PEMANFAATAN ALGORITMA GREEDY DALAM APLIKASI PERMAINAN "*WORMS*"

Laporan Tugas Besar 1 IF2211 Strategi Algoritma

Semester II Tahun Akademik 2020/2021



Oleh:

Kelompok 8 - Anang Hijau

Dionisius Darryl Hermansyah	13519058
Josep Marcello	13519164
Wilson Tandya	13519209

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

BAB I
DESKRIPSI TUGAS

Worms adalah sebuah *turned-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.

Gambar 1. Aplikasi permainan *Worms* (Sumber: Facebook)

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

1. Unduh *latest release starter pack.zip Entelect Challenge*
2. Untuk menjalankan permainan, dibutuhkan beberapa *requirement* dasar sebagai berikut.
    a. Java (minimal Java 8)
    b. IntelIJ IDEA
3. Permainan dapat dijalankan menggunakan "run.bat"
4. Secara *default*, permainan akan dilakukan diantara reference bot 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. Bot dapat dimodifikasi menggunakan *source code* yang disediakan di starter-bot. Bot harus dirancang menggunakan bahasa Java dan di-build menggunakan IntelIJ. Dilarang menggunakan kode program tersebut untuk pemainnya atau kode program lain yang diunduh dari Internet.
6. Hasil pertandingan dapat dilihat dengan menggunakan visualizer .

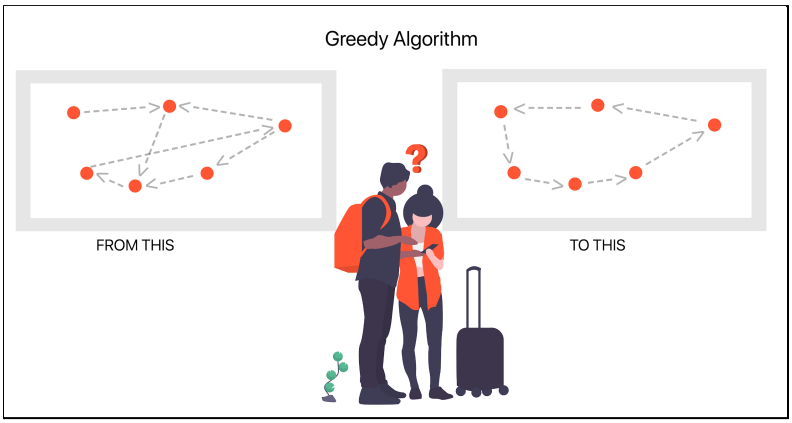
Strategi greedy yang diimplementasikan harus dikaitkan dengan fungsi objektif dari permainan itu sendiri, yaitu memenangkan permainan dengan cara mengeliminasi seluruh worms lawan dengan senjata dan skill yang sudah disediakan dalam permainan. Salah satu contoh pendekatan greedy yang bisa digunakan (pendekatan tak terbatas pada contoh ini saja) adalah menyerang pasukan lawan dengan senjata dengan hitpoint / damage terbesar. Buatlah strategi greedy terbaik, karena setiap "pemain" dari masing-masing kelompok akan diadu satu sama lain dalam suatu kompetisi Tubes 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. 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 lainnya.

BAB II
LANDASAN TEORI

2.1 Dasar algoritma *greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah sedemikian sehingga, mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu 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* terdiri atas elemen-elemen antara lain:

1. Himpunan kandidat, *C* : berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, *job*, *task*, koin, benda, karakter, dll).
2. Himpunan solusi, *S* : berisikan 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 (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi objektif : memaksimumkan atau meminimumkan.



Gambar 2.1 Ilustrasi algoritma *greedy* pada graph (Sumber: Medium)

Dengan memanfaatkan elemen-elemen diatas, 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* dioptimisasi oleh fungsi objektif. Berikut merupakan 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 himpunan 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 }
      S ← S ∪ {x} { masukkan x ke dalam himpunan solusi }
    endif
  endwhile
  {SOLUSI(S) or C = {} }
  if SOLUSI(S) then { solusi sudah lengkap }
    -> S
  else
    write('tidak ada solusi')
  endif
```

Pada skema diatas, algoritma *greedy* akan mencari sebuah himpunan solusi, *S*, yang merupakan himpunan bagian dari himpunan kandidat, *C*. Dalam hal ini, himpunan *S* merupakan himpunan kandidat, *C*, yang dikenai fungsi seleksi dan kelayakan lalu solusi akhir yang diambil adalah himpunan solusi, *S*, yang dikenai fungsi objektif. Pilihan yang dibuat menggunakan algoritma *greedy* ditentukan oleh pilihan-pilihan yang telah dibuat sampai saat ini namun tidak oleh pilihan-pilihan yang akan datang. Secara iteratif, pilihan *greedy* dilakukan sehingga membuat permasalahan yang ada menjadi permasalahan yang lebih kecil. Perbedaan mendasar antara algoritma *greedy* dengan program dinamis adalah pada algoritma *greedy* tidak pernah

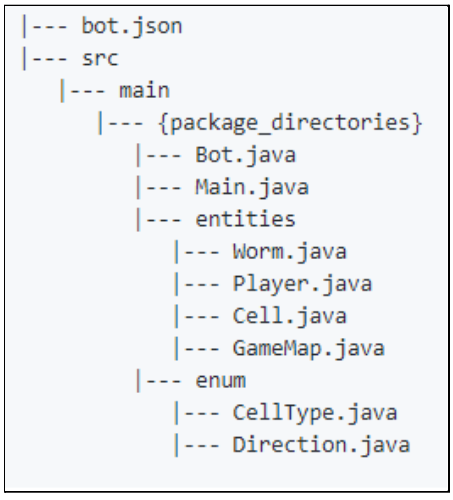
merubah pilihan yang telah dibuat sebelumnya. Perbedaan lain adalah pada algoritma *greedy* hanya dilakukan perhitungan untuk sebuah kemungkinan solusi saja sedangkan pada program dinamis akan dilakukan perhitungan untuk banyak kemungkinan solusi.

2.2 Pemanfaatan *game engine*

Game engine yang digunakan dalam permainan *Worms* ini dibuat pada acara Entelect Challenge 2019. Dalam *game engine* terdapat beberapa komponen penting yaitu

a. game-engine-interface: antarmuka yang menjadi *runner* bagi program utama,
b. game-engine: mendefinisikan aturan-aturan yang akan digunakan dalam permainan,
c. game-runner: menjalankan game dengan memanggil perintah-perintah yang tersedia,
d. reference-bot: bot yang menjadi referensi bagi pemain untuk bertanding,
e. starter-bot: bot sederhana yang dapat dimodifikasi oleh pemain.

Untuk memodifikasi bot yang telah disediakan sebelumnya, pemain dapat melakukan penyesuaian terhadap kode yang terdapat di folder starter-bot. Di folder ini, terdapat berbagai macam bot sederhana yang telah didefinisikan dalam berbagai bahasa, seperti Java, Python, Javascript, dan sebagainya. Bot berbahasa Java dapat dimodifikasi dengan beberapa *requirements* seperti Java SE Development Kit 8 dan IntelliJ IDEA. Java digunakan sebagai bahasa pemrograman bot, sedangkan IntelliJ IDEA digunakan untuk melakukan *compile* pada bot. Tampilan struktur folder bot berbahasa Java sendiri dapat dilihat pada gambar 2.2.



Gambar 2.2 Struktur folder bot berbahasa Java

Modifikasi bot dapat langsung dilakukan pada file Bot.java. Di file ini, terdapat kelas Bot yang telah didefinisikan sebelumnya, sehingga pemain langsung dapat mengubah, menambahkan, atau menghapus algoritma yang telah ada. Penambahan algoritma, termasuk algoritma *greedy*, sendiri dapat dilakukan dengan membuat fungsi baru pada kelas Bot. Fungsi tersebut nantinya akan diselaraskan dengan *method* utama yaitu run() yang akan mengeksekusi seluruh jalan kerja dari bot yang telah terdefinisi.

Di dalam folder starter-bot berbahasa Java, terdapat pula kelas-kelas lain yang telah didefinisikan sebelumnya yang berkaitan dengan objek-objek yang ada pada game seperti cell, senjata, jenis *worm*, dan masih banyak lagi. Seluruh objek-objek ini pun dapat diubah dengan leluasa dan nantinya dapat dijalankan bersama-sama dengan bot yang telah dimodifikasi.

Untuk mengkompilasi bot menggunakan IntellijIDEA dapat dilakukan langkah-langkah berikut:

a. Buat sebuah *package* dari bot yang telah dimodifikasi dengan membuka tab “Maven Projects” pada bagian kanan layar IntellijIDEA.
b. Masuk ke folder “java-sample-bot” > “Lifecycle” dan klik “Install” untuk meng-*compile* bot.
c. Setelah bot di-*compile* akan muncul file .jar di dalam folder target.
d. Bot telah berhasil di-*compile* sebagai “java-sample-bot-jar-with-dependencies.jar”

Dalam menjalankan bot dan *game*, dapat pula diatur beberapa konfigurasi game dengan mengedit game-config.json atau game-config-runner.json. Bagian-bagian yang dapat diubah meliputi jumlah ronde, tingkat kerusakan senjata, konfigurasi senjata, dan masih banyak lagi.

BAB III
PEMANFAATAN STRATEGI *GREEDY*

Dalam tugas besar kali ini, kami mendesain beberapa strategi *greedy* yang akan dimanfaatkan melalui kombinasi strategi satu sama lain untuk menghasilkan strategi terbaik. Objektif utama dari bot yang akan didesain adalah memenangkan permainan *worms* secara agresif dengan mengeliminasi musuh semaksimal mungkin. Untuk mencapai tujuan utama tersebut, tentunya diperlukan solusi-solusi untuk setiap persoalan yang ada di permainan *worms*. Contoh persoalan-persoalan tersebut adalah persoalan pergerakan *worm*, penyerangan, strategi *health pack*, dan sebagainya. Berikut merupakan penjelasan detail dari strategi *greedy* yang kami rancang berdasarkan mapping persoalan pada permainan *Worms*:

1. **Strategi pergerakan (*move*) berdasarkan *health***
    - Himpunan kandidat: Seluruh titik cell pada map.
    - Himpunan solusi: Titik cell yang dipilih berdasarkan strategi terkait.
    - Fungsi solusi: Memilih titik cell sebagai titik *move* berdasarkan prioritas *health* terendah.
    - Fungsi seleksi: Mencari musuh dengan health terendah pada *map* yang ada.
    - Fungsi kelayakan: Titik *move* yang terpilih harus *valid*, tidak menghasilkan *do nothing* serta musuh harus dalam keadaan hidup.
    - Fungsi objektif: Meminimumkan jarak dengan musuh berdasarkan prioritas *health* terendah agar dapat menyerang musuh

2. **Strategi pergerakan (*move*) berdasarkan jarak**
    - Himpunan kandidat: Seluruh titik cell pada map.
    - Himpunan solusi: Titik cell yang dipilih berdasarkan strategi terkait.
    - Fungsi solusi: Memilih titik cell sebagai titik *move* berdasarkan prioritas jarak dari musuh
    - Fungsi seleksi: Menghitung jarak minimal dari *worm* ke musuh terdekat.
    - Fungsi kelayakan: Titik *move* yang terpilih harus *valid*, tidak menghasilkan *do nothing* serta musuh harus dalam keadaan hidup.

- Fungsi objektif: Meminimumkan jarak dengan musuh berdasarkan prioritas jarak terdekat dari *worm* terkait ke musuh, dengan tujuan untuk menyerang musuh.

3. **Strategi pergerakan (*move*) gabungan**
   - Himpunan kandidat: Seluruh titik cell pada map.
   - Himpunan solusi: Titik cell yang dipilih berdasarkan strategi terkait.
   - Fungsi solusi: Memilih titik cell sebagai titik *move* berdasarkan prioritas *health* terendah kemudian jarak dari musuh.
   - Fungsi seleksi: Mencari musuh dengan health terendah, jika semua health musuh selisihnya dalam batas tertentu (20 HP), menghitung jarak minimal dari *worm* ke musuh terdekat.
   - Fungsi kelayakan: Titik *move* yang terpilih harus *valid*, tidak menghasilkan *do nothing* serta musuh harus dalam keadaan hidup.
   - Fungsi objektif: Meminimumkan jarak dengan musuh berdasarkan prioritas *health* terendah atau jarak terdekat untuk menyerang musuh.

4. **Strategi pemilihan senjata (*weapon*) berdasarkan kakas (*utilities)***
   - Himpunan kandidat: Seluruh senjata yang dimiliki oleh *worm* tertentu (*gun*, *banana bomb*, atau *snowball*).
   - Himpunan solusi: Senjata yang terpilih untuk ditembakkan.
   - Fungsi solusi: Memilih senjata berdasarkan ketersediaan kakas dengan konstrain seleksi.
   - Fungsi seleksi: Memilih kakas jika tersedia, dapat berupa *banana bomb* atau *snowball*, pada kelas *worm* tertentu, dengan kendala tambahan:
     - *Worm* memiliki kakas yang akan digunakan
     - Musuh bisa dilempari kakas (*banana bomb* atau *snowball*)
     - Musuh ada di jari-jari ledakan *banana bomb* atau *snowball*
     - Untuk *snowball*: musuh tidak beku (*frozen*)
   - Fungsi kelayakan: Kakas harus tersedia, kelas *worm* harus sesuai, dan tidak menghasilkan *do nothing*.
   - Fungsi objektif: Mengurangi darah musuh sebanyak mungkin dengan kendala *worm* milik bot tidak mati.

5. **Strategi pemilihan senjata (*weapon*) berdasarkan *damage***
    - Himpunan kandidat: Seluruh senjata yang dimiliki oleh *worm* tertentu (*gun*, *banana bomb*, atau *snowball*).
    - Himpunan solusi: Senjata yang terpilih untuk ditembakkan.
    - Fungsi solusi: Memilih senjata dengan damage terbesar.
    - Fungsi seleksi: Mencari musuh yang berada di dalam range senjata yang dimiliki oleh seekor *worm* tergantung *role*-nya, kemudian memilih senjata dengan damage terbesar jika ada.
    - Fungsi kelayakan: Senjata yang terpilih masih dapat digunakan oleh Worm dan musuh berada dalam jarak tembak.
    - Fungsi objektif: Mengurangi darah musuh sebanyak mungkin dengan kendala *worm* milik bot tidak mati.

6. **Strategi pengambilan *health pack***
    - Himpunan kandidat: Lokasi *health pack*.
    - Himpunan solusi: Lokasi *health pack* untuk dikunjungi.
    - Fungsi solusi: Memilih *health pack* jika ada di radius tertentu dari *worm*.
    - Fungsi seleksi: Mengecek *radius* sekitar *worm*, bila terdapat *health pack*, *worm* akan menuju ke *health pack*, jika tidak, *worm* akan berjalan sesuai dengan strategi pergerakan (*move*) yang terpilih.
    - Fungsi kelayakan: Lokasi *health pack* yang berada di dalam radius *worm* dan *health pack* belum terambil.
    - Fungsi objektif: Menambahkan darah *worm* dengan *health pack* dan/atau mencegah musuh mengambil *health pack*.

Strategi-strategi *greedy* yang telah dirancang dan dieksplorasi di atas dapat diklasifikasikan ke dalam beberapa kelompok besar berdasarkan alternatif kandidat-kandidat yang tersedia antara lain:

a. Persoalan pergerakan (*move*): Strategi pergerakan berdasarkan *health,* strategi pergerakan berdasarkan jarak, dan strategi pergerakan gabungan.

b. Persoalan pemilihan senjata (*weapon*): Strategi pemilihan senjata berdasarkan kakas, Strategi pemilihan senjata berdasarkan *damage*
c. Persoalan *health pack*: Strategi pengambilan *health pack*

Berdasarkan eksplorasi berbagai alternatif solusi serta rancangan algoritma *greedy* yang ada, berikut merupakan analisis kami terhadap setiap strategi:

Tabel 3.1 Analisis hasil eksplorasi alternatif solusi *greedy*

Strategi	**Analisis efektivitas dan efisiensi**
Pergerakan berdasarkan *health*	● Efektivitas: Strategi *greedy* ini mangkus untuk digunakan jika terdapat perbedaan yang signifikan antara *worms* musuh yang ada. Jika *health* musuh memiliki jumlah yang hampir sama, maka strategi ini menjadi tidak mangkus, karena bisa saja semisalnya algoritma memilih untuk menyerang *worm* dengan jarak yang lebih jauh, padahal selisih *health*-nya hanya 2 poin. ● Efisiensi: Algoritma ini mangkus karena hanya membutuhkan pengecekan atribut *health* yang terdapat pada data *game state*, serta memanfaatkan atribut lain seperti *position*, sehingga kompleksitas waktu yang diperkirakan adalah O(n).
Pergerakan berdasarkan jarak	● Efektivitas: Strategi *greedy* ini mangkus untuk digunakan jika tidak terdapat perbedaan yang signifikan antara *worms* musuh yang ada. Jika terdapat suatu perbedaan *health* yang signifikan dengan selisih jarak yang minim, maka algoritma *greedy* ini tidak akan dapat menghasilkan solusi optimal. Contohnya adalah algoritma akan memilih *worm* dengan *health* 90 dan berjarak 2 kotak dibandingkan *worm* dengan *health* 20 dan berjarak 3 kotak. ● Efisiensi: Algoritma ini cenderung lebih kompleks karena harus mengecek kombinasi jarak antar tiap *worm* yang ada, namun sisa dari implementasinya hanya tinggal menggunakan *game state* yang telah tersedia, sehingga kompleksitas waktu yang diperkirakan adalah $O(n^2)$.

Pergerakan gabungan	● Efektivitas: Strategi ini merupakan strategi yang mengombinasikan antara *greedy* berdasarkan *health* dan jarak. Strategi ini relatif lebih mangkus karena *worm* akan memprioritaskan musuh dengan *health* terendah namun berdasarkan *threshold* atau batas kondisi tertentu. Hal ini ditujukan untuk meminimalisir pemilihan solusi non optimal dari alternatif-alternatif sebelumnya. ● Efisiensi: Algoritma yang menggabungkan dua modifikasi strategi ini cenderung lebih kompleks karena selain harus mengecek kombinasi jarak tiap *worm*, algoritma juga harus mengecek perbedaan *health* yang ada. Namun, karena hanya ada 3 *worms* yang diperlukan untuk mengecek health, maka kompleksitas waktunya dapat disederhanakan menjadi $O(3) * O(n^2) = O(n^2)$.
Pemilihan senjata (*weapon*) berdasarkan kakas (*utilities*)	● Efektivitas: Strategi *greedy* ini sangat mangkus dengan tujuan untuk mengurangi *health* musuh semaksimal mungkin dalam satu kali serangan. Tujuan ini dapat dicapai dengan menggunakan setiap kakas jika masih ada di dalam *inventory*. Hal ini dengan asumsi bahwa kakas memiliki serangan dan efek terbaik sehingga harus digunakan jika masih terdapat di *inventory*. Dalam kasus tertentu, strategi ini menjadi kurang mangkus jika terdapat kondisi dimana *health* musuh relatif jauh lebih kecil dibandingkan *damage* senjata yang dipilih, dalam hal ini *banana bomb* dan *snowball*. Contohnya jika *worm* memiliki sebuah *banana bomb* dan *health* musuh yang tersisa adalah 3, maka *worm* akan tetap memutuskan untuk menggunakan *banana bomb* sehingga terkesan membuang-buang *utilities* yang ada. Dengan demikian, strategi *greedy* ini tak selalu dijamin menghasilkan solusi yang optimal. ● Efisiensi: Algoritma ini efisien karena hanya perlu mengecek atribut *utilities* berupa *banana bomb* atau *snowball* yang terdapat pada data *game state*, sehingga kompleksitas waktu yang diperkirakan adalah O(n).
Pemilihan senjata berdasarkan *damage*	● Efektivitas: Strategi *greedy* ini sangat mangkus dengan tujuan untuk mengurangi *health* musuh semaksimal mungkin dalam satu kali serangan. Tujuan ini dapat dicapai dengan menggunakan senjata dengan *damage* terbesar yang ada di dalam *inventory*. Dalam kasus

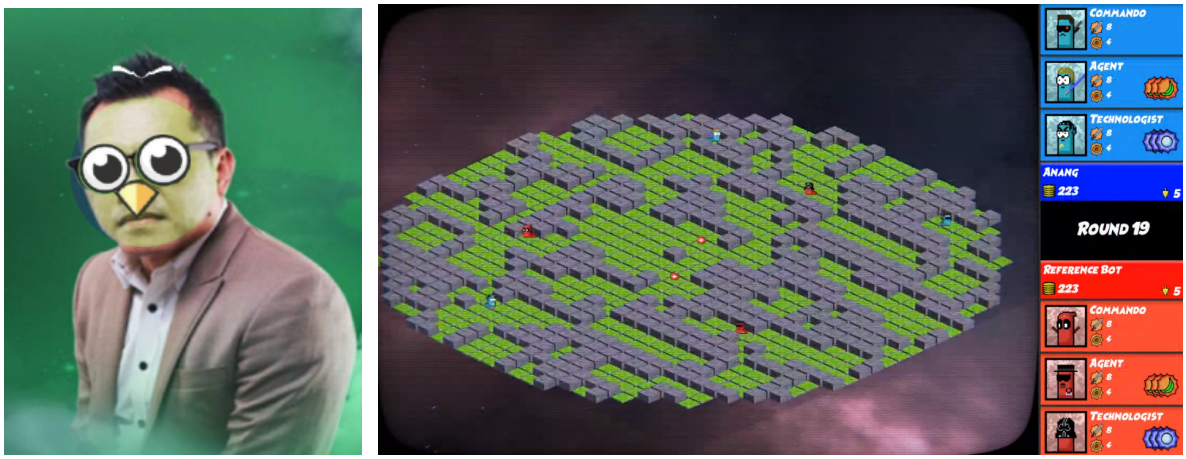
	tertentu, strategi ini menjadi kurang mangkus jika terdapat kondisi dimana *health* musuh relatif jauh lebih kecil dibandingkan *damage* senjata yang dipilih, dalam hal ini *banana bomb* dan *snowball*. Contohnya jika *worm* memiliki sebuah *banana bomb* dan *health* musuh yang tersisa adalah 3, maka *worm* akan tetap memutuskan untuk menggunakan *banana bomb* sehingga terkesan membuang-buang senjata dengan damage terbesar yang ada. Dengan demikian, strategi *greedy* ini tak selalu dijamin menghasilkan solusi yang optimal. ● Efisiensi: Algoritma ini efisien karena hanya perlu mengecek atribut *damage* dan *weapon* yang terdapat pada data *game state*, sehingga kompleksitas waktu yang diperkirakan adalah O(n).
Pengambilan *health pack*	● Efektivitas: Strategi *greedy* ini memiliki 2 tujuan yaitu untuk menambah *health worm* pemain dan/atau mencegah musuh untuk mengambil *health pack*. Strategi ini mangkus untuk memastikan bahwa *worm* pemain dapat menyelamatkan dirinya dengan cara mengisi *health* saat ada *health pack* di sekitar radius yang didefinisikan. Selain itu, strategi ini juga akan berguna jika *worm* musuh memiliki *health* yang kecil sehingga tidak dapat mengisi *health*-nya. Namun, strategi ini akan menjadi kurang mangkus jika *health worm* musuh dan *worm* pemain masih dalam keadaan penuh atau masih relatif banyak. Hal ini dapat terjadi di awal permainan saat kedua tim belum saling menyerang. Jadi, strategi *greedy* ini belum memastikan bahwa solusi yang didapatkan adalah solusi terbaik. ● Efisiensi: Algoritma ini relatif kompleks pada awalnya karena perlu mengecek lokasi *health pack* di setiap ronde permainan dengan kompleksitas waktu $O(n^2)$. Namun, pengecekan ini hanya dilakukan sekali pada ronde 1. Untuk selanjutnya, algoritma hanya perlu mengecek ketersediaan *health pack* di lokasi yang telah disimpan dan mengecek *radius* di sekitar pemain yang relatif kecil. Jadi, kompleksitas algoritma totalnya adalah O(n).

Setelah melalui analisis yang detail dan seksama, maka kami memutuskan untuk memilih strategi *greedy* berikut:

1. Strategi pergerakan gabungan (berdasarkan *health* dan jarak): Strategi ini dipilih karena merupakan strategi yang paling mangkus dalam tujuan untuk mendekati lawan yang ingin diserang. Hal ini karena parameter yang digunakan merupakan *weighted calculation* dengan dua parameter yaitu *health* dan jarak, sehingga lebih mewakilkan kondisi permainan keseluruhan.
2. Strategi pemilihan senjata (*weapon*) berdasarkan kakas (*utilities*): Strategi ini dipilih dengan asumsi *greedy* bahwa kakas memiliki efek dan *damage* yang paling kuat, sehingga algoritma ini dapat digunakan untuk mencapai tujuan utama yaitu mengurangi *health* musuh sebanyak mungkin.
3. Strategi pengambilan *health pack*: Strategi *greedy* ini dipilih dengan tujuan utama untuk menambah *health worm* pemain dan/atau mencegah musuh untuk mengambil *health pack*. Dengan pengimplementasian strategi ini, Strategi ini mangkus untuk memastikan bahwa *worm* pemain dapat menyelamatkan dirinya dengan cara mengisi *health* saat ada *health pack* di sekitar radius yang didefinisikan. Selain itu, strategi ini juga akan berguna jika *worm* musuh memiliki *health* yang kecil sehingga tidak dapat mengisi *health*-nya.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Berdasarkan perancangan dan seleksi strategi *greedy* yang telah dianalisis sebelumnya, kumpulan strategi-strategi dan algoritma-algoritma tersebut kami wujudkan dalam bentuk sebuah bot yang diberi nama *Anang Hijau*.



(a) (b)

Gambar 4.1 (a) Logo bot *Anang Hijau* dan (b) proses visualisasi pengujian permainan bot *Anang Hijau* melawan *reference bot*

Berikut ini merupakan implementasi program utama dan algoritma bot *Anang Hijau* dalam *game engine* yang digunakan dalam bentuk *pseudocode*:

```
class Bot
    {- Atribut-}
    gameState   : GameState
    Opponent    : Opponent
    currentWorm : MyWorm
    hpLoc       : array of Position

    {- Metode -}
    function run() -> Command
        {- Fungsi utama yang akan dieksekusi setiap kali mengkompilasi bot -}
        Kamus
            enemyWorm                       : Worm
            targetCell, enemyCell, hpCell   : Cell
            x, y                            : integer
            enX, enY                        : integer
            direction                       : Direction
```

```
Algoritma
    if (hpLoc.size > 0) then
            { jika sebelumnya masih ada health pack, dicek di round ini ada atau tidak}
         checkHealthPack()
     endif

    { Usaha untuk mem-banana bomb musuh }
    if (currentWorm.bananaBombs ≠ null) then
        enemyWorm <- getFirstWormInRangeBanana()
        if (enemyWorm ≠ null and isBananaBombable(enemyWorm)) {
           { Banana bomb ada dan dapat dilemparkan }
           -> RunCommand(ThrowBanana, enemyWorm.position.x, enemyWorm.position.y)
       endif
    endif

    { Usaha untuk melempar musuh dengan snowball atau menembak musuh }
    enemyWorm <- getFirstWormInRange()
    if (enemyWorm ≠ null) then
           { Cari posisi musuh }
           enX <- enemyWorm.position.x
           enY <- enemyWorm.position.y

           { Snowball musuh kalau bisa }
           if (isSnowballable(enemyWorm))
               ->  MakeCommand(ThrowSnowball, enX, enY)
            endif

       { Jika snowball tidak dapat dilempar, tembak dengan senjata }
       { Mengambil arah tembakan ke musuh }
        direction <- resolveDirection(currentWorm.position,enemyWorm.position)
        ->  MakeCommand(Shoot, direction)
    endif

    { Nyari health pack terdekat }
    hpCell <- getCloseHealthPack();
    if (hpCell ≠ null) then
           { Kalau ada health pack yang dekat, menentukan move ke sel mana }
           x <- currentWorm.position.x
           y <- currentWorm.position.y

           { Bila sudah di sebelah health pack, health pack akan diambil }
           if (euclideanDistance(x, y, hpCell.x, hpCell.y) ≤ 1) then
               targetCell <- hpCell
           else
               targetCell <- getSurroundCellNearestToTarget(hpCell)
           endif
    else
       { Menentukan sebaiknya ke musuh terdekat atau paling sekarat }
       enemyCell <- getLowetOrNearest()

       { Menentukan gerak ke sel yang mana }
       targetCell <- getSurroundCellNearestToTarget(enemyCell)
    endif

    { Jika worm yang mau berjalan ada di Lava }
    if(getCell(currentWorm.position.x,currentWorm.position.y).type = CellType.LAVA)then
       { Cek tipe sel yang dituju }
       if (targetCell.type = CellType.AIR or targetCell.type = CellType.LAVA) then
           ->  MakeCommand(Move, targetCell.x, targetCell.y)
       else if (targetCell.type = CellType.DIRT) then
```

```
            ->  MakeCommand(Dig, targetCell.x, targetCell.y)
        endif
    endif

{ Worm yang mau berjalan tidak ada di lava }
{ Cek tipe sel yang dituju }
if (targetCell.type = CellType.AIR) then
        -> MakeCommand(Move, targetCell.x, targetCell.y)
else if (targetCell.type = CellType.DIRT) then
        -> MakeCommand(Dig, targetCell.x, targetCell.y)
endif

->  MakeCommand(DoNothing)
```

Berdasarkan program bot yang telah diimplementasikan, dalam praktiknya digunakan beberapa struktur data dalam bentuk objek baik yang telah didefinisikan pada *starter-bot* maupun yang didefinisikan sendiri. Berikut merupakan penjelasan singkat dari masing-masing struktur data yang ada:

Tabel 4.1 Penjelasan struktur data dan kelas program

Struktur Data dan Kelas	**Penjelasan Singkat**
Main.java	Program utama untuk menghungkan kelas-kelas yang ada dengan *game engine*
Bot.java	Kelas dari bot yang didesain. Berisi metode-metode yang dapat digunakan oleh bot termasuk strategi-strategi *greedy* yang didefinisikan. Terdapat pula metode utama yaitu run() untuk menjalankan bot sesuai perintah yang dikeluarkan.
Package: **Command**	
BananaCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *banana bomb*.
Command.java	Deklarasi kelas utama *command* untuk me-*render* perintah yang ingin dieksekusi.
DigCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *dig* pada *cell* bertipe *dirt*.
DoNothingCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *do nothing*.

MoveCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *move*.
ShootCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *shoot*.
SnowballCommand.java	Deklarasi kelas beserta konstruktor untuk melakukan aksi *snowball*.
Package: **Entities**	
BananaBomb.java	Deklarasi kelas dari *banana bomb* dengan atribut *damage, range, count,* dan *damageRadius*.
Cell.java	Deklarasi kelas dari *cell* dengan atribut *x, y, type,* dan *powerUp.*
GameState.java	Deklarasi kelas *game state* dari dengan atribut *currentRound, maxRounds, mapSize, currentWormId, consecutiveDoNothingCount, myPlayer, opponents,* dan *map.*
MyPlayer.java	Deklarasi kelas *my player* dari dengan atribut *id, score, health,* dan *worms.*
MyWorm.java	Deklarasi kelas *my worm* dari dengan atribut *weapon, snowballs,* dan *bananaBombs.*
Opponent.java	Deklarasi kelas *opponent* dari dengan atribut *id*, *score*, dan *worms*
Position.java	Deklarasi kelas *position* dari dengan atribut *x* dan *y*.
PowerUp.java	Deklarasi kelas *power up* dari dengan atribut *type* dan *value*.
Snowball.java	Deklarasi kelas *snowball* dari dengan atribut *freezeDuration, range, count,* dan *freezeRadius*.
Weapon.java	Deklarasi kelas *weapon* dari dengan atribut *damage* dan *range*.
Worm.java	Deklarasi kelas *worm* dari dengan atribut *id, health, position, diggingRange, movementRange, roundsUntilUnfrozen,* dan *profession*.
Package: **Enums**	
CellType.java	Deklarasi semua tipe *cell* yang ada yaitu *deep_space*, *dirt*,

	dan *air*.
Direction.java	Deklarasi semua tipe *direction* yang ada yaitu *N, NE, E, SE, S, SW, W,*dan *NW* (arah mata angin).
PowerUpType.java	Deklarasi semua tipe *power up* yang ada dalam hal ini hanya ada satu tipe yaitu *health_pack*.
Profession.java	Deklarasi semua tipe *profession* yang ada yaitu *agent, commando,* dan *technologist*.

Menurut proses pembuatan dan pengujian yang telah dilakukan pada program atau *bot*, berikut merupakan analisis dari desain yang telah dipilih dan diimplementasikan:

1. Strategi pergerakan gabungan (berdasarkan *health* dan jarak): Strategi untuk memecahkan persoalan pergerakan *worm* ini dapat dikatakan memiliki kualitas yang baik. Hal ini karena, hasil strategi *greedy* ini hampir dapat dijamin bahwa menghasilkan solusi paling optimal dengan adanya sistem *threshold* atau batas kondisi yang terkait dengan selisih *health* dari *worm* musuh. Dengan demikian, pergerakan *worm* dapat selalu mendukung objektif utama dalam permainan yaitu mengeliminasi musuh semaksimal mungkin.
2. Strategi pemilihan senjata (*weapon*) berdasarkan kakas (*utilities*): Strategi untuk menyelesaikan persoalan pemilihan senjata dalam proses penyerangan musuh ini memiliki kualitas yang cukup baik karena akan menghasilkan pilihan optimal dalam kasus umum. Hal ini karena, dengan pemilihan prioritas penyerangan dengan *utilities*, maka strategi utama untuk mengurangi *health* musuh dapat tercapai dengan maksimal. Namun, dalam beberapa kasus minor, strategi pemilihan senjata ini sayangnya tidak dapat menghasilkan solusi optimal untuk beberapa kasus, dimana, *health worm* musuh yang ingin diserang jauh lebih sedikit dibandingkan *damage* yang diberikan, sehingga jika kasus tersebut terjadi, kakas akan terkesan dikeluarkan secara sia-sia.
3. Strategi pengambilan *health pack*: Strategi pengambilan *health pack* sangat berguna dalam menambahkan *health worm* yang rendah maupun mencegah musuh mengambil *health pack*, oleh karenanya strategi ini sangatlah efisien untuk diterapkan dalam tujuan untuk mencapai objektif utama yaitu mengeliminasi lawan. Dalam kasus umum, strategi ini akan menghasilkan *output* terbaik karena disertai fungsi seleksi dengan jari-jari

threshold tertentu dalam pengecekan posisi *health pack*. Akan tetapi, setelah dianalisis lebih lanjut, dalam beberapa kasus minor, strategi ini akan menjadi kurang mangkus jika *health worm* musuh dan *worm* pemain masih dalam keadaan penuh atau masih relatif banyak. Hal ini dapat terjadi di awal permainan saat kedua tim belum saling menyerang. Jadi, strategi *greedy* ini belum dapat memastikan 100% bahwa solusi yang didapatkan adalah solusi terbaik.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari tugas besar ini adalah permainan *worms* dapat dirancang strateginya secara mangkus pada *game engine* yang tersedia dengan mengimplementasikan algoritma *greedy* berupa kombinasi strategi *greedy* beragam, seperti *greedy* untuk memilih arah pergerakan bot, memilih senjata, memilih *command* terbaik, dan mengambil *health pack*.

5.2 Saran

Dalam pengerjaan proyek yang bertopik serupa kedepannya, penulis memberikan beberapa saran untuk meningkatkan performa bot yaitu:

a. Melakukan pengujian tidak hanya melawan *reference bot* namun melawan bot-bot pemenang *challenge* yang diselenggarakan untuk dapat mengevaluasi bot lebih baik dan menambah inspirasi strategi yang digunakan.
b. Merancang strategi *greedy* yang digunakan secara lebih mendetail menggunakan komponen-komponen *greedy* serta mengutamakan perancangan lebih detail dari strategi yang digunakan.
c. Membuat lebih banyak variasi strategi *greedy* dan menguji berbagai kombinasi strategi *greedy* pada bot terkait secara lebih beragam.
d. Memperdalam modifikasi bot dengan menggali dan memodifikasi *source code* dalam skala yang lebih besar.

DAFTAR PUSTAKA

N.N. 2019. *Entelect Challenge 2019 - Worms: Main repository for the Entelect Challenge 2019 tournament*. Dilansir dari www.github.com/EntelectChallenge/2019-Worms

N.N. 2021. *Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan "Worms"*. Bandung: Institut Teknologi Bandung.

Munir, R. 2021. *Algoritma Greedy (2021)*. Bandung: Institut Teknologi Bandung.