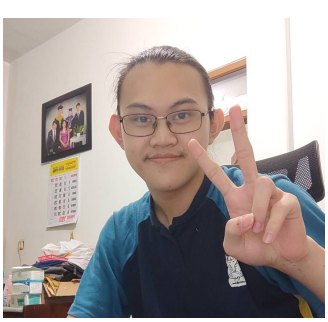
LAPORAN TUGAS BESAR 1

Mata Kuliah IF2211 Strategi Algoritma

Dosen Pengampu: Rinaldi Munir, Nur Ulfa Maulidevi

Dwi Hendratmo Widyantoro, Rila Mandala



Disusun Oleh:

Alexander	13519090
Jeane Mikha Erwansyah	13519116
Hera Shafira	13519131

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

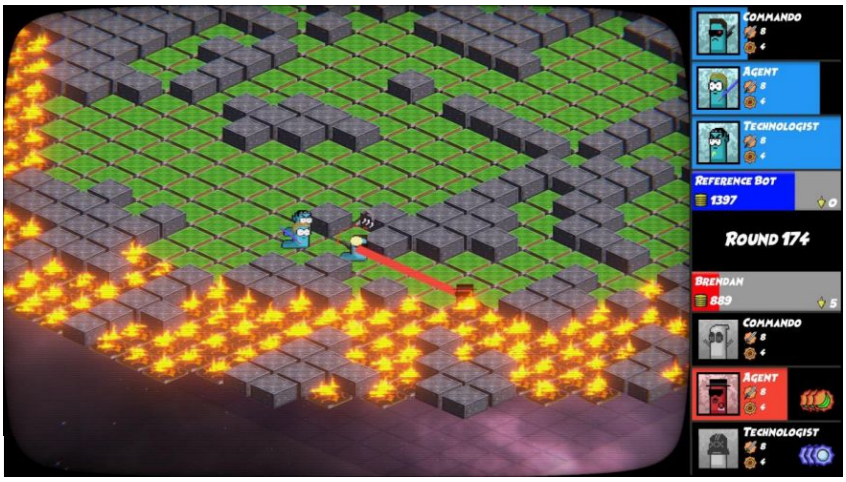
BAB I

DESKRIPSI TUGAS

Worms (atau cacing dalam Bahasa Indonesia) merupakan sebuah gim yang berbasis giliran yang memerlukan strategi untuk dimenangkan. Setiap pemain memiliki 3 cacing dengan peran yang unik. Pemain menang jika pemain sudah mengeliminasi semua cacing lawan. Pemain dapat menang jika pemain memiliki strategi permainan.

Tugas besar pertama mata kuliah IF2211 Strategi Algoritma adalah membuat implementasi strategi *greedy algorithm* untuk memenangkan permainan. Implementasi dibuat dalam bahasa pemrograman Java.

Gim Worms memiliki arena permainan seluas 33 x 33 *cells* dengan 4 tipe *cell* yaitu, *air, dirt, deep space dan lava*. Setiap pemain memiliki 3 cacing yaitu, *Commando, Agent,* dan *Technologist*. Setiap ronde, pemain dapat melakukan satu buah komando (kecuali select). Komando-komando tersebut berupa, *move, dig, shoot, do nothing, banana bomb* (hanya dapat dilakukan oleh *Agent*), *snowball* (hanya dapat dilakukan oleh *Technologist*), dan *select*. Komando dari setiap pemain dieksekusi secara bersamaan setiap rondenya.



(Gambar 1.1 Contoh tampilan permainan Worms)

BAB II

LANDASAN TEORI

2.1 *Algoritma Greedy*

Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah sedemikian rupa sehingga pada setiap langkah diambil pilihan langkah terbaik yang dapat dilakukan saat itu tanpa perlu memikirkan konsekuensi dari langkah yang diambil. Dengan mengambil langkah yang dianggap optimum untuk setiap langkah yang diambil atau optimum lokal, diharapkan algoritma dapat berakhir dengan solusi yang merupakan optimum global. Elemen-elemen yang wajib ada dari sebuah algoritma greedy adalah :

1. Himpunan kandidat (C) adalah himpunan yang berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi (S) adalah himpunan yang berisi kandidat yang sudah dipilih.
3. Fungsi solusi adalah fungsi untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi (*selection function*) adalah fungsi untuk memilih kandidat berdasarkan strategi greedy tertentu.
5. Fungsi kelayakan (*feasible*) adalah fungsi untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi objektif adalah fungsi untuk memaksimumkan atau meminimumkan aksi yang akan dilakukan.

Berdasarkan elemen-elemen diatas maka algoritma greedy juga bisa didefinisikan sebagai algoritma yang 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.

2.2 Pemanfaatan *Game Engine*

Setelah mengunduh starter bot yang disediakan oleh Entellect Challenge 2019, kelompok kami mengubah susunan folder menjadi tiga folder besar yaitu bin, doc, dan src. Folder bin berisi executable program, folder doc berisi laporan tugas besar, dan folder src berisi seluruh source code yang digunakan dalam program. Di dalam folder src terdapat dua folder yang tersedia yaitu folder starter bot dan reference bot, seluruh penambahan kode dilakukan di folder java pada folder starter bot yang berisi seluruh kelas pada program, sedangkan folder reference bot berisi bot referensi yang bisa digunakan untuk bertanding dengan bot yang sudah dibuat oleh pemain.

Penambahan kode berupa algoritma greedy yang akan digunakan oleh worm dilakukan pada file Bot.java. Pemain bisa menambah method-method baru yang bertujuan agar cacing bisa melakukan suatu aksi dengan kondisi tertentu. Aksi yang dapat dilakukan oleh sebuah worm adalah move, dig, shoot, banana bomb (khusus cacing agent), dan snowball (khusus cacing technologist). Pada starter bot, command banana bomb dan snowball belum diimplementasikan dalam kelas java sehingga pemain harus membuatnya terlebih dahulu. Command banana dibuat dengan cara merender string ‘banana x y’, sedangkan command snowball dibuat dengan cara merender string ‘snowball x y’. Pemain juga bisa memasukkan data-data yang dibutuhkan untuk jalannya permainan dari file GlobalState.json sebagai atribut dari sebuah kelas Java dengan menggunakan SerializedName agar data tersebut dapat diakses pada Bot.java. Terakhir, agar seluruh method aksi bisa dieksekusi oleh program utama, maka seluruh method ini harus dipanggil dalam method public run yang ada pada Bot.java.

Sebelum menjalankan program, pemain bisa mengatur bot mana yang akan menjadi player 1 dan player 2 melalui file game-runner-config.json, apabila pemain ingin mengganti reference bot sebagai player 2 dengan bot java lain maka pemain harus memiliki .jar dari bot java lain tersebut. Kemudian, untuk menjalankan program, source code harus di-install terlebih dahulu dengan menggunakan Maven IntelliJ IDEA agar program bisa berjalan melalui file executable yang sudah tersedia. Untuk menjalankan permainan, pemain cukup menjalankan file executable yang sudah diinstall sebelumnya, kemudian state dari setiap ronde akan tercetak pada command prompt. Jika pemain ingin melihat permainan dengan lebih jelas maka pemain bisa memindahkan match log hasil permainan ke visualizer yang sudah disediakan oleh Entellect Challenge 2019.

BAB III

PEMANFAATAN STRATEGI *GREEDY*

3.1 Mapping Persoalan Worms Menjadi Algoritma *Greedy*

Kami mendekomposisi persoalan Worms menjadi tiga bagian utama yaitu menyerang, bergerak, dan bertahan yang ketiganya kami buat algoritma greedy nya masing-masing. Berikut ini adalah hasil mapping algoritma greedy untuk ketiga bagian tersebut.

3.1.1 *Greedy* Menyerang

Algoritma *greedy* menyerang yang kami terapkan adalah kami mengincar untuk menyerang cacing yang kami anggap paling berbahaya terlebih dahulu, dengan harapan apabila cacing yang berbahaya telah mati di awal permainan maka tim kami akan semakin diuntungkan di akhir permainan. Urutan tingkat bahaya dari cacing musuh yang kami buat adalah pertama cacing agent, kedua cacing technologist, dan terakhir cacing commando, urutan ini dibuat berdasarkan cacing mana yang memiliki kemungkinan untuk menghasilkan damage paling tinggi. Agar cacing yang berbahaya semakin cepat untuk mati maka penyerangan cacing musuh dilakukan dengan jumlah cacing kami yang maksimal. Kemudian, mengingat ada kemungkinan bahwa sebuah cacing bisa diserang oleh cacing musuh ketika cacing dalam perjalanan menuju ke target, maka kami juga menambahkan strategi *self defense* pada cacing yaitu cacing akan menyerang cacing musuh yang menghampiri cacing pada saat itu.

1. Himpunan kandidat: cacing commando, cacing agent, dan cacing technologist milik lawan.
2. Himpunan solusi: cacing musuh yang dipilih untuk diserang, cacing dipilih berdasarkan prioritas serang.
3. Fungsi solusi: memeriksa apakah urutan cacing musuh yang dipilih untuk diserang sudah sesuai dengan prioritas serang yang sudah dibuat.
4. Fungsi seleksi: memilih cacing musuh untuk diserang berdasarkan prioritas serang yang dibentuk dari urutan cacing mana yang memungkinkan untuk menghasilkan damage paling tinggi.

5. Fungsi kelayakan: mengecek apakah cacing musuh yang hendak diserang sesuai prioritas masih hidup atau tidak, jika cacing sudah mati maka program akan berpindah untuk memilih cacing pada prioritas berikutnya.
6. Fungsi objektif: memilih cacing musuh paling berbahaya yaitu cacing yang memungkinkan untuk menghasilkan damage paling tinggi untuk diserang terlebih dahulu.

3.1.2 *Greedy* Bergerak

Algoritma *greedy* bergerak yang kami gunakan adalah *pathfinding* dengan menggunakan algoritma Dijkstra yang menghasilkan rute dengan *cost* yang paling minimal dari suatu posisi awal ke suatu posisi akhir. Cacing hanya bisa bergerak sejauh 1 cell dalam setiap gilirannya, oleh karena itu, dengan membuat cacing bergerak ke cell pertama dari rute yang dihasilkan oleh algoritma Dijkstra pada setiap giliran (optimum lokal), diharapkan akan juga diperoleh rute dengan cost minimum secara global setelah cacing berhasil sampai ke posisi tujuan (optimum global).

1. Himpunan kandidat: 8 cell di sekitar cacing (N,NE,E,SE,S,SW,W,NW).
2. Himpunan solusi: satu cell yang telah terpilih.
3. Fungsi solusi: memeriksa apakah rute yang terbentuk berhasil menghubungkan posisi awal dengan posisi akhir.
4. Fungsi seleksi: memilih rute dengan *cost* paling rendah dari seluruh kemungkinan rute yang dihasilkan oleh algoritma Dijkstra.
5. Fungsi kelayakan: memeriksa apakah di dalam suatu rute yang terbentuk memiliki cacing atau tidak beserta tipe dari cell tersebut, dimana dirt memiliki cost 2, dan air memiliki cost 1, dan deep space tidak akan dilewati.
6. Fungsi objektif: cacing menempuh rute dengan *cost* yang minimum.

3.1.3 *Greedy* Bertahan

Algoritma *greedy* bertahan yang kami gunakan adalah mencari *cell* yang tidak dapat diserang oleh cacing musuh yang akan berjalan namun dapat diserang oleh cacing lainnya sehingga pada ronde berikutnya cacing tersebut akan menyerang *cell* tersebut. *Cell* yang dipilih tidak boleh diduduki oleh cacing lain, jauh dari lava, bukan lava atau *deep space*. Jika semua syarat tersebut tidak terpenuhi maka cacing akan berpindah ke *cell* lain yang aman dari semua

tembakan musuh. Jika tidak ada *cell* yang memenuhi kondisi tersebut maka cacing akan menyerang cacing musuh.

1. Himpunan kandidat: 8 *cell* di sekitar cacing (N,NE,E,SE,S,SW,W,NW).
2. Himpunan solusi: satu *cell* yang telah terpilih.
3. Fungsi solusi: memeriksa apakah *cell* yang dipilih merupakan *cell* yang paling aman untuk kelangsungan hidup cacing
4. Fungsi seleksi: memilih *cell* tipe *air* atau *dirt* dengan air yang diprioritaskan. *Cell* tipe *air* pertama akan terpilih jika tidak ada maka akan dipilih *dirt cell* terakhir dari larik sel sekitar cacing pemain.
5. Fungsi kelayakan: memeriksa *cell* yang terpilih berada dalam garis tembak musuh yang akan berjalan ke posisi cacing yang sekarang akan berjalan atau tidak. *Cell* yang dipilih tidak boleh ada dalam garis tersebut namun boleh berada dalam garis tembak lainnya. *Cell* yang dipilih sekitarnya tidak ada lava. *Cell* yang dipilih harus berada dalam *range* yang ditentukan.
6. Fungsi objektif: cacing pindah ke *cell* tipe *air* yang tidak dapat terkena tembakan musuh yang berjalan bersamaan namun dapat ditembak oleh musuh berikutnya.

3.2 Eksplorasi Alternatif Solusi *Greedy*

Berikut ini adalah solusi *greedy* yang sempat kami coba, namun tidak kami gunakan dalam permainan.

3.2.1 *Greedy Health*

Alternatif solusi greedy yang kami buat adalah *greedy health* yang merupakan sebuah algoritma bertahan, yaitu ketika healthpack masih tersedia pada map, maka cacing akan bergerak menuju health pack untuk menambah health. Hal ini dilakukan dengan harapan jika health cacing tinggi maka cacing akan bertahan semakin lama dalam permainan dan memiliki kemungkinan lebih tinggi untuk menang. Selain itu, pengambilan healthpack juga dilakukan untuk mencegah musuh menambah health di tengah pertandingan, sehingga peluang kemenangan kami akan bertambah.

3.3 Analisis Efisiensi & Efektivitas dari Alternatif Solusi *Greedy* yang Dirumuskan

No.	Nama Strategi	Efisiensi	Efektivitas
1.	Greedy Menyerang	- *Best case* : O(1), terjadi ketika hanya dilakukan pengecekan kondisi dan return command shoot/banana bomb/snowball - *Worst case* : O(e log v), terjadi ketika seluruh kondisi untuk menyerang tidak terpenuhi sehingga terjadi pemanggilan algoritma Dijkstra untuk bergerak.	Strategi menyerang dengan memilih cacing musuh yang paling berbahaya terlebih dahulu hanya efektif untuk paruh awal hingga tengah permainan karena di akhir permainan ritme permainan berdasarkan strategi ini sudah terganggu oleh gangguan serangan musuh yang berakibat pada cacing kami menjadi cenderung untuk melakukan strategi self defense dibanding dengan mengejar target.
2.	Greedy Bergerak	- *Best case* : O((v + e) log v), terjadi ketika algoritma Dijkstra dilakukan dengan normal. - *Worst case* : O(e log v), terjadi ketika algoritma Dijkstra dilakukan dengan nilai e yang jauh lebih besar dari v.	Strategi ini efisien dalam proses mendekati cacing musuh untuk menyerang dan efisien juga digunakan untuk mengejar musuh yang kabur.
3.	Greedy Bertahan	- *Best case* & *worst case* : O(n), terjadi ketika dilakukan loop pengecekan kondisi dan pengembalian command yang harus dilakukan.	Strategi ini efektif karena mampu mencari posisi yang aman dari tembakan musuh dan aman juga dari cell lava sehingga cacing dapat bertahan lebih lama di permainan walau sudah memiliki *health* yang rendah.

4.	Greedy Health	- *Best case* : O(1), terjadi ketika hanya dilakukan pengecekan kondisi dan return command. - *Worst case* : O(e log v), terjadi ketika seluruh kondisi untuk mengambil healthpack tidak terpenuhi sehingga terjadi pemanggilan algoritma Dijkstra untuk bergerak.	Strategi mencari healthpack kurang efektif untuk dilakukan karena cacing menjadi kurang *offensive* dalam permainan, serta cacing memiliki resiko yang tinggi ketika bergerak menuju healthpack.

3.3 Strategi *Greedy* yang Diimplementasikan Dalam Program

Kami memilih untuk mengkombinasikan tiga strategi greedy yang akan diimplementasikan dalam program kami yaitu, greedy menyerang, greedy bergerak, dan greedy bertahan. Ketiga strategi dikombinasikan dengan cara melakukan pengecekan kondisi untuk pemanggilan masing-masing method. Pemanggilan method bertahan dilakukan ketika health cacing sudah lebih kecil dari 20 dan juga ketika ada musuh di sekitar cacing, kemudian untuk kondisi selain itu, cacing akan masuk ke method menyerang. Kemudian, strategi greedy bergerak dipanggil dalam metode serang dan kabur. Alasan dipilihnya strategi greedy menyerang dan strategi greedy bergerak adalah karena memang permainan baru dapat berlangsung dengan baik jika bot setidaknya mampu untuk bergerak mendekati musuh dan menyerang musuh, kami pun hanya memiliki satu jenis strategi menyerang dan bergerak sehingga tentu kedua strategi ini lah yang kami pilih. Lalu, untuk memperkuat bot, kami memutuskan untuk menambah strategi baru untuk bertahan. Kami memiliki dua pilihan untuk strategi bertahan yaitu, strategi greedy menghindar dan strategi greedy health. Setelah melakukan banyak pengujian, kami menganggap strategi greedy menghindar lebih baik jika dibandingkan dengan strategi greedy health karena walaupun sebuah cacing memiliki health yang tidak begitu tinggi, namun jika cacing tersebut dapat menghindar dengan baik dari musuh maka akan ada suatu kemungkinan dimana cacing musuh akan mati terkena *cell* lava atau cacing kami akan menang berdasarkan poin yang terkumpul ketika ronde permainan sudah mencapai ronde maksimal.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Program

Berikut adalah *pseudocode* beberapa potongan kode program dalam Bot.java

```
PROGRAM Bot.java
Program bot untuk memainkan permainan worms
// Urutan prioritas strategi:
// 1. Strategi serang: (1) agent, (2) technologist, dan (3) commando.
// 2. Strategi kabur

function run():
// Fungsi untuk menjalankan bot
// currentWorm adalah worm yang akan berjalan
  SET count to 0
  FOR eachWorm in opponentWorm
    IF euclideanDistance(position(currentWorm), position(eachWorm)) <= 7
          AND health(eachWorm) > 0 THEN
      SET count to count + 1
    ENDIF
  ENDFOR
  IF health(currentWorm) < 20 AND count > 0 THEN
    RETURN kaburEuy()
  ELSE
    RETURN serang()
  ENDIF
```

```
function serang():
  // Merupakan fungsi bantuan yang akan memanggil huntByIDAgent, huntByIDCommando,
  // atau huntByIDTechnologist

function huntByIDCommando(integer idTargetWorm):
  //Fungsi ini bertujuan untuk membuat cacing commando kami mengejar cacing musuh
  // dengan ID sesuai pada parameter

  SET locTarget to position(targetWorm)
  SET canAttack to FALSE
  SET isFriendlyFire to FALSE

// Strategi self defense
// resolve direction adalah fungsi pembantu untuk menentukan di arah mana cacing musuh
// berada
```

```
// friendlyFire dihasilkan oleh pemanggilan fungsi pembantu untuk mengecek apakah terjadi
// friendly fire
  SET oppWorm to getFirstWormInRange(4)
  IF oppWorm ≠ NULL AND NOT friendlyFire THEN
    SET shootdir to resolveDirection(position(currentWorm),postion(oppWorm))
    RETURN ShootCommand(shootdir)
  ENDIF

// Strategi menyerang target
// cellsFireLineDirection adalah cells dimana currentWorm bisa mengarahkan shoot yang tidak
// out of range
// moveOrDig adalah fungsi pembantu untuk memanggil algoritma greedy bergerak
  SET cells to cellsFireLineDirection
  IF cells CONTAINS locTarget AND health(opponent(worms[idTargetWorm])) > 0 THEN
    SET canAttack to TRUE
  ENDIF

  IF cells CONTAINS myPlayer(worms) THEN
    SET isFriendlyFire to TRUE
  ENDIF

  IF NOT isFriendlyFire AND canAttack THEN
    SET shootdir to resolveDirection(position(currentWorm),position(oppWorm))
    RETURN ShootCommand(shootdir)
  ELSE
    RETURN moveOrDig()
END IF

function huntByIDAgent(integer idTargetWorm):
        //Fungsi ini bertujuan untuk membuat cacing agent kami mengejar cacing musuh
dengan ID sesuai pada parameter

        SET locTarget to position(targetWorm)
        SET canAttack to FALSE
        SET canBananaBomb to FALSE
        SET isFriendlyFire to FALSE
        SET isFriendlyFireB to FALSE

        SET cells to cellsFireLineDirection
        SET cellsBB to cellsBananaBombDamageArea

        IF cells CONTAINS locTarget AND health(opponent(worms[idTargetWorm-1]))>0
THEN
                SET canAttack to TRUE
        ENDIF
```

```
        IF euclideanDistance(position(currentWorm), position(opponent(worms
idTargetWorm-1])) <= 5 AND count(banana(currentWorm)) > 0 THEN
                SET canBananaBomb to TRUE
        ENDIF

        IF cells CONTAINS myPlayer(worms) THEN
                SET isFriendlyFire to TRUE
        ENDIF

        IF cellsBB CONTAINS myPlayer(worms) THEN
                SET isFriendlyFireB to TRUE
        ENDIF

        //Strategi self defense : mengeluarkan banana bomb jika syarat terpenuhi
        FOR eachWorm in opponent(Worms)
                SET cellsBB2 to cellsBananaBombDamageAreaAroundEachWorm
                IF cellsBB2 CONTAINS myPlayer(worms) THEN
                        SET isFriendlyFireBB2 to TRUE
                ENDIF
                IF health(eachWorm) > 0 AND euclideanDistance(position(currentWorm),
position(opponent(worms idTargetWorm-1]) AND NOT isFriendlyFireBB2 AND
count(banana(currentWorm)) > 0) <= 5 THEN
                        SET target to position(eachWorm)
                        RETURN BananaBombCommand(x(target),y(target))
                ENDIF
        ENDFOR

        //Strategi self defense : menembak musuh di sekitar
        SET oppWorm to getFirstWormInRange(4)
        IF oppWorm ≠ NULL AND NOT friendlyFire THEN
                SET shootdir to resolveDirection(position(currentWorm),postion(oppWorm))
                RETURN ShootCommand(shootdir)
        ENDIF

        //Strategi menyerang target
        //moveOrDig adalah fungsi pembantu untuk memanggil algoritma greedy bergerak
        IF canBananaBomb AND NOT isFriendlyFireB THEN
                RETURN BananaBombCommand(x(locTarget),y(locTarget))
        ENDIF
        ELSE IF canAttack AND NOT isFriendlyFire THEN
                SET shootdir to resolveDirection(position(currentWorm),locTarget)
                RETURN ShootCommand(shootdir)
        ENDIF
        ELSE
                RETURN moveOrDig()
```

```
function huntByIDTechnologist(integer idTargetWorm):
        //Fungsi ini bertujuan untuk membuat cacing technologist kami mengejar cacing musuh dengan ID sesuai pada parameter

        SET locTarget to position(targetWorm)
        SET canAttack to FALSE
        SET canSnowBall to FALSE
        SET isFriendlyFire to FALSE
        SET isFriendlyFireS to FALSE

        SET cells to cellsFireLineDirection
        SET cellsSB to cellsSnowBallDamageArea

        IF cells CONTAINS locTarget AND health(opponent(worms[idTargetWorm-1]))>0 THEN
                SET canAttack to TRUE
        ENDIF

        IF euclideanDistance(position(currentWorm), position(opponent(worms idTargetWorm-1])) <= 5 AND count(snow(currentWorm)) > 0 THEN
                SET canSnowBall to TRUE
        ENDIF

        IF cells CONTAINS myPlayer(worms) THEN
                SET isFriendlyFire to TRUE
        ENDIF

        IF cellsSB CONTAINS myPlayer(worms) THEN
                SET isFriendlyFireS to TRUE
        ENDIF

        //Strategi self defense : mengeluarkan snow ball jika syarat terpenuhi
        FOR eachWorm in opponent(Worms)
                SET cellSB2 to cellsSnowBallDamageAreaAroundEachWorm
                IF cellsSB2 CONTAINS myPlayer(worms) THEN
                        SET isFriendlyFireSB2 to TRUE
                ENDIF
                IF health(eachWorm) > 0 AND euclideanDistance(position(currentWorm), position(opponent(worms idTargetWorm-1]) AND NOT isFriendlyFireSB2 AND count(snow(currentWorm)) > 0) <= 5 THEN
                        SET target to position(eachWorm)
                        RETURN SnowBallCommand(x(target),y(target))
                ENDIF
        ENDFOR

        //Strategi self defense : menembak musuh di sekitar
```

```
        SET oppWorm to getFirstWormInRange(4)
        IF oppWorm ≠ NULL AND NOT friendlyFire THEN
                SET shootdir to resolveDirection(position(currentWorm),postion(oppWorm))
                RETURN ShootCommand(shootdir)
        ENDIF

        //Strategi menyerang target
        //moveOrDig adalah fungsi pembantu untuk memanggil algoritma greedy bergerak
        IF canSnowBall AND NOT isFriendlyFireS THEN
                RETURN SnowBallCommand(x(locTarget),y(locTarget))
        ENDIF
        ELSE IF canAttack AND NOT isFriendlyFire THEN
                SET shootdir to resolveDirection(position(currentWorm),locTarget)
                RETURN ShootCommand(shootdir)
        ENDIF
        ELSE
                RETURN moveOrDig()
```

```
function pathFinding(peta, x, y, xend, yend):
  SET jalan to Alamatpath(x, y, xend, yend, 33)
  // mapArea adalah 0-33, eksklusif
  FOR i in mapArea
    FOR j in Maparea
      jalan.penyimpananjalan[i][j].ubahNilai(peta[i][j]);
    ENDFOR
  ENDFOR

  SET jarak(start(jalan)) to 0
  function adjacencyComparator(left, right):
    IF jarak(left) > jarak(right)
      RETURN 1
   ELSE
     RETURN -1
   ENDIF


SET queuejalan to PriorityQueue(33, adjacencyComparator)
  add(queuejalan(jalan.start))

  WHILE size(queuejalan) > 0 DO
    SET pinpoint to remove(queuejalan)
    IF (y(pinpoint) - 1 > = 0) THEN
      SET queuejalan TO helperPath(jalan, 0, -1, pinpoint, queuejalan) // UP
      IF (x(pinpoint) + 1 < 33) THEN
        SET queuejalan to helperPath(jalan, 1, -1, pinpoint, queuejalan)
      ENDIF // RIGHT
```

```
      IF (x(pinpoint) - 1 >= 0) THEN
        SET queuejalan to helperPath(jalan, -1, -1, pinpoint, queuejalan)
      ENDIF // LEFT
    ENDIF
  IF (y(pinpoint) + 1 < 33) THEN // DOWN
    SET queuejalan TO helperPath(jalan, 0, 1, pinpoint, queuejalan) // DOWN
      IF (x(pinpoint) + 1 < 33) THEN
        SET queuejalan to helperPath(jalan, 1, 1, pinpoint, queuejalan)
      ENDIF // RIGHT
      IF (x(pinpoint) - 1 >= 0) THEN
        SET queuejalan to helperPath(jalan, -1, 1, pinpoint, queuejalan)
      ENDIF // LEFT
    ENDIF
    IF (x(pinpoint) + 1 < 33) THEN
      SET queuejalan to helperPath(jalan, 1, 0, pinpoint, queuejalan)
    ENDIF // RIGHT
    IF (x(pinpoint) - 1 >= 0) THEN
      SET queuejalan to helperPath(jalan, -1, 0, pinpoint, queuejalan)
    ENDIF // LEFT
    SET visited(pinpoint) TO true;
  ENDWHILE
  SET path to arrayList;
  IF jalan.penyimpananjalan[x(end(jalan))][y(end(jalan))].jarak ≠ MAX_VALUE_INT THEN
    SET nodeCurrent to jalan.penyimpananjalan[x(end(jalan))][y(end(jalan))]
    WHILE current.parent ≠ null
      add(path(current.parent))
      SET current to current.parent
    ENDWHILE
  RETURN path
function helperPath(Alamatpath jalan, int x, int y, Node pinpoint, Queue<Node> queuejalan
  SET check to jalan.penyimpananjalan[pinpoint.x + x][pinpoint.y + y]
  IF value(check) ≠ 0 AND NOT visited(check) AND jarak(check) > jarak(pinpoint) +
          value(check) THEN
    SET jarak(check) to (jarak(pinpoint) + value(check))
    SET parent(check) to pinpoint;
     add(queuejalan(check))
   RETURN queuejalan
```

```
function kaburEuy():
// Memanggil fungsi helper yaitu kaburEuyHelper.
// Merupakan fungsi untuk mengatur jarak peta yang dapat dimainkan sesuai dengan ronde
// untuk menghindari lava.
  IF currentRound < 120 THEN
    RETURN kaburEuyHelper(4, 29)
  ELSE IF currentRound < 100 THEN
    RETURN kaburEuyHelper(lowerRange, upperRange)
```

```
  // ...
  ELSE IF currentRound < 290 THEN
    RETURN kaburEuyHelper(12, 20)
  ELSE
    RETURN kaburEuyHelper(13, 19)
  ENDIF

function kaburEuyHelper(lower, upper):
// lower: batas bawah x dan y peta yang harus diduduki untuk memenuhi syarat
// upper: batas atas x dan y peta yang harus diduduki untuk memenuhi syarat
  SET surroundingBlock to getSurroundingCells(position(currentWorm))
  SET ketemu to FALSE
  FOR block in surroundingBlock
    SET temp to block
    IF type(block) ≠ "lava" AND type(block) ≠ "deep space" AND
            isInEnemyLineOfFire(block) AND unoccupied(position(block)) AND
            lower <= position(block) AND position(block) <= upper AND
            isAwayFromLava(block) AND position(block) ≠ position(currentWorm) THEN
      // mencari block yang kosong tidak dapat ditembak oleh musuh yang berjalan sekarang
      // tapi dapat diserang oleh musuh selanjutnya, block tidak diduduki oleh cacing lainnya,
      // block dalam rangem block jauh dari lava, dan block berbeda dengan posisi cacing
      // sekarang
      IF type(block) = "dirt" THEN
        SET ketemu to TRUE
        SET temp to block
      ELSE
        SET ketemu to TRUE
        SET temp to block
        EXIT FOR
      ENDIF
  ENDFOR
  IF ketemu AND health(currentWorm) <= 25 THEN
    IF type(block) = "dirt" THEN
      RETURN DigCommand(positionX(block), posistionY(block))
    ELSE
      RETURN MoveCommand(positionX(block), posistionY(block))
  ENDIF
  // jika tidak ditemukan cell yang dapat ditembak oleh musuh selanjutnya dan dalam proses
  // pindah cacing aman, maka dicari cell yang aman dari semua tembakan musuh (menjauh)
  FOR block in surroundingBlock
    SET temp to block
    IF type(block) ≠ "lava" AND type(block) ≠ "deep space" AND
            isAwayFromEnemy(block) AND unoccupied(position(block)) AND
            lower <= position(block) AND position(block) <= upper AND
            isAwayFromLava(block) AND position(block) ≠ position(currentWorm) THEN
      // mencari block yang kosong tidak dapat ditembak oleh musuh yang berjalan sekarang
```

```
    // tapi dapat diserang oleh musuh selanjutnya, block tidak diduduki oleh cacing lainnya,
    // block dalam rangem block jauh dari lava, dan block berbeda dengan posisi cacing
    // sekarang
    IF type(block) = "dirt" THEN
      SET ketemu to TRUE
      SET temp to block
    ELSE
      SET ketemu to TRUE
      SET temp to block
      EXIT FOR
    ENDIF
ENDFOR
IF ketemu AND health(currentWorm) <= 25 THEN
  IF type(block) = "dirt" THEN
    RETURN DigCommand(positionX(block), posistionY(block))
  ELSE
    RETURN MoveCommand(positionX(block), posistionY(block))
ENDIF
// jika tidak ada cell sama sekali yang memenuhi syarat
RETURN serang()
```

4.2 Penjelasan Struktur Data yang Digunakan Dalam program Worms

Dalam folder java terdapat 3 folder yang berisi kelas-kelas yang digunakan pada Bot.java yaitu folder command, folder entities, dan folder enums. Berikut ini adalah detail dari seluruh kelas yang kami gunakan dalam program:

No.	Nama kelas	Jenis Kelas	Penjelasan Kelas
1.	`public interface Command`	Command	Interface ini berfungsi untuk merender string command.
2.	`public class BananaBombCommand implements Command`	Command	Pada kelas ini terdapat method public `BananaBombCommand` yang berfungsi untuk merender string command banana bomb ke program utama agar command Banana Bomb bisa digunakan pada `Bot.java`
3.	`public class`	Command	pada kelas ini terdapat method public `DigCommand` yang

	`DigCommand implements Command`		berfungsi untuk merender string command dig ke program utama agar command Dig bisa digunakan pada `Bot.java`
4.	`public class DoNothing implements Command`	Command	pada kelas ini terdapat method public `DoNothingCommand` yang berfungsi untuk merender string command DoNothing ke program utama agar command DoNothing bisa digunakan pada `Bot.java`
5.	`public class Move implements Command`	Command	pada kelas ini terdapat method public `MoveCommand` yang berfungsi untuk merender string command move ke program utama agar command move bisa digunakan pada `Bot.java`
6.	`public class ShootCommand implements Command`	Command	pada kelas ini terdapat method public `ShootCommand` yang berfungsi untuk merender string command shoot ke program utama agar command shoot bisa digunakan pada `Bot.java`
7.	`public class SnowballComma nd implements Command`	Command	pada kelas ini terdapat method public `SnowballCommand` yang berfungsi untuk merender string command snowball ke program utama agar command snowball bisa digunakan pada `Bot.java`
8.	`Public class BananaBombs`	Entities	Pada kelas ini terdapat atribut count yang merupakan jumlah banana bomb saat ini.
9.	`Public class Cell`	Entities	Pada kelas ini terdapat atribut integer koordinat x, integer koordinat y, CellType tipe sel, dan PowerUp powerup dari cell tersebut.
10.	`Public class GameState`	Entities	Kelas ini berisi atribut-atribut yang menyatakan *state game* saat ini. Atribut yang tersedia adalah integer current round, integer max round, integer map size, integer current worm id, integer consecutive do nothing (jumlah do nothing berturut-turut yang dilakukan), MyPlayer myPlayer (berisi data pemain), Opponent opponents (berisi data musuh), dan Cell[][] map.
11.	`Public class MyPlayer`	Entities	Kelas ini berisi atribut-atribut mengenai data pemain saat ini. Atribut yang tersedia adalah integer id (cacing pemain yang bermain pada ronde ini), integer score, integer health, dan MyWorm[] worms (berisi data 3 cacing milik pemain).

12.	`Public class MyWorm`	Entities	Kelas ini berisi atribut-atribut mengenai data cacing milik pemain saat ini, kelas ini merupakan *child class* dari kelas Worm. Atribut tambahan yang membedakan kelas ini dengan kelas Worm adalah adanya atribut weapon yang bertipe data Weapon.
13.	`Public class Opponent`	Entities	Kelas ini berisi atribut-atribut mengenai data musuh saat ini. Atribut yang tersedia adalah integer id (cacing musuh yang bermain pada ronde ini), integer score, integer health, dan Worm[] worms (berisi data 3 cacing milik musuh).
14.	`Public class Position`	Entities	Kelas ini menyatakan posisi dari suatu cacing dalam peta dengan dua atribut yaitu integer koordinat x dan integer koordinat y.
15.	`Public class PowerUp`	Entities	Kelas ini berisi atribut mengenai power up yang tersedia. Atribut yang tersedia adalah PowerUpType jenis power up dan integer value.
16.	`Public class SnowBalls`	Entities	Pada kelas ini terdapat atribut count yang merupakan jumlah snowball saat ini.
17.	`Public class Weapon`	Entities	Kelas ini berisi atribut weapon dari suatu cacing. Atribut yang tersedia adalah integer damage dan integer range.
18.	`Public class Worm`	Entities	Kelas ini berisi atribut yang menyatakan data dari cacing. Atribut yang tersedia adalah integer sampeMeleleh (jumlah ronde hingga cacing tidak lagi membeku akibat snowball), Profession prof (profesi dari suatu cacing), integer id (id cacing yang sedang bermain di ronde ini), integer health, integer initHP (health awal cacing), Position position (posisi cacing saat ini), integer diggingRange (range dig cacing), integer movementRange(range move cacing), BananaBombs banana (untuk mengakses data banana bomb yang dimiliki cacing, akan bernilai null jika cacing bukan berprofesi agent), SnowBalls snow (untuk mengakses data snowball yang dimiliki cacing, akan bernilai null jika cacing bukan berprofesi technologist)
19.	`Public class Alamatpath`	Entities	Kelas ini berisikan atribut untuk mencatat semua Node yang digunakan pada Algoritma Dijkstra, kelas ini berisi Node posisi awal, Node posisi akhir dan sebuah List Node untuk

			setiap vertex yang ada (dalam permainan ini adalah setiap Cell)
20.	`Public class Node`	Entities	Kelas ini berisikan sebuah Node, yang mencatat, lokasi (x,y) dari Node tersebut, jarak (untuk menyimpan total cost jalur yang diambil), value (menyimpan cost dari sebuah Node), sebuah boolean visited (untuk menunjukkan apakah suatu cell sudah dilewati oleh algoritma Dijkstra), dan sebuah Node parent, untuk menunjuk langkah yang dilakukan sebelum berjalan ke Node yang dilihat sekarang, dan akan bernilai null pada langkah pertama.
21.	`public enum CellType`	Enums	Enum ini berfungsi untuk mengakses tipe sel dari suatu sel dalam Bot.java.
22	`Public enum Direction`	Enums	Enum ini berfungsi untuk mengakses 8 mata arah dalam Bot.java.
23	`Public enum PowerUpType`	Enums	Enum ini berfungsi untuk mengakses tipe powerup dari suatu powerup yang tersedia dalam Bot.java.
24.	`Public enum Profession`	Enums	Enum ini berfungsi untuk mengakses profesi dari suatu cacing dalam Bot.java.
25.	`Public class bot`	-	Kelas ini berisi seluruh strategi yang diimplementasikan pada cacing. Terdapat 1 method yang bersifat public yaitu run, run berfungsi sebagai tempat pemanggilan method-method strategi lain yang sudah dibuat agar bisa berjalan di program utama.
26.	`Public class main`	-	Kelas ini berfungsi sebagai program utama dari permainan.

4.3 Analisis Desain Solusi Algoritma Greedy yang Diimplementasikan

No.	Nama Strategi	Analisis
1.	Greedy Menyerang	Berdasarkan hasil pengujian yang telah kami lakukan, algoritma greedy menyerang yang kami terapkan hanya efektif di fase awal hingga tengah permainan yaitu ketika tim musuh belum bermain secara terlalu *offensive.* Hal ini terjadi karena selain menerapkan prioritas serang cacing musuh, kami juga menerapkan sistem *self defense* pada cacing yang terjadi ketika cacing diserang oleh cacing musuh lain ketika dalam perjalanan mengejar cacing target. Akibatnya, cacing menjadi cenderung untuk melakukan self defense dibanding dengan mengejar cacing target. Strategi ini akan optimal untuk dilakukan jika lawan memiliki gaya permainan defensive.
2.	Greedy Bergerak	Berdasarkan hasil pengujian, strategi bergerak ini optimal jika cacing musuh yang ditargetkan tidak berpindah posisi. Sebaliknya algoritma ini kurang optimal jika cacing musuh yang ditargetkan berpindah-pindah. Strategi ini hanya optimal di awal permainan, untuk mengejar cacing musuh, namun setelah bertemu musuh mulai berperang strategi ini mulai berkurang dan perannya diambil alih oleh strategi greedy menyerang dan greedy bertahan.
3.	Greedy Bertahan atau Menghindar	Berdasarkan hasil pengujian, strategi ini optimal digunakan untuk menghindari serangan musuh dan memanen poin jika cacing pemain yang hidup adalah satu atau tidak ada lebih dari satu cacing yang berdekatan dengan musuh yang sama.. Selain kasus itu algoritma ini optimal ketika cacing dapat bergerak ke *cell* tipe *air* yang berada dalam garis tembak musuh. Strategi ini kurang optimal digunakan untuk melawan musuh yang *defensive/pacifist*.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah sedemikian rupa sehingga pada setiap langkah diambil pilihan langkah terbaik yang dapat dilakukan saat itu tanpa perlu memikirkan konsekuensi dari langkah yang diambil. Dengan mengambil langkah yang dianggap optimum untuk setiap langkah yang diambil atau optimum lokal, diharapkan algoritma dapat berakhir dengan solusi yang merupakan optimum global. Algoritma greedy dapat diterapkan sebagai strategi bermain dalam berbagai permainan seperti Othello dan Worms yang kini menjadi objek pembelajaran pada tugas besar kami.

Algoritma greedy memang tidak menjamin solusi yang dihasilkannya merupakan solusi optimum dari suatu persoalan. Hal ini terlihat dari hasil permainan Worms yang telah kami lakukan, walaupun kami sudah menggunakan algoritma greedy dalam permainan, namun tetap *win rate* yang dihasilkan tidak bisa mencapai 100%.

5.2 Saran

Saran yang dapat kami berikan setelah mengerjakan tugas besar ini adalah :

- Rencanakanlah algoritma greedy yang ingin diimplementasikan dengan matang sebelum benar-benar diimplementasikan.
- Jangan lupa untuk melihat permainan Worms sebagai sebuah *big picture*, jangan terlalu berfokus untuk menyusun strategi dengan objektif yang kecil.

DAFTAR PUSTAKA

Entellect. 2019. “Entellect Challenge 2019 - Worms” dari https://github.com/EntelectChallenge/2019-Worms diakses pada Februari 2021

Nagarajah, Thiloshon. April 3, 2017. “Dijkstra's Algorithm for Path Finding Problems” diakses https://thiloshon.wordpress.com/2017/04/03/dijkstras-algorithm-for-path-finding-problems/ pada Februari 2021

Munir, Rinaldi. 2019. Tugas Besar I IF2211 Strategi Algoritma Semester II Tahun 2020/2021 Pemanfaatan Algoritma Greedy dalam Aplikasi Permainan “Worms” dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-1-IF2211-Strategi-Algoritma-2021.pdf diakses pada Februari 2021