURN_RIGHT

        // Accelerate logic
        if my car's speed is less than 5 do
            return command ACCELERATE

        // Finish boost logic
        if blocks contains FINISH and my car has BOOST power-up do
            return command BOOST

        // EMP logic
        if my car has EMP power-up and my car is behind my opponent's and my opponent is within 1
lane of my car do
            return command EMP

        // Tweet logic
        if my car has TWEET power-up and my car's speed is at least the maximum speed and blocks
doesn't contain FINISH do
            depend on my opponent's car speed do
                if 0 : return command TWEET at opponentLane, 4 blocks ahead of opponentBlock
                if 3 : return command TWEET at opponentLane, 7 blocks ahead of opponentBlock
                if 6 : return command TWEET at opponentLane, 9 blocks ahead of opponentBlock
                if 8 : continue below
                if 9 : return command TWEET at opponentLane, 10 blocks ahead of opponentBlock
                if 15 : return command TWEET at opponentLane, 16 blocks ahead of opponentBlock

        // Oil logic
        if my car has OIL power-up and my car is ahead of my opponent's and my opponent is within
1 lane of my car do
            return command OIL

        // Boost logic
        if my car has BOOST power-up and my car isn't boosting do
            return command BOOST
```

```
        // If program hasn't picked a command
        return command ACCELERATE


define private method getBlocksInFront(lane: int, block: int) → List of Object as
        map = game's map
        blocks = array to store blocks in front of block in the lane
        startBlock = first block in the lane (start block) at coordinate (lane,0)
        laneList = list of all block in the lane that going to be checked
        for (i = call max(block - startBlock, 0); i <= block - startBlock + Bot's maximum speed;
i++) {
            if block in laneList is null or block is FINISH do
                stop loop
            add block to blocks
        return blocks
```

Struktur Data

Dengan paradigma pemrograman berorientasi objek, terdapat berbagai kelas dalam struktur data *source code*. Berikut struktur data dari Bot yang dibuat.

Entities

1. Car
   Kelas yang merepresentasikan entitas mobil. Terdapat atribut, position, speed, state, damage, powerups, boosting, dan boostCounter.
2. GameState
   Kelas yang merepresentasikan hal terkait pertandingan. Terdapat atribut currentRound, maxRounds, player, opponent, dan array lanes.
3. Lane
   Kelas yang mengisiasikan suatu baris, yang dimasukkan ke dalam array lanes dalam kelas GameState. Terdapat atribut position, surfaceObject, dan occupiedbyPlayerId.
4. Position
   Kelas yang merepresentasikan posisi pemain. Terdapat atribut lane dan block.

Command

Seluruh perintah yang digunakan menurunkan (*inherit*) kelas utama Command.

1. AccelerateCommand
   Kelas yang merepresentasikan command accelerate, untuk menambah kecepatan dari mobil.
2. BoostCommand

Kelas yang menginisaisikan command boost, untuk menggunakan powerup boost.

3. ChangeLaneCommand
   Kelas yang menginisaisikan command untuk berbelok ke kanan dan ke kiri.
4. DecelerateCommand
   Kelas yang merepresentasikan command accelerate, untuk mengurangi kecepatan dari mobil.
5. DoNothingCommand
   Kelas yang merepresentasikan perintah bahwa mobil tidak akan melakukan perintah lain.
6. EmpCommand
   Kelas yang merepresentasikan perintah untuk mengaktifkan powerup Emp.
7. FixCommand
   Kelas yang merepresentasikan perintah untuk memperbaiki mobil.
8. LizardCommand
   Kelas yang merepresentasikan perintah untuk menggunakan powerup Lizard.
9. OilCommand
   Kelas yang merepresentasikan perintah untuk menggunakan powerup Oil.
10. TweetCommand
    Kelas yang merepresentasikan perintah untuk menggunakan powerup Tweet.

Enums

1. Direction
   Kelas Direction merepresentasikan daftar arah. Terdapat atribut lane, block, dan label. Terdapat method getLabel untuk mengembalikan atribut label
2. PowerUps
   Kelas PowerUps merepresentasikan daftar PowerUp, yaitu BOOST, OIL, TWEET, LIZARD, dan EMP.
3. State
   Kelas State merepresentasikan kondisi mobil, yaitu ACCELERATING, READY, NOTHING, TURNING_RIGHT, TURNING_LEFT, HIT_MUD, HIT_OIL, DECELERATING, PICKED_UP_POWERUP, USED_BOOST, USED_OIL, USED_LIZARD, USED_TWEET, HIT_WALL, HIT_CYBER_TRUCK, dan FINISHED.
4. Terrain
   Kelas Terrain mengenumerasi kondisi blok lintasan, yaitu EMPTY, MUD, OIL_SPILL, OIL_POWER, FINISH, BOOST, WALL, LIZARD, TWEET, dan EMP.

Bot

Kelas Bot adalah kelas utama yang digunakan untuk merepresentasikan keseluruhan dari Bot pemain yang akan dipertandingkan. Terdapat konstruktor, dan method utama run() untuk menjalankan algoritma yang digunakan untuk memberi perintah pada Bot dalam pertandingan.

Analisis Desain Solusi

Berdasarkan pengamatan yang telah dilakukan, algoritma ini cukup kompetitif. Pada awal permainan, mobil akan berusaha untuk menghindari tabrakan dan meningkatkan kecepatan. Ketika stok *powerup* meningkat, mobil akan berusaha menyerang mobil lain secara bertubi-tubi terutama menggunakan *tweet*. Namun, karena fokus algoritma ini adalah menyerang, mobil masih kurang lihai dalam menghindari serangan *tweet* lawan sehingga tetap terkena truck di depannya.

BAB V: Kesimpulan, Saran, dan Link *Source Code*

Kesimpulan

Algoritma *greedy* adalah algoritma yang digunakan untuk menyelesaikan sebuah persoalan dengan cara mencari dan memilih solusi optimum lokal persoalan. Harapannya, himpunan solusi optimum lokal yang dipilih dapat mendekati solusi optimum global dari persoalan. Algoritma *greedy* dapat digunakan untuk mengaproksimasi solusi berbagai macam persoalan, salah satunya yaitu permainan *Overdrive*.

Algoritma *greedy offensive* yang telah kami implementasikan dapat menjalankan permainan dengan baik serta bersifat kompetitif terhadap lawan. Dari hasil pengujian, bot buatan kami berhasil mengalahkan bot lain yang menerapkan pendekatan *speed*, *boost*, dan *conservative*. Walaupun begitu, diperlukan pengujian lebih lanjut terhadap variasi strategi yang lebih luas untuk dapat menyimpulkan algoritma *greedy* yang kami implementasikan berhasil mendekati solusi optimum global.

Saran

Dalam penyusunan algoritma bot, kami tidak berhasil mengimplementasikan algoritma pendeteksi lokasi Cybertruck karena keterbatasan dokumentasi yang tersedia secara publik akan *game* ini. Pemrogram yang hendak mengembangkan bot ini lebih lanjut sebaiknya mendalami lebih lanjut peluang ini untuk mengoptimalisasi *collision avoidance logic* dari algoritma yang telah ada.

Tautan *Source Code*

https://github.com/Wiradhika6051/Tubes-1-Stima-Overdrive

DAFTAR PUSTAKA

Entelect Challenge, 2020. *Entelect Challenge 2020 - Overdrive*. [online] GitHub. Available at: <https://github.com/EntelectChallenge/2020-Overdrive> [Accessed 4 Feb. 2022].

Levitin, A., 2012. *Introduction to the Design & Analysis of Algorithms*. Essex: Pearson.