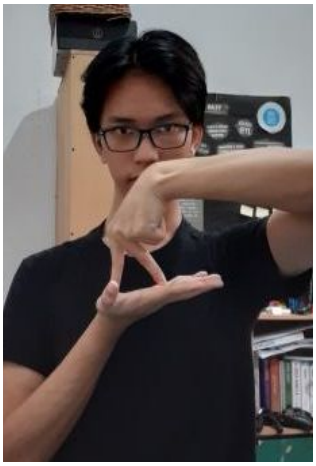
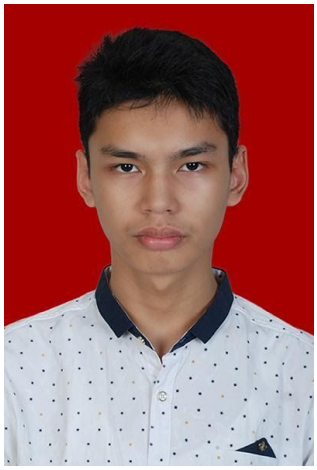
LAPORAN TUGAS BESAR

IF2211 STRATEGI ALGORITMA


Oleh

Kelompok 20 American Standard

13519014	**Mahameru Ds**
13519023	**Ilyasa Salafi Putra Jamal**
13519203	**R. B. Wishnumurti**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA INSTITUT TEKNOLOGI BANDUNG BANDUNG

2021

Bab 1
Deskripsi Tugas

1.1 Deskripsi Masalah

Worms adalah sebuah *turn-based game* yang memerlukan strategi untuk memenangkannya. Setiap pemain akan memiliki 3 *worms* dengan perannya masing-masing. Pemain dinyatakan menang jika ia berhasil bertahan hingga akhir permainan dengan cara mengeliminasi pasukan *worms* lawan menggunakan strategi tertentu. Pada tugas besar pertama Strategi Algoritma ini, gunakanlah sebuah game engine untuk mengimplementasikan permainan *Worms. Game engine* dapat diperoleh pada laman berikut https://github.com/EntelectChallenge/2019-Worms.

Tugas mahasiswa adalah mengimplementasikan seorang "pemain" *Worms*, dengan menggunakan strategi *greedy* untuk memenangkan permainan. Untuk mengimplementasikan seorang "pemain" tersebut, mahasiswa disarankan melanjutkan program yang terdapat pada *starter bot* di dalam *starter pack* pada laman berikut ini: (https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2)

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

1. Peta permainan berukuran 33x33 *cells.* Terdapat 4 tipe *cell*, yaitu *air, dirt, deep space*, dan *lava* yang masing-masing memiliki karakteristik berbeda. *Cell* dapat memuat *powerups* yang bisa diambil oleh *worms* yang berada pada *cell* tersebut.
2. Di awal permainan, setiap pemain akan memiliki 3 pasukan *worms* dengan peran dan nilai *health points* yang berbeda, yaitu::
   a. Commando
   b. Agent
   c. Technologist
3. Pada setiap *round*, masing-masing pemain dapat memberikan satu buah *command* untuk pasukan *worm* mereka yang masih aktif (belum tereliminasi). Berikut jenis-jenis *command* yang ada pada permainan:
   a. Move
   b. Dig

c. Shot
d. Do Nothing
e. Banana Bomb
f. Snowball
g. Select

4. *Command* dari kedua pemain akan dieksekusi secara bersamaan (bukan sekuensial) dan akan divalidasi terlebih dahulu. *Command* juga akan dieksekusi sesuai urutan prioritas
   tertentu.
5. Beberapa *command*, seperti “shot” dan “banana bomb” dapat memberikan *damage* pada *worms* target yang terkena serangan, sehingga mengurangi *health points*-nya. Jika *health points* suatu *worm* sudah habis, maka *worm* tersebut dinyatakan tereliminasi dari permainan.
6. Permainan akan berakhir ketika salah satu pemain berhasil mengeliminasi seluruh pasukan *worms* lawan atau permainan sudah mencapai jumlah *round* maksimum (400 *rounds*).

Adapun peraturan yang lebih lengkap dari permainan *Worms*, dapat dilihat pada laman https://github.com/EntelectChallenge/2019-Worms/blob/develop/game-engine/game-rules.md.

1.2 Spesifikasi Tugas

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

1. Unduh *latest release* starter pack.zip dari tautan berikut: https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2.
2. Untuk menjalankan permainan, kalian butuh beberapa *requirement* dasar sebagai berikut.
   a. Java (minimal Java 8): https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
   b. IntelIJ 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/Mac dapat buka dengan *double-click*, Untuk Linux dapat menjalankan *command* “make run”).
4. Secara *default*, permainan akan dilakukan diantara *reference bot* (*default*-nya berbahasa JavaScript) dan *starter bot* yang disediakan. Untuk mengubah hal tersebut, silahkan *edit* file “game-runner-config.json”. Anda juga dapat mengubah file “bot.json” untuk mengatur informasi terkait *bot* anda.
5. (Opsional) Anda dapat melihat hasil permainan dengan menggunakan *visualizer* berikut https://github.com/dlweatherhead/entelect-challenge-2019-visualiser/releases/tag/v1.0f1

Strategi *greedy* yang diimplementasikan tiap kelompok harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mengeliminasi seluruh *worms* lawan dengan senjata dan skill yang sudah disediakan dalam permainan. Salah satu contoh pendekatan *greedy* yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menyerang pasukan lawan dengan senjata dengan *hitpoint* / *damage* terbesar. Setiap “pemain” dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tugas Besar 1 (secara daring).

Strategi *greedy* harus dituliskan secara eksplisit pada laporan, karena akan diperiksa pada saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Implementasi pemain harus dapat dijalankan pada *game engine* yang telah disebutkan pada spesifikasi tugas besar, serta dapat dikompetisikan dengan pemain dari kelompok lain.

Bab 2
Landasan Teori

2.1 Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) sedemikian sehingga pada setiap langkah mengambil pilihan terbaik yang dapat diperoleh tanpa memperhatikan konsekuensi kedepan (prinsip "*take what you can get now*") dan "berharap" bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Algoritma *greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi, yaitu persoalan yang mencari solusi paling optimal dari suatu masalah.

Elemen-elemen dari algoritma *greedy* adalah:

1. Himpunan kandidat, *C* : berisi kandidat yang akan dipilih pada setiap langkah.
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*): memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
5. Fungsi kelayakan (*feasibility function*): memeriksa apakah kandidat yang dipilih layak dimasukkan ke dalam himpunan solusi.
6. Fungsi objektif: memaksimumkan atau meminimumkan (optimasi).

Dari elemen-elemen di atas, maka dapat dikatakan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, *S,* dari himpunan kandidat, *C*; yang dalam hal ini, *S* harus memenuhi beberapa kriteria yang ditentukan, yaitu *S* menyatakan suatu solusi dan *S* dioptimasi oleh fungsi objektif.

Perlu diingat bahwa solusi yang diberikan algoritma *greedy* belum tentu merupakan solusi yang optimum (terbaik), bisa saja yang dihasilkan merupakan solusi *sub-optimum* atau *pseudo-optimum*. Hal ini disebabkan algoritma *greedy* tidak memperhitungkan setiap kemungkinan solusi secara menyeluruh sebagaimana pada metode *exhaustive search*. Selain itu, penyebab lainnya adalah terdapat beberapa

fungsi seleksi berbeda yang dapat digunakan sehingga perlu dipilih fungsi yang tepat supaya algoritma dapat menghasilkan solusi yang optimal.

Skema umum algoritma *greedy*:

```
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 ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
            S ← S ∪ {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 Pemanfaatan *Game Engine*

Game engine dan sekaligus template *project* yang digunakan untuk tugas besar ini dapat diakses dan diunduh dari laman: https://github.com/EntelectChallenge/2019-Worms/releases/tag/2019.3.2.

Project yang diperoleh dari laman diatas berisi::

1. *Game engine interface*, antarmuka yang digunakan *game runner* untuk dapat digunakan *game engine.*
2. *Game engine*, bertanggung jawab dalam menerapkan peraturan permainan *Worms,* memberikan informasi *state* permainan, dan menerapkan *commands.*
3. *Game runner*, menjalankan permainan antar pemain, membaca *commands* yang diberikan *bot* dan menyampaikan *commands* tersebut kepada *game engine*.
4. *Reference bot*, *bot* yang berisi beberapa *AI logic* yang akan memainkan permainan *Worms* sebagai lawan berdasarkan peraturan yang berlaku. *Bot* ini dapat digunakan sebagai lawan untuk kebutuhan *testing*.

5. *Starter bots*, *bot* dasar dalam berbagai bahasa, termasuk Java, yang memiliki kecerdasan sangat terbatas. Digunakan sebagai *starting point* dalam membuat *bot*.

Game runner dibuat dengan menggunakan Java, sedangkan *reference bot* dibuat menggunakan Javascript, sehingga Java dan Nodejs perlu terinstall pada perangkat untuk menjalankan proyek ini.

Untuk menjalankan sebuah *game*, *double-click* pada file “run.bat” atau buka *Command Prompt* dalam direktori proyek dan jalankan *command* “run.bat”. Jika semua berjalan dengan benar, seharusnya akan muncul *console* yang memainkan permainan hingga muncul pemenang.

Bot yang digunakan untuk melawan *reference bot* dapat diganti dengan cara memodifikasi file “game-runner-config.json”. *Value* yang diubah dalam file tersebut adalah *value* “player-a”. Isikan *value* tersebut dengan direktori menuju folder yang berisi file “bot.json”. Contohnya “./starter-bots/java” untuk mengganti *bot* yang digunakan menjadi *bot* pada folder java.

Untuk Java, supaya dapat mengubah perilaku *bot*, dapat dilakukan modifikasi pada file “Bot.java” dalam folder src pada folder java. Semua entitas yang digunakan, seperti “Position”, “MyWorm”, dan “Opponent”, terdapat pada folder “entites”, sementara semua *commands* yang dapat dijalankan, seperti “DigCommand” dan “ShootCommand”, dapat ditemukan dalam folder “command”. Perlu diperhatikan bahwa *starter bot* tidak memiliki entitas banana bomb dan snowball dan tidak memiliki *command* untuk menggunakan keduanya, sehingga perlu dibuat *entity* dan *command* baru pada folder berkaitan.

Cara kerja *bot* adalah sebagai berikut:

1. *Bot* mulai berjalan saat permulaan permainan dan akan terus berjalan/beriterasi hingga permainan selesai.
2. *Bot* berkomunikasi dengan *game runner* melalui stdin dan stdout.
3. *Bot* pertama akan membaca ronde dari stdin dan membaca “state.json” untuk mendapatkan informasi mengenai kondisi dan properti *game* pada ronde tersebut.

4. *Bot* kemudian menjalankan dan menerapkan informasi yang didapatkan sebelumnya kepada kode yang sesuai dengan Bot.java untuk menentukan *command* yang dilakukan.
5. *command* yang diberikan kemudian disampaikan kepada *game runner* melalui stdout.

Untuk mengubah Bot.java dan menerapkan perubahan yang telah dibuat pada Bot.java, pertama proyek perlu dibuka menggunakan IntelliJ IDEA. Setelah melakukan modifikasi, klik pada tombol palu hijau pada bagian atas kanan IDEA. Setelah proses selesai, akan muncul folder baru bernama “target” dan muncul *pop-up* yang bertuliskan “Maven” pada bagian kanan atas IDEA. Klik pada *pop-up* tersebut, kemudian klik pada “Lifecycle” dan terakhir klik “install”. Setelah beberapa saat, proses install akan selesai dan *bot* akan diperbarui dengan modifikasi yang diterapkan.

Terdapat pula *visualizer* yang dapat memudahkan proses analisis dan *debugging*. *Visualizer* ini dapat diunduh pada laman:
https://github.com/dlweatherhead/entelect-challenge-2019-visualiser/releases/tag/v1.0f1

Visualizer ini akan memvisualisasikan salah satu *match* yang telah dijalankan pada “run.bat”. Untuk menggunakan *visualizer* ini, salin salah satu *match log* dalam folder “match-logs” pada folder proyek, kemudian tempelkan dalam folder “Matches” pada folder *visualizer*. Jika tidak terdapat folder tersebut, buatlah folder baru dengan nama yang sama dan tempelkan *match log* ke dalamnya. Selanjutnya jalankan “entelect-visualizer.exe” dan pilih *match log* yang hendak divisualisasikan.

Bab 3

Pemanfaatan Strategi *Greedy*

3.1 Pemetaan *Worms* dalam Algoritma *Greedy*

Berikut pemetaan persoalan permainan *Worms* dalam elemen-elemen algoritma *greedy* :

- Himpunan kandidat: Himpunan MyWorm, Himpunan worm lawan, PowerUp
- Himpunan solusi: Worm yang dipilih untuk diserang atau diikuti serta powerUp yang ingin di ambil
- Fungsi solusi: Memeriksa apakah Himpunan Solusi memberikan hasil yang dapat memaksimalkan potensi menang dengan mempertimbangkan gamestate.
- Fungsi seleksi: Memilih apakah mendekati worm tertentu, menyerang lawan tertentu, atau mengambil PowerUp yang akan memaksimalkan potensi kemenangan
- Fungsi kelayakan: Memeriksa apakah *command* yang terpilih dapat dieksekusi
- Fungsi objektif: Memenangkan permainan

3.2 Eksplorasi Alternatif Solusi

Berikut ini adalah beberapa strategi/solusi yang muncul dalam diskusi kami untuk menyelesaikan/memenangkan persoalan permainan *worms* ini:

1. Attack Closest Strategy

Salah satu cara memaksimalkan *damage* yang dikeluarkan dengan mengurangi *downtime* antar pertempuran. *Worm* akan memilih lawan yang terdekat terhadap dirinya diantara himpunan lawan dan bergerak secara individu menuju lawan masing-masing yang terpilih.

2. Power Up Strategy

Pada pengujian, terbukti bahwa power up healthpack memiliki pengaruh yang tidak dapat diabaikan pada *survivability worm* dalam pertempuran. Strategi ini mengutamakan mengambil power up terlebih dahulu dibandingkan menyerang lawan dengan tujuan memanfaatkan power up atau mencegah lawan memanfaatkannya sambil berkumpul dengan *worm* teman di titik power up.

3. Leader Strategy

Worm yang sendiri menjadi target yang mudah bagi lawan, sehingga diperlukan suatu koordinasi antar *worm* supaya dapat bergerak dalam kelompok. Strategi ini mengutamakan tiap *worm* bergerak mendekati leader *worm* yang terpilih, dengan mempertimbangkan pula lawan dalam jarak serangan.

4. Score Strategy

Kemenangan pada permainan ini juga dapat dicapai dengan memiliki skor tertinggi pada akhir permainan. Pada strategi ini, *worm* dengan nyawa kecil tidak akan menyerang dan akan kabur dan berusaha melakukan *commands* yang menghasilkan banyak skor.

5. Swarm Attack Strategy

Sebelumnya telah dibuktikan bahwa *worm* yang sendiri adalah target yang mudah. Strategi bertujuan menyerang satu target lawan secara berbarengan dengan *worm* teman lainnya.

3.3 Analisis Alternatif Solusi

Berikut ini adalah analisis efektivitas setiap strategi yang disebutkan diatas secara sendiri-sendiri:

Strategi	Keunggulan	Kelemahan
Attack Closest	1. *Damage output* relatif tinggi dari awal 2. Memberikan tekanan pada lawan sejak dini, mencegah melakukan strategi tertentu	1. Tiap *worm* beraksi sendiri, kondisi yang sangat rentan 2. Kekalahan satu *worm* akan berakibat fatal karena dapat berujung pada 2v1 untuk salah satu *worm* dan berujung pada 3v1 3. Banana bomb dan snowball cepat habis
Power Up	1. Mencegah lawan menggunakan power up *2.* *Worms* menjadi berkumpul pada titik tertentu	1. Rentan saat perjalanan menuju power up 2. Kurang terkoordinasi setelah power up diambil, sehingga menjadi rentan
Leader	1. *Worms* berkumpul saat bertarung dan menjelajahi peta 2. Bagus dalam	1. Rawan terhadap banana bomb yang memiliki *damage AOE*. 2. *Worm* non-leader tidak berkontribusi banyak karena

	mengeksploitasi lawan yang sendirian	kadang akan terhalang *dirt* atau leader sendiri 3. Sulit menggunakan banana bomb dan snowball karena akan terkena teman
Score	1. Memperlama durasi permainan 2. Memperoleh banyak *score* dari eksekusi *commands*	1. Bergantung pada permainan untuk berlangsung dalam waktu yang lama 2. *Worms* cenderung bertindak sendiri, sehingga menjadi rentan
Swarm Attack	1. Fokus pada satu lawan, sehingga cepat membunuh lawan satu per satu 2. Karena fokus pada satu lawan, *worms* akan berkumpul pada target 3. Jika terdapat *Worm* yang perlu mengambil power up, *worm* lainnya tetap dapat memberikan tekanan pada lawan 4. Banana bomb dan snowball masih disimpan untuk kondisi genting, tidak dihabiskan di awal	1. Rawan *friendly fire* karena menggunakan taktik yang agresif dengan teman yang berdekatan

Sementara analisis dari segi efisiensi untuk strategi-strategi diatas adalah sebagai berikut:

1. Attack Closest Strategy

Algoritma untuk strategi ini cukup menyeleksi dari *loop* yang menelusuri himpunan lawan kemudian menghitung jaraknya dan memilih yang terdekat. Dari jumlah *loop* yang diperlukan pada tiap iterasi/ronde, maka dapat disimpulkan kompleksitasnya adalah **O(n) untuk tiap iterasi/ronde**.

2. Power Up Strategy

Algoritma untuk strategi ini memerlukan *loop* yang memeriksa sumbu y pada peta dan *loop* yang memeriksa sumbu x pada peta untuk mencari Cell yang memiliki power up. Dari jumlah *loop* yang diperlukan pada tiap iterasi, maka dapat disimpulkan kompleksitasnya adalah **O(n^2) untuk tiap iterasi/ronde**.

3. Leader Strategy

Algoritma untuk strategi ini memerlukan sebuah *loop* untuk mencari dan menentukan *worm* yang akan ditunjuk menjadi leader dalam himpunan MyWorms. Dari jumlah *loop* yang diperlukan pada tiap iterasi/ronde, maka dapat disimpulkan kompleksitasnya adalah **O(n) untuk tiap iterasi/ronde**.

4. Score Strategy

Algoritma untuk strategi ini cukup memeriksa apakah nyawa *worm* sudah dibawah suatu batas pada tiap iterasi/ronde. Jika sudah *worm* tidak akan melakukan serangan apapun dan akan terus kabur untuk mendapatkan skor. Dari sini dapat disimpulkan bahwa kompleksitasnya adalah **O(1) untuk tiap iterasi/ronde**. Karena algoritma ini memiliki kompleksitas yang rendah, dalam praktiknya algoritma ini dapat digabung dengan algoritma strategi lain yang lebih agresif seperti strategi Attack Closest. Dalam kasus itu, kompleksitasnya menjadi kompleksitas algoritma yang digabungkannya.

5. Swarm Attack Strategy

Algoritma untuk strategi ini memerlukan *loop* untuk menyeleksi lawan mana yang dijadikan target dari himpunan lawan. Untuk mengatasi kekurangan strategi ini, dibuat pula *loop* yang memeriksa apakah terdapat teman dalam arah serangan dengan melihat posisi teman pada himpunan MyWorms. Dari jumlah *loop* yang diperlukan pada tiap iterasi/ronde, maka dapat disimpulkan kompleksitasnya adalah **O(n^2) untuk tiap iterasi/ronde**.

3.4 Strategi *Greedy* yang dipilih

Setelah melakukan analisis terhadap semua strategi diatas, kami memutuskan untuk menggunakan Swarm Attack Strategy sebagai strategi utama karena keunggulannya yang banyak dan kelemahannya yang dapat dimitigasi. Akan tetapi, kami juga berhipotesis bahwa dengan menggabungkan beberapa strategi di atas dengan kadar tertentu akan dapat menghasilkan strategi yang memiliki lebih banyak keunggulan tanpa kelemahan yang terlalu fatal. Untuk menentukan kombinasi ini, perlu diimplementasikan beberapa versi dan dilakukan beberapa kali pengujian. Hasil implementasi dan pengujian ini akan dibahas pada bab selanjutnya.

Bab 4

Implementasi dan Pengujian

4.1 Implementasi pada *Game Engine*

Garis besar algoritma yang telah dipilih sebagai strategi paling efektif adalah sebagai berikut :

1. Lakukan SwarmAttack kepada satu worm lawan. Worm ini merupakan worm target. Swarm Attack berarti semua worm akan bergerak mendekati dan berusaha menyerang Worm Target
    a. Apabila ada worm lawan lainnya yang terlalu dekat, maka ubah fokus ke worm tersebut
2. Jika HP dibawah 50 dan jarak ke healthpack terdekat < 2, cari healthpack, jika tidak, lanjutkan algoritma
3. Jika dapat melakukan attack (bomb, snowball, shoot), maka lakukan attack
    a. bananabomb dan snowball dihabiskan hingga tersisa 1 yang hanya dikeluarkan ketika nyawa worm player < 30
    b. Jika terdapat worm teman yang berada pada arah tembakan command Shoot, maka bergerak ke arah sembarang terlebih dahulu.
4. Jika tidak dapat melakukan attack, lanjutkan dengan bergerak mendekati worm target.
5. Ulangi dari langkah 1

Pseudocodenya adalah sebagai berikut :

```
{ Memilih worm terdekat dengan prioritas Distance > IsFrozen > LowHealth }
wormTerdekat <- cariWormMusuhTerdakat(posisiWormX, posisisWormy);
{ Worm yang menjadi Worm Target }
targetId = 1;
distance = euclideanDistance(posisiWormX, posisiWormY, wormTerdekatX, wormTerdekatY);
{ JIka ada worm yang jaraknya < 5, maka pilih worm tersebut sebagai worm target. }
target <- swarmAttack(targetId) :
if(distance > 5 ) then
      target <= closestWorm.position;
healthpack = cariHealthpack();
```

```
if( healthpack = null) then
      jarakKeHealthpack <= 100000
else then
      jarakKeHealthpack      <=euclideanDistance(posisiWormX,      posisiWormY,
healthpackX, healthpackY)
{ Jika memungkinkan untuk mencari healthpack }
if(healthPointWorm <= 50 and healthPointWorm >= 20 and healthpack != null and
jarakKeHealthpack <= 2) then
            { Ubah target menjadi healthpack}
                  target = healthpack;
else then
      { Cek apakah target dapat dilempari bananaBomb }
      if ( idWorm == 2 && canBananaBomb(posisiTargetX, posisiTargetY)) {
            ThrowBananaCommand
      }
      { Cek apakah target dapat dilempari snowball }
      if ( idWorm  == 3 && canSnowball(posisiTargetX, positiTargetY)) {
            ThrowSnowballCommand()
       }
      { Check if you can shoot the closest enemy worm }
      If ( canShootEnemy(targetPositionX, targetPositionY)) then
                           direction <= resolveDirection(currentWorm.position,
targetWorm);
                           { Apabila ada worm sendiri pada jarak tembak, maka
batalkan serangan }
                     if(!checkFriendlyFire(direction)) then
                        ShootCommand()
moveDirection <= resolveDirection(posisiWorm, posisiTarget);
{ Cek apakah block target ditempati oleh worm lain }
block = getCellToMove(moveDirection);
if(!IsCellOccupied(block)) then
      { Jika block ditempati, maka pilih langkah lain }.
      moveDirection <= getBestMove(moveDirection);
block <= getCellToMove(moveDirection);

{ Cek tipe block }
if (block.type == CellType.AIR)  then
      MoveCommand()
```

```
else if (block.type == CellType.DIRT) then
      DigCommand()
DoNothingCommand()
```

4.2 Penjelasan Struktur Data yang Digunakan

Berikut ini adalah struktur data yang digunakan dalam permainan *worms* dan digunakan pada algoritma kami:

1. GameState

   Menyimpan segala informasi mengenai permainan yang didapatkan dari *game runner* untuk disampaikan dan disimpan struktur data lainnya serta diproses dalam algoritma, seperti himpunan lawan dan ukuran peta permainan.
2. CellType

   Menyimpan dan menyatakan tipe-tipe Cell, seperti “DIRT’ dan “AIR”
3. Direction

   Menyimpan dan menyatakan arah dalam bentuk mata angin (N, E, S, W, NE, SE, etc) dari integer x dan y bernilai 1 atau 0.
4. PowerUpType

   Menyimpan dan menyatakan tipe power up, hanya terdapat satu tipe dalam *project* ini yaitu “HEALTH_PACK”.
5. Cell

   Menyatakan peta permainan dan titik dalam peta tersebut serta jenis *cell*, seperti “AIR” atau “DIRT” dan PowerUp.
6. Position

   Menyatakan posisi suatu objek dalam peta permainan.
7. Worm

   Menyatakan karakter pemain dan menyimpan informasi yang diperlukan karakter tersebut seperti *position*, *health,* dan id.
8. Weapon

   Menyatakan senjata standar pemain yang digunakan pada ShootCommand dengan informasi *damage* dan *range*.
9. MyWorm

Ekstensi dari Worm, menyimpan informasi persenjataan *worm* seperti Weapon dan BananaBomb.

10. MyPlayer

    Menyimpan informasi *worm* pemain seperti himpunan *worm* pemain, previousCommand, dan score.

11. Opponent

    Menyimpan informasi *worm* lawan sebagaimana MyPlayer.

12. PowerUp

    Menyimpan informasi Power Up yaitu type dan value.

13. BananaBomb

    Ekstensi dari Weapon, menyimpan informasi untuk banana bomb seperti count dan damageRadius.

14. Snowball

    Ekstensi dari Weapon, menyimpan informasi untuk snowball seperti count dan freezeRadius.

15. Command

    Base class untuk *commands* lainnya, menyatakan *commands* menjadi string untuk diberikan kepada *game-runner* dalam bentuk stdout.

16. DoNothingCommand

    Salah satu implementasi Command, mengembalikan “nothing” yang menyatakan perintah untuk tidak melakukan apa-apa.

17. MoveCommand

    Salah satu implementasi Command, membutuhkan atribut integer x dan y sebagai tujuan gerak dan menyatakan perintah untuk bergerak ke arah position x,y tersebut.

18. DigCommand

    Salah satu implementasi Command, membutuhkan atribut integer x dan y sebagai arah menggali dan menyatakan perintah untuk menggali pada arah position x,y tersebut.

19. ShootCommand

Salah satu implementasi Command, membutuhkan atribut Direction dan menyatakan perintah menembak senjata *default* ke arah Direction tersebut.

20. ThrowBananaBomb

    Salah satu implementasi Command, membutuhkan atribut integer x dan y sebagai arah melempar banana bomb dan menyatakan perintah melempar banana bomb.

21. ThrowSnowball

    Salah satu implementasi Command, membutuhkan atribut integer x dan y sebagai arah melempar snowball dan menyatakan perintah melempar snowball.

4.3 Analisis Solusi yang Terimplementasikan

Dari hasil pengujian menggunakan kombinasi strategi yang ada, akhirnya kami menentukan urutan terbaik yang memiliki hasil optimal berupa konsistensi kemenangan terhadap reference bot. Strategi utama yang paling optimal adalah dengan mengimplementasi ketiga strategi ini Swarm Attack, Attack Closest, Power Up (prioritas dari tinggi ke rendah).Solusi ini dipilih dengan mempertimbangkan titik titik lemah pada setiap strategi.

Pada pengujian, terbukti penggabungan ketiga strategi ini berhasil memenangkan pertandingan. Titik titik lemah pada setiap strategi berhasil ditutupi dengan memanfaatkan strategi yang lain.

Pada implementasi, bot pertama yang diserang setelah game dimulai adalah bot commando. Pilihan ini lebih efektif dibanding penyerangan terhadap worm lain karena dua worm terdekat dengan worm commando adalah worm technologist dan agent yang memiliki kemampuan freeze dengan snowball dan banana bomb yang memiliki daya serang besar, sehingga dapat dengan mudah mengalahkan worm commando yang bisa membuat kesulitan pada ronde-ronde terakhir dikarenakan health pointnya yang 1.5 kali lebih besar dari worm lainnya.

Dengan memilih untuk mencari powerUp ketika healthpoint worm player berada pada range tertentu, worm berhasil memaksimalkan antara konsep menyerang dan

bertahan. Hal ini terlihat pada pengujian worm secara otomatis menghindari serangan worm lawan dan mulai bergerak mendekati powerUp untuk menambah health point miliknya.

Satu konsep yang berpengaruh besar adalah keputusan untuk menyimpan bananaBomb dan snowball hingga akhir permainan. Pada implementasi, dilakukan tambahkan algoritma kecil yang memungkinkan worm untuk hanya menggunakan dua masing masing bananaBomb dan snowball, dan menyimpan sisanya untuk pertarungan di ronde-ronde terakhir. BananaBomb dan snowball yang disimpan berkali kali berhasil menyelamatkan worm dari situasi genting, karena umumnya worm lawan akan menghabiskan bananaBomb dan snowball yang mereka miliki di ronde-ronde awal. Keuntungan ini terutama terlihat jelas pada keputusan untuk menyimpan snowball. Snowball yang membekukan lawan pada saat ronde-ronde terakhir hampir menjadi penentu untuk kemenangan mutlak.

Bab 5
Kesimpulan dan Saran

5.1 Kesimpulan

Pada akhir tugas besar ini, telah dibuat *bot* untuk permainan *Worms* pada *game engine* yang disediakan. *Bot* tersebut dibuat dalam bahasa Java dengan memanfaatkan prinsip algoritma *greedy* supaya dapat menghasilkan solusi optimal, yaitu kemenangan dalam permainan. Pembuatan *bot* ini juga menerapkan paradigma pemrograman berorientasi objek.

5.2 Saran

Masih ada banyak ruang untuk perbaikan pada algoritma kami supaya menjadi lebih optimal dalam melawan *bot* musuh seperti dengan mengimplementasikan algoritma *greedy* pada fungsi lain yang mungkin belum terpikirkan oleh kami. *Bot* juga masih dapat dibuat lebih cerdas lagi dengan mencoba menerapkan prinsip lainnya selain algoritma *greedy*.

Daftar Pustaka

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/stima20-21.htm
- https://github.com/EntelectChallenge/2019-Worms